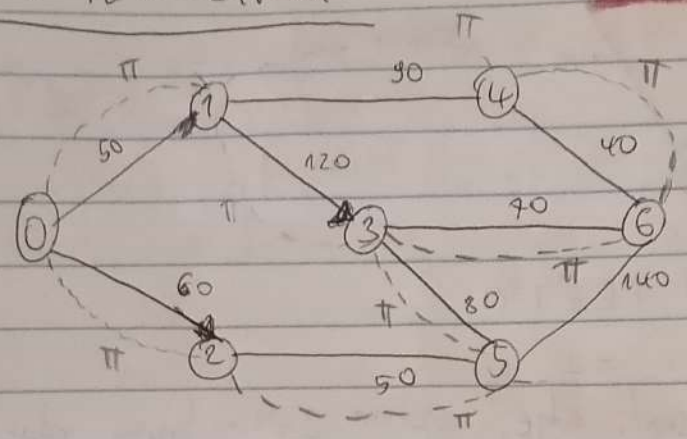
Zahoda 11

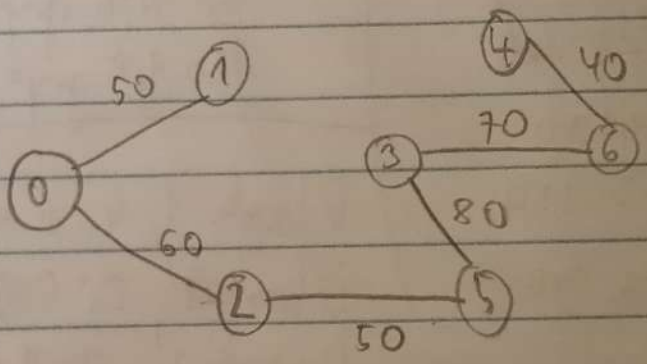
Zad. 3.

PRIMOV ALGORITAM:

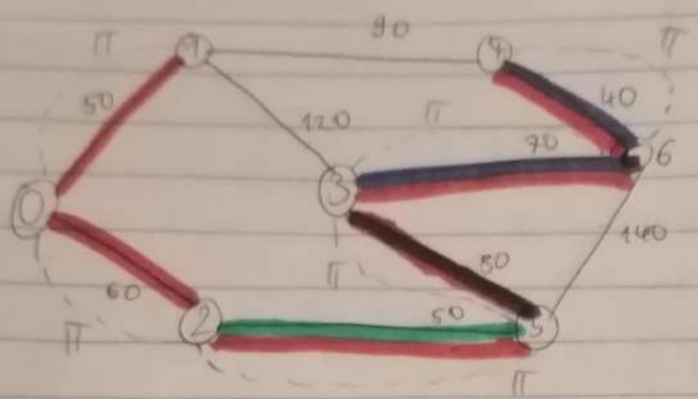


Q:	0	1	2	3	4	5	6
	0	∞	∞	∞	∞	∞	∞
=	50	60	∞	∞	∞	∞	
=	60	120	90				
=	(120)	90	50	∞			
	80	90	= 140				
=	90		70				
	40		=				
=							

PRIPADAJOČI MST:



KRUSKALOV ALGORITAM:



PRVO, SORTIRAMO BRIDOVE PO TEŽINI:

4-6 : 40
2-5 : 50
0-1 : 50
0-2 : 60
3-6 : 70
3-5 : 80
~~5-6 : 140~~
1-4 : 90
1-3 : 120
5-6 : 140

Zatim, iteriramo kroz svaki edge i vidimo 2 čvora kojima edge pripada. Ako su već unificirane u istoj grupi, preskačemo korak, inače unificiramo nodeove.
Color coded su grupe u grafu
2. slučaj dijeli se na 2 slučaja:
Ako A, B grupe, |A| ≥ |B|, A ∪ B
Ako A, B grupe, |B| < |A|, B ∪ A

U koraku 1, 4 i 6 nisu unificirane, dakle, {4} ∪ {6} = {4, 6}
U koraku 2, 2 i 5 nisu unificirane, dakle {2} ∪ {5} = {2, 5}
U koraku 3: 0 i 1 nisu unificirane, dakle {0} ∪ {1} = {0, 1}
U koraku 4, {0, 1} i {2, 5} nisu unificirane, dakle {2, 5} ∪ {0, 1} = {0, 1, 2, 5}
U koraku 5, {4, 6} i 3 nisu unificirane, dakle {3} ∪ {4, 6} = {3, 4, 6}
U koraku 6, 2 grupe koje imamo nisu unificirane, pa dobivamo:
{3, 4, 6} ∪ {0, 1, 2, 5} = {0, 1, 2, 3, 4, 5, 6}
Dalje nije bitno jer su svi nodes unificirani

- ALGORITAM KORISTI ~~xxx~~ HEURISTIKU TEŽINSKE UNIJE I DISJUNKTNE SKUPOVE ZA UNION-FIND OPERACIJE.

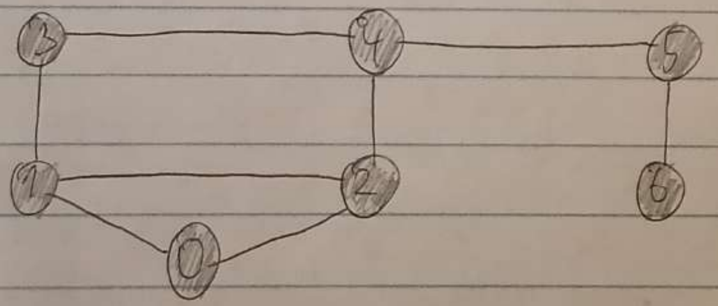
Objašnjenje Primovog algoritma:

1. Napravi 2 polja key i π. Sve $i \in V$ postavi na ∞, sve $v \in \pi$ postavi na NIL. Posebno, key[root] = 0.
2. Inicijaliziraj PRIORITY QUEUE Q sa vrhovima $v \in V$.
3. Dok $Q \neq \emptyset$, za sve susjede v od u, u = extract_min(Q):
   Ako je $v \in Q$ i $w(u,v) < key[v]$:
   $\pi[v] = u$, $key[v] = w(u,v)$

π sadrži parente svakog vrha, key sadrži minimalan cost (u,v) $\forall v$, u direktno spojen s v.

Zad. 1.

BFS:



1. Stavi bilo koji vrh u red
2. Uzmi prvi item u redu i postavi mu visited na true
3. $\forall v$, t.d. $(u,v) \in E$ i !visited(v), stavi v u queue red.
4. Ponavljaj dok red nije prazan

Vrh visited ⇒ vrh bojam u sivo.
Za primjer krećem od vrha (1).

RED: ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~

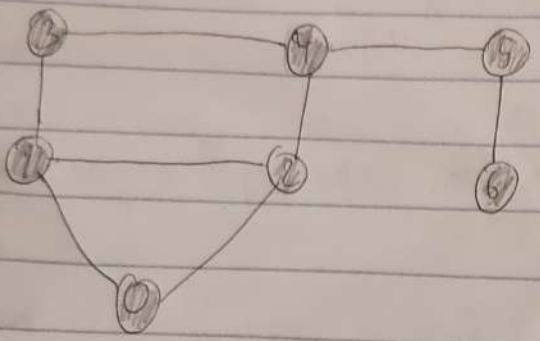
VISITED: ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~
1. 2. 3. 4. 5. 6. 7.

* U 4. ponavljaj 2 i 3.

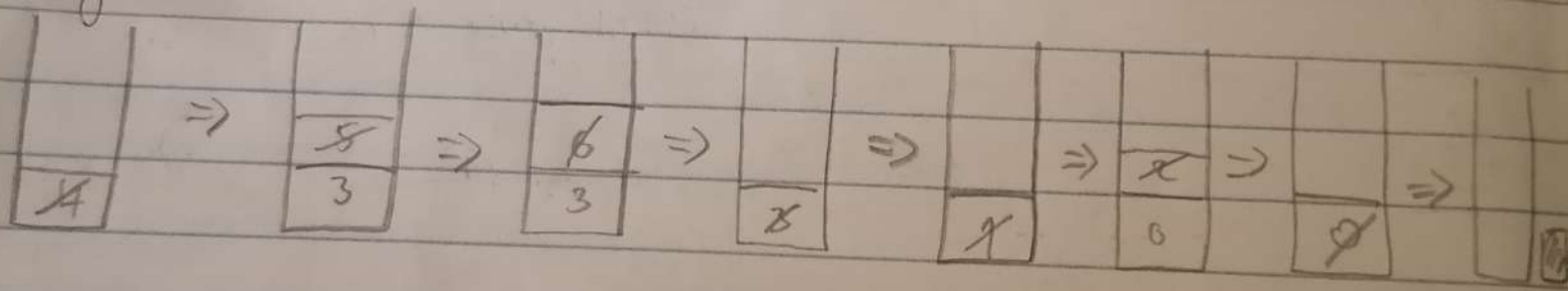
DFS:

1. Stavi bilo koji vrh na stack
2. Uzmi vrh s vrha stoga i postavi visited[u] = true
   ↑ u
3. $\forall v : (u,v) \in E$: ! visited v ⇒ stavi v na vrh stoga.
4. Ponavljaj korake 2. i 3. dok stog nije prazan. Paziti pritom da se stavljaju redundantne vrhove na stack (gray i black s predavanja).

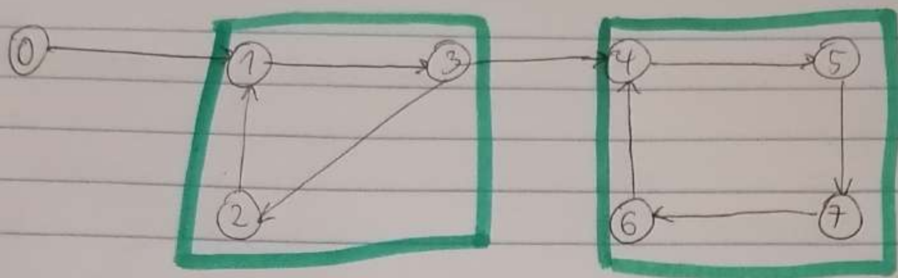
Pr. (krećem od vrha 4.)



Stog:



Zad. 2. Strongly connected components.



SCC: Dijelovi u grafu (podgrafovi) gdje iz svakog vrha možemo doći u svaki. Također, ne možemo izaći iz jednog SCC i vratiti se u njega.
{0} nije SCC jer nema petlju sam na sebe (tj. vezu).

Algoritam za pronalazak: Tarjanov algoritam.
Koristimo tehniku low-link vrijednosti: ~~vrijednost~~ LL vrijednost nodea je node s najmanjim vremenom otkrivanja kod vođenja DFS, a taj node s najmanjim vremenom otkrivanja mora biti dohvatljiv iz našeg nodea.

Problem: Node iz kojeg radimo DFS određuje LOW-LINK vrijednosti ⇒ mijenja SSC-ove ili čak daje netočne SSC.
Rješenje: Tarjanov algoritam: održava invarijantu t.d. početni node postaje nerelevantan

Objašnjenje:

1. Označi svaki node kao neposjećen.
2. Započni DFS. Kod posjete nodea daj mu vrijeme posjeta i LLV. Također, označi trenutni node kao visited i postavi ga u „seen" stack.
3. Na DFS ~~[illegible]~~ povratku, ako se prethodni node nalazi na stacku onda je LLV trenutnog nodea: min (LLV(nt), LLV(u))

4. Nakon posjeta svim susjedima, ako je trenutni vrh ~~[illegible]~~ započeo parnu komponentu popaj vrhove sa stoga dok ne dođeš do trenutnog vrha.