BFS, Дейкстра

1 A. Наименьшее кратное

Разбор

Напишем поиск в ширину по неявному графу состояний: вершинами будут числа, ориентированное ребро между вершинами будет обозначать возможные преобразования. Будем складывать в очередь строчки с числами as is, вычисляя онлайн остаток от деления нашей строки x с добавленными к ней цифрами на k. Заметим, что если мы припишем к строке x какую-нибудь цифру и возьмём её остаток от деления на k, то получим такое же значение, как если бы мы приписали цифру к остатку от деления x без этой цифры на k:

$$17284128 \equiv 142 \pmod{673}$$

$$172841283 \equiv 77 \pmod{673}$$

$$1423 \equiv 77 \pmod{673}$$

Значит, мы можем посчитать остаток от длинной строки всего один раз, а дальше работать с числами напрямую. Дальше остаётся корректно реализовать обработку циклов (сколько всего можем получить различных остатков от деления на k?) и восстановление ответа.

Асимптотика решения: $O(K \times D)$

Решение

```
from collections import deque
from sys import stdin

input = stdin.readline

x, k = input().split()
k = int(k)
_ = input()
d = sorted(list(map(int, input().split())))

def solve(a: str, b: int, digits: list):
    r = 0
    for c in a:
        r = (r * 10 + int(c)) % b
    q = deque()
    q.append(r)
    used, to = [0] * b, [None] * b
    used[r] = 1
    while q:
        v = q.popleft()
        if v == 0:
            res = ''
            while to[v]:
                v, digit = to[v]
                res += str(digit)
            return a + res[::-1]
        for digit in digits:
            w = (v * 10 + digit) % b
            if not used[w]:
                used[w] = 1
                to[w] = (v, digit)
                q.append(w)
    return -1

print(solve(x, k, d))
```

2 B. Дейкстра

Разбор

В этой задаче требовалось реализовать поиск кратчайшего пути алгоритмом Дейкстры. Из-за ограничений на время работы можно было воспользоваться линейностью данных в этой задаче и реализовать жадность не через сортировку/кучу, а через поиск минимума с ключом.

Асимптотика решения: $O(|V| + |E|)$

Решение

```
INF = 10 ** 9 + 7
fin, fout = open('dijkstra.in'), open('dijkstra.out', 'w')

n, s, f = map(int, fin.readline().split())
g = [r for r in fin.readlines()]
s, f = s - 1, f - 1
d = [INF] * n
d[s] = 0
q = set(list(range(n)))
while q:
    node = min(q, key=lambda v: d[v])
    path_weight = d[node]
    if node == f or path_weight == INF:
        break
    q.remove(node)
    row = list(map(int, g[node].split()))
    for to in range(n):
        edge_weight = row[to]
        if edge_weight < 0:
            continue
        d[to] = min(d[to], path_weight + edge_weight)
print(-1 if d[f] == INF else d[f], file=fout)
```

3 C. Автобусы

Разбор

Реализуем поиск в ширину. Учтём, что нельзя сесть на автобус раньше, чем приехать в деревню.

Асимптотика решения: $O(|V| + |E|)$

Решение

```
from collections import defaultdict
from sys import stdin

input = stdin.readline
INF = 10 ** 9 + 7

n = int(input())
d, v = map(int, input().split())
r = int(input())
g = defaultdict(lambda: defaultdict(list))
for _ in range(r):
    from_, time_from, to_, time_to = map(int, input().split())
    g[from_ - 1][to_ - 1].append((time_from, time_to))

d, v = d - 1, v - 1
t = [INF] * n
t[d] = 0
q = set(list(range(n)))
while q:
    node = min(q, key=lambda v: t[v])
    if node == v or t[node] == INF:
```

```
        break
    q.remove(node)
    path_time = t[node]
    for to in g[node]:
        for time_from, time_to in g[node][to]:
            if time_from >= path_time:
                t[to] = min(t[to], time_to)
print(-1 if t[v] == INF else t[v])
```

4 D. Числа

Разбор

Решим эту задачу также при помощи поиска в ширину, дополнительно научимся поддерживать список предков, чтобы восстанавливать ответ.

Асимптотика решения: $O(|V| + |E|)$

Решение

```
from collections import deque

INF = 10 ** 9 + 7

def inc_one(v: int):
    if v // 1000 < 9:
        v += 1000
    return v

def dec_one(v: int):
    if v % 10 > 1:
        v -= 1
    return v

def shift_r(v: int):
    return (v % 1000) * 10 + (v // 1000)

def shift_l(v: int):
    return (v % 10) * 1000 + (v // 10)

a = int(input())
b = int(input())
d = [INF] * 10000
f = [INF] * 10000
d[a] = 0
f[a] = -1
q = deque()
q.append(a)
while q:
    n = q.popleft()
    if n == b:
        break
    to = [inc_one(n), dec_one(n), shift_l(n), shift_r(n)]
    for t in to:
        if d[t] == INF:
            q.append(t)
        if d[t] > d[n] + 1:
            d[t] = d[n]
            f[t] = n
ans = [b]
v = f[b]
while v > 0:
    ans.append(v)
    v = f[v]
for v in reversed(ans):
    print(v)
```

5 E. Портал

Разбор

Будем решать задачу поиска кратчайшего пути с небольшой модификацией. Учтём, что мы можем за раз поставить 2 портала, произвольно выбрав два из четырёх возможных направлений. При этом в любом из порталов мы окажемся за время пути до **ближайшей** стены. Тогда просто добавим точки, в которые мы можем попасть через эти порталы, в очередь. Когда мы до них дойдём – мы сможем проверить, сможем ли мы через них добраться до конечной точки быстрее, чем по другим маршрутам.

Асимптотика решения: $O(|V|)$

Решение

```
from sys import stdin

input = stdin.readline
INF = 10 ** 9 + 7
DIRS = [(-1, 0), (1, 0), (0, -1), (0, 1)]

n, m = map(int, input().split())
g = [input() for _ in range(n)]
s, f = None, None
for i in range(n):
    for j in range(m):
        if g[i][j] == 'S':
            s = (j, i)
        if g[i][j] == 'T':
            f = (j, i)

def get_portals(x_start: int, y_start: int) -> tuple:
    res = []
    for i in range(y_start, n):
        if g[i][x_start] == '#':
            res.append((x_start, i - 1))
            break
    for i in range(x_start, m):
        if g[y_start][i] == '#':
            res.append((i - 1, y_start))
            break
    for i in range(y_start, -1, -1):
        if g[i][x_start] == '#':
            res.append((x_start, i + 1))
            break
    for i in range(x_start, -1, -1):
        if g[y_start][i] == '#':
            res.append((i + 1, y_start))
            break
    return res

q = set([s])
d = [[INF] * m for _ in range(n)]
d[s[1]][s[0]] = 0
while q:
    x, y = min(q, key=lambda v: d[v[1]][v[0]])
    q.remove((x, y))
    if (x, y) == f:
        break
    for d_x, d_y in DIRS:
        x_new, y_new = x + d_x, y + d_y
        if g[y_new][x_new] == '#':
            continue
        if d[y_new][x_new] == INF:
            q.add((x_new, y_new))
        d[y_new][x_new] = min(d[y_new][x_new], d[y][x] + 1)
    portals = get_portals(x, y)
    min_d = min(map(lambda v: abs(v[0] - x) + abs(v[1] - y) + 1, portals))
    for p in portals:
        x_new, y_new = p
```

```
        if d[y_new][x_new] == INF:
            q.add(p)
        d[y_new][x_new] = min(d[y_new][x_new], d[y][x] + min_d)

print(d[f[1]][f[0]])
```

6 F. Недалёкий Маршалл

Разбор

Поскольку нам надо двигаться по графу, соблюдая некоторые условия на вершины, в которых находятся наши герои, то построим граф состояний и будем искать в нем кратчайший путь от стартового состояния до желаемого. Состоянием мы будем называть пару из текущих вершин Маршалла и Лили. Заметим, что из каждого состояния мы можем сделать переходы трех типов:

- Ходит только Маршалл
- Ходил только Лили
- Ходят оба

Заметим, что если добавить петлю в каждой вершине, то все случаи сводятся к последнему. Когда мы определили граф состояний(строить его полностью не нужно, поскольку в нем будет много недостижимых состояний), остается только запустить в нем BFS и найти кратчайшее расстояние от стартовой конфигурации до желаемой.

Асимптотика решения: $O(V^2 + (V + E)^2)$, поскольку каждый переход в графе состояний соответствует паре переходов в исходном графе, к которым мы еще добавили V петель.

Решение

```
#include <cstdio>
#include <algorithm>
#include <iostream>
#include <vector>
#include <queue>
#include <stdlib.h>
#include <stdio.h>

using namespace std;

struct point
{
    int x, y;
    point(int _x, int _y)
    {
        x = _x;
        y = _y;
    };
    point(){};
};
point ans[100000];
int numans;
bool matrix2[801][801];
int nai[108][108];
int vis[1001][1001];
  point pre[801][801];
int main()
{
    int n, m, sx, sy, fx, fy;


    for (int i = 0; i <= 100; i++)
        for (int j = 0; j <= 100; j++)
        {
```

```
        vis[i][j] = 0;
        pre[i][j] = point(-1, -1);
        matrix2[i][j] = false;
        matrix2[i][i] = true;
        nai[i][j] = 1000000;

    }
cin >> n >> m >> sx >> sy >> fx >> fy;
vector<vector<int> > matrix(n + 10);

for (int i = 0; i <= n; i++) matrix[i].clear();
for (int i = 0; i < m; i++)
{
    int a, b;
    cin >> a >> b;
    matrix[a].push_back(b);
    matrix[b].push_back(a);
    matrix2[a][b] = true;
    matrix2[b][a] = true;
}
queue <point> q;
q.push(point(sx, sy));
vector<int> ::iterator myi;
vector<int> ::iterator myj;
nai[sx][sy] = 0;

while (!q.empty())
{
    point temp = q.front();
    q.pop();
    if (temp.x == fx && temp.y == fy) break;
    int x = temp.x;
    int y = temp.y;
    vis[x][y] = 2;

    for (myi = matrix[x].begin(); myi != matrix[x].end(); myi ++)
        for (myj = matrix[y].begin(); myj != matrix[y].end(); myj ++)
        {
            int tx = *myi;
            int ty = *myj;
            if (vis[x][ty] < 1  && matrix2[x][ty] == true && x!= ty)
           {
               vis[x][ty] = 1;
               q.push(point(x, ty));
               nai[x][ty] = nai[x][y] + 1;
               pre[x][ty] = point(x, y);
               if (x == fx && ty == fy) break;
           }
            //
           if (vis[x][ty] == 1  && matrix2[x][ty] == true && x!= ty
               && nai[x][y] +  1 < nai[x][ty])
           {
               nai[x][ty] = nai[x][y] + 1;
               pre[x][ty] = point(x, y);
           }
            //

            //
           if (vis[tx][y] < 1 && matrix2[tx][y] == true  && tx != y)
           {
               vis[tx][y] = true;
               q.push(point(tx, y));
               pre[tx][y] = point(x, y);
                nai[tx][y] = nai[x][y] + 1;
               if (tx == fx && y == fy) break;
           }
            //2
           if (vis[tx][y] == 1  && matrix2[tx][y] == true && tx!= y
               && nai[x][y] +  1 < nai[tx][y])
           {
```

```
                    nai[tx][y] = nai[x][y] + 1;
                    pre[tx][y] = point(x, y);
                }

                if (vis[tx][ty]< 1 && matrix2[tx][ty] == true  && !(tx == y && x == ty) && !(tx
     == ty))
                 {
                     vis[tx][ty] = true;
                     q.push(point(tx, ty));
                     pre[tx][ty] = point(x, y);
                     nai[tx][ty] = nai[x][y] + 2;
                     if (tx == fx && ty == fy) break;
                 }

                if (vis[tx][ty] == 1 &&  matrix2[tx][ty] == true  && !(tx == y && x == ty) &&
    !(tx == ty)
                    && nai[x][y] +  2 < nai[tx][ty])
                {
                    nai[tx][ty] = nai[x][y] + 2;
                    pre[tx][ty] = point(x, y);
                }


            }
    }

    numans = 0;
    int x = fx;
    int y = fy;
    int c = 0;
    while (x > 0 && y >0)
    {
        ans[numans] = point(x, y);
        numans ++;
        int tempx = x;
        int tempy = y;
        x = pre[tempx][tempy].x;
        y = pre[tempx][tempy].y;
        if (x > 0 && y > 0)
        {
        if (x != tempx) c++;
        if (y != tempy) c++;
        }
    }
    cout << c << " " << numans << endl;
}
```

7 G. Лабиринт

Разбор

Заметим, что если мы можем дойти до какой-то клетки, то разница количества шагов вправо и шагов влево – фиксированная. А, значит, если мы дошли до клетки, совершив минимальное число ходов влево, то количество ходов вправо тоже будет минимальным. Поэтому мы можем построить граф переходов между клетками, в котором ходы влево имеют вес 1, а все остальные имеют вес 0, а потом запустить на этом графе $0-1$ BFS. Тогда расстояния до клеток, которые найдет наш обход и будут минимальным количеством шагов влево, чтобы дойти до клетки. А количество шагов вправо мы посчитаем, зная разницу x-координат между клеткой и начальной клеткой.

Асимптотика решения: $O(V+E)$

Решение

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

```cpp
using ull = uint64_t;
using ll = int64_t;
using ld = long double;

const int MAXN = 2228;
int d[MAXN][MAXN];
bool w[MAXN][MAXN];

const int INF = 1000 * 1000 * 1000 + 228;

string s[MAXN];

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);

    int n, m, r, c, cl, cr;
    cin >> n >> m >> r >> c >> cl >> cr;
    --r, --c;
    deque<pair<int, int>> q = { {r, c} };

    for (int i = 0; i < n; ++i) {
        cin >> s[i];
        for (int j = 0; j < m; ++j) {
            d[i][j] = INF;
        }
    }

    int ans = 0;
    d[r][c] = 0;
    while (!q.empty()) {
        auto[x, y] = q.front();

        q.pop_front();

        if (s[x][y] == '*') {
            continue;
        }

        if (w[x][y]) {
            continue;
        }


        w[x][y] = true;
        int diff = y - c;
        int ll = d[x][y];
        int rr = d[x][y] + diff;
        if (ll > cl || rr > cr) {
            continue;;
        }

        ++ans;
        if (x > 0 && d[x - 1][y] > d[x][y]) {
            d[x - 1][y] = d[x][y];
            q.emplace_front(x - 1, y);
        }

        if (x + 1 < n && d[x + 1][y] > d[x][y]) {
            d[x + 1][y] = d[x][y];
            q.emplace_front(x + 1, y);
        }


        if (y + 1 < m && d[x][y + 1] > d[x][y]) {
            d[x][y + 1] = d[x][y];
            q.emplace_front(x, y + 1);
        }
```

```
            if (y > 0 && d[x][y - 1] > d[x][y] + 1) {
                d[x][y - 1] = d[x][y] + 1;
                q.emplace_back(x, y - 1);
            }
        }

        cout << ans << "\n";
    }
```