|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 8** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «**Алгоритмические стратегии или методы разработки алгоритмов. Перебор и  методы его сокращения.**»** | |
|  | |
| Выполнил студент группы ИКБО-03-21 | Хречко С.В. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2022

# **Цель работы**

Получить навыки применения методов, позволяющих сократить число переборов в задачах, которые могут быть решены только методом перебора всех возможных вариантов решения.

# **Постановка задачи**

1. Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.

a. Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы.

b. Привести анализ снижения числа переборов при применении метода.

2. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант №29. Условие задания:

|  |  |
| --- | --- |
| Областная администрация планирует оптимизировать расходы на поддержание дорог. Между поселениями проведены дороги. Требуется избавиться от максимального количества дорог так, чтобы ни одно поселение не оказалось отрезанным от остальных. | Жадный алгоритм |

# **Решение**

Жадный алгоритм (англ. Greedy algorithm) — алгоритм, заключающийся в принятии локально оптимальных решений на каждом этапе, допуская, что конечное решение также окажется оптимальным. Известно, что если структура задачи задается матроидом, тогда применение жадного алгоритма выдаст глобальный оптимум.

В данной задаче, жадный алгоритм будет принимать решения относительно каждой дороги: возможно, ли ее удалить, не нарушив возможность связь между городами.

По заданию требовалось реализовать жадный алгоритм решающий поставленную задачу. Функция принимает на вход матрицу смежности для городов, которая понадобится для функции проверки соединены ли города, а также список ребер. Функция проходит по всем ребрам и проверяет, возможно ли удаление, без нарушения связи городов. Если удаление возможно, то удаление происходит. Проверка происходит путем временного удаления дороги и вызова функции проверки связи между городами, между которыми была удалена дорога, предполагается, что изначально все города связаны. В таком случае если удалять только дороги, удаление которых не приводят к разрыву связи, то и в результате разрывов связи не будет. Таким образом работает жадный алгоритм, решающий данную задачу.

|  |
| --- |
| int deleteRoads(vector<vector<int>>& matrix, const vector<int>& roads){      int sum = 0;      for(int i = 0; i < roads.size(); i+=2){          if(roads[i] == roads[i+1]){              sum++;              matrix[roads[i]][roads[i+1]] -= 1;          }          matrix[roads[i]][roads[i+1]] -= 1; //we delete the road          matrix[roads[i+1]][roads[i]] -= 1;          vector<bool> visited(matrix.size(), false);          if (areCitiesConnected(matrix, roads[i], roads[i+1], visited)){ //we check if it is possible to get from city i to city j without the deleted road              sum++;          }          else{              matrix[roads[i]][roads[i+1]] += 1; //if it is not, we add the road back;              matrix[roads[i+1]][roads[i]] += 1;          }      }      return sum;  } |

Для успешной работы алгоритма требуется функция проверки связи между городами. Функция принимает на вход матрицу смежности городов, два номера города, а также массив посещенных городов. Функция проверяет присутствует ли прямая связь между городами, а также не являются ли переданные города одним и тем же городом. После чего выполняется рекурсивно для городов, которые не были посещены и имеют связь с первым переданным городом.

|  |
| --- |
| bool areCitiesConnected(const vector<vector<int>>& matrix, int city1, int city2, vector<bool>& visited){      if (city1 < 0 || city1 >= matrix.size() || city2 < 0 || city2 >= matrix.size()){          return false;      }      if(city1 == city2){          return true;      }      if (matrix[city1][city2] != 0){          return true;      }      visited[city1] = true;      for (int i = 0; i < matrix.size(); i++){          if (matrix[city1][i] != 0 && !visited[i]){              if (areCitiesConnected(matrix, i, city2, visited)){                  return true;              }          }      }      return false;  } |

По условию задания требовалось оценить количество переборов при решении «в лоб». Таким решением будет являться перебор всех возможных вариантов удаления дорог, а затем выбор лучшего возможного из них. Количество таких вариантов можно оценить как сумму количества перестановок из n по k, где n это количество дорог, а k изменяется от 0 до n c шагом в 1. Максимальный член такой суммы будет иметь следующий примерный вид: , что можно оценить как сложность O(n!). Также для каждого из этих вариантов, пришлось бы вызывать функцию, проверяющую корректность варианта, что не увеличило бы количество переборов, но еще увеличило бы сложность.

По условию задания требовалось привести анализ снижения числа переборов при применении алгоритма. В случае применения алгоритма, мы перебираем лишь все дороги, для каждой впоследствии вызываем функцию проверки возможности удаления дороги, которая по своей сложности очень близка функции, которую пришлось бы использовать в решении «в лоб». Таким образом количество переборов будет снижено с примерно факториального, до линейного.

Пользовательский интерфейс программы предлагает пользователю ввести количество городов, дорог между ними и сами дороги. После чего предлагает протестировать функцию проверки связи между городами, а затем выводит ответ на поставленную задачу, а также получившуюся в результате удаления матрицу смежности городов.

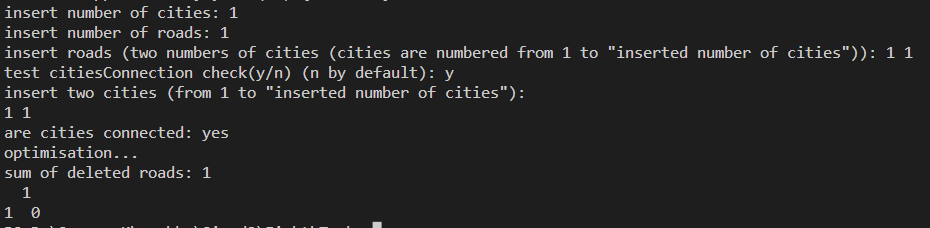


Рисунок 1. Интерфейс программы

# **Тестирование**

Тестирование функции определения связанности городов. На рисунке 2 приведены результаты тестирования, в котором от одного города можно добраться до другого, пусть и не напрямую. На рисунке 3 представлен результат тестирования функции для ситуации, когда два города не связаны вообще никак.

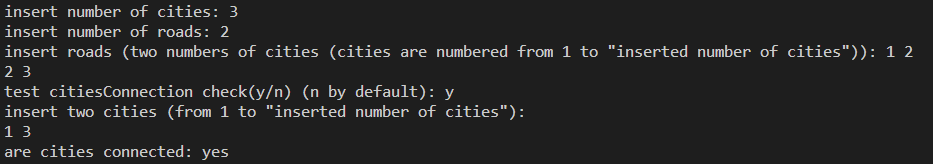


Рисунок 2. Тестирование связанности городов

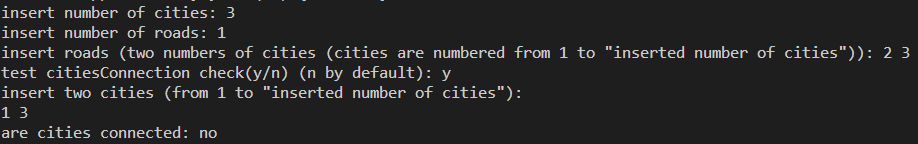


Рисунок 3. Тестирование связанности городов

Тестирование функции решающей поставленную задачу. Граф связей между городами, использованный для тестирования представлен на рисунке 4. Тестирование представлено на рисунке 5.

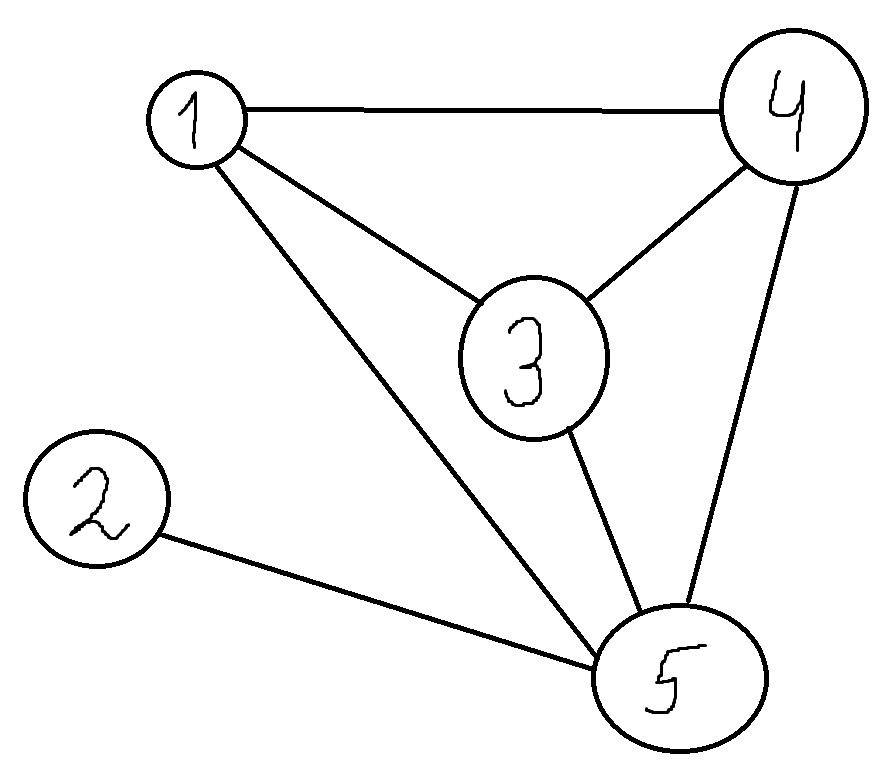


Рисунок 4. Граф для тестирования

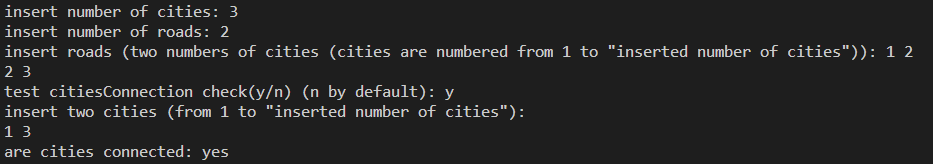


Рисунок 5. Тестирование удаления дорог

Из результатов выполнения программы видно:

1. Программа работает корректно, все функции работоспособны.

# **Вывод**

В результате выполнения работы я:

1. Получил навыки применения методов сокращения числа переборов.
2. Научился программировать жадные алгоритмы.

# **Исходный код программы**

|  |
| --- |
| #include <iostream>  #include <vector>  #include <iomanip>  using namespace std;  bool areCitiesConnected(const vector<vector<int>>& matrix, int city1, int city2, vector<bool>& visited){      if (city1 < 0 || city1 >= matrix.size() || city2 < 0 || city2 >= matrix.size()){          return false;      }      if(city1 == city2){          return true;      }      if (matrix[city1][city2] != 0){          return true;      }      visited[city1] = true;      for (int i = 0; i < matrix.size(); i++){          if (matrix[city1][i] != 0 && !visited[i]){              if (areCitiesConnected(matrix, i, city2, visited)){                  return true;              }          }      }      return false;  }  int deleteRoads(vector<vector<int>>& matrix, const vector<int>& roads){      int sum = 0;      for(int i = 0; i < roads.size(); i+=2){          if(roads[i] == roads[i+1]){              sum++;              matrix[roads[i]][roads[i+1]] -= 1;              continue;          }          matrix[roads[i]][roads[i+1]] -= 1; //we delete the road          matrix[roads[i+1]][roads[i]] -= 1;          vector<bool> visited(matrix.size(), false);          if (areCitiesConnected(matrix, roads[i], roads[i+1], visited)){ //we check if it is possible to get from city i to city j without the deleted road              sum++;          }          else{              matrix[roads[i]][roads[i+1]] += 1; //if it is not, we add the road back;              matrix[roads[i+1]][roads[i]] += 1;          }      }      return sum;  }  int main(){      cout << "insert number of cities: ";      int n;      cin >> n;      vector<vector<int>> matrix(n, vector<int>(n));      cout << "insert number of roads: ";      int m;      cin >> m;      cout << "insert roads (two numbers of cities (cities are numbered from 1 to \"inserted number of cities\")): ";      vector<int> roads(m\*2);      for (int i = 0; i < n; i++){          for(int j = 0; j < n; j++){              matrix[i][j] = 0;          }      }      for (int i = 0; i < m; i++){          int a, b;          cin >> a >> b;          if(a == b) matrix[a-1][b-1] += 1;          else{              matrix[a-1][b-1] += 1;              matrix[b-1][a-1] += 1;          }          roads[i\*2] = a-1;          roads[i\*2+1] = b-1;      }      cout << "test citiesConnection check(y/n) (n by default): ";      char c;      cin >> c;      if (c == 'y'){          cout << "insert two cities (from 1 to \"inserted number of cities\"): " << endl;          int a, b;          cin >> a >> b;          vector<bool> visited(matrix.size(), false);          cout << "are cities connected: " << ((areCitiesConnected(matrix, a-1, b-1, visited))?"yes":"no") << endl;      }      cout << "optimisation..." << endl;      cout << "sum of deleted roads: " << deleteRoads(matrix, roads) << endl;      for(int j = 0; j < matrix.size(); j++){          cout << setw(3) << j+1;      }      cout << " ";      for (int i = 0; i < matrix.size(); i++){          cout << endl << i+1;          for (int j = 0; j < matrix.size(); j++){              cout << setw(3) << matrix[i][j];          }          cout << endl;      }      return 0;  } |