Najkraći putevi iz zadatog čvora

Ulaz

- Aciklički usmereni težinski graf $G = (V, E)$
- čvor v

Izlaz

Najkraći putevi iz v do svih ostalih čvorova.

Rešenje

Induktivno, posmatramo čvor z koji je poslednji u top. sort.

- do svih ostalih znamo da dođemo.

Proverimo čvorove koji su imaju direktnu vezu sa z:

```
z.cena = min { w.cena + cenaGrane(w, z) | (w, z) ∈ E }
```

Algoritam

```
for (w : cvorovi) {
    w.cena = ∞;
}

Inicijalizujemo `ulazniStepen` za svaki čvor (DFS).

Napravimo listu svih čvorova ulaznog stepena 0.

v.cena = 0; // Cena puta od v do v je nula

while (lista nije prazna) {
    w = uzmi prvi čvor liste;

    for (sve grane (w, z)) {
        z.cena = min(z.cena, w.cena + cenaGrane(w, z))

        z.ulazniStepen = z.ulazniStepen - 1;

        if (z.ulazniStepen == 0) {
            stavi z u listu;
        }
    }
}
```

Složenost

Ista kao kod topološkog sortiranja.

Najkraći putevi iz zadatog čvora

Ulaz

- Težinski graf G = (V, E)
- čvor v

Izlaz

Najkraći putevi iz v do svih ostalih čvorova.

Primena

Često (u primeni teorije grafova) je potrebno odrediti najkraći put između dva čvora u težinskom grafu. Najčešće primene su u određivanju najkraće rute između dva grada (gde se može tražiti i put sa najmanjom potrošnjom goriva ili slično). Isto tako postoje i primene gde se traži najduži put između dva čvora – tipično kod određivanja kritičnog puta u mrežnom planiranju. Jedna od znacajnijih primena je Open Shortest Path First protokol pri IP rutiranju.

Don't compete with me: firstly, I have more experience, and secondly, I have chosen the weapons.
Edsger Dijkstra (1930-2002)

Rešenje

Tražimo najbliži čvor čvoru v, neka je to x.

Biramo sledeći najbliži čvor čvoru v i biramo

```
min { cenaGrane(v, z), cenaGrane(v, x) + cenaGrane(x, z) }
```

Indukciju radimo po broju čvorova kojima je dužina već izračunata. Neka je V_k skup k najbližih čvorova. Najbliži čvor w van V_k će biti čvor koji ima najmanju dužinu:

```
min { u.cena + cenaGrane(u, w) | u ∈ Vk }
```

Algoritam Dijkstra- 1959

```
for (w : cvorovi) {
    w.oznaka = false;
}

v.cena = 0;

while (postoji neoznačen čvor) {
    w = neoznačen čvor najmanje vrednosti w.cena;
    w.oznaka = true;

    for (sve grane(w, z), z nije označen) {
        z.cena = min (z.cena, w.cena + cenaGrane(w, z));
    }
}
```

Složenost

•često traženje minimuma, konstantno vreme ako koristimo hip
•ažuriranje vrednosti u hipu O(log m), m - veličina hipa (brisanje elemenata |V| + popravke |E|)
•ukupno: O(log|V|(|V| + |E|))

Napomena

U svakom koraku mogu se popraviti samo putevi do onih `čvorova u kojima nismo bili, te je dovoljno popravke` ponavljati `|V|-1` puta.

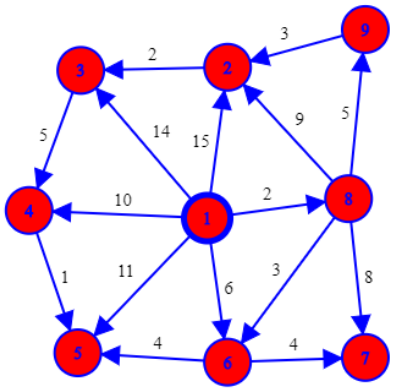
Kompleksnost Dijkstrinog algoritama može se jednostavno odrediti. Spoljna while petlja je reda O(n), jer se svaki čvor tačno jednom uzima za pivota.
Za svaki pivot čvor gledamo susedne čvorove i određujemo nove udaljenosti (kroz celu glavnu petlju radimo sve skupa m puta), tako da je kompleksnost sada O(n+m). Najveći posao je u određivanju sledećeg pivot čvora i kompleksnost zavisi od strukture podataka koju koristimo za polje udaljenosti.

Ako je polje jednostavna linearna struktura, onda je traženje pivot čvora kompleksnosti O(n) i ukupna kompleksnost algoritma je $O(n^2)$. Ako je polje složenije strukture, (npr. red sa prioritetima, Fibonacci heap, gde je kompleksnost traženja pivot čvora O(ln(n)), konačna kompleksnost algoritma je $O((m+n)\cdot ln(n))$.

Ako želimo kompleksnost izraziti zavisno samo od broja čvorova, moramo uzeti najteži slučaj, a to je potpuni graf. Tada, kompleksnost algoritma Dijkstra iznosi $O(n^2)$

Zadaci – (rađeno na Diskretnim strukturama???)

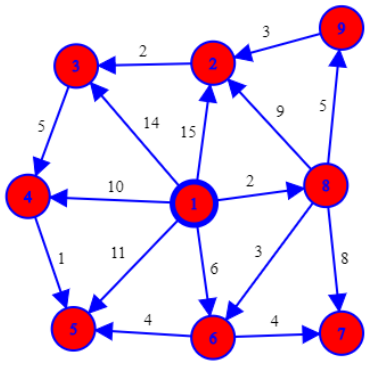
Zadatak 1
Dat je usmereni graf *G=(V,E)* sa skupom čvorova *V={1,2,3,4,5,6,7,8,9}* i skupom grana *E* sa težinama kao na slici. Odrediti težine najkraćih puteva od čvora 1 do ostalih čvorova u grafu.



REŠENJE
Koristimo Dijkstrin algoritam za nalaženje težina najkraćih puteva od zadatog čvora 1 do ostalih čvorova u grafu (glava 6.5 u knjizi Algoritmika).
Prikaz postupka i rešenje je

naveden u tabeli .
Najpre se popuni 1. red tabele tako da se unesu vrednosti težina grana sa slike grafa.
Potom se izabere najmanja težina te vrste. Konkretno to je ovde vrednost 2 čvora 8.
Potom se označi čvor 8 i formira 2. red tabele u kom se menjaju težine onih puteva koje bivaju kraće ako se u njih uključi čvor 8. I tako redom sve dok postoji neoznačen čvor.

označeni	**čvor w (najmanje SP)**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	-	15	14	10	11	6	*	**2**	*
1,8	8	11	14	10	11	**5**	10	**2**	7
1,6,8	6	11	14	10	9	**5**	9	**2**	**7**
1,6,8,9	9	10	14	10	**9**	**5**	9	**2**	**7**
1,5,6,8,9	5	10	14	10	**9**	**5**	**9**	**2**	**7**
1,,5,6,7,8,9	7	**10**	14	10	**9**	**5**	**9**	**2**	**7**
1,2,5,6,7,8,9	2	**10**	12	**10**	**9**	**5**	**9**	**2**	**7**
1,2,4,5,6,7,8,9	4	**10**	**12**	**10**	**9**	**5**	**9**	**2**	**7**
1,2,3,4,5,6,7,8,9	3	**10**	12	**10**	**9**	**5**	**9**	**2**	**7**



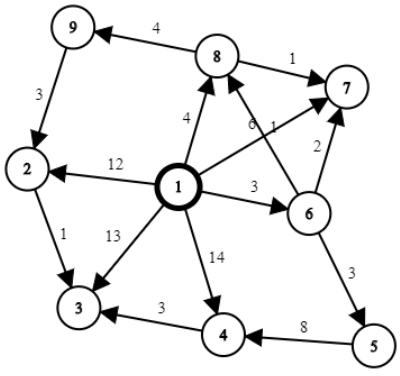
Konstruisimo Dijkstra algoritam (cija je slozenost: O((V+E)log V)) koji ce nam ispisati udaljenost cvora 1 od cvora 7 (u nasem programu to ce biti udaljenost cvora 0 od cvora 6)

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <queue>
#define NMAX 100001
#define INF 1000000000
using namespace std;
int n;
struct CvorGrafa
{
   int rastojanje;
   vector<int> sused;
   vector<int> tezina;
};
CvorGrafa graf[NMAX];
bool mark[NMAX];
```

```
struct cvorRed
{    int cvor, rastojanje;
     bool operator <(const cvorRed &a) const
     {    if (rastojanje != a.rastojanje) return (rastojanje > a.rastojanje);
          return (cvor > a.cvor);
      }
};


inline void Dijkstra(int CvorPocetak)
{
   priority_queue<cvorRed> red;
   cvorRed P;
   for (int i=0;i<n;i++)
   {
      if (i == CvorPocetak)
      {
         graf[i].rastojanje = 0;
         P.cvor = i;
         P.rastojanje = 0;
         red.push(P);
      }
      else graf[i].rastojanje = INF;
   }


   while (!red.empty())
   {
      cvorRed tekuci = red.top();
      red.pop();
      int tekCvor = tekuci.cvor;
      int tekRastojanje = tekuci.rastojanje;
      for (int i=0;i<graf[tekCvor].sused.size();i++)
      {
         if (!mark[graf[tekCvor].sused[i]])
         {
            int nextNode = graf[tekCvor].sused[i];
            if (tekRastojanje + graf[tekCvor].tezina[i] < graf[nextNode].rastojanje)
            {
               graf[nextNode].rastojanje = tekRastojanje + graf[tekCvor].tezina[i];
               P.cvor = nextNode;
               P.rastojanje = graf[nextNode].rastojanje;
               red.push(P);
            }
         }
      }
      mark[tekCvor] = true;
   }
}
```

```
int main()
{
   n = 9;


   graf[0].sused.push_back(1);    graf[0].tezina.push_back(15);
   graf[0].sused.push_back(2);    graf[0].tezina.push_back(14);
   graf[0].sused.push_back(3);    graf[0].tezina.push_back(10);
   graf[0].sused.push_back(4);    graf[0].tezina.push_back(5);
   graf[0].sused.push_back(5);    graf[0].tezina.push_back(6);
   graf[0].sused.push_back(7);    graf[0].tezina.push_back(2);


   graf[1].sused.push_back(2);    graf[1].tezina.push_back(2);


   graf[2].sused.push_back(3);    graf[2].tezina.push_back(5);


   graf[3].sused.push_back(4);    graf[3].tezina.push_back(1);


   graf[5].sused.push_back(4);    graf[5].tezina.push_back(4);
   graf[5].sused.push_back(6);    graf[5].tezina.push_back(4);


   graf[7].sused.push_back(5);    graf[7].tezina.push_back(3);
   graf[7].sused.push_back(6);    graf[7].tezina.push_back(8);
   graf[7].sused.push_back(8);    graf[7].tezina.push_back(5);


   graf[8].sused.push_back(1);    graf[8].tezina.push_back(3);


   Dijkstra(0);
   printf("%d\n",graf[6].rastojanje);
   return 0;
}
```

Zadatak 2. Dat je usmereni graf *G=(V,E)* sa skupom čvorova *V={1,2,3,4,5,6,7,8,9}* i skupom grana *E* sa težinama kao na slici.



Odrediti težine najkraćih puteva od čvora 1 do ostalih čvorova u grafu.

k	Skup *S*		niz *d*								niz *p*						
			2	3	4	5	6	7	8	9	2	3	4	5	6	7	8
1		-	12	13	14	¥	***3***	6	4	¥	1	1	1	0	1	1	1
2	6	6	12	13	14	6	***3***	5	***4***	¥	1	1	1	6	1	6	1
3	6, 8	8	12	13	14	6	***3***	***5***	***4***	8	1	1	1	6	1	6	1
4	6, 7, 8	7	12	13	14	***6***	***3***	***5***	***4***	8	1	1	1	6	1	6	1
5	5, 6, 7, 8	5	12	13	14	***6***	***3***	***5***	***4***	***8***	1	1	1	6	1	6	1
6	5, 6, 7, 8, 9	9	***11***	13	14	***6***	***3***	***5***	***4***	***8***	9	1	1	6	1	6	1
7	2, 5, 6, 7, 8, 9	2	***11***	***12***	14	***6***	***3***	***5***	***4***	***8***	9	2	1	6	1	6	1
8	2, 3, 5, 6, 7, 8, 9	3	***11***	***12***	***14***	***6***	***3***	***5***	***4***	***8***	9	2	1	6	1	6	1
9	2, 3, 4, 5, 6, 7, 8, 9	4	***11***	***12***	***14***	***6***	***3***	***5***	***4***	***8***	9	2	1	6	1	6	1

Zadatak 3

Na zabavi se nalazi n ljudi koji su numerisani brojevima od 0 do *n*-1. Matricom a(nxn) dato je ko se s kim poznaje, tako je *a*[*i,j*]*=1* ako osoba *i* poznaje osobu *j*, inače je *a*[*i,j*]*=0*.Osoba s zna vest koju treba da saopšti ostalima. Osoba *i* može prepričati vest samo osobama koje poznaje. Svako prepričavanje traje 1 minut. Prepričavanja se mogu odvijati paralelno i svaka osoba može okupiti svoje poznanike kojima će saopštiti vest. Svaka osoba čim čuje vest odmah je prepričava, bez gubitka vremena. Napisati metod kojim se određuje minimum vremena u minutima koji je potreban da svi koji mogu čuju vest.

> Problem možemo rešiti korišćenjem prethodno opisanog metoda *Dijkstra* kojim određujemo dužine najkraćih puteva od osobe s koja zna vest do svih ostalih osoba. Minimum vremena u minutima koji je potreban da svi koji mogu čuju vest je maksimalni član niza d.

Zadatak 4

Država alhemičara ima N naseljenih punktova, numerisanih od 1 do N, i M puteva. Naseljeni punktovi su dva tipa: sela i gradovi. Osim toga u državi je jedna prestonica (ona može biti u selu ili gradu). Putovanje između dva naseljena punkta (ako postoji put) traje T_i minuta. Nakon odluke da se u prestonici organizuje olimpijada alhemičara u svaki grad su upućeni kuriri sa informacijom o olimpijadi.

Napišite program koji određuje u kom poretku i za koje vreme svaki od kurira stiže do svog grada. Pretpostavlja se da se kuriri u toku puta nigde ne zadržavaju.

U prvoj liniji ulaznog fajla su zapisana 3 broja: N – broj naseljenih punktova ($2 \le N \le 1000$), M – broj puteva ($1 \le M \le 10000$) i K – broj gradova ($1 \le K \le N$). Dalje je zapisan broj prestonice C ($1 \le C \le N$). U sledećem redu su K brojeva gradova. Dalje sledi M trojki brojeva S_i, E_i, T_i, gde su S_i, E_i brojevi naseljenih punktova koje povezuje put, a T_i – trajanje putovanja tim putem ($1 \le T_i \le 100$). Garantuje se da se iz prestonice može stići do svakog grada.

test.txt	izlaz.txt
5 4 5 1 1 2 3 4 5 1 2 1 2 3 10 3 4 100 4 5 100	1 0 2 1 3 11 4 111 5 211
5 5 3 1	5 1

2 4 5 2 1 1 2 3 10 3 4 100 4 5 100 1 5 1	2 1 4 101

Napomena. Alhemija je disciplina koja kombinuje elemente mnogih nauka i filozofskih disciplina, poput hemije, metalurgije, fizike, medicine, astrologije, misticizma i umetnosti.

Rešenje. Za svako naselje korišćenjem Dijkstrinog algoritma nađemo najmanje vreme, za koje kurir do njega stiže. Nakon toga ispisati naselja koja su gradovi sortirane po vremenu u neopadajućem poretku.

Zadatak 5

Pokazati da je izlaz Dijkstrinog algoritma korensko stablo.

Zadatak 6

Neka je G neusmereni težinski graf i neka je T njegovo stablo najkraćih puteva od v. Da li će T ostati stablo najkraćih puteva ako se težine uvećaju za c?

Zadatak 7

Da li algoritam Dijkstra radi nad grafovima koji imaju negativne težine?

Zadatak 8 – MATF 2016

Временско ограничење 1 секунда

Меморијско ограничење 64 MB

Авио компанија обавља летове међу n градова означених редом 0, 1, ..., n-1. За сваки лет познати су полазни и долазни град, као и број путника, које авион може да превезе на том лету. Летови су двосмерни. Седиште авио компаније је у граду 0. За сваки град i треба израчунати макисмални број путника на линији која директним летом или са неколико преседања превози путнике из града 0 у град i (i=1,2,..., n-1).

Напишите програм **AVIONI,** који обавља неопходна израчунавања и налази максимални број путника, који може стићи у сваки град почевши од града 0.

Улаз

У првом реду стандардног улаза дата су два цела броја n и m – број градова и број летова. Сваки од наредних m редова садржи три цела броја – редне број градова између којих се обавља тај лет и број путника, који може да се превезе на том лету.

Излаз

У једином реду стандардног излаза програма треба да испише један цео број за сваки град, осим града 0 – максималан број путника, који могу стићи до тог града. У испису, бројеве раздвојити размаком.

Ограничења

$1 \leq n \leq 20000$

$1 \leq m \leq 200000$

$1 \leq$ број путника по једном лету ≤ 10000

Пример

Улаз

```
5 6
0 1 60
4 0 70
1 2 60
3 1 10
3 2 40
3 4 30
```

Излаз

```
60 60 40 70
```

Решење:

Задатак се решава у форми прилагођавања Дајкстриног алгоритма за тражење најкраћих путева од једног чвора до свих осталих.

Идеја: Означимо са

U – скуп обрађених (маркираних) чворова;

V – скуп свих чворова;

N – број чворова;

Решење задатка за сваки град се чува у низу dist.

На почетку: U = {0}; dist[0]=∝;

Остали елементи низа dist се поставе на 0, док елементи, који су суседни граду 0, добијају вредности једнаке тежинама грана које су инцидентне с градом 0.

Док (V≠U) поновити:

У скупу V\U наћи чвор с највећом вредности - dist[w]

Сместити га у скуп U

За свако чвор v који је сусседан чвору w:

Ако (v ∉ U) dist[v] = max (d[v], min (d[w], l(w, v)))

Да би се задовољило временско ограничење и 100% тест примера неопходно је да се реализује Дајкстрин алгоритам с хипом и граф да се представи као листа повезаности. Тада је сложност решења $\Theta(n*\log_2 n)$.

```
#include <stdio.h>
#include <list>
#define MAX_VALUE 1000000
#define MAX 20000

using namespace std;

struct Edge {
   int x;
   int d;
};

list<Edge> B[MAX];

int dist[MAX];
int visit[MAX];

int posHeap[MAX];
int heap[MAX];
```

```
int hsize;

int visitCount;
int n,m,k;
int minP;
int maxH,minH;

void read() {
   int i, x,y,d;
   scanf("%d %d", &n, &m);
   for(i=0; i<m; i++) {
      scanf("%d %d %d", &x, &y, &d);
      Edge Edge1, Edge2;
      Edge1.x = y; Edge1.d = d;
      Edge2.x = x; Edge2.d = d;
      B[x].push_back(Edge1);
      B[y].push_back(Edge2);
   }
}

void hswap(int i, int j) {
   posHeap[heap[i]] = j;
   posHeap[heap[j]] = i;
   int k = heap[i];
   heap[i] = heap[j];
   heap[j] = k;
}

void moveDown(int x) {
   while (x< (hsize>>1)) {
      int child = 2*x +1;
      if (child + 1 < hsize && dist[heap[child + 1]] > dist[heap[child]]) {
         child++;
      }
      if (dist[heap[x]] > dist[heap[child]]) {
         break;
      }
      hswap(x, child);
      x = child;
   }
}

void moveUp(int x) {
   while (x>0) {
      int parent = (x-1)>>1;
      if (dist[heap[x]] < dist[heap[parent]]) break;
      hswap(x, parent);
      x = parent;
   }
}

int removeMin() {
```

```
    int res = heap[0];
    if (hsize <= 0) return -1;
    else {
        int last = heap[--hsize];
        if (hsize<minH) minH = hsize;

        if (hsize > 0) {
            heap[0] = last;
            moveDown(0);
        }
        return res;
    }
}

void add(int x) {
    posHeap[x] = hsize;
    heap[hsize++] = x;
    moveUp(hsize-1);
}

int mini(int x, int y) {
    return (x<y)? x : y;
}

void dijkstra() {

    dist[0] = MAX_VALUE;
    hsize = 0;
    add(0);

    int k;
    bool flag = true;
    while (flag) {
        k = removeMin();
        if (k == -1) flag = false;
        else {
            visit[k] = 0;
            visitCount++;
            for(list<Edge>::iterator it=B[k].begin(); it !=B[k].end(); it++) {
                if(visit[it->x] && dist[it->x]<mini(dist[k],it->d)) {
                    dist[it->x] = mini(dist[k], it->d);
                    if (posHeap[it->x]>=0) moveUp(posHeap[it->x]);
                    else add(it->x);
                }
            }
        }
    }
}


int main() {
    read();
```

```
    int i;
    for(i=0; i<n; i++) {
       visit[i] = 1;
       dist[i] = 0;
       posHeap[i] = -1;
    }
    visitCount = 0;
    dist[0] = MAX;
    dijkstra();

    for(int i=1; i<n; i++)
       printf("%d ", dist[i]);
    printf("\n");

        return 0;
}
```