Преподаватели:
к.ф.-м.н., заведующий кафедрой ВТиКГ ДВГУПС,
преподаватель программы IT-школа Samsung,
Пономарчук Юлия Викторовна
E-mail: yulia.ponomarchuk@gmail.com

Понятие жадного алгоритма

Задачи оптимизации, как правило, решаются алгоритмами:

- представляющими последовательность шагов, на каждом из которых возможен выбор из нескольких вариантов действий
- определение наилучшего варианта методами динамического программирования не всегда возможно

В **жадном алгоритме** всегда делается выбор, который кажется *самым лучшим в данный момент*

- Т.е. выбирается *локально оптимальный вариант* в надежде, что он приведет к оптимальному решению глобальной задачи
- Жадные алгоритмы *не всегда* приводят к оптимальному решению

В каждой точке принятия решения делается выбор, который в данный момент выглядит самым лучшим

- Эвристическое решение не всегда дает оптимальное решение

Процесс разработки жадных алгоритмов

1. Привести задачу оптимизации к виду, когда после сделанного выбора остается решить только одну подзадачу

2. Доказать, что всегда существует такое оптимальное решение исходной задачи, которое можно получить путем жадного выбора, так что такой выбор всегда допустим

3. Показать, что после жадного выбора остается подзадача, обладающая тем свойством, что объединение оптимального решения подзадачи со сделанным жадным выбором приводит к оптимальному решению исходной задачи

Стратегия жадного выбора

Глобальное оптимальное решение можно получить, делая локально оптимальный (жадный) выбор.

- Выбор, сделанный в жадном алгоритме, может зависеть от сделанных ранее выборов, но он *не может зависеть от* каких бы то ни было выборов или *решений последующих подзадач*
- Необходимо доказать, что *жадный выбор* на каждом этапе *приводит к глобальному оптимальному решению*
  - Обычно исследуется глобальное оптимальное решение некоторой подзадачи
  - Затем демонстрируется, что решение можно преобразовать так, чтобы в нем использовался жадный выбор, в результате чего, получится аналогичная, но более простая подзадача
- Часто благодаря предварительной обработке входных данных или применению подходящей структуры данных можно ускорить процесс жадного выбора

Петя разгадывает головоломку, которая устроена следующим образом. Дана квадратная таблица размера *N*x*N*, в каждой клетке которой записана какая-нибудь латинская буква. Кроме того, дан список ключевых слов. Пете нужно, взяв очередное ключевое слово, найти его в таблице. То есть найти в таблице все буквы этого слова, причем они должны быть расположены так, чтобы клетка, в которой расположена каждая последующая буква слова, была соседней с клеткой, в которой записана предыдущая буква (клетки называются соседними, если они имеют общую сторону — то есть соседствуют по вертикали или по горизонтали). Например, на рисунке ниже показано, как может быть расположено в таблице слово olympiad.

P	**O**	**L**	T	E
R	W	**Y**	**M**	S
O	**A**	**I**	**P**	T
B	**D**	A	N	R
L	E	M	E	S

Когда Петя находит слово, он вычеркивает его из таблицы. Использовать уже вычеркнутые буквы в других ключевых словах нельзя. *После того, как найдены и вычеркнуты все ключевые слова, в таблице остаются еще несколько букв, из которых Петя должен составить слово, зашифрованное в головоломке.*

Помогите Пете в решении этой головоломки, написав программу, которая по данной таблице и списку ключевых слов выпишет, из каких букв Петя должен сложить слово, то есть какие буквы останутся в таблице после вычеркивания ключевых слов.

Входные данные

В первой строке входного файла записаны два числа N ($1 \le N \le 10$) и M ($0 \le M \le 200$). Следующие N строк по N заглавных латинских букв описывают ребус. Следующие M строк содержат слова. Слова состоят только из заглавных латинских букв, каждое слово не длиннее 200 символов. *Гарантируется, что в таблице можно найти и вычеркнуть по описанным выше правилам все ключевые слова.*

Выходные данные

В единственную строку выходного файла выведите в любом порядке буквы, которые останутся в таблице.

Примеры

№	INPUT.TXT	OUTPUT.TXT
1	5 3 POLTE RWYMS OAIPT BDANR LEMES OLYMPIAD PROBLEM TEST	AENRSW
2	3 2 ISQ ABC IQW I IS	ABCQQW

Если посмотреть в условие внимательно, то можно заметить, что нам уже дано, что головоломка решаема.

То есть, самый простой метод решения

- сохранить все введенные символы, для каждого запоминая, сколько раз он встречается в тексте,
- затем для каждого символа уменьшить соответствующее ему число на 1 за каждый раз, когда этот символ встречается в "отгадках".
- выводим символ столько раз, сколько он у нас остался.

Задача 12.1. Перед праздниками Шеф получает очень много приглашений на торжественные заседания. Чтобы лучше планировать свое время, Шеф ввел правило, чтобы в каждом i-м приглашении был четко указан отрезок времени заседания $[a_i; b_i]$. Шеф не любит половинчатых решений, поэтому или находится на заседании все указанное время, или не приходит на него. Между посещениями заседаний должен быть хотя бы минимальный перерыв, т.е. Шеф может успеть на j-е (по списку приглашений) после i-го, если $a_j > b_i$.

Напишите программу, помогающую Шефу посетить как можно больше заседаний. Если различные расписания позволяют посетить одинаковое максимальное количество заседаний, найти любое из них.

Вход. В первой строке записано количество заседаний N, где $2 \le N \le 5000$, в следующих N строках — по два целых числа a_i и b_i $(0 \le a_i < b_i < 10^9)$.

Выход. Строка с N символами `0` и `1`, обозначающими, согласен ли Шеф приехать на i-е (в порядке входных данных) заседание.

Пример

Вход	`5`
	`2 17`
	`26 50`
	`17 20`
	`10 15`
	`20 25`
		Выход	`11001`

- На первом шаге выбрать заседание с наименьшим значением b
- На каждом следующем шаге – с наименьшим значением b, но только среди тех заседаний, которые начинаются после конца предыдущего выбранного:

$$i_1 = \arg\min_j \{b_j\}, \; i_k = \arg\min_{j: a_j > b_{i_{k-1}}} \{b_j\} \text{ при } k \geq 2.$$

- Доказательство оптимальности выбора в соответствии с основной идеей:
  - Требование, что любое непервое посещаемое заседание нужно выбирать среди тех j, для которых $a_j > b_{i_k}$, заложено в условии
  - Предположим, что существует оптимальное расписание посещения заседаний, где в качестве некоторого k-го (k>=1) по порядку посещения заседания выбрано не заседание с наименьшим из допустимых b.
  - Тогда ничто не мешает построить расписание, в котором все пункты, кроме k-го, совпадают, а в качестве k-го выбрано заседание с наименьшим из допустимых b

- Итак, никакое расписание посещения заседаний не может быть лучше расписания, построенного по указанному выше алгоритму
- Это не значит, что любое отступление от алгоритма приведет к неоптимальному решению. Алгоритм дает *одно из лучших решений*

Задача о расписании

- На первом шаге выбрать заседание с наименьшим значением b
- На каждом следующем шаге – с наименьшим значением b, но только среди тех заседаний, которые начинаются после конца предыдущего выбранного:
- Доказательство оптимальности выбора в соответствии с основной идеей:
  - Требование, что любое непервое посещаемое заседание нужно выбирать среди тех j, для которых $a_j > b_{i_k}$, заложено в условии
  - Предположим, что существует оптимальное расписание посещения заседаний, где в качестве некоторого k-го (k>=1) по порядку посещения заседания выбрано не заседание с наименьшим из допустимых b.
  - Тогда ничто не мешает построить расписание, в котором все пункты, кроме k-го, совпадают, а в качестве k-го выбрано заседание с наименьшим из допустимых b

```
const MAXN = 5000;
type Inv = record
             a, b : longint; idx : word
           end;
  aInv = array [1..MAXN] of Inv;
  aAns = array [1..MAXN] of byte;

... { процедура эффективной сортировки массива записей }

var invit : aInv; { интервалы приглашений }
    i, N : word;
    res : aAns;  { массив для ответа }
    b_prev : longint; { момент b последнего выбранного заседания }
BEGIN
  read(N);
  for i := 1 to N do begin { заполняем массив интервалов
приглашений: }
    read(invit[i].a, invit[i].b);  { интервал приглашения }
    invit[i].idx := i; { его начальный номер }
  end;

  ...{ вызов процедуры эффективной сортировки,
       упорядочивающей записи массива по возрастанию b }

  fillchar(res, N, #0);  { массив ответа инициализируем нулями }
  b_prev := -1;  { -1 меньше любого допустимого a; такая
    инициализация позволяет не выделять выбор первого
    приглашения }
  for i := 1 to N do begin
    { первый интервал, не пересекающийся с предыдущим,
      принимается }
    if invit[i].a > b_prev then begin
      res[invit[i].idx] := 1; { указываем его в ответе и }
      b_prev := invit[i].b;   { изменяем последнее b }
    end;
  end;
  for i := 1 to N do { вывод ответа }
    write(res[i]);
END.
```