**24-Laboratoriya mashg’uloti**

**Maʻlumotlar tarmoq tuzilmalari. Graf tushunchasi va uning ko‘rinishlari. Graflarni tasvirlash usullari.**

**1. Graflar nazariyasining asosiy tushunchalari**

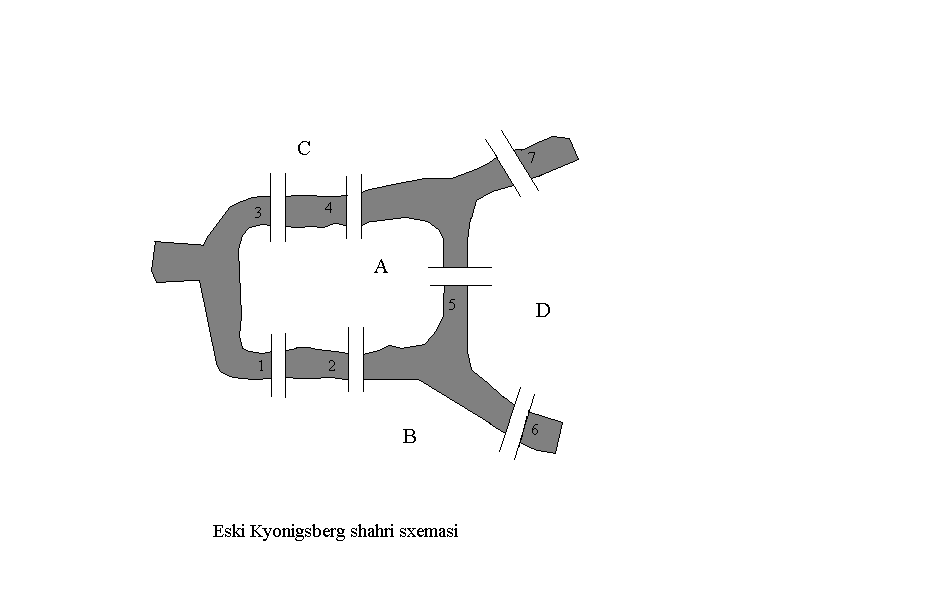
Matematik nazariyada va informatikada graf — bu tugunlar(uchlar)dan iborat bo'lgan bo'sh bo'lmagan to'plam va tugunlarni birlashtiruvchi yoylar majmuidir.

Graf - bu murakkab chiziqsiz ko'pbog'lamli dinamik tuzilma bo'lib, murakkab ob'ektlarning xususiyatlari va munosabatlarini aks ettiradi.

Ob'ektlar tugun yoki graf uzellari ko'rinishida va munosabatlar yoy yoki yo'naltirilgan qirralar kabi ifodalanadi.

«Graf» tushunchasini birinchi marotaba 1936 yil vengriya matematigi Denni Kyonig kiritgan. Lekin graflar nazariyasi bo'yicha 1-ish Leonard Eylerga tegishli bo'lgan va u 1736 yilda bajarilgan edi.

XVIII asrda mashhur shvetsariyalik matematik, mexanik va fizik Leonard Eyler (1707-1783 yy) Kyonigsberg ko’prigi haqidagi masalani yechish uchun birinchi marta graf tushunchasidan foydalanadi.



*1.-rasm. Eski Kyonigsberg shahri sxemasi*

Graflar nazariyasi diskret matematika fanining bir bo’limi bo’lib, unda masalalar yechimlari chizmalar shaklida izlanadi. Keyingi paytlarda turli xil diskret xususiyatlarga ega bo‘lgan xisoblash qurilmalarini loyihalashda graflarning ahamiyati yanada oshdi.

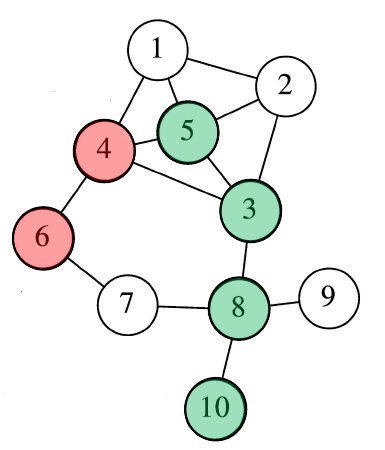
(*V, E*) sonlar juftligiga graf deyiladi, bu yerda *V* – ixtiyoriy bo`sh bo`lmagan to`plam, *E* esa http://abc.vvsu.ru/Books/l_diskrmat3/obj.files/image008.gifning qism to`plami http://abc.vvsu.ru/Books/l_diskrmat3/obj.files/image010.gif, bunda http://abc.vvsu.ru/Books/l_diskrmat3/obj.files/image008.gif *V* to`plam elementlarining tartiblanmagan juftliklari to`plami.

*V* – to`plam elementlari **grafning uchlari** deyiladi.

*E*  – to`plam elementlari esa **grafning qirralari** deyiladi.

Agar *V* chekli bo`lsa, graf**chekli** deyiladi, aks holda **cheksiz** **graf** deyiladi.

**Yo'l** (path) – bu bironta tugundan boshqa bir tugungacha bo'lgan yonma-yon joylashgan tugunlar ketma-ketligidir.



B

*2.-rasm. Grafning asosiy alomatlari*

Grafning uchlari va qirralari to`plamini mos ravishda http://abc.vvsu.ru/Books/l_diskrmat3/obj.files/image015.gif va http://abc.vvsu.ru/Books/l_diskrmat3/obj.files/image017.gif kabi belgilanadi. http://abc.vvsu.ru/Books/l_diskrmat3/obj.files/image015.gif ushbu to’plamda uchlar berilgan bo’ladi. http://abc.vvsu.ru/Books/l_diskrmat3/obj.files/image017.gifto’plamida esa qirallarning qushni uchlar juftligi bilan aniqlanadi.

Masalan:



Bu holda grafning grafik ko’rinishi quyidagicha bo’ladi (3.-rasm):

*3.-rasm. Grafga misol*

Qirra ikkita uch bilan aniqlanadi. Umumiy uchga ega bo`lgan ikkita qirra qo`shni hisoblanadi. Agar grafning ikkita uchi qirra bilan tutashtirilgan bo`lsa, bu uchlar**qo`shni uchlar** deyiladi. Grafning bir uchdan chiqqan ikki qirrasi **qo`shni qirralar** deyiladi. Agar grafda boshi va oxiri bitta tugunda tutashadigan qirra mavjud bo'lsa, unga **ilmoqli** qirra deyiladi.

1 uch

2 uch

qirra

*4.-rasm. Qirra tushunchasi*

Agar grafda takroriy (karrali) qirralar mavjud bo`lsa, bunday grafga **multigraf** deyiladi. Agar grafda karrali qirralar bilan birga uchni o`z-o`zi bilan tutashtiruvchi ilmoqlar ham mavjud bo`lsa, bunday grafga **psevdograf**deyiladi (5.-rasm).

A)

B)

*5.-rasm. A) multigraf; B) psevdograf*

Ixtiyoriy tugundan boshqa bironta tugunga murojaat mavjud va murojaat ikki tomonlama bo’lsa, bu holda bunday graf **yo’naltirilmagan** **graf** (graph) deyiladi (6.-rasm. A).

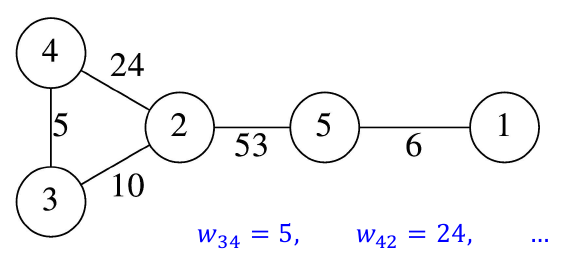
Agar graf tugunlari o'zaro bog'langan bo'lsa, lekin bu yoylar orqali munosabat faqat bir tomonlama bo'lsa, u xolda bunday graflar **yo'naltirilgan graflar** (oriented graph) deyiladi (6.-rasm. B).

A)

B)

*6.-rasm. A) yo’naltirilmagan graf; B) yo’naltirilgan graf*

**Og’irlikka (vaznga) ega bo’lgan graf** (weighted graph) – bu qirralari (yo’ylari) og’irliklari bilan berilgan graf hisoblandi. (i,j) qirraning og’irligi wij kabi belgilanadi (7.-rasm).

**

*7.-rasm. Og’irlikka ega bo’lgan graf*

Agar *V* to`plamning quvvati *n* ga teng bo`lsa, *n* soni **grafning tartibi**deyiladi. Agar *V* to`plamning quvvati *n* ga teng bo`lsa, E to`plamning quvvati *m* ga teng bo`lsa, graf **(n, m) graf** deyiladi.

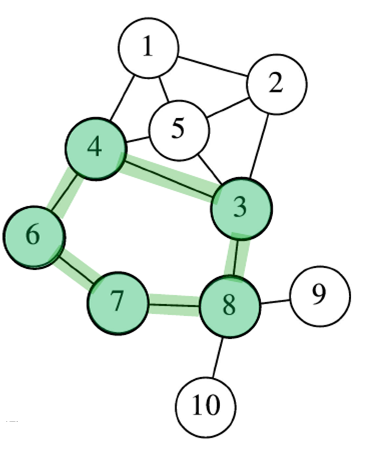
**Tugun darajasi** (vertex degree) – bu undan chiquvchi yoylar soni xisoblanadi. deg(7) = 2, deg(1) = 3

**Halqa** (cycle) – bu boshi va oxiri tutashuvchi tugundan iborat yo'l hisoblanadi: (4, 6, 7, 8, 3, 4) (1.2.8.-rasm).

G(V, E) grafda uning barcha tugunlari darajalari yig'indisi – juft, grafning qirralari sonining ikkilanganiga teng.



Tugunlar darajasiga nisbatan **juft yoki toq deyiladi**, agar ularning darajalari mos ravishda juft yoki toq qiymatga teng bo'lsa.



*8.-rasm. Grafning halqa va tugun darajasiga misol*

**To'liq graf** (complete graph) – bu istalgan tugunlari qo'shni bo'lgan graf xisoblanadi yani barcha tugunlar o'zaro birlashtirilgan. (9. a -rasm)

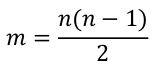
Grafni to'ldiruvchisi bu aynan bir tugunlar va aynan bir qirralardan tashkil topgan va mavjud grafni to'liq bo'lishini ta'minlovchi grafga aytiladi. (9. b -rasm)

a)

b)

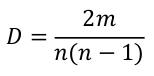
*9.-rasm.a) to’liq graf b) graf va uning to’ldiruvchisi*

To'liq, yo'naltirilmagan grafda qirralar soni quyidagi formula (1) orqali aniqlanadi:

 (1)

qaerda n – yoylar(tugunlar) soni.

D grafning to'yinganligi (density) grafning qirralrining tugunlar nisbatiga to’liqlik munosabat koefitsientini belgilaydi va quyidagi formula (2) orqali aniqlanadi:

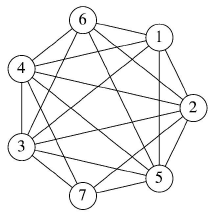
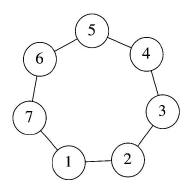
 (2)

qaerda n – grafning tugunlar soni, m – grafning qirralar soni.

Grafning to'yinganligi koefitsientiga qarab ikki hil graf ko’rinishi aniqlash mumkin: to'yingan graf va siyrak graf (10.-rasm).

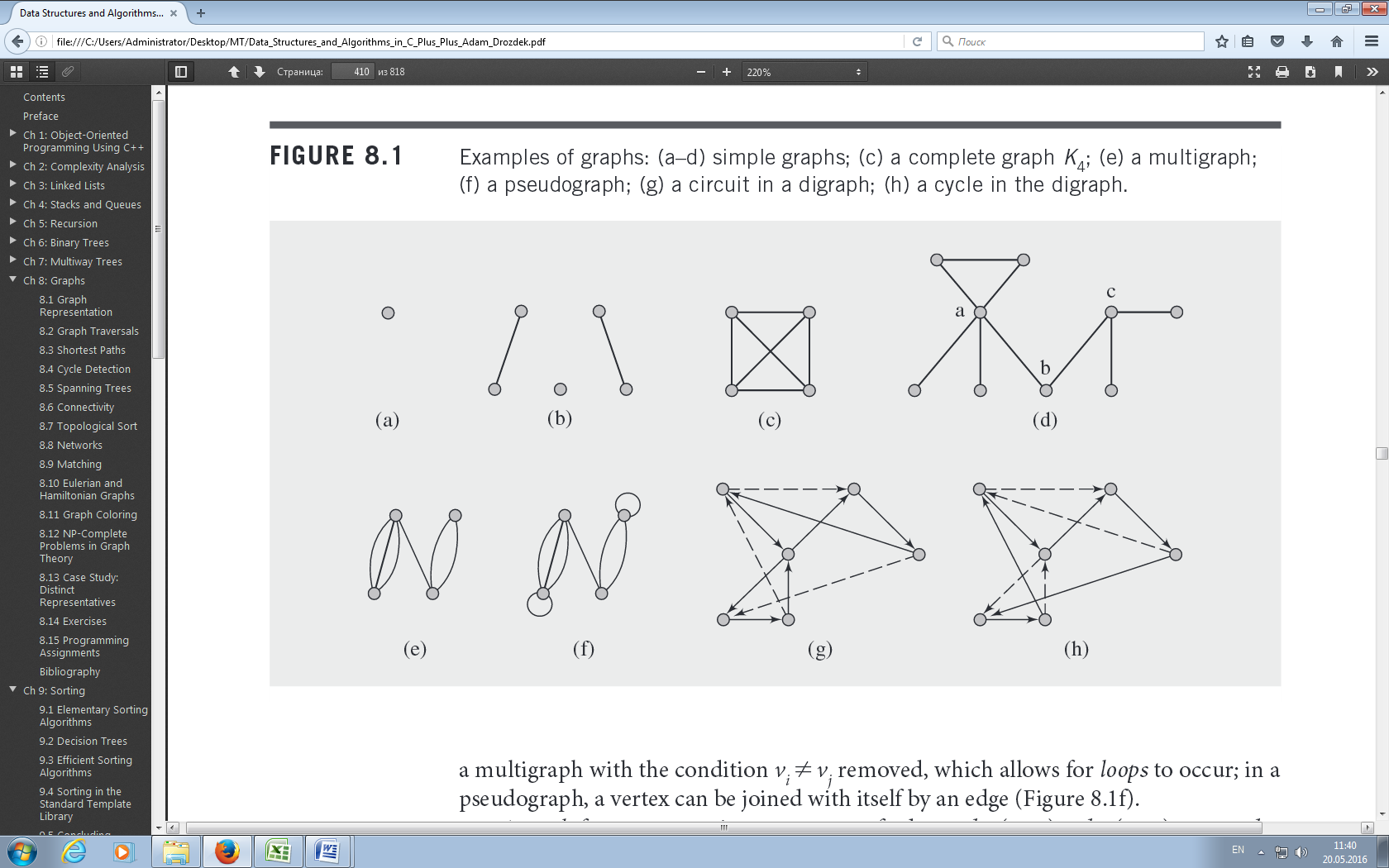
**To'yingan graf** (dense graph) – bu qirralar soni bo'lishi mumkin bo'lgan maksimalga teng bo'lgan graf xisoblanadi. (D>0.5)

**Siyrak graf** (sparse graph) – bu qirralari soni tugunlar soniga yaqin bo'lgan grafdir. (D<0.5)

a) b) 

*10-rasm.a) to'yingan graf (D=0.9) b) siyrak graf (D=0.33)*

Quyida turli ko’rinishdagi graflar keltirilgan.



11-rasm.a-d oddiy graf, c-to’liq graf, e-multigraf, f-psevdograf, g-digrafdagi zanjir, h-digrafda xalqa.

**2. Graflarni ifodalash usullari**

Yo’naltirilmagan, yo’naltirilgan va o’girlikka ega bo’lgan graflarni kompyuter dasturlash tillari hotirasida ifodalash, ya'ni xotirada tashkil etish uchun statik tuzilmasi matritsadan yoki dinamik tuzilmasi ro’yxatlardan foydalanish mumkin. Har qanday masalalarida har bitta usulining o’zining afzalligi va kamchiliklariga egadir. Yo’naltirilmagan, yo’naltirilgan va o’girlikka ega bo’lgan graflarni ifodalash uchun har usulining o’zining qoida asosida shakllanadi. Shunday to’rtta usullarga to’xtalib o’tamiz:

* Qo'shma matritsa (adjacency matrix);
* Intsidientlik matritsa (incidence matrix);
* Qo'shnilik ro'yxati (adjacency list);
* Qirralar ro'yxati (edges list).

G grafning **qo'shma matritsasi** bu n-o'lchamli A kvadrat matritsa bo'lib,

*graf uchun:*

Aij = 1 agar i va j tugunlar qirra bilan birlashtirilgan bo'lsa

Aij = 0 agar i va j tugunlar o’rtasida qirra mavjud bo'lmasa

*orgraf uchun:*

Aij = 1 agar i tugundan j tugunga yoy mavjud bo'lsa

Aij = 0 agar i va j tugunlarda yoy tugallanmagan bo'lsa

*vaznga ega graf uchun:*

Aij = Wij agar i va j tugunlar qirra (yoy) bilan birlashtirilgan bo'lsa

Aij = ∞ agar i va j tugunlar qirra (yoy) mavjud bo’lmasa

Qo'shma matritsa asosiy diagonaliga semmitrik bo’ladi, agar yo’naltirilmagan grafni ifodalasa, orgraflarda esa nosimmetrik bo’ladi.

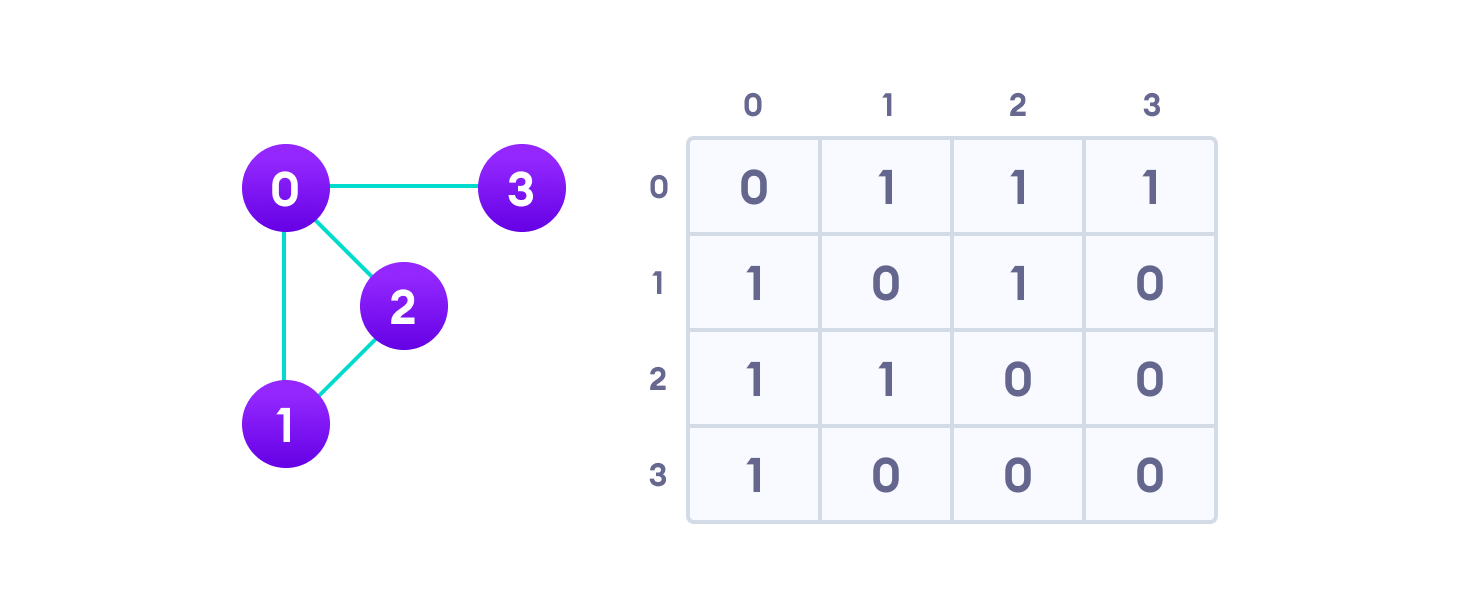
Qo'shma matritsaning qulaylik tomonlari quyidagilarda:

* Qirra(yoy) qushish va o’chirish oson;
* Tugunlar qo’shniligini tekshirish.

Qo'shma matritsaning noqulayliklari esa quyidagicha:

* Tugunlarni kiritish yoki o’chirish;
* Siyrak graflar bilan ishlash.

Quyidagi rasmda grafik va unga teng keladigan qo'shni matritsa ko'rsatilgan.



#include <iostream>

using namespace std;

class Graph {

private:

bool\*\* adjMatrix;

int numVertices;

public:

Graph(int numVertices) {

this->numVertices = numVertices;

adjMatrix = new bool\*[numVertices];

for (int i = 0; i < numVertices; i++) {

adjMatrix[i] = new bool[numVertices];

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

adjMatrix[i][j] = false;}}

void addEdge(int i, int j) {

adjMatrix[i][j] = true;

adjMatrix[j][i] = true;}

void removeEdge(int i, int j) {

adjMatrix[i][j] = false;

adjMatrix[j][i] = false;}

void toString() {

for (int i = 0; i < numVertices; i++) {

cout << i << " : ";

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

cout << adjMatrix[i][j] << " ";

cout << "\n";}}

~Graph() {

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

delete[] adjMatrix[i];

delete[] adjMatrix;}};

int main() {

Graph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.toString();

}

Natija:

0:0 1 1 0

1:1 0 1 0

2:1 1 0 1

3:0 0 1 0

G grafning **intsidientlik matritsasi** bu n-satr(tugunlar) va m-ustunlar (qirralar)dan tashkil topgan B matritsa bo'lib, unda:

*graf uchun:*

Bij = 1 agar i tugun j qirra bilan to'qnashgan bo'lsa

Bij = 0 agar i tugun j qirra bilan to'qnashmagan bo'lsa

*orgraf uchun:*

Bij = -1 agar i tugun j yoyning boshi bo'lsa

Bij = 0 agar i tugun j yoy bilan to'qnashmagan bo'lsa

Bij = 1 agar i tugun j yoyning oxiri bo'lsa

*vaznga ega graf uchun:*

Bij = ±Wij agar i tugun yoy boshi(oxiri) bo'lsa

Bij = 0 agar i tugun j yoy bilan to'qnashmagan bo'lsa

Intsidientlik matritsaning qulaylik tomonlari quyidagilarda:

* Qirra(yoy) o’lchamini yoki yo’nalishini o’zgartirish;
* Qirra(yoy)larni qushish yoki o’chirish;
* To’qnashuv(intsidientlik)ni tekshirish.

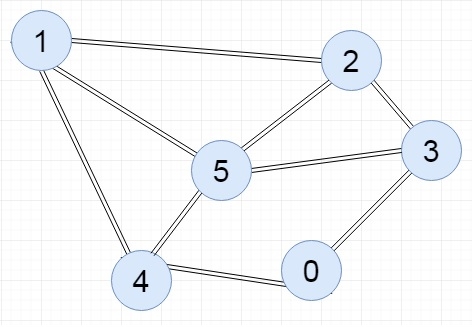
Intsidientlik matritsaning noqulayliklari esa quyidagicha:

* Tugunlarni qushish yoki o’chirish;
* Siyrak graflar bilan ishlash.

**Adjacency Matrix**

• Incidene matritsasi vakili u hisoblanayotganda O (Vx E) bo'sh joyni oladi. To'liq grafik uchun qirralarning soni V (V-1) / 2 bo'ladi. Shunday qilib, Incidene matritsasi xotirada katta joy egallaydi

**Kirish**



**Chiqish**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **E0** | **E1** | **E2** | **E3** | **E4** | **E5** | **E6** | **E7** | **E8** |
| **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| **3** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **4** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| **5** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |

**Algorithm**

**add\_edge(u, v)**

**Kirish -** chekkaning u va v {u, v}

**Chiqish -** G grafasining incedence matritsasi

**Dastlab, incidene matritsasi uchun ed\_cnt 0 ga teng chekka hisoblash mavjud.**

#include<iostream>

using namespace std;

int inc\_arr[20][20]; // incidence matritsasini ushlab turish uchun dastlabki massiv

int ed\_no = 0;

void displayMatrix(int v, int e) {

int i, j;

for(i = 0; i < v; i++) {

for(j = 0; j < e; j++) {

cout << inc\_arr[i][j] << " ";}

cout << endl;}}

void add\_edge(int u, int v) { // chekka raqami bilan matritsaga chekka qo'shish funktsiyasi

inc\_arr[u][ed\_no] = 1;

inc\_arr[v][ed\_no] = 1;

ed\_no++; // chekka raqamini oshirish

}

main(int argc, char\* argv[]) {

int v = 6; // grafada 6 ta tepalik mavjud

int e = 9; // grafada 9 ta chekka mavjud

add\_edge(0, 4);

add\_edge(0, 3);

add\_edge(1, 2);

add\_edge(1, 4);

add\_edge(1, 5);

add\_edge(2, 3);

add\_edge(2, 5);

add\_edge(5, 3);

add\_edge(5, 4);

displayMatrix(v, e);

}

**Qo'shnilik(qo'shni tugunlar) ro'yxati** – bu A[n] massiv bo'lib, A[i] xar bir elementi i tugun bilan qo'shni tugunlar ro'yxatini o'zida saqlaydi.

Qo'shnilik(qo'shni tugunlar) ro'yxati qulaylik tomonlari quyidagilarda:

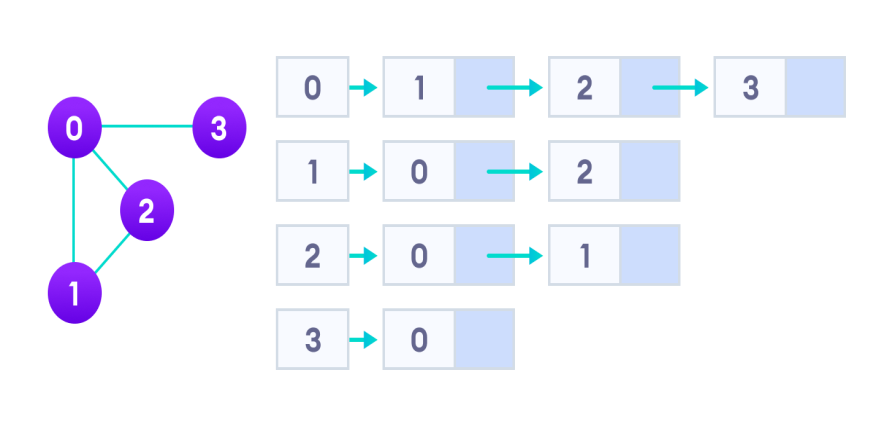
* Joriy (berilgan) tugunga qo’shni tugunni izlash;
* Tugun yoki qirra(yoy)larni qushish;
* Siyrak graflar bilan ishlash.

Qo'shnilik(qo'shni tugunlar) ro'yxati noqulayliklari esa quyidagicha:

* Qirra(yoy)ning mavjudligini tekshirish;
* Tugun yoki qirra(yoy)larni o’chirish.

**Adjacency List**

Quyida grafik va unga teng keladigan qo'shni ro'yxatning namoyishi ko'rsatilgan.



// Adjascency List C++

#include <bits/stdc++.h>

using namespace std;

// chekka qo'shish

void addEdge(vector<int> adj[], int s, int d) {

adj[s].push\_back(d);

adj[d].push\_back(s);}

// Grafikni chop eting

void printGraph(vector<int> adj[], int V) {

for (int d = 0; d < V; ++d) {

cout << "\n Tepalik "

<< d << ":";

for (auto x : adj[d])

cout << "-> " << x;

printf("\n");}}

int main() {

int V = 5;

// Grafik yaratish

vector<int> adj[V];

// Qirralarning qo'shish

addEdge(adj, 0, 1);

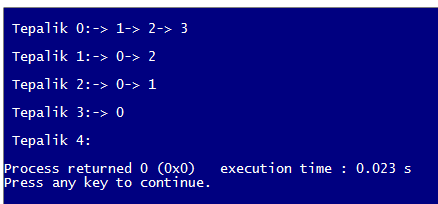
addEdge(adj, 0, 2);

addEdge(adj, 0, 3);

addEdge(adj, 1, 2);

printGraph(adj, V);

}



**Qirralar ro'yxati** – qirralarning qo'shni tugunlar juftliklaridan iborat chiziqli ro'yxatdir.

Qo'shnilik(qo'shni tugunlar) ro'yxati qulaylik tomonlari quyidagilarda:

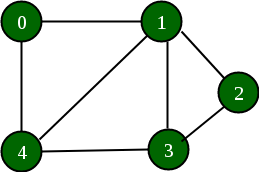
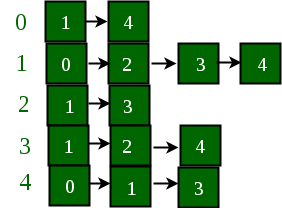
* Qirra(yoy)larni qushish yoki o’chirish;
* Yoylarning yuklanishi bo’yicha tartiblash;
* Siyrak graflar bilan ishlash.

Qo'shnilik(qo'shni tugunlar) ro'yxati noqulayliklari esa quyidagicha:

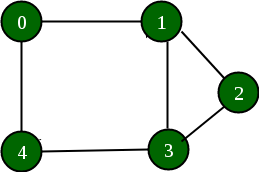
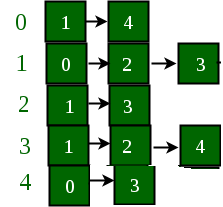
* Tugun va qirra(yoy)ning qo’shniligini aniqlash;
* Berilgan tugunga intsidient qirra(yoy)larni tanlash.

**Grafni qo'shni ro'yxatdagi qo'shilish va olib tashlash**

Quyida grafik va uning qo'shni ro'yxati ko'rsatilgan:

Agar 1 va 4 orasidagi chekkani olib tashlash kerak bo'lsa, yuqoridagi grafik va qo'shni ro'yxat quyidagiga aylanadi:

Yondashuv: G'oyani grafikani vektorlar qatori sifatida ko'rsatishdir, shunda har bir vektor tepalikning qo'shni ro'yxatini aks ettiradi.

Chegarani qo'shish: chekka qo'shish shu chekka bilan bog'langan ikkala tepalikni bir-birining ro'yxatiga kiritish orqali amalga oshiriladi. Masalan, (u, v) orasidagi chekka qo'shilishi kerak bo'lsa, u v ning vektorlar ro'yxatida va v u ning vektorlar ro'yxatida saqlanadi. (Orqaga surish)

Chegarani o'chirish: (u, v) orasidagi chekkani o'chirish uchun u ning qo'shni ro'yxati v topilguncha va undan o'chirilguncha o'tadi. Xuddi shu amal v. (O'chirish) uchun ham amalga oshiriladi.

// Yuqoridagi yondashuvni C ++ dasturida amalga oshirish

#include <bits/stdc++.h>

using namespace std;

// ga chekka qo'shish uchun yordamchi funktsiya

// yo'naltirilmagan grafik.

void addEdge(vector<int> adj[], int u, int v){

adj[u].push\_back(v);

adj[v].push\_back(u);}

// an-dagi chekkani o'chirish uchun yordamchi funktsiya

// yo'naltirilmagan grafik.

void delEdge(vector<int> adj[], int u, int v){

// Birinchi vektor ro'yxati orqali o'tish

// va undan ikkinchi elementni olib tashlash

for (int i = 0; i < adj[u].size(); i++) {

if (adj[u][i] == v) {

adj[u].erase(adj[u].begin() + i);

break;}}

// Ikkinchi vektor ro'yxati orqali o'tish

// va undan birinchi elementni olib tashlash

for (int i = 0; i < adj[v].size(); i++) {

if (adj[v][i] == u) {

adj[v].erase(adj[v].begin() + i);

break;}}}

// Qo'shni ro'yxatni chop etish uchun yordamchi funktsiya

// grafani aks ettirish

void printGraph(vector<int> adj[], int V){

for (int v = 0; v < V; ++v) {

cout << "vertex " << v << " ";

for (auto x : adj[v])

cout << "-> " << x;

printf("\n");}

printf("\n");}

int main(){

int V = 5;

vector<int> adj[V];

// Misol rasmda ko'rsatilgandek chekka qo'shish

addEdge(adj, 0, 1);

addEdge(adj, 0, 4);

addEdge(adj, 1, 2);

addEdge(adj, 1, 3);

addEdge(adj, 1, 4);

addEdge(adj, 2, 3);

addEdge(adj, 3, 4);

// adjacency matrix chiqarish

printGraph(adj, V);

// edge ni o'chirish (1, 4)

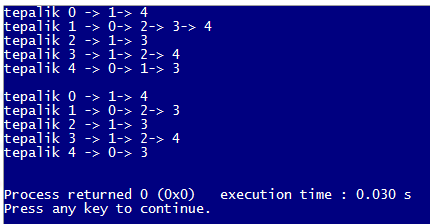
// misol rasmda ko'rsatilgandek

delEdge(adj, 1, 4);

// adjacency matrix ni chiqarish

printGraph(adj, V);

return 0;}



**3. Graflarda ko'rik o'tkazish**

**Grafni ko'rikdan o'tkazish** (Graph traversal) – bu berilgan tugundan boshlab barcha tugunlarni bir martadan ko'rib chiqish amalidir.

Ko’rikdan o’tkazish ikkita usuli mavjud:

Chuqurligiga (tubiga) qarab qidirish (Depth-First Search – DFS)

Kengligiga (eniga) qarab qidirish (Breadth-First Search – BFS)

Bu usullar berilgan X tugundan boshlab bironta konteynerni qo'llagan holda barcha tugunlarni ko'rib chiqadi.

Chuqurlikka qarab ko'rishda *stek tuzilmasi* qo'llaniladi.

Kenglikka qarab ko'rishda esa *navbat tuzilmasidan* foydalaniladi.

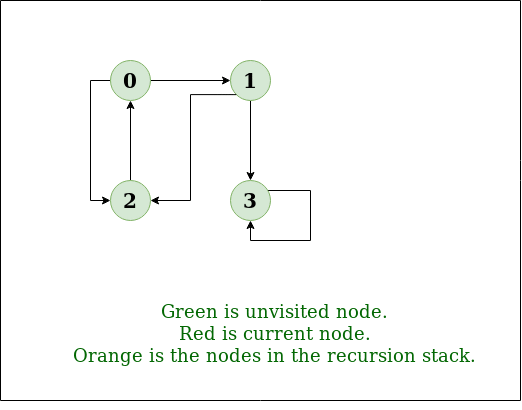
Tubiga qarab ko’rikni o’tqazish psevdokodi quyidagicha amalga oshiriladi

Kirish: n = 4, e = 6

2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Chiqish: tepalikdan DFS 2: 2 0 1 3

DFS diagrammasi:



Algoritm:

Tugun va tashrif buyurilgan qator indeksini oladigan rekursiv funktsiyani yarating.

Joriy tugunni tashrif buyurgan sifatida belgilang va tugunni chop eting.

Barcha qo'shni va belgilanmagan tugunlarni aylanib o'ting va qo'shni tugun ko'rsatkichi bilan rekursiv funktsiyani chaqiring.

// dan DFS o'tkazilishini chop etish uchun C ++ dasturi

// berilgan grafadagi berilgan tepalik

#include <bits/stdc++.h>

using namespace std;

// Grafik klassi yo'naltirilgan grafikani ifodalaydi

// qo'shni ro'yxatidan foydalanib

class Graph {

int V; // Tepaliklar soni

// o'z ichiga olgan qatorga ko'rsatgich

// qo'shni ro'yxatlar

list<int>\* adj;

// DFS tomonidan qo'llaniladigan rekursiv funktsiya

void DFSUtil(int v, bool visited[]);

public:

Graph(int V); // Konstruktor

// grafikka chekka qo'shish funktsiyasi

void addEdge(int v, int w);

// tepaliklarning DFS o'tishi

// v dan foydalanish mumkin

void DFS(int v);};

Graph::Graph(int V){

this->V = V;

adj = new list<int>[V];}

void Graph::addEdge(int v, int w){

adj[v].push\_back(w); }// V ning ro'yxatiga w ni qo'shing.

void Graph::DFSUtil(int v, bool visited[]){

// Joriy tugunni tashrif buyurgan sifatida belgilang va

// uni chop eting

visited[v] = true;

cout << v << " ";

// Qo'shni bo'lgan barcha tepaliklar uchun takrorlang

// ushbu tepalikka

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFSUtil(\*i, visited);}

// tepaliklarning DFS oralig'ida v ga erishish mumkin.

// Bu rekursiv DFSUtil () dan foydalanadi

void Graph::DFS(int v){

// Barcha tepaliklarni tashrif buyurilmagan deb belgilang

bool\* visited = new bool[V];

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

visited[i] = false;

// Rekursiv yordamchi funktsiyani chaqiring

// DFS-ni bosib o'tishni chop etish uchun

DFSUtil(v, visited);}

int main(){

// Yuqoridagi diagrammada berilgan grafikani yarating

Graph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

cout << "Quyidagi birinchi traversal chuqurligi"

" (tepalik 2 dan boshlab) \n";

g.DFS(2);

return 0;

}

Natija: Quyidagi birinchi traversal chuqurligi (tepalik 2 dan boshlab) : 2013

Kenglik bo'yicha birinchi qidiruv yoki grafik uchun BFS

Grafika uchun kenglik birinchi o'tish (yoki qidirish) daraxtning birinchi kenglik bo'ylab harakatlanishiga o'xshaydi ( [ushbu](https://www.geeksforgeeks.org/level-order-tree-traversal/) xabarning 2-uslubiga qarang ). Daraxtlardan farqli o'laroq, grafikalar tsikllarni o'z ichiga olishi mumkin, shuning uchun biz yana bitta tugunga kelishimiz mumkin. Tugunni bir necha marta qayta ishlashga yo'l qo'ymaslik uchun biz mantiqiy tashrif buyurilgan qatordan foydalanamiz. Oddiylik uchun barcha tepaliklarga boshlang'ich tepalikdan erishish mumkin deb taxmin qilinadi.  
Masalan, quyidagi grafada biz o'tishni 2-chi tepadan boshlaymiz. 0-chi cho'qqiga kelganda, uning barcha qo'shni tepalarini izlaymiz. 2 ham 0 ga teng bo'lgan vertexdir. Agar biz tashrif buyurgan tepalarni belgilamasak, u holda yana 2 qayta ishlanib, u tugamaydigan jarayonga aylanadi. Quyidagi grafikning birinchi kengligi 2, 0, 3, 1 ga teng.



Quyida berilgan manbadan oddiy Breadth First Traversal dasturining amallari keltirilgan.

// Berilganidan BFS o'tishini chop etish dasturi

// manba vertex. BFS (int s) tepaliklarni kesib o'tadi

// lardan foydalanish mumkin.

#include<iostream>

#include <list>

using namespace std;

// Bu sinf yordamida yo'naltirilgan grafikani ifodalaydi

// qo'shni ro'yxatining namoyishi

class Graph{

int V; // tepaliklar soni

// Yaqinlikni o'z ichiga olgan qatorga ko'rsatgich

// ro'yxatlar

list<int> \*adj;

public:

Graph(int V); // Konstruktor

// grafaga chekka qo'shish funktsiyasi

void addEdge(int v, int w);

// berilgan manbadan BFS o'tkazilishini chop etadi

void BFS(int s);};

Graph::Graph(int V){

this->V = V;

adj = new list<int>[V];}

void Graph::addEdge(int v, int w){

adj[v].push\_back(w);} // V ning ro'yxatiga w ni qo'shing.

void Graph::BFS(int s){

// Barcha tepaliklarni tashrif buyurilmagan deb belgilang

bool \*visited = new bool[V];

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

visited[i] = false;

// BFS uchun navbat yaratish

list<int> queue;

// Joriy tugunni tashrif buyurgan sifatida belgilang va uni ro'yxatdan o'tkazing

visited[s] = true;

queue.push\_back(s);

// 'i' barcha qo'shni joylarni olish uchun ishlatiladi

// tepalik tepalari

list<int>::iterator i;

while(!queue.empty()){

// Tepalikni navbatdan oling va uni chop eting

s = queue.front();

cout << s << " ";

queue.pop\_front();

// Dequiatsiyalangan barcha qo'shni tepaliklarni oling

// vertex s. Agar qo'shni joyga tashrif buyurilmagan bo'lsa,

// keyin tashrif buyurganligini belgilang va uni ro'yxatdan o'tkazing

for (i = adj[s].begin(); i != adj[s].end(); ++i){

if (!visited[\*i]){

visited[\*i] = true;

queue.push\_back(\*i);}}}}

// Grafik klassi usullarini sinash uchun haydovchi dasturi

int main()

{

// Yuqoridagi diagrammada berilgan grafikani yarating

Graph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

cout << "Quyidagi kenglik birinchi traversal "

<< "(tepalik 2 dan boshlab) \n";

g.BFS(2);

return 0;

}

Quyida birinchi kenglik bo'ylab o'tish (2 vertikaldan boshlab)

2 0 3 1

### Interfeysli grafik sinflar

|  |  |
| --- | --- |
| Color | Chiziq, matn yaratish va shaklni to‗ldirish uchun  foydalaniladi |
| Line\_style | Chiziq chizish uchun foydalaniladi |
| Point | Window sinfi obyekti ichida va ekranida joy borligini  tekshiruvchi vazifalar uchun foydalaniladi |
| Line | Point sinfi ikkita obyekti ekranida ko‗ringan chiziqlarni  kesish |
| Open\_polyline | Point sinfi obyekti ketma-ketligida aniqlangan kesilgan  chiziqlarni bir biri bilan bog‗lash ketma-ketligi |
| Closed\_polyline | Open\_polyline sinfiga o‗xshash, lekin farqi shundaki chiziqlar kesimi Point sinfi oxirgi obyektini birinchisi  bilan bog‗laydi |
| Polygon | Closed\_polyline sinfi, bu yerda kesmalar hyech qachon |

|  |  |
| --- | --- |
|  | kesishmaydi |
| Text | Belgilar satri |
| Lines | Point sinfi aniqlagan kesmalar chizig‗i to‗plami |
| Rectangle | Tez va qulay tasvirlash uchun optimallashgan shakl |
| Circle | Radius va markazi aniqlangan aylana |
| Ellipse | Markazi va ikkita o‗qlari aniqlangan ellips |
| Function | Aniqlangan kesmada berilgan bitta o‗zgaruvchi  funksiyasi |
| Axis | Belgilangan koordinata o‗qi |
| Mark | Belgi bilan belgilangan nuqta (masalan, x yoki 0) |
| Marks | Belgilar bilan belgilangan nuqtalar ketma-ketligi  (masalan, x yoki 0) |
| Marked\_polyline | Belgilar bilan belgilangan Open\_polyline sinfi |
| Image | Rasmli fayllar tarkibi |

**Grafik foydalanuvchi interfeysi sinfi**

|  |  |
| --- | --- |
| Window | Grafik obyektlar aks etadigan ekran maydoni |
| Simple\_window | Next tugmachali oyna |
| Button | Tugmachani bosib biron bir funksiyani chaqirish mumkin  bo‗lgan oynadagi to‗g‗ri burchak |
| In\_box | Foydalanuvchi satrni kiritishi mumkin bo‗lgan oyna  maydoni |
| Out\_box | Satrni chiqarish mumkin bo‗lgan oyna maydoni |
| Menu | Button sinfi obyektlari vektori |

**Deykstra algoritmi**

Gollandiyalik olim Edsger Deykstra algoritmi grafning boshlang’ich berilgan tugunidan boshlab qolgan barcha tugunlargacha bo'lgan barcha eng qisqa yo'llarni topadi. Uning yordamida, agar barcha zarur ma'lumotlar berilgan bo'lsa, masalan, neft va shu kabi mahsulotlarni eksport qilish uchun bitta shahardan boshqa shaharlarning har biriga borish uchun qaysi yo'llar ketma-ketligini tanlash afzalroq ekanligini bilib olish mumkin. Ushbu usulning salbiy tomoni shundaki, manfiy vaznga ega bo’lgan qirralari mavjud bo'lgan graflarni qayta ishlash imkonining mavjud emasligi, ya'ni, masalan, ba'zi tizim birorta kompaniya uchun foydasiz bo'lgan marshrutlarni taqdim qilsa, u holda u bilan ishlash uchun Dijkstraning algoritmidan foydalanib bo’lmaydi.

Algoritmni dasturiy ta'minotini amalga oshirish uchun ikkita massiv kerak bo'ladi: mantiqiy toifadagi *visited* - tashrif buyurilgan tugunlar haqidagi ma'lumotlarni saqlash uchun va topilgan eng qisqa yo'llar kiritiladigan butun toifadagi - *distance*. G={V,E} graf berilgan bo’lsin. V to’plamga tegishli barcha tugunlar dastlab tashrif buyurilmagan deb belgilanadi, ya’ni *visited* massivining elementlariga *false* qiymat berib chiqiladi. Eng afzal yo’lni topish masalasi qaralyapti. *Distance* massivining har bir elementiga shunday qiymat beriladiki, ixtiyoriy potensial yo’ldan katta bo’lsin (odatda, bu qiymatni cheksiz katta qiymat deb qaraladi, ammo dasturda berilgan toifaning qiymatlar diapazonidagi eng katta qiymat sifatida olinadi). Boshlang'ich nuqta sifatida s tugun tanlanadi va unga nol yo'l belgilanadi: distance [s] = 0, chunki s-dan s-gacha hech qanday qirra yo'q (bu usulda ilmoqlar qaralmaydi).

Shundan keyin, barcha qo'shni tugunlar topiladi (s dan chiquvchi qirralar orqali) [ularni t va u deb belgilaylik] va ular birma-bir tekshirib ko'riladi, ya'ni s tugundan har bir tugungacha birma-bir marshrut bahosi hisoblanadi:

- distance[t]=distance[s]+ s va t orasidagi qirraning vazni;

- Distance[u]=distance[s]+ s va u orasidagi qirraning vazni.

Ehtimoldan xoli emaski, u yoki bu tugunga s dan bir qancha yo’llar bo’lishi mumkin. Shu sababli, *distance* massivida bu tugunga bo’lgan yo’lning vaznini qayta ko’rib chiqish kerak bo’ladi. Shunda kattaroq (nooptimal) qiymat yo’qotiladi va tugunga mos yo’lning vazniga kichikroq qiymat beriladi. s tugun bilan qo’shni bo’lgan va qarab chiqilgan tugunlar tashrif buyurilgan sifatida belgilab chiqiladi, yani visited[s]=true va natijada, s dan chiquvchi, minimal vaznga ega bo’lgan yo’l eltuvchi tugun faol element sifatida belgilab olinadi. Faraz qilamiz, s dan u gacha masofa t ga qaraganda qisqa bo’lsin. Kelib chiqadiki, u tugun faollashadi va yuqoridagi kabi uning qo’shnilari ( s dan tashqari) o’rganilib chiqiladi. u tugun tashrif buyurilgan deb belgilanadi: visited[u]=true, endi t tugun faollashadi va yuqoridagi prosedura uning uchun takrorlanadi. Deykstra algoritmi s tugundan borish mumkin bo’lgan barcha tugunlar tadqiq qilinmaguncha davom ettiriladi.

**Bellman-Ford algoritmi**

Algoritm tarixi uchta mustaqil matematiklar bilan bog'liq: Lester Ford, Richard Bellman va Edward Moore. Ford va Bellman algoritmni 1956 va 1958 yillarda nashr etishdi, Moore esa 1957 yilda taqdim qilgan. Va ba'zan uni Bellman-Ford-Moore algoritmi deb ham atashadi. Usul ba'zi vektorli-marshrutlash protokollarida, masalan, RIPda (Routing Information Protocol) qo'llaniladi. Deykstra algoritmi singari, Bellman-Ford algoritmi ham vaznga ega bo’lgan graflarda bitta tugundan qolgan barcha tugunlarga bo’lgan eng qisqa masofani aniqlashda ishlatiladi. Bu algoritm manfiy vaznga ega bo’lgan graflar bilan ishlashda ham qo’llanilishi mumkin (istisno holatlar ham mavjud).

s tugundan qolgan barcha tugunlargacha bo’lgan qisqa masofani Bellman-Ford algoritmidan foydalanib topish dinamik dasturlashtirish usulini qo’llash demakdir, ya’ni uni qism masalalarga ajratib, ularni yechimi orqali umumiy asosiy masalani hal qilishdir. Bunda qism masala bo’lib, bitta alohida qaralayotgan tugundan boshqasigacha eng qisqa yo’lni aniqlash masalasi hisoblanadi. Algoritm natijalarini saqlash uchun d[] bir o’lchovli massiv qabul qilamiz. Uning har bir i-elementida s gundan i-elementgacha qisqa masofa qiymatini saqlanadi (agar mavjud bo’lsa). Dastlab, d[] massiv elementlariga shartli sheksiz katta qiymat berib chiqiladi, d[s] ga 0 o’zlashtiriladi.

G={V, E}, n=|V|, m=|E| graf berilgan bo’lsin. Qo’shni tugunlarni v va u deb, (v,u) orasidagi qirrani w deb belgilab olamiz. Boshqacha aytganda, v tugundan chiquvchi va u tugunga kiruvchi qirra vazni w ga teng. U holda, Bellman-Ford algoritmining muhim qismi quyidagicha ko’rinishga ega bo’ladi:

* I=1 dan n-1 gacha bajaramiz выполнять  
  j=1 dan m gacha bajaramiz  
  agar d[v] + w(v, u) < d[u] bo’lsa, u holda  
  d[u] < d[v] + w(v, u)

Har bir n-qadamda d[] massiv elementlari qiymatlarini yashilashga harakat qilinadi: agar w(v,u) qirra vazni va d[v] element qiymati yig’indisi d[u] qiymaridan kichik bo’lsa, u holda bu qiymat d[u] ga o’zlashtiriladi.

*#include "stdafx.h"*  
*#include <iostream>*  
*#define inf 100000*  
using namespace std;  
struct Edges{  
int u, v, w;  
};  
const int Vmax=1000;  
const int Emax=Vmax\*(Vmax-1)/2;  
int i, j, n, e, start;  
Edges edge[Emax];  
int d[Vmax];  
*//Bellman-Ford algoritmi*  
void bellman\_ford(int n, int s)  
{  
int i, j;  
for (i=0; i<n; i++) d[i]=inf;  
d[s]=0;

for (i=0; i<n-1; i++)  
for (j=0; j<e; j++)  
if (d[edge[j].v]+edge[j].w<d[edge[j].u])  
d[edge[j].u]=d[edge[j].v]+edge[j].w;  
for (i=0; i<n; i++) if (d[i]==inf)  
cout<<endl<<start<<"->"<<i+1<<"="<<"Not";  
else cout<<endl<<start<<"->"<<i+1<<"="<<d[i];  
}  
*//asosiy funksiya*  
void main()  
{  
int w;  
cout<<"tugunlar soni > "; cin>>n;  
e=0;  
for (i=0; i<n; i++)  
for (j=0; j<n; j++)  
{  
cout<<"Вес "<<i+1<<"->"<<j+1<<" > "; cin>>w;  
if (w!=0)  
{

{  
edge[e].v=i;  
edge[e].u=j;  
edge[e].w=w;  
e++; } }  
cout<<"boshlang’ich tugun > "; cin>>start;  
cout<<"qisqa masofalar ro’yhati:";  
bellman\_ford(n, start-1);  
system("pause>>void"); }

Bu yerda graf soddalashtirilgan qirralar ro’yhati ko’rinishida ifodalangan va foydalanuvchi tomonidan vazn matrisasi kiritiladi. Bellman-Ford algoritmining asosiy qismi m\*(n-1) marta bajariladi, tashqi sik n-1 marta takrorlanadi, ichki sikl esa m marta. N-iterasiyadan voz kechish esa maqsadga muvofiqdir, chunki algoritm n-1 ta iterasiyada ham o’z vazifasini to’liq amalga oshira oladi. Ammo tashqi siklni n marta amalga oshirish grafda manfiy sikl mavjudligini aniqlashga imkon beradi.

**Floyd-Uorshell algoritmi**

Bu usulning nomlanishi 1962 yilda bir vaqtning o'zida kashf etgan ikkita amerikalik tadqiqotchilar Robert Floyd va Stiven Uorshell sharafiga qo’yilgan. Nomlanishning boshqa variantlari kamroq tarqalgan: Roy - Uorshall algoritmi yoki Roy - Floyd algoritmi. Roy - bu algoritmni hamkasblaridan 3 yil oldin (1959 yilda) ishlab chiqqan professorning familiyasidir, ammo uning kashfiyoti nashr qilimay qolgan. Floyd-Uorshell algoritmi – grafning har bir tugunigacha bo’lgan qisqa masofalarni hisoblashning dinamik algoritmidir. Bu metodni musbat va manfiy vaznga ega bo’lgan graflarda qo’llash mumkin, faqat manfiy sikllardan tashqari. Shu sababli oldin ko’rib o’tilgan Deykstra algoritmiga qaraganda umumiyroq hisoblanadi. Floyd-Uorshell algoritmini amalga oshirish uchun har bir tuguni 1 dan |V| gacha nomerlangan G=(V,E) grafning qo’shma matrisasi D[][] ni tashkil etamiz. Bu matrisa |V|x|V| o’lchamga ega bo’ladi va har bir D[i][j] elementga i dan j gacha bo’lgan qirralarning vazni o’zlashtiriladi. Algoritm bajarilishi mobaynida ushbu matrisa qayta yoziladi: uning har bir yacheykasiga i tugundan j tugungacha hisoblangan eng optimal, qisqa masofa yoziladi.Endi algoritmning asosiy qismini tuzishdan oldin, eng qisqa yo'llarning matritsasi tarkibini tushunish kerak. Uning har bir elementi D[i][j] mavjud bo'lgan marshrutlarning eng kichikini o'z ichiga olishi kerakligi sababli, biz darhol ayta olamizki, yagona tugundan uchun u nolga teng, hattoki pastadir (manfiy sikllar hisobga olinmaydi), shuning uchun asosiy diagonalning barcha elementlari (D[i][i]) ni 0 qilish kerak.

Diagonalda bo’lmagan 0 qiymatli elementlar (matrisaning i va j tugunlari o'rtasida bevosita qirra mavjud bo'lmagan joylarida nollar bo'lishi mumkin) ularning qiymatini iloji boricha o'zgartirish uchun biz ularni cheksizlik bilan tenglashtiramiz, bu dasturda bo'lishi mumkin, masalan, grafda mumkin bo'lgan maksimal yo'l, yoki shunchaki katta son. Algoritmning uchta - sikl, ifoda va shartli operatordan iborat asosiy qismi juda ixcham tarzda yoziladi:

* k=1 dan |V| gacha bajariladi
* i=1 dan |V| gacha bajariladi
* j=1 dan |V| gacha bajariladi
* agar D[i][k]+D[k][j]<D[i][j] bo’lsa, u holda D[i][j] ←D[i][k]+D[k][j]

i tugunidan j tugunigacha eng qisqa yo'l ular orqali va boshqa tugunlar to'plamidan o'tishi mumkin k∈(1, ..., |V|). i dan j gacha bo'lgan yo'l k tugundan o’tishi yoki o'tmasligi ham mumkin. Agar boshqa yo'l mavjud bo’lsa, u i dan k ga, keyin k dan j gacha o'tishini anglatadi, shuning uchun u qisqa yo'lning qiymati D[i][j] ni D[i][k] + D[k][j] yig'indi bilan almashtirish kerak.

Floyd-Worshell algoritmining to'liq kodini C ++ va Paskalda ko'rib chiqamiz va keyin u bajaradigan harakatlar ketma-ketligini batafsil tahlil qilamiz.

***C++ da dastur kodi:***

*#include "stdafx.h"*  
*#include <iostream>*  
using namespace std;  
const int maxV=1000;  
int i, j, n;  
int GR[maxV][maxV];

*//алгоритм Флойда-Уоршелла*  
void FU(int D[][maxV], int V)  
{  
int k;  
for (i=0; i<V; i++) D[i][i]=0;  
for (k=0; k<V; k++)  
for (i=0; i<V; i++)  
for (j=0; j<V; j++)  
if (D[i][k] && D[k][j] && i!=j)  
if (D[i][k]+D[k][j]<D[i][j] || D[i][j]==0)  
D[i][j]=D[i][k]+D[k][j];  
for (i=0; i<V; i++)  
{  
 for (j=0; j<V; j++) cout<<D[i][j]<<"\t";  
 cout<<endl;  
} }

*//главная функция*  
void main()  
{  
setlocale(LC\_ALL, "Rus");  
cout<<"Количество вершин в графе > "; cin>>n;  
cout<<"Введите матрицу весов ребер:\n";  
for (i=0; i<n; i++)  
for (j=0; j<n; j++)  
{  
cout<<"GR["<<i+1<<"]["<<j+1<<"] > ";  
cin>>GR[i][j];  
}  
cout<<"Матрица кратчайших путей:"<<endl;  
FU(GR, n);  
system("pause>>void");  
}

**Nazorat savollar.**

1. Graf nima va uning turlarini aytib bering.
2. Graflarni ko’rikdan o’tkazish algoritmlari.
3. Graflarni dasturda ifodalash usullari?

**Adabiyotlar.**

1. AdamDrozdek. Data structure and algorithms in C++. Fourth edition. 2013. Chapter 8. 391-490 betlar.
2. A computer science portal for geeks. <http://www.geeksforgeeks.org/data-structures/#Graph>
3. <http://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm>

**Topshriqlar**

1. Foydalanuvchidan vaznli yo’nalishsiz grafning uchlari va qovurg’alari sonini, mavjud qovurg’alarning ro’yhati va og’irligini so’rovchi;
2. Berilgan ma’lumotlar asosida grafning qo’shnilik matritsasini tashkil qiluvchi;
3. Garfning boshlang’ich va oxirgi uchlarini so’rovchi;
4. Ekranga berilgan uchlar orasidagi qisqa masofani va uning og’irligini chiqaruvchi;
5. Quyidagi graf asosida tekshirib ko’ruvchi:

