EC 504
Spring, 2021
HW 5

Due Tuesday, March 30, 8PM on Gradescope.

1. (10 points) Assume you have a scheduling problem with 10 customers. Customer i has value V_i, and must be processed before deadline T_i. The processing time of each customer is exactly one unit of time. The values of V_i and T_i are listed below as arrays:

   ```
   V = {3,4,7,5,2,3,5,8,9,6}
   T = {2,3,4,3,1,5,7,3,4,6}
   ```

   Find the optimal sequence of jobs that can be scheduled in order to complete as much value as possible.

   Solution:

   This algorithm admits a greedy solution. Rank stuff by decreasing value, and schedule it if it fits in partial schedule, otherwise skip it. To check it fits into partial schedule, order tasks in increasing deadline and see if it is executable.

   Sorting by decreasing value, the task order is: 9,8,3,10,7,4,2,1,6,5. Thus, we first add task 9, then 8, then 3, then 10, then 7, then 4, all of which fit into a schedule. To check this, schedule this group in order of earliest deadline first. This yields a schedule of 4, 8, 9, 3, 10, 7, which meets all the deadlines.

   Now try to add task 2, with deadline 3. It would have to be added in the first 3 slots, which would then violate the deadline for task 3, which would slip to slot 5. Thus, task 2 is skipped. Similarly, task 1 is skipped. Task 6 has deadline 5, so it can be added to the schedule to form: 4,8,9,3,6,10,7. Task 5 cannot be added because it has deadline 1 and some other task would have to be dropped.

   The final value is 43.

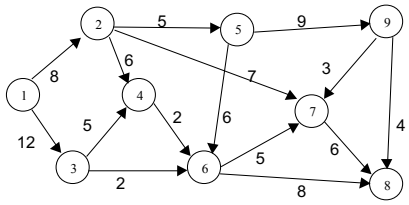
2. (10 points) Assume you have a scheduling problem with 10 customers. Customer i has processing time T_i. The value of each customer is the same. The objective is to minimize the sum across all customers of the completion time of each customer. The values of T_i are listed below as an array:

   ```
   T = {2,3,4,9,1,5,7,8,10,6}
   ```

   Find the optimal sequence of jobs to minimize the sum of the completion times.

   Solution: Another greedy problem, with solution to schedule customers in order of increasing processing time. This leads to the order by task number as: 5,1,2,3,6,10,7,8,4,9.

3. (10 points) Consider the directed weighted graph on the right. Show the distance estimates computed by each step of the Bellman-Ford algorithm for finding a shortest path tree in the graph, starting from vertex 3 to all other vertices.

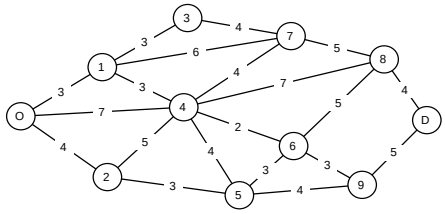


   Solution:

   ```
   Initial Distances:  D(1) = inf; D(2) = inf; D(3) = 0; D(4) = inf; D(5) = inf;
                       D(6) = inf; D(7) = inf; D(8) = inf; D(9) = inf;
   Q = {3}
   Scan vertex 3:          D(1) = inf; D(2) = inf; D(3) = 0; D(4) = [5]; D(5) = inf;
                       D(6) = [2]; D(7) = inf; D(8) = inf; D(9) = inf;
   Q = {4,6}
   Scan vertex 4:          D(1) = inf; D(2) = inff; D(3) = 0; D(4) = 5; D(5) = inf;
   ```

```
                    D(6) = 2; D(7) = inf; D(8) = inf; D(9) = inf;
Q = {6}
Scan vertex 6:          D(1) = inf; D(2) = inf; D(3) = 0; D(4) = [4]; D(5) = inf;
                    D(6) = 2; D(7) = [7]; D(8) = [10]; D(9) = inf;
Q = {7,8}
Scan vertex 7:          D(1) = inf; D(2) = inf; D(3) = 0; D(4) = 4; D(5) = inf;
                    D(6) = 2; D(7) = 7; D(8) = 10; D(9) = inf;
Q = {8}
Scan vertex 8:          D(1) = inf; D(2) = inf; D(3) = 0; D(4) = 4; D(5) = inf;
                    D(6) = 2; D(7) = 7; D(8) = 10; D(9) = inf;

Done.  No more vertices in Queue means no other vertices can be reached.
Shortest path tree:
  4    7
 /    /
3 -- 6 -- 8
```

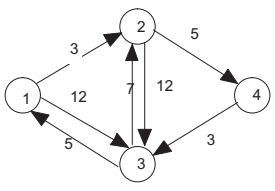
4. (10 points) Consider the weighted, undirected graph shown on the right. Assume that the edges can be traveled in both directions. Illustrate the steps of Dijkstra's algorithm for finding a shortest path from vertex O to vertex D.



Solution: As before, keep track of distances which change using brackets, and delete vertices already scanned from distance list

```
Initial Distances: D(O)=0; D(1)=inf; D(2) = inf; D(3)=inf; D(4)=inf; D(5) = inf;
                    D(6) = inf; D(7) = inf; D(8) = inf; D(9) = inf; D(D) = inf
Scan vertex O: D(1)=[3]; D(2) = [4]; D(3)=inf; D(4)=[7]; D(5) = inf;
                    D(6) = inf; D(7) = inf; D(8) = inf; D(9) = inf; D(D) = inf

Scan vertex 1: D(2) = 4; D(3)=[6]; D(4)=[6]; D(5) = inf;
                    D(6) = inf; D(7) = [9]; D(8) = inf; D(9) = inf; D(D) = inf
Scan vertex 2: D(3)=6; D(4)=6; D(5) = [7];
                    D(6) = inf; D(7) = 9; D(8) = inf; D(9) = inf; D(D) = inf
Scan vertex 3: D(4)=6; D(5) = 7;
                    D(6) = inf; D(7) = 9; D(8) = inf; D(9) = inf; D(D) = inf
Scan vertex 4: D(5) = 7;
                    D(6) = [8]; D(7) = 9; D(8) = [14]; D(9) = inf; D(D) = inf
Scan vertex 5: D(6) = 8; D(7) = 9; D(8) = 14; D(9) = [11]; D(D) = inf
Scan vertex 6: D(7) = 9; D(8) = [13]; D(9) = 11; D(D) = inf
Scan vertex 7: D(8) = 13; D(9) = 11; D(D) = inf
Scan vertex 8: D(9) = 11; D(D) = [17]
Scan vertex 9: D(D) = [16]

Shortest path in reverse:  D--9--5--2--O, length 16.
```

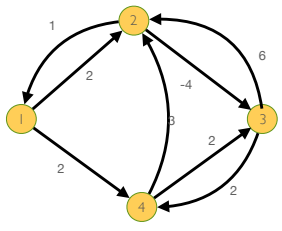
5. (10 points) Consider the graph indicated on the right. This is a directed graph. Use the Floyd-Warshall algorithm to find the shortest distance for all pairs of vertices. Show your work as a sequence of 4 by 4 tables.



Solution: Here is the initial table, obtained from the edges in the graph. the following tables expand the algorithm.

```
D(i,j)    j=1   j=2    j=3   j=4
i=1        0     3     12    inf
i=2       inf    0     12     5
i=3        5     7      0    inf
i=4       inf   inf     3     0

iteration: k = 1 as intermediate vertex
D(i,j)    j=1   j=2    j=3   j=4
i=1        0     3     12    inf
i=2       inf    0     12     5
i=3        5     7      0    inf
i=4       inf   inf     3     0

iteration: k = 2 as intermediate vertex
D(i,j)    j=1   j=2    j=3   j=4
i=1        0     3     12    [8]
i=2       inf    0     12     5
i=3        5     7      0    [12]
i=4       inf   inf     3     0

iteration: k = 3 as intermediate vertex
D(i,j)    j=1   j=2    j=3   j=4
i=1        0     3     12     8
i=2       [17]   0     12     5
i=3        5     7      0    12
i=4       [8]   [10]    3     0

iteration: k = 4 as intermediate vertex
D(i,j)    j=1   j=2    j=3   j=4
i=1        0     3     [11]   8
i=2      [13]    0      [8]   5
i=3        5     7      0    12
i=4        8    10      3     0
```

6. (10 points) Consider the graph indicated on the right. This is a directed graph. Note the negative weight edge from 2 to 3. Use Johnson's algorithm, followed by Dijkstra's algorithm, to find the shortest distance for all pairs of vertices.

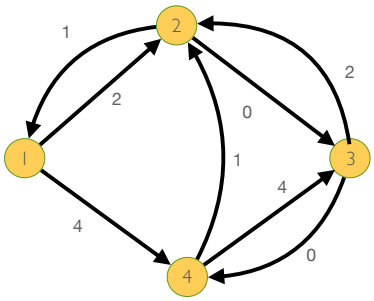


Solution: The first step in Johnson's algorithm is to add a vertex s with outgoing edges to vertices 1, 2, 3, 4, with weight 0. Then, we find the shortest distance to each vertex from s using the Bellman-Ford algorithm. Let's do that first.

```
Initial Distances:  D(s) = 0; D(1) = inf; D(2) = inf; D(3) = 0; D(4) = inf
Q = {s}
Scan vertex s:         D(1) = 0; D(2) = 0; D(3) = 0; D(4) = 0;
Q = {1,2,3,4}
Scan vertex 1:         D(1) = 0; D(2) = 0; D(3) = 0; D(4) = 0;
Q = {2,3,4}
```

```
Scan vertex 2:        D(1) = 0; D(2) = 0; D(3) = -4; D(4) = 0;
Q = {3,4}
Scan vertex 3:        D(1) = 0; D(2) = 0; D(3) = -4; D(4) = -2;
Q = {4}
Scan vertex 4:        D(1) = 0; D(2) = 0; D(3) = -4; D(4) = -2;

Done. These distances now form the weights for the second part of the algorithm, where
$h(1) =  h(2) = 0, h(3) = -4, h(4) = -2.$  The revised weights on edge $(u,v)$ are
$w_h(u,v) = w(u,v) + h(u) - h(v)$.  Thus, the revised weights have $w_h(2,3) = 0, w_h(3,2) = 2$, and
```

The revised weights are shown below:



We now use Dikjstra's algorithm from each of the 4 starting vertices.

```
Starting at 1,    D[1] = 0, D[2] = inf, D[3] = inf, D[4] = inf.
PQ = {1, 2, 3, 4}
Scan 1:   D[2] =2, D[4] = 4, D[3] = inf
PQ = {2,4,3}
Scan 2: D[3] = 0, D[4] = 4;
PQ = {3,4}
Scan 3:   D[4] = 2
PQ = {4}
Pop 4 and we are done.  Shortest paths are 1 --> 2 --> 3 --> 4, Distances 0, 2, -2, 0.

Starting at 2, D[1]= inf, D[2] = 0, D[3] = inf, D[4] = inf;
PQ = {2, 1, 3, 4}
Scan 2: D[1] = 1, D[3] = 0, D[4] = inf;
PQ = {3,1,4]
Scan 3: D[1] = 1, D[4] = 0;
PQ = {4,1}
Scan 4: D[1] = 1;
PQ = {1}
Pop 1, we are done.   Shortest paths are 2 --> 3 --> 4,
                                          \
                                           1
True distances D[2] = 0, D[1] = 1, D[3] = -4, D[4] = -2;

Starting at 3, D[1]= inf, D[2] = inf, D[3] = 0, D[4] = inf;
PQ = {3, 1, 2, 4}
Scan 3: D[1] = inf, D[2] = 2, D[4] = 0;
PQ = {4,2,1]
Scan 4: D[2] = 1, D[1] = inf;
PQ = {2,1}
Scan 2: D[1] = 2;
PQ = {1}
```

```
Pop 1, we are done.   Shortest paths are 3 --> 4 --> 2 --> 1,
True distances D[3] = 0, D[1] = 6, D[2] = 5, D[4] = 2;

Starting at 4, D[1]= inf, D[2] = inf, D[3] = inf, D[4] = 0;
PQ = {4, 1, 2, 3}
Scan 3: D[1] = inf, D[2] = 1, D[3] = 4;
PQ = {2,3,1]
Scan 2: D[3] = 1, D[1] = 2;
PQ = {3,1}
Scan 3: D[1] = 2;
PQ = {1}
Pop 1, we are done.   Shortest paths are 4 --> 2 --> 3 ,
                                                  \
                                                   1
True distances D[3] = -1, D[1] = 4, D[2] = 3, D[4] = 0;
```

7. (10 points) Explain how you modify Dijkstra's shortest path algorithms on a directed graph with non-negative edge weights to count the number of shortest paths from a given origin n to a destination vertex d.

   Solution: Note the following step in Dijkstra's algorithm for a directed graph $G = (V, E)$ with non-negative weight function $w()$: Let $D(k)$ denote the estimated distance to vertex k, and assume we are scanning vertex n (that is, vertex n is the vertex that we have pulled out of the priority queue that has the shortest distance estimate $D(n)$. The scan step is:

   For each edge $(n, k) \in E$, if $D(n) + w(n, k) < D(k)$, then $D(k) = D(n) + w(n, k), Parent(k) = n$.

   We modify Dijkstra's algorithm as follows: We initialize the number of shortest paths to each vertex as $Paths(k) = 0$ for $k \in V$. For the origin o, we set $Paths(o) = 1$.

   Then, we modify the above step in Dijkstra's algorithm as follows:

   For each $(n, k) \in E$,

   - if $D(n) + w(n, k) < D(k)$, then $D(k) = D(n) + w(n, k), Parent(k) = n, Paths(k) = Paths(n)$
   - else if $D(n) + w(n, k) == D(k)$, $Paths(k) = Paths(k) + Paths(n)$

   What this does is, when we find a path whose distance estimate to vertex k is equal to the distance to vertex k, we increment the number of shortest paths by the number of shortest paths to this new parent. If we find a shorter distance estimate, we initialize the number of shortest paths found to the number of shortest paths to the parent. This has the same complexity and optimality properties of Dijkstra's algorithm.

8. (10 points) In city streets, the length of an edge often depends on the time of day. Suppose you have a directed graph of streets connecting vertices that represent intermediate destinations, and you are given the travel time on the edge as a function of the time at which you start to travel that edge. Thus, for edge e, you are given $d_e(t)$, the time it takes to travel edge e if you start at time t. These travel times must satisfy an interesting ordering property: You can't arrive earlier if you started later. That is, if $s < t$, then $s + d_e(s) <= t + d_e(t)$. Suppose that you start at time 0 at the origin vertex 1. Describe an algorithm for computing the minimum time path to all vertices when travel times on edges are time dependent and satisfy the ordering property.

   Solution: Turns out Dijkstra's algorithm is exactly right for this problem. Dijkstra's algorithm scans vertices in the order of arrival time. The travel time on the edges for the vertex that are being scanned can be computed because we know the time at which that vertex is reached. The only difference is that the travel times on edges are computed as a function of the travel time to the origin vertex of the edge, when that vertex is permanently labeled by the algorithm.