Міністерство освіти і науки України

Національний Технічний Університет

«Харківський Політехнічний Інститут»

Кафедра «Стратегічного Керування»

Лабораторна робота № 6

«ФУНДАМЕНТАЛЬНІ АЛГОРИТМИ НА ГРАФАХ І ДЕРЕВАХ»

Перевірила: Мошко Є. О.

Виконав:

Харків, 2018

Лабораторна робота № 6

Тема: Фундаментальні Алгоритми На Графах І Деревах

Завдання 1

Розробити програму, яка читає з клавіатури числа N, M (1 < N, M < 256) —кількість вершин та ребер графу; послідовність M пар цілих чисел -ребра графу. Програма зберігає граф та виконує над ним алгоритм згідно варіанту.

Варіанти представлення графів

(2 варіант)

1 Матриця суміжності.

2 Пошук у глибину. На екран потрібно вивести вершини у порядку обходу. Для кожної вказати час початку розгляду, кінця розгляду та предка у дереві обходу.

Хід роботи

|  |
| --- |
| #include <iostream>  using namespace std;  enum Color {White, Gray, Black};  struct List{  int data;  List\* next;  int Color = White;  int time\_start;  int time\_finish;  List\* predecessor;  };  class Graph{  private:  List\*\* arr;  List\*\* nodes;  int size;  List\* sequence = new List;  public:  Graph(){}  Graph(int N) : size(N){  arr = new List\*[N];  nodes = new List\*[size];  for (int i=1; i<=size;i++){  arr[i]=NULL;  nodes[i]=nodeCreation(i); //New array of type List\* fo storing colors  }  }  ad\_matrix(){  int\*\* ad\_arr = new int\*[size]; // Creating adjacency matrix  for (int i=0; i<size; i++){  ad\_arr[i] = new int[size];  for (int j=0; j<size; j++){  ad\_arr[i][j] = 0;  }  }  for (int i=1; i<=size; i++){ // Putting values in the adjacency matrix  if (arr[i]!=NULL){  for (List\* j = arr[i]; j!=NULL; j=j->next){  ad\_arr[i-1][j->data-1]=1;  }  }  }  for (int i=0; i<size; i++){ // Displaying adjacency matrix  cout << "Node " << i+1 << ":\t\t";  for (int j=0; j<size; j++){  cout << ad\_arr[i][j] << " ";  }  cout << endl;  }  }  void DFS(){  int time = 0;  cout << "Node 1";  //for (int i = 1; i<size; i++){  if (arr[1]!=NULL){  time++;  nodes[1]->Color = Gray;  nodes[1]->time\_start=time;    sequence->next=NULL;  List\* newnode = nodeCreation(1);  newnode->next = sequence->next;  sequence->next = newnode;    for (List\* j = arr[1]; j!=NULL; j=j->next){  if (nodes[j->data]->Color==White){  nodes[j->data]->predecessor=nodes[1];  DFS\_Visit(j, time);  cout << " > (" << nodes[1]->data << ")";  }  }  nodes[1]->time\_finish=++time;  cout << endl << endl;  // }  }  for (List\* i = sequence->next; i!=NULL; i=i->next){  cout << i->data << " (" << nodes[i->data]->time\_start << "/" << nodes[i->data]->time\_finish << ")";  if (i->data!=1) cout << " Predecessor: " << nodes[i->data]->predecessor->data << endl;  cout << endl;  }  }  void DFS\_Visit(List\*& j, int& time){  time++;  List\* newnode = nodeCreation(j->data);  newnode->next = sequence->next;  sequence->next = newnode;    cout << " > " << j->data;  nodes[j->data]->Color = Gray;  nodes[j->data]->time\_start=time;  for (List\* i = arr[j->data]; i!=NULL; i=i->next){  if (nodes[i->data]->Color==White){  nodes[i->data]->predecessor=nodes[j->data];  DFS\_Visit(i, time);  cout << " > (" << nodes[j->data]->data << ")";  }  }  nodes[j->data]->Color = Black;  time++;  nodes[j->data]->time\_finish=time;  }  put(int v1, int v2){ // Putting elements in the array of pointers to the lists  if ((v2<=size)&&(v1<=size)){  List\* newnode=nodeCreation(v2);  if (arr[v1]==NULL){  arr[v1]=newnode;  }  else  {  bool lock;  for (List \*i=arr[v1];i!=NULL;i=i->next){  if (i->data==v2) {  lock=true;  break;  }  }  if (!lock){  newnode->next=arr[v1]->next;  arr[v1]->next=newnode;  }  else{  cout << "This arc already exists" << endl;  }  }  }  else{  v1>size?cout << "Node " << v1 << " doesn't exist" << endl:cout << "Node " << v2 << " doesn't exist" << endl;  }  }  List\* nodeCreation(int value){  List\* node = new List;  node->data = value;  node->next = NULL;  return node;  }  };  int main() {  int N, M;  int v1,v2;  // cout << "Enter N nodes and M arcs: ";  // cin >> N >> M;  N=4;  Graph g(N);  g.put(1,2);  g.put(1,4);  g.put(2,3);  g.put(3,1);  g.put(4,2);    // for (int i=0; i<M; i++){  // cin >> v1 >> v2;  // g.put(v1,v2);  // }  g.ad\_matrix();  cout << endl;  g.DFS();  } |

Приклад виконання програми:

|  |
| --- |
| Node 1: 0 1 0 1  Node 2: 0 0 1 0  Node 3: 1 0 0 0  Node 4: 0 1 0 0  Node 1 > 2 > 3 > (2) > (1) > 4 > (1)  4 (6/7) Predecessor: 1  3 (3/4) Predecessor: 2  2 (2/5) Predecessor: 1  1 (1/8) |

Висновок

При виконанні цієї лаби я ознайомився зі способами представлення графів, з пошуком у ширину, та глибину. Здійснив алгоритм пошуку у глибину. Для здійснення було використано рекурсивну функцію DFS\_Visit. Для збереження інформації про колір та предка довелося створити додатковий масив вузлів List\*\* nodes, що містить у собі тільки номери вузлів та не містить інформації про їх зв’язки на відмінно від List\*\* arr де зосереджена інформація про ребра.