**Практична робота № 21-22**

*Виконав студент 35 групи*

*Рижій Ілля*

**Завдання №1.** Реалізувати задачі (пошук у глибину; пошук у ширину; алгоритм Дейкстри)*.*

*// Пошук у ширину*

#include <iostream>

#include <queue> // черга

using namespace std;

int main()

{

queue<int> Queue;

int mas[7][7] = {

{ 0, 1, 1, 0, 0, 0, 1 }, // матрицяя суміжності

{ 1, 0, 1, 1, 0, 0, 0 },

{ 1, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 1, 0, 0 },

{ 0, 0, 0, 1, 0, 1, 0 },

{ 0, 0, 0, 0, 1, 0, 1 },

{ 1, 0, 0, 0, 0, 1, 0 } };

int nodes[7]; // вершини графа

for (int i = 0; i < 7; i++)

nodes[i] = 0; // всі вершини рівні 0

Queue.push(0); // поміщаемо в чергу першу вершину

while (!Queue.empty())

{ // пока черга не порожня

int node = Queue.front(); // дістаємо вершину

Queue.pop();

nodes[node] = 2; // відмічаемо її як відвідану

for (int j = 0; j < 7; j++)

{ // перевіряємо для неї всі суміжні вершини

if (mas[node][j] == 1 && nodes[j] == 0)

{ // якщо вершина суміжна і не знайдена

Queue.push(j); // додаємо її в чергу

nodes[j] = 1; // відмічаємо вершину як знайдену

}

}

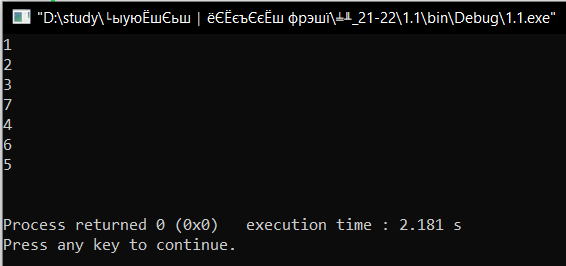
cout << node + 1 << endl; // виводимо номер вершини

}

cin.get();

return 0;

}



*// Пошук у глибину*

#include <iostream>

#include <stack> // стек

using namespace std;

int main()

{

stack<int> Stack;

int mas[7][7] = {

{ 0, 1, 1, 0, 0, 0, 1 }, // матрица смежности

{ 1, 0, 1, 1, 0, 0, 0 },

{ 1, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 1, 0, 0 },

{ 0, 0, 0, 1, 0, 1, 0 },

{ 0, 0, 0, 0, 1, 0, 1 },

{ 1, 0, 0, 0, 0, 1, 0 } };

int nodes[7]; // вершины графа

for (int i = 0; i < 7; i++) // исходно все вершины равны 0

nodes[i] = 0;

Stack.push(0); // помещаем в очередь первую вершину

while (!Stack.empty())

{ // пока стек не пуст

int node = Stack.top(); // извлекаем вершину

Stack.pop();

if (nodes[node] == 2) continue;

nodes[node] = 2; // отмечаем ее как посещенную

for (int j = 6; j >= 0; j--){ // проверяем для нее все смежные вершины

if (mas[node][j] == 1 && nodes[j] != 2){ // если вершина смежная и не обнаружена

Stack.push(j); // добавляем ее в cтек

nodes[j] = 1; // отмечаем вершину как обнаруженную

}

}

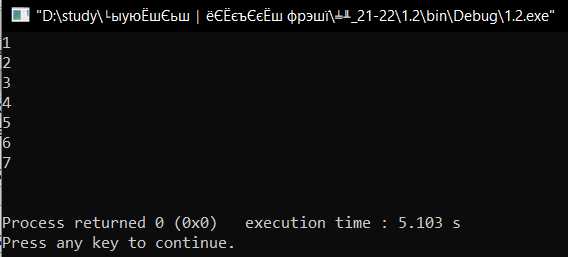
cout << node + 1 << endl; // выводим номер вершины

}

cin.get();

return 0;

}



*// Алгоритм Дейкстри*

//#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define SIZE 6

int main()

{

int a[SIZE][SIZE]; // матриця зв’язків

int d[SIZE]; // мінімальна відстань

int v[SIZE]; // відвідані вершини

int temp, minindex, min;

int begin\_index = 0;

system("cls");

// Ініціалізація матриці зв’язків

for (int i = 0; i<SIZE; i++)

{

a[i][i] = 0;

for (int j = i + 1; j<SIZE; j++) {

printf("vvedte vidctani %d - %d: ", i + 1, j + 1);

scanf("%d", &temp);

a[i][j] = temp;

a[j][i] = temp;

}

}

// Виведення матриці зв’язків

for (int i = 0; i<SIZE; i++)

{

for (int j = 0; j<SIZE; j++)

printf("%5d ", a[i][j]);

printf("\n");

}

//Ініціалізація вершин і відстаней

for (int i = 0; i<SIZE; i++)

{

d[i] = 10000;

v[i] = 1;

}

d[begin\_index] = 0;

// Крок алгоритму

do

{

minindex = 10000;

min = 10000;

for (int i = 0; i<SIZE; i++)

{ // Якщо вершину ще не обійшли і вага менше min

if ((v[i] == 1) && (d[i]<min))

{ // Переприсвоюємо значення

min = d[i];

minindex = i;

}

}

// Додаємо знайдену мінімальну вагу

// до поточної ваги вершини

// і порівнюємо з поточною мінімальною вагою вершини

if (minindex != 10000)

{

for (int i = 0; i<SIZE; i++)

{

if (a[minindex][i] > 0)

{

temp = min + a[minindex][i];

if (temp < d[i])

{

d[i] = temp;

}

}

}

v[minindex] = 0;

}

} while (minindex < 10000);

// Виведення найкоротших відстаней до вершин

printf("\n naikorotsha vidstan do vershin: \n");

for (int i = 0; i<SIZE; i++)

printf("%5d ", d[i]);

// Відновлення шляху

int ver[SIZE]; // масив відвіданих вершин

int end = 4; // індекс кінцевої вершини = 5 - 1

ver[0] = end + 1; // початковий елемент - кінцева вершина

int k = 1; // індекс попередньої вершини

int weight = d[end]; // вага кінцевої вершини

while (end != begin\_index) // поки не дійшли до початкової вершини

{

for (int i = 0; i<SIZE; i++) // переглядаємо всі вершини

if (a[end][i] != 0) // якщо є зв’язок

{

int temp = weight - a[end][i]; // визначаємо вагу шляху з попередньої вершини

if (temp == d[i]) // якщо вага співпала з розрахованою

{ // отже з цієї вершини і був перехід

weight = temp; // зберігаємо нову вагу

end = i; // зберігаємо попередню вершину

ver[k] = i + 1; // і записуємо її у масив

k++;

}

}

}

// Виведення шляху (початкова вершина розмістилась у кінці масиву из k елементів)

printf("\n Vivedenny naikorotshogo shlyhu\n");

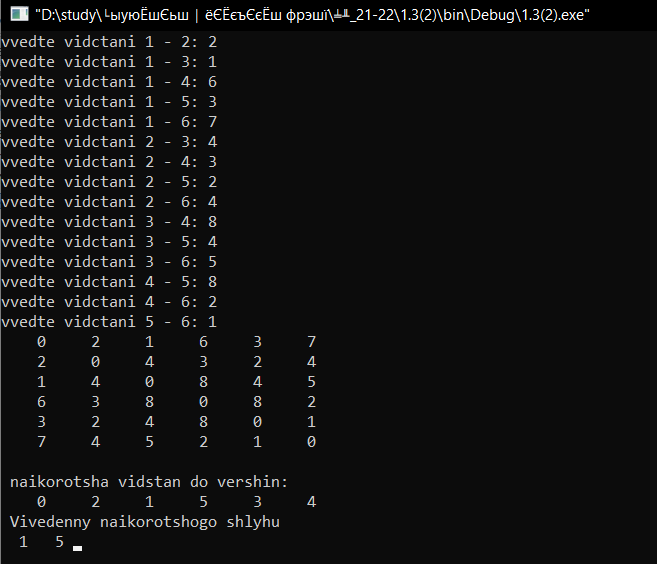
for (int i = k - 1; i >= 0; i--)

printf("%3d ", ver[i]);

getchar(); getchar();

return 0;

}



**Завдання №2.** Скласти програму, яка реалізує обхід графа (через рекурсію).

#include <iostream>

using namespace std;

int arr[7][7] = {

{ 0, 1, 1, 0, 0, 0, 1 }, // матрица смежности

{ 1, 0, 1, 1, 0, 0, 0 },

{ 1, 1, 0, 0, 0, 0, 0 },

{ 0, 1, 0, 0, 1, 0, 0 },

{ 0, 0, 0, 1, 0, 1, 0 },

{ 0, 0, 0, 0, 1, 0, 1 },

{ 1, 0, 0, 0, 0, 1, 0 } };

int nodes[7]; // вершины графа

void search(int st, int n)

{

int r;

cout << st + 1 << " ";

nodes[st] = 1;

for (r = 0; r < n; r++)

if ((arr[st][r] != 0) && (nodes[r] == 0))

search(r, n);

}

int main()

{

for (int i = 0; i < 7; i++) // все вершины равны 0

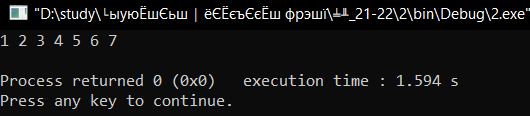
nodes[i] = 0;

search(0, 7);

cin.get();

return 0;

}



**Висновок:** під час практичної роботи закріпили знання про графи і алгоритми їх обробки.