

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Информатика и вычислительная техника

Кафедра Кибербезопасность информационных систем

**ЛАБОРАТОРНАЯ РАБОТА № 2**

на тему «Алгоритмы построения остова минимальной стоимости»

Выполнил обучающийся гр. ВКБ33

Короп Анастасия

Проверил

Доцент, Савельев Василий Александрович

Ростов-на-Дону

2022

**Задание 1**

Реализовать алгоритм построения остовного дерева минимальной стоимости. Создать тесты и отладить.

|  |
| --- |
| import os.path as osp  def load\_graph(file,is\_bin=False):  if is\_bin:  graph=open(file,'rb')  fsize=osp.getsize(file)  reblist=[]  for e in range(int(fsize/12)):  temp=[]  for k in range(3):  temp.append(int.from\_bytes(graph.read(4),'little'))  reblist.append(temp)  else:  graph=open(file,'r')  reblist=[[int(k) for k in e.split()] for e in graph.readlines()]  graph.close()  return reblist  def save\_graph(graph,file\_name,is\_bin=False):  if is\_bin:  newf=open(file\_name,'wb') |

**Задание 2**

Использовать алгоритм построения остова минимальной стоимости для вывода графа в виде XML-документа. Использовать атрибуты спецификации XLINK для представления ребер графа не вошедших в остов. Создать тесты и отладить. Оформить в библиотеку. Сравнить по используемой памяти и времени работы с реализованными в первой лабораторной работе способами вывода графа.

|  |
| --- |
| import os.path as osp  def load\_graph(file,is\_bin=False):  if is\_bin:  graph=open(file,'rb')  fsize=osp.getsize(file)  reblist=[]  for e in range(int(fsize/12)):  temp=[]  for k in range(3):  temp.append(int.from\_bytes(graph.read(4),'little'))  reblist.append(temp)  else:  graph=open(file,'r')  reblist=[[int(k) for k in e.split()] for e in graph.readlines()]  graph.close()  return reblist  def save\_graph(graph,file\_name,is\_bin=False):  if is\_bin:  newf=open(file\_name,'wb')  for e in graph:  newf.write(e[0].to\_bytes(4, byteorder='little'))  newf.write(e[1].to\_bytes(4, byteorder='little'))  newf.write(e[2].to\_bytes(4, byteorder='little'))  else:  newf=open(file\_name,'w')  for e in graph:  newf.write('{} {} {}\n'.format(e[0],e[1],e[2]))  newf.close() |

**Задание 3**

Реализовать ввод графа из XML-документа. Создать тесты и отладить. Оформить в библиотеку. Сравнить по используемой памяти и времени работы с реализованными в первой лабораторной работе способами ввода графа.

|  |
| --- |
| from sys import argv  import graphlib as gr  if len(argv)>1:  if argv[1]!='/?':  filename=argv[1]  else:  print('Поиск в графе кратчайшего пути между заданными вершинами.\nlaba1.py [имя\_файла\_графа] (тип\_файла /b-бинарный файл или /n -небинарный) (номер\_начальной\_вершины) (номер\_конечной\_вершины)')  exit()  if len(argv)>4:  if argv[2]=='/b':  is\_bin=True  else:  is\_bin=False  start=int(argv[3])  fin=int(argv[4])  else:  is\_bin=False  start=0  fin=3  else:  is\_bin=False  start=0  fin=3  filename='input.txt'  def graph\_short\_path\_find(cpoint,tpoint,rebrs,length=0,dellst=[],lens=[],path=''):  if length==0:  lens=[]  path+=str(cpoint)+'-'  if cpoint==tpoint:  lens.append(length)  #print('Найден путь ',path[:-1],':',length)  return None  for num in dellst:  rebrs.pop(num)  dellst=[]  for c,d in enumerate(rebrs):  if (cpoint in rebrs[c]):  dellst.append(c)  dellst.reverse()  for num in dellst:  if rebrs[num][0]!=cpoint:  nextp=rebrs[num][0]  else:  nextp=rebrs[num][1]  graph\_short\_path\_find(nextp,tpoint,rebrs[:],length+rebrs[num][2],dellst,lens,path)  if not lens:  otv='{} {} -1'.format(cpoint,tpoint)  else:  otv=min(lens)  lens=''  return otv  graph=gr.load\_graph(filename,is\_bin)  length=graph\_short\_path\_find(start,fin,graph)  print(length) |

**Задание 4**

Реализовать алгоритм Прима. Создать тесты и отладить.

|  |
| --- |
| #include <iostream>  //Сначало пишешь количество вершин, потом матрицу смежности и все, считает сам (матрица смежности это M На M матрица где нумерация вбок и вниз это вершины, короче как табличка умножения ток расстояние от 3 до 3 равно 0  using std::cout;  using std::cin;  int a, b, u, v, ne = 1;  int visited[10] = {0}, min = 0, mincost=0, cost[10][10];  int main()  {  int n = 0;  int path[100] = {0}; //массив для вершин  int path\_index = 0;  //clrscr();  cout << "Number of vertices"; //кол-во вершин  cin >> n;  cout << "Adjacency matrix\n"; //матрица смежности  for(int i=1; i<=n; i++){  for(int j=1; j<=n; j++){  cin >> cost[i][j];  if(cost[i][j] == 0)  cost[i][j] = 999; //999 - типо бесконечное расстояние, необходимо для вычислений в алгоритме  }  }  visited[1] = 1;  cout << "\n";  while(ne < n){  int min = 999;  int i = 1;  for(; i<=n; i++){  for(int j=1; j<=n; j++){  if(cost[i][j] < min){  if(visited[i] != 0)  {  min = cost[i][j];  a = u = i;  b = v = j;  }  }  }  }  if(visited[u] == 0 || visited[v] == 0) {  path[path\_index] = b;  path\_index++;  //cout<<"\n "<<ne++<<" "<<a<<" "<<b<<min; //  ne++; // либо эту строчку, либо ту что выше  mincost+= min;  visited[b] = 1;  }  cost[a][b] = cost[b][a] = 999;  }    cout << "\n";  cout << 1 << " --> ";  for (int i=0; i<n-1; i++)  {  cout << path[i];  if (i < n-2) cout <<" --> ";  }  cout << "\n Minimum cost " << mincost;  } |

**Задание 5**

Реализовать алгоритм Крускала. Создать тесты и отладить.

|  |
| --- |
| #include <stdio.h>  #include <malloc.h>  #include <stdlib.h>  //Туториал  //сначало пишешь количество вершин, потом количество ребер (2 цыфры)  //затем пишешь N строчек - их столько сколько ребер, где 1 и 2 это пометки ребра от вершины к другой вершине, а 3 число это вес ребра  typedef struct {  int from; // начальная вершина ребра  int to; // вершина  double w; // вес края  } edge\_t;  typedef struct set\_t {  struct set\_t \*p; // link on parent  } set\_t;  int NS; // количество комплектов  set\_t \*sets; // массив наборов  int NE; // количество ребер  edge\_t \*E; // массив ребер  int NV; // количество ребер 2  // функция сравнения для сортировки ребер по весу  int cmpw(edge\_t \*a, edge\_t \*b)  {  if(a->w > b->w ) return 1;  if(a->w < b->w ) return -1;  return 0;  }  set\_t\*  get\_set\_id(set\_t\* s)  {  if(s == s->p )  return s;  else {  set\_t \*p = get\_set\_id(s->p);  s->p = p;  return p;  }  }  set\_t\*  join\_sets(set\_t \*a, set\_t \*b)  {  a->p = b;  return a;  }  void  take\_edge(int edge\_id)  {  printf("%d %d %lf\n", E[edge\_id].from, E[edge\_id].to, E[edge\_id].w);  }  int  main()  {  int i;  double W = 0;  scanf("%d%d", &NV, &NE);  E = (edge\_t\*) malloc(NE \* sizeof(edge\_t));  sets = (set\_t\*) malloc(NV \* sizeof(set\_t));  for(i = 0; i < NE ; i++)  {  scanf("%d%d%lf", &E[i].from, &E[i].to, &E[i].w);  }  // Сортировка ребер по весу  qsort(E, NE, sizeof(edge\_t), (int (\*)(const void\*, const void\*)) cmpw);  // Создание набор одноточечных наборов, тупо, но так  NS = NV;  for(i = 0; i < NS ; i++)  sets[i].p = &sets[i];  // Извлечение следующего ребра с минимальным весом  for(i=0; NS > 1 && i < NE; i++)  {  // если ребро не может быть добавлено к дереву, то переход к следующему ребру  if ( get\_set\_id ( &sets[E[i].from]) == get\_set\_id ( &sets[E[i].to]) )  continue;  // добавление ребра к дереву  join\_sets ( get\_set\_id (&sets[E[i].from] ), get\_set\_id ( &sets[E[i].to]) );  NS--;  take\_edge(i);  W += E[i].w;  }  if(NS != 1)  fprintf(stderr, "warning: Graph is not connected.\n");  printf("Covering tree weight = %lf\n", W);  } |