Министерство образования, культуры и исследований Республики Молдова Технический Университет Молдовы Департамент Программной Инженерии и Автоматики

Лабораторная работа №2

Поиск в графе.

Алгоритм поиска в глубину. Алгоритм поиска в ширину.

По предмету

Дискретная математика

Подготовил: Оларь Артём,

Группа TI – 210

г. Кишинев 2021

Цель работы:

Изучение алгоритмов поиска в графе, а также различных форм организации хранения и обработки данных.   
Разработка программы, реализующей алгоритм поиска в глубину и алгоритм поиска в ширину.

  Задание:

1. Основываясь на одном из алгоритмов, разработать процедуры, выполняющие обход графа в глубину и в ширину.

2.  Используя разработанные процедуры создать программу, выполняющую следующие функции:

        -- ввод графа в виде списка смежности;

        -- обход введенного графа в глубину и в ширину;

        -- выдачу номеров пройденных вершин на дисплей;

Теория:

/ Структура данных – Очередь /

        Очередь используется для реализации такой дисциплины обработки элементов списка, при которой элементы списка удаляются из него в порядке их включения в список (т.е. по принципу "Первым пришел - первым ушел").

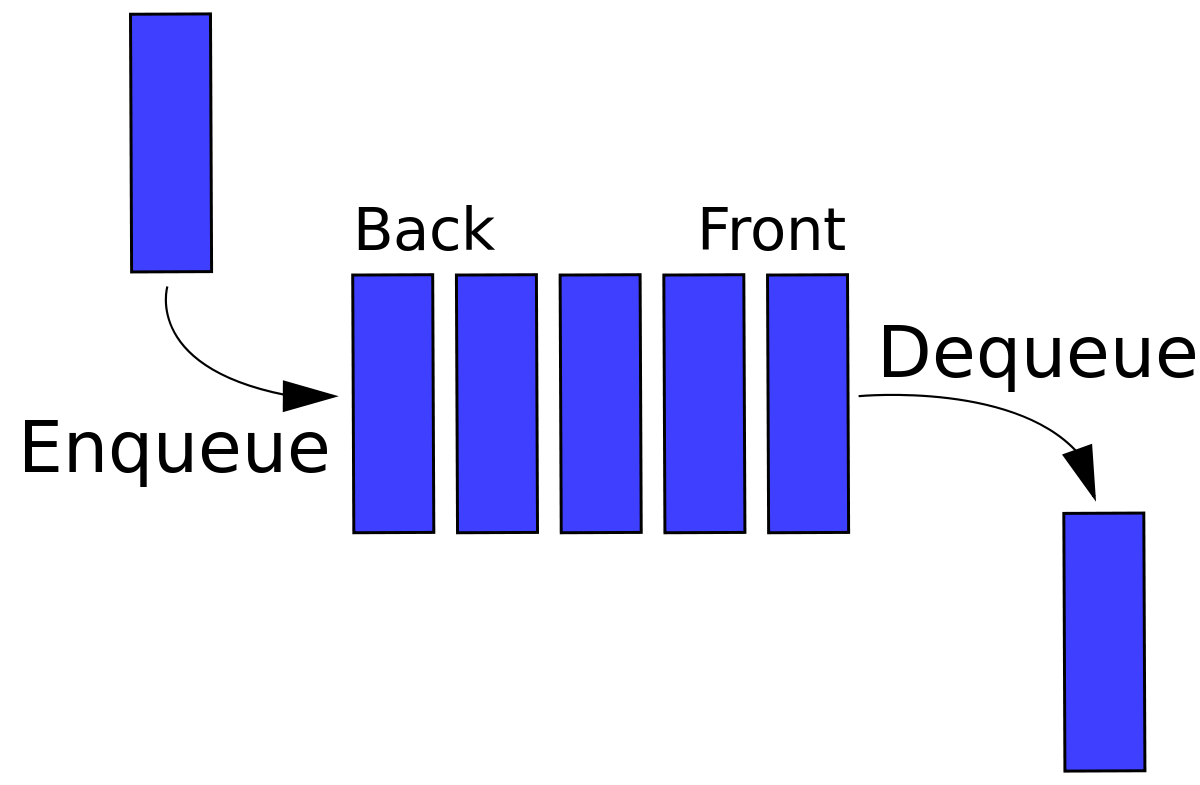
        Набор базисных операций над списками, являющимися очередями, состоит из четырех операций:

        1) Создание пустой очереди;

        2) Проверка очереди на пустоту;

        3) Выборки первого элемента из очереди с одновременным его удалением;

        4) Занесения некоторого значения базисного типа в качестве нового последнего элемента очереди.



/ Структура данных – Стек /

Стек используется для реализации такой дисциплины обработки элементов списка, при которой элементы списка удаляются из него в порядке, обратном порядку их занесения в стек (т.е. по принципу "Последним пришел - первым ушел").

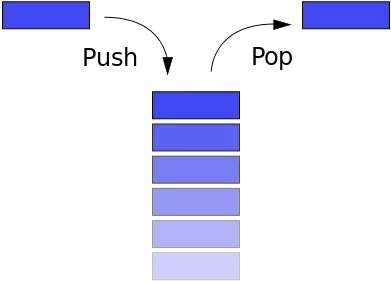
Набор базисных операций над списками, являющимися стеками, состоит из пяти операций:

        1) Создание пустого стека;

        2) Проверка стека на пустоту;

        3,4) Выборки последнего элемента из стека без удаления его из стека или с его удалением;

        5) Занесения некоторого значения базисного типа в качестве нового последнего элемента стека.



/ Обход графа в глубину /

Поиск в глубину предполагает продвижение вглубь до тех пор, пока это возможно. Невозможность продвижения означает, что следующим шагом будет переход на последний, имеющий несколько вариантов движения (один из которых исследован полностью), ранее посещенный узел (вершина).

Применения алгоритма поиска в глубину

1. Поиск любого пути в графе.

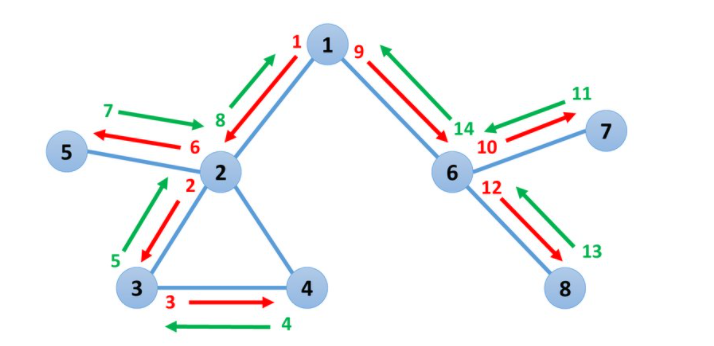
2. Поиск лексикографически первого пути в графе.

3. Проверка, является ли одна вершина дерева предком другой.

4. Поиск наименьшего общего предка.

5. Топологическая сортировка.

6. Поиск компонент связности.



/ Обход графа в ширину /

Подразумевает поуровневое исследование графа:

1. Вначале посещается корень – произвольно выбранный узел,

2. Затем – все потомки данного узла,

3. После этого посещаются потомки потомков и т.д.

Применения алгоритма поиска в ширину

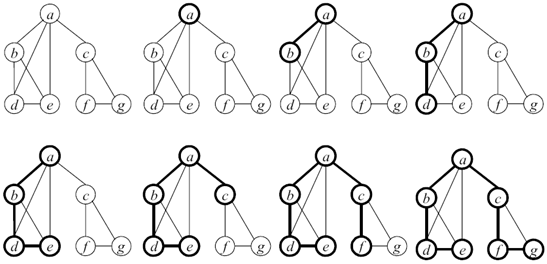
1. Поиск кратчайшего пути в невзвешенном графе (ориентированном или неориентированном).

2. Поиск компонент связности.

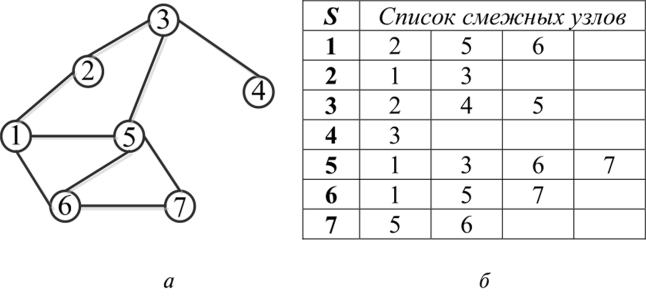
3. Нахождения решения какой-либо задачи (игры) с наименьшим числом ходов.

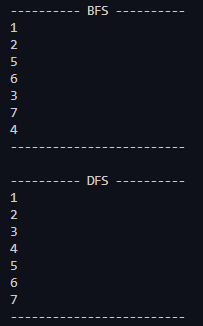
4. Найти все рёбра, лежащие на каком-либо кратчайшем пути между заданной парой вершин.

5. Найти все вершины, лежащие на каком-либо кратчайшем пути между заданной парой вершин.



Работа программы и скриншоты





BFS – обход в ширину

DFS – обход в глубину

Код программы

#include <iostream>

#include <stack>

#include <vector>

#include <queue>

using namespace std;

void addAdjacencyList(vector <vector <int>> &adjList, int &rows) {

  cout << "Number of peaks? : ";

  cin >> rows;

  int peaksInRow = 0;

  int tempNum = 0;

  vector <int> row;

  for (int i = 0; i < rows; i++) {

    cout << "Numbers in a line? : ";

    cin >> peaksInRow;

    for (int j = 0; j < peaksInRow; j++) {

      cin >> tempNum;

      row.push\_back(tempNum);

    }

    adjList.push\_back(row);

    row.clear();

  }

}

int checkNumber(vector <int> partOfList, const int step) {

  if (partOfList.size() == 0) {

    return 0;

  } else {

    for (int i = 0; i < partOfList.size(); i++) {

      if (partOfList[i] == step + 1) {

        return 1;

      }

    }

    return 0;

  }

}

void adjacencyList\_adjacencyMatrix(vector <vector <int>> &adjList, int\*\* &array, int &rows) {

  array = new int\*[rows];

  for (int i = 0; i < rows; i++) {

    array[i] = new int[rows];

  }

  for (int i = 0; i < rows; i++) {

    vector <int> smallPart = adjList[i];

    for (int j = 0; j < rows; j++) {

      int temp = checkNumber(smallPart, j);

      array[i][j] = temp;

    }

  }

}

void BFS(int \*\* &array, queue<int> localQueue, int rows) {

  cout << "\n---------- BFS ----------" << endl;

  int temp[rows];

  for (int i = 0; i < rows; i++) {

    temp[i] = 0;

  }

  localQueue.push(0);

  while (!localQueue.empty()) {

    int node = localQueue.front();

    localQueue.pop();

    temp[node] = 2;

    for (int j = 0; j < rows; j++) {

      if (array[node][j] == 1 && temp[j] == 0) {

        localQueue.push(j);

        temp[j] = 1;

      }

    }

    cout << node + 1 << endl;

  }

  cout << "-------------------------" << endl;

}

void DFS(int \*\* &array, stack <int> localStack, int rows) {

  cout << "\n---------- DFS ----------" << endl;

  int temp[rows];

  for (int i = 0; i < 7; i++) {

    temp[i] = 0;

  }

  localStack.push(0);

  while (!localStack.empty()) {

    int node = localStack.top();

    localStack.pop();

    if (temp[node] == 2) {

      continue;

    }

    temp[node] = 2;

    for (int j = rows - 1; j >= 0; j--) {

      if (array[node][j] == 1 && temp[j] != 2) {

        localStack.push(j);

        temp[j] = 1;

      }

    }

    cout << node + 1 << endl;

  }

  cout << "-------------------------" << endl;

}

int main() {

  int \*\* adjacencyMatrix = 0;

  int rows = 0;

  vector <vector <int>> adjacencyList;

  queue <int> bfsQueue;

  stack <int> dfsStack;

  addAdjacencyList(adjacencyList, rows);

  adjacencyList\_adjacencyMatrix(adjacencyList, adjacencyMatrix, rows);

  BFS(adjacencyMatrix, bfsQueue, rows);

  DFS(adjacencyMatrix, dfsStack, rows);

}