Data Structures and Algorithms – (COMP SCI 2C03)
Winter 2021
Tutorial - 8

March 29, 2021

1. Draw the SPT for source 0 of the edge-weighted digraph shown in Figure 1, and give the parent-link representation of the Shortest path tree (SPT) using the below algorithms:

   (a) Dijkstra's algorithm

   (b) Queue Bellman-Ford algorithm

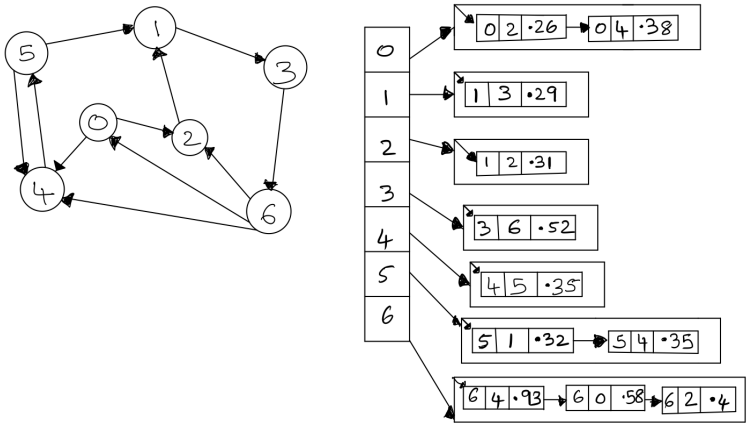
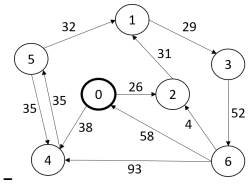


Figure 1: Undirected weighted edge graph

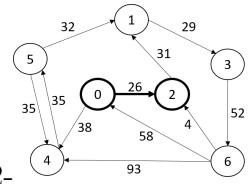
Answer: (a) Dijkstra's algorithm steps:

1-



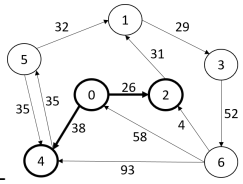
Vertex	EdgeTo	DistTo
1		
2	0	26
3		
4	0	38
5		
6		

2-



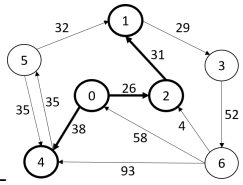
Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3		
4	0	38
5		
6		

3-



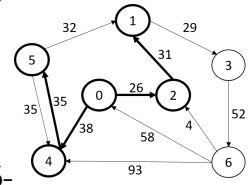
Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3		
4	0	38
5	4	73
6		

4-



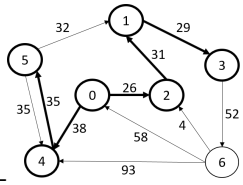
Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3	1	86
4	0	38
5	4	73
6		

5-



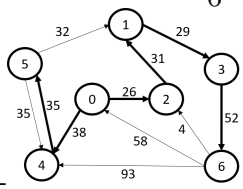
Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3	1	86
4	0	38
5	4	73
6		

6-



Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3	1	86
4	0	38
5	4	73
6	3	108

7-

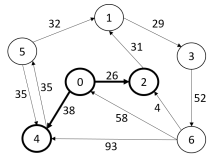


Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3	1	86
4	0	38
5	4	73
6	3	108

(b) Bellman-Ford algorithm steps:

1-

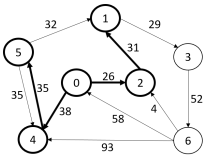
Queue
0
2
4



Vertex	EdgeTo	DistTo
1		
2	0	26
3		
4	0	38
5		
6		

2-

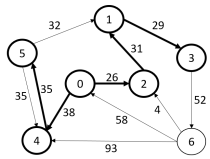
Queue
2
4
1
5



Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3		
4	0	38
5	4	73
6		

3-

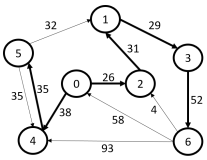
Queue
1
5
3



Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3	1	86
4	0	38
5	4	73
6		

4-

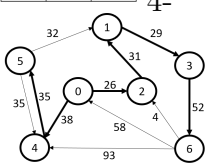
Queue
3
6



Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3	1	86
4	0	38
5	4	73
6	3	138

5-

Queue
6



Vertex	EdgeTo	DistTo
1	2	57
2	0	26
3	1	86
4	0	38
5	4	73
6	3	138

2. Draw the SPT for source 0 of the edge-weighted DAG shown in Figure 2, and give the parent-link representation of the Shortest path tree (SPT) using the topological sort shortest path algorithm.

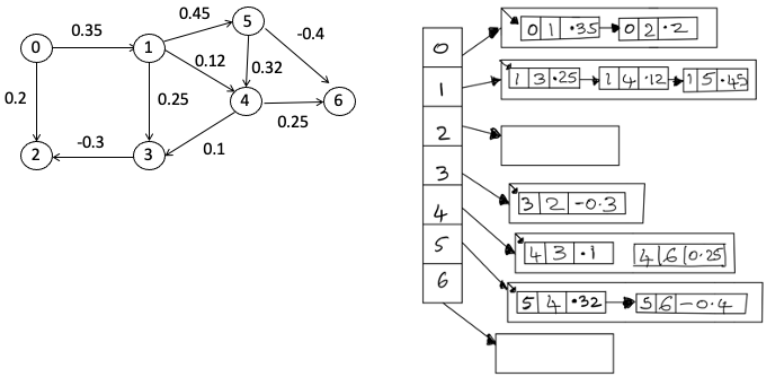
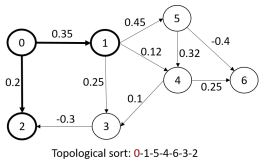


Figure 2: Undirected weighted edge graph

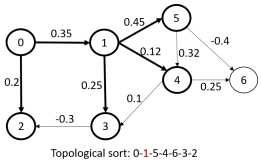
Answer: The steps of the algorithm are as follows:

1-



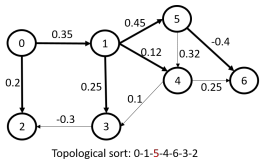
Vertex	EdgeTo	DistTo
1	0	0.35
2	0	0.2
3		
4		
5		
6		

2-



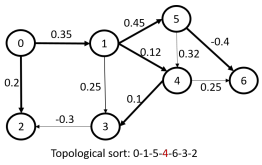
Vertex	EdgeTo	DistTo
1	0	0.35
2	0	0.2
3	1	0.6
4	1	0.47
5	1	0.8
6		

3-



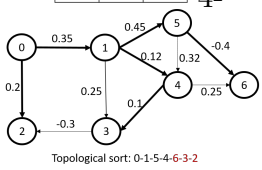
Vertex	EdgeTo	DistTo
1	0	0.35
2	0	0.2
3	1	0.6
4	1	0.47
5	1	0.8
6	5	0.4

4-



Vertex	EdgeTo	DistTo
1	0	0.35
2	0	0.2
3	4	0.57
4	1	0.47
5	1	0.8
6	5	0.4

5-



Vertex	EdgeTo	DistTo
1	0	0.35
2	0	0.2
3	4	0.57
4	1	0.47
5	1	0.8
6	5	0.4

3. What happens to Bellman-Ford if there is a negative cycle on the path from s to v?

   Answer: (Proof of Proposition Y in page 673) If there does exist a negative cycle reachable from s, the queue never empties.

4. What happens if you allow a vertex to be enqueued more than once in the same pass in the Bellman-Ford algorithm?
   Answer: Duplicated and unnecessary relax functions will be executed several times which increase the useless complexity.

5. The key-indexed counting sorts the input array of length n in ascending order. Present an algorithm for a version of key-indexed counting on an input array of length n which sorts the array in descending order.
   Answer: The solution uses the *count* array of length R and not $R+1$ as in the case of the original key-index counting solution. It stores the frequency cumulates starting with the largest number in the input array a, then storing the frequency of the next largest number in a, and so on. Then, similar to the original algorithm, it uses these values as the guide to sort and store the keys in the auxiliary array aux. Finally, it copies the sorted elements in aux to a.

   procedure *key_index_counting_desc*($A[0..n-1]$)
   $aux[0..n-1]$ ▷ aux is an array used for storing sorted elements
   $count[0..R-1]$ ▷ R = radix. *count* stores freqency cumulates
   for $i = 0$ to n **do** ▷ Compute frequency counts
   if $a[i] \neq 0$ **then**
   $count[R - a[i]] = count[R - a[i]] + 1$
   for $r = 0$ to $R - 2$ **do** ▷ Transform counts to indices
   $count[r + 1] = count[r + 1] + count[r]$
   for $i = 0$ to n **do** ▷ Distribute the records
   $aux[count[R - a[i] - 1]] = a[i]$
   $count[R - a[i] - 1] = count[R - a[i] - 1] + 1$
   for $i = 0$ to n **do** ▷ Copy back
   $a[i] = aux[i]$

6. Give a trace for LSD string sort for the keys
   Answer:

Input keys		Keys after sorting characters in the last place		Keys after sorting characters in the second last place
no		pa		ai
is		pe		al
th		of		co
ti		th		fo
fo		th		go
al		th		is
go	⇒	ti	⇒	no
pe		ai		of
to		al		pa
co		no		pe
to		fo		th
th		go		th
ai		to		th
of		co		ti
th		to		to
pa		is		to

7. How would you modify LSD string sort algorithm presented on Slide# 16 in C5P1.pdf to sort variable length strings?
   Answer: Pad the smaller length strings with a new symbol, which is smaller than all the symbols in the alphabet, to make the input strings of equal lengths. For instance, if $u = abaaa$, $v = ba$, $w = c$ are input strings of unequal length, then we pad strings v and w as follows: $v = ba - 1 - 1 - 1$ and $w = c - 1 - 1 - 1 - 1$, where $-1 < a < b < c$.