Первый отборочный этап

Первая попытка. 5 декабря 2023 г.

Задача A. Пеш-хматы

Стандартный ввод.

Стандартный вывод.

2 секунды.

256 мегабайт.

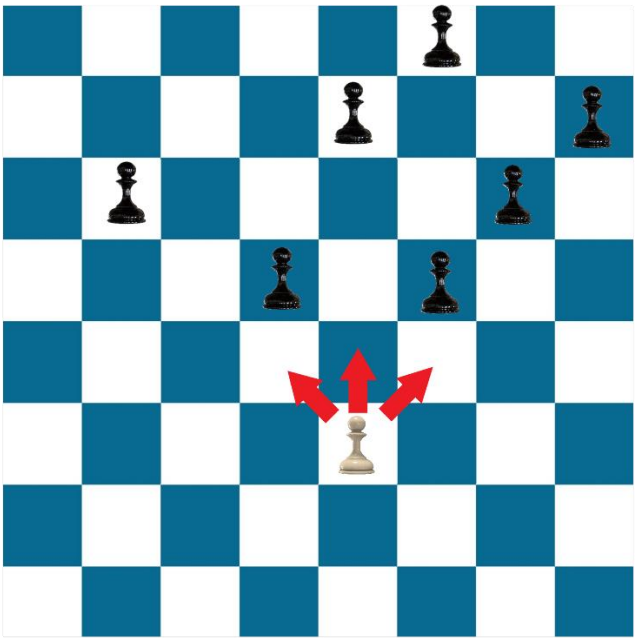
Сергею подарили игру «Пеш-хматы». В этой игре предоставлено шахматное поле размером *w* x *h*. Клетка (1; 1) расположена в левом нижнем углу. На нем расставлены черные пешки.

Черные пешки не могут двигаться, в то время как белая может ходить прямо вверх (*U*), по диагонали вправо и вверх (*R*) и по диагонали влево и вверх (*L*). В этой игре пешка может взять любую черную фигуру, которая расположена впереди неё на одну клетку прямо вверх или по диагонали.

Задача игрока состоит в том, чтобы выбрать куда поставить пешку и составить маршрут для белой пешки так, чтобы она съела как можно больше черных фигур. Белую пешку нельзя ставить на поле, где есть черная пешка.

Маршрут описывается последовательностью букв:

- *U* – пешка идет вперед,
- *R* – пешка идет по диагонали вправо,
- *L* – пешка идет по диагонали влево.



Даны четыре тестовых картинки, для которых требуется найти маршрут. Если вы не знаете ответ для теста, запишите *X*.

Как ответ на эту задачу прикрепите код или *txt* файл. Если отсылаете *txt* файл то на первой строчке будет записан маршрут для первого теста, во второй строчке будет записан маршрут для второго теста и тд. Если отсылаете код, то на первой строке выходных данных выведите маршрут для первого теста, во второй строчке будет записан маршрут для второго теста и тд.

По этой задаче на проверку принимается не более трех файлов.

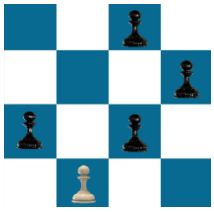
Каждый тест оценивается от 0 до 25 баллов по формуле

$$25 * \frac{points_{participant}}{points_{max}}$$

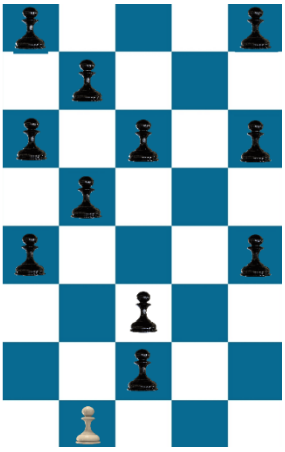
где $points_{participant}$ количество съеденных фигур маршрутом участника, $points_{max}$ – количество съеденных фигур маршрутом жюри. Если маршрут содержит символы кроме *L, R, U, X* или выходит за границы поля, то оценивается он будет 0 баллов.

Номер теста	Ответ
1	(поле ответа)
2	(поле ответа)
3	(поле ответа)
4	(поле ответа)

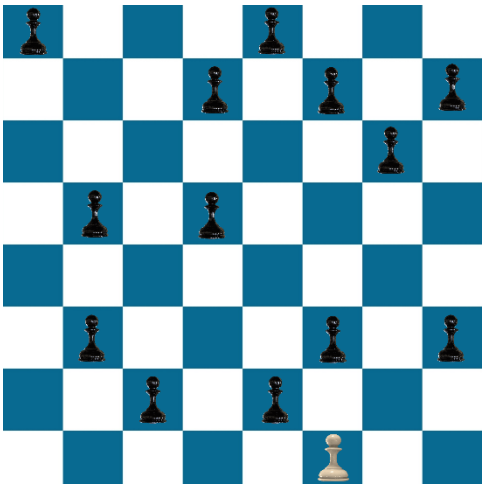
Тест 1



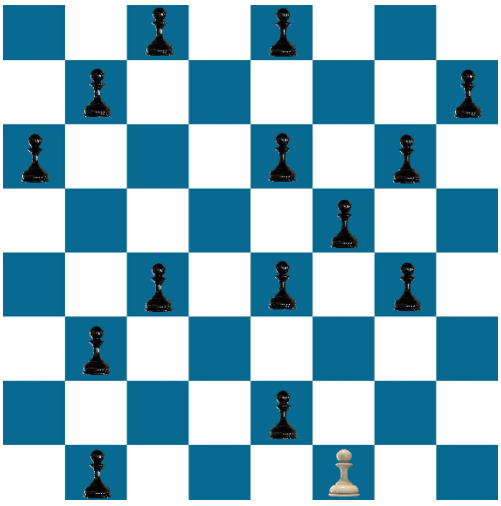
Тест 2



Тест 3



Тест 4



Решение:

Постановка задачи: Дано шахматное поле стоят черными пешками, нужно составить маршрут для белой пешки так, чтобы она съела как можно больше черных фигур.

Тест 1

Для первого теста можно было перебрать все варианты маршрутов и проверить каждый.

Тест 2

Для второго теста можно было перебрать варианты маршрута, отсеивая большинство заведомо плохих ходов.

Тесты 3 и 4

Для полного решения задачи используем метод динамического программирования. Динамическое программирование позволяет избежать повторных вычислений и значительно ускорить процесс.

Для удобства пусть клетка *(1; 1)* находится в левом верхнем углу.

Пусть *dp[i][j]* — количество съеденных черных фигур, если мы стартуем с *i* строчки и *j* столбца (допускаем, что если на *i* строчке и *j* столбце стоит пешка, то мы можем поставить туда пешку и съесть фигуру).

База динамики: инициализируем базу динамики значениями на первой строке, учитывая наличие черных фигур. То есть *dp[1][j]* = 1 при наличии черной фигуры и *dp[1][j]* = 0 при отсутствии в *(1; j)* клетке.

Так как для первой строчки ответ посчитан, то переходы для строк *i*, где *i* > 1. Динамика заполняется, учитывая возможность движения белой пешки вверх и по диагонали. Если пешка стоит на (*i; j*) позиции, то она может пойти в *(i − 1; j − 1), (i − 1; j)* и *(i − 1; j + 1)*, для которых ответы уже посчитаны. Переходы динамики: *dp[i][j] = max(dp[i−1][j −1], dp[i−1][j], dp[i−1][j + 1])*. Если *(i; j)* клетке стоит черная пешка, то прибавим 1 к *dp[i][j]*.

Параллельно вычисления динамики поддерживаем маршрут.

Номер теста	Ответ
1	RRL
2	RUULRLL
3	LRLLUUR
4	LUURRR

Задача B. Цирк

Стандартный ввод.

Стандартный вывод.

2 секунды.

256 мегабайт

В честь дня рождения цирка клоун Пилюлькин раздает детям бесплатные билеты на свободные места, которые остались после продажи билетов. Арена цирка разделена на четыре сектора *A*, *B*, *C* и *D*.

Пилюлькин раздает билеты по следующему принципу: первому ребенку достается билет в сектор *A*, второму – в сектор *B*, третьему – в сектор *C*, четвертому – в сектор *D*, пятому опять в сектор *C*, шестому – в сектор *B* и так далее (Последовательность секторов имеет вид: *A-B-C-D-C-B-A-B-C-D-…*).

Как только у Пилюлькина не будет возможности подарить билет в очередной сектор, он прекращает бесплатную раздачу билетов и начинает представление.

Сколько детей смогут посетить цирк бесплатно, если никаким другим способом бесплатный билет получить нельзя?

Формат входных данных

В первой строке входного файла задано целое число A $(0 \le A \le 2*10^{16})$ – количество билетов в сектор A.

Во второй строке входного файла задано целое число B $(0 \le B \le 2*10^{16})$ – количество билетов в сектор B.

В третьей строке входного файла задано целое число C $(0 \le C \le 2*10^{16})$ – количество билетов в сектор C.

В четвертой строке входного файла задано целое число D $(0 \le D \le 2*10^{16})$ – количество билетов в сектор D.

Формат выходных данных

Выведите количество билетов, которые сможет раздать Пилюлькин до начала представления.

Система начисления баллов

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
0	0	Тесты из условия	–	полная
1	15	$A=B=C=D \le 10^6$	–	первая ошибка
2	25	$A, B, C, D \le 10^6$	1	первая ошибка
3	15	$A=B=C=D \le 2*10^{16}$	1	первая ошибка
4	45	Нет дополнительных ограничений	1–3	первая ошибка

Решение:

Постановка задачи: определить количество детей, которые могут посетить цирк бесплатно в соответствии с порядком раздачи билетов в сектора $A, B, C, D, C, B, A,\ldots$.

Подзадача 1

Моделирование

Для первой подзадачи можно написать решение, которое будет моделировать процесс раздачи билетов.

```
1. vector < long long > ticket = {a , b , c , d };
2. vector < long long > tickets = {0 , 1 , 2 , 3 , 2 , 1};
3. long long ans = 0;
4. while ( ticket [ tickets [ it % 6]] != 0) {
5. ticket [ tickets [ it % 6]] - -;
6. ans ++;
7. }
```

Полное решение

Для полного решения задачи оптимизируем моделирование. Заметим, что последовательность секторов повторяется циклически *(A, B, C, D, C, B)*. В цикле в сектора *A* и *D* было отдано по одному билету, в сектора *B* и *C* — по два билета. Посчитаем сколько полных циклов могло быть сделано *cycle = min(A, D, [B/2], [C/2])*. Запись *[x]* обозначает округление вниз значения *x* (например, *[1.4]* = 1, *[2]* = 2).

После того как мы вычислили количество полных циклов осталось посчитать сколько будем отдано билетов в последнем не полном цикле. Для этого изменим $A = A - cycle$, $B = B - 2 \cdot cycle$, $C = C - 2 \cdot cycle$, $D = D - cycle$.

После этого можно использовать алгоритм из первой подзадачи.

```
1. long long cycle = min (
2.     min (a , d ) ,
3.     min (( long long ) b /2 , ( long long ) c /2)
4. ) ;
5. long long ans = cycle * 6 LL ;
6. a -= cycle ;
7. b -= 2 LL * cycle ;
8. c -= 2 LL * cycle ;
9. d -= cycle ;
10. vector < long long > ticket = {a , b , c , d };
11. vector < long long > tickets = {0 , 1 , 2 , 3 , 2 , 1};
12. long long ans = 0;
13. while ( ticket [ tickets [ ans % 6]] != 0) {
14.     ticket [ tickets [ ans % 6]] - -;
15.     ans ++;
16. }
17. cout << ans + cycle * 6 LL ;
```

Задача C. Слова

Стандартный ввод.
Стандартный вывод.
2 секунды.
256 мегабайт

Вы упражняетесь в слепой печати, и сегодня у вас по плану напечатать m слов. Для этого вы можете выбрать произвольные m слов из набора $s_1, \dots, s_n$ $(1 \le n \le 2*10^5)$, и напечатать их через пробел. Причем каждое слово может быть выбрано не более одного раза.

Кроме того, в каждом из слов набора могут встречаться только первые k $(1 \le k \le 26)$ букв латинского алфавита.

Вы знаете, сколько времени у вас занимает перенести палец с одной буквы на клавиатуре на другую. Более формально, известна матрица $t_{i,j}$ $(1 \le i, j \le k)$, где $t_{i,j}$ равняется времени, которое вы потратите на перенос пальца с i-го символа на j-й символ. В этой задаче мы пренебрежем временем на перенос пальца с символа на пробел и наоборот.

Таким образом, чтобы напечатать два слова *«ab»* и *«b»* потребуется перенести палец с буквы *«a»* на букву *«b»*, потратив $t_{1,2}$ времени, после чего мгновенно перенести палец с буквы *«b»* на пробел и затем мгновенно перенести палец с пробела на букву *«b»*. Значит, суммарно потребуется $t_{1,2}$ времени.

Вы хотите напечатать m слов из набора таким образом, чтобы минимизировать затраченное время. Найдите это время.

Формат входных данных

В первой строке записано натуральное число n $(1 \le n \le 2*10^5)$ – количество слов в наборе.

Во второй строке записано n строк $s_1, \dots, s_n$, каждая из которых состоит из строчных латинских букв.

В третьей строке записано натуральное число m $(1 \le m \le n)$ – количество слов, которые необходимо выбрать для печати.

В четвертой строке записано натуральное число k $(1 \le k \le 26)$, задающее множество первых k латинских букв, из которых могут состоять слова.

В следующих строках записана матрица $t_{i,j}$ $(1 \le t_{i,j} \le 10^9)$ размера k x k, состоящая из неотрицательных целых чисел.

Гарантируется, что суммарная длина всех слов не превосходит $2*10^5$.

Формат выходных данных

Выведите единственное число --- минимальное время, необходимое для набора m слов.

Система начисления баллов

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
0	0	Тесты из условия	–	полная
1	20	$k = 1$	–	первая ошибка
2	20	Сумма длин всех слов не больше 1000	–	первая ошибка
3	30	$n \le 1000$	2	первая ошибка
4	30	Нет дополнительных ограничений	1–3	первая ошибка

Решение:

Заметим, что время набора каждого слова не зависит от того, каким по счету мы его будем набирать, если вообще будем. Поэтому для каждого слова можно сразу посчитать это время. Чтобы это сделать, пройдемся по каждому слову $s_1, \dots, s_l$ и добавим к ответу $t_{s1,s2} + \dots + t_{sl-1,sl}$.

Далее нужно выбрать m слов с минимальным временем набора. Для этого отсортируем слова в соответствии с необходимым для набора временем, после чего ответ – это сумма m минимальных временем.

Задача D. Пеш-хматы

Стандартный ввод.

Стандартный вывод.

2 секунды.

256 мегабайт.

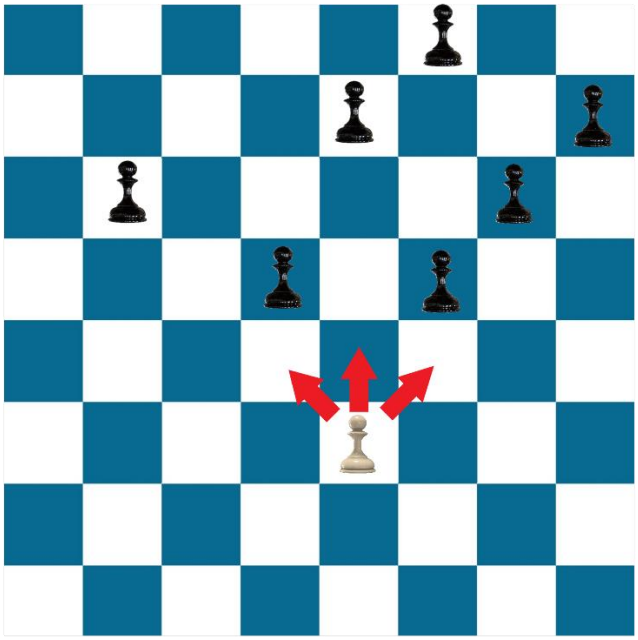
Сергею подарили игру «Пеш-хматы». В этой игре предоставлено шахматное поле размером w x h. Клетка (1; 1) расположена в левом нижнем углу. На нем расставлены черные пешки.

Черные пешки не могут двигаться, в то время как белая может ходить прямо вверх (*U*), по диагонали вправо и вверх (*R*) и по диагонали влево и вверх (*L*). В этой игре пешка может взять любую черную фигуру, которая расположена впереди неё на одну клетку прямо вверх или по диагонали.

Задача игрока состоит в том, чтобы выбрать куда поставить пешку и составить маршрут для белой пешки так, чтобы она съела как можно больше черных фигур. Белую пешку нельзя ставить на поле, где есть черная пешка.

Маршрут описывается последовательностью букв:

- *U* – пешка идет вперед,
- *R* – пешка идет по диагонали вправо,
- *L* – пешка идет по диагонали влево.



Формат входных данных

В первой строке входного файла задано два целых числа h и w ($1 \le w, h \le 2 * 10^5$) – размеры шахматного поля.

Гарантируется, что $w * h \le 2 * 10^5$ и что существует свободная / пустая клетка.

В каждой из следующих h строк записано по w символов a_{ij} (a_{ij} {*, B}), где

- * – пустая клетка,
- *B* – на клетке стоит черная пешка.

Формат выходных данных

Выведите в первой строке два целых числа, куда поставить пешку (номер строки и номер столбца).

Во второй строке выведите количество съеденных пешек.
Если пешка съела хотя бы одну фигуру, то на третей строке выведите маршрут, чтобы белая пешка съела наибольшее количество черных фигур.
Если ответов несколько, то выведите любой.

Система начисления баллов
Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
0	0	Тесты из условия	–	Полная
1	30	$h, w \leq 10$	–	Первая ошибка
2	35	$h, w \leq 30$	1	Первая ошибка
3	35	Нет дополнительных ограничений	1-2	Первая ошибка

Решение:
Постановка задачи: Дана матрица, на которой стоят черные фигуры, нужно поставить пешку и составить маршрут для белой пешки так, чтобы она съела как можно больше черных фигур. Белую пешку нельзя ставить на поле, где есть черная пешка.
Подзадача 1
Наивный рекурсивный перебор: для первой подзадачи можно написать наивное рекурсивное решение. Рекурсивная функция будет рассматривать все возможные позиции белой пешки, проверять их наличие на поле и, при возможности, рекурсивно вызывать саму себя для следующего шага. Такое решение может быть полезным для небольших размеров поля, но оно неэффективно для больших размеров из-за повторных вычислений.

```
def rec (x , y ) :
    global ans , pathAns , curPath , cur , a , w
    if x == 0:
        if cur > ans :
            ans = cur
            pathAns = curPath
        return

    if y - 1 >= 0:
```

```
    curPath += "L"
    cur += ( a [ x - 1][ y - 1] == 'B')
    rec ( x - 1 , y - 1)
    cur -= ( a [ x - 1][ y - 1] == 'B')
    curPath = curPath [: -1]
  if y + 1 < w :
    curPath += "R"
    cur += ( a [ x - 1][ y + 1] == 'B')
    rec ( x - 1 , y + 1)
    cur -= ( a [ x - 1][ y + 1] == 'B')
    curPath = curPath [: -1]

  curPath += "U"
  cur += ( a [ x - 1][ y] == 'B')
  rec ( x - 1 , y )
  cur -= ( a [ x - 1][ y] == 'B')
  curPath = curPath [: -1]
  return

def solve () :
  global ans , pathAns , curPath , cur , a , w , h , ansX , ansY
  h , w = map (int , input () . split () )

  a = []
  for _ in range ( h ) :
    a . append ( input () )

  last = -1
  for j in range (1 , h ) :
    for i in range ( w ) :
      if a [ j ][ i ] == '*':
        last = ans
        rec (j , i )
        if ans > last :
          ansX , ansY = j , i
  print (h - ansX , ansY + 1)
  print ( ans )
```

```
    print ( pathAns )
    return

ans = 0
cur = 0
pathAns = ""
curPath = ""
ansX = 0
ansY = 0
w = 0
h = 0
a = []

if __name__ == " __main__ ":
    solve ()
```

Полное решение

Для удобства пусть клетка (1; 1) находится в левом верхнем углу. Для полного решения задачи используем метод динамического программирования. Динамическое программирование позволяет избежать повторных вычислений и значительно ускорить процесс.

Пусть *dp[i][j]* – количество съеденных черных фигур, если мы стартуем с *i* строчки и *j* столбца (допускаем, что если на *i* строчке и *j* столбце стоит пешка, то мы можем поставить туда пешку и съесть фигуру).

База динамики: инициализируем базу динамики значениями на первой строке, учитывая наличие черных фигур. То есть *dp[1][j]* = 1 при наличии черной фигуры и *dp[1][j]* = 0 при отсутствии в *(1; j)* клетке.

Так как для первой строчки ответ посчитан, то переходы для строк *i*, где *i* > 1. Динамика заполняется, учитывая возможность движения белой пешки вверх и по диагонали. Если пешка стоит на *(i; j)* позиции, то она может пойти в *(i − 1; j − 1)*, *(i − 1; j)* и *(i − 1; j + 1)*, для которых ответы уже посчитаны. Переходы динамики: *dp[i][j] = max(dp[i−1][j −1], dp[i−1][j], dp[i−1][j + 1])*. Если *(i; j)* клетке стоит черная пешка, то прибавим 1 к *dp[i][j]*.

Ответ находится в *(i; j)* клетке, где не стоит черная фигура и наибольшее значение. Так как мы поменяли местоположение клетки *(1; 1)*, то выводить надо *(h − i + 1, j)*. Для восстановления маршрута воспользуемся массивом предков, то есть запомним куда мы пошли из клетки *(i; j)*.

```
def solve () :
    h,w = map(int, input().split())

    a = [input() for _ in range(h)]

    dp = [[0] * w for _ in range(h)]
    parent = [[-1] * w for _ in range(h)]
    for i in range(w):
        dp[0][i] = int(a[0][i] == 'B')
    for i in range(1,h):
        dp[i][0] = int(a[i][0] == 'B') + dp[i - 1][0]
        parent[i][0] = 0
        if 1 < w and int(a[i][0] == 'B') + dp[i - 1][1] > dp[i][0]:
            dp[i][0] = int(a[i][0] == 'B') + dp[i - 1][1]
            parent[i][0] = 1

        dp[i][w - 1] = int(a[i][w - 1] == 'B') + dp[i - 1][w - 1]
        parent[i][w - 1] = w - 1
        if w-2 >= 0 and int(a[i][w - 1] == 'B') + dp[i-1][w-2] > dp[i][w-1]:
            dp[i][w - 1] = int(a[i][w - 1] == 'B') + dp[i - 1][w - 2]
            parent[i][w - 1] = w - 2

        for j in range (1 , w - 1) :
            if dp[i - 1][j - 1] > dp[i][j]:
                dp[i][j] = dp[i - 1][j - 1]
                parent[i][j] = j - 1

            if dp[i - 1][j] > dp[i][j]:
                dp[i][j] = dp[i - 1][j]
                parent[i][j] = j
            if dp[i - 1][j + 1] > dp[i][j]:
                dp[i][j] = dp[i - 1][j + 1]
                parent[i][j] = j + 1
                dp[i][j] += int(a[i][j] == 'B')

    ans = (-1 , -1)
    for i in range(h) :
```

```
    for j in range(w) :
      if a[i][j]==’*’ and (ans[0]==-1 or dp[ans[0]][ans[1]]<dp[i][j]):
        ans = (i,j)

print(h - ans[0], ans[1] + 1)
print(dp[ans[0]][ans[1]] - int(a[ans[0]][ans[1]] == ’B’))
s = ""
while parent[ans[0]][ans[1]] != -1:
  if parent[ans[0]][ans[1]] - ans[1] == -1:
    s += "L"
  elif parent[ans[0]][ans[1]] - ans[1] == 0:
    s += "U"
  elif parent[ans[0]][ans[1]] - ans[1] == 1:
    s += "R"
  ans = (ans[0] - 1, parent[ans[0]][ans[1]])
print(s)

if __name__ == " __main__ ":
  solve()
```

Задача E. Плейлист

Стандартный ввод.

Стандартный вывод.

2 секунды.

256 мегабайт

Единорог очень любит прослушивать свой плейлист. Для разнообразия единорог каждый раз выбирает новый порядок песен, и этот раз не стал исключением. Единорог решил выбрать такой порядок песен, чтобы первое время продолжительность песен неубывала, а потом невозрастала. Помогите единорогу – найдите количество способов перемешать песни так, чтобы единорогу было интересно слушать плейлист.

Более формально: дан массив $a_1, \ldots, a_n$ $(1 \le a_i \le 10^9)$, где a_i – это продолжительность i-й песни. Требуется найти количество способов переупорядочить элементы массива a таким образом, чтобы существовал такой индекс k, что $a_i \le a_{i+1}$ для всех $1 \le i \le k-1$, а также $a_i \le a_{i+1}$ для всех $k \le i \le n-1$.

Так как ответ может быть очень большим, выведите его остаток от деления на 10^9+7.

Формат входных данных

В первой строке входного файла задано число n $(1 \le n \le 2*10^5)$ – количество песен в плейлисте Спаркса.

Во второй строке входного файла задано n чисел $a_1, \ldots, a_n$ $(1 \le a_i \le 10^9)$, где a_i – это продолжительность i-й песни.

Формат выходных данных

Выведите единственное число – ответ на задачу по модулю 10^9+7.

Система начисления баллов

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи
0	0	Тесты из условия	–
1	10	$n \le 10$	–
2	20	Все a_i попарно различны	–
3	20	$n \le 20$	1
4	50	Нет дополнительных ограничений	1–3

Решение:

Для решения первой подзадачи достаточно перебрать все перестановки из n элементов и для каждой проверить, будет ли она интересна Спарксу.

Рассмотрим решение подгруппы, где все числа различны. Заметим, что до максимального элемента все числа будут возрастать, а после него – убывать. Таким образом, каждый элемент, кроме максимума, может находиться либо в левой (возрастающей) части, либо в правой (убывающей). Кроме того, если мы зафиксировали, в какой части находится каждый элемент, то ответ определяется единственным образом. Это значит, что каждый способ распределить элементы по частям соответствует одному корректному порядку, и наоборот. Поэтому ответ – это количество способов так распределить элементы. Это число равно 2^{n-1}, потому что каждый элемент, кроме максимума, может быть либо в левой части, либо в правой.

Для решения на полный балл заметим следующее. Во-первых, все вхождения максимального элемента будут идти подряд в пиковой части последовательности, по аналогии со случаем с уникальными элементами. Для остальных элементов сколько-то может находиться слева от всех максимумов, сколько-то – справа. Обозначим за $c_1, \ldots, c_{max}$ количество вхождений в массив a чисел $1, 2, \ldots, max$. Тогда

есть $(c_1 + 1) \cdot \ldots \cdot (c_{max-1} + 1)$ способов выбрать разбиение всех чисел. Однако, мы не учли то, что для нас порядок всех вхождений каждого уникального числа i важен. Поэтому нужно еще домножить на $c_1! \cdot \ldots \cdot c_{max}!$. Все числа c_i не превосходят n, поэтому факториалы можно предподсчитать за $O(n)$. Итого ответ равен $(c_1 + 1)! \cdot \ldots (c_{max-1} + 1)! \cdot c_{max}!$. Чтобы реализовать это решение за $O(n)$ или $O(n \log n)$, можно применить сжатие координат (заменить каждое значение a_i на индекс в отсортированном массиве), либо использовать ассоциативную структуру данных (например, «std::map» или «std::unordered_map» в языке C++).

Вторая попытка. 24 декабря 2023 г.

Задача A. Лист бумаги

Стандартный ввод.
Стандартный вывод.
1 секунда.
256 мегабайт

Мария в преддверии нового года, решила подарить открытки своим друзьям. Она хочет, чтобы открытки были размера x x y, ширины – x и высоты – y. Мария вырезает их из листа размера w на h, все открытки расположены в одном положении вертикально или горизонтально. Стороны открыток должны быть параллельны границам листа. Маша хочет узнать какое максимальное количество открыток она сможет сделать?

Формат входных данных

В первой строке входного файла записаны через пробел два целых числа w и h ($1 \le w * h \le 10^{18}$) – размеры листа бумаги.
Во второй строке входного файла записаны через пробел два целых числа x и y ($1 \le x * y \le 10^{18}$) – размеры открытки.

Формат выходных данных

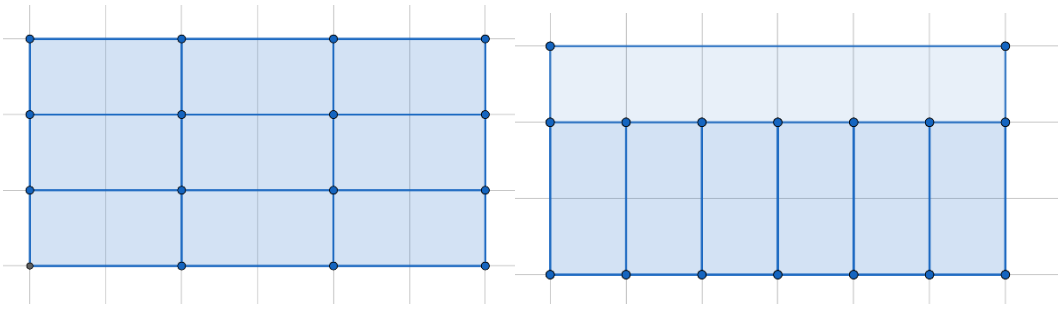
Выведите одно число – наибольшее количество открыток, которое сможет сделать Мария.

Система начисления баллов

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
0	0	Тесты из условия	–	полная
1	30	$x = 1$, $y = 1$	–	первая ошибка
2	30	$x = y$	1	первая ошибка
4	40	Нет дополнительных ограничений	1–2	первая ошибка

Пример



Два способа изготовления картинок

Решение:

Полное решение: всего два положения открыток, разберем каждый:

- Если располагать открытоки вертикально, то по ширине листа умещается $\lfloor x/w \rfloor$ открыток, а по высоте $\lfloor y/h \rfloor$. Значит всего открыток таким расположением можно добиться $\lfloor x/w \rfloor \cdot \lfloor y/h \rfloor$.
- Если располагать открытоки горизонтально, то по ширине листа умещается $\lfloor x/h \rfloor$ открыток, а по высоте $\lfloor y/w \rfloor$. Значит всего открыток таким расположением можно добиться $\lfloor x/h \rfloor \cdot \lfloor y/w \rfloor$.

Ответом будет являться наибольшее из двух чисел, то есть $max(\lfloor x/w \rfloor \cdot \lfloor y/h \rfloor, \lfloor x/h \rfloor \cdot \lfloor y/w \rfloor)$.

Задача B. Баскетбольный турнир

Стандартный ввод.

Стандартный вывод.

1 секунда.

256 мегабайт

В турнире по баскетболу участвовало *n* команд, Арсений знает сколько побед одержала каждая команда, но он не знает формат чемпионата. У Арсения есть два предположения как проходили соревнования. Команды играли круговую систему или олимпийскую.

В круговой системе каждая команда играла с каждой, а в олимпийской участник выбывает из турнира после первого же проигрыша, по итогам одной игры или серии из нескольких игр между двумя участниками, позволяющей однозначно определить безусловного победителя.

Надо сказать какой формат был.

Формат входных данных

В первой строке входного файла записано целое число *n* ($3 \le n \le 10^5$) – количество команд.

В каждой из следующих *n* строк находится название команды и число побед.

Гарантируется, что не было матча закончившиеся ничьей и сумма длин названий команд не больше $3*10^5$.

Формат выходных данных

Выведите Round-robin если команды играли круговой турнир, иначе Olympic.

Система начисления баллов

В задаче 20 тестов, каждый тест оценивается в 5 баллов.

Решение:

Постановка задачи: надо научиться разделять две ситуации.

Полное решение:

- Если команды играли круговой турнир, то общее количество матчей будет равно $\frac{n\cdot(n-1)}{2}$. Соответственно, суммарное количество побед будет $\frac{n\cdot(n-1)}{2}$.
- Если команды играли олимпийскую систему, то общее количество матчей будет равно $n - 1$. Соответственно, суммарное количество побед будет $n - 1$.

Посчитаем суммарное количество побед. Если оно будет равно $\frac{n\cdot(n-1)}{2}$, то ответом будет Round-robin, иначе Olympic.

Задача C. Лифт

Стандартный ввод.

Стандартный вывод.

1 секунда.

256 мегабайт

Тимур приехал на финал олимпиады IO в университет Иннополис. Ему нужно подняться на этаж под номером *n*, войти в лифт он может в только на первом этаже. В здании университета Иннополис есть лифты двух типов:

- Лифт останавливается только на четных этажах, не считая первый этаж
- Лифт останавливается только на нечетных этажах

Каждый лифт вмещает *c* людей.

Всего *k* лифтов, которые изначально находятся на первом этаже. В каждый лифт есть уже выстроившаяся очередь людей, для каждого из которых мы знаем, на какой этаж ему нужно. В каждый лифт заходит *c* человек (если очередь длины меньше *c*, то заходит вся оставшаяся очередь). Далее лифт едет с первого этажа до самого верхнего этажа, на который нужно кому-либо из присутствующих в данный момент в лифте, останавливаясь на всех промежуточных этажах, на которых кому-либо из находящихся в лифте нужно выйти.

После того как из лифта вышли все пассажиры, лифт спускается на первый этаж, нигде не останавливаясь. На первом этаже люди заходят в лифт моментально.

Тимур может выбрать лифт и встать последним в очередь к этому лифту.

У Тимура есть 4 стратегии:

1. Пойти пешком на этаж *n*.
2. Встать в очередь на лифт, который едет до этажа *n*, и доехать до него.
3. Встать в очередь на лифт, который едет до этажа *n - 1*, и доехать до него и подняться на один этаж вверх.
4. Встать в очередь на лифт, который едет до этажа *n + 1*, и доехать до него и спуститься на один этаж вниз.

Лифт перемещается на один этаж за *lift_time* секунд. Если происходит остановка на этаже, то она длится *h* секунд (за это время выходят из лифта все люди, которые хотели выйти на данном этаже).

Тимур может подниматься на один этаж выше за *timur_up_time* секунд или спускается на один этаж ниже за *timur_down_time* секунд.

Выведите минимальное время, которое нужно Тимуру, чтобы добраться до этажа *n*.

Формат входных данных

В первой строке входного файла записаны через пробел два целых числа k и n ($1 \le k \le 10^5$) – количество лифтов и нужный этаж Тимуру.

Во второй строке входного файла записано одно число c ($1 \le c \le 10^9$) – вместительность лифтов.

В третьей строке входного файла записано два целых числа $lift\_time$ и h ($0 \le lift\_time, h \le 10^9$) – время, за которое лифт поднимается или спускается на один этаж и время, за которое люди выходят из лифта.

В четвертой строке входного файла записано два целых числа $timur\_up\_time$ и $timur\_down\_time$ ($0 \le timur\_up\_time, timur\_down\_time \le 10^9$) – время, за которое Тимур спускается и поднимается на один этаж.

В пятой строке входного файла записано k целых чисел $lift\_type_i$ ($lift\_type_i \in \{1, 2\}$) – тип каждого лифта.

В каждой из следующих k строках содержатся описание очередей. Каждая очередь содержится в отдельной строке.

Каждая строка начинается с целого числа cnt_i ($0 \le cnt_i \le 10^5$) – количество людей в очереди в i лифт. Далее в строке идут cnt_i целых положительных чисел $f_1, f_2, \ldots, f_{cnti}$ ($2 \le f_j \le 10^8$) – f_j номер этажа нужного j человеку в очереди.

Гарантируется, что суммарное количество людей не более 10^5, и людям, стоящим в очереди в определенный лифт, нужен этаж такой же четности, что и сам лифт.

Формат выходных данных

Выведите минимальное время, которое нужно Тимуру, чтобы добраться до этажа n.

Система начисления баллов

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи	Информация о проверке
0	0	Тесты из условия	–	полная
1	15	$k = 1$	–	первая ошибка
2	15	$k = 2$	–	первая ошибка
3	15	$h = 0$	–	первая ошибка

4	15	*timur_up_time* = *timur_down_time*	–	первая ошибка
5	10	$c = 1$	–	первая ошибка
6	30	Нет дополнительных ограничений	1–5	первая ошибка

Решение:

Данная задача связана с моделированием и реализацией функционирования лифтов.

Необходимо провести моделирование работы каждого лифта, разберем основные моменты в реализации:

1. Предположим, что лифт заполняется полностью людьми, которым необходимы этажи с номерами $a_1, \ldots, a_c$, и Тимура в лифте нет. Тогда время развозки людей представляет собой сумму времени, затраченного на остановки на каждом этаже, поездку до максимального этажа и спуск до первого этажа.

а) Общее количество различных остановок равно количеству уникальных чисел среди $a_1, \ldots, a_c$, обозначим это как *cnt_dif f*. Тогда время, затраченное на остановки на этажах, составляет $h \cdot$ *cnt_dif f*.

b) Время, затраченное на поездку до максимального этажа, равно *lif t_time* $\cdot (\max(a_1, \ldots, a_c) - 1)$.

c) Время, затраченное на спуск до первого этажа, также равно *lif t_time* $\cdot (\max(a_1, \ldots, a_c) - 1)$.

В итоге, время развозки людей равно $h \cdot$*cnt_dif f* + 2*lif t_time* $\cdot (\max(a_1, \ldots, a_c) - 1)$.

2. Предположим, что лифт заполняется людьми, которым нужны этажи с номерами $a_1, \ldots, a_t, n$, и в лифте присутствует Тимур (которому нужен этаж с номером n).

а) Если лифт посещает этаж n, то время на поездку до этого этажа будет равно сумме времени, затраченного на остановки на этажах до n и времени поездки до n. Количество остановок до этажа n определяется как количество уникальных чисел среди $a_1, \ldots, a_t$, меньших n, обозначим это число как q. Время, затраченное на поездку до этажа n, равно $(n - 1) \cdot$ *lif t_time*. Таким образом, время развозки людей составляет $(q + 1) \cdot h + (n - 1) \cdot$ *lif t_time* (к q прибавили 1, так как Тимур выходит на n этаже).

b) Если лифт не посещает этаж n, то запустим алгоритм из пункта a) для этажей *n* – 1 и *n* + 1 соответственно. В первом случае добавим к ответу *timur_up_time*, во втором случае добавим к ответу *timur_down_time*.
Необходимо также учесть случай, когда Тимур поднимается на n этаж пешком.

Задача D. Треугольники

Стандартный ввод.
Стандартный вывод.
1 секунда.
256 мегабайт

Сегодня на уроке математики учитель дал классу следующую задачу.
Дано пять палочек разных длин: 1, 2, 3, 4, 5. Найдите количество способов выбрать из них три палочки так, чтобы из них можно было составить треугольник.
Вам эта задача показалась тривиальной, поэтому вы решили её обобщить: дано *n* ($1 \le n \le 10^9$) палочек разных длин 1, 2, …, *n*. Требуется найти количество способов составить из них треугольник. Так как ответ может быть очень большим, требуется вывести его по модулю 10^9+7.

Формат входных данных

В единственной вводится натуральное число *n* ($1 \le n \le 10^9$) – количество палочек.

Формат выходных данных

В единственной строке выведите количество способов составить треугольник по модулю 10^9+7.

Система начисления баллов

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи и необходимых подзадач успешно пройдены.

Подзадача	Баллы	Ограничения	Необходимые подзадачи
0	0	Тесты из условия	–
1	10	$n \le 200$	–
2	20	$n \le 2000$	1
3	30	$n \le 2*10^5$	1–2
4	40	Нет	1–3

Решение:

Для решения подзадачи, где $n \le 200$, достаточно перебрать все тройки чисел $1 \le a < b < c \le n$, и для каждой тройки проверить, что $a + b > c$. Такие решение набирает 10 баллов.

Чтобы ускорить решение, заметим, что при фиксированных $1 \le a < b \le n$ нам подходят все c из отрезка $[b+1, \min(n, a+b-1)]$. Значит, достаточно перебрать a, b и для каждой пары прибавить к ответу количество подходящих c, то есть $\max(0, \min(n, a+b-1)-b)$.

Для дальнейшей оптимизации заметим, что при фиксированном a у нас есть несколько интересных отрезков b. Первый из них соответствует решению неравенства $a + b - 1 \le n \Leftrightarrow b \le n - a + 1$. Второй $- a + b - 1 > n \Leftrightarrow b > n - a + 1$. Не будем забывать, что $a + 1 \le b \le n$, то есть на самом деле отрезки выглядят так: $b \in [a+1, n-a+1]$ и $b \in [\max(a+1, n-a+2), n]$. Для каждого b из первого отрезка нужно прибавить к ответу $a-1$. А для каждого b из второго отрезка значение $n-b$.

Теперь заметим, что при $a \le \frac{n}{2}$ первый отрезок остается $b \in [a+1, n-a+1]$, а второй $[n-a+2, n]$. Для начала рассмотрим этот случай. Тогда для фиксированного a нужно прибавить к ответу

$(n-2\cdot a+1)\cdot(a-1) + \sum_{n-a+2\le b\le n}[n-b]=(n-2\cdot a+1)\cdot(a-1)+(a-1)\cdot n-\sum_{n-a+2\le b\le n} b$.

Обозначим $f(i,j)$ сумму натуральных чисел от i до j включительно. Тогда для каждого $1 \le a \le \frac{n}{2}$ нужно прибавить к ответу $(2\cdot n-2\cdot a+1)\cdot(a-1)-f(n-a+2, n)$. Для случая, когда $a > \frac{n}{2}$, первый отрезок вырождается в пустой, в остальном случай разбирается аналогично.

Теперь мы получили выражение, зависящее только от a. Нетрудно видеть, что при раскрытии скобок полученное значение можно выразить через f, а также сумму $\sum_{1\le i\le k} i^2 = \frac{k\cdot(k+1)\cdot(2\cdot k+1)}{6}$. В итоге получается следующая формула $\frac{n\cdot(n+2)\cdot(2\cdot n-5)}{24}$.

Чтобы посчитать значение по модулю $10^9 + 7$ в языках без встроенной поддержки длинной арифметики, можно использовать расширения некоторых компиляторов, например __int128_t в g + +.