Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Алгоритмы и структуры данных

Задачи M, N, O, P (Яндекс.Контест)

Выполнил: студент группы P3208,
Васильев Н. А.

Преподаватель: Косяков М. С.

Санкт-Петербург 2025

Задача M. Цивилизация

Для задачи нахождения кратчайшего пути на сетке используем алгоритм Дейкстры, так как у нас по условию веса рёбер не одинаковые (1 или 2), нужен путь с минимальной суммарной стоимостью, граф – это сетка, где каждая клетка – вершина.

Используем `priority_queue` для эффективного выбора следующей клетки с наименьшей стоимостью перемещения, массив `distances` хранит минимальное время для достижения каждой клетки, а `parent` и `from_dir` используются для восстановления пути.

Всего у нас N * M клеток, каждая клетка может попасть в кучу до 4 раз максимум. В цикле используются операции `push` и `pop`, выполняющиеся за O(log (N * M)), тогда выполнение всего алгоритма занимает **O((N * M) * log (N * M))**.

Мы храним: `grid[n][m]` – char: 1 байт на клетку, `distances[n][m]` – int: 4 байта, `from_dir[n][m]` – char: 1 байт и `parent[n][m]` – пара int: 8 байт. Также есть `priority_queue`, где в худшем случае может содержаться до N * M элементов, каждый из которых имеет постоянный размер, таким образом получаем **O(N * M).**

Код:

```
#include <climits>
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

struct Node {
  int cost;
  int x;
  int y;

  bool operator>(const Node& other) const {
    return cost > other.cost;
  }
};

int main() {
  int n, m, x, y, x_cell, y_cell;
  cin >> n >> m >> x >> y >> x_cell >> y_cell;
  x--;
  y--;
  x_cell--;
  y_cell--;

  vector<vector<char>> grid(n, vector<char>(m));
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
      cin >> grid[i][j];
    }
  }
```

```
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};
char directions[4] = {'N', 'E', 'S', 'W'};

vector<vector<int>> distances(n, vector<int>(m, INT_MAX));
vector<vector<char>> from_dir(n, vector<char>(m, 0));
vector<vector<pair<int, int>>> parent(n, vector<pair<int, int>>(m, {-1, -1}));

priority_queue<Node, vector<Node>, greater<Node>> pq;
distances[x][y] = 0;
pq.push({0, x, y});

while (!pq.empty()) {
  Node current = pq.top();
  pq.pop();

  if (current.cost > distances[current.x][current.y])
    continue;

  for (int d = 0; d < 4; d++) {
    int nx = current.x + dx[d];
    int ny = current.y + dy[d];

    if (nx < 0 || ny < 0 || nx >= n || ny >= m) {
      continue;
    }
    if (grid[nx][ny] == '#') {
      continue;
    }

    int step_cost = (grid[nx][ny] == '.') ? 1 : 2;
    int new_cost = current.cost + step_cost;

    if (new_cost < distances[nx][ny]) {
      distances[nx][ny] = new_cost;
      pq.push({new_cost, nx, ny});
      from_dir[nx][ny] = directions[d];
      parent[nx][ny] = {current.x, current.y};
    }
  }
}

if (distances[x_cell][y_cell] == INT_MAX) {
  cout << -1 << endl;
} else {
  cout << distances[x_cell][y_cell] << endl;

  vector<char> path;
  int cx = x_cell, cy = y_cell;
  while (cx != x || cy != y) {
    char d = from_dir[cx][cy];
```

```
      path.push_back(d);
      tie(cx, cy) = parent[cx][cy];
    }

    for (int i = path.size() - 1; i >= 0; i--) {
      cout << path[i];
    }
    cout << endl;
  }

  return 0;
}
```

Задача N. Свинки-копилки

У нас есть `n` копилок, пронумерованных от 1 до n, у каждой из которых есть ключ, и этот ключ лежит в другой копилке (или в ней самой). Тогда каждая копилка – вершина неориентированного графа, если ключ от копилки `i` лежит в копилке `j`, создается ребро `ij`. Построение графа и поиск по нему занимают **O(n)** времени. Использование `keys[n]`, `graph[n]` и `used[n]` занимает **O(n)** памяти.

Код:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> keys(n);
  for (int i = 0; i < n; ++i) {
    cin >> keys[i];
    keys[i]--;
  }

  vector<vector<int>> graph(n);
  for (int i = 0; i < n; ++i) {
    graph[i].push_back(keys[i]);
    graph[keys[i]].push_back(i);
  }

  vector<bool> used(n, false);
  int counter = 0;

  for (int i = 0; i < n; ++i) {
    if (!used[i]) {
      counter++;
      vector<int> stack = {i};
      while (!stack.empty()) {
```

```
        int v = stack.back();
        stack.pop_back();

        if (used[v])
          continue;
        used[v] = true;

        for (int u : graph[v]) {
          if (!used[u]) {
            stack.push_back(u);
          }
        }
      }
    }
  }

  cout << counter << endl;
  return 0;
}
```

Задача O. Долой списывание!

У нас есть `n` лкшат, и `m` пар. Необходимо построить неориентированный граф и попытаться раскрасить его в два цвета так, чтобы соседние вершины были покрашены разными цветами, тогда можно разделить лкшатов на две группы, иначе нет. Обход графа занимает **O(n + m)** времени. Хранение графа занимает **O(n + m)** памяти.

Код:

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n, m;
  cin >> n >> m;
  vector<pair<int, int>> lkshata(m);
  for (int i = 0; i < m; ++i) {
    int u, v;
    cin >> u >> v;
    u--;
    v--;
    lkshata[i] = {u, v};
  }
  vector<vector<int>> graph(n);
  for (const auto& p : lkshata) {
    int u = p.first;
    int v = p.second;
    graph[u].push_back(v);
    graph[v].push_back(u);
```

```
  }
  vector<int> ans(n, -1);
  bool flag = true;
  for (int i = 0; i < n && flag; i++) {
    if (ans[i] == -1) {
      vector<int> stack = {i};
      ans[i] = 0;
      while (!stack.empty() && flag) {
        int v = stack.back();
        stack.pop_back();

        for (int u : graph[v]) {
          if (ans[u] == -1) {
            ans[u] = 1 - ans[v];
            stack.push_back(u);
          } else if (ans[u] == ans[v]) {
            flag = false;
            break;
          }
        }
      }
    }
  }
  cout << (flag ? "YES" : "NO") << endl;
  return 0;
}
```

Задача P. Авиаперелёты

У нас есть `n` городов, для каждой пары городов `(i, j)` задано, сколько топлива нужно, чтобы перелететь напрямую из `i` в `j`. Нам нужно найти минимальный возможный размер топливного бака, при котором можно долететь из любого города в любой другой.

Если мы можем долететь из любого города в любой другой, то граф сильно связный.

Выполним бинарный поиск по значениям топлива `fuel[i][j]`. Внутри поиска построим граф, где оставим только те рёбра, где `fuel[i][j] <= mid`. Если в обоих направлениях мы посещаем все города, значит граф сильно связный.

Бинарный поиск по сложности выполняется за O(log(p)), каждый шаг в проверке за $O(n^2)$. Тогда общая сложность **$O(n^2 * log(max\_fuel))$**. Матрица `fuel[i][j]` занимает $O(n^2)$ памяти.

Код:

```
#include <iostream>
#include <vector>

using namespace std;

vector<vector<long long>> fuel;
vector<vector<int>> graph;
vector<bool> visited;
```

```cpp
long long fuelLimit;
int n;

void dfs(int node, vector<bool>& visited, bool reverse) {
  visited[node] = true;
  for (int neighbor = 0; neighbor < n; neighbor++) {
    if (!visited[neighbor]) {
      if ((!reverse && fuel[node][neighbor] <= fuelLimit) ||
          (reverse && fuel[neighbor][node] <= fuelLimit)) {
        dfs(neighbor, visited, reverse);
      }
    }
  }
}

bool check(long long maxFuel) {
  fuelLimit = maxFuel;

  vector<bool> visited(n, false);
  dfs(0, visited, false);
  for (int i = 0; i < n; i++) {
    if (!visited[i]) {
      return false;
    }
  }

  visited.assign(n, false);
  dfs(0, visited, true);
  for (int i = 0; i < n; i++) {
    if (!visited[i]) {
      return false;
    }
  }

  return true;
}

int main() {
  cin >> n;
  fuel.assign(n, vector<long long>(n));
  long long maximum = 0;

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      cin >> fuel[i][j];
      if (fuel[i][j] > maximum) {
        maximum = fuel[i][j];
      }
    }
  }
```

```
  long long left = 0;
  long long right = maximum;
  long long result = maximum;

  while (left <= right) {
    long long mid = (left + right) / 2;

    if (check(mid)) {
      result = mid;
      right = mid - 1;
    } else {
      left = mid + 1;
    }
  }

  cout << result << endl;
  return 0;
}
```