Кротков Павел Андреевич,
Поромов Сергей Сергеевич,
Ульянцев Владимир Игоревич

ЗАДАЧА «ДЕЛОВЫЕ ВСТРЕЧИ»

Этой статьей мы продолжаем цикл публикаций олимпиадных задач по информатике для школьников. Решение таких задач и изучение разборов поможет Вам повысить уровень практических навыков программирования и подготовиться к олимпиадам по информатике.

В этой статье рассматривается задача «Деловые встречи», которая предлагалась на Шестой командной интернет-олимпиаде по программированию в 2011–2012 учебном году. Материалы этой олимпиады можно найти на сайте http://neerc.ifmo.ru/school/io/.

УСЛОВИЕ ЗАДАЧИ

Алексей – успешный предприниматель, и в течение одного дня у него бывает много встреч с разными деловыми партнерами. К сожалению, встречи бывают разные, и не все из них приносят ему радость. Предпринимательская практика показывает, что на многие встречи не стоит приходить в слишком плохом или хорошем настроении – результат таких встреч может оказаться не таким, какой хочется Алексею.

К счастью, недавно Алексей научился оценивать свое настроение с помощью целых чисел. После этого для каждой встречи он оценил, при каком максимальном и минимальном значении настроения стоит на нее приходить, а также то, как изменится его настроение после этой встречи. Теперь он хочет распланировать порядок встреч так, чтобы в течение дня совершить максимальное число встреч.

Ваша задача – написать программу, которая по информации обо всех встречах и настроении Алексея в начале дня находит такой порядок проведения встреч, что их количество при этом максимально.

Формат входного файла

Первая строка входного файла содержит два целых числа n и k ($1 \le n \le 20$, $-100 \le k \le 100$) – количество встреч и настроение Алексея в начале дня, соответственно.

Следующие n строк содержат по три целых числа a_i, b_i и c_i ($-100 \le a_i, b_i, c_i \le 100$) – минимальное и максимальное настроение, при котором встреча возможна, а также величина, на которую изменяется настроение по окончании встречи, соответственно.

Формат выходного файла

В первой строке выходного файла выведите число m – максимальное возможное число встреч, которые Алексей сможет провести. В следующей строке выведите через пробел m целых чисел – номера встреч в порядке их проведения. Встречи пронумерованы в порядке описания во входном файле, начиная с единицы.

Если ответов с максимальным числом встреч несколько, выведите любой из них.

Примеры входных и выходных данных

meetings.in	meetings.out
3 0 1 3 3 0 1 2 1 3 1	3 2 3 1
3 1 -10 -5 3 -5 5 -2 -3 2 1	2 3 2

РАЗБОР ЗАДАЧИ

Данная задача является классическим примером задачи, которую можно решить с помощью динамического программирования по подмножествам. Этот метод описан в различной литературе (например, в [1]) и на образовательных интернет-порталах, посвященных олимпиадному программированию [2]. Приведем описание применения этого метода при решении данной задачи.

Во время написания решения нам неоднократно придется работать с таким понятием, как подмножество встреч из описанного в тесте набора. Так, нам понадобятся следующие операции:

– добавить встречу в подмножество;

– удалить встречу из подмножества;

– узнать количество встреч в подмножестве;

– узнать, содержит ли подмножество встречу с номером t.

Во многих языках программирования предусмотрены структуры данных, которые поддерживают все описанные операции и накладывают некоторые ограничения на работу с ними. Однако в данной задаче различных объектов, которые будут находиться в подмножествах, очень мало (не более 20). Стандартным приемом, используемым для работы с множествами из такого количества элементов, является представление множеств в виде *битовых масок*.

Как известно, целые числа в памяти компьютера представляются с помощью их двоичной записи. Например, одно число типа **longint** в языке программирования Pascal представляется с помощью 32 бит, один из которых отвечает за знак числа, а все остальные – за его модуль. Не будем вдаваться в подробности представления чисел в памяти, скажем только, что при хранении положительных чисел нулевой бит отвечает за последний разряд в двоичной записи числа, первый бит – за предпоследний и т. д.

Кроме этого, ко всем целочисленным типам в большинстве языков программирования применимы логические операции, такие как конъюнкция (**and**), дизъюнкция (**or**), исключающая дизъюнкция (**xor**). Применение данных операций к числам сводится к применению этих операций к соответствующим битам чисел по отдельности, после чего получившиеся биты записываются в число, являющееся результатом выполнения операции. Кроме побитовых логических операций, мы будем использовать операцию **shl**, при выполнении которой вся битовая запись числа сдвигается влево на некоторое количество бит, а освободившиеся биты в конце числа заполняются нулями. Более подробное описание работы с двоичными представлениями чисел и битовыми операциями можно найти в [3].

Будем хранить подмножество встреч в виде числа **mask** типа **longint**. Если встреча с номером **t** содержится в подмножестве, то бит **t-1** числа **mask** равен единице. Остальные биты числа **mask** должны быть равны нулю. Теперь мы мо-

жем реализовать все необходимые нам операции работы с множествами (листинг 1).

После реализации всех требуемых действий, обрабатывающих множества, можно приступить к решению самой задачи. Для этого нам потребуется сделать одно важное наблюдение. Заключается оно в том, что то, как влияет некоторый набор встреч на настроение Алексея, никак не зависит от порядка, в котором эти встречи были выполнены – в каком бы порядке Алексей ни провел эти встречи, изменение настроения будет равно сумме изменений настроения после всех проведенных им встреч. Посчитаем эти изменения для всех возможных наборов встреч, соответствующий код приведен в листинге 2. Заметим, что при количестве встреч, не превышающем 20, количество возможных наборов из этих встреч не превысит 2^{20}, а значит, мы можем себе позволить хранить все посчитанные значения в памяти компьютера.

В листинге 2 массив **change** содержит информацию, которую мы считали из входного файла – для встречи с номером **i** значение **change[i]** равно влиянию этой встречи на настроение, а значения массива **maskChange** равны суммарным влияниям различных подмножеств встреч **mask**.

Далее необходимо для каждого набора встреч выяснить, может ли Алексей провести именно эти встречи в каком-нибудь порядке. Делать это мы будем следующим образом. В массиве **possible** для каждого набора будем хранить логическую переменную, равную **true**, если Алексей может провести эти встречи. Заметим, что до начала подсчета этих значений, все ячейки этого массива, кроме нулевой, будут равны **false**, поскольку вообще ни с кем не встречаться Алексей точно может.

После инициализации массива переберем все возможные подмножества. В каждое множество, которое Алексей может

Листинг 1. Функции для работы с множествами

```
// Проверка того, есть ли встреча номер t в множестве mask
function contain(mask, t : longint) : boolean;
begin
  result := (mask and (1 shl (t - 1))) > 0;
end;

// Добавление встречи номер t в множество mask
function add(mask, t : longint) : longint;
begin
  result := mask or (1 shl (t - 1));
end;

// Удаление встречи номер t из множества mask
function remove(mask, t : longint) : longint;
begin
  result := mask xor (1 shl (t - 1));
end;

// Проверка количества встреч в множестве mask
function count(mask: longint) : longint;
var
  ans : longint;
begin
  ans := 0;
  while (mask > 0) do begin
    ans := ans + (mask and 1);
    mask := mask shr 1;
  end;
  result := ans;
end;
```

Листинг 2. Подсчет влияния различных наборов встреч на настроение Алексея

```
maskCount := (1 shl n) - 1;
for mask := 0 to maskCount do begin
  for i := 1 to n do begin
    if (contain(mask, i)) then begin
      maskChange[mask] := maskChange[mask] + change[i];
    end;
  end;
end;
```

выполнить, попробуем добавить одну встречу, которой в нем еще нет, и, если настроение Алексея позволяет выполнить ему эту встречу после выполнения этого множества, запишем **true** в соответствующую ячейку массива **possible**.

Заметим, что для корректной работы описанного алгоритма нам необходимо перебирать множества в таком порядке, чтобы при рассмотрении очередного множества все его подмножества были уже рассмотрены. Несложно понять, что в данном случае это требование гарантируется простым перебором множеств в порядке соответствующих чисел **mask**, однако порядок перебора множеств может отличаться в других похожих задачах.

В приведенном листинге 3 в массивах **min** и **max** хранятся минимальное и максимальное возможное настроение Алексея перед проведением соответствующей встречи. Эти данные также необходимо считать из входного файла перед решением задачи.

После подсчета данных значений несложно понять, как найти наибольшее подмножество встреч, которое Алексей может провести. Однако в задаче от нас требуется не только это – необходимо также восстановить и порядок, в котором Алексей будет проводить эти встречи. Для этого нам придется несколько модифицировать приведенный в листинге 3 код и для каждого допустимого множества хранить в массиве **last** встречу, которая в этот набор была добавлена последней.

Вторым изменением приведенного в листинге 3 кода станет поиск допустимого множества с максимальным количеством встреч, который мы будем выполнять вместе с подсчетом значений массивов **possible** и **last**. Хранить множество, которое станет ответом, мы будем в переменной **answer**. Модифицированный код приведен в листинге 4.

Теперь нам осталось только восстановить ответ и вывести его, поскольку вся необходимая для этого информация у нас уже есть. Запишем встречи из найденного набора в массив **res** в том порядке, в котором Алексей будет их выполнять (листинг 5).

Листинг 3. Подсчет значений массива **possible**

```
possible[0] := true;
for mask := 0 to maskCount do begin
  if (not possible[mask]) then
    continue;
  for i := 1 to n do begin
    if (contain(mask, i)) then
      continue;
    if (start + maskChange[mask] < min[i]) then
      continue;
    if (start + maskChange[mask] > max[i]) then
      continue;
    possible[add(mask, i)] := true;
  end;
end;
```

Листинг 4. Модификация подсчета значений массива possible

```
answer := 0;
possible[0] := true;
for mask := 0 to maskCount do begin
  if (not possible[mask]) then
    continue;
  for i := 1 to n do begin
    if (contain(mask, i)) then
      continue;
    if (start + maskChange[mask] < min[i]) then
      continue;
    if (start + maskChange[mask] > max[i]) then
      continue;
    possible[add(mask, i)] := true;
    last[add(mask, i)] := i;
  end;
  if (count(mask) > count(answer)) then
    answer := mask;
end;
```

Листинг 5. Восстановление ответа

```
mask := answer;
for i := count(mask) downto 1 do begin
  res[i] := last[mask];
  mask := remove(mask, last[mask]);
end;
```

По приведенному коду несложно оценить время его работы: суммарно программа выполнит $O(2^n \times n)$ операций, и ее время выполнения при данных в условии ограничениях однозначно уложится в требование на время работы программы.

Литература

1. *Романовский И. В.* Дискретный анализ. СПб.: Невский Диалект; БХВ-Петербург, 2003.
2. Задача коммивояжера, ДП по подмножествам. Материал вики-конспектов НИУ ИТМО. (neerc.ifmo.ru/wiki/index.php?title=Задача_коммивояжера,_ДП_по_подмножествам).
3. Битовые операции. Википедия. (ru.wikipedia.org/wiki/Битовые_операции).

Кротков Павел Андреевич,
студент четвертого курса кафедры «Компьютерные технологии» НИУ ИТМО, член жюри Интернет-олимпиад по информатике,

Поромов Сергей Сергеевич,
студент шестого курса кафедры «Компьютерные технологии» НИУ ИТМО, член жюри Интернет-олимпиад по информатике,

Ульянцев Владимир Игоревич,
студент шестого курса кафедры «Компьютерные технологии» НИУ ИТМО, член жюри Интернет-олимпиад по информатике.