XXIV Межрегиональная олимпиада по программированию
Вологодский государственный университет
16 апреля 2022

Разбор задач

Контакты:
Андрианов Игорь Александрович, к.т.н., доцент ВоГУ
igand@mail.ru
vk.com/andrianov.igor
Telegram: @igand

Задача A. Последовательность

Достаточно заметить, что при больших n (а конкретно, при n ≥ 15) ответ всегда равен 153.
Поэтому можно остановить цикл, как только получили 153.
Источник: Ж. Арсак «Программирование игр и головоломок», 1990 г.

Пример решения на Python:

```
a = int(input())
n = int(input())
for i in range(2, n + 1):
    s = 0
    while a > 0:
        s += (a % 10) ** 3
        a //= 10
    a = s
    if a == 153:
        break
print(a)
```

Задача B. Треугольники - 1

В качестве ответа подойдут, например, числа Фибоначчи:
1 1 2 3 5 8 13 …

Пример решения на Python:

```
n = int(input())
a = [1, 1]
for i in range(2, n):
   a.append(a[-1] + a[-2])
print(*a)
```

Задача C. Треугольники - 2

Сначала отсортируем входные числа.

Утверждение: ответ можно искать в виде непрерывного отрезка данного массива.
Доказательство. Допустим, оптимальное решение выглядит так (звездочки – взятые числа, точки – пропущенные):

.**..*..*..

Все звёздочки можно сдвинуть вправо (к крайней правой), и решение останется корректным: .....****..
Почему это верно? В решении каждое число меньше суммы любых двух других. Возьмём предпоследнюю звездочку и сдвинем к последней:

.**..*..*.. → .**....**..

Все неравенства остаются верны: в частности, сдвинутая звездочка не может стать ≥ сумме никаких двух предыдущих, так как для последней звездочки верно неравенство ‘ ≤ ’ с любыми двумя другими. Далее аналогично.

Решение 1.
Можно использовать метод двух указателей и получить линейный алгоритм.
Левый указатель идёт по левому концу отрезка, правый – по правому.
Пока число на правом конце отрезка меньше суммы первых двух чисел на левом конце, двигаем правый указатель.
Когда это перестало выполняться – двигаем левый указатель.

Решение 2.
Перебираем один конец отрезка, второй получаем двоим поиском. Сложность – O(n·log(n)).

Задача C. Треугольники – 2 (продолжение)

Пример решения методом двух указателей на Python:

```
n = int(input())
a = list(map(int, input().split()))
a.sort()
ans = 0
i = 0
for j in range(2, n):
    while i + 1 < j and a[j] >= a[i] + a[i + 1]:
        i += 1
    if i + 1 < j:
        ans = max(ans, j - i + 1)
print(ans)
```

Задача D. Девять шаров

Наиболее компактный способ расположить шары: 8 шаров по углам коробки и один в центре.

Пусть m – длина стороны коробки. Сделаем началом координат один из её углов.
Центр центрального шара имеет координаты (m/2, m/2, m/2).
Найдём координаты центра углового шара, ближайшего к началу координат. Для этого проведем из центра коробки в направлении к точке (0, 0, 0) вектор длиной 2R. Это можно сделать так:
а). единичный вектор с направлением «центр - начало координат»: (-1/√3, -1/√3, -1/√3).
б). вектор длины 2R в том же направлении: (-2R/√3, -2R/√3, -2R/√3).
Конечная точка вектора, если провести его из центра коробки:

$$(m/2 - 2R/\sqrt{3}, m/2 - 2R/\sqrt{3}, m/2 - 2R/\sqrt{3}).$$

Итак, мы получили координаты центра углового шара. Чтобы он касался сторон коробки, его координаты должны быть равны R, то есть $R = m/2 - 2R/\sqrt{3}$. Отсюда:

$$m = 2 \cdot R \cdot (1 + 2/\sqrt{3})$$

Пример однострочного решения на Python:

```
print(2 * int(input()) * (1 + 2 / 3 ** 0.5))
```

Задача E. Калькулятор

Будем решать задачу в обратную сторону – из N получать единицу.
Утверждение. Если число N – точный квадрат, то из него выгоднее извлечь корень, чем вычесть единицу.
Доказательство. Допустим, мы вычли единицу. Тогда мы и дальше вынуждены вычитать единицу, пока не дойдём как минимум до следующего точного квадрата – а это число $(\sqrt{N}-1)^2 = N+1-2\sqrt{N}$. Итого $2\sqrt{N} - 1$ операций.
А если сразу извлечь корень, то получим число $\sqrt{N}$. Даже если дальше делать одни вычитания, то потребуется всего $\sqrt{N}-1$ операций для получения единицы, что выгоднее.

Решение. Пока N > 1, делаем следующее:
- Находим максимальное x, такое что $x^2 \le N$.
- Добавляем к ответу $N - x^2$ (сколько сделать вычитаний до ближайшего полного квадрата)
- Если x > 1, то добавляем к ответу 1 (операция извлечения корня)
- Присваиваем N = x

В конце добавляем к ответу 1, так как нужно еще из единицы получить 0.

Пример решения на Python:

```
n = int(input())
ans = 0
while n > 1:
  x = int(n**0.5)
  if (x + 1)**2 <= n: x += 1
  if x * x > n: x -= 1
  ans += n - x**2
  if x > 1: ans += 1
  n = x
print(ans + 1)
```

Задача F. Сортировка

Вычислим p[i][j] – количество позиций, на которых лежит карточка i, а должна лежать карточка j (i, j от 1 до 3, i ≠ j).

Сравним p[1][2] и p[2][1]. Минимум из них определяет, сколько мы можем сделать таких замен, при которых сразу две неправильно лежащих карточки встают на правильные места. Тогда:
m = min(p[1][2], p[2][1])
ответ += m, p[1][2] -= m, p[2][1] -= m (при этом один из них p[1][2] и p[2][1] обнулится)
Аналогично сделаем для p[1][3] и p[3][1].
Аналогично сделаем для p[2][3] и p[3][2].

Если ещё не всё отсортировано, то возможны только два варианта:
либо p[1][2] == p[2][3] == p[3][1] ≠ 0, а остальные p – нули,
либо p[2][1] == p[1][3] == p[3][2] ≠ 0, а остальные p - нули
Теперь можно брать тройки карточек, и класть каждую тройку на свои места за два перекладывания.
ответ += 2 · max(p[1][2], p[2][1])

Набросок доказательства. Построим граф, где дуга u → v означает, что карточка в позиции u лежит неправильно, но её можно переложить в позицию v.
Построим в этом графе непересекающиеся циклы так, чтобы каждая вершина входила в какой-то цикл. Каждый такой цикл задаёт последовательность обменов. Если в цикле m вершин, то всего будет (m-1) обмен.
Заметим, что выгодно делать как можно более короткие циклы. Например, пусть у нас 6 вершин. Если взять 2 цикла длины 3, то будет 4 обмена, а если 3 цикла длины 2 – всего 3 обмена.
В описанном выше решении как раз, по сути, сначала строится максимальное количество циклов длины 2, а затем из оставшихся вершин строятся циклы длины 3.

Задача F. Сортировка (продолжение)

Пример решения на Python:

```
n = int(input())
a = list(map(int, input().split()))
b = sorted(a)
p = [[0, 0, 0, 0] for i in range(4)]
for i in range(len(a)):
  p[a[i]][b[i]] += 1
ans = 0
for i in range(3):
  for j in range(i + 1, 4):
    m = min(p[i][j], p[j][i])
    ans += m
    p[i][j] -= m
    p[j][i] -= m
ans += 2 * max(p[1][2], p[2][1])
print(ans)
```

Задача G. Проверка на дорогах - 1

Решение 1. Находим в графе мосты. Находим любой путь от a до b. Считаем, сколько мостов лежат на этом пути.

Решение 2. Находим в графе количество мостов. Добавляем в граф новую вершину c и рёбра a-c и b-c. Снова считаем количество мостов. Разность этих двух чисел – ответ. Промежуточная вершина нужна, чтобы не возникло кратных рёбер между a и b. Пример решения этим способом на C++:

```
#include <bits/stdc++.h>
using namespace std;

int n, m, a, b, timer;
vector<vector<int> > g;
vector<int> tin, fup;

int dfs(int u, int parent) {
    int bridges = 0;
    tin[u] = ++timer;
    fup[u] = tin[u];
    for (int v : g[u]) {
      if (tin[v] == 0) {
        bridges += dfs(v, u);
        if (fup[v] > tin[u])
          bridges++;
        fup[u] = min(fup[u], fup[v]);
      } else if (v != parent) {
        fup[u] = min(fup[u], tin[v]);
      }
    }
    return bridges;
}
```

```
int main(int argc, char * argv[]){
    cin >> n >> m >> a >> b;
    g.resize(n + 1);
    for (int i = 0; i < m; i++){
      int u, v;
      cin >> u >> v;
      g[u].push_back(v);
      g[v].push_back(u);
    }
    tin.assign(n + 1, 0);
    fup.assign(n + 1, 0);
    int bridges1 = dfs(a, -1);
    g[a].push_back(0); g[0].push_back(a);
    g[b].push_back(0); g[0].push_back(b);
    tin.assign(n + 1, 0);
    fup.assign(n + 1, 0);
    int bridges2 = dfs(a, -1);
    cout << bridges1 - bridges2;
}
```

Задача H. Проверка на дорогах – 2

1. Найдём алгоритмом Дейкстры длины кратчайших путей от вершины a до всех остальных (массив cost_a).

2. Найдём алгоритмом Дейкстры длины кратчайших путей от вершины b до всех остальных (массив cost_b).

3. Переберём все рёбра графа и проверим, лежит ли каждое ребро на каком-то кратчайшем пути.
Ребро u-v лежит на кратчайшем пути, если cost_a[u] + длина_ребра_uv + cost_b[v] == cost_a[b].
Из таких рёбер строим новый граф.

4. В полученном графе ищем мосты. Их количество – это ответ.

Задача H. Проверка на дорогах – 2 (продолжение). Пример решения на C++:

```
#include <bits/stdc++.h>
using namespace std;

int n, m, a, b;
long long Inf = 1e18;
vector<vector<pair<int, int> > > g;
vector<vector<int> > gr;

vector<long long> dijkstra(int start) {
  vector<long long> d(n + 1, Inf);
  set<pair<long long, int> > cost;
  d[start] = 0;
  for (int i = 1; i <= n; i++)
    cost.insert({d[i], i});
  for (int i = 0; i < n; i++) {
    auto p = *cost.begin();
    cost.erase(cost.begin());
    int u = p.second;
    for (auto q : g[u]) {
      int v = q.first, uv_len = q.second;
      if (d[u] + uv_len < d[v]) {
        cost.erase({d[v], v});
        d[v] = d[u] + uv_len;
        cost.insert({d[v], v});
      }
    }
  }
  return d;
}

int timer;
vector<int> tin, fup;

int dfs(int u, int parent) {
  int bridges = 0;
  tin[u] = ++timer;
  fup[u] = tin[u];
  for (int v : gr[u]) {
    if (tin[v] == 0) {
      bridges += dfs(v, u);
      if (fup[v] > tin[u]) {
        bridges++;
      }
      fup[u] = min(fup[u], fup[v]);
    } else if (v != parent) {
      fup[u] = min(fup[u], tin[v]);
    }
  }
  return bridges;
}

int main() {
  cin >> n >> m >> a >> b;
  g.resize(n + 1);
  for (int i = 0; i < m; i++) {
    int u, v, len;
    cin >> u >> v >> len;
    g[u].push_back({v, len});
    g[v].push_back({u, len});
  }
  vector<long long> cost_a = dijkstra(a);
  vector<long long> cost_b = dijkstra(b);
  gr.resize(n + 1);
  for (int u = 1; u <= n; u++) {
    for (auto p : g[u]) {
      int v = p.first, len = p.second;
      if (cost_a[u] + len + cost_b[v] == cost_a[b]) {
        gr[u].push_back(v);
        gr[v].push_back(u);
      }
    }
  }
  tin.assign(n + 1, 0);
  fup.assign(n + 1, 0);
  cout << dfs(a, -1) << endl;
}
```

Задача I. Прогулки и сражения

Перейдем к новой системе координат: координаты (x, y) преобразуются в (x’=x – y, y’=x + y).

В старой системе расстояние между двумя клетками (x1, y1) и (x2, y2) равно |x1-x2|+|y1-y2|. А в новой системе расстояние можно найти так:

$$\max(|x1'-x2'|, |y1'-y2'|)$$

Доказательство. Заменим переменные со штрихом на определения через старые координаты:
max(|x1-y1-(x2-y2)|, |x1+y1-(x2+y2)|).
Перегруппируем слагаемые:
max(|x1-x2 + y2-y1|, |x1-x2 + y1-y2|)
Теперь несложно видеть, что при любом взаимном расположении клеток формула даёт расстояние между ними.

Заметим, что в формуле max(|x1’-x2’|, |y1’-y2’|) можно отдельно искать максимум по x’ и по y’, а потом взять наибольший из них. Отсюда получаем решение:
1. Для каждого игрока найдём минимальный и максимальный x’ и y’, то же самое сделаем для каждого монстра.
2. Переберем все пары: min_x’ игрока и max_x’ монстра, max_x’ игрока и min_x’ монстра, аналогично для y’.
Среди всех пар выберем наилучшую.

Идея отсюда: Антти Лааксонен «Олимпиадное программирование», второе издание, 2020. Глава 13 - “Геометрия”, подраздел «Метрики».

Задача I. Прогулки и сражения (продолжение)

Пример решения на Python:

```
Inf = 2_000_000_001

n, m = map(int, input().split())

def getminmax(n):
  min_x, min_y, max_x, max_y = Inf, Inf, 0, 0
  for i in range(n):
    x, y = map(int, input().split())
    min_x = min(min_x, x - y)
    min_y = min(min_y, x + y)
    max_x = max(max_x, x - y)
    max_y = max(max_y, x + y)
  return min_x, min_y, max_x, max_y

char_min_x, char_min_y, char_max_x, char_max_y = getminmax(n)
mons_min_x, mons_min_y, mons_max_x, mons_max_y = getminmax(m)

print(max(char_max_x - mons_min_x, mons_max_x - char_min_x, \
        char_max_y - mons_min_y, mons_max_y - char_min_y))
```

Задача J. Призовая игра

Будем подбирать ответ k, начиная с 2, и далее по возрастанию. Взяв очередное значение, проверим, хватит ли этого числа попыток.
Закодируем каждую карточку двоичным кодом длины k, в котором будет ровно k div 2 единиц. Таких кодов можно построить C(k, k div 2) – соответственно, это максимальное число карточек для данного k. При i-й попытке будем называть номера тех карточек, в кодах которых в i-й позиции стоит 1.
Пример. Пусть у нас 5 карточек, k=4. Возможные коды:
1) 0011
2) 0101
3) 1001
4) 0110
5) 1010
При первой попытке назовем номера 3 и 5 (если позиции в кодах нумеровать слева направо), при второй – 2 и 4, затем 1, 4, 5, и в последней попытке – 1, 2, 3.

Докажем, что при таком подходе хотя бы в одной из попыток мы выберем карточку с «призом» и не выберем с «отменой». Пусть карточка u содержит «приз», карточка v – «отмену». Предположим, нет такой позиции i, что код[u][i]=1, код[u][j]=0. Это значит, что во всех позициях, где в коде карточки u стоит единица, в коде карточки v тоже стоит единица. Но все коды содержат одинаковое число единиц, поэтому два этих кода совпадают, а этого быть не может.

Задача J. Призовая игра (продолжение)

Итак, k – минимальное натуральное число, что C(k, k div 2) ≥ n. Докажем, что при меньших k решения нет.

Допустим, что существует решение при меньшем k. Закодируем карточки двоичным кодом длины k, где 1 в i-й позиции говорит о том, что эту карточку будем называть при i-й попытке. Каких-то ограничений на код нет (в отличие от предыдущего слайда), поэтому так можно закодировать любую стратегию.

Любая пара карточек (u, v) может содержать слова “приз” и “отмена”. Поэтому должна найтись позиция i, что:

код[u][i]=1, код[v][i]=0

Но слова могут располагаться и наоборот, поэтому также должна найтись позиция j , что:

код[u][j]=1, код[v][j]=0

То есть, любые два кода должны выглядеть как-то так:

.....1....0....

.....0....1....

Заметим, что двоичные коды длины k можно рассматривать как подмножества некоторого множества из k элементов (где 1 означает, что элемент берем, 0 – что не берем). Тогда из предыдущих утверждений следует: **никакое подмножество не должно содержаться ни в каком другом.**

А сколько максимум таких подмножеств можно построить, что никакое не содержится ни в каком другом? **Теорема Шпернера** утверждает, что не более C(k, k div 2). Поэтому, если C(k, k div 2) < n, то решения нет.

Источник: https://problems.ru/view_problem_details_new.php?id=66096

Задача J. Призовая игра (продолжение)

Пример решения на Python:

```
from math import factorial
n = int(input())
for k in range(1, n + 1):
    c = factorial(k) / (factorial(k // 2) * factorial(k - k // 2))
    if c >= n:
        print(k)
        break
```