

Vaja 5 – Floyd-Warshallov algoritem za iskanje najkrajših poti med vsemi vozlišči grafa

1. Splošna predstavitev problema

V prejšnji nalogi smo iskali najkrajše poti v grafu iz enega samega vozlišča v vsa ostala. Naravna razširitev problema pa je iskanje najkrajših poti med vsemi vozlišči grafa. Pri tem gre za težak problem, ki ga uspešno rešuje Floyd-Warshallov algoritem (v literaturi se pojavljajo tudi imena Floydov algoritem, Roy-Floydov algoritem, WFI algoritem). Ker iščemo najkrajše poti med vsemi vozlišči v grafu, je treba hraniti veliko količino podatkov. Floyd-Warshallov algoritem za to uporablja dve matriki $\mathbf{D}$ in $\mathbf{\Pi}$. V prvi se v procesu iskanja hranijo cene najkrajših poti, v drugi pa so podatki s pomočjo katerih lahko najkrajše poti rekonstruiramo. Algoritem uporablja tehniko dinamičnega programiranja.

Denimo, da imamo graf $G = \{V, P\}$, pri čemer imamo v množici V n vozlišč, ($n = 0, 1, \dots, n-1$). Ceno najkrajše poti med poljubnima vozliščema i in j v grafu G , v kateri se lahko pojavijo zgolj vozlišča od $0, \dots, k$ označimo z $d_{i,j}^{(k)}$. Pri tem velja, da $d_{i,j}^{(-1)}$ označuje ceno direktne povezave med vozliščema i in j . Če taka povezava ne obstaja, to označimo z vrednostjo neskončno, saj to pomeni, da izostanek take povezave doprinese neskončen strošek k iskani poti. Pri iskanju najcenejše poti obstajata dve možnosti. Pot, ki vsebuje vozlišča iz $\{0, \dots, k\}$ je lahko dejansko najcenejša, ali pa je dejanska optimalna pot sestavljena iz poti, iz vozlišča i do vozlišča k in naprej iz vozlišča k do j . Dolžina najcenejše poti iz vozlišča i v vozlišče j , ki vsebuje vozlišča iz množice $\{0, \dots, k\}$, je določena s ceno $d_{i,j}^{(k)}$ in če obstaja cenejša pot, bo ta sestavljena iz najcenejše poti med i in k , ki vsebuje vozlišča iz množice $\{0, \dots, k\}$ in najcenejše poti iz vozlišča k do j . Iz tega lahko zapišemo rekurzivno formulo za izračun najkrajše poti, ki je naslednja:

$$d_{ij}^{(k)} = \begin{cases} c_{ij} & , \text{ če je } k = -1 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & , \text{ če je } k \geq 0 \end{cases} \quad (1)$$

Z izračunom rekurzivne formule 1, dobimo vrednosti najcenejših poti med vsemi vozlišči v grafu, toda če želimo te poti tudi rekonstruirati, potrebujemo še eno matriko, to je matriko predhodnikov, ki jo označimo s $\mathbf{\Pi}^{(k)}$. V tej matriki je vedno, ko najdemo cenejšo pot to potrebno shraniti, kar naredimo z naslednjo enačbo:

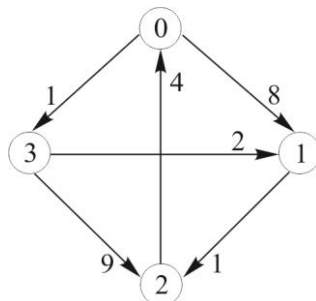
$$\pi_{ij}^{(-1)} = \begin{cases} \text{NIL} & , \text{ če velja } i = j \vee w_{ij} = \infty \\ i & , \text{ če velja } i \neq j \vee w_{ij} < \infty \end{cases}$$

in

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & , \text{ če velja } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & , \text{ če velja } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases} \quad (2)$$

Čeprav lahko s pomočjo enačbe 1 izračunamo najcenejšo pot med poljubnima vozliščema v grafu, pa bi bila rekurzivna uporaba formule vseeno precej draga, saj bi lahko nekatere dele poti računali večkrat. Zaradi tega raje uporabimo iterativno varianto, kjer bomo iskali najkrajšo pot med vozlišči tako, da bomo najprej poiskali direktno povezavo med vozliščema i in j , če ta obstaja, nato pa bomo postopoma množico vmesnih vozlišč povečevali, dokler ne bodo v njen vključena vsa možna vozlišča.

Poglejmo delovanje Floyd Warshallovega algoritma na grafu s slike 1.



Slika 1: Izhodiščni graf

Graf na sliki 1 lahko zapišemo z naslednjo matriko sosednosti, ki je hkrati tudi izhodiščna matrika cen najkrajših poti:

$$\mathbf{D}^{(-1)} = \mathbf{C} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & \infty & 0 & \infty \\ \infty & 2 & 9 & 0 \end{bmatrix}. \quad (3)$$

V skladu z enačbo 2 je potrebno določiti še izhodiščno matriko sosednosti $\Pi^{(-1)}$, ki je v našem primeru naslednja:

$$\Pi^{(-1)} = \begin{bmatrix} \text{NIL} & 0 & \text{NIL} & 0 \\ \text{NIL} & \text{NIL} & 1 & \text{NIL} \\ 2 & \text{NIL} & \text{NIL} & \text{NIL} \\ \text{NIL} & 3 & 3 & \text{NIL} \end{bmatrix}. \quad (4)$$

Če se osredotočimo na vozlišče 0, lahko vidimo, da lahko pridemo iz tega vozlišča direktno v vozlišči 1 in 3. Cena te poti pa je 8 in 1. Če k povečamo za 1, na vrednost 0, dobimo naslednji matriki $\mathbf{D}$ in Π :

$$\mathbf{D}^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & 9 & 0 \end{bmatrix} \quad \Pi^{(0)} = \begin{bmatrix} \text{NIL} & 0 & \text{NIL} & 0 \\ \text{NIL} & \text{NIL} & 1 & \text{NIL} \\ 2 & 0 & \text{NIL} & 0 \\ \text{NIL} & 3 & 3 & \text{NIL} \end{bmatrix}. \quad (5)$$

Z razširitvijo množice vmesnih vozlišč na poti, se za prvo vozlišče ni spremenilo nič, vidimo pa lahko, da smo preko vozlišča 0 med seboj povezali vozlišči 2 in 1 ter 2 in 3. Prva pot je sedaj $2 \rightarrow 0 \rightarrow 1$, cena te poti pa je $4 + 8 = 12$. Druga od obeh poti je $2 \rightarrow 0 \rightarrow 1$, cena te poti pa je $4 + 1 = 5$. Zaradi sprememb v matriki najcenejših poti, je potrebno spremeniti tudi matriko predhodnikov. Vse spremembe so v enačbi 5 tudi označeni.

S povečanjem vrednosti k na 1, je potrebno na novo izračunati matriko najcenejših poti in matriko sosednosti. Ti matriki sta prikazani v enačbi 6.

$$\mathbf{D}^{(1)} = \begin{bmatrix} 0 & 8 & \mathbf{9} & 1 \\ \infty & 0 & 1 & \infty \\ 4 & 12 & 0 & 5 \\ \infty & 2 & \mathbf{3} & 0 \end{bmatrix} \quad \Pi^{(1)} = \begin{bmatrix} \text{NIL} & 0 & \mathbf{1} & 0 \\ \text{NIL} & \text{NIL} & 1 & \text{NIL} \\ 2 & 0 & \text{NIL} & 0 \\ \text{NIL} & 3 & \mathbf{1} & \text{NIL} \end{bmatrix} \quad (6)$$

Vidimo lahko, da smo z vključitvijo vozlišča 1 v množico vmesnih vozlišč povezali vozlišče 0 z vozliščem 2 in sicer je trenutna najkrajša pot naslednja: $0 \rightarrow 1 \rightarrow 2$. Cena te na novo najdene poti je 9 ($8+1$). Našli smo tudi novo cenejšo pot med vozliščema 3 in 2, ki je naslednja: $3 \rightarrow 1 \rightarrow 2$. Cena te poti je 3 ($2+1$). Ker v množici vmesnih vozlišč nimamo še vseh vozlišč, k povečamo za 1, na vrednost 2. Dobimo naslednji matriki najkrajših poti in predhodnikov:

$$\mathbf{D}^{(2)} = \begin{bmatrix} 0 & 8 & 9 & 1 \\ \mathbf{5} & 0 & 1 & \mathbf{6} \\ 4 & 12 & 0 & 5 \\ \mathbf{7} & 2 & 3 & 0 \end{bmatrix} \quad \Pi^{(2)} = \begin{bmatrix} \text{NIL} & 0 & 1 & 0 \\ \mathbf{2} & \text{NIL} & 1 & \mathbf{0} \\ 2 & 0 & \text{NIL} & 0 \\ \mathbf{2} & 3 & 1 & \text{NIL} \end{bmatrix} \quad (7)$$

Iz enačbe 7 lahko vidimo, da smo z vključitvijo vozlišča 2 v množico vmesnih vozlišč našli kar tri nove cenejše poti. Najprej smo med seboj povezali vozlišči 1 in 0 s potjo $1 \rightarrow 2 \rightarrow 0$, s ceno 5 ($1+4$), med seboj pa smo povezali tudi vozlišči 3 in 0. Pot med njima je naslednja: $3 \rightarrow 1 \rightarrow 2 \rightarrow 0$, s ceno 7 ($2+1+4$). Našli pa smo tudi cenejšo pot med vozliščema 1 in 3, ki je $1 \rightarrow 2 \rightarrow 0 \rightarrow 3$ s ceno 6 ($1+4+1$). Ker v množici vmesnih vozlišč še vedno manjka eno vozlišče, k povečamo na 3. Pri tem dobimo:

$$\mathbf{D}^{(3)} = \begin{bmatrix} 0 & \mathbf{3} & \mathbf{4} & 1 \\ 5 & 0 & 1 & 6 \\ 4 & \mathbf{7} & 0 & 5 \\ 7 & 2 & 3 & 0 \end{bmatrix} \quad \Pi^{(3)} = \begin{bmatrix} \text{NIL} & \mathbf{3} & \mathbf{1} & 0 \\ 2 & \text{NIL} & 1 & 0 \\ 2 & \mathbf{3} & \text{NIL} & 0 \\ 2 & 3 & 1 & \text{NIL} \end{bmatrix} \quad (8)$$

Z vključitvijo še zadnjega vozlišča v množico vmesnih vozlišč, smo našli še tri ugodnejše poti in sicer iz vozlišča 0 do vozlišča 1 in do vozlišča 2. Prva pot je tako $0 \rightarrow 3 \rightarrow 1$ s ceno 3 ($1+2$), druga pa je $0 \rightarrow 3 \rightarrow 1 \rightarrow 2$ s ceno 4 ($1+2+1$). Tretja pot povezuje vozlišče 2 z 1 in je $2 \rightarrow 0 \rightarrow 3 \rightarrow 1$ s ceno 7 ($4 + 1 + 2$).

Ker smo v množico vmesnih vozlišč vključili vsa vozlišča, z iskanjem zaključimo. V matriki najcenejših poti imamo sedaj cene vseh najcenejših poti znotraj grafa G . Rekonstrukcijo poti z matriko predhodnikov pa opravimo s klicem algoritma v izpisu 2.

2. Pomoč pri implementaciji

Implementacija predstavljenega algoritma je dokaj preprosta. Kot vhod bo služila matrika sosednosti C , ki jo do sedaj že dobro poznate. Potrebovali pa bomo še matriko najkrajših poti D in matriko predhodnikov Π , ki pa ju definiramo lokalno. Ker v posameznih iteracijah ne naredimo v matrikah nobene spremembe, ki bi vplivala na pravi rezultat, je dovolj za vsako od matrik le eno dvodimenzionalno polje. Pseudokod Floyd-Warshallovega algoritma si lahko ogledamo v izpisu 1.

```
function FLOYD_WARSHALL(C)
begin
  for i:=0 to n-1 do
    for j:=0 to n-1 do
      D[i][j] := C[i][j]
      if i ≠ j and C[i][j] ≠ ∞ then
        Π[i][j] := i
      else
        Π[i][j] := NIL
    for k:=0 to n-1 do
      for i:=0 to n-1 do
        for j:=0 to n-1 do
          if D[i][j] > D[i][k]+D[k][j] then
            D[i][j] := D[i][k]+D[k][j]
            Π[i][j] := Π[k][j]
        return D, Π
    end
```

Izpis 1: Pseudokod procedure FLOYD_WARSHALL

Algoritem iz izpisa 1 nam tvori matriko dolžin najkrajših poti in matriko predhodnikov. Seveda pa je sedaj potrebno te najkrajše poti tudi rekonstruirati. Ker je enačba 1, s katero smo najkrajše pot iskali rekurzivna, mora biti taka tudi funkcija za rekonstrukcijo poti. Ta funkcija je prikazana v izpisu 2:

```
procedure IZPIS_POTI(Π, s, g)
begin
  if s = g then
    izpiši s
  else
    if Π[s][g] = NIL then
      izpiši „med “ s „ in “ g „ ni poti.“
    else
      IZPIS_POTI(Π, s, Π[s][g])
      izpiši g
  end
```

Izpis 2: Pseudokod procedure IZPIS_POTI

Vhod v funkcijo `IZPIS_POTI` je matrika predhodnikov Π in začetno in končno vozlišče s in g , nimamo pa podatka o ceni poti, tega je potrebno predhodno prebrati iz matrike najkrajših poti in ga tudi izpisati pred ali po klicu te funkcije.

3. Zahteve naloge

Implementirati je potrebno Floyd-Warshallov algoritem za iskanje najkrajših poti v danem usmerjenem grafu, ki ne bo imel več kot 20 vozlišč.

Ob zagonu programa se mora zagnati meni, ki je prikazan na sliki 2.

Floyd-Warshallov algoritem - izbira: 1 Preberi graf iz datoteke 2 Zagon algoritma 3 Izpis najkrajše poti med vozliščema s in g 4 Konec Vaša izbira:

Slika 2: Začetni meni ob zagonu aplikacije

Ob izbiri menijske postavke *Preberi graf iz datoteke* poženite priloženo proceduro, za branje podatkov iz datoteke, ki naj prebere datoteko z imenom *graf.txt*. Pri izbiri postavke 2 je potrebno zagnati proceduro za generiranje matrike najkrajših poti in matrike sosednosti. Dokler ti dve matriki nista generirani, uporabnik ne more izbrati menijske vrstice 3. Ob pritisku na to postavko, mora namreč uporabnik vpisati začetno in končno vozlišče, nakar program izpiše najkrajšo pot in njeno ceno. Program se konča, ko uporabnik izbere menijsko postavko *Konec*.