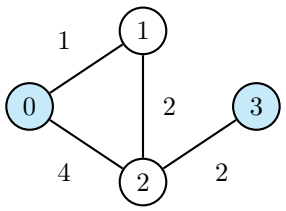
Recimo da nam je poznata mapa gradova i puteva jedne države. Za svaki grad znamo sa kojim gradovima je on povezan i koje su dužine putevi koji ih povezuju. Ovo naravno možemo prikazati grafom.



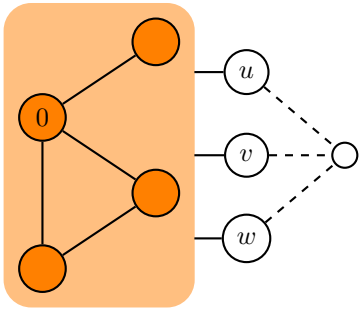
Najkraći put od čvora 0 do čvora 3 je $0 - 1 - 2 - 3$ i njegova dužina je 5. Primetimo da to nije put sa najmanjim brojem čvorova, ali jeste put sa najmanjim zbirom težina grana. Put $0 - 2 - 3$ ima manji broj čvorova, ali je njegova dužina jednaka 6.

U zavisnosti od strukture grafa (da li je usmeren, da li je acikličan, da li postoje grane negativne težine) i najkraćih puteva koje želimo da izračunamo (od jednog čvora ka svim ostalim ili između svaka dva čvora) postoji više različitih algoritama koje možemo primeniti.

Dajkstrin algoritam

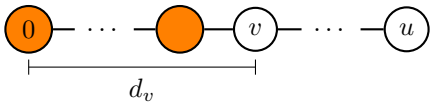
Dajkstrin algoritam koristimo za određivanje najkraćih puteva od jednog čvora u do svih ostalih u grafu bez grana negativnih težina. Algoritam radi isto bez obzira na to da li je graf usmeren ili neusmeren.

Dajkstrin algoritam se zasniva na svojstvu koje ćemo prikazati u narednom primeru.



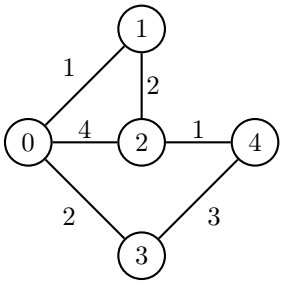
Obrađeni (narandžasti) čvorovi predstavljaju one za koje već imamo izračunat najkraći put do čvora 0. Za neobrađene (bele) čvorove su nam poznati njihovi najkraći putevi do 0 **koji koriste isključivo narandžaste čvorove** i njihove dužine su d_u, d_v i d_w.

Recimo da je d_u najmanja od te tri dužine. Tvrdimo da je taj put p od 0 do u zaista i najkraći (da ne postoji put manje dužine). Pretpostavimo suprotno. To znači da najkraći put s od 0 do u sadrži bar jedan beli čvor.

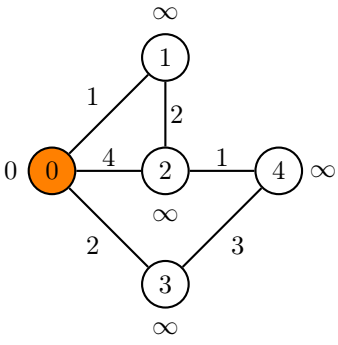


Neka je v prvi beli čvor na putu s. To znači da je dužina puta s barem d_v, ali kako je $d_u \leq d_v$ to je dužina puta p manja ili jednaka dužini puta s. Kontradikcija.

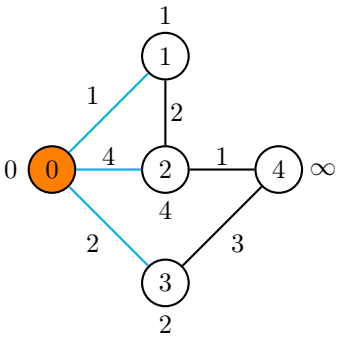
Pogledajmo primer izvršavanja algoritma i kako koristimo dokazano svojstvo.



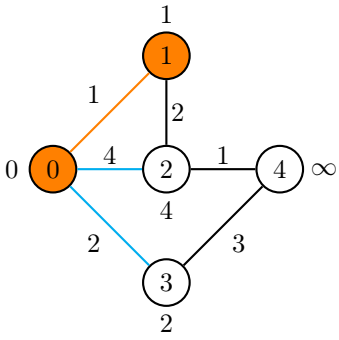
Tražimo dužinu najkraćeg puta od čvora 0 do svih ostalih čvorova.



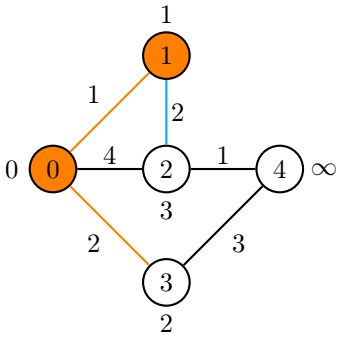
Postavljamo udaljenost čvora 0 od samog sebe na 0 i udaljenost svih ostalih čvorova na ∞. Za čvor 0 znamo da smo odredili njegovu optimalnu udaljenost.



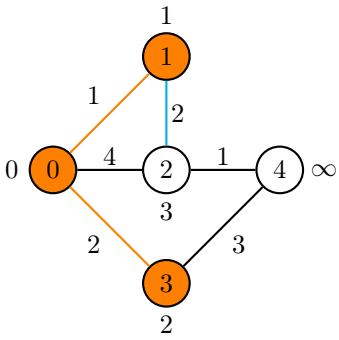
Ažuriramo udaljenosti svih suseda čvora 0.



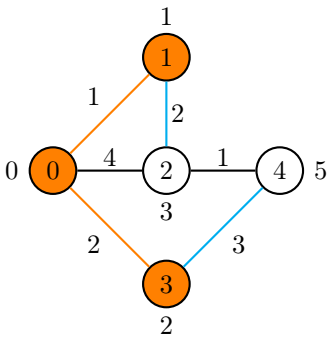
Od neobrađenih čvorova biramo onaj čija je trenutna udaljenost najmanja. U ovom slučaju je to čvor 1. Znamo da je to ujedno i njegova optimalna udaljenost od 0. Pretpostavimo da nije - najkraći put od 0 do 1 bi onda morao da sadrži ili čvor 2 ili čvor 3. Kako je dosadašnja udaljenost i čvora 2 (= 4) i čvora 3 (= 2) veća od dosadašnje udaljenosti čvora 1 (= 1), taj put bi bio duži od ovog koji direktno ide do čvora 1.



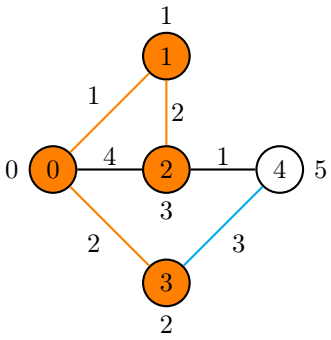
Ažuriramo udaljenosti svih suseda čvora 1. Čvor 2 dobija novu udaljenost jer je dužina puta preko čvora 1 (1 do čvora 1 plus 2 preko grane (1,2)) manja od njegove dosadašnje (4).



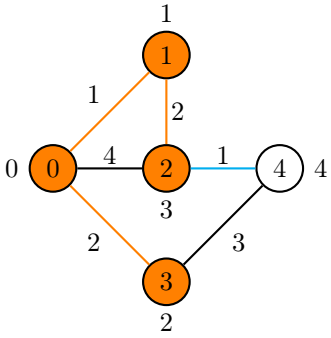
Od neobrađenih čvorova biramo onaj čija je trenutna udaljenost najmanja. U ovom slučaju je to čvor 3. Znamo da je to ujedno i njegova optimalna udaljenost od 0. Pretpostavimo da nije - najkraći put od 0 do 3 bi onda morao da sadrži čvor 2. Kako je dosadašnja udaljenost čvora 2 (= 3) veća od dosadašnje udaljenosti čvora 3 (= 2), taj put bi bio duži od ovog koji direktno ide do čvora 3.



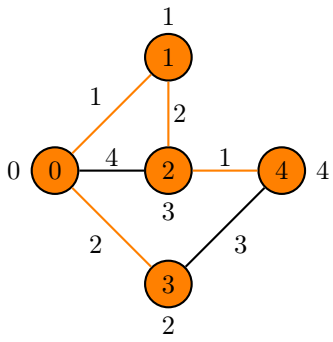
Ažuriramo udaljenosti svih suseda čvora 3. Čvor 4 dobija novu udaljenost jer je dužina puta preko čvora 3 (2 do čvora 3 plus 3 preko grane (3,4)) manja od njegove dosadašnje (∞).



Od neobrađenih čvorova biramo onaj čija je trenutna udaljenost najmanja. U ovom slučaju je to čvor 2. Znamo da je to ujedno i njegova optimalna udaljenost od 0. Pretpostavimo da nije - najkraći put od 0 do 2 bi onda morao da sadrži čvor 4. Kako je dosadašnja udaljenost čvora 4 ($= 5$) veća od dosadašnje udaljenosti čvora 2 ($= 4$), taj put bi bio duži od ovog koji direktno ide do čvora 2.



Ažuriramo udaljenosti svih suseda čvora 2. Čvor 4 dobija novu udaljenost jer je dužina puta preko čvora 2 (3 do čvora 2 plus 1 preko grane (2,4)) manja od njegove dosadašnje (5).



Od neobrađenih čvorova biramo onaj čija je trenutna udaljenost najmanja. U ovom slučaju je to čvor 4. Znamo da je to ujedno i njegova optimalna udaljenost od 0. Kako smo obradili sve čvorove postupak je gotov i za svaki čvor imamo određenu optimalnu udaljenost od čvora 0.

Algoritam možemo implementirati na sledeći način. Za svaki čvor nam je potrebno da čuvamo dva podatka - da li smo pronašli dužinu najkraćeg puta od početnog (tj. da li je čvor obrađen) i dužinu dosada pronađenog puta. U svakom koraku linearnim prolaskom kroz sve čvorove određujemo onaj neobrađeni čvor čija je dužina trenutnog puta najmanja. Ažuriramo udaljenosti svih njegovih neobrađenih suseda i markiramo da je obrađen. Taj postupak ponavljamo dok ne markiramo sve čvorove. Nakon što su svi čvorovi markirani, za svaki od njih nam je poznata i dužina najkraćeg puta od početnog do tog čvora.

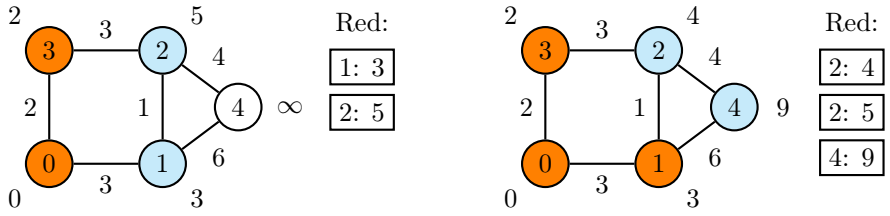
Ukoliko nam je potrebno, za svaki čvor možemo pamtiti i koji čvor mu je poslednji ažurirao udaljenost (prikazano obojenim granama na gornjem primeru). To nam omogućava da rekonstruišemo najkraći put do bilo kog čvora.

```
vector<int> dijkstra(int v,
                     vector< vector< pair<int, int> > >& g) {
  int n = g.size();
  // Za svaki čvor pamtimo
  vector<bool> mark(n, false); // Da li je obrađen
  vector<int> prev(n, -1); // Sa kog smo čvora došli
  vector<int> dist(n, numeric_limits<int>::max()); // Udaljenost
  dist[v] = 0;

  for(int i = 0; i < n; i++) {
    int mind = numeric_limits<int>::max(), minv = -1;
    for(int j = 0; j < n; j++)
      if(!mark[j] && dist[j] < mind) {
        mind = dist[j];
        minv = j;
      }

    mark[minv] = true;
    for(auto& e : g[minv]) {
      int u = e.first, w = e.second;
      if(!mark[u] && dist[minv] + w < dist[u]) {
        dist[u] = dist[minv] + w;
        prev[u] = minv;
```

```
      }
    }
  }
  return dist;
}
```

Složenost ovako implementiranog algoritma je $O(|V|^2)$. U svakom koraku obrađujemo po jedan novi čvor, a pronalaženje čvora koji ćemo obraditi (onog sa najmanjom trenutnom udaljenošću) je linearan prolaz kroz sve neobrađene. Obrađivanje grana je ovde ukupne složenosti $O(|E|)$ jer svaku granu koristimo tačno jednom, pa to ne utiče na složenost.

Ovde je moguće napraviti optimizaciju, konkretno pri određivanju čvora sa najmanjom trenutnom udaljenošću. Ukoliko sve čvorove (čija trenutna udaljenost nije ∞) čuvamo u redu sa prioritetom, u složenosti $O(1)$ možemo odrediti naredni čvor koji obrađujemo. Onda prilikom ažuriranja udaljenosti njegovih suseda u red dodajemo suseda zajedno sa njegovom novom udaljenošću.



Nakon obrade čvora 1 (jer se on nalazi na vrhu reda sa prioritetom) i dodavanja čvorova 2 i 4 u red imamo dve kopije čvora 2 u redu. Ovo ne predstavlja problem zato što će iz reda prvo izaći ona kopija sa najmanjom udaljenošću, pa sve naredne možemo ignorisati. Time postižemo efekat brisanja te druge kopije, samo što to brisanje odlažemo. Primetimo da nismo mogli da obrišemo tu kopiju odmah kada smo dodavali novu, ažurniju, jer bi to zahtevalo pretragu prioritetnog reda.

```
vector<int> dijkstra(int v,
                     vector< vector< pair<int, int> > >& g) {
  int n = g.size();
  vector<int> dist(n, numeric_limits<int>::max());
  dist[v] = 0;

  priority_queue< pair<int, int>,
                  vector< pair<int, int> >,
                  greater< pair<int, int> > > q;
  q.push({0, v});

  while(!q.empty()) {
    int mind = q.top().first, minv = q.top().second;
```

```
    q.pop();
    if(dist[minv] != mind)
      continue;

    for(auto& e : g[minv]) {
      int u = e.first, w = e.second;
      if(dist[minv] + w < dist[u]) {
        dist[u] = dist[minv] + w;
        q.push({dist[u], u});
      }
    }
  }

  return dist;
}
```

Složenost ove implementacije je $O((|V| + |E|) \log |V|)$. Pri obradi svakog čvora ažuriramo sve njegove susede i dodajemo ih u red sa prioritetom. Ukupan broj ažuriranja suseda jednak je broju grana tj. $|E|$. Operacija dodavanja je $O(\log |E|) = O(\log |V|)$ jer se u jednom trenutku može naći $O(|E|)$ čvorova u redu (zbog mogućnosti prisustva više kopija). U $O(1)$ određujemo čvor koji sledeći obrađujemo, a uklanjamo ga sa vrha reda u $O(\log |E|)$.

Složenost dodatnog prostora je ovde $O(|V| + |E|)$. Za svaki od čvorova čuvamo njegovu udaljenost, a pritom u redu sa prioritetom čuvamo $O(|E|)$ čvorova. Ovo možemo poboljšati korišćenjem skupa umesto reda sa prioritetom, jer tada osim brzog pristupa najmanjem elementu i dodavanja imamo mogućnost brisanja proizvoljnog elementa. Svaki put kada ažuriramo udaljenost nekog čvora, pre dodavanja u skup možemo obrisati onu kopiju tog čvora koja se već nalazi u skupu. Time smo prostornu složenost smanjili na $O(|V|)$.

```
vector<int> dijkstra(int v,
                     vector< vector< pair<int, int> > >& g) {
  int n = g.size();
  vector<int> dist(n, numeric_limits<int>::max());
  dist[v] = 0;

  set< pair<int, int> > q;
  q.insert({0, v});

  while(!q.empty()) {
    int minv = q.begin()->second;
    q.erase(q.begin());

    for(auto& e : g[minv]) {
      int u = e.first, w = e.second;
      if(dist[minv] + w < dist[u]) {
```

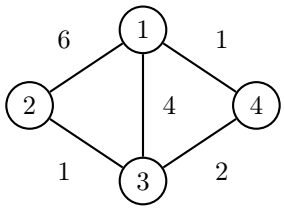
```
        q.erase({dist[u], u});
        dist[u] = dist[minv] + w;
        q.insert({dist[u], u});
      }
    }
  }

  return dist;
}
```

Naglasimo da je ovakva implementacija u praksi za nijansu sporija od implementacije pomoću reda sa prioritetom.

Flojd-Varšalov algoritam

Flojd-Varšalovim algoritmom možemo odrediti dužine najkraćih puteva između svih parova čvorova datog grafa. Graf može sadržati i grane negativnih težina. Kao što je slučaj sa Dajkstrinim algoritmom, Flojd-Varšalov radi i za usmerene i za neusmerene grafove.

Algoritam ćemo konstruisati određivanjem rekurzivne veze za izračunavanje najkraćeg puta. Označimo sa $d(i, j, K)$ dužinu najkraćeg puta od čvora i do čvora j **takvog da su svi unutrašnji čvorovi puta iz skupa** K.



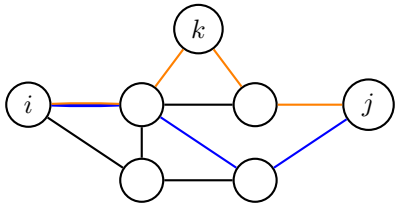
Neka je $K = \{1, 2\}$. Najkraći put od čvora 2 do čvora 4 za taj skup K je put $2 - 1 - 4$ i njegova dužina je 7, pa je $d(2, 4, K) = 7$. Vidimo da postoji i kraći put $2 - 3 - 4$, ali on koristi čvor $3 \notin K$ kao unutrašnji čvor puta.

Na osnovu ovako definisane funkcije d umemo da izračunamo dužinu najkraćeg puta za par čvorova i i j - to je $d(i, j, V)$, odnosno najkraći put od čvora i do čvora j pri čemu dozvoljavamo da se kao unutrašnji čvor puta pojavi bilo koji čvor.

Izvedimo sada rekurzivnu vezu za izračunavanje funkcije d. Prilikom određivanja vrednosti $d(i, j, K)$ postoje dva slučaja:

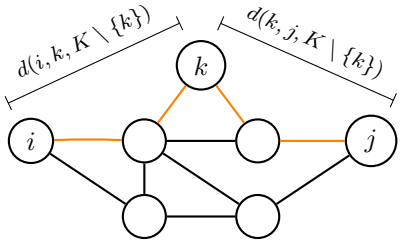
1. Najkraći put od i do j za skup K ne sadrži čvor k
2. Najkraći put od i do j za skup K sadrži čvor k

pri čemu je k neki čvor iz skupa K. Mi naravno ne znamo koji od ova dva uslova je tačan. Na slici su prikazani ovi slučajevi, prvi je označen plavom, drugi narandžastom bojom.



Ukoliko je prvi uslov tačan, važi da je $d(i, j, K) = d(i, j, K \setminus \{k\})$ jer u tom slučaju znamo da k nije unutrašnji čvor najkraćeg puta od i do j.

Ukoliko je drugi uslov tačan, dužinu najkraćeg puta možemo računati kao $d(i, j, K) = d(i, k, K) + d(k, j, K)$ jer znamo da k jeste unutrašnji čvor najkraćeg puta od i do j. Pri tome važi i da je $d(i, k, K) = d(i, k, K \setminus \{k\})$ jer je čvor k krajnji čvor puta od i do k, pa sigurno nije i unutrašnji čvor tog puta. Slično važi i za $d(k, j, K)$. Odatle dobijamo da je $d(i, j, K) = d(i, k, K \setminus \{k\}) + d(k, j, K \setminus \{k\})$.



Kombinovanjem oba slučaja dobijamo rekurzivnu relaciju

$$d(i, j, K) = min\{d(i, j, K \setminus \{k\}),\ d(i, k, K \setminus \{k\}) + d(k, j, K \setminus \{k\})\}$$

pri čemu je bazni slučaj za $K = \emptyset$. U tom slučaju na putu od čvora i do čvora j ne postoje unutrašnji čvorovi, pa važi

$$d(i, j, \emptyset) = \begin{cases} 0 & i = j \\ d(i, j) & (i, j) \in E \\ \infty & (i, j) \notin E \end{cases}$$

gde $d(i, j)$ predstavlja težinu grane (i, j).

Primetimo da možemo odabrati da uvek posmatramo skup oblika $K = \{1, 2, \ldots, k\}$, pa funkciju možemo preformulisati da zavisi od broja k tj.

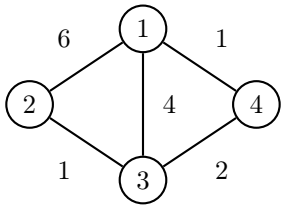
$$d(i, j, k) = min\{d(i, j, k-1),\ d(i, k, k-1) + d(k, j, k-1)\}$$

pri čemu je sada bazni slučaj za $k = 0$.

Naravno, direktno izračunavanje po rekurzivnoj formuli je prevelike složenosti, ali ovo lako možemo popraviti dinamičkim programiranjem. Primetimo da za svaki par čvorova i i j vrednost funkcije $d(i, j, k)$ zavisi od vrednosti funkcije nekih parova za $k-1$. Definišemo matrice $D_k = [d(i, j, k)]$. Matricu D_k možemo

jednostavno izračunati na osnovu matrice D_{k-1} pomoću izvedene rekurzivne formule. Izračunavanjem matrice $D_{|V|}$ dobijamo željene udaljenosti.

Pogledajmo postupak na narednom primeru.



$$D_0 = \begin{bmatrix} 0 & 6 & 4 & 1 \\ 6 & 0 & 1 & \infty \\ 4 & 1 & 0 & 2 \\ 1 & \infty & 2 & 0 \end{bmatrix}$$

Polja na glavnoj dijagonali imaju vrednost 0. Polja koja odgovaraju paru čvorova koji su povezani granom imaju vrednost jednaku težini te grane. Ostala polja imaju vrednost ∞.

$$D_1 = \begin{bmatrix} 0 & 6 & 4 & 1 \\ 6 & 0 & 1 & 7 \\ 4 & 1 & 0 & 2 \\ 1 & 7 & 2 & 0 \end{bmatrix}$$

Računamo $d(i, j, 1)$ za sve parove čvorova. Proveravamo da li je put od i do j koji ide preko čvora 1 bolji od dosadašnjeg najboljeg. To važi samo u slučaju $d(2, 4, 1) = d(2, 1, 0) + d(1, 4, 0) = 7$.

$$D_2 = \begin{bmatrix} 0 & 6 & 4 & 1 \\ 6 & 0 & 1 & 7 \\ 4 & 1 & 0 & 2 \\ 1 & 7 & 2 & 0 \end{bmatrix}$$

Računamo $d(i, j, 2)$ za sve parove čvorova. Proveravamo da li je put od i do j koji ide preko čvora 2 bolji od dosadašnjeg najboljeg. To ne važi ni za jedan par čvorova.

$$D_3 = \begin{bmatrix} 0 & 5 & 4 & 1 \\ 5 & 0 & 1 & 3 \\ 4 & 1 & 0 & 2 \\ 1 & 3 & 2 & 0 \end{bmatrix}$$

Računamo $d(i, j, 3)$ za sve parove čvorova. Proveravamo da li je put od i do j koji ide preko čvora 3 bolji od dosadašnjeg najboljeg. To važi za $d(1, 2, 3) = d(1, 3, 2) + d(3, 2, 2) = 5$ i $d(2, 4, 3) = d(2, 3, 2) + d(3, 4, 2) = 3$.

$$D_4 = \begin{bmatrix} 0 & 4 & 3 & 1 \\ 4 & 0 & 1 & 3 \\ 3 & 1 & 0 & 2 \\ 1 & 3 & 2 & 0 \end{bmatrix}$$

Računamo $d(i, j, 4)$ za sve parove čvorova. Proveravamo da li je put od i do j koji ide preko čvora 4 bolji od dosadašnjeg najboljeg. To važi za $d(1, 2, 4) = d(1, 4, 3) + d(4, 2, 3) = 4$ i $d(1, 3, 4) = d(1, 4, 3) + d(4, 3, 3) = 3$.

Možemo primetiti da nema potrebe pamtiti sve matrice prilikom izvršavanja, već samo onu iz poslednjeg koraka. To je zato što D_k diretkno zavisi samo od vrednosti u matrici D_{k-1}. Štaviše, možemo ceo postupak uraditi nad jednom matricom. U prvom slučaju (za najkraći put od i do j koji ne sadrži k) se odgovarajuće polje matrice ne menja jer je $d(i,j,k) = d(i,j,k-1)$. U drugom slučaju polje matrice postaje $d(i,j,k) = d(i,k,k-1) + d(k,j,k-1)$, pri čemu smo videli da važi $d(i,k,k-1) = d(i,k,k)$ i $d(k,j,k-1) = d(k,j,k)$ što nam omogućava da polja matrice ažuriramo bilo kojim redosledom jer $d(i,j,k)$ zavisi samo od onih polja koja su jednaka u D_{k-1} i D_k.

```
vector< vector<int> >
floyd_warshall(vector< vector< pair<int, int> > >& g) {
  int n = g.size();

  // Postavljamo početne vrednosti matrice D po formuli
  // za bazni slučaj
  vector< vector<int> >
    D(n, vector<int>(n, numeric_limits<int>::max()));
  for(int u = 0; u < n; u++)
    for(auto e : g[u]) {
      int v = e.first, w = e.second;
      D[u][v] = w;
    }
  for(int u = 0; u < n; u++)
    D[u][u] = 0;

  // Redom prolazimo kroz k i računamo Dk
  for(int k = 0; k < n; k++)
    // Ovde je redosled obilaska polja nebitan, pa idemo
    // najjednostavnije - po redovima, a mogli smo i nasumično
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        D[i][j] = min(D[i][j], D[i][k] + D[k][j]);

  return D;
}
```

Složenost ovog algoritma je očigledno $O(|V|^3)$.

Zadatak - Najkraći put sa najmanje grana

Zadat je graf G i čvor u tog grafa. Za svaki čvor v grafa G odrediti najkraći put od u do v. Ukoliko postoji više takvih puteva odabrati onaj koji ima najmanji broj grana.

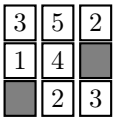
Ideja rešenja je da udaljenost čvora od početnog ne pamtimo više kao jedan broj d već kao par vrednosti (d,k) gde je d dužina najkraćeg puta, a k broj grana na tom putu. Tada možemo reći da je čvor u bliži početnom od čvora v ako

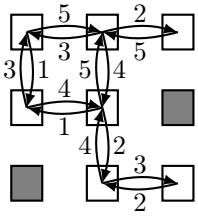
je $(d_u, k_u) \leq (d_v, k_v)$ pri čemu ove parove poredimo leksikografski (tj. prvo po dužini najkraćeg puta, a ako su dužine jednake onda po broju grana na putu). Ovime smo osigurali da će Dajkstrin algoritam proizvesti najkraći put i to baš onaj put te dužine koji ima najmanji broj grana.

Zadatak - Lavirint

Matricom je zadat lavirint. Neka polja su neprohodna, a neka prohodna. Svako prohodno polje u sebi ima upisan broj. Odrediti put od gornjeg levog do donjeg desnog polja koji ima minimalan zbir vrednosti upisanih u polja.

Ovako zadatu matricu možemo posmatrati kao graf. Polja matrice poistovećujemo sa čvorovima grafa. Kako sa jednog polja možemo preći na prohodna, njemu susedna polja, možemo ih povezati granama. Težina svake grane je onda jednaka broju upisanom u polje do kog ta grana vodi. Posmatranjem ovog problema na taj način uočavamo da Dajkstrinim algoritmom jednostavno određujemo optimalnu putanju.





Pri samoj implementaciji ne moramo zapravo da formiramo ovakav graf. Za svako polje čuvamo njegovu dosadašnju udaljenost. Prolazak kroz susede čvora i njihovo ažuriranje pri Dajkstrinom algoritmu zamenjujemo prolaskom kroz susede polja. Njihove udaljenosti ažuriramo na isti način kao kod klasičnog Dajkstrinog algoritma, pri čemu sada za težinu grane uzimamo broj upisan u tom polju.

Zadatak - Policijske stanice

Dat je skup gradova zajedno sa putevima koji ih povezuju i njihovim dužinama. Poznato je u kojim gradovima postoje policijske stanice. Za svaki grad odrediti njegovu minimalnu udaljenost od njemu najbliže policijske stanice.

Zadatak se svodi na jednostavnu modifikaciju Dajkstrinog algoritma. Umesto da na početku postavljamo samo jedan čvor na udaljenost 0, to činimo za sve čvorove koji odgovaraju gradovima sa policijskom stanicom. Nakon takve inicijalizacije pokretanjem Dajkstrinog algoritma dobićemo minimalnu udaljenost svakog čvora do njemu najbliže policijske stanice. Ovo funkcioniše zato što Dajkstrin algoritam radi i onda kada imamo više odvojenih oblasti obrađenih čvorova. To se vidi u dokazu optimalnosti kod Dajkstrinog algoritma - nigde nismo koristili činjenicu da je indukovani podgraf dobijen uzimanjem samo obrađenih čvorova povezan.

Zadatak - Aerodromi

Dat je skup aerodroma zajedno sa letovima koji ih povezuju i njihovim dužinama. Odrediti minimalnu ukupnu dužinu puta jednog putnika ako su zadati početni i krajnji aerodrom i ako je pritom poznat tačno jedan aerodrom preko kog putnik želi obavezno da putuje.

Pošto tražimo najkraći put od čvora u do čvora v pri čemu zahtevamo da taj put sadrži čvor w, koristimo činjenicu da je njegova dužina jednaka $d(u, w) + d(w, v)$, gde je $d(x, y)$ dužina najkraćeg puta od čvora x do čvora y. Dovoljno je pokrenuti Dajkstrin algoritam dva puta: prvo iz čvora u kako bismo odredili $d(u, w)$, a zatim iz čvora w za $d(w, v)$. Sabiranjem te dve vrednosti dobijamo traženu vrednost.

Zadatak - Ciklus negativne težine

Neka je dat graf G. Ispitati da li G sadrži ciklus negativne težine (težina ciklusa je zbir težina njegovih grana).

Zadatak možemo rešiti primenom Flojd-Varšalovog algoritma. Nakon određivanja najkraćih puteva između svaka dva čvora, dovoljno je proveriti da li postoji bar jedan čvor v takav da je $d(v, v) < 0$. Ako postoji, to nam direktno govori da postoji put od v do v (dakle ciklus) negativne težine. Sa druge strane, ako postoji ciklus negativne dužine, mora postojati i bar jedan čvor sa pomenutim svojstvom. Lako se uveravamo u to odabirom bilo kog čvora sa tog ciklusa.