1. Eng keng tarqalgan graf algoritmlari

Graflar turli xil muammolarni modellashtirish uchun juda foydali ma’lumotlar tuzilmalari. Bu algoritmlar ijtimoiy tarmoq saytlari to‘g’ridan-to‘g’ri dasturlar mavjud. Ba’zi Umumiy Graf Algoritmlari

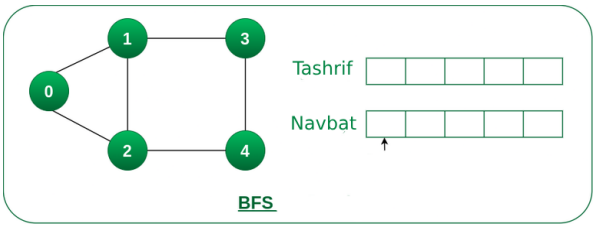
- Kenglik bo‘yicha qidiruv (Breadth First Search (BFS))
- Bo‘yi bo‘yicha qidiruv (Depth First Traversal (DFS))
- Dijkstra algoritmi

1.1 Kenglik bo‘yicha qidiruv (Breadth First Search)

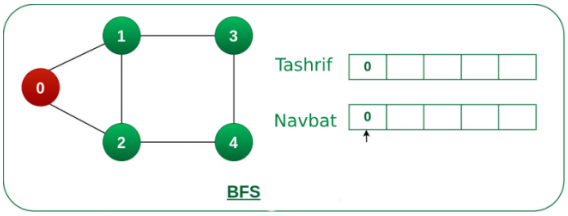
Kenglik bo‘yicha qidiruv (BFS) algoritmi bir qator mezonlarga javob beradigan tugun uchun graf ma’lumotlar tuzilishini qidirish uchun ishlatiladi. U grafning ildizidan boshlaydi va keyingi chuqurlik darajasidagi tugunlarga o‘tishdan oldin joriy chuqurlik darajasidagi barcha tugunlarga tashrif buyuradi. Buning uchun navbat ma'lumotlar tuzilmasi ishlatiladi. Joriy darajadagi barcha qo‘shni tashrif buyurilmagan tugunlar navbatga suriladi va joriy darajadagi tugunlar belgilanadi tashrif buyuradi va navbatdan chiqiladi.

Keling, quyidagi misol yordamida algoritmning ishlashini ko‘rib chiqamiz.

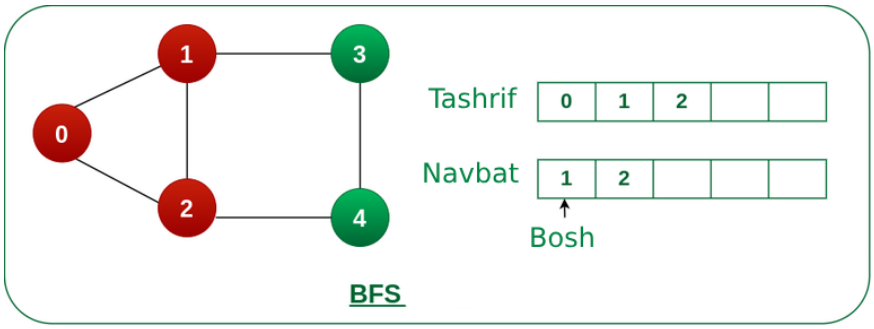
1-qadam: Dastlab navbat va tashrif buyurilgan massivlar bo‘sh.



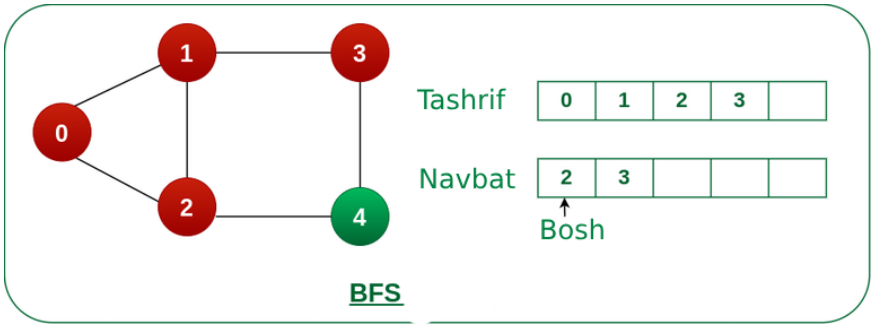
2-qadam: 0 tugunini navbatga suring va tashrif buyurganini belgilang.



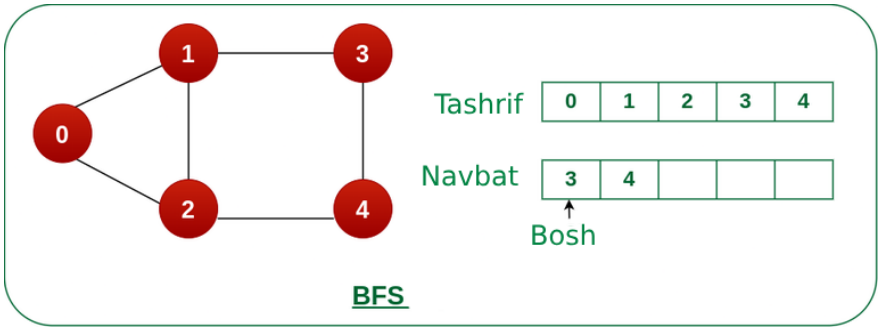
3-qadam: 0-tugunini navbat boshidan olib tashlang va tashrif buyurmagan qo‘shnilarga tashrif buyuring va ularni navbatga qo‘ying.



4-qadam: Navbatlarning old qismidan 1-tugunni olib tashlang va tashrif buyurilmagan qo‘shnilarga tashrif buyuring va ularni navbatga qo‘ying.

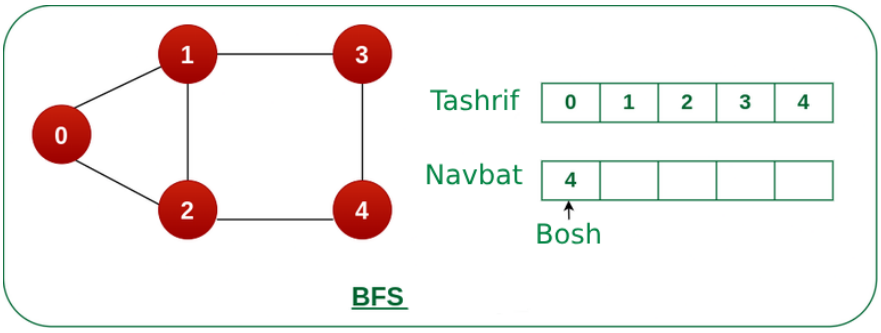


5-qadam: Navbatning old qismidan 2-tugunni olib tashlang va tashrif buyurilmagan qo‘shnilarga tashrif buyuring va ularni navbatga suring.

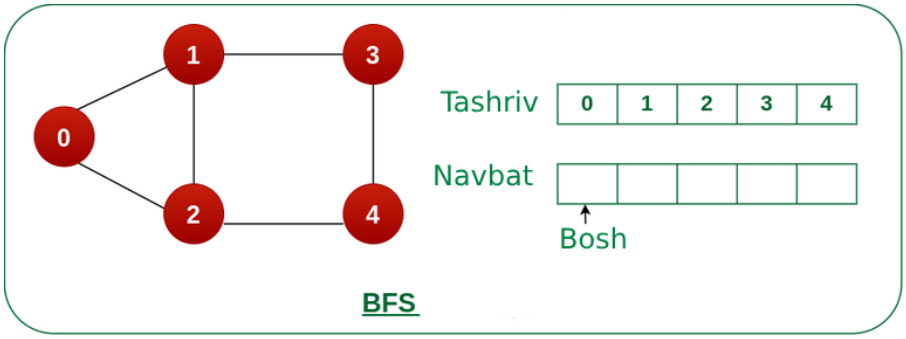


6-qadam: 3-tugunni navbat boshidan olib tashlang va tashrif buyurmagan qo‘shnilarga tashrif buyuring va ularni navbatga qo‘ying. Biz 3-tugunning barcha

qo‘shnilari tashrif buyirilanligini ko‘rib turibmiz, shuning uchun navbatning boshida joylashgan keyingi tugunga o‘tamiz.



4-tugunni navbatning old qismidan olib tashlang va tashrif buyurmagan qo‘shnilarga tashrif buyuring va ularni navbatga qo‘ying.



Endi navbat bo‘sh bo‘ladi, shuning uchun takrorlash jarayonini yakunlaymiz.BFS algoritmning C++ dasturlash tilida implementatsiaysi.

```
// BFS algoritmining c++ dasturlash kodi

#include <iostream>
using namespace std;

// Graf sinfini yaratish
class Graph {

        int V;

  // ro'yxat ko'rsatkichi
  vector<list<int> > adj;
```

```
public:
   // Konstruktor
   Graph(int V);

   // qirlarni birlashtirish funktsiyasi
   void addEdge(int v, int w);

   // BFS algoritmni chiqarish
   void BFS(int s);
};

Graph::Graph(int V)
{
   this->V = V;
   adj.resize(V);
}

void Graph::addEdge(int v, int w)
{
        adj[v].push_back(w);
}

void Graph::BFS(int s)
{
        vector<bool> visited;
   visited.resize(V, false);

   // BFS ga navbatga qo‘shish
   list<int> queue;

   // Hozirgi tugunni ko‘rsatish
   visited[s] = true;
   queue.push_back(s);

   while (!queue.empty()) {

                  s = queue.front();
        cout << s << " ";
        queue.pop_front();
```

```
                    for (auto adjacent : adj[s]) {
                if (!visited[adjacent]) {
                    visited[adjacent] = true;
                    queue.push_back(adjacent);
                }
            }
        }
    }

    // Bosh funktsiya
    int main()
    {
        // Berilgan diagrammadan grafni yaratish
        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 << "BFS algoritmning tugunlarga tashrifi "
             << "(2-tugunda boshlaganda) \n";
        g.BFS(2);

        return 0;
    }
```

Dastur natijasi:

```
BFS algoritmning tugunlarga tashrifi (2-tugunda boshlaganda)
2 0 3 1
```

Vaqtning murakkabligi:O(V+E), bu erda V tugunlar soni va E qirralarning soni.

Xotira murakkabligi: O (V)

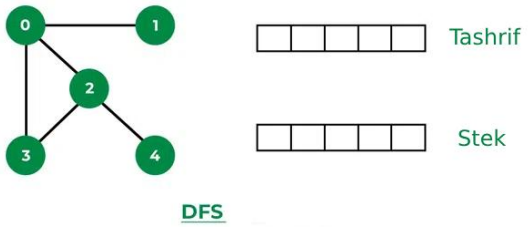
1.2 Bo‘yi bo‘yicha qidiruv (Depth First Traversal (DFS))

Bo‘yi bo‘yicha qidirish - bu daraxt yoki graf ma’lumotlar tuzilmalarini chetlab o‘tish orqali qidirish algoritmi. Algoritm ildiz tugunidan boshlanadi (graf holatida

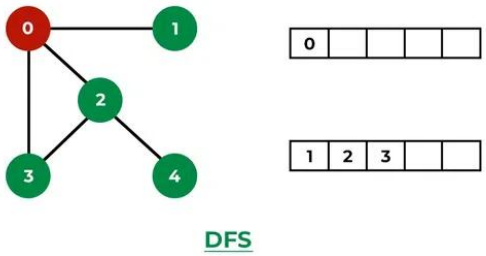
ba’zi o‘zboshimchalik bilan kod ildiz tugun sifatida tanlanadi) va orqaga qaytishdan oldin har bir tugun bo‘ylab iloji boricha uzoqroq o‘rganadi.

Keling, quyidagi rasm yordamida qidiruv algoritmini ko‘rib chiqamiz:

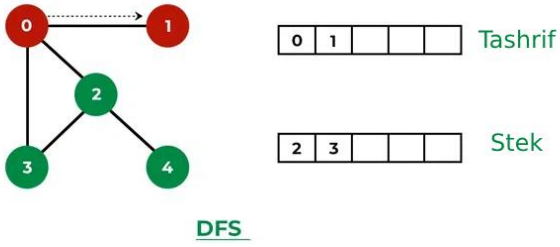
1- qadam: Boshida berilgan graftda tugunlarg’a hali tashriv buyirilmagan.



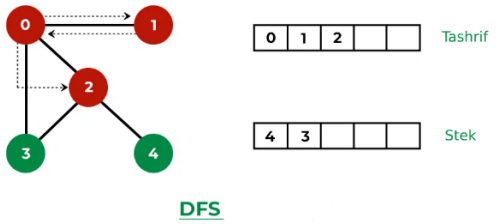
2-qadam: 0 ga tashrif buyuriladi va u tashrif buyurmagan qo‘shni tugunlarni stekka joylashtiradi.



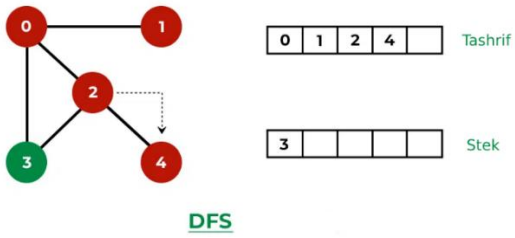
3-qadam: Endi 1-tugun stekning yuqori qismida joylashgan, shuning uchun 1-tugunga tashrif buyuriladi va uni stekdan chiqarib olinadi va tashrif buyurilmagan barcha qo‘shni tugunlarni stekka joylashtirladi.



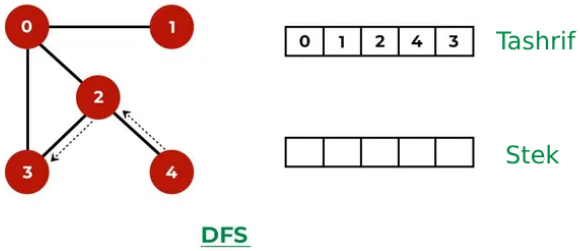
4-qadam: Endi 2-tugun stekning yuqori qismida joylashgan, shuning uchun 2-tugunga tashrif buyuring va uni stekdan chiqarib olinadi va tashrif buyurilmagan barcha qo‘shni tugunlarni (ya’ni 3, 4) stekka joylashtirladi.



5-qadam: Endi 4-tugun stekning yuqori qismida joylashgan, shuning uchun 4-tugunga tashrif buyurish kerak. Uni stekdan chiqarib olinadi va tashrif buyurilmagan barcha qo‘shni tugunlarni stekka joylashtirladi.



6-qadam: Endi 3-tugun stekning yuqori qismida joylashgan, shuning uchun 3-tugunga tashrif buyuring va uni stekdan chiqarib oling va tashrif buyurilmagan barcha qo‘shni tugunlarni stekka joylashtiring.



Endi Stack bo‘sh bo‘lib qoladi, ya’ni biz barcha tugunlarga tashrif buyurdik yani DFS algoritmi bilan barcha tugunlarga tashrif buyurdik.Berilgan DFS algoritmning C++ dasturlash tilida implementatsiyasi.

```
// C++ dasturlash tilida DFS algoritming implementatsiyasi
#include <bits/stdc++.h>
using namespace std;

// yo‘naltirlgan grafning sinfi
class Graph {
public:
   map<int, bool> visited;
   map<int, list<int> > adj;

   // Grafka qirra qo‘shish
   void addEdge(int v, int w);

   // tugunlarga tashriv
   void DFS(int v);
};

void Graph::addEdge(int v, int w)
{
   // tugunlarni listka joylash
   adj[v].push_back(w);
}

void Graph::DFS(int v)
{
   // Tshriv buyirilgan tugunni belgilash
   // va chiqarish
   visited[v] = true;
   cout << v << " ";

   list<int>::iterator i;
   for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
             DFS(*i);
}
```

```
// Bosh funktsiya
int main()
{
   // Create a graph given in the above diagram
   Graph g;
   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 << "DFS algoritmining tugunlarga tashrifi"
           " (2-tugundan boshlash) \n";

   // Funktsiyani chaqirish
   g.DFS(2);

   return 0;
}
```

Dastur natijasi:

```
DFS algoritmining tugunlarga tashrifi (2-tugundan boshlash)
2 0 1 3
```

Vaqt murakkabligi: O (V + E), bu erda V tepaliklar soni va E grafadagi qirralarning soni.

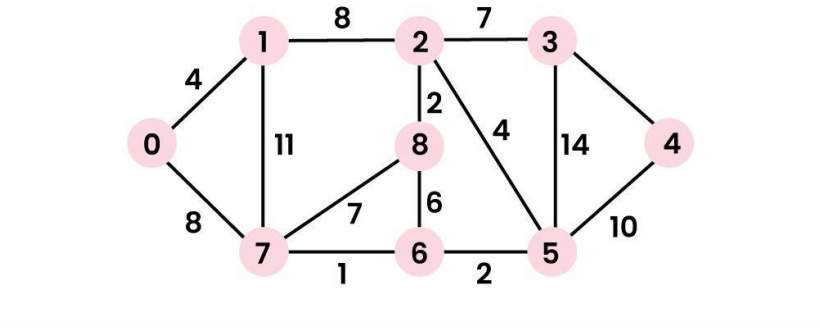
Xotira murakkabligi: O (V + E), DFS funktsiyasini interaktiv ravishda chaqirish uchun qo‘shimcha tashrif buyurilgan massiv va Stack hajmini talab qiladi.

1.3 Dijkstra algoritmi

Grafdagi o‘lchangan tugunlar hisobga olgan holda, berilgan grafdagi barcha boshqa tugunlarga manbadan eng qisqa yo‘llarni toping.

Eslatma: ushbu grafda teskari chekka mavjud emas. Misollar:

Kiritish: src = 0, graf quyida ko‘rsatilgan.



Dijkstra's Algorithm

Natija: 0 4 12 19 21 11 9 8 14

- 0 dan 1 gacha bo‘lgan masofa 4 ga teng 0->1.
- 0 dan 2 gacha bo‘lgan minimal masofa 12 ga teng. 0->1->2
- Minimal masofa 0 dan 3 gacha 19 ga teng. 0->1->2->3
- 0 dan 4 gacha bo‘lgan minimal masofa 21 ga teng. 0->7->6->5->4
- Minimal masofa 0 dan 5 gacha 11 ga teng . 0->7->6->5
- Minimal masofa 0 dan 6 gacha 9 ga teng. 0->7->6
- Minimal masofa 0 dan 7 gacha 8 ga teng. 0->7
- Minimal masofa 0 dan 8 gacha 14 ga teng. 0->1->2->8

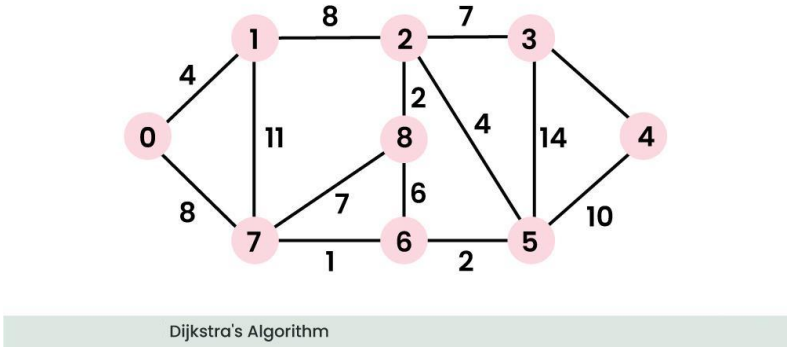
G’oya ildiz sifatida berilgan manba bilan SPTset (eng qisqa yo‘l daraxti) INF(cheksiz) hosil qilishdir. Ikki to‘plam bilan qo‘shnilik matritsani saqlang:

- bitta to‘plamda eng qisqa yo‘l daraxtiga kiritilgan tugunlar mavjud,
- boshqa to‘plamga hali eng qisqa yo‘l daraxtiga kiritilmagan tugunlar kiradi.

Algoritmning har bir bosqichida boshqa to‘plamda joylashgan (to‘plam hali kiritilmagan) va manbadan minimal masofaga ega bo‘lgan tepalik topiladi.

Dijkstra algoritmini tushunish uchun graf olish va manbadan barcha tugunlarga eng qisqa yo‘lni topish mumkin.

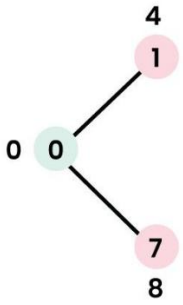
Quyida ko‘rib chiqing graf va src = 0



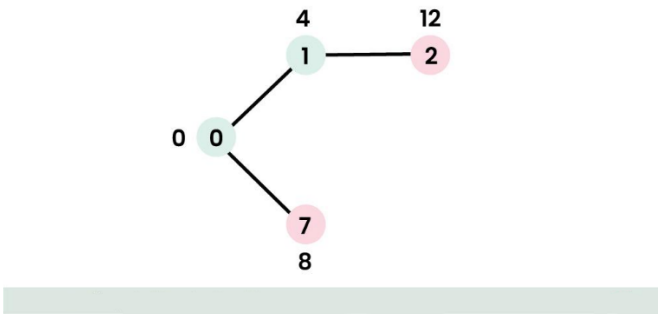
Dijkstra's Algorithm

1-qadam: To‘siq majmuasi dastlab bo‘sh va masofalar qator {0, inf, inf, inf, inf, inf, inf, inf} uchlari tayinlangan cheksiz bildiradi. Endi minimal masofa qiymatiga ega tugunni tanlang. 0 yoki 1 va 7 ning qo‘shni uchlari. 1 va 7 masofa qiymati 4 va 8 sifatida yangilanadi.

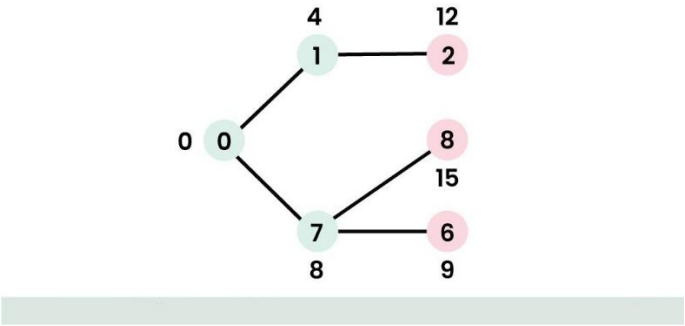
Quyidagi subgrafda tepaliklar va ularning masofa qiymatlari ko‘rsatilgan, faqat cheklangan masofa qiymatlari bo‘lgan tepaliklar ko‘rsatilgan. SPT-ga kiritilgan tepaliklar yashil rangda ko‘rsatilgan.



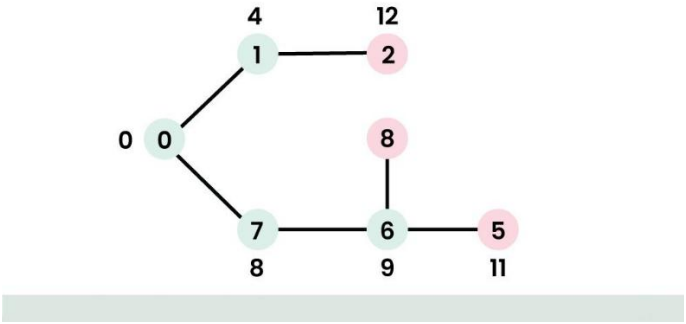
2-qadam: Minimal masofa qiymatiga ega bo‘lgan va SPOT-ga kiritilmagan (SPT to‘plamida emas) tugunni tanlang. Vertex 1 olinadi va SPT to‘plamiga qo‘shiladi. Shunday qilib, sptSet endi {0, 1} ga aylanadi. 1 ning qo‘shni tepaliklarining masofa qiymatlarini yangilanadi. Tugun 2 ning masofa qiymati 12 ga aylanadi.



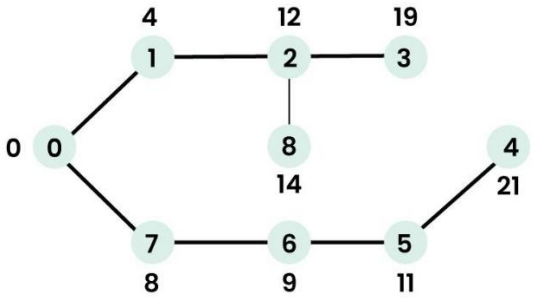
3-qadam: Minimal masofa qiymatiga ega bo‘lgan va SPOT-ga kiritilmagan (SPT to‘plamida emas) tugunni tanlang. Tugun 7 olinadi. Shunday qilib, spt Set endi {0, 1, 7} ga aylanadi. Qo‘shni tepaliklarning masofa qiymatlarini yangilang 7. Tugun 6 va 8 masofa qiymati cheklangan bo‘ladi (mos ravishda 15 va 9).



4-qadam: Minimal masofa qiymatiga ega bo‘lgan va SPOT-ga kiritilmagan (SPT to‘plamida emas) tugunni tanlang. Tugun 6 olinadi. Shunday qilib, spt Set endi bo‘ladi {0, 1, 7, 6}. 6 ning qo‘shni tepaliklarining masofa qiymatlarini yangilang. Tugun 5 va 8 masofa qiymati yangilanadi.



SPT to‘plami berilgan grafikning barcha uchlarini o‘z ichiga olguncha yuqoridagi amallarni takrorlaymiz. Nihoyat, biz quyidagi eng qisqa yo‘l daraxtini (SPT) olamiz.




```
// Dijkstra shortest path algoritmining
//C++ dasturlas tilida implementatsiyasi
#include <iostream>
using namespace std;
#include <limits.h>

// Graftagi tugunlar soni
#define V 9
// Tugunlarni minimal bilan topish uchun yordamchi funktsiya
// masofa qiymati, hali kiritilmagan tugunlar to‘plamidan
// eng qisqa yo‘l daraxti
int minDistance(int dist[], bool sptSet[])
{

   // minimum qiymatini joriy yetish
   int min = INT_MAX, min_index;

   for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
             min = dist[v], min_index = v;

   return min_index;
}

// Qurilgan masofani chop etish uchun yordamchi funktsiya
// massiv
```

```
void printSolution(int dist[])
{
   cout << "Tugun \t manba orasidagi masofa" << endl;
   for (int i = 0; i < V; i++)
        cout << i << " \t\t\t\t" << dist[i] << endl;
}
// Dijkstra-ning yagona manbasini amalga oshiradigan funktsiya
//  yordamida ko‘rsatilgan grafik uchun eng qisqa yo‘l
algoritmi
// qo‘shnillik matritsasi
void dijkstra(int graph[V][V], int src)
{
   int dist[V];
   // Natija eng qisqa masofani ozida saqlaydi
   // src da i gacha masofa

   bool sptSet[V];
   for (int i = 0; i < V; i++)
        dist[i] = INT_MAX, sptSet[i] = false;

   // Manba tugunning o‘zidan o‘zigacha masofasi har doim 0 ga
teng
   dist[src] = 0;

   // Barcha tugunlar orasidagi eng qisqa masofa
   for (int count = 0; count < V - 1; count++) {
        // Berilgan setdan tugunlar orasidagi eng qisqa masofa
        int u = minDistance(dist, sptSet);

        sptSet[u] = true;

        // Update dist value of the adjacent vertices of the
        // picked vertex.
        for (int v = 0; v < V; v++)

             if (!sptSet[v] && graph[u][v]
                  && dist[u] != INT_MAX
                  && dist[u] + graph[u][v] < dist[v])
                  dist[v] = dist[u] + graph[u][v];
   }
```

```
    // qisqa masofani chiqarish
    printSolution(dist);
}

// Bosh funktsiya
int main()
{
    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
                        { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
                        { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
                        { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
                        { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
                        { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
                        { 0, 0, 0, 0, 0, 2, 0, 1, 6 },
                        { 8, 11, 0, 0, 0, 0, 1, 0, 7 },
                        { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    // Funktsiyani chaqirish
    dijkstra(graph, 0);

    return 0;
}
Dastur natijasi:
```

```
Tugun    manba orasidagi masofa
0               0
1               4
2               12
3               19
4               21
5               11
6               9
7               8
8               14
```

Vaqt murakkabligi: O (V^2)

Xotira murakkabligi: O (V)

Eslatmalar:

- Kod eng qisqa masofani hisoblab chiqadi, lekin yo‘l ma’lumotlarini hisoblamaydi. Ajdotlar massivini yarating, masofa yangilanganda

ajdotlar massivini yangilang va manbadan turli tugunlarga eng qisqa yo‘lni ko‘rsatish uchun foydalaning.

- Amalga oshirishning vaqt murakkabligi O(V2). Agar kirish grafigi qo‘shni ro‘yxat yordamida taqdim etilsa, uni ikkilik uyum yordamida O(E * log V) ga kamaytirish mumkin. Qo‘shimcha ma’lumot olish uchun bog’langan ro‘yxati vakillik uchun Dijkstra ning algoritmi qarang.
- Dijkstra algoritmi salbiy vazn davrlari bo‘lgan grafikalar uchun ishlamaydi.

Dijkstra algoritmining qo‘llanilishi:

- Google xaritalar manba va manzil orasidagi eng qisqa masofani ko‘rsatish uchun Dijkstra algoritmidan foydalanadi.
- Kompyuter tarmog’ida, Dijkstra algoritmi kabi turli xil marshrutlash protokollari uchun asos yaratadi OSPF (birinchi navbatda eng qisqa yo‘lni oching) va IS-IS (oraliq tizimdan oraliq tizimga).
- Transport va transportni boshqarish tizimlari transport oqimini optimallashtirish, tirbandlikni minimallashtirish va transport vositalari uchun eng samarali yo‘nalishlarni rejalashtirish uchun Dijkstra algoritmidan foydalanadi.
- Aviakompaniyalar yoqilg’i sarfini minimallashtiradigan, sayohat vaqtini qisqartiradigan parvoz yo‘llarini rejalashtirish uchun Dijkstra algoritmidan foydalanadilar.
- Dijkstra algoritmi integral mikrosxemalar va juda keng ko‘lamli integratsiya (VLSI) chiplarida ulanishlarni yo‘naltirish uchun elektron dizaynni avtomatlashtirishda qo‘llaniladi.

Mavzu yuzasidan savollar

1. Graf muammolarini hal qilish uchun DFS yoki BFS dan qachon foydalanish kerak

2. DFS algoritmi nima
3. BFS algoritmi nima
4. BFS,DFS va dijkstra algoritmining qo‘llanilishi.
5. BFS va DFS algoritmlarining ishlashtagi asosiy farqlari.