Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3213

Поленов К.А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

Задача № N «Свинки-копилки»

```cpp
#include <cstdint>
#include <iostream>
#include <vector>

int main() {
  uint64_t n;
  std::cin >> n;

  auto* key_location = new uint64_t[n + 1];
  for (size_t i = 1; i <= n; i++) {
    uint64_t location;
    std::cin >> location;
    key_location[i] = location;
  }

  auto* globally_visited = new bool[n + 1]{false};
  uint64_t breaks = 0;

  for (uint64_t i = 1; i <= n; i++) {
    if (!globally_visited[i]) {
      auto* visited_on_current_path = new bool[n + 1]{false};
      uint64_t current_node = i;

      while (!globally_visited[current_node] && !visited_on_current_path[current_node]) {
        visited_on_current_path[current_node] = true;
        current_node = key_location[current_node];
      }

      if (!globally_visited[current_node] && visited_on_current_path[current_node]) {
        breaks++;
      }

      uint64_t visited_node = i;
      while (!globally_visited[visited_node]) {
        globally_visited[visited_node] = true;
        visited_node = key_location[visited_node];
      }

      delete[] visited_on_current_path;
    }
  }

  std::cout << breaks;
  delete[] globally_visited;
  delete[] key_location;
  return 0;
}
```

Пояснение к примененному алгоритму:

Задача сводится к поиску циклов в функциональном графе. Представим каждую копилку (и соответствующий ей ключ) как узел в графе. Если ключ от копилки i находится в копилке j, то проведем направленное ребро от узла i к узлу j. Такой граф будет состоять из одного или нескольких компонентов, где каждый компонент представляет собой набор деревьев, ведущих к одному уникальному циклу.

Чтобы получить доступ ко всем ключам в одном таком компоненте, необходимо разбить одну любую копилку, принадлежащую циклу этого компонента. Как только одна копилка в цикле разбита, мы получаем ключ, который она содержит. Этот ключ позволяет открыть другую копилку (возможно, в том же цикле или ведущую к нему), что дает нам еще больше ключей, и так далее, пока все копилки в этом компоненте не будут открыты.

Следовательно, минимальное количество копилок, которые нужно разбить, равно количеству таких независимых циклических компонентов в графе.

Задача № O «Долой списывание»

```cpp
#include <cstdint>
#include <iostream>
#include <queue>
#include <vector>

bool IsBipartite(uint64_t n, const std::vector<std::vector<int64_t>>& adj) {
  std::vector<int64_t> color(n + 1, 0);  // 0: не посещен, 1: цвет 1, 2: цвет 2
  std::queue<int64_t> q;

  for (uint64_t start_node = 1; start_node <= n; ++start_node) {
    if (color[start_node] == 0) {
      color[start_node] = 1;
      q.push(start_node);

      while (!q.empty()) {
        int64_t u = q.front();
        q.pop();

        for (int64_t v : adj[u]) {
          if (color[v] == 0) {
            color[v] = 3 - color[u];  // Раскрашиваем соседний узел в противоположный цве
            q.push(v);
          } else if (color[v] == color[u]) {
            return false;  // Найдено ребро между узлами одного цвета, не двудольный
          }
        }
      }
    }
  }
  return true;  // Все компоненты связности двудольные
}
int main() {
  uint64_t n, m;
  std::cin >> n >> m;

  auto* adj = new std::vector<int64_t>[n + 1];

  for (uint64_t i = 0; i < m; ++i) {
    int64_t u, v;
    std::cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
  }

  std::vector<std::vector<int64_t>> adj_vec(adj, adj + n + 1);
  delete[] adj;

  if (IsBipartite(n, adj_vec)) {
    std::cout << "YES";
  } else {
    std::cout << "NO";
  }

  return 0;
}
```

Пояснение к примененному алгоритму:

Каждый школьник представляется как узел графа. Если два школьника обмениваются записками, между соответствующими вершинами проводится ребро. Задача разделения школьников на две группы так, чтобы ни одна пара обменивающихся не оказалась в одной группе, эквивалентна проверке, является ли построенный граф двудольным, то есть его вершины можно разделить на два непересекающихся множества так, что каждое ребро соединяет вершину из одного множества с вершиной из другого множества. Алгоритм итерируется по всем вершинам графа и красит его вершины в три цвета: 0 – не посещена, 1 – цвет A, цвет B. Если вершина еще не покрашена, запускается поиск в ширину с этой вершины. Таким образом, код правильно определяет, что граф, представляющий отношения обмена, является двудольным, и следовательно, школьников можно разделить на две группы так, чтобы никто из обменивающихся не оказался в одной группе

Задача №P «Авиаперелеты»

```cpp
#include <algorithm>
#include <iostream>
#include <vector>

constexpr int INF = 1e9 + 10;

int main() {
  int n;
  std::cin >> n;

  auto dist = std::vector<std::vector<int>>(n, std::vector<int>(n));

  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      int val;
      std::cin >> val;
      dist[i][j] = val;
    }
  }

  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        dist[i][j] = std::min(dist[i][j], std::max(dist[i][k], dist[k][j]));
      }
    }
  }

  int answer = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      answer = std::max(answer, dist[i][j]);
    }
  }

  std::cout << answer;
  return 0;
}
```

Пояснение к примененному алгоритму:

Используем Модифицированную версию этого алгоритма, которая для каждой пары городов i и j ищет такой маршрут (с учетом пересадок, если нужно), где максимальное значение на маршруте минимально.