Аделаида и очередь

Медведи и зайцы решили пойди в лесной кинотеатр на премьеру фильма «Медвежья услуга». В кинотеатре работала только одна касса, в которой работала сова Аделаида, которая никогда никуда не спешила. Очередь в кассу выстроилась приличная, а время сеанса неумолимо приближалось. Очередь начала роптать. В этих условиях умный медведь Гоша заметил, что если Аделаида продает кому-то несколько билетов сразу, то она тратит на это меньше времени, как если бы она продавала эти билеты по одному. Поэтому Гоша разработал схему: группа из нескольких подряд стоящих зайцев и медведей передает деньги стоящему из них впереди, и он покупает билеты на всю группу. Но тут Аделаида вспомнила, что по правилам кинотеатра нельзя продавать более трех билетов в одни руки. Поэтому стоящим в очереди можно было объединиться только в пары или тройки. Гоша, зная время, которое тратит Аделаида на продажу каждому зайцу или медведю из очереди одного, двух или трех билетов, задался вопросом: за какое минимальное время Аделаида сможет обслужить всю очередь? Помогите Гоше найти ответ на этот вопрос.

Требуется написать программу, которая поможет медведю Гоше вычислить минимальное время для покупки билетов всей очереди.

Формат входных данных:

На вход подается натуральное число N – количество зайцев и медведей в очереди ($1 \le N \le 5000$). Далее в N строках следуют тройки натуральных чисел $T1_i$, $T2_i$, $T3_i$, задающие количество секунд, которое тратит Аделаида на продажу одного, двух, трех билетов i-тому покупателю ($1 \le T1_i$, $T2_i$, $T3_i \le 3600$). Покупатели в очереди нумеруются обычным образом: первым считается тот, кто стоит первым в кассу.

Формат выходных данных:

На выходе должно получиться единственное натуральное число, равное времени в секундах, за которое все зайцы и медведи, стоящие в очереди, получат билеты от Аделаиды.

Пример входных и выходных данных:

Вход	*Выход*
2 5 6 7 2 2 2	6
3 5 6 8 2 2 2 1 3 5	7
5 5 10 15 2 10 15 5 5 5	12

20 20 1 20 1 1	

Решение:

Решение задач основано на идее динамического программирования.

Будем записывать в i-тый элемент вспомогательного массива возможное минимальное время на покупку билетов очередью, состоящей из i покупателей. Если очередь пуста, то и времени не будет затрачено ни секунды. Поэтому:

`time[0]=0;`

Если очередь состоит из одного покупателя билетов, то ответом будет число $T1_1$. Запишем его в первый элемент вспомогательного массива time:

`time[1]=`$T1_1$`;`

Допустим, что очередь состоит из двух зайцев. Они могут купить билеты вместе. В этом случае им понадобится время $T2_1$, т.к. покупать билеты будет первый заяц. Если же каждый из них будет покупать билет отдельно, то время работы Аделаиды будет составлять $T1_1+T1_2$. Находим минимальное значение из $T2_1$ и $T1_1+T1_2$ и записываем во второй элемент массива time:

`time[2]=min(`$T2_1$`, `$T1_1+T1_2$`);`

Допустим, что очередь состоит из трех зайцев. Если третий заяц покупает билет отдельно, то первым двум понадобится время, которое уже записано в time[2]. Поэтому время покупки будет составлять time[2]+$T1_3$. Если же второй и третий заяц решат объединиться для покупки билетов, то общее время на покупку составит time[1]+$T2_2$. Если же все три зайца купят билеты все вместе, то время покупки будет равно $T3_1$. Опять нужно найти минимальное значение из всех возможных вариантов и записать в time[3]:

`time[3]=min(time[2]+`$T1_3$` , time[1]+ `$T2_2$`, `$T3_1$`);`

Продолжаем рассуждать аналогичным образом. В результате получим общую формулу для расчета минимального времени покупки билетов i-тым покупателем:

`time[i]=min(time[i-1]+`$T1_i$`, time[i-2]+`$T2_{i-1}$`, time[i-3]+`$T3_{i-3}$`);`

Таким образом, в каждом элементе вспомогательного массива хранится текущая оценка минимального количества времени на покупку для очереди из i покупателей соответственно. Результатом решения задачи будет значение элемента массива time с индексом N. Значения переменных T1, T2 и T3 для каждого покупателя удобнее хранить в двумерном массиве.

```
#include <fstream>
#include <iostream>
# define M 5000
using namespace std;

int min2(int x, int y)
{ if (x<y) return x;
  else return y;
}

int main()
{
  ifstream in("input.txt");
  ofstream out("output.txt");
```

```
int T[M][3], time[M], i, N;
in >> N;
for (i=1; i<=N; i++)
{
        in >> T[i][0];
        in >> T[i][1];
        in >> T[i][2];
}
time[0] = 0;
time[1] = T[1][0];
time[2] = min2(T[1][0]+T[2][0], T[1][1]);
for (i=3; i<=N; i++)
  time[i]= min2(time[i-1]+T[i][0], min2(time[i-2]+T[i-1][1], time[i-3]+T[i-2][2]));
cout << time[N]<< endl;
out << time[N];
system("pause");
return 0;
}
```

Гоша и квадраты

Однажды умный медведь Гоша решил сделать подарок своим четырем друзьям – братьям-зайцам Ольгерду, Иннокентию, Йенсу и Никодиму. Для подарка он решил отрезать от земли, на которой располагалась его пасека, четыре квадратных участка, чтобы братья смогли посадить на них морковку с капустой. Для этого Гоша отмерил по тропинке, шедшей вдоль пасеки, отрезок длиной N, по которому собрался отмерять участки братьям. Гоша подумал и решил, что негоже будет, если старший брат Ольгерд получит такой же участок, что и младший Никодим. Поэтому Гоша постановил, что длина (она же и ширина) участков должна убывать от участка старшего к участку младшего из братьев-зайцев и не повторяться. Гоша еще подумал и решил, что уважающий себя медведь никогда не будет действовать себе в убыток, поэтому дареные участки должны быть возможно минимальной площади. А дальше Гоша, который очень любил математику, задумался, а как легко и быстро вычислить размеры четырех участков для отрезков тропинки различной длины?

Требуется написать программу, которая поможет медведю Гоше вычислить размеры участков каждого брата по заданной длине отрезка тропинки.

Формат входных данных:

На вход подается натуральное число N – длина отрезка тропинки, вдоль которого нужно нарезать квадратные участки огорода ($10 \le N \le 5000$).

Формат выходных данных:

На выходе должны получиться четыре натуральных числа K1, K2, K3, K4, обозначающие линейные размеры квадратных участков зайцев. При этом, должны быть выполнены условия Гоши:

1) $K1 + K2 + K3 + K4 = N$
2) $K1 > K2 > K3 > K4$
3) $K1^2 + K2^2 + K3^2 + K4^2$ – минимальная из всех возможных

Пример входных и выходных данных:

Вход	*Выход*
10	4 3 2 1

Решение:

Для решения задачи необходимо рассмотреть несколько первых значений N и выявить закономерность получения четырех чисел:

```
#include <iostream>
using namespace std;

int main()
{
  int N, k1, k2, k3, k4;
  cin >> N;


  k3 = (N-2) / 4; k4 = k3-1; k2 = k3+1; k1 = k3+2;
  if (N % 4 == 0)
  {
    k2++; k1++;
  }
  if (N % 4 == 1)
  {
    k3++; k2++; k1++;
  }
  if (N % 4 == 3)
  {
    k1++;
  }
  cout << k1 << " "<< k2 << " "<< k3 << " "<< k4 << endl;
  system("pause");
  return 0;
}
```

Ольгерд и команда

Однажды умный и деятельный заяц Ольгерд решил собрать команду медведей и зайцев для игры в хоккей с командой волков и бобров. Ольгерд точно знал, что для успеха команды важны не только умения отдельных игроков, но и сплоченность команды и слаженность действий ее участников. Для оценки качеств каждого игрока команды мудрый заяц ввел особую величину – коэффициент профессионализма – КП. Ольгерд вывел правило, согласно которому команда является единым слаженным коллективом, если КП каждого ее участника не больше суммарного КП любых двух других игроков. По этому правилу команда из одного или двух игроков всегда будет слаженной группой. На основании данного правила и КП отдельных игроков Ольгерд решил составить максимально слаженную команду медведей и зайцев, т.е. команду с максимальным суммарным КП. При этом, волки с бобрами постоянно меняли правила игры в хоккей, касающиеся количественного состава команд. Поэтому ограничений по количеству игроков при составлении «команды мечты» у Ольгерда не было.

Требуется написать программу, которая поможет зайцу Ольгерду составить максимально слаженную команду игроков, для каждого из которых посчитан его КП.

Формат входных данных:

На вход подается натуральное число N – исходное количество медведей и зайцев, для которых известны их КП ($1 \le N \le 30000$). В следующих N строках записаны N целых неотрицательных чисел КП, соответствующих значениям КП каждого потенциального игрока команды ($0 \le КП_i \le 60000$)

Формат выходных данных:

На выходе через пробел выводятся два числа: количество игроков в команде и их суммарный КП.

Пример входных и выходных данных:

Вход	*Выход*
4 1 5 3 3	3 11
5 100 20 20 20 20	2 120
15 1 6 1 7 1 1 6 1 1 6 6 6 7 1 1	8 45
15 1 6 1 7 7 1 6 1 1 6 6 6 7 7 1	10 59

Решение:

Отсортируем игроков по порядку неубывания их КП, используя какой-либо эффективный способ сортировки (в нашем случае мы взяли быструю сортировку).

Условием слаженности команды является выполнение требования: значение КП самого сильного участника команды на должен быть выше суммы значений КП двух самых слабых игроков. Поэтому для выбора максимально слаженной команды нужно найти непрерывную последовательность элементов отсортированного массива, для которой выполняется данное требование. При этом, команда из одного или двух игроков всегда будет слаженным коллективом, т.к. для такой команды данное требование точно выполнено. Если же количество игроков, для каждого из которых известен КП, превосходит 2, то выполняем следующую последовательность действий:

шаг (1): будем запоминать номера самого слабого (min) и самого сильного (max) игроков в отсортированной последовательности, а также сумму КП двух самых слабых игроков (sum_k) и суммарное значение КП всей команды (sum). Первоначально:
min=0; max=1; sum=igr[min]+igr[min+1]; sum_k =sum;

шаг (2): начиная с игрока под номером 2, проверяем, выполняется ли для него требование слаженности команды. Если требование выполнено, то увеличиваем КП команды на КП данного игрока и проверяем, есть ли еще претенденты на добавление в команду. Если игроки с неменьшим значением КП еще есть, то повторяем **шаг (2)**. В противном случае заканчиваем вычисления и выполняем **шаг (4)**.

шаг (3): в случае, если для очередного игрока требование слаженности команды не выполнено, то выбираем новую пару самых слабых игроков, сместив min на одну позицию вправо и, в случае необходимости, выполняем пересчет суммы. При этом, мы должны контролировать, чтобы в эту пару не попал бы текущий самый сильный игрок команды с номером max. Если сильный игрок не попал в пару, то пересчитываем суммарный КП новой пары и КП всей команды и повторяем **шаг (2)**. В противном случае увеличиваем суммарный КП всей команды на КП сильного игрока, заканчиваем вычисления и выполняем **шаг (4)**.

шаг (4): выводим количество игроков команды как значение выражения max-min+1 и суммарный КП как значение переменной sum.

```
#include <fstream>
#include <iostream>
# define M 5000
using namespace std;

void quick_sort(int le, int ri, int *ar)
{
  int i=le, j=ri, t=ar[(i+j)/2];
  do {
    while (ar[i]<t) i++;
    while (ar[j]>t) j--;
    if (i<=j)
    {
            int z=ar[i]; ar[i]=ar[j]; ar[j]=z; i++; j--;          }
    } while (i<=j);
    if (le<j) quick_sort(le,j,ar);
    if (i<ri) quick_sort(i,ri,ar);
}
```

```
int main()
{
  int N, i, rez[M];
  int igr[M];
  ifstream in("input.txt");
  ofstream out("output.txt");
  in >> N;
  for(i=0; i<N; i++)
    in >> igr[i];
  quick_sort(0, N-1, igr);

if (N == 1)
  {
    cout << "1 " << igr[0] << endl;
  }
  else
  {
    int min, max, sum, sum_2, sum_k,kol_k, flag=1;
    min = 0; max = 1;
    sum = igr[min] + igr[min+1];
    sum_k = sum_2 = sum;
    if (N == 2)
    {
      cout << "2 " << sum << endl;
    }
    else
    {
      max = 2; kol_k = 2;
      while ((max < N) && flag)
      {
        if (igr[max] <= sum_2)
        {
          sum += igr[max];
          if (max != N-1)
           max++;
          else {
              if (sum > sum_k) {
                  sum_k = sum; kol_k = max - min+1;
              }
              flag = 0;
          }
        }
        else
        {
          if (min < max-1)
          {
              if (sum > sum_k) {
                  sum_k = sum; kol_k = max - min;
              }
              sum-=igr[min]; min++; sum_2=igr[min]+igr[min+1];}
          else
          {
              sum = sum_2; max++;
          }
        }
       }
      // cout << kol_k << " " << sum_k << endl;
               out << kol_k << " " << sum_k << endl;
    }
  }

in.close();out.close();
  return 0;
}
```

Гоша и маршруты

Дома N своих друзей – медведей и зайцев, - мудрый медведь Гоша пронумеровал для себя натуральными числами от 1 до N. Некоторые дома соединены между собой тропинками и дорожками. Каждую такую лесную дорогу Гоша охарактеризовал двумя числами: d – длиной и r – уровнем надежности (ну, чем меньше там болот и завалов всяких, тем дорога надежней).

Гоша хочет добраться от зайца Никодима с номером 1 к зайцу Йенсу с номером N. Он считает маршрут *устойчивым*, если его минимальный уровень надежности среди пройденных им дорог максимально возможен. Среди всех устойчивых маршрутов мудрый Гоша выбирает тот, у которого минимальная суммарная длина пути.

Требуется написать программу, которая поможет медведю Гоше найти оптимальный маршрут для перемещения от дома одного друга к дому другого.

Формат входных данных:

В первой строке даны два числа N ($2 \le N \le 2*10^5$) и M ($1 \le M \le 3*10^5$), задающие количество домов друзей Гоши и лесных дорог.

В следующих M строках задана каждая дорога четверкой натуральных чисел u, v, d, r, где: u и v определяют номера домов ($1 \le u,v \le N$, $u \ne v$), d соответствует длине дороги ($1 \le d \le 10^9$), а r ($1 \le r \le 10^9$) – ее надежности.

Формат выходных данных:

На выходе через пробел выводятся два числа: максимальный уровень надежности маршрута и минимальная длина пути при таком уровне. Если маршрута не существует, то выводятся два числа -1 -1.

Пример входных и выходных данных:

Вход	*Выход*
4 5 1 2 5 7 2 3 3 6 1 3 10 9 3 4 2 6 2 4 7 8	7 12

Решение задачи

Суть задачи

Для заданного графа нужно найти маршрут из вершины 1 в вершину N, у которого среди всех дорог минимальный уровень надежности максимален. Среди таких маршрутов требуется выбрать тот, что имеет минимальную длину.

Алгоритм решения

1. **Поиск максимального минимума (reliability) с помощью бинарного поиска**
   Можно переформулировать задачу:
   - Зафиксируем некоторое значение надежности X.
   - Можем ли мы из города 1 попасть в город N, используя только дороги с надежностью $\geq X$ и подсчитать минимальный путь?
   - Хотим максимизировать такое X.
2. Это позволяет воспользоваться бинарным поиском по возможным значениям уровня надежности, а внутри бинарного поиска — запускать алгоритм Дейкстры для проверки достижимости и минимального расстояния по подграфу.
3. **Алгоритм**
   - Бинарный поиск по надежности X на промежутке $[1, 10^9]$.
   - Для каждого кандидата X, рассмотрим только те дороги, у которых надежность $\geq X$.
   - Проведём Дейкстру по этим дорогам: если добрались до N — обновляем ответ и увеличиваем X, иначе уменьшаем X.
   - После завершения бинарного поиска получаем максимальный возможный уровень надежности и минимальное расстояние.
4. **Сложность**
   - Бинарный поиск: $O(\log (maxR))$
   - Дейкстра: $O(M \log N)$ на каждую итерацию бинарного поиска.
   - Итоговая сложность: $O(M \log N \cdot \log (maxR))$, что укладывается в ограничения.

Примерный код решения на Python

```
import heapq


def dijkstra(n, adj, min_r):
    dist = [float('inf')] * (n + 1)
```

```
    dist[1] = 0
    hq = [(0, 1)]
    while hq:
        cur_dist, u = heapq.heappop(hq)
        if cur_dist > dist[u]:
            continue
        for v, d, r in adj[u]:
            if r < min_r:
                continue
            if dist[v] > dist[u] + d:
                dist[v] = dist[u] + d
                heapq.heappush(hq, (dist[v], v))
    return dist[n]

def solve(n, m, edges):
    adj = [[] for _ in range(n+1)]
    for u, v, d, r in edges:
        adj[u].append((v, d, r))
        adj[v].append((u, d, r))
    left, right = 1, max(r for _, _, _, r in edges)
    ans_r, ans_d = -1, -1
    while left <= right:
        mid = (left + right) // 2
        d = dijkstra(n, adj, mid)
        if d < float('inf'):
```

```
            ans_r, ans_d = mid, d
            left = mid + 1
        else:
            right = mid – 1
    return f"{ans_r} {ans_d}" if ans_r != –1 else "–1 –1"


# Для примера
n = 4
m = 5
edges = [
    (1, 2, 5, 7),
    (2, 3, 3, 6),
    (1, 3, 10, 9),
    (3, 4, 2, 6),
    (2, 4, 7, 8)
]
print(solve(n, m, edges))  # Выводит: 7 12
```

Итоги

- Используется бинарный поиск по допустимой надежности и Дейкстра для вычисления длины пути.
- Алгоритм работает эффективно даже для больших входных данных.

Если хочется, можно разобрать по шагам конкретный запуск на примере или прокомментировать каждую строку кода.

Гоша и конкурс

Совет зверей леса постановил, что всем его обитателям необходимо развивать математические способности для того, чтобы рационально применять их в организации жизни леса. Для этого решено было провести конкурс «Удивительные звери», а придумать задания поручили умному медведю Гоше.

Гоша был очень образованным и любознательным, а тут еще нашел в одном лесном домике забытую кем-то книгу «Комбинаторика». Именно из нее Гоша узнал о том, что есть интересные натуральные числа: факториалы и сочетания без повторений.

Он рассказал участникам, что факториалом натурального числа ***n*** называется произведение чисел от ***1*** до ***n*** включительно и имеет специальное обозначение. Так, ***n!=1*2*3*…*n***. И уточнил, что ***0!=1, 5!=120, 7!=5040.***

Число сочетаний *без повторений* из ***n*** элементов по ***m*** (***n*** и ***m*** – натуральные числа и ***n*** ≥***m***) обозначается C_n^m .

$$C_n^m = \frac{n!}{(n-m)! \bullet m!}$$

И привел пример вычислений: $C_{10}^8 = \frac{10!}{2! \bullet 8!} = 45.$ А затем уточнил, что это всегда натуральные числа и при ***m=0*** и ***m=n*** число сочетаний равно единице. Особенное внимание Гоша уделил тому, что другим способом это число можно вычислить по формуле: $C_n^m = C_{n-1}^{m-1} + C_{n-1}^m$

Все участники конкурса были под впечатлением от обширных знаний Гоши в области математики и внимательно слушали его задание для конкурса.

Участник должен выписать в строку непосредственно друг за другом несколько (***K***) значений сочетаний без повторений из ***n*** элементов по ***m,*** а затем определить цифру, которая стоит в данной последовательности цифр на конкретном месте (***B***), которое укажет ведущий конкурса.

Требуется написать программу, которая поможет медведю Гоше быстро определить цифру в указанной позиции.

Формат входных данных:

На вход подается натуральное число ***B*** – позиция искомой цифры в последовательности (1≤ ***B*** ≤5000) и ***K*** – количество числа сочетаний без повторений, которые предстоит вычислить и выписать в строчку участникам конкурса (1≤ ***K*** ≤ 500). При этом гарантировано, что в полученной последовательности обязательно есть цифра с индексом ***B.***

Далее в каждой из ***K*** строк через пробел записаны пары натуральных чисел n_i и m_i (0≤ n_i ,m_i ≤30), которые определяют вычисление очередного сочетания без повторений из ***n*** элементов по ***m (1≤ i ≤K)***.

Формат выходных данных:

На выходе должна получиться цифра, которая стоит в указанной позиции.

Пример входных и выходных данных:

Вход	*Выход*
6 3 12 6 8 4 16 12	1
5 3 6 3 12 1 14 2	9

Решение:

Решение может быть организовано непосредственно по определению числа сочетаний без повторений из ***n*** элементов по ***m*** (***n*** и ***m*** – натуральные числа и ***n*** ≥***m***). Для небольших значений ***n*** и ***m*** такой поход может привести к результату. Но это очень рискованный путь, поскольку содержит много нерациональных действий при вычислении факториалов.

Гораздо рациональнее вычислять число сочетаний без повторений по рекуррентной формуле, приведенной в условии. Базовые случаи: при ***m=0*** и ***m=n*** число сочетаний равно единице. Однако, этот подход может привести к рекурсивному взрыву, поскольку содержит опасность многократных вычислений одних и тех же значений.

Выход из этого может быть при использовании дополнительной памяти в виде двумерного массива, в котором мы храним вычисленные ранее значения числа сочетаний без повторений из ***n*** элементов по ***m.*** При необходимости мы просто будем брать из него эти данные.

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
#define N 40
int soch[N][N] = { 0 };
int ch_soch(int n, int m)
{  if (soch[n][m] == 0)
    {   int temp1 = ch_soch(n - 1, m - 1);
        int temp2 = ch_soch(n - 1, m);
        soch[n][m] = temp1 + temp2;
    }
return soch[n][m];
}
void inicializacia()
{   int i, j;
    for (i = 0; i < N; i++)
        for (j = 0; j <= i; j++)
            if ((j == 0) || (j == i))  soch[i][j] = 1;
            else soch[i][j] = 0;
}
```

```
int main()
{   inicializacia();
    ifstream input;
    ofstream output;
    input.open("input.txt"); output.open("output.txt");
    int pozicia, kolichestvo_chisel, n, m;
    input >> pozicia;
    input >> kolichestvo_chisel;
    string stroka;
    bool flag = true;
    for (int i = 1; i <= kolichestvo_chisel && flag; i++)
    {   input >> n; input >> m;
        int chislo = ch_soch(n, m);
        //cout << "n=" << n << "   m=" << m << "  cochetania   " << chislo << endl;
        stroka = to_string(chislo);
        int dlina = stroka.size();
        if (dlina < pozicia) pozicia -= dlina;
        else {
            output << stroka[pozicia - 1];
            flag = 1 == 2;
        }
    }
    input.close(); output.close();
    return 0;
}
```