**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Потоки в сетях»**

**Вариант 2**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9302 |  | Сиротин П.О. |
| Преподаватель |  | Тутуева А.В. |

Санкт-Петербург

2021

1. **Постановка задачи.**

* *Задание:*

Входные данные: текстовый файл со строками в формате V1, V1, P, где V1, V2 направленная дуга транспортной сети, а P – ее пропускная способность. Исток всегда обозначен как S, сток – как T

* *Найти:* максимальный поток в сети используя алгоритм Эдмондса-Карпа.
* *Пример данных:*

S O 3

S P 3

O Q 3

O P 2

P R 2

Q R 4

Q T 2

R T 3

1. **Краткое описание реализуемого алгоритма и используемых структур данных. Оценка временной сложности каждого метода**

Для выполнения поставленной задачи я пользовался классами ListL2 и lst\_node, которые были реализованы в прошлой лабораторной работе.

Класс ListL2 необходим, поскольку он подходит для использования как стек (а в некоторых случаях он тождественно равен векторам vector<typename T>).

В классе Graph должны быть только два поля – это размерность графа dimG и матрица размером dimGхdimG.

Класс Graph необходим в таком виде потому, что объектами данного класса в таком случае можно хранить матрицу пропускных способностей рёбер графа и матрицу величин потока.

Программа работает следующим образом: из входного файла собираются все вершины, которые сохраняются в массив char (где каждый элемент – это название той или иной вершины). Кроме того, из входного файла собираются данные для заполнения матрицы пропускных способностей рёбер графа. Поскольку пары вершин во входном файле могут быть абсолютно случайными, то массив названий вершин следует отсортировать, не затрагивая первый и последний символы (поскольку вершина ‘S' всегда является истоком, а 'T’ – стоком). После этого, наконец, начинает свою работу алгоритм Эдмондса-Карпа.

template <class T>

class lst\_node

|  |  |
| --- | --- |
| Элемент | Описание |
| T info | Информация, сохраненная внутри узла списка |
| lst\_node\* next | Указатель на следующий элемент списка |
| lst\_node\* prev | Указатель на предыдущий элемент списка |

template <class T>

class ListL2

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент | Описание | | |
| lst\_node\* top | Указатель на первый элемент списка | | |
| lst\_node\* tail | Указатель на последний элемент списка | | |
| Метод | | Описание | Оценка временной сложности |
| void push\_front(T data) | | Вставка нового элемента в начало списка | О(1) |
| void push\_back(T data) | | Вставка нового элемента в конец списка | О(1) |
| T pop\_front() | | Удаление элемента из начала списка | О(1) |
| T pop\_back() | | Удаление элемента из конца списка | О(1) |
| int FindAnElement(T dataToFind) | | Поиск элемента в списке – если элемент найден, возвращается его индекс, иначе же возвращается -1 | О(n) |
| size\_t give\_size() | | Возвращает количество элементов списка | O(n) |
| void print() | | Вывод информации из элементов стека в консоль | О(n) |
| T what\_at(int index) | | Возвращает данные, находящиеся в элементе, имеющим данный индекс в списке | O(n) |
| lst\_node\* give\_top() | | Возвращает указатель на первый элемент списка | О(1) |
| bool IsEmpty() | | Проверка на пустой список | O(1) |

class Graph

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент | Описание | | |
| int dimG | Указатель на первый элемент списка | | |
| int\*\* matrixG; | Указатель на последний элемент списка | | |
| Метод | | Описание | Оценка временной сложности |
| void Set(int i, int j, int info) | | Вспомогательный метод для упрощения написания алгоритма: установка значения массива по индексам | О(1) |
| void Get(int i, int j) | | Вспомогательный метод для упрощения написания алгоритма: получение значения массива по индексам | O(1) |
| ListL2<int>\* bfs(Graph\*& Flow) | | Алгоритм поиска в ширину | O(n2) |

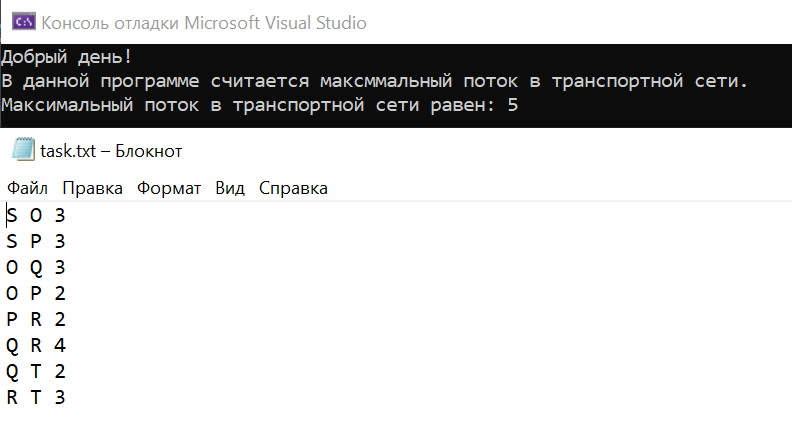
Реализованные функции:

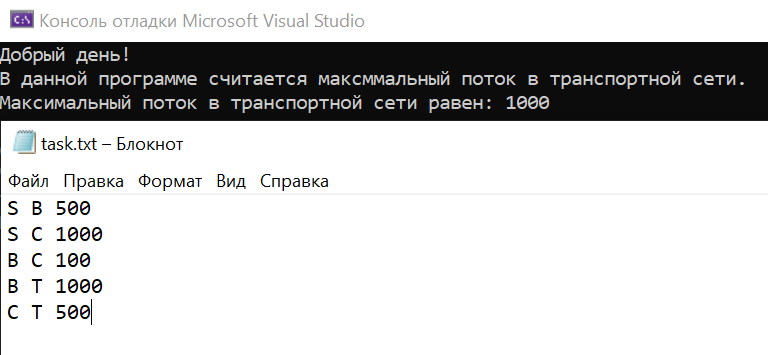
|  |  |  |
| --- | --- | --- |
| Функция | Описание | Оценка временной сложности |
| void ReadFileAndTakeNames(string& names, int& numOfV, ifstream& file) | Поиск имён вершин во входном файле | О(n2) |
| void SortBetweenSandT(int names\_size, char\* names | Сортировка внутри массива имён вершин: поскольку S всегда начало, а T – всегда конец, то надо сортировать значения в массиве без этих двух крайних значений | О(n2) |
| void SearchCapacity(int Vs, Graph\* Capacity, char\* names, ifstream& file) | Поиск пропускных способностей во входном файле | О(n2) |
| unsigned EdKaAlgo(int Vs, Graph& Capa) | Алгоритм Эдмонса-Карпа (V – количество вершин графа, Е – количество рёбер графа) | О(|V|\*|E|2) |
| void Summing(unsigned& MaxFlow) | Процесс работы программы | О(|V|\*|E|2) |

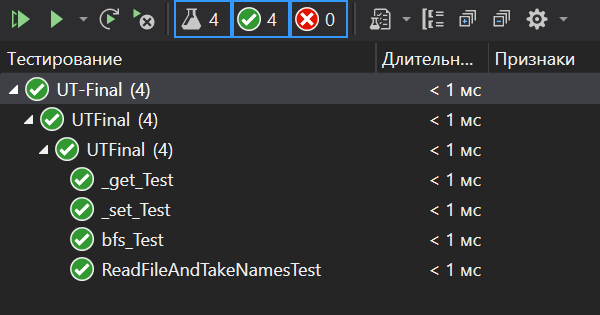
Unit-tests:

|  |  |
| --- | --- |
| **Название теста** | **Что проверяет** |
| ReadFileAndTakeNamesTest | Проверка метода ReadFileAndTakeNames |
| \_get\_Test | Проверка метода Get |
| \_set\_Test | Проверка метода Set |
| bfs\_Test | Проверка метода BFS |

# Пример работы







# Листинг

|  |
| --- |
| ListL2.h |
| #pragma once  #include <stdexcept>  #include <iostream>  using namespace std;  template <class T>  class ListL2  {  private:  class lst\_node  {  public:  lst\_node(T info, lst\_node\* next = nullptr, lst\_node\* prev = nullptr)  {  this->info = info;  this->next = next;  this->prev = prev;  }  lst\_node()  {  this->prev = nullptr;  this->next = nullptr;  }  ~lst\_node() {};  T info;  lst\_node\* prev;  lst\_node\* next;  };  public:  lst\_node\* top;  lst\_node\* tail;  size\_t size;  ListL2()  {  top = nullptr;  tail = nullptr;  size = 0;  }  void push\_back(T data)  {  if (isEmpty())  {  top = tail = new lst\_node(data);  }  else  {  lst\_node\* temp = new lst\_node(data, nullptr, tail);  tail->next = temp;  tail = temp;  }  size++;  }  void push\_front(T data)  {  if (!isEmpty())  {  lst\_node\* cur = top;  top = new lst\_node(data, top);  cur->prev = top;  }  else  {  top = tail = new lst\_node(data, nullptr, nullptr);  }  size++;  }  T pop\_front()  {  if (!top)  throw out\_of\_range("An error occured.");  if (top->next == nullptr)  {  T tmp = top->info;  tail = nullptr;  top = nullptr;  delete top;  delete tail;  size--;  return tmp;  }  else  {  lst\_node\* node\_temp = top;  T tmp = node\_temp->info;  top = top->next;  top->prev = nullptr;  node\_temp->next = nullptr;  delete node\_temp;  size--;  return tmp;  }  }  T pop\_back()  {  if (!tail)  throw out\_of\_range("An error occured.");  if (tail->prev == nullptr)  {  T tmp = tail->info;  tail = nullptr;  top = nullptr;  delete top;  delete tail;  size--;  return tmp;  }  else  {  lst\_node\* node\_temp = tail;  T tmp = node\_temp->info;  tail = tail->prev;  tail->next = nullptr;  node\_temp->prev = nullptr;  delete node\_temp;  size--;  return tmp;  }  }  void print()  {  lst\_node\* current = top;  if (top)  {  while (current->next)  {  cout << current->info << " ";  current = current->next;  }  cout << current->info;  cout << endl;  }  else throw out\_of\_range("An error occured: list is empty.");  }  T what\_at(int index)  {  if (top)  {  lst\_node\* current = top;  int i = 0;  while (i != index)  {  if (current->next)  current = current->next;  else  if (i == index)  return current->info;  i++;  }  return current->info;  }  else throw "An error occured.";  }  int FindAnElement(T dataToFind)  {  int i = 0;  if (top != nullptr)  {  lst\_node\* current = top;  while(current)  {  if (dataToFind == current->info)  {  return i;  }  current = current->next;  i++;  }  }  return -1;  }  size\_t give\_size()  {  return size;  }  ~ListL2()  {  T tmp;  while (top)  {  tmp = pop\_front();  }  }  lst\_node\* give\_top()  {  return top;  }  bool isEmpty()  {  return (top == nullptr);  }  }; |
| GraphsAndStuff.h |
| #pragma once  #include "ListL2.h"  #include <iostream>  #include <fstream>  using namespace std;  class Graph  {  public:  int dimG;  int\*\* matrixG;  Graph(int dimG);  ~Graph();  void Set(int i, int j, int info);  int Get(int i, int j);  ListL2<int>\* BFS(Graph& Flow);  };  Graph::Graph(int dimG)  {  this->dimG = dimG;  matrixG = new int\*[dimG];  for (int i = 0; i < dimG; i++)  matrixG[i] = new int[dimG];  for (int i = 0; i < dimG; i++)  for (int j = 0; j < dimG; j++)  matrixG[i][j] = 0;  }  Graph::~Graph()  {  for (int i = 0; i < this->dimG; ++i)  delete[] matrixG[i];  delete[] matrixG;  }  // in C# it can be made as {get; set;}  void Graph::Set(int i, int j, int info)  {  matrixG[i][j] = info;  }  int Graph::Get(int i, int j)  {  return matrixG[i][j];  }  ListL2<int>\* Graph::BFS(Graph& Flow)  {  ListL2<int> queue;  ListL2<int>\* path = new ListL2<int>;  int\* level = new int[dimG];  bool\* visited = new bool[dimG];  for (int i = 0; i < dimG; i++)  visited[i] = false;  level[0] = 0;  queue.push\_back(0);  visited[0] = true;  while (!(queue.isEmpty()))  {  int u = queue.what\_at(0);  queue.pop\_front();  for (int i = 0; i < dimG; i++)  {  if (u == i)  continue;  int V = matrixG[u][i];  int first\_ = Flow.Get(u, i);  int second\_ = Flow.Get(i, u);  int third\_ = matrixG[i][u];  if (!visited[i] && ((V != first\_) || ((third\_ != second\_) && (second\_ != 0))))  {  queue.push\_back(i);  visited[i] = true;  level[i] = level[u] + 1;  }  }  }  int Max = level[dimG - 1];  path->push\_front(dimG - 1);  while (Max != 0 && Max >= 0)  {  for (int j = 0; j < dimG; ++j)  {  int V = matrixG[j][path->what\_at(0)];  int first\_ = Flow.Get(j, path->what\_at(0));  int second\_ = Flow.Get(path->what\_at(0), j);  int third\_ = matrixG[path->what\_at(0)][j];  if ((level[j] == Max - 1) && ((V != first\_) || ((third\_ != second\_) && (second\_ != 0))))  {  if (level[j] == Max - 1)  {  path->push\_front(j);  Max--;  break;  }  }  }  }  delete[] level;  delete[] visited;  return path;  }  // Search for unique vertices in file  void ReadFileAndTakeNames(string& names, int& numOfV, ifstream& file)  {  char temp = ' ';  while (!file.eof())  {  temp = file.get();  if (temp == ' ' || isdigit(temp) || temp == '\n') // we should skip it here  continue;  if (!isdigit(temp)) // getline gives us "V U 3", we need V and U  {  for (int i = 0; i < numOfV; ++i)  {  if (names[i] == temp)  {  temp = ' ';  break;  }  }  if (temp == ' ')  continue;  else  {  names += temp;  numOfV++;  }  }  }  }  // Now we are making a capacity matrix from a file  void SearchCapacity(int Vs, Graph\* Capacity, char\* names, ifstream& file)  {  char temp = ' ';  unsigned from = 0, to = 0;  string temp\_number;  while (!file.eof())  {  temp\_number = "";  temp = file.get();  if ((temp == '\n') || (temp == ' '))  continue;  else if (!isdigit(temp))  {  for (int i = 0; i < Vs; ++i)  {  if (names[i] == temp)  from = i; // it's an analog for ReadFile() from Lab3  }  temp = file.get(); // spacebar  temp = file.get();  for (int i = 0; i < Vs; ++i)  {  if (names[i] == temp)  to = i; // it's an analog for ReadFile() from Lab3  }  }  else if (isdigit(temp))  {  temp\_number.push\_back(temp); // we can't do (string\_var + char\_var) so we need some standard methods  while (isdigit(temp = file.get())) // in case if there's a number >10  temp\_number.push\_back(temp);  Capacity->Set(from, to, atoi(temp\_number.c\_str())); // capasity->matrixG[from][to] = number  if (file.eof())  break;  }  }  }  void SortBetweenSandT(int names\_size, char\* names)  {  // S is always start, T is always end  // but we should sort all between this two  for (int i = 0; i < names\_size - 1; ++i)  {  for (int j = 0; j < names\_size - 1; ++j)  {  if ((names[j] > names[j + 1]) && (names[j] != 'S') && (names[j + 1] != 'T'))  {  char temp = names[j];  names[j] = names[j + 1];  names[j + 1] = temp;  }  }  }  }  // The main algorithm of program  unsigned EdKaAlgo(int Vs, Graph& Capa)  {  Graph\* pathFlow = new Graph(Vs);  unsigned toFindMaxFlow = 0;  ListL2<int>\* Path = new ListL2<int>();  Path = Capa.BFS(\*pathFlow); // the best way, found by bfs  int\* Remainder = new int[Path->give\_size() - 1];  while (!(Path->isEmpty()) || (Path->give\_size() >= 1))  {  Path = Capa.BFS(\*pathFlow);  Remainder = new int[Path->give\_size() - 1];  if ((Path->isEmpty()) || (Path->give\_size() == 1))  break;  for (int i = 0; i < Path->give\_size() - 1; ++i)  {  unsigned to = Capa.Get(Path->what\_at(i), Path->what\_at(i + 1));  //looking for an edge with minimum bandwidth in the residual network  unsigned from = Capa.Get(Path->what\_at(i + 1), Path->what\_at(i));  if (to != 0)  Remainder[i] = to - pathFlow->Get(Path->what\_at(i), Path->what\_at(i + 1));  else if (from != 0)  Remainder[i] = pathFlow->Get(Path->what\_at(i + 1), Path->what\_at(i));  else  Remainder = 0;  }  //looking for a minimum bandwidth in the residual network  unsigned minCapa = Remainder[0];  for (int i = 1; i < Path->give\_size() - 1; i++)  if (Remainder[i] < minCapa)  minCapa = Remainder[i];  // minCapa is the minimum bandwidth in the residual network  for (int i = 0; i < Path->give\_size() - 1; i++)  {  // counting new bandwidths  unsigned to = pathFlow->Get(Path->what\_at(i), Path->what\_at(i + 1));  unsigned from = pathFlow->Get(Path->what\_at(i + 1), Path->what\_at(i));  if (from == 0)  pathFlow->Set(Path->what\_at(i), Path->what\_at(i + 1), to + minCapa);  else  pathFlow->Set(Path->what\_at(i + 1), Path->what\_at(i), from - minCapa);  }  }  for (int i = 0; i < Vs; i++)  {  if (pathFlow->Get(0, i) != 0)  toFindMaxFlow += pathFlow->Get(0, i);  }  delete Path;  delete pathFlow;  delete[] Remainder;  return toFindMaxFlow;  }  void Summing(unsigned& MaxFlow)  {  int VNum = 0;  ifstream file;  file.open("task.txt");  string tempStr = "";  ReadFileAndTakeNames(tempStr, VNum, file);  VNum = VNum - 1;  char\* names = new char[VNum];  for (int k = 0; k < VNum; ++k)  names[k] = tempStr[k];  SortBetweenSandT(VNum, names);  file.clear();  file.seekg(0, ios::beg);  Graph\* Throughput = new Graph(VNum);  SearchCapacity(VNum, Throughput, names, file);  MaxFlow = EdKaAlgo(VNum, \*Throughput);  delete Throughput;  delete[] names;  } |
| Term4\_TermWork.cpp |
| #include <iostream>  #include "GraphsAndStuff.h"  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  cout << "Добрый день!" << endl;  cout << "В данной программе считается максммальный поток в транспортной сети." << endl;  unsigned MaxFlow = 0;  Summing(MaxFlow);  cout << "Максимальный поток в транспортной сети равен: " << MaxFlow << endl;  return 0;  } |
| UT-Final.cpp |
| #include "CppUnitTest.h"  #include "../Term4\_TermWork/ListL2.h"  #include "../Term4\_TermWork/GraphsAndStuff.h"  using namespace Microsoft::VisualStudio::CppUnitTestFramework;  namespace UTFinal  {  TEST\_CLASS(UTFinal)  {  public:  TEST\_METHOD(\_get\_Test)  {  int dim = 3;  Graph Test(dim);  Assert::IsTrue(Test.Get(0, 1) == 0);  }  TEST\_METHOD(\_set\_Test)  {  int dim = 3;  Graph Test(dim);  Test.Set(2, 2, 3);  Assert::IsTrue(Test.Get(2, 2) == 3);  }  TEST\_METHOD(bfs\_Test)  {  int dim = 3;  Graph\* Test = new Graph(dim);  Graph\* Flow = new Graph(3);  Test->Set(0, 1, 1);  Test->Set(0, 2, 1);  Test->Set(1, 2, 1);  Assert::IsTrue(Test->BFS(\*Flow)->what\_at(0) == 0);  Assert::IsTrue(Test->BFS(\*Flow)->what\_at(1) == 2);  }  TEST\_METHOD(ReadFileAndTakeNamesTest)  {  ifstream read("C:\\Users\\Paul\\source\\repos\\Term4\_TermWork\\task.txt");  int VertexesNumber = 0;  string UniqueVertexes = "";  ReadFileAndTakeNames(UniqueVertexes, VertexesNumber, read);  VertexesNumber = VertexesNumber - 1;  char\* names = new char[VertexesNumber];  for (int k = 0; k < VertexesNumber; ++k)  names[k] = UniqueVertexes[k];  read.close();  Assert::IsTrue(names[0] == 'S');  Assert::IsTrue(names[1] == 'O');  Assert::IsTrue(names[2] == 'P');  Assert::IsTrue(names[3] == 'Q');  Assert::IsTrue(names[4] == 'R');  Assert::IsTrue(names[5] == 'T');  Assert::IsTrue(VertexesNumber == 6);  delete[] names;  }  };  } |

# Вывод

Для написания данной курсовой работы мне потребовались знания, полученные за четвертый семестр обучения. Программа успешно находит максимальный поток в транспортной сети с помощью алгоритма Эдмондса-Карпа.