Г. П. Волчкова, В. М. Котов
Е.П. Соболевская

Сборник задач по «Теории алгоритмов»

Минск
2004

ОГЛАВЛЕНИЕ

Тема 1. Бинарные поисковые деревья

Основные понятия

Деревом называется связный граф без циклов. На практике часто приходится иметь дело со специальными видами деревьев. Наиболее распространенным среди них является корневое дерево.

Корневое дерево – это ориентированный граф, который удовлетворяет следующим условиям:

1) имеется в точности одна вершина, в которую не входит ни одна дуга, которая называется корнем;

2) в каждую вершину, кроме корня, входит ровно одна дуга;

3) из корня имеется путь к каждой вершине.

Если в корневом дереве существует путь из v в w, то v называют *предком* вершины w, а w - *потомком* вершины v. Если вершина не имеет потомков, то ее называют *висячей вершиной*, или *листом*. Остальные вершины называют *внутренними*. Если (v, w)- дуга v корневого дерева, то v называется *отцом* (непосредственным предком) вершины w, а w называется *сыном* (непосредственным потомком) вершины v. Вершина v корневого дерева T и все ее потомки вместе образуют *поддерево* корневого дерева T с корнем в вершине v.

Глубиной вершины v в корневом дереве называется длина в дугах пути (этот путь единственный) из корня в эту вершину.

Высотой вершины v в корневом дереве называется длина (в дугах) максимального пути из вершины v до одного из его потомков. Высота корневого дерева - высота корня. Высота дерева, состоящего из одной единственной вершины, полагается равной 0.

Уровнем вершины v называется разность высоты дерева и глубины вершины v.

Упорядоченное корневое дерево - это корневое дерево, у которого дуги, выходящие из каждой вершины, упорядочены (в дальнейшем будем считать, что они упорядочены слева направо).

Бинарное дерево - это упорядоченное корневое дерево, у каждой вершины которого имеется не более двух сыновей. В бинарном дереве каждый сын произвольной вершины определяется как *левый* или *правый*. Поддерево (если оно существует), корнем которого является левый сын вершины *v*, называется *левым поддеревом* вершины *v*. Аналогичным образом определяется *правое поддерево* для вершины *v*.

Существует несколько способов представления бинарных деревьев (предполагаем, что вершины дерева занумерованы целыми числами от 1 до *n*).

1. Представление в виде двух массивов: *Left* и *Right*:

- если вершина *j* является левым (правым) сыном вершины *i*, то $Left[i] = j\,(Right[j] = i)$;
- если у вершины *i* нет левого (правого) сына, то $Left[i] = 0\ (Right[j] = 0)$.

2. Представление в виде списковой структуры:

```
type
  tree_ptr = ^tree node;
  tree_node = record
        element : element type;
        left    : tree_ptr;
        right   : tree_ptr;
  end;
```

Предположим, что каждой вершине бинарного дерева соответствует некоторое ключевое значение (например, целое число). Бинарное дерево называется деревом поиска (*бинарным поисковым деревом*), если оно организовано так, что для каждой вершины *v* справедливо утверждение, что все ключи в

левом поддереве вершины v меньше ключа вершины v, а все ключи в правом поддереве больше. В поисковом дереве нет двух вершин с одинаковыми ключевыми значениями. Минимальный (максимальный) элемент бинарного поискового дерева соответствует ключевому значению самой левой (правой) вершины дерева.

Во многих задачах требуется найти *среднюю по значению* из вершин дерева. Средней по значению является та из вершин дерева, которой соответствует такое ключевое значение x, что количество вершин дерева имеющих ключевое значения строго меньшее x, равно количеству вершин дерева, имеющих ключевые значения строго большее x. Если количество вершин в дереве четно, то будем считать, что средней по значению вершины не существует. Если же дерево состоит из единственной вершины, то эта вершина является средней по значению.

Максимальным путем в дереве будем называть неориентированный путь наибольшей длины (в ребрах). Следует отметить, что этот путь не обязательно соединяет некоторые два листа дерева. Так, например, если у корня дерева только одно поддерево, то максимальный путь соединяет корень дерева и один из листьев. *Корнем пути максимальной длины* будем называть ту из вершин этого пути, которая находится на наибольшей высоте. Заметим, что в дереве может существовать несколько корней путей максимальной длины, а через один и тот же корень пути максимальной длины может проходить несколько различных путей максимальной длины.

Многие алгоритмы, работая с бинарными корневыми деревьями, посещают один раз каждую вершину дерева в определенном порядке. Существуют три наиболее распространенных способа обхода вершин бинарного дерева

(предполагаем, что бинарное дерево задано списковой структурой).

1. *Прямой порядок обхода* (сверху вниз) заключается в том, что корень некоторого дерева посещается раньше, чем его поддеревья. Если после корня посещается его левое (правое) поддерево, то обход называется прямым левым (правым) обходом. Приведем процедуру прямого левого обхода.

```
procedure order1 (v : tree_ptr);
        begin
           if v ≠ nil then
             begin
               solve (v);
               order1 (v^. left);
               order1 (v^. right);
             end;
      end;
```

2. *Обратный порядок обхода* (снизу вверх) заключается в том, что корень дерева посещается после его поддеревьев. Если сначала посещается левое (правое) поддерево корня, то обход называется обратным левым (правым) обходом. Приведем процедуру обратного левого обхода.

```
procedure order2 (v : tree_ptr);
        begin
           if v ≠ nil then
             begin
               order2 (v ^. left);
               order2 (v ^. right);
               solve (v);
             end;
        end;
```

3. *Внутренний порядок обхода* (слева направо) заключается в том, что корень посещается после посещения одного из его поддеревьев. Если корень посещается после его левого (правого) поддерева, то обход называется левым (правым) внутренним обходом. Приведем процедуру левого внутреннего обхода.

```
procedure order3 (v : tree_ptr);
        begin
           if v ≠ nil then
              begin
                 order3 (v ^. left);
                 solve (v);
                 order3 (v ^. right);
              end;
        end;
```

Для решения многих задач может понадобится для каждой вершины *v* вычислять некоторые метки, например, ее высоту или глубину, количество потомков в дереве, корнем которого является данная вершина, и т. д. Для выполнения этих действий можно использовать соответствующие процедуры обхода вершин дерева, а вычисления меток будет выполняться в процедуре solve (*v*).

В некоторых задачах требуется выполнить *удаление вершины* *v* из дерева (предположим, что *f* – отец этой вершины).

Задача удаления достаточно проста, если у удаляемой вершины *v* не более одного поддерева (предположим, что если поддерево существует, то *w* – его корень). В этом случае происходит непосредственное удаление вершины *v*, после чего выполняются следующие действия:

• если *v* - корень дерева, то корнем дерева станет вершина *w* (если у вершины *v* поддеревьев не было, то получаем пустое дерево);

• если *v* - лист, то ссылка у вершины *f*, указывающая на вершину *v*, станет пустой;

• если *v* не лист и не корень дерева, то ссылка у *f*, указывающая на *v*, будет указывать на *w* (рис. 1).

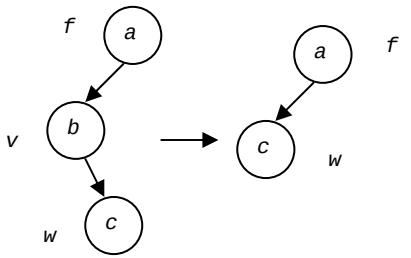


Рис. 1

Случай удаления вершины *v*, у которой два поддерева, можно свести к предыдущему случаю. Для этого непосредственное удаление вершины *v* из дерева не происходит, а ключ вершины *v* изменяется на значение минимального (максимального) ключа среди вершин правого (левого) поддерева вершины *v* (такое удаление называют правым (левым)).

Предположим, что минимальный ключ в правом поддереве вершины *v* имела вершина *z*. Теперь в дереве появились две вершины: *v* и *z* с одинаковыми ключевыми значениями, что не допустимо для бинарного поискового дерева (рис. 2).

Удаляем вершину *z* из дерева (заметим, что вершина *z* не имеет правого поддерева) и завершаем процедуру удаления.

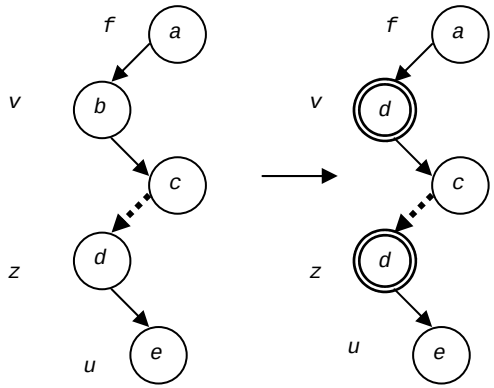


Рис. 2

Список задач

Необходимо построить бинарное поисковое дерево для последовательности вводимых чисел и выполнить для него требуемые действия. В задачах предполагается, что у пустого дерева количество вершин равно 0, а высота пустого дерева равна -1.

1. Найти и удалить (правым удалением) среднюю по значению вершину из вершин дерева, у которых количество потомков в левом поддереве не равно количеству потомков в правом поддереве. Выполнить прямой (левый) обход полученного дерева.

2. Найти и удалить (правым удалением) среднюю по значению вершину из вершин дерева, у которых высота левого поддерева не равна высоте правого поддерева. Выполнить прямой (левый) обход полученного дерева.

3. Найти и удалить (правым удалением) среднюю по значению вершину из вершин дерева, у которых количество потомков в левом поддереве отличается от количества потомков в правом поддереве на 1. Выполнить прямой (левый) обход полученного дерева.

4. Найти высоту дерева h и удалить (правым удалением) среднюю по значению вершину из вершин дерева на уровне [$h/2$], для которых количество потомков в левом поддереве больше, чем количество потомков в правом поддереве. Выполнить прямой (левый) обход полученного дерева.

5. Среди минимальных путей между листьями выбрать тот, у которого сумма ключей всех его вершин минимальна, и удалить (правым удалением) центральную вершину этого пути. Выполнить прямой (левый) обход полученного дерева.

6. Среди максимальных путей в дереве выбрать тот, у которого сумма ключей всех его вершин максимальна, затем удалить центральную вершину и корень этого пути (правым удалением). Выполнить прямой (левый) обход полученного дерева.

7. Найти высоту дерева h и удалить (правым удалением) среднюю по значению из вершин на уровне [$h/2$], у которых высота левого поддерева равна высоте правого поддерева. Выполнить прямой (левый) обход полученного дерева.

8. Среди максимальных путей в дереве выбрать тот, у которого корневая вершина находится на наименьшей глубине, и удалить (правым удалением) корневую вершину этого пути. Выполнить прямой (левый) обход полученного дерева.

9. Найти все вершины дерева, через которые проходят пути максимальной длины, и удалить (правым удалением) ту из них, ключ которой является вторым по значению. Выполнить прямой (левый) обход полученного дерева.

10. Найти все вершины дерева, через которые проходит наибольшее количество путей максимальной длины, и удалить их (правым удалением). Выполнить прямой (левый) обход полученного дерева.

11. Среди максимальных путей дерева выбрать тот, у которого сумма ключей конечных вершин минимальна, и удалить (правым удалением) корневую вершину этого пути. Выполнить прямой (левый) обход полученного дерева.

12. Среди всех путей дерева, соединяющих вершины с разным числом потомков, выбрать путь максимальной длины, у которого сумма ключей конечных вершин минимальна. Удалить (правым удалением) среднюю по значению вершину этого пути. Выполнить прямой (левый) обход полученного дерева.

13. Среди всех путей дерева, соединяющих вершины на разной высоте, выбрать путь максимальной длины, у которого сумма ключей конечных вершин минимальна. Удалить (правым удалением) среднюю по значению вершину этого пути. Выполнить прямой (левый) обход полученного дерева.

14. Среди всех путей, соединяющих корень и листья дерева, выбрать пути минимальной длины. Для каждого из этих путей найти и удалить (левым удалением) среднюю по значению

вершину пути. Выполнить прямой (левый) обход полученного дерева.

15. Заданы два дерева. Удалить (правым удалением) корень каждого дерева и определить, являются ли два полученных дерева зеркальным отражением друг друга по структуре. Выполнить прямой (левый) обход каждого из полученных деревьев.

16. Заданы два дерева. Определить, можно ли одно дерево получить из второго (по структуре) в результате удаления некоторой вершины. Если можно, то указать ту из возможных для удаления вершин, которая имеет наибольший ключ.

17. Найти и удалить (правым удалением) среднюю по значению из вершин дерева, у которых высота левого поддерева отличается от высоты правого поддерева наибольшим образом. Выполнить прямой (левый) обход полученного дерева.

18. Найти все вершины, через которые проходит четное число путей максимальной длины, и удалить (правым удалением) ту из них, ключ которой наименьший. Выполнить прямой (левый) обход полученного дерева.

19. Найти все вершины, через которые проходят пути максимальной длины, и удалить (правым удалением) самую высокую из них их. Выполнить прямой (левый) обход полученного дерева.

20. Среди всех путей дерева, соединяющих вершины на разном уровне, выбрать путь максимальной длины, для которого сумма ключей конечных вершин минимальна.

Сделать центральную вершину этого пути корнем дерева методом поворотов (дерево не обязательно останется поисковым). Выполнить прямой (левый) обход полученного дерева.

21. Найти и удалить среднюю по значению вершину из вершин дерева, у которых высоты поддеревьев равны, а количество потомков в правом и левом поддеревьях не равны. Выполнить прямой (левый) обход полученного дерева.

22. Найти и удалить среднюю по значению вершину из вершин дерева, у которых высоты поддеревьев не равны, а количество потомков в правом и левом поддеревьях равны. Выполнить прямой (левый) обход полученного дерева.

23. Пусть k – некоторое неотрицательное число. Найти все вершины, через которые проходят пути длины k, соединяющие листья дерева. Выбрать те из них, которые находятся на наименьшей глубине, и удалить (левым удалением) среднюю по значению вершину. Выполнить прямой (левый) обход полученного дерева.

24. Найти все пути длины h, соединяющие листья дерева (h – высота дерева). Среди корней этих путей выбрать те, которые расположены на максимальной глубине, и удалить (правым удалением) тот из них, который является средним по значению. Выполнить прямой (левый) обход полученного дерева.

25. Найти и удалить (правым удалением) среднюю по значению вершину из вершин дерева, у которых количество потомков в левом поддереве отличается от количества потомков в правом поддереве наибольшим образом. Выполнить прямой (левый) обход полученного дерева.

26. Найти и удалить (правым удалением) вершину с наибольшим ключевым значением среди вершин дерева, у которых количество потомков в правом и левом поддеревьях отличается наибольшим образом. Выполнить прямой (левый) обход полученного дерева.

27. Найти все листья дерева, выбрать из них средний по значению и удалить (правым удалением) его отца (если дерево состоит из единственной вершины, то удаляем эту вершину). Выполнить прямой (левый) обход полученного дерева.

28. Найти и удалить (правым удалением) среднюю по значению вершину из вершин дерева, у которых высоты поддеревьев равны (листья дерева рассматриваются в качестве вершин, у которых высоты поддеревьев равны). Выполнить прямой (левый) обход полученного дерева.

29. Найти и удалить (левым удалением) среднюю по значению вершину из вершин дерева, у которых количество потомков в левом поддереве отличается от количества потомков в правом поддереве на 2. Выполнить прямой (левый) обход полученного дерева.

30. Найти и удалить (левым удалением) среднюю по значению вершину из вершин дерева, у которых высота левого поддерева отличается от высоты правого поддерева на 2. Выполнить прямой (левый) обход полученного дерева.

31. Найти вершины, через которые проходит нечетное число путей максимальной длины, и удалить (правым удалением) ту из них, ключ которой наибольший. Выполнить прямой (левый) обход полученного дерева.

32. Найти и удалить (правым удалением) среднюю по значению из вершин дерева, через которые проходит четное число путей максимальной длины. Выполнить прямой (левый) обход полученного дерева.

Тема 2. Разработка эффективных алгоритмов

Основные понятия

Метод "*разделяй и властвуй*" заключается в следующем.

- Исходная задача разбивается на независимые подзадачи (подзадача - это та же задача, но меньшей размерности; у независимых подзадач нет общих подподзадач).
- Каждая подзадача решается отдельно (опять разбивается на подзадачи и т. д.).
- Из отдельных решений подзадач строится решение исходной задачи, т. е. метод "разделяй и властвуй" собирает решение исходной задачи снизу вверх.

Заметим, что если бы подзадачи пересекались, т. е. имели общие подподзадачи, то метод "разделяй и властвуй" делал бы лишнюю работу, решая некоторые подподзадачи по нескольку раз.

При разбиении задачи на подзадачи полезен принцип балансировки, который предполагает, что задача разбивается на подзадачи приблизительно равных размерностей, т. е. идет поддержание равновесия. Обычно такая стратегия приводит к разделению исходной задачи на две близкие по размеру подзадачи с последующим решением каждой из них тем же способом до тех пор, пока подзадачи не станут такими, что их можно будет обрабатывать непосредственно. Часто такой

процесс приводит к логарифмической трудоемкости алгоритма. Таким образом, в основе техники рассматриваемого метода лежит процедура разделения на подзадачи. Если разделение удается произвести без слишком больших затрат, то может быть построен эффективный алгоритм.

Метод "разделяй и властвуй" является эффективным, когда задачу можно разбить на независимые подзадачи за разумное время, а суммарный размер подзадач невелик. Однако когда это не удается, (применение техники приводит к экспоненциальным алгоритмам), то применяется принцип "*динамического программирования*".

Суть этого метода заключается в следующем.

- Задача погружается в семейство задач той же природы: разбивается на зависимые подзадачи (у подзадач могут быть общие подподзадачи).
- Каждая подзадача решается один раз. Оптимальные значения решений всех подзадач запоминаются, что позволяет не решать повторно встречавшиеся ранее подзадачи.
- Для исходной задачи строится возвратное соотношение, связывающее между собой оптимальные значения зависимых подзадач.

Стратегия метода "динамического программирования" заключается в эффективном сведении рассматриваемой задачи к подзадачам. Техника "динамического программирования" находит свое применение, когда все подзадачи помещаются в память. Вычисление решений идет от малых подзадач к большим, а ответы запоминаются в таблице. Одна из клеток таблицы и дает оптимальное решение исходной задачи (метод иногда называют "*табличным*").

Примерами применения этой техники могут служить алгоритмы перемножения группы матриц, перестановки сегментов массива разной размерности, определения

минимального количества операций вставки, замены и удаления символа при преобразовании одной строки в другую и др.

Список задач

1. Максимальная стоимость товара. Покупатель имеет *n* купюр достоинством $a_1, a_2, \ldots, a_n$ и продавец имеет *m* купюр достоинством $b_1, b_2, \ldots, b_n$. *Необходимо* найти максимальную стоимость товара *p*, которую покупатель не может купить, потому что нет возможности точно рассчитаться за этот товар с продавцом, хотя денег на покупку у него достаточно.

2. Точная сдача. У покупателя есть *n* монет достоинством $a_1, a_2, \ldots, a_n$ и у продавца есть *m* монет достоинством $b_1, b_2, \ldots, b_n$. *Необходимо* определить может ли покупатель приобрести вещь стоимости *s* так, чтобы у продавца нашлась точная сдача (если она необходима).

3. Проход по матрице. Задана матрица натуральных чисел *A* из *n* строк и *m* столбцов. За каждый проход через клетку матрицы с индексами $[i, j]$ взимается штраф $a[i, j]$. *Необходимо* с минимальным штрафом пройти из какой-либо клетки первой строки в *n*-ю строку. Движение по матрице осуществляется следующим образом: из текущей клетки с координатами $[i, j]$ можно перейти в любую из 3-х соседних клеток: $[i+1, j]$, $[i+1, j-1]$, $[i+1, j+1]$ (если они существуют).

4. Триангуляция треугольника. Выпуклый *n*-угольник $(n > 2)$ задается координатами своих вершин в порядке обхода по контуру. *Необходимо* разбить *n*-угольник на треугольники

$(n-3)$-мя диагоналями, не пересекающимися кроме как в вершинах многоугольника, таким образом, чтобы сумма их длин была минимальной.

5. Триангуляция треугольника с максимальной диагональю. Выпуклый n-угольник $(n>2)$ задается координатами своих вершин в порядке обхода по контуру. *Необходимо* разбить n-угольник на треугольники $(n-3)$-мя диагоналями, не пересекающимися кроме как в вершинах многоугольника, таким образом, чтобы максимальная из диагоналей имела наименьшую длину.

6. Строго возрастающая без разрывов подпоследовательность. *Необходимо* из заданной числовой последовательности A, состоящей из n элементов, вычеркнуть минимальное количество элементов так, чтобы оставшиеся элементы образовали строго возрастающую подпоследовательность элементов. Построенный алгоритм должен иметь трудоемкость $O(n\log n)$.

7. Строго возрастающая с одним разрывом подпоследовательность. *Необходимо* из заданной числовой последовательности A, состоящей из n элементов, вычеркнуть минимальное количество элементов так, чтобы в оставшейся подпоследовательности каждый последующий элемент был строго больше предыдущего кроме не более одной пары соседних элементов (одного "разрыва"). Алгоритм должен иметь трудоемкость $O(n\log n)$.

8. Строго возрастающая с m разрывами подпоследовательность. *Необходимо* из заданной числовой последовательности A, состоящей из n элементов, вычеркнуть

минимальное число элементов так, чтобы в оставшейся подпоследовательности каждый последующий элемент был строго больше предыдущего кроме, быть может, не более *m* пар соседних элементов.

9. Выбрать подпоследовательность. Из элементов заданной последовательности целых чисел *A* длины *n необходимо* выбрать элементы таким образом, чтобы они образовали подпоследовательность наибольшей длины, в которой каждый последующий элемент делился бы нацело на предыдущий. Порядок следования элементов можно менять.

10. Параллелепипеды. Заданы некоторое число n $(n>1)$ - размерность пространства и m параллелепипедов в этом пространстве, которые заданы своими размерами $(a^i[1], a^i[2], \ldots, a^i[n])$, где $(i=1,\ldots,m)$. Параллелепипед может располагаться в пространстве любым из способов, при котором его ребра параллельны осям координат. Будем считать, что параллелепипед с размерами $(a^i[1], a^i[2], \ldots, a^i[n])$ помещается в параллелепипед с размерами $(a^j[1], a^j[2], \ldots, a^j[n])$, если $(a^i[k] \le a^j[k])$ для всех $k=1,\ldots,n$. *Необходимо* построить максимальную по количеству последовательность вкладываемых друг в друга параллелепипедов.

11. Разложение числа. Заданы три числа *a*, *b*, *c*. *Можно ли* представить число *a* таким образом, чтобы

$$a = x_1 \cdot x_2 \cdot \ldots \cdot x_k ,$$

где $b \le x_i \le c$ и x_i, a, b, c – целые числа $(1 \le i \le k)$. Лучшим считается алгоритм, находящий такое представление с

наименьшим числом множителей. Предусмотреть вариант, когда такого представления не существует.

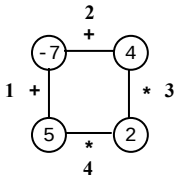
12. Бинарные последовательности. Пусть x и y - две бинарных последовательности (элементами последовательностей являются цифры 0 и 1) размерности n и m соответственно. Сами последовательности x и y можно рассматривать как запись в двоичной форме некоторых двух натуральных чисел. *Найти* максимальное число z, двоичную запись которого можно получить вычеркиванием цифр как из x, так и из y. Ответ выдать в виде бинарной последовательности.

13. Пробирки. Заданы три пробирки по 100 литров каждая. Две пробирки имеют деления, причем деления у первой и второй пробирки совпадают. Третья пробирка делений не имеет. Деления – это n натуральных чисел $(n < 100)$. Имеется 100 литров воды, которая разлита в три пробирки. Известно, какое количество воды находится в каждой из двух пробирок с делениями в начальный момент. *Необходимо* определить, можно ли отмерить в третью пробирку x литров воды. Если "да", то за какое минимальное количество переливаний. Разрешается выливание из пробирки до метки или доливание в пробирку до метки. Разрешается также выливать всю воду из пробирки.

14. Игра. Задается натуральное число n $(n < 1000)$. Двое играющих называют по очереди числа, меньшие 1000, по следующим правилам. Начиная с числа n, каждое новое число должно увеличивать одну из цифр предыдущего числа (возможно незначащий нуль) на 1, 2 или 3. Проигравшим считается тот, кто называет число 999. Для заданного n

необходимо определить, может ли выиграть игрок, делающий первый ход, при наилучших последующих ходах противника. В случае возможности выигрыша первым игроком, требуется напечатать все его возможные выигрышные первые ходы.

15*. *Полигон. Существует игра для одного игрока, которая начинается с задания полигона с n $(3 \le n \le 50)$ вершинами. Пример графического представления полигона показан на рисунке, где $n = 4$.



Для каждой вершины полигона задаётся значение - целое число, а для каждого ребра - метка операции: '+' (сложение) и '*' (умножение). Ребра полигона пронумерованы числами от 1 то n.

Первым ходом в игре удаляется одно из ребер. Каждый последующий ход состоит из следующих шагов:

- выбирается ребро (v_1, v_2);
- две вершины v_1 и v_2 заменяются новой вершиной со значением, которое равно результату выполнения операции, определенной меткой ребра (v_1, v_2), над значениями в вершинах v_1 и v_2.

Игра заканчивается, когда больше нет ни одного ребра. Результат игры – число, равное значению в оставшейся вершине.

Необходимо по заданному полигону, вычислить максимально возможное значение в оставшейся вершине и

определить список всех тех ребер, удаление которых на первом ходу игры позволяет получить это значение.

16. Эпидемия. В связи с эпидемией гриппа в больницу направляется *a* больных гриппом "*A*" и *b* больных гриппом "*B*". Больных гриппом "*A*" нельзя помещать в одну палату с больными гриппом "*B*". Имеется информация об общем количестве палат *p* в больнице, пронумерованных числами от 1 до *p*, и о распределении уже имеющихся там больных. *Необходимо* определить максимальное количество больных *m*, которое больница в состоянии принять. Если все поступившие больные размещены, то указать номера палат, куда помещаются больные гриппом "*A*" (палаты указываются в порядке возрастания номеров). При размещении новых больных не разрешается переселять уже имеющихся больных из палаты в палату.

17. Торговые скидки. В магазине каждый товар имеет цену. Например, цена одного цветка равна 2$, а цена одной вазы равна 5$. Чтобы привлечь покупателей, магазин ввел скидки. Скидка заключается в том, чтобы продавать набор одинаковых или разных товаров по пониженной цене. Примеры: три цветка продавать за 5$ вместо 6$, или две вазы вместе с одним цветком продавать за 10$ вместо 12$. *Необходимо* вычислить наименьшую цену, которую покупатель должен заплатить за нужные ему покупки и определить, какими скидками это можно сделать. Набор товаров, который требуется купить, нельзя дополнять ничем, даже если бы это снизило общую стоимость набора. Для описанных выше цен и скидок наименьшая цена за три цветка и две вазы равна 14$ (2 вазы и 1 цветок продаются по сниженной цене 10$ и 2 цветка - по обычной цене 4$).

18. Больше не запишешь. В файловой системе персонального компьютера файлы организованы в каталоги. В компьютере полное имя файла является строкой, состоящей из имен каталогов и имени файла, разделенных символом "\", причем "\" не может быть первым, последним символом, а также идти два раза подряд. Имя файла (каталога) может быть произвольной длины, но длина полного имени файла не может быть длиннее n символов. В качестве символов, допустимых к употреблению в именах файлов (каталогов), могут использоваться символы из алфавита, состоящего из k букв (символ "\" не входит в их число). *Необходимо* для данных k $(1 \le k \le 13)$ и n $(1 \le n \le 50)$ определить максимальное число файлов l, которое можно записать на данный компьютер.

19. Концерт. Во время трансляции концерта предприниматель решил сделать бизнес на производстве кассет. Он имеет m кассет с длительностью звучания d каждая и хочет записать на них максимальное число песен. Песни (их общее количество n) транслируются в порядке $1, 2, \ldots, n$ и имеют заранее известные ему длительности звучания $l(1), l(2), \ldots, l(n)$. Предприниматель, прослушивая по порядку песни, может выполнять одно из следующих действий:

- если песня на текущую кассету помещается, то он может записать ее на кассету или пропустить;
- если песня на кассету не помещается, то он может пропустить песню или записать ее на новую кассету (при этом старая кассета откладывается и туда уже ничего не может быть записано). *Необходимо* определить максимальное количество песен, которые предприниматель может записать на кассеты

20. Палиндром. Вводится непустая строка S, которая имеет длину не более 100 символов и состоит только из прописных

латинских букв. Палиндромом называется такая строка символов, которая читается слева направо и справа налево одинаково. *Необходимо* для заданного палиндрома

1) удалить из строки минимальное количество символов так, чтобы получился палиндром;

2) вставить в строку минимальное количество символов так, чтобы получился палиндром.

21. Юбилей. В связи с праздником *n* человек $(n \le 10)$ решили устроить вечеринку. Для проведения вечеринки достаточно купить *mf* бутылок фанты, *mb* бананов и *mc* тортов. При покупке действуют правила оптовой торговли: стоимость набора товаров может отличаться от суммарной стоимости его отдельных частей. Для каждого из *m* наборов указано количество бутылок фанты, штук бананов и тортов, входящих в него, а также стоимость набора. *Необходимо* определить минимальный взнос участника вечеринки и наборы товаров, которые необходимо купить на эти деньги.

22. Острова в море. Расположение островов в море представлено с помощью сетки размера *n* на *n*. Каждый остров обозначается символом '*' в узле этой сетки. Задача заключается в том, чтобы восстановить карту морских островов по закодированной информации о распределении островов по горизонталям и вертикалям. Для объяснения принципа кодирования рассмотрим следующую карту и соответствующие ей коды:

*	-	*	*	-	-	**1 2**
-	*	*	*	-	*	**3 1**
*	-	*	-	*	-	**1 1 1**
-	*	*	*	*	*	**5**
*	*	-	*	-	*	**2 1 1**
-	-	-	*	-	-	**1**
1	**1**	**4**	**2**	**2**	**1**	

1 **1**	**2**		**3**		**2**	

Числа справа от карты являются кодами и представляют порядок и размер групп островов в соответствующих горизонталях сетки. Например, цифры "1 2" в первой строке означают, что первая горизонталь содержит группу из одного острова, за которой следует группа из двух островов. Слева и справа от каждой группы островов расположено произвольной протяженности море. Аналогично, последовательность "1 1 1" первого столбца означает, что первая вертикаль содержит три группы островов, в каждой из которых по одному острову.

Необходимо по кодовой информации о горизонтальном и вертикальном расположении островов восстановить карту (или все карты в случае не однозначности решения). Информация о каждой карте выводится по строкам: строка соответствует одной линии сетки карты, где каждый остров обозначается символом '*', а море – символом '-'. Если карту восстановить невозможно, то выдать число 0.

23. Канадские авиалинии. Вы победили в соревновании, организованном Канадскими авиалиниями. Приз – бесплатное путешествие по Канаде. Путешествие начинается с самого западного города, в который летают самолеты, проходит с запада на восток, пока не достигнет самого восточного города, в который летают самолеты. Затем путешествие продолжается обратно с востока на запад, пока не достигнет начального

города. Ни один из городов нельзя посещать более одного раза, за исключением начального города, который надо посетить ровно дважды (в начале и конце путешествия). Вам также нельзя пользоваться авиалиниями других компаний или другими способами передвижения. *Необходимо* для данного списка городов и списка прямых рейсов между парами городов найти маршрут, включающий максимальное количество городов и удовлетворяющий выше указанным условиям.

24. Блок. В таблице размера *n* на *m*, состоящей из 0 и 1, *необходимо* найти квадратный блок максимального размера, состоящий из одних 1.

25. ***Преобразование строк.*** На вход подаются две символьные строки *A* и *B*. *Необходимо* преобразовать строку *B* в строку *A* с минимальным суммарным штрафом, который определяется следующим образом:

- штраф за удаление символа из строки *B* равен *x* баллов;
- штраф за вставку символа в строку *A* равен *y* баллов;
- штраф за выполнение замены символа в строке *B* на любой другой символ равен *z* баллов.

26. ***Гвозди.*** Имеется деревянная планка, параллельная оси *OX*, в которую вбито n $(n \le 100)$ гвоздей. Известны координаты этих гвоздей $x[i]$ $(i = 1,\ldots,n)$, причем $x[i] \le x[i+1], \forall i = 1,\ldots,n-1$. *Необходимо* привязать к гвоздям веревочки таким образом, чтобы

1) каждая веревочка связывала ровно два гвоздя;
2) к каждому гвоздю была привязана хотя бы одна веревочка;
3) суммарная длина веревочек была бы минимальна.

27. Снегоуборочная машина. Главная улица города представляет собой прямоугольник с вершинами в точках $(0,0)$, $(0,2)$, $(n,2)$, $(n,0)$ разбитый квадраты единичной длины. На некоторых единичных квадратах улицы лежит снег.

*	*		*	*			*
	*	*		*		*	*

Снегоуборочная машина представляет собой отрезок длины 1, который первоначально расположен так, что его концы имеют координаты $(k,0)$ и $(k,1)$. Снегоуборочная машина может за 1 секунду переместиться в горизонтальном направлении на 1 единичный квадрат, при этом ее отрезок "заметает" пройденный квадрат (если на этом единичном квадрате был снег, то он исчезает). Кроме того, машина может мгновенно (не затрачивая на это время) переместиться на 1 единичный квадрат вверх или вниз (при этом отрезок снегоуборочной машины перемещается по прямой, на которой он лежит). *Необходимо* определить за какое минимальное время можно убрать весь снег и указать последовательность действий машины, которая позволяет это сделать. Для указания последовательности действий снегоуборочной машины воспользоваться следующими обозначениями:

символ	*действие*
R	переместиться вправо на клетку
L	переместиться влево на клетку
U	переместиться вверх на клетку
D	переместиться вниз на клетку

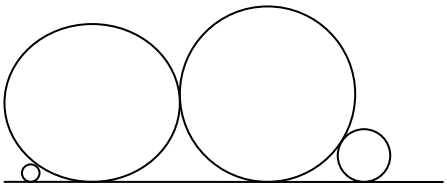
28. Сотовый телефон. Современные сотовые телефоны поддерживают так называемый набор SMS, т.е. коротких текстовых сообщений. К сожалению, клавиатура сотового

телефона имеет только 10 клавиш, а в русском алфавите 33 буквы. Поэтому на одной клавише телефона изображают сразу несколько букв, например на цифре 1 – "*АБВ*", на 2 – "*ГДЕЁ*" и т. д. Тогда, чтобы набрать букву, которая написана на клавише первой, надо нажать на эту клавишу 1 раз, чтобы набрать вторую букву – 2 раза и т. д., чтобы набрать букву, написанную *k*-ой по счету, надо нажать на клавиатуру *k* раз. Ученые подсчитали, что человек в среднем набирает за время жизни букву '*А*' – z_A раз, '*Б*' – $z_Б$ раз и т. д. Чтобы человек не путался, буквы должны быть расположены на клавишах в алфавитном порядке. Однако с целью уменьшения среднего общего количества нажатий на клавиши, требуется так расположить буквы на клавишах, чтобы сумма

$$s = z_A \cdot c_A + z_Б \cdot c_Б + \ldots + z_Я \cdot c_Я$$

была минимальна, где, например, за $c_Б$ обозначен номер буквы '*Б*' на клавише, на которой она находится. *Необходимо* найти такое размещение *m* букв русского алфавита на клавиатуре с *n* клавишами, чтобы минимизировать сумму s.

29. ***Диски.*** Имеется *n* дисков одинаковой толщины с радиусами $R_1, R_2, \ldots, R_n$. Эти диски упаковываются в коробку таким образом, что каждый из них стоит ребром на дне коробки, и все диски находятся в одной плоскости.



Необходимо для последовательности дисков

1) найти длину коробки, в которую все диски могут быть

упакованы, при условии, что диски можно упаковывать только в порядке их следования;

2) определить минимальную длину коробки, если порядок упаковки дисков в коробку может быть произвольным (указать при этом порядок, в котором диски упаковываются).

30. ***Паркет.*** Коридор имеет размеры $3 \cdot n$. Для любого заданного *n*, *необходимо* определить максимальное количество различных вариантов, которыми можно уложить паркет в этом коридоре плитками размером $[1 \times 2]$ или $[2 \times 2]$ (сами варианты укладки паркета определять не надо).

31. ***Валюта.*** Матрица *A* размера *n* на *n* заполнена неотрицательными вещественными числами. Число $a[i, j]$ определяет курс обмена валюты *i* на валюту *j*. Так, например, если $a[i, j] = 2{,}5$, то это значит, что за единицу валюты *i* дают 2,5 единиц валюты *j*. Если $a[i, j] = 0$, то считаем, что курс обмена валюты *i* на валюту *j* прямо не установлен. *Необходимо* определить, можно ли, имея некоторую сумму денег в одной из валют, получить большую сумму денег в той же валюте, совершив несколько обменов.

32. Скачки. План трассы, исходное положение лошади и линия финиша находятся во входном файле.

Блок информации начинается с данных о размерах поля:

- первое число – количество строк n $(3 < n < 21)$; второе – количество столбцов m $(5 < m < 80)$; далее следует кодовая информация о трассе: код для каждой отдельной горизонтали является отдельной строкой файла и представляет собой последовательность цифр:

1 – граница трассы (например, куст);

0 – свободное пространство;
2 – барьер;
3 – финиш;
4 – исходное положение лошади;

• далее идет информация о скоростных характеристиках лошади: первое число задает максимальную скорость *v*, которую может развить лошадь $(1 < v < 10)$; второе число – максимальное количество ходов l $(l > 0)$, которое лошадь может поддерживать максимальную скорость.

Скорость лошади на *n*-ом шаге определяется по формуле: $|X_n - X_{n-1}| + |Y_n - Y_{n-1}|$, где X_{n-1}, Y_{n-1} – положение лошади после *n*-1 хода, а X_n, Y_n – положение лошади после *n*-го хода. Начальная скорость лошади равна 0. На каждом шаге можно изменять скорость лошади: увеличить (уменьшить) скорость на 1 или оставить ее без изменения, а также изменить направление движения лошади. Положение лошади после очередного хода не должно совпадать с границей трассы или с положением барьера (т. е. нельзя приземляться на барьер и на границу трассы) и не должно выходить за границы поля.

Необходимо сначала считать план трассы и вывести его в виде прямоугольной матрицы размера *n* на *m* (при этом символ ' ' (пробел) обозначает свободное пространство, символ '*' (звездочка) – границу трассы, символ *'x'* (икс) – барьеры, символ '+' (плюс) - исходное положение лошади, символ '0' - линию финиша). Затем нужно определить оптимальный маршрут лошади для прохождения всей дистанции и минимальное количество ходов, необходимое ей для этого. Вывести положение лошади на поле после каждого ее хода (символом *'s'*). В случае невозможности прохождения дистанции, вывести соответствующее сообщение.

Уточним сейчас следующие высказывания: "прохождение дистанции" и "изменение скорости движения".

В зависимости от того, какое задано правило изменения скорости, получаются различные подзадачи:

1) можно увеличивать либо уменьшать на 1 скорость лошади только при движении строго по горизонтали;

2) можно увеличивать либо уменьшать на 1 скорость лошади только при движении строго по вертикали;

3) можно увеличивать либо уменьшать на 1 скорость скорость лошади, двигаясь при этом произвольно.

В зависимости от того, что понимать под "прохождением дистанции", возникают следующие подзадачи:

1) остановиться на финише с нулевой скоростью;

2) пересечь линию финиша с любой скоростью (т. е. можно перепрыгнуть финиш, не останавливаясь);

3) оказаться в точке финише с любой скоростью.

33. ***Пещера*****.** Все пещеры удовлетворяют следующим свойствам:

- все гроты и переходы расположены на одном уровне;
- переходы не пересекаются;
- некоторые из гротов расположены на внешнем кольце и называются *внешними гротами*;
- все остальные гроты расположены внутри внешнего кольца и называются *внутренними гротами*;
- имеется вход в пещеру, ведущий к одному из внешних гротов;
- существует ровно три прохода, ведущие от каждого грота к трем другим гротам; если грот является внешним, то два перехода ведут к двум соседним гротам внешнего кольца и точно один переход ведет к внутреннему гроту;
- переходы, связывающие внешние гроты, называются *внешними переходами,* а оставшиеся переходы называются *внутренними переходами*;

- всегда можно пройти от одного грота к другому, используя только внутренние переходы; при этом существует только один такой путь, если разрешается использовать каждый внутренний переход не более одного раза;
- не все переходы равнозначны по сложности преодоления, они делятся на две группы: легкие и сложные.

Было решено открыть все пещеры для посещения. Чтобы обеспечить легкое и безопасное движение по пещере, посетители должны следовать по выбранному заранее пути и посетить каждый из гротов ровно один раз. Исключением является входной грот, где каждый тур начинается и заканчивается. Этот грот посещается дважды. Путь должен содержать как можно меньше опасных переходов. *Необходимо* найти путь через пещеру, который начинается и заканчивается входным гротом, позволяет посетителям увидеть все другие гроты ровно один раз и содержит как можно меньше тяжелых переходов.

34. Доминошки. Доминошка – это прямоугольная плитка, лицевая сторона которой разделена на два квадрата, каждый из которых содержит от 0 до 6 точек. Ряд доминошек выложен на столе:

$$\begin{array}{cccc} \underline{6} & \underline{1} & \underline{1} & \underline{1} \\ 1 & 5 & 3 & 2 \end{array}.$$

Количество точек в верхней строке равно $6+1+1+1=9$ и количество в нижней $1+5+3+2=11$. Разница между нижней и верхней строкой равна $|11-9|=2$. Разница – это абсолютное значение разности двух сумм. Каждая доминошка может быть повернута на $180°$, меняя местами верхний и нижний квадраты. *Необходимо* определить какое минимальное количество поворотов необходимо для минимизации разности? В приведенном примере нужно повернуть последнюю

доминошку для того, чтобы уменьшить разницу до нуля. В этом случае ответом будет 1.

35. Симпатичные узоры. Компания планирует заняться выкладыванием во дворах у состоятельных клиентов узора из черных и белых плиток, каждая из которых имеет размер 1 на 1 метр. Известно, что дворы у всех состоятельных людей имеют наиболее модную на сегодня форму прямоугольника *m* на *n* метров. Однако при составлении финансового плана у директора этой организации появились две проблемы:

- каждый новый клиент, очевидно, захочет, чтобы узор, выложенный у него во дворе, отличался от узоров всех остальных клиентов этой фирмы;
- узор каждого клиента должен быть симпатичным.

Как показало исследование, узор является симпатичным, если в нем нигде не встречается квадрата 2 на 2 метра, полностью покрытого плитками одного цвета.

Так симпатичными являются узоры:

ч б б	ч б ч	б б б
б ч б	б б б	б ч б
б б ч	ч б ч	б б б

Несимпатичными являются узоры:

ч б ч	ч ч б	б б ч
б б б	ч ч б	б б ч
б б ч	б б ч	б б ч

Для составления финансового плана директору *необходимо* узнать, сколько клиентов он сможет обслужить, прежде чем симпатичные узоры данного размера закончатся.

Узоры, получающиеся друг из друга сдвигом, поворотом или отражением, считаются различными.

36. Принцип домино. Из *n* костяшек домино выстроена фигура, которая в дальнейшем будет красиво обрушена. Для

каждой костяшки с номером i известно множество D_i костяшек, которые упадут в момент времени $t+1$, если они остались стоять к этому моменту, а костяшка i падает в момент времени t. *Необходимо* определить, какую костяшку необходимо обрушить в момент времени 0, чтобы фигура падала максимально долго. Требуется также рассчитать время, которое пройдет до полного обрушения фигуры. Если полное обрушение конструкции невозможно, то необходимо выдать соответствующее сообщение.

37. Почта. Вдоль большой широкой дороги располагаются деревни. Дорога представляет собой ось с целочисленными координатами, а позиции каждой деревни соответствует её координата (единственное целое число). Известно, что все деревни расположены в разных местах (нет двух деревень с одинаковой координатой). Расстояние между двумя координатами вычисляется как абсолютная разность их величин. Было решено построить почтовые отделения в деревнях, но не обязательно во всех. Будем считать, что координаты деревни и почтового отделения, построенного в этой деревне, совпадают. Для построения почты в деревнях, координаты почтовых отделений должны быть выбраны таким образом, чтобы сумма расстояний от каждой деревни до ближайшего почтового отделения была минимальна. *Необходимо* по данным координатам деревень и количеству почтовых отделений вычислить минимальную сумму из всех возможных сумм расстояний между деревнями и их ближайшими почтовыми отделениями. Необходимо также определить соответствующие координаты почтовых отделений.

38. Японская полоса. На клетчатой полоске бумаги высотой в одну клетку и длиной n клеток некоторые клетки раскрашены в зеленый и белые цвета. По такой японской полоске строится

код, которым является последовательность чисел: количество подряд идущих зеленых клеток слева направо. При этом количество белых клеток, которыми разделяются группы зеленых клеток, нигде не учитывается (главное, что две группы зеленых клеток, разделяются, по крайней мере, одной белой клеткой). Заметим, что одному и тому же коду могут соответствовать несколько полосок. *Необходимо* найти количество различных японских полосок длины *n*, которые соответствуют заданному коду.

39. Телефонные номера. В окружающем мире вы часто встречаете много телефонных номеров, и они становятся все длиннее. Вам надо запомнить некоторые из телефонных номеров. Один из методов, как это легче сделать, это – сопоставить буквам цифры, как это показано ниже.

цифра	*буквы*
1	I J
2	A B C
3	D E F
4	G H
5	K L
6	M N
7	P R S
8	T U V
9	W X Y
0	O Q Z

Таким образом, некоторое слово или группа слов могут быть сопоставлены одному номеру, и вы можете запоминать последовательность слов вместо номеров. Очевидно, что особенно приятно, если есть возможность найти какую-то простую связь между словом и самим человеком.

Вы легко можете запомнить, что номер вашего друга по

шахматной игре:

9	4	1	8	3	7	2	9	6
W	H	I	T	E	P	A	W	N

и номер доктора собаки:

2	8	5	5	3	0	4
B	U	L	L	D	O	G

Необходимо для некоторого телефонного номера и последовательность, содержащую наименьшее количество слов словаря, которая соответствует данному номеру и данному списку слов. Соответствие цифр и букв приведено выше в таблице.

40. Ребенок. Ребенок нарисовал n $(n \le 100)$ окружностей разных цветов (каждая из окружностей имеет номер от 1 до n). Затем ребенок соединил эти окружности цветными ориентированными отрезками. Каждая пара окружностей может иметь любое количество ориентированных отрезков любых цветов их соединяющих. Каждому цвету (окружности или отрезка) назначен свой собственный положительный уникальный номер (целое число не превосходящее 100). Начиная игру, ребенок, прежде всего, выбирает 3 различных целых числа: l, k и q (все числа в интервале от 1 до n). Затем он ставит одну пешку в окружность с номером l, другую пешку в окружность с номером k, откуда он начинает двигать их, используя следующие правила:

- пешка может быть передвинута только вдоль отрезка, который имеет один цвет с окружностью, в которой находится другая пешка;
- пешка может передвигаться только в направлении отрезка (все отрезки ориентированные);

• две пешки никогда не могут находиться в одной и той же окружности в одно и то же время;

• порядок ходов пешек произвольный (т. е. нет необходимости двигать пешки поочередно);

• игра заканчивается, когда одна из пешек (любая из двух) достигает окружности с номером q.

Необходимо найти кратчайшее (по количеству ходов) решение этой игры. В случае, когда ни одна из пешек не может достичь окружности с номером q, выдать соответствующее сообщение.

41. ***Арифметический показатель.*** Арифметическим показателем натурального числа m по цифре n назовем минимальное количество цифр n, которые необходимо использовать в некотором арифметическом выражении, содержащем только цифры n, арифметические знаки операций: '+', '–', '·', '/', а также скобки: '(' и ')', результат которого равен m. Арифметический показатель будем обозначать через $Ar(m, n)$,

Например, $Ar(17, 3) = 5$, а соответствующим выражением может быть, например, следующее: $3 \cdot (3 + 3) - 3 / 3 = 17$.

Необходимо вычислить арифметический показатель для заданной пары чисел при следующих ограничениях:

1) результат любой промежуточной операции должен быть целым неотрицательным числом, меньшим 1 000;

2) деление числа x на число y допустимо, если y является делителем числа x;

3) вычитание числа y из числа x допустимо, если $y \le x$.

42. ***Лента.*** Задана лента шириной в одну клетку и длиной в n клеток. На каждой клетке написано некоторое

целое число. Играют два игрока, которые ходят поочередно. За один ход игрок отрезает от ленты одну из крайних клеток и забирает ее. Игра останавливается, когда лента заканчивается. Выигрыш игрока равен сумме чисел на находящихся у него клетках ленты. *Необходимо* определить величину выигрыша, которую может себе гарантировать игрок, начинающий игру первым.

Тема 3. Структуры данных

Основные понятия

Для того чтобы разработать эффективный алгоритм решения поставленной задачи, необходимо проанализировать несколько различных структур данных. К основным структурам данных относятся: массивы, списки, стеки, очереди, приоритетные очереди ("кучи") и хеш-таблицы. Каждая из структур данных поддерживает свои базовые операции и, вне зависимости от способа реализации этой структуры, требует выполнять эти операции с заданной трудоемкостью. Может оказаться, что каждая из структур данных позволяет нам выполнять одну из операций легко, а другие - с большим трудом. Поэтому часто выбирают ту структуру, которая лучше всего подходит для решения всей задачи в целом (не делает максимально легким выполнение ни одной операции, но позволяет выполнить всю работу лучше, чем при любом очевидном подходе). Поэтому разработка эффективного алгоритма напрямую связана с разработкой хорошей структуры данных.

Наиболее известной и распространенной структурой данных является *массив*. Эта структура однородна, так как все компоненты имеют один и тот же тип и все из них равнодоступны. К базовым операциям над массивами относятся: поиск и выборочное изменение некоторых

элементов. Операция поиска некоторого элемента x имеет трудоемкость в произвольном массиве - $O(n)$, а в отсортированном массиве - $O(\log n)$ (поиск делением пополам). Следует отметить, что основные операции над массивами не предполагают включения или исключения некоторых элементов. Однако на практике часто необходимо выполнять эти операции, что приводит к необходимости разработки структур данных, поддерживающих такие операции. К одной из таких структур данных относится список.

Список - конечная последовательность элементов, порядок которых определяется с помощью ссылок. К базовым операциям для списка относятся:

1) поиск некоторого заданного элемента с ключом x; трудоемкость $O(n)$;

2) поиск максимального (минимального) элемента; трудоемкость $O(n)$;

3) включение некоторого элемента; трудоемкость $O(1)$;

4) исключение некоторого элемента; трудоемкость $O(1)$;

5) формирование списка - трудоемкость $O(n)$;

6) просмотр списка; трудоемкость $O(n)$.

Существуют различные способы реализации списка. Наиболее распространенными являются: реализация в виде двух массивов и с использованием последовательно связанных компонент.

Иногда при работе со списковой структурой возникает потребность включать и исключать элементы в определенном порядке.

Если реализуется принцип "последний вошел - первый вышел" (*LIFO*), то такую списковую структуру называют *стеком*. Трудоемкость процедур включения и исключения элемента из стека есть $O(1)$.

Очередь является еще одним специальным видом списка, когда элементы всегда включаются в конец списка, а исключаются из начала списка, т. е. реализуется принцип "первый вошел – первый вышел" (*FIFO*). Трудоемкость добавления и удаления элемента из очереди есть $O(1)$.

Часто в задачах возникает необходимость выполнять операцию удаления минимального элемента. Ни одна из рассмотренных ранее структур данных не выполняла эту операцию эффективно. *Приоритетной очередью* (или "*кучей*") будем называть такую структуру данных, которая позволяет выполнять две базовые операции:

- добавление элемента;
- поиск и удаление минимального элемента.

Предполагается, что для каждого элемента определено некоторое значение (ключ), по которому определяется "минимальность". Существует несколько способов реализации структуры данных куча: бинарные кучи, биномиальные кучи и др. Основным свойством структуры данных куча является условие, что элементы в ней организованы таким образом, что приоритет любой вершины не ниже приоритета каждого из ее сыновей. Так, например, если мы реализуем кучу в виде полного бинарного дерева, для которого выполняется основное свойство структуры данных куча, то получим бинарную кучу. Если реализовать кучу в виде семейства биномиальных деревьев, в котором не существует двух деревьев одинаковой высоты, а для каждого из деревьев выполняется основное свойство структуры данных куча, то получим биномиальную кучу. Далее в таблице указаны трудоемкости основных базовых операций для бинарной и биномиальной куч.

Таблица 1.

Способ	Создание	Удаление	Поиск	Добавление

реализации	кучи	минимального элемента	минимального элемента	элемента
Бинарная куча	O(1)	O(log *n*)	O(1)	O(log *n*)
Биномиальная куча	O(*n*)	O(log *n*)	O(1)	O(log *n*)

Если в задаче необходимо выполнять только словарные операции поиска, добавления и удаления некоторого элемента, то часто применяют так называемое хеширование, а соответствующая структура данных называется *хеш-таблицей*. Если выполнять эти операции (для последовательности из n элементов) с использованием массивов, то трудоемкость каждой из них в среднем равна $O(n)$. Подобная трудоемкость достигается и при использовании списков. Если использовать двоичные вектора, то все эти три операции выполняются за фиксированное время, независимо от размера множества, но в этом случае элементами обрабатываемой последовательности могут быть только целые числа из небольшого конечного интервала. В худшем случае поиск в хеш-таблице может занимать столько же времени, сколько поиск в некотором списке, но на практике хеширование достаточно эффективно. При выполнении некоторых естественных условий среднее время выполнения базовых операций есть $O(1)$. Существуют две основные формы хеширования: открытое (внешнее хеширование) и закрытое (внутреннее хеширование).

Основная идея *открытого хеширования* заключается в том, что исходное множество (возможно бесконечное) разбивается на конечное число классов. Для этих m классов, пронумерованных от 0 до $m-1$, строится хеш-функция h такая, что для любого элемента x исходного множества функция $h(x)$ принимает целочисленное значение из интервала 0 до $m-1$, которое соответствует классу, которому принадлежит элемент

x. Элемент x называют ключом, $h(x)$ – хеш-значением, а классы – сегментами. Если для некоторых двух ключей хеш-значения совпадают, то говорят, что произошла коллизия. При прямом хешировании разрешение коллизий происходит с помощью цепочек, т. е. элементы, которым соответствует одно и то же хеш-значение, связываются в сегменте в цепочку-список, а соответствующие элементы хеш-таблицы содержат адреса цепочек-списков. На практике при выборе хеш-функции пользуются различными эвристиками. Например, один из способов построения функции - деление с остатком:

$$h(x) = x \bmod m .$$

При *закрытом хешировании* в таблице сегментов T хранятся непосредственно ключи, а не заголовки списков. Поэтому в каждом сегменте хранится только одно число. При этом способе хеширования применяется методика повторного хеширования (выполняется последовательность проб), которая заключается в следующем: если мы пытаемся поместить элемент в сегмент $h(x)$, который уже занят другим ключом, то выбирается последовательность других номеров сегментов $h_1(x), h_2(x), \ldots$, куда можно поместить данное ключевое значение. Каждое из местоположений последовательно проверяется, пока не будет найдено свободное место. Если свободных мест нет, то таблица полностью заполнена, и элемент вставить нельзя. Обычно применяют три способа вычисления последовательности испробованных мест: линейный, квадратичный и двойное хеширование. Например, функция для линейной последовательности проб задается формулой:

$$h(x,i) = \left(h'(i) + i\right) \bmod m,$$

где $h'(x)$– некоторая хеш-функция. Таким образом, при работе с ключом *x* начинают с ячейки $T(h'(x))$, а затем перебирают ячейки таблицы подряд $T(h'(x)+1)$, $T(h'(x)+2)$ и т. д. После ячейки $T(m-1)$ переходят к ячейке $T(0)$.

Список задач

1. Считалка. Вокруг считающего стоят *n* человек, из которых выделен первый, а остальные занумерованы по часовой стрелке числами от 2 до *n* . Считающий, начиная с кого-то, ведет счет до *m*. Человек, на котором остановился счет, выходит из круга. Счет продолжается со следующего человека и так до тех пор, пока не останется один человек.

Необходимо определить

1) номер оставшегося человека, если известно *m* и то, что счет начинался с первого человека;

2) номер человека, с которого начинался счет, если известно *m* и номер оставшегося человека *l*.

2. Полоска. Задана полоска длиной 2^k клеток и шириной в одну клетку. Полоску сгибают пополам так, чтобы правая половинка оказалась под левой. Сгибание продолжают до тех пор, пока сверху находится больше одной клетки. *Необходимо* пронумеровать клетки таким образом, чтобы после окончания сгибания полосы номера клеток в получившейся колонке были расположены в порядке $1, 2, \ldots, 2^k$.

3. Квадрат. Квадрат разбит на 4^k равновеликих квадратных клеток. Квадрат перегибается поочередно относительно вертикальной, (правая половина подкладывается под левую) и

горизонтальной (нижняя половина подкладывается под верхнюю) оси симметрии до тех пор, пока все клетки не будут расположены друг под другом. *Необходимо* занумеровать клетки исходного квадрата таким образом, чтобы в результате выполнения операций перегиба номера клеток, расположенных друг под другом, образовали, начиная с верхней клетки, числовую последовательность: $1, 2,\ldots,4^k$.

4. Таблица. В массиве *A* размера *n* за одно обращение к каждому элементу массива *необходимо* каждый элемент заменить ближайшим большим следующим за ним. Если такого элемента нет, то необходимо заменить его нулем. Можно использовать дополнительную память.

5. Остановки. В городе имеется *n* автобусных остановок. Остановки пронумерованы числами от 1 до *n*. Имеется *r* автобусных маршрутов, которые задаются последовательностями соседних остановок при движении автобуса в одном направлении:

$$M_1 = \{I_{11}, I_{12}, \ldots, I_{1m_1}\},$$
$$M_1 = \{I_{11}, I_{12}, \ldots, I_{1m_2}\},$$
$$\ldots$$
$$M_1 = \{I_{11}, I_{12}, \ldots, I_{1m_r}\},$$

где l_{jk} натуральные числа.

Время движения между соседними остановками у всех маршрутов одинаково и в 3 раза меньше времени изменения маршрута. Кроме того, автобусы могут двигаться в обоих направлениях и на маршруте некоторые остановки могут повторяться $(1 \to 2 \to 4 \to 3 \to 2 \to 6)$.

Необходимо по заданным номерам остановок *i* и *j* определить наиболее быстрый путь перемещения пассажира из

остановки i в остановку j с использованием имеющихся маршрутов автобусов.

6. Король. Одинокий король долго ходил по бесконечной шахматной доске. Известна последовательность из n его ходов (вверх, вниз, влево, вправо, вверх влево и т. д.). *Необходимо* за минимально возможное число вычислений определить, побывал ли король дважды на одном и том же поле, если направление каждого хода занумеровано цифрами

8	1	2
7	+	3
6	5	4

(символом '+' помечена позиция короля).

7. Карточки. Имеется n карточек черного и белого цвета, которые сложены в стопку. Карточки раскладываются на стол в одну линию следующим образом: первая кладется на стол, вторая под низ стопки, третья - на стол, четвертая - под низ стопки и т. д., пока все карточки не будут выложены на стол. *Необходимо* определить, каким должно быть начальное расположение карточек в стопке, чтобы разложенные на столе карточки чередовались по цвету: белая, черная, белая, черная и т. д. (выдать для заданного n последовательность из 0 и 1, которая определяет начальное расположение карточек; при этом цифра 0 соответствует белой карточке, а цифра 1 – черной).

8. Форма с погружением. Форма тела задана матрицей A из n строк u m столбцов. Элементы матрицы есть натуральные числа. Элемент матрицы $A[i, j]$ соответствует высоте горизонтальной квадратной площадки размера 1×1

относительно нижнего основания. Нижнее основание формы горизонтально. *Необходимо* определить объем невытекшей воды, если тело опускается полностью в воду, а затем поднимается.

Например, после подъема формы, заданной матрицей,

5	3	1
5	**2**	5
2	5	5

вода останется только во внутреннем углублении, над элементом $A[2,2]$. Объем не вытекшей воды будет равен 1.

9. Форма без погружения. Форма тела задана матрицей *A* из *n* строк *и m* столбцов. Элементы матрицы есть натуральные числа. Элемент матрицы $A[i, j]$ соответствует высоте горизонтальной квадратной площадки размера 1×1 относительно нижнего основания. Нижнее основание формы горизонтально. *Необходимо* определить объем невытекшей воды, если в позицию $A[i_0, j_0]$ выливается объем воды *V*. Предполагается, что вода сначала стремится занять углубления формы и лишь затем вытекает за ее поверхность.

10. Лабиринт и конкретный выход. Матрица из *n* строк и *m* столбцов определяет некоторый лабиринт. В матрице элемент 1 обозначает стену, а 0 определяет свободное место. В первой строке матрицы определяются входы $x[i]$, а в последней - выходы $y[i]$ $(i = 1,\ldots,k)$, которые должны быть нулевыми элементами. Однако не каждый нулевой элемент первой (последней) строки является входом (выходом). Возле каждого из входов $x[i]$ стоит один человек, который может выйти только через выход $y[i]$. Движение в лабиринте

осуществляется только по вертикали или горизонтали. *Необходимо* определить, какое максимальное количество человек можно провести по лабиринту, чтобы каждое свободное место посещалось не более одного раза.

11. Лабиринт и произвольный выход. Матрица из n строк и m столбцов определяет некоторый лабиринт. В матрице элемент 1 обозначает стену, а 0 определяет свободное место. В первой строке матрицы определяются входы $x[i]$, а в последней - выходы $y[i]$ $(i = 1,\ldots,k)$, которые должны быть нулевыми элементами (однако не каждый нулевой элемент первой (последней) строки является входом (выходом)). Возле каждого из входов $x[i]$ стоит один человек. Движение в лабиринте осуществляется только по вертикали или горизонтали. *Необходимо* определить какое максимальное количество людей можно провести по лабиринту, чтобы каждое свободное место лабиринта посещалось не более одного раза и, учитывая, что

1) человека можно выводить через любой из выходов;
2) через один выход могут выйти несколько человек.

12. Чиновники. Есть министерство из n чиновников. Каждый из чиновников министерства может иметь только одного непосредственного начальника и произвольное количество подчиненных. Однако в министерстве работает только один чиновник, не имеющий начальников – главный чиновник. Для того чтобы предприниматель мог получить некоторую лицензию, необходимо получить подпись на документе главного чиновника. Каждый чиновник для выставления своей подписи требует подпись на документе хотя бы одного из своих непосредственных подчиненных и, может быть, некоторую взятку – известное число у. е.

Необходимо определить какое минимальное количество денег предпринимателю нужно заплатить, чтобы получить лицензию, и указать соответствующий этой сумме порядок получения подписей.

13. Караван с движением по меньшей высоте. Имеется план местности, разбитой на квадраты, заданный матрицей $A[n \times n]$. Каждый квадрат с координатами $[i, j]$ имеет высоту относительно уровня моря, значение которой определяется натуральным числом $A[i, j]$. Караван может двигаться по местности только параллельно осям *Ox* и *Oy* между центрами квадратов и только в соседний квадрат с меньшей высотой. *Необходимо* определить наименьший (по количеству перемещений между квадратами) маршрут каравана из позиции (x_1, y_1) в позицию (x_2, y_2).

14. ***Караван с движением по крутизне*** $\le k$. Имеется план местности, разбитой на квадраты, заданный матрицей $A[n \times n]$. Каждый квадрат с координатами $[i, j]$ имеет высоту относительно уровня моря $A[i, j]$. Караван может двигаться по местности только параллельно осям *Ox* и *Oy* между центрами квадратов. Перемещение в соседний квадрат возможно лишь при условии, что крутизна подъема (спуска) не будет превышать величины *k* (при переходе в соседний квадрат крутизна подъема (спуска) равна модулю разности высот квадратов). *Необходимо* определить наименьший (по количеству перемещений между квадратами) маршрут каравана из позиции (x_1, y_1) в позицию (x_2, y_2).

15. Минимальная длина маршрута робота. Имеется план местности, разбитой на квадраты, заданный матрицей

$A[n \times n]$. Каждый квадрат с координатами $[i, j]$ имеет высоту относительно уровня моря, значение которой определяется натуральным числом $A[i, j]$. Робот может двигаться по местности только параллельно осям *Ox* и *Oy* между центрами квадратов. При переходе в соседний квадрат длина подъема (спуска) равна модулю разности высот квадратов, а длина перемещения из квадрата в квадрат равна величине *k*. Длина всего маршрута робота определяется как суммарная длина подъемов и спусков плюс суммарная длина перемещений из квадрата в квадрат. *Необходимо* определить маршрут робота из позиции (x_1, y_1) в позицию (x_2, y_2) минимальной длины.

16. Минимальное количество ходов коня. Имеется шахматная доска *n* на *n* клеток. Некоторые поля на ней заняты белыми фигурами (каждое занятое поле определяется числом -1). *Необходимо* определить какое минимальное количество ходов белого коня требуется для его перемещения с поля (x_1, y_1) на поле (x_2, y_2) и вывести его маршрут.

17. Минимальное количество ходов белого коня со сбитием черных фигур. Имеется шахматная доска *n* на *n* клеток. Некоторые поля на ней заняты белыми и черными фигурами (каждое занятое поле белыми фигурами определяется числом -1, а черными фигурами - числом 1). Белый конь ходит по шахматной доске и ему разрешено сбивать черные фигуры. Сбитие черной фигуры считается дополнительным ходом (т. е. ход на поле, занятое черной фигурой, "стоит" 2 хода). *Необходимо* определить маршрут белого коня с поля (x_1, y_1) на поле (x_2, y_2), при котором количество его ходов минимально.

18. Минимальное количество белых коней. Имеется шахматная доска n на n клеток. Некоторые поля на ней заняты белыми фигурами (не конями). Каждое занятое поле определяется числом -1. *Необходимо* определить минимальное количество белых коней, которое необходимо расставить на доске, чтобы при постановке черной фигуры в любое оставшееся свободное поле она могла бы быть сбита одним из этих коней за некоторое количество ходов.

19. Минимальное количество белых ладей. Имеется шахматная доска n на n клеток. Некоторые поля не ней заняты белыми фигурами (не ладьями). Каждое занятое поле определяется числом -1. *Необходимо* определить минимальное количество белых ладей, которое необходимо расставить на доске, чтобы при постановке черной фигуры в любое оставшееся свободное поле она могла бы быть сбита одной из этих ладей за некоторое количество ходов.

20. Минимальное количество белых слонов. Имеется шахматная доска n на n клеток. Некоторые поля не ней заняты белыми фигурами (не слонами). Каждое занятое поле определяется числом -1. *Необходимо* определить минимальное количество белых слонов, которое необходимо расставить на доске, чтобы при постановке черной фигуры в любое оставшееся свободное поле она могла бы быть сбита одним из этих слонов за некоторое количество ходов.

21. Цилиндр. Лист клетчатой бумаги размером n на m сначала склеили в цилиндр высотой n, а затем из него вырезали некоторые из клеток. *Необходимо* определить, на какое количество кусков распадется цилиндр. Например, если из шахматной доски удалить все клетки одного цвета, то доска распадется на 32 куска.

22. Прямоугольники. На белом прямоугольном листе бумаги, имеющем размеры *a* см в ширину и *b* см в длину (*a* и *b* - положительные четные целые числа, не превосходящие 30), располагается *n* прямоугольников различных цветов. Стороны прямоугольников параллельны краям листа, а сами прямоугольники не выходят за его пределы и могут накладываться друг на друга. Если два прямоугольника одного цвета имеют хотя бы одну общую точку, то они являются частями одной фигуры. В результате образуются различные одноцветные фигуры. Для каждого из прямоугольников задаются целочисленные координаты точек, в которые помещены его левая нижняя и правая верхняя вершины, а также цвет (целое число от 1 до 64; белый цвет представлен числом 1). Данные о прямоугольниках следуют в том порядке, в котором они выкладывались на лист бумаги. *Необходимо* вычислить площадь каждой из видимых одноцветных фигур (включая и белые фигуры), если предполагается, что начало системы координат находится в центре листа, а оси координат параллельны краям листа (для каждой из одноцветных фигур сначала выдается ее цвет, а затем - площадь).

23. Замок. Имеется замок, который разделен на комнаты (имеется, по крайней мере, две комнаты). Условно замок можно представить замок разделенным на *m* x *n* клеток и его план задать следующим образом:

• *m* - число клеток в направлении с севера на юг $(m \le 50)$ и *n* число клеток в направлении с запада на восток $(n \le 50)$;

• каждая клетка замка будет описываться одним числом *p* $(0 \le p \le 15)$; это число является суммой следующих чисел: 1, если клетка имеет западную стену, 2 – северную, 4 –

восточную, 8 – южную (внутренняя стена принадлежит обеим клеткам).

Необходимо для данного замка определить количество комнат в нем, площадь наибольшей комнаты, а также определить, какую стенку в замке нужно разрушить, чтобы получить комнату наибольшей площади (выдать площадь получившейся комнаты).

24. Компании. Некоторые из компаний (число компаний не превышает 100) являются совладельцами других компаний, так как приобрели часть их акций. Говорят, что компания *A* контролирует компанию *B*, если имеет место, по меньшей мере, одно из следующих условий:

1) $A = B$;

2) *A* владеет более чем 50% акций *B*;

3) *A* контролирует k $(k > 0)$ компаний $C_1, C_2, \ldots, C_k$ таких, что компания C_i владеет соответственно X_i процентами акций компании B $(i = 1, \ldots, k)$ и $X_1 + X_2 + \ldots + X_k > 50\%$.

Необходимо определить все пары чисел (h, s), при которых компания h контролирует компанию s.

25. ***Кубик***. Задано поле *A*, размера *n* на *m*, разбитое на квадраты единичной длины. Каждому квадрату поля с координатами $[i, j]$ ставится в соответствии некоторое положительное число $A[i, j]$ (т. е. задана матрица целых положительных чисел). Есть кубик с 6-ю гранями (грань кубика - это квадрат единичной длины). На *i*-ой грани кубика написано некоторое число $s[i] > 0$ $(i = 1, \ldots, 6)$. Если в некоторый момент времени кубик лежит гранью *k* на единичном квадрате с координатами $[i, j]$, то за прохождение

кубиком этого поля взимается штраф $|A[i,j]-s[k]|$. Кубик перемещается по полю, путем его переворачиваний с одной грани на любую другую из 4-х соседних, и за один такой поворот оказывается на соседнем единичном квадрате поля. *Необходимо* определить, какой гранью (и какой ориентацией) должен лежать кубик в позиции поля с координатами (x_1, y_1), чтобы суммарный штраф, который нужно заплатить за перекатывание кубика в позицию поля с координатами (x_2, y_2), был минимален.

26. ***Черный ящик***. Черный ящик организован наподобие примитивной базы данных. Он может хранить набор целых чисел и имеет выделенную переменную *i*. В начальный момент времени черный ящик пуст, а значение переменной *i* равно нулю. Черный ящик обрабатывает некоторую последовательность поступающих команд (запросов). Существует два вида запросов:

1) Add (*x*) - положить в черный ящик элемент *x*;

2) Get - увеличить значение переменной *i* на 1 и выдать копию *i*-го минимального элемента черного ящика.

Напомним, что *i*-ым минимальным элементом последовательности называется число, которое стоит на *i*-ом месте в отсортированной по не убыванию последовательности.

Пример.

Запрос	*i*	Черный ящик	Ответ
Add (3)	0	3	
Get	1	3	3
Add (1)	1	1 3	
Get	2	1 3	3
Add (-4)	2	-4 1 3	
Add (2)	2	-4 1 2 3	

Add (8)	2	-4 1 2 3 8	
Add (-1000)	2	-1000 -4 1 2 3 8	
Get	3	-1000 -4 1 2 3 8	1
Get	4	-1000 -4 1 2 3 8	2

Необходимо разработать эффективный алгоритм, обрабатывающий последовательность из n запросов Get и m запросов Add (x) и имеющий трудоемкость не более, чем $O(m \log m)$.

27. Межшкольная сеть. Некоторые школы связаны компьютерной сетью. Между школами заключены соглашения: каждая школа имеет список школ-получателей, которым она рассылает программное обеспечение всякий раз, получив новое бесплатное программное обеспечение (извне сети или из другой школы). При этом если школа *B* есть в списке получателей школы *A*, то школа *A* может и не быть в списке получателей школы *B*. *Необходимо* решить следующие две подзадачи.

1. Определить минимальное количество школ, которым надо передать по экземпляру нового программного обеспечения, чтобы распространить его по всем школам сети в соответствии с соглашениями**.**

2. Надо обеспечить возможность рассылки нового программного обеспечения из любой школы по всем остальным школам. Для этого можно расширять списки получателей некоторых школ, добавляя в них новые школы. Требуется найти минимальное суммарное количество расширений списков, при которых программное обеспечение из любой школы достигло бы всех остальных школ. Одно расширение означает добавление одной новой школы-получателя в список получателей одной из школ.

28. Расписание. Имеется одна машина и n работ, которые должны быть выполнены на этой машине. Одновременно машина может выполнять только одну работу, и если работа выполняется, то она должна быть завершена полностью. Для каждой работы i определены две величины:

1) длительность выполнения $p(i)$;

2) директивный срок $d(i)$ (это тот срок, к которому работу надо выполнить от момента начала выполнения всех работ).

Некоторые из работ должны предшествовать другим. Правило предшествования работ задается ациклическим орграфом: дуга (i, j) этого графа означает, что работа i должна быть выполнена раньше, чем работа j.

Если задана некоторая последовательность выполнения работ X, то $c^X(i)$ будет обозначать время завершения работы i в этой последовательности.

Необходимо определить в какой последовательности Y должны обрабатываться работы для того, чтобы минимизировать функцию:

$$\max\{c^Y(j) - d(j), 0\}, \quad j = 1, \ldots, n$$

и вычислить суммарный штраф для этой последовательности работ. Штраф работы j для последовательности работ Y определяется как величина $\max\{c^Y(j) - d(j), 0\}$.

29. Пересечение реки. На реке имеется n островков, пронумерованных числами от 1 до n слева направо поперек реки. Жители должны пересечь реку, начиная с ее левого берега, используя некоторые островки, для достижения правого берега. Левый берег расположен на одну позицию левее

первого островка, а правый берег расположен на одну позицию правее островка с номером *n*. В момент времени $t = 0$ житель находится на левом берегу реки и имеет целью добраться до правого берега за минимальное время.

В каждый момент времени каждый из островков поднят или опущен, а житель стоит на островке или на берегу. Житель может стоять на островке только тогда, когда он поднят. Такой островок называется доступным.

В начальный момент времени каждый островок *i* опущен, затем островок поднят a_i моментов времени, затем опущен b_i моментов времени, поднят a_i моментов, опущен b_i моментов и т. д. Константы a_i и b_i задаются отдельно для каждого островка *i*. Например, если для *i*-го островка $a_i = 2$ и $b_i = 3$, то он опущен в момент времени 0, поднят в моменты времени 1 и 2, опущен в моменты времени 3, 4 и 5 и т. д.

В момент времени $t + 1$ житель может выбрать

1) островок или берег в пределах 5 ближайших с каждой стороны от его местоположения в момент времени *t*;

2) остаться на месте (если возможно).

Необходимо вычислить минимальный момент достижения правого берега, если это возможно.

30. Шланги. Два шланга разных цветов перепутаны между собой. Заданы координаты точек, в которых они перепутаны, кроме того, для каждой такой точки указано, какой из шлангов находится сверху (точки указываются в порядке следования вдоль одного из шлангов). Имеются две стенки, расположенные одна напротив другой. Каждый из запутанных шлангов одним концом закрепляется на одной стенке, а другим – на противоположной стенке. *Необходимо* определить, можно ли распутать шланги, не освобождая их концы.

31. Химическая реакция. Есть n различных химических веществ, которые занумерованы целыми числами от 1 до n. Матрица A размера n на n задает результаты химических реакций этих веществ. Элемент матрицы $A[i, j]$ равен номеру того вещества, которое получится в результате химической реакции веществ i и j (если вещества не вступают в реакцию, то $A[i, j] = 0$). Имеется пробирка, в которую последовательно добавляются химические вещества во время эксперимента. Вещество i вступает в химическую реакцию с веществом j, если находится в пробирке непосредственно над ним. Вещества, которые не вступают в реакцию друг с другом, в пробирке не смешиваются. *Необходимо* определить, какие вещества и в какой последовательности будут находиться в пробирке после окончания эксперимента (указать номера веществ, находящихся в пробирке, начиная от верхнего уровня и заканчивая самым нижним).

Тема 4. Графы

Основные понятия

Граф $G = (V, E)$ состоит из непустого множества узлов V и множества пар узлов (вершин) E. В дальнейшем будем предполагать, что $|V| = n$ и $|E| = m$.

Если множество E представлено в виде упорядоченных пар узлов (v, w), то граф называется ориентированным (орграфом), а упорядоченная пара узлов (v, w) называется *дугой*; в противном случае граф называется неориентированным, а пара узлов называется *ребром*.

Неориентированный (ориентированный) граф называется *связным* (*сильно связным*), если для любой пары вершин существует путь, их соединяющий. В противном случае, граф называется *несвязным*.

Если v - некоторая вершина графа, то максимальное количество вершин, в которые существует путь из v, порождают *компоненту связности* графа.

Граф называется *двудольным*, если множество его вершин можно разбить на два подмножества (доли) таким образом, что каждое ребро соединяет только вершины различных подмножеств.

Паросочетанием в графе называется такое подмножество его ребер, что никакие два ребра не имеют общей вершины. Паросочетание, которое покрывает все вершины графа, называется *совершенным*.

Существует несколько способов представления графа в памяти машины. В таблице 2 приведены некоторые из способов представления графов и указаны преимущества и недостатки каждого из них (через *out_v* обозначается множество дуг орграфа, выходящих из вершины v).

Многие практические задачи могут быть сформулированы в терминах задач на графах. Рассмотрим следующие алгоритмы:

1) поиск в ширину;
2) поиск в глубину;
3) максимальный поток и его приложения;
4) кратчайший путь в графе.

Поиск в глубину в графе использует структуру данных *стек* и имеет трудоемкость $O(n+m)$. Этот поиск может быть применен для нахождения произвольного пути между некоторой вершиной графа и всеми оставшимися вершинами, для определения связности неориентированного графа, построения максимального по количеству ребер пути. Модификация поиска в глубину в ориентированном графе с

использованием обратных и поперечных дуг используется для выделения сильно-связных компонент орграфа. Используя поиск в глубину, можно также легко определить, является ли граф (орграф) двудольным.

Поиск в ширину в графе использует структуру данных *очередь* и имеет трудоемкость $O(n+m)$. Используя данный алгоритм, мы можем найти наименьший по количеству ребер (дуг) путь между некоторой вершиной и всеми оставшимися. Поиск в ширину используется для выделения связных компонент неориентированного графа и для определения двудольности.

Таблица 2.

Способ задания	Объем памяти	Существование дуги (v,w).	Построение множества out_v	Преимущества и недостатки
Матрица смежности	$O(n \cdot n)$	$O(1)$	$O(n)$	Большой объем памяти. Наилучшая трудоемкость определения существования дуги .
Матрица инцидентности	$O(n \cdot m)$	$O(m)$	$O(n \cdot m)$	Память экономится, но возрастает трудоемкость основных операций.
Списки пар	$O(m)$	$O(m)$	$O(m)$	Большая трудоемкость определения существования дуги. Существенная экономия памяти.
Списки смежности	$O(n+m)$	$O(\|out\_v\|)$	$O(\|out\_v\|)$	Наилучшая из всех способов

				задания трудоемкость построения *out_v*.

Максимальный поток в графе. Предположим, что каждой дуге e графа G приписана некоторая пропускная способность $c(e)$. Выделены две вершины: s (в которую нет входящих дуг - источник) и t (из которой нет выходящих дуг - сток). Таким образом, мы имеем некоторую сеть. *Потоком* в сети называется такая функция $f : E \to R$, которая удовлетворяет следующим свойствам:

1) поток по дуге не превосходит пропускной способности дуги;

2) количество потока, выходящего из вершины s, равно количеству потока, входящего в вершину t;

3) для каждой промежуточной вершины w (отличной от s и t) количество входящего потока равно количеству выходящего потока.

Величиной потока называется количество потока, выходящего из вершины s. Максимальный поток – поток, величина которого максимальна. Идея алгоритма построения максимального потока состоит в построении увеличивающего пути. Для построения увеличивающего пути используется расширенный поиск в ширину (глубину) по прямым недогруженным дугам и обратным дугам, поток по которым больше 0.

Используя алгоритм построения максимального потока, могут быть решены следующие задачи: нахождение максимального количества путей в графе не пересекающихся по ребрам (вершинам), построение максимального паросочетания в двудольном графе, определение допустимой циркуляции и др.

Поиск кратчайшего пути в графе. Кратчайшим путем в графе называется такой путь между парами вершин, суммарная стоимость ребер (дуг) которого минимальна. На практике, когда стоимости ребер положительны и требуется найти кратчайший путь из некоторой вершины во все оставшиеся, используют алгоритм Дейкстры или его эффективную реализацию с использованием структуры данных *приоритетная очередь*. Трудоемкость этой эффективной реализации равна $O(m+n+m\log m)$, а трудоемкость алгоритма Дейкстры - $O(n^2)$. Таким образом, когда граф является полным $(m=O(n^2))$, то более эффективно является стандартная реализация алгоритма Дейкстры. В случае же графа, у которого $m=O(n)$ (например, планарные графы), более предпочтительна реализация алгоритма Дейкстры с помощью куч.

В некоторых практических задачах кратчайший путь может проходить через некоторые вершины графа несколько раз, а по каждой дуге (ребру) графа - не более одного раза. В таких случаях при реализации алгоритма Дейкстры можно использовать приоритетную очередь, в которую заносят дуги (ребра).

Эффективная реализация алгоритма Дейкстры оказывается полезной и при построении *k*-го минимального пути в графе. В этом случае вершина, которая на некотором шага алгоритма удаляется из приоритетной очереди, будет считаться просмотренной (игнорироваться в дальнейшем) лишь при ее *k*-ом удалении из очереди.

Список задач

1. Роботы с различными скоростями. Между n пунктами $(n \le 50)$ заданы дороги единичной длины. Дороги проложены на разной высоте и пересекаются только в общих пунктах. В начальный момент времени из заданных пунктов начинают двигаться с постоянной скоростью m роботов (m равно 2 или 3), независимо меняя направление движения только в пунктах. Роботы управляются таким образом, чтобы минимизировать время до встречи всех роботов в одном месте. Скорость i-го робота может быть равна 1 или 2. Остановка роботов запрещена. *Необходимо* при заданных n, m определить минимальное время, через которое может произойти встреча всех m роботов, если начальное положение роботов и скорость их движения известны. В случае невозможности встречи всех m роботов в одном месте должно быть сформировано соответствующее сообщение.

2. Робот на 90 градусов. На плоскости расположено n точек. Имеется робот, который двигается следующим образом. Стартуя с некоторой начальной точки и имея некоторое начальное направление, робот движется до первой встреченной на его пути точки, изменяя в ней свое текущее направление на 90 градусов, т. е. поворачивая налево или направо. После этого он продолжает движение аналогичным образом. Координаты точек, расположенных на плоскости, являются целыми числами, а угол измеряется в радианах относительно оси Ox. *Необходимо* определить, может ли робот посетить все n точек ровно один раз и обязательно вернуться в точку старта (начальная точка посещается дважды: в начале и конце пути), если:

1) определены начальные точка и направление движения робота;

2) определена начальная точка, а начальное направление движения робота можно выбирать;

3) начальную точку и направление робота можно выбирать.

3. Максимальное количество путей. Задан граф, в котором выделены две вершины *v* и *w*. *Необходимо* найти максимальное количество путей между этими двумя вершинами, не пересекающихся по

1) ребрам;

2) вершинам.

4. Лабиринт. Лабиринт задается матрицей смежности *C* размера *n* на *n*, где элемент $C[i, j] = 1$, если узел *i* связан узлом *j* посредством дороги (*i* и *j* – целые числа от 1 до *n*). Часть узлов назначается входами, часть - выходами. Входы и выходы задаются последовательностями узлов $x(1), x(2), \ldots, x(p)$ $(p \le n)$ и $y(1), y(2), \ldots, y(k)$ $(k \le n)$ соответ-ственно. Возле каждого входа стоит произвольное количество людей. *Необходимо* определить какое максимальное количество людей можно провести от входов к выходам таким образом, чтобы

1) их пути не пересекались по дорогам, но могли пересекаться по узлам;

2) их пути не пересекались по узлам.

Следует учесть, что через некоторый выход можно вывести произвольное количество человек.

5. Шестеренки. Заданы *n* шестеренок, которым присвоены номера - числа от 1 до *n* . Задано *m* соединений пар шестеренок в виде пар (i, j) (шестеренка с номером *i* находится в зацеплении с шестеренкой с номером *j*). *Необходимо* определить, можно ли повернуть шестеренку с

номером 1. Если "да", то найти количество шестеренок, которые пришли при этом в движение.

6. Открытки и конверты. Имеется *n* прямоугольных конвертов и *n* прямоугольных открыток различных размеров. *Необходимо* определить, можно ли разложить все открытки по конвертам так, чтобы в каждом конверте было по одной открытке. Открытки нельзя складывать и, но можно помещать в конверт под углом.

Например, открытка с размерами сторон 5:1 помещается в конверты с размерами 5:1, 6:3, но не входит в конверты с размерами 4:1, 10:0,5, 4,2:4,2.

7. Разбиение на команды по численности. Имеется группа из *n* студентов. Некоторые из студентов знают друг друга. Круг знакомств задается матрицей *A* размера *n* на *n*. Элемент матрицы $A[i, j] = 1$, если *i*-ый студент знаком с *j*-ым, и $A[i, j] = 0$ в противном случае. *Необходимо* определить, можно ли произвольно разбить *n* студентов на 2 команды, численность которых отличается не более чем в 2 раза, если известно, что в любой команде должны быть студенты, которые обязательно знакомы друг с другом

8. Разбиение на группы не знакомых. Имеется *n* человек и матрица *A* размера *n* на *n*. Элемент матрицы $A[i, j] = 1$, если *i*-ый человек знает *j*-го, и $A[i, j] = 0$ в противном случае (если *i*-ый человек знает *j*-го, то считаем, что и *j*-ый человек знает *i*-го). *Необходимо* определить, можно ли разбить людей на 2 группы таким образом, чтобы в каждой группе были только незнакомые люди.

9. Олимпиада. На олимпиаду прибыло *n* человек. Некоторые из них знакомы между собой. Круг знакомств задается матрицей *A* размера *n* на *n*. Элемент матрицы $A[i, j] = 1$, если *i*-ый человек знает *j*-го, и $A[i, j] = 0$, в противном случае (если *i*-ый человек знает *j*-го, то считаем, что и *j*-ый человек знает *i*-го). *Необходимо* определить, можно ли опосредованно перезнакомить всех людей между собой (незнакомые люди могут познакомиться только через общего знакомого)? Если "нет", то какое максимальное количество людей будут знать друг с друга?

10. Книга. Пусть группа состоит из *n* человек. В ней каждый имеет $n/2$ друзей и не более *k* врагов. У одного человека в группе есть книга, которую все хотели бы прочитать и потом обсудить с некоторыми из остальных.
Необходимо определить способ передачи книги таким образом, чтобы она побывала у каждого в точности один раз, переходя от друга к другу, и возвратилась к своему владельцу;

11. Станки. Задано *n* различных станков, которые один за другим связаны в конвейер. Имеется *n* рабочих. Задана матрица *C* размера *n* на *n*, где элемент матрицы $C[i, j]$ задает производительность *i*-го рабочего на *j*-ом станке (если рабочий *i* не может работать на *j*-ом станке, то значение элемента матрицы $C[i, j]$ равно 0). *Необходимо* определить, каким должно быть распределение рабочих по станкам (каждый рабочий может быть назначен только на один станок и на каждом станке может работать только один рабочий), чтобы производительность конвейера была максимальной.

12. Встреча. Вводятся два числа: *n* – количество домиков и *k* - количество дорог. Домики пронумерованы целыми числами

от 1 до *n*. Каждая дорога определяется тройкой чисел: двумя номерами домиков, которые являются концами этой дороги, и длиной дороги (длины дорог - положительные целые числа). В каждом домике живет по одному человеку. *Необходимо* найти точку (место встречи всех людей), от которой суммарное расстояние до всех домиков будет минимальным. Если точка лежит на дороге, то указать номера домиков, которые являются концами этой дороги, и расстояние от первого из этих домиков. Если точка совпадает с домиком, то указать его номер.

13. Янка. Янка положил на стол *n* выпуклых *k*-гранников и *n* различных типов наклеек по *k* штук каждой. Ночью кто-то наклеил наклейки на грани, по одной на грань (на одном и том же многограннике могло оказаться несколько наклеек одного типа). *Необходимо* расставить многогранники так, чтобы наклейка каждого типа была видна ровно $k-1$ раз. Ответ задачи выдать в виде строки. В строке *i*-ый элемент равен номеру наклейки на *i*-ом многограннике, на которую его нужно поставить.

14. Второй по длине путь. Задано *n* городов, которые занумерованы целыми числами от 1 до *n*, и сеть из *m* дорог с односторонним движением между ними. Каждая дорога задается тройкой (i, j, k), где *i* - номер города, в котором дорога начинается, *j* - номер города, в котором дорога заканчивается, а *k* - ее длина (число *k* - натуральное). Дороги друг с другом могут пересекаться только в городах. Все пути между двумя указанными городами *A* и *B* можно упорядочить в список по не убыванию их длин. *Необходимо* найти путь, который может быть вторым по длине в упорядоченном списке. Вывести сам путь и его длину.

15. Четно-нечетная магистраль. Предположим, что есть страна с n городами. Дана система магистралей, соединяющая напрямую города между собой. Длина любого прямого соединения равна 1. Система магистралей называется "четно-нечетной", если любые два города, соединены путем, как четной длины, так и нечетной длины. *Необходимо* определить является ли система магистралей "четно-нечетной". Если ответ на вопрос отрицательный, то найти одно из подмножеств множества городов, в котором максимальное количество элементов, и которое удовлетво-ряет следующему условию: если какие-либо два города из этого подмножества соединены путем, то его длина четна.

16. Пересекающиеся дороги. Сеть дорог определяется следующим образом. Имеется n перекрестков, (занумерованных целыми числами от 1 до n) и k дорог, связывающих перекрестки. Каждая дорога определяется тройкой чисел: двумя номерами перекрестков и временем, требующимся на проезд по этой дороге. Движение по дороге возможно в обоих направлениях. *Необходимо* найти кратчайший по времени маршрут машины от перекрестка с номером i до перекрестка с номером j, если на перекрестке машина должна ждать время, равное числу пересекающихся в этом перекрестке дорог (для преодоления начального (i) и конечного (j) перекрестков машина также должна ждать).

17. Домики. На прямоугольном участке размера 100×100 расположено n домиков. Левый нижний угол участка имеет координаты $(0,0)$, а правый верхний - $(100,100)$. Место-положение каждого домика задается целочисленными координатами его нижнего левого угла. Каждый домик имеет размер 5×5. Стороны домиков параллельны сторонам участка. Домики отстоят друг от друга не меньше, чем на 1 координату.

Необходимо найти один из кратчайших по длине путей от точки $(0,0)$ до точки $(100,100)$. Найденный путь представить в виде координат концов прямолинейных отрезков, составляющих этот путь. Вывести длину найденного пути с точностью до 0,1.

18. Хакеры. Компьютерная сеть состоит из связанных между собой больших ЭВМ, к каждой из которых подключается несколько терминалов (в сети ни у каких двух ЭВМ количество терминалов не совпадает). Подключение к одной из больших ЭВМ, позволяет получить информацию, содержащуюся в памяти этой ЭВМ, а также всю информацию доступную для ЭВМ, к которым данная ЭВМ могла направлять запросы. Необходимо защитить компьютерную сеть от хакеров, которые выкачивают из компьютеров секретную информацию. Хакеры и раньше нападали на подобные компьютерные сети, и их тактика была известна. Тактика хакеров: при нападении хакеры всегда получают доступ к информации всех ЭВМ сети. Добиваются они этого, захватывая некоторые ЭВМ сети так, чтобы от них можно было запросить информацию, которая имеется у оставшихся ЭВМ. Существует много способов захвата, например, захватить все ЭВМ. Однако хакеры всегда выбирают такой вариант, при котором суммарное количество терминалов у захваченных ЭВМ минимально. *Необходимо* определить список номеров ЭВМ, которые могут быть выбраны хакерами для захвата сети согласно их тактике.

19. Пирамида Хеопса. Внутри пирамиды Хеопса есть n комнат, в которых установлены $2m$ модулей, составляющих m устройств. Каждое устройство состоит из двух модулей, которые располагаются в разных комнатах, и предназначены для перемещения между парой комнат, в которых они установлены. Перемещение происходит за 0,5 единиц времени.

В начальный момент времени модули всех устройств переходят в подготовительный режим. Каждый из модулей имеет некоторый свой целочисленный период времени, в течение которого он находится в подготовительном режиме. По истечении этого времени модуль мгновенно включается, после чего опять переходит в подготовительный режим. Устройством можно воспользоваться только в тот момент, когда одновременно включаются оба его модуля. Преступник сумел проникнуть в гробницу фараона. Обследовав ее, он включил устройства и собрался уходить, но в этот момент проснулся хранитель гробницы. Теперь преступнику необходимо как можно быстрее попасть из комнаты номер 1 в комнату с номером *n*, в которой находится выход из пирамиды. При этом из комнаты в комнату он может попадать только при помощи устройств, так как проснувшийся хранитель закрыл все двери в комнатах пирамиды. *Необходимо* написать программу, которая получает на входе описание расположения устройств и их характеристик, а выдает значение оптимального времени и последовательность устройств, которыми надо воспользоваться, чтобы попасть из комнаты номер 1 в комнату номер *n* за это время.

20. Без левых поворотов. План города представляет собой множество перекрестков, которые соединены дорогами. Перекрестки обозначаются точками на плоскости с координатами $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, а дорогам соответствуют отрезки (i, j), где *i* и *j* обозначают номера перекрестков, которые эта дорога соединяет. *Необходимо* определить, можно ли проехать от перекрестка с номером *s* до перекрестка с номером *f*, не делая левых поворотов, т. е. двигаясь только прямо или вправо. Если да, то указать последовательность перекрестков. Первоначально мы будем предполагать, что

если у перекрестка *s* координаты (x_s, y_s), то машина подъехала к этому перекрестку из точки с координатами $(x_s, y_s - 1)$.

21. Кратчайший путь с дополнительным временем преодоления перекрестка. План города представляет собой множество из *n* перекрестков (занумерованных целыми числами от 1 до *n*), соединенных *m* дорогами. Каждая дорога задается номерами перекрестков, которые она соединяет, и временем движения по ней. Кроме того, время преодоления перекрестка с номером *i* равно $q \cdot k_i$, где *q* - заданная константа, а k_i - количество дорог, инцидентных перекрестку *i*. *Необходимо* найти кратчайший по времени маршрут от перекрестка с номером *s* до перекрестка с номером *f*. Вершины *s* и *f* преодолеваются аналогично преодолению перекрестков. Если пути не существует, то выдать соответствующее сообщение.

22. Железные и шоссейные дороги. Имеется *n* городов, которые пронумерованы целыми числами от 1 до *n*. Некоторые из городов соединены двухсторонними дорогами, которые пересекаются только в городах. Имеется два типа дорог - шоссейные и железные ('0' – шоссейная дорога, '1' – железная дорога). Для каждой дороги известна базовая стоимость проезда по ней. Стоимость проезда зависит от набора проезжаемых дорог и от способа проезда. Так, если вы подъехали к городу *C* по шоссейной (железной) дороге $X \to C$ и хотите ехать дальше по дороге $C \to Y$ того же типа, то вы должны уплатить только базовую стоимость проезда по дороге $C \to Y$. Если тип дороги $C \to Y$ отличен от типа дороги $X \to C$, то вы должны уплатить базовую стоимость проезда по дороге $C \to Y$ плюс 10% от базовой стоимости проезда по

этой дороге (страховой взнос). *Необходимо* найти самый дешевый маршрут проезда из города A в город B в виде последовательности городов и вычислить стоимость проезда по этому маршруту. Если пути не существует, то выдать соответствующее сообщение. При выезде из города A страховой взнос платится всегда.

23. Кратчайший путь в зависимости от направления поворота. Заданы декартовы координаты n перекрестков города, которые пронумерованы целыми числами от 1 до n. На каждом перекрестке имеется светофор. Некоторые из перекрестков соединены дорогами с двухсторонним (правосторонним) движением, которые пересекаются только на перекрестках. Для каждой из m дорог известно время, которое требуется для проезда по ней от одного перекрестка до другого. Время проезда зависит от набора проезжаемых дорог и от времени ожидания на перекрестках. Так, если вы подъехали от перекрестка X к перекрестку C по дороге $X \to C$ и хотите ехать дальше по дороге $C \to Y$, то время ожидания на перекрестке C зависит от того, поворачиваете вы налево или нет. Если вы поворачиваете налево, то время ожидания равно $d \cdot k$, где число d - количество дорог, пересекающихся на перекрестке C, а число k – некоторая константа (одна и та же для всех перекрестков). Если вы не поворачиваете налево, то время ожидания на перекрестке равно нулю. Первоначально мы будем предполагать, что если у перекрестка A координаты (x_A, y_A), то машина подъехала к этому перекрестку из точки с координатами $(x_A, y_A - 1)$. *Необходимо* проехать от перекрестка с номером A до перекрестка с номером B за минимальное время.

24. Уличная гонка. План улицы для гонки задан в виде точек, которые помечены числами от 0 до n и стрелок, которые соединяют их. Точка 0 является стартовой, а точка n – финишной. Стрелками представлены улицы с односто-ронним движением. Участники гонки передвигаются от точки к точке по улицам только в направлении стрелок. В каждой точке участник гонки может выбрать любую из исходящих стрелок. Назовем "план улицы хорошим", если он обладает следующими свойствами:

1) каждая точка плана может быть достигнута со старта;

2) финиш может быть достигнут из любой точки плана;

3) у финиша нет исходящих стрелок.

Для достижения финиша участник не обязан пройти через все точки. Однако некоторые точки невозможно обойти. Назовем их "неизбежными". Предположим, что гонка проводится за два последовательных дня. Для этой цели план должен быть разбит на два "хороших плана", по одному на каждый день. В первый день стартом является точка 0, а финишем служит некоторая "точка разбиения". Во второй день старт находится в этой "точке разбиения", а финиш находится в точке n. Точка s является "точкой разбиения" для "хорошего плана" C, если:

1) s отличается от старта и финиша плана C;

2) план разбивается ею на два "хороших плана" без общих стрелок (т. е. между планами нет стрелок их соединяющих) и с единственной общей точкой s.

Необходимо для заданного "хорошего плана"

1) Определить множество «неизбежных точек», которые должны посетить все участники гонки (за исключением старта и финиша).

2) Определить множество всех возможных "точек разбиения".

25. Прогулка. Хозяин вышел на прогулку с собакой. Известно, что путь хозяина представляет собой ломаную линию, координаты отрезков ломаной заданы: (x_i, y_i), $i = 1, \ldots, n$. Точка (x_1, y_1) – начальная точка ломаной линии, а точка (x_n, y_n) – конечная точка ломаной. Хозяин двигается во время прогулки от стартовой точки, далее по отрезкам ломаной линии, и заканчивает путь в конечной точке ломаной. У собаки есть свои любимые места с координатами $(xdog_j, ydog_j)$, $j = 1, \ldots, m$, которые собака хотела бы посетить. В то время, пока хозяин проходит один отрезок ломаной, собака может посетить только одно из своих любимых мест. В начальной и конечной координате каждого отрезка ломаной собака обязана подбежать к хозяину. Известно, что скорость собаки в два раза выше скорости хозяина. *Необходимо* определить, какое наибольшее количество своих любимых мест, и в какой последовательности сможет посетить собака за время прогулки.

26. Сборка прибора. Прибор спроектирован таким образом, что он будет собираться из отдельных узлов, причем каждый узел уникален. Сами узлы в свою очередь могут требовать предварительной сборки. Для сборки каждого узла необходимо, чтобы все узлы, комплектующие его, были уже собраны. Узлы, не требующие сборки, обязательно тестируются на работоспособность. Собираемые узлы тестировать не требуется. На сборку одного узла или на его тестирование тратится один день. Готовый узел должен быть помещен на склад и может быть взят со склада только тогда, когда он необходим для сборки очередного узла или самого прибора. Хранение узла на складе в течение одного дня требует оплаты в размере одной условной денежной единицы.

Необходимо организовать сборку таким образом, чтобы плата за аренду склада была минимальной.

27. Скрудж Мак-Дак. Скрудж Мак-Дак решил сделать прибор для управления самолетом. Как известно, положение штурвала зависит от состояния входных датчиков, но эта функция довольно сложна. Его механик устройство, вычисляющее эту функцию за несколько этапов с использованием промежуточной памяти и вспомогательных функций. Для вычисления каждой из функций требуется, чтобы в ячейках памяти уже находились вычисленные параметры (которые являются значениями вычисленных функций), необходимые для ее вычисления. Вычисление функции без параметров может производиться в любое время. После вычисления функции ячейки могут быть использованы повторно (хотя бы для записи результата вычисленной функции). Структура вызова функций такова, что каждая функция вычисляется не более одного раза, и любой параметр используется не более одного раза. Любой параметр есть имя функции. Так как Скрудж не хочет тратить лишних денег на микросхемы, он поставил задачу минимизировать память прибора. По заданной структуре вызовов функций *необходимо* определить минимально возможный размер памяти прибора и указать последовательность вычисления функций.

28. Заправочные станции. Имеется n городов, соединенных двухсторонними дорогами с правосторонним движением. Для каждой дороги заданы города, которые она соединяет, и ее протяженность в километрах. Машина может поворачивать (изменять направление движения) только в городах. Машина имеет бак вместимостью z литров и для нее задан расход бензина (x литров) на один километр. В некоторых городах имеются заправочные станции (всего не

более 8). Для каждой заправочной станции определена своя цена за один литр бензина. Машина сможет заправиться только в том случае, если ее бак заполнен менее чем на половину. Если машина заправляется, то она должна заполнить бак полностью. Слив бензина запрещен. *Необходимо* определить самый дешевый маршрут из города *A* в город *B*, если первоначально бак машины заполнен полностью.

29. Слова. Задана последовательность слов. Игра заключается в том, что игроки по очереди называют слова из заданной последовательности. Задано правило, по которому называются слова. Если названо некоторое слово, то игрок может назвать не названное ранее слово, начинающееся с буквы, на которую заканчивалось предыдущее слово. *Необходимо* определить, можно ли выстроить цепочку из всех слов, причем последнее слово должно заканчиваться на ту букву, с которой начиналось первое слово.

30. Цепочка из доминошек. На столе выложены в ряд доминошки. Каждая доминошка состоит из двух частей. На каждой из частей доминошек изображено некоторое число точек. Доминошки могут выкладываться в ряд одна за другой по правилу: количества точек на примыкающих частях соседних в ряду доминошек должны совпадать. *Необходимо* определить, можно ли выстроить цепочку, в которой каждая доминошка встречается ровно один раз, причем количества точек на свободных концах доминошек, которые являются крайними в цепочке, должны совпадать.

31. Почтальон. Задан район, состоящий из n населенных пунктов, которые занумерованы целыми числами от 1 до n. Районы соединены между собой m дорогами. Известна протяженность в километрах каждой дороги. Вдоль каждой

дороги располагаются почтовые ящики. В районе есть один почтальон, который осуществляет доставку почты. *Необходимо* так спланировать маршрут доставки почты, при котором почтальон должен пройти по каждой дороге на менее одного раза, вернуться в начальную точку, а длина маршрута при этом должна быть минимальной.

32. Отрезки. Пусть на плоскости с Евклидовой метрикой задано n красных и n синих точек (красные точки имеют номера от 1 до n, а синие – от $n+1$ до $2n$). Имеется n отрезков, соединяющих эти точки таким образом, что каждый отрезок соединяет точки различного цвета и каждая точка является концевой точкой точно одного отрезка. *Необходимо* определить, является ли заданное соединение минимальным по длине (сумма длин всех отрезков) среди всех возможных соединений, удовлетворяющих заданным свойствам.

33. Трубопроводы. Имеется n пунктов, которые занумерованы целыми числами от 1 до n, и сеть трубопроводов по перекачке нефти из пункта A в пункт B. Для каждого участка трубопровода указаны пункты, которые он соединяет, пропускная способность (тонна/час) и стоимость транспортировки (транзита) тонны нефти по участку. *Необходимо* организовать перекачку максимального количества нефти при минимальных суммарных затратах на транзит из пункта A в пункт B.

34. Таксист. Имеется n городов, занумерованных целыми числами от 1 до n, которые связаны m дорогами. Движение по дорогам осуществляется только в одном направлении и дороги пересекаются только в городах. Для каждой дороги известны начальный и конечный города, которые она соединяет, а также ее длина. *Необходимо* найти все дороги, по которым

потенциально может проехать таксист в маршруте из 1-го города *n*-ый город таким образом, чтобы длина его маршрута отличалась не более, чем на величину *k* от длины минимального маршрута из 1-го в *n*-ый.

35. Стрельба. На соревнованиях по стрельбе каждый участник будет стрелять в цель, которая представляет собой прямоугольник, разделенный на квадраты. Цель содержит r·c квадратов, расположенных в *r* строках и *q* столбцах. Квадраты выкрашены в белый или черный цвет. В каждом столбце находится ровно 2 белых и $(r-2)$ черных квадрата. Строки пронумерованы от 1 до *r* сверху вниз, а столбцы от 1 до *q* слева направо. Стрелок имеет ровно *q* стрел. Последовательность из *q* выстрелов называется корректной**,** если в каждом столбце поражен ровно один белый квадрат, а в каждой строке – не менее одного белого квадрата. *Необходимо* определить, существует ли корректная последовательность выстрелов, и если да, то найти одну из них.

ЛИТЕРАТУРА

1. *Ахо А. В.*, *Хопкрофт Д. Э.*, *Ульман Д. Д.* Структуры данных и алгоритмы: Учеб. пособие/ Пер. с англ. М.: Изд. Дом "Вильямс", 2000. 384 с.

2. *Ахо А. В.*, *Хопкрофт Д. Э.*, *Ульман Д. Д.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979. 536 с.

3. *Емеличев В. А.*, *Мельников О. И.*, *Сарванов В. И.*, *Тышкевич Р. И.* Лекции по теории графов. М.: Наука, 1990. 383 с.

4. *Кормен Т.*, *Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 1999. 960 с.

5. *Котов В. М., Волков И. А., Лапо А. И.* Методы алгоритмизации: Учеб. Пособие для 8-9 классов. Мн.: Нар. Асвета, 2000. 300 с.

6. *Котов В. М., Мельников О. И.* Информатика. Методы алгорит-мизации: Учеб. Пособие для 10-11 классов. Мн.: Нар. Асвета, 2000. 221 с.

7. *Кристофидес Н.* Теория графов. М.: Мир, 1978. 432 с.

8. *Липский В.* Комбинаторика для программистов. М.: Мир, 1988. 214 с.

9. *Пападимитриу Х.*, *Стайглиц К.* Комбинаторная оптимизация: Алгоритмы и сложность. М.: Мир, 1971. 512 с.