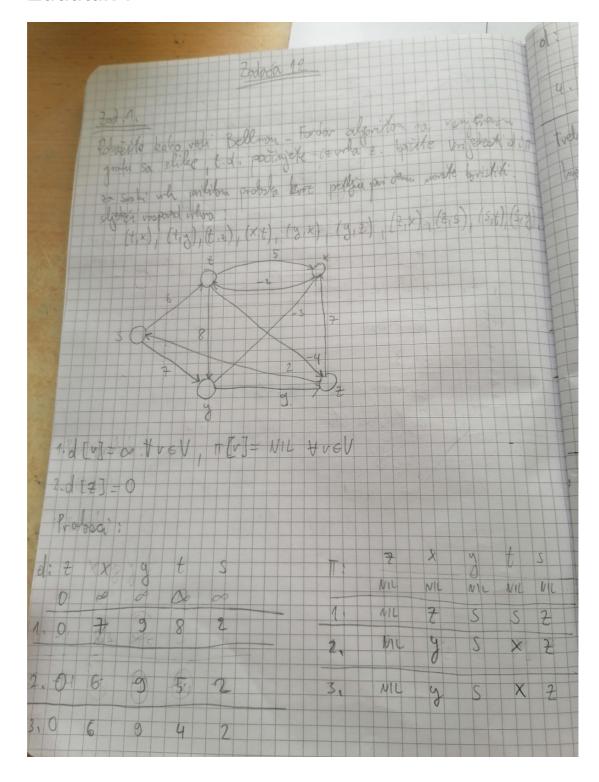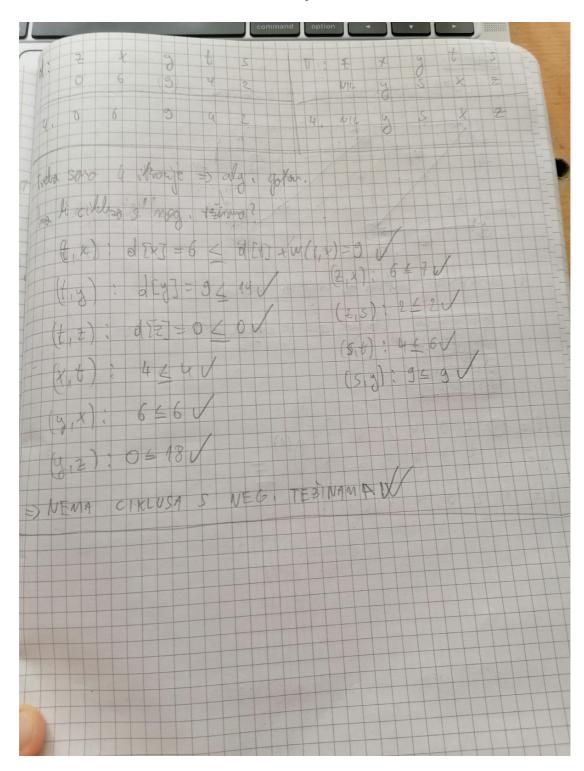# Zadaca 12

Zadatak 1

Zadatak 2

D0: 0 INF INF INF 5 INF
1 0 INF 2 INF INF
INF 2 0 INF INF -8
-4 INF INF 0 3 INF
INF 7 INF INF 0 INF
INF 5 10 INF INF 0

D1: 0 INF INF INF 5 INF
1 0 INF 2 6 INF

```
INF 2 0 INF INF -8
-4 INF INF 0 1 INF
INF 7 INF INF 0 INF
INF 5 10 INF INF 0

D2: 0 INF INF INF 5 INF
1 0 INF 2 6 INF
3 2 0 4 8 -8
-4 INF INF 0 1 INF
8 7 INF 9 0 INF
6 5 10 7 11 0

D3: 0 INF INF INF 5 INF
1 0 INF 2 6 INF
3 2 0 4 8 -8
-4 INF INF 0 1 INF
8 7 INF 9 0 INF
6 5 10 7 11 0

D4: 0 INF INF INF 5 INF
-2 0 INF 2 3 INF
0 2 0 4 5 -8
-4 INF INF 0 1 INF
5 7 INF 9 0 INF
3 5 10 7 8 0

D5: 0 12 INF 14 5 INF
-2 0 INF 2 3 INF
0 2 0 4 5 -8
-4 8 INF 0 1 INF
5 7 INF 9 0 INF
3 5 10 7 8 0

D6: 0 12 INF 14 5 INF
-2 0 INF 2 3 INF
-5 -3 0 -1 0 -8
-4 8 INF 0 1 INF
5 7 INF 9 0 INF
3 5 10 7 8 0
```

Zadatak 3

```
In [ ]:  # a)
         from queue import PriorityQueue
         INF = 1e9
         def dijkstra(graph, start, end, parent, prob):
             prob[start] = 1 # vjerojatnost da ce se moci doci do starta je maksim
             pq = PriorityQueue()
```

```python
        # -1, jer trebam extraxt-max umjesto extract-min operaciju
        pq.put((-1, start))
        while(not pq.empty()):
            curr = pq.get()[1]
            for edge in graph.edges.whereStart(curr):
                if(prob[edge.end] < prob[curr] * edge.prob):
                    prob[edge.end] = prob[curr] * edge.prob
                    pq.put((-prob[edge.end], edge.end))



    def shortestPath(graph, start, end):
        for edge in graph.edges:
            if edge.prob == 1:
                graph.removeEdge(edge) # makni ove s vjerojatnoscu 1 jer sigu
        parent = [None] * graph.vertices
        prob = [0] * graph.vertices
        dijkstra(graph, start, end, parent, prob)
        path = []
        curr = end
        while(curr != None):
            path.append(curr)
            curr = parent[curr]
        path.reverse()
        return path
```

```python
    # b)
    # 1)
    def heaviestTruckInit(graph, start, end, weight):
        dp = [-1 for i in range(graph.size)]
        dp[0] = INF
        heaviestTruck(graph, start, end, dp)
        return dp

    def heaviestTruck(graph, start, end, dp):
        max_weight = -1
        prev = None
        for edge in graph.edges.whereEnd(end):
            if(dp[edge.start][end] == -1):
                dp[edge.start] = heaviestTruck(graph, start, edge.start, dp)
            max_weight = max(max_weight, min(dp[edge.start], edge.weight))
        dp[end] = max_weight
        return dp[end]

    # objasnjenje:
    # Do svakog vrha v iz V moze se doci sa maksimalnom tezinom kamiona w. Ta
    # Npr, ako od osijeka do Belog manastira mogu otputovati kamionom tezine
```

```python
    # 2)
    # Posto su tezine sigurno pozitivne (ceste ne mogu imati negativnu duzinu
    def dijkstra(graph, start, end, truckWeight):
        dp = heaviestTruckInit(graph, start, end, truckWeight)
        for vertex in graph.vertices:
            if(dp[vertex] < truckWeight):
                for edge in graph.edges.whereStart(vertex):
                    graph.removeEdge(edge)
        dist = [INF for i in range(graph.size)]
        dist[start] = 0
        pq = PriorityQueue()
```

```python
        pq.put((0, start))
        while(not pq.empty()):
            curr = pq.get()[1]
            for edge in graph.edges.whereStart(curr):
                if(dist[edge.end] > dist[curr] + edge.weight):
                    dist[edge.end] = dist[curr] + edge.weight
                    pq.put((dist[edge.end], edge.end))
        return dist[end]
```

In [ ]:
```python
# c)
# vertexi predstavljaju gradove
# Istocno bi znacilo da od odredista moze ici samo prema istoku, tj. da n
# takoder, ne smije ici juzno/sjeverno, sto znaci da ne smije 2x posjetit

# IDEJA: augmentirati graf tako da svakom vertexu u grafu nadodam vrijedn

from queue import Queue
def modified_bfs(graph, start, end):
    level = [-1 for i in range(graph.size)]
    level[start] = 0
    q = Queue()
    q.put(start)
    while(not q.empty()):
        curr = q.get()
        for edge in graph.edges.whereStart(curr):
            if(level[edge.end] == -1):
                level[edge.end] = level[curr] + 1
                q.put(edge.end)

def dijkstra(graph, start, end, truckWeight):
    dp = heaviestTruckInit(graph, start, end, truckWeight)
    # O(V)
    for vertex in graph.vertices:
        if(dp[vertex] < truckWeight):
            for edge in graph.edges.whereStart(vertex):
                graph.removeEdge(edge)
    # O(V + E)
    modified_bfs(graph, start, end) # ovo augmentira graf s levelima

    # O(E * lg V)
    dist = [INF for i in range(graph.size)]
    dist[start] = 0
    pq = PriorityQueue()
    pq.put((0, start))
    while(not pq.empty()):
        curr = pq.get()[1]
        for edge in graph.edges.whereStart(curr):
            if(dist[edge.end] > dist[curr] + edge.weight):
                dist[edge.end] = dist[curr] + edge.weight
                pq.put((dist[edge.end], edge.end))
    return dist[end]
```

Zadatak 4

Zad. 4.

a) Iz (1) $\forall i, j \in \{1, 2, \dots, n\} : x_i + A[i,j] \leq x_j$

(2) $\forall i, j \in \{1, 2, \dots, n\} : x_i + B[i,j] \geq x_j$

Premetanjem prve jednadžbe, imamo:

$$x_i - x_j \leq -A[i,j] \quad (4)$$

Premetanjem druge jednadžbe, imamo:

$$x_i + B[i,j] \geq x_j$$
$$x_i - x_j \geq -B[i,j]$$
$$x_j - x_i \leq B[i,j] \iff x_i - x_j \leq B[j,i] \quad (5)$$

$$x_i - x_j \leq -A[i,j] \quad i \quad x_i - x_j \leq B[j,i] \iff x_i - x_j \leq \min(-A[i,j], B[j,i])$$

$$\implies C[i,j] = \min(-A[i,j], B[j,i])$$

b) Pokažite da Bellman-Ford algo, koji se pokrene na grafu s ograničenja jednadžbe (3) minimizira izraz (max$\{x_i\}$ - min$\{x_i\}$) podložan je uz (3) i ograničenje $x_i \leq 0 \ \forall x_i$.

Rj.

Kad se Bellman-Ford pokrene na grafu s ograničenjima, $w_{i,j} = C[i,j]$

Također, moramo dodati jedan vertex u graf, s, ako ga nema, t.d. je za svakim drugim pravcem s bridom 0. To je jer ... ... od ... veža do ... koji nema nova bridova.

- ovim pristupom slične s ug. uvjetima je više kompatibilni s B-F algo. [3]

Lema: $\exists x_i \leq 0$. Potrebno: $\max\{x_i\} = 0$.

Neka je $p = \langle s, v_1, \ldots, v_k \rangle$ najkraći put $s \to v_k \Rightarrow (s, v_1)$ najkraći put od s do v_1. Po konstrukciji $\langle s, v_1 \rangle = 0$, a svaki $x_j \leq 0 \Rightarrow \max\{x_j\} \leq$

Tm. Kad se B-F pokrene na grafu s ograničenjima, minimizira $\{\max\{x_i\} - \min\}$

Neka je p put $\langle v_0, v_1, \ldots, v_k \rangle$ i $v_0 = S$.

$$W(p) = \sum_{i=0}^{k-1} W_i(i,i+1) = \underbrace{W(\langle v_0, v_1 \rangle)}_{0} + \sum_{i=1}^{k} \underbrace{C[i, i+1]}_{} =$$

$\max\{x_i\} = 0 \Rightarrow \max\{x_i\} - \min\{x_i\}$
$= -\min\{x_i\}$

$\Rightarrow$ Umanjuje $-\min\{x_i\} \Leftrightarrow$ uvećava $\min\{x_i\}$

$$= \underbrace{\sum_{i=1}^{k-1} C[i, i+1]}$$

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

$$(\ast) \quad x_0 + x_k = x_k \leq \sum_{i=1}^{k-1} C[i, i+1]$$

$$= W(p)$$

c) **Algoritam:**

1) Konstruiraj graf s ograničenjima u grupi (3), ako $C[i,j] = \infty$, nema odgovarajući put (v_j, v_i) u graf

2) Pokreni Bellman-Forda na grafu s ograničenjima da dobiješ vrijednosti $\{x_1, x_2, \ldots, x_n\}$

3] Kada je $w = \max\{x_i\}$, izračunaj novu grupu gdje $x_i' = x_i - w, \forall x_i$.

Ispisati $\{x_1', x_2', \ldots, x_n'\}$