Zadaća 12

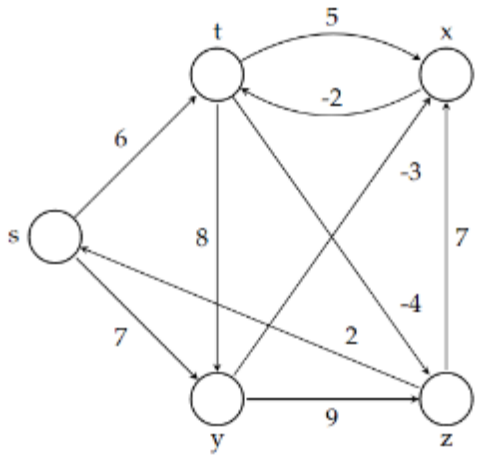
1. Bellman – Fordov algoritam na usmjerenom grafu algoritam je koji pronalazi najkraći put u grafu s težinama koje mogu biti negativne. To čini uz pomoć funkcije RELAX koja za svaki brid provjerava može li se udaljenost od izvornog vrha smanjiti tako što uzmemo u obzir težinu tog brida (put kroz taj vrh). Ako može, smanji je i spoji vrh s novim prethodnikom. Algoritam na grafu sa slike radi tako što prvo stavi sve udaljenosti od izvornog vrha (d[v] za svaki vrh) na beskonačno i prethodnike svih vrhova (π(v) za svaki vrh) na NIL. Udaljenost d[s] (udaljenost izvornog vrha od izvornog vrha) stavi na 0. Vrhova ima 5 pa ćemo 5 puta proći kroz petlju u kojoj je još jedna petlja koja za svaki brid poziva funkciju RELAX. Pseudokod algoritma izgleda ovako:

```
BELLMAN-FORD(G, w, s):
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |V| - 1:
        for (u, v) in E:
                RELAX(u, v, w)
for (u, v) in E:
        if d[v] > d[u] + w(u, v)
        return FALSE
return TRUE
```

```
RELAX(u, v, w):
if d[v] > d[u] + w(u, v):
    d[v] = d[u] + w(u, v)
    π(v) = u
```

For petlja for (u, v) in E nakon relaxanja svih bridova još provjerava postoji li ciklus negativne težine dohvatljiv iz izvornog vrha. Ako postoji, algoritam se može vrtiti do -beskonačno pa vratimo false. Inače vratimo true.

Na grafu sa slike to izgleda ovako:



Dobili smo redoslijed bridova, pa ulazimo u prvu for petlju koja ide od 1 do 4. Napravit ću tablicu za prvu iteraciju petlje kada se svi bridovi procesiraju:

	x	t	y	z
0, NIL	∞, NIL	∞, NIL	∞, NIL	∞, NIL
0, NIL	7, r	∞, NIL	∞, NIL	∞, NIL
0, NIL	7, r	∞, NIL	∞, NIL	2, r
0, NIL	7, r	8, s	9, s	2, r

Tablica u prvom redu prikazuje vrijednosti d i π za svaki vrh na početku, u drugom redu isto nakon procesiranja bridova (t, x), (t, y), (t, z), (x, t), (y, z) i (z, x). U trećem redu nakon procesiranja (z, s) i u četvrtom redu nakon procesiranja (s, t) i (s, y).
Sada idemo u drugu iteraciju petlje.

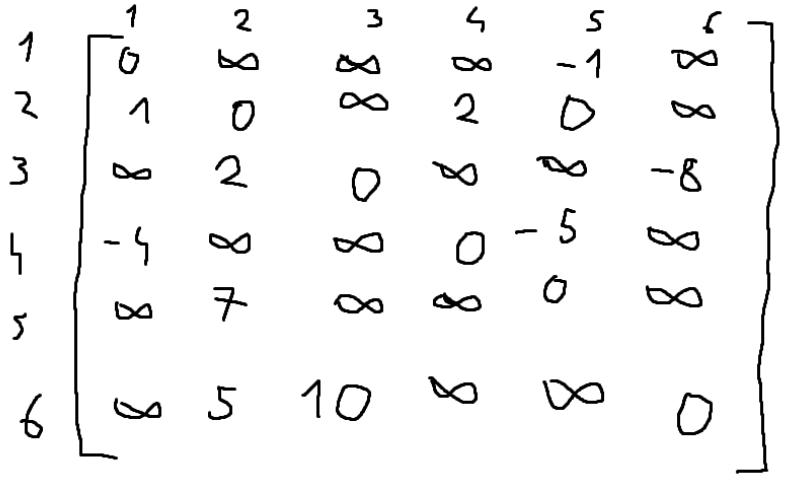
	x	t	y	z
0, NIL	7, r	8, s	9, s	2, r
0, NIL	7, r	5, x	9, s	2, r

U drugoj iteraciji petlje tablica izgleda ovako. Prvi red je nakon procesiranja bridova (t, x), (t, y), (t, z). Drugi red nakon procesiranja brida (x, t) i ostalih bridova. U trećoj i četvrtoj iteraciji petlje ništa se neće mijenjati jer smo već minimizirali udaljenosti pa imamo udaljenosti i π za svaki vrh na kraju algoritma u drugom redu tablice gore.

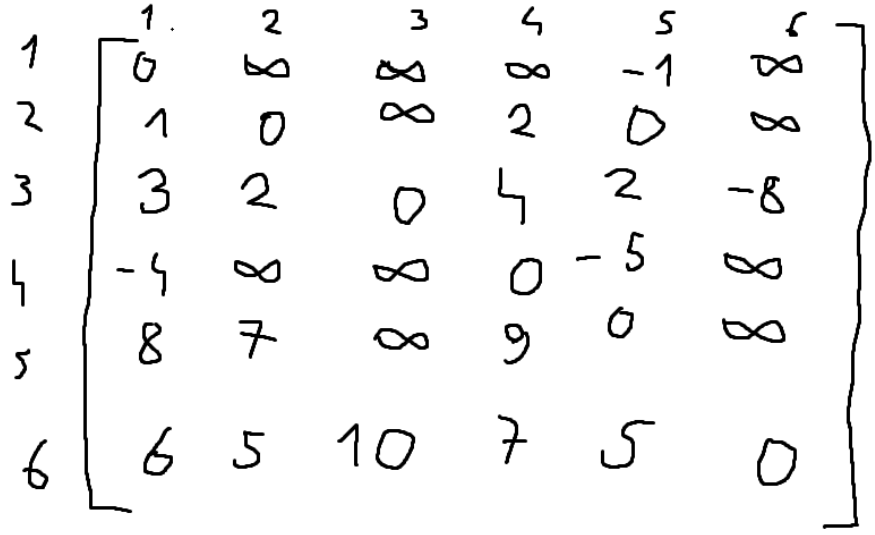
2. Floyd – Warshallov algoritam na usmjerenom grafu pronalazi najkraće puteve između svaka dva vrha u grafu s težinama koje mogu biti negativne. To radi tako što prvo inicijalizira matricu D^0 dimenzija V * V i svaku ćeliju A[i][j] ispuni s direktnim udaljenostima između i-tog i j-tog vrha. Ako ne postoji brid između i-tog i j-tog vrha, tu ćeliju ispuni s beskonačno. Na primjeru grafa sa slike koji ima 6 vrhova, ovako će izgledati matrica D^0:

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	∞	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	3	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

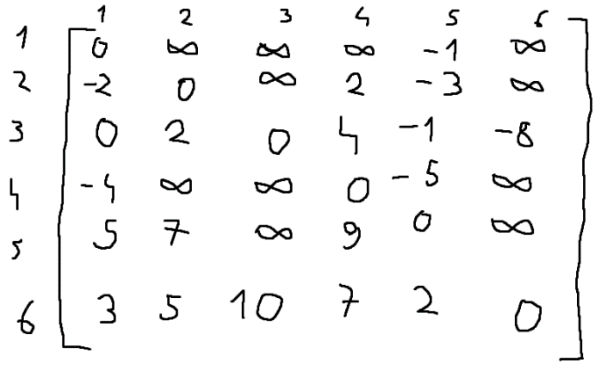
Sada ostavi vrijednosti u prvom redu i prvom stupcu, a ostatak puni na sljedeći način: Neka je k srednji vrh između neka dva vrha. D[i][j] se puni s vrijednošću D[i][k] + D[k][j] ako je D[i][j] > D[i][k] + D[k][j]. U prvoj iteraciji vanjske petlje k je vrh 1 pa izračunavamo udaljenosti od početnog do krajnjeg vrha kroz vrh 1. Na primjer, u početnoj matrici D^0 udaljenost D[2][5] je beskonačno, a kada zbrojimo udaljenosti D[2][1] + D[1][5] udaljenost će se ažurirati na 0. Nakon prve iteracije matrica D^1 izgleda ovako:



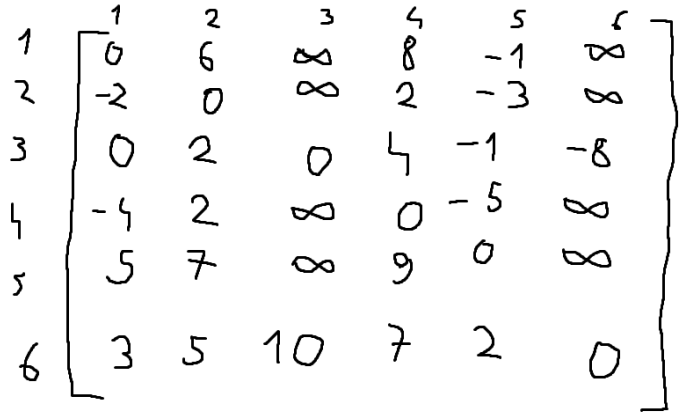
Slično napravimo za D^2 samo što ostavimo vrijednosti u drugom retku i drugom stupcu i ažuriramo ostatak pa će D^2 izgledati ovako:



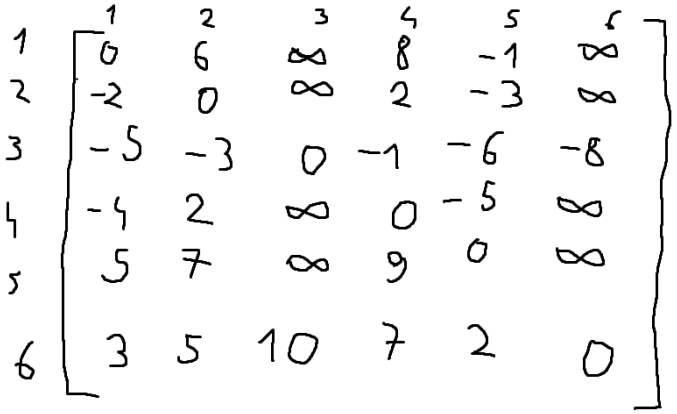
Slično za D^3, s tim da kod D^3 nemamo promjena pa će ona izgledati isto kao i D^2. D^4 će izgledati ovako:



D^5 će izgledati ovako:



i na kraju D^6 ovako:



Sada smo dobili najkraći put između svih parova vrhova.

3. 1. Koristit ćemo vjerojatnost q(e) = 1 – p(e) da je određena cesta e otvorena. Možemo maknuti bridove kod kojih je p(e) = 1 jer će oni uvijek biti zatvoreni. Vjerojatnost da će neki put r koji prolazi kroz bridove e1, e2, …, en biti otvoren je dan s produktom vjerojatnosti da će ti bridovi biti otvoreni, tj:

$$Q(r) = \prod_{i=1}^{n} q(e_i)$$

Nama treba put r koji počinje iz vrha z do vrha o koii ima maksimalni Q(r). Možemo uzeti negativan logaritam obje strane i dobijemo

$$-\lg Q(r) = -\lg \prod_{i=1}^{n} q(e_i) = \sum_{i=1}^{n} -\lg q(e_i)$$

pa -lgq(ei) možemo označiti s w(ei).

Koristeći Dijkstra algoritam možemo odrediti put od z do o s minimalnim vrijednostima w(ei) kojeg možemo rekonstruirati zbog određivanja π u RELAX koraku Dijkstra algoritma. Tako smo minimizirali -lgQ(r), tj. maksimizirali Q(r) pa smo dobili put s najmanjom vjerojatnosti da će se cesta zatvoriti. Vremenska složenost ovog algoritma je O(E + VlgV) jer Dijkstra algoritam dominira runtime.

2. Modificiramo Dijkstra algoritam kako bi našli najteži kamion koji može ići od vrha z do vrha o. Najveća dopuštena težina je informacija na bridu. Modificiramo Dijkstru na sljedeći način:
1. INITIALIZE-SINGLE-SOURCE dodjeljuje vrijednost beskonačno početnom vrhu i vrijednost 0 svim ostalim vrhovima.
2. Umjesto EXTRACT-MIN koristimo EXTRACT-MAX i maksimalno prioritetni red kako bi odredili koji vrh posjećujemo sljedeći, dakle prvo posjećujemo onaj put koji može podnijeti najveću težinu.
3. u RELAX koraku koristimo min umjesto zbrajanja. Tako ćemo održavati najveću dopuštenu težinu.
Što se tiče najkraćeg puta kojim taj kamion može proći, možemo maknuti sve bridove koji imaju maksimalnu dopuštenu težinu manju od težine tog najtežeg kamiona. Tada možemo pokrenuti Dijkstra algoritam na taj graf (na bridovima su nam l – duljine dijelova autoceste) i dobiti taj najkraći put. Ukupna vremenska složenost biti će vremenska složenost Dijkstre – O(E + VlgV).

3 – Iz grafa maknemo sve bridove koji nemaju istočnog susjeda i na tom grafu pozovemo Dijkstra algoritam. Na kraju ćemo dobiti najkraći put od vrha z do vrha o ako idemo samo istočno.

4. a) Trebamo modelirati latenciju kao skup linearnih diferencijalnih jednadžbi. Kada prilagodimo jednadžbu (1) dobijemo:

$$\forall\, i,j \in \{1,2,\ldots,n\}: \quad x_i - x_j \;\leq\; -A[i,j] \qquad (4)$$

, kada prilagodimo jednadžbu (2) dobijemo:

$$\begin{aligned}
&\forall\, i,j \in \{1,2,\ldots,n\}: \quad x_i - x_j \;\geq\; -B[i,j] \\
&\forall\, i,j \in \{1,2,\ldots,n\}: \quad x_j - x_i \;\leq\; B[i,j] \\
&\forall\, i,j \in \{1,2,\ldots,n\}: \quad x_i - x_j \;\leq\; B[j,i]
\end{aligned} \qquad (5)$$

, a jednadžbe (4) i (5) ekvivalentne su jednadžbi (3) ako ovako konstruiramo matricu:

$$\forall\, i,j \in \{1,2,\ldots,n\}: \quad C[i,j] = \min(-A[i,j], B[j,i])$$

b) Trebamo pokazati da Bellman – Fordov algoritam, kada se pokrene na grafu s ograničenjima (3) minimizira izraz (max{xi} – min{xi}), podložan jednadžbi (3) i s ograničenjem xi <= 0 za svaki xi.

Prvo ćemo pokazati lemu:
Lema 1. Kad Bellman – Fordov algoritam pokrenemo na grafu s ograničenjima (3), max{xi} = 0.
Dokaz.
Neka je p = <s, v1, …, vk> najkraći put između vrha s i vrha vk kojeg smo dobili Bellman -Fordovim algoritmom na grafu s ograničenjima. Po strukturi najkraćih puteva, <s, v1> mora biti najkraći put od s do v1. Po konstrukciji, brid od s do v1 ima težinu 0. Po tome

znamo da je x1 = delta(s, v1) = w(<s, v1>) = 0, a u kombinaciji s ograničenjem xi <= 0 za svaki xi dobijemo da je max{xi} = 0.

S tom lemom možemo dokazati teorem:

Teorem 1. Kada Bellman – Fordov algoritam pokrenemo na grafu s ograničenjima (3), on minimizira izraz (max{xi} – min{xi}).
Dokaz.
Pošto je max{xi} = 0 (po prethodnoj lemi), to pokazuje da Bellman – Ford maksimizira min{xi}. Neka je xk = min{xi} u najkraćem putu kojeg smo dobili Bellman - Fordovim algoritmom i neka je p najkraći put od s = v0 do vk, p = <v0, v1, …, vk>. Težina tog puta je $w(p) = \sum_{i=0}^{k-1} w_{i(i+1)} = w(\langle v_0, v_1 \rangle) + \sum_{i=1}^{k-1} C[i, i+1] = \sum_{i=1}^{k-1} C[i, i+1]$.
Put korespondira sljedećem skupu ograničenja:

$$\begin{aligned} x_1 - x_0 &\leq 0 \\ x_2 - x_1 &\leq C[1,2] \\ x_3 - x_2 &\leq C[2,3] \\ &\cdots \\ x_k - x_{k-1} &\leq C[k-1,k] \end{aligned}$$

, a kad sumiramo ograničenja dobijemo:

$$\begin{aligned} x_k &\leq \sum_{i=1}^{k-1} C[i, i+1] \\ &= w(p) \end{aligned}$$

, što znači da u svakom rješenju koje ima ograničenja (3), xk ne može biti veći od w(p) što je težina najkraćeg puta od s do vk. Kako Bellman – Ford postavi xk na vrijednost najkraćeg puta, to implicira da je xk najveći što može biti.

c) Trebamo dati efikasan algoritam koji minimizira ukupnu duljinu izvođenja radova.
Algoritam je sljedeći:

1. Konstruiraj graf s ograničenjima (3). Ako je C[i, j] = beskonačno, nemoj dodati brid (vj, vi) u graf.
2. Pokreni Bellman – Forda na tom grafu kako bi dobio rješenje {x1, x2, …, xn}.
3. Postavi y = min{xi} i izračunaj novo rješenje xi' = xi – y za svaki xi. Izlaz treba biti {x1', x2', …, xn'}.

Ovaj algoritam ne dodaje bridove koji imaju beskonačnu težinu C[i, j] za popravljanje vremenske složenosti (ti bridovi neće imati utjecaj na najkraći put kojeg će pronaći Bellman - Ford). min{xi'} = min{xi} – y = 0 implicira da algoritam minimizira max{xi} u skladu s ograničenjima.

Vremenska složenost algoritma je O(V^2) za prvi korak, O(VE) za drugi korak i O(V) za treći korak, pa je ukupna složenost algoritma O(V^2 + VE).