Homework 4
Data Structures and Algorithms Spring 2020

Name: Vikhyat Dhamija
RUID: 194003013
NetID: vd283

1. **Write a program that answers the following for an undirected graph: Is a graph acyclic? Run your program on graph (linked after Q2)**

   Solution:

   The Graph is called Cyclic when there is a path from one vertex in the graph ending the same vertex.

   So, observation made that if there is a cycle , then while traversing we will be moving from visited node to a already visited node. And this visited status we maintain in our normal DFS . But in case of undirected graph , our node will also look at the parent and it will be visited so that means we have to check the status of the adjacent node as well as that it is not the parent.

   .So, using the above logic I made the program . Initially I used two DFS , one to create parents of all nodes and second traversal to detect the cycle but with ideas over the internet I found that parent can be passed as parameter to DFS function again and again. So , then I used that.

   Program has been attached for the detection of this cycle in the graph. Note the result was : Graph is cyclic.

2. **Implement and execute Prim's and Kruskal's algorithms on the graph linked below (the third field is the weight of an edge). Which performs better? Explain your answer.**

   Solution:

   Three Algorithms namely Kruskal and Prims – Lazy and Eager Approach have been implemented.

   Kruskals : The code here is clearly divided into parts like:
   a. Sorting of edges with respect to weight (E Log E)
   b. Then in the ascending order we go through the edges and check whether both the vertices of edges are not making the MST cyclic using Quick Union-Find So E Log V

   So in Total our algorithm is **(E Log E)**

   Prims:

   1. **Lazy approach**
      a. In this approach , what we are doing is that we are starting with 0 then taking out the minimum weight edge
      b. Then checking the unmarked vertex in the edge
      c. Putting all the edges adjacent to that vertex
      d. Finding the minimum again
      e. Then Repeating

   The lazy approach is this that we are undecidedly putting all the edges in the priority queue and we are thinking that Ok we will decide later that it can be part of MST or not

   So here the complexity is : E edges are put in the Priority queue and every edge is pulled in/out with Log E time for Binary Heap Data Structure used for Priority Queue.

   Hence the complexity is order of **E LogE**

2. **Eager approach**

The difference between the eager approach and the Lazy approach is that as said earlier , in the lazy approach , we were undecidedly putting all the edges in the priority queue and we are thinking that Ok we will decide later that it can be part of MST or not But in eager approach as we are maintaining the indexed Priority queue of the length of the vertices , we decrease the key i.e. the edge with less weights we put in or insert the key when that index of vertex has not yet received any edge or we receive the edge to a vertex with smaller weight

So in totality we are always maintaining the heap of V size and from that we are pulling out minimum using the delete min operation.

So here the complexity is **(E Log V)**

Comparison Table :

Algorithm	Kruskals	Prims – Lazy approach	Prims – Eager approach
Complexity	E Log E	E LogE	E Log V

Time performance after executing in the Dataset:

Algorithm	Kruskals	Prims – Lazy approach	Prims – Eager approach
Time(ms)	7	16.9	Around 7

Here , Observations :

1. Prims Eager approach is faster than the Lazy approach as obvious from the complexity calculations.
2. Kruskals in my case is coming close to the Prims Eager in implementation , I think because I have implemented Kruskals based on its general approach i.e. sort the edges in terms of weights and then find cycle using union- find. I have kept here the Graph as the list of edges and not as the adjacency list forms as in the Prims – Eager and Lazy approach where we are traversing the edges which are twice inserted in the adjacency list in undirected graph to check whether its other vertex is marked or not . So with implementation constraints of data structures and as well as the fact that in our graph E= 1273 and V=250 so E/V = 4 only so E Log E is E Log V +2E so over all complexity seems to be closer towards E Log V.
3. Dense graph is a graph in which the number of edges is close to the maximal number of edges i.e. N (N-1) /2 here N is 250. **So Graph here is not so dense here.** Sparse graph is a graph in which the number of edges is close to the minimal number of edges.
4. So Data Structural issues subsumed the algorithmic effectiveness in the case and also because of the less density of the graph , differences have not come.

Note : I have taken references from Class Notes and Robert Sedgewick BOOK/Site

Note results have been checked and kept in last and the MST is coming to be unique as for distinct weights graph MST is unique(Quiz 3)

Note that the dataset used for 1,2: https://content.sakai.rutgers.edu/access/content/group/3c4e126c-962f-4655-97c6-d7299fa5ef53/Homework%20DataSet/mediumEWG.txt

3. For the edge-weighted directed acyclic graph given below, compute (i.e., manually trace) both the longest path and the shortest path.

8
13
5 4 0.35
4 7 0.37
5 7 0.28
5 1 0.32
4 0 0.38
0 2 0.26
3 7 0.39
1 3 0.29
7 2 0.34
6 2 0.40
3 6 0.52
6 0 0.58
6 4 0.93

4. (a) For the digraph with negative weights, compute (i.e. manually trace) the progress of the Bellman-Ford Algorithm.

8
15
4 5 0.35
5 4 0.35
4 7 0.37
5 7 0.28
7 5 0.28
5 1 0.32
0 4 0.38
0 2 0.26
7 3 0.39
1 3 0.29
2 7 0.34
6 2 -1.20
3 6 0.52
6 0 -1.40
6 4 -1.25

4. (b) For the digraph with a negative cycle, compute (i.e. manually trace) the progress of the Bellman-Ford Algorithm.

8
15
4 5 0.35
5 4 -0.66
4 7 0.37
5 7 0.28
7 5 0.28
5 1 0.32
0 4 0.38
0 2 0.26
7 3 0.39
1 3 0.29
2 7 0.34
6 2 0.40
3 6 0.52
6 0 0.58
6 4 0.93

Solution 3 and 4 : The hand-written computation with **detailed explanation** has been attached for both**(at end)**

5. Implement a DFS and BFS traversal for the data-set of the undirected road network of New York City. The graph contains 264346 vertices and 733846 edges. It is connected, contains parallel edges, but no self-loops. The edge weights are travel times and are strictly positive.

Solution:

DfS and BFS have been implemented. **DFS** use the stack and **BFS** use queue and rest of the logic is same.

The logic in case of DFS is that we are adding the adjacent vertices popping out the last as in stack and then doing the same . But in case of **BFS** we are going level by level i.e. i.e. we put the adjacent vertices take out the vertices put at the end of the queue and like this.

During traversal I have commented the print but counted the nodes and they are coming the same in both i.e. 264346.

6. Implement the shortest path using Dijkstra's Algorithm for the graph in HW5 Q 4(b). Then run your implementation of Dijkstra's on HW5 4(a). What happens? Explain.

Solution:

Output for Dataset on 4(A)

F	T	Distance
5	1	0.9300000000000002
0	2	0.26
7	3	0.9900000000000001
6	4	0.26000000000000023
4	5	0.6100000000000002
3	6	1.5100000000000002
2	7	0.6000000000000001

***F is from and T is to**

So here T are the points/vertices and corresponding to them are their final edges and then it is the whole distance from 0 to that point for example for 1 distance is total 0.93 but the path is 1<- 5<-4<-6<-3<-7<-2<-0. Like this all needs to be backtracked for their path from source i.e. here 0

Output for Dataset on 4(B):

Program goes into infinite loop

Explanation :

General Shortest Path algorithm is :

Move to vertices :

Relax all edges

This will continue till the distances to the particular points become stable.

I have put my Bellman Ford solution at the back but if you look at it that the above result is same as that of bellman ford after all rounds of relaxation .

So the question is that there is no difference between Dijkstra and Bellman ford . Bellman ford is a general dynamic programming algorithm which check all cases as explained earlier above.

But the Dijkstra is that it goes to one vertex , relax its edges and then select the minimum weighted edge which is the confirmed shortest path to that vertex , and then it move forward by looking all the edges originating from the vertex(V1 whose shortest path has been confirmed) and then select the minimum weighted edge and then select the vertex it has been destined to(v2) and then we have found the minimum path to v2 also. Note that other edge destined to v1 can not come and replace the previous selected edge to v1 because of the positive weights every other path will be weighted more to v1 hence we are satisfied that nothing will change the entry in the priority queue on the index v1 once it is deleted as nothing smaller weighted edge destined to v1 will come and so as for v2 and then same for v3 and so on. So our Priority queue will become empty in V-1 iterations with delete minimum in each iteration in priority queue but Important Point to be noted we have put general condition in the loop i.e. till the priority queue become empty hoping that both conditions for number of iterations equal to V-1 will occur. But if our Program loop for V-1 times

5	1	1.05
0	2	0.26
7	3	0.9900000000000001
0	4	0.38
4	5	0.73
3	6	1.5100000000000002
2	7	0.6000000000000001

Then above result will come i.e. equivalent(not equal) to only one round of Bell Man ford. But because we have loop terminating condition as emptiness of priority queue(same is also being mentioned in the book of Robert Sedgewick) , it becomes Bellman ford because it will keep changing with the further reduction of distance for particular vertex and till the priority become empty and we have come up with minimum edges to all vertices and it may have taken many iterations and hence the result we got in our program is Totally Right for even the Dataset(4a with no negative cycle) with negative weights.

But for(4b with negative cycle) Priority Queue never becomes empty as always the edges are being keeping inserted as the distance to certain vertex like here 4 keep decreasing and decreasing. So Infinite Loop.

That is why it is said – Negative Cycle -> No SPT

SPT-> No negative Cycle

Note I have studied from Class Notes and Robert Sedgewick site/book so some similarity may appear.

Result set for question 2 :

Kruskals MST Output	Prims Eager- MST	Prims Lazy- MST
0 225 0.02383	0 225 0.02383	0 225 0.02383
1 107 0.07484	1 107 0.07484	1 107 0.07484
1 72 0.06506	1 72 0.06506	1 72 0.06506
10 123 0.00886	10 123 0.00886	10 123 0.00886
10 175 0.07429	10 175 0.07429	10 175 0.07429
101 139 0.03983	101 139 0.03983	101 139 0.03983
101 196 0.03115	101 196 0.03115	101 196 0.03115
102 226 0.07775	102 226 0.07775	102 226 0.07775
103 174 0.03708	103 174 0.03708	103 174 0.03708
103 192 0.04287	103 192 0.04287	103 192 0.04287
103 243 0.05731	103 243 0.05731	103 243 0.05731
104 201 0.02597	104 201 0.02597	104 201 0.02597
104 248 0.0664	104 248 0.0664	104 248 0.0664
105 106 0.01034	105 106 0.01034	105 106 0.01034
106 193 0.02119	106 193 0.02119	106 193 0.02119
108 196 0.03132	108 196 0.03132	108 196 0.03132
109 137 0.05083	109 137 0.05083	109 137 0.05083
109 215 0.06179	109 215 0.06179	109 215 0.06179
11 152 0.0414	11 152 0.0414	11 152 0.0414
11 246 0.06678	11 246 0.06678	11 246 0.06678
11 82 0.03687	11 82 0.03687	11 82 0.03687
110 214 0.0292	110 214 0.0292	110 214 0.0292
112 159 0.04347	112 159 0.04347	112 159 0.04347
113 198 0.05009	113 198 0.05009	113 198 0.05009
113 223 0.04665	113 223 0.04665	113 223 0.04665
114 222 0.0206	114 222 0.0206	114 222 0.0206
116 164 0.0406	116 164 0.0406	116 164 0.0406
118 124 0.063	118 124 0.063	118 124 0.063
118 197 0.06902	118 197 0.06902	118 197 0.06902
119 134 0.04953	119 134 0.04953	119 134 0.04953
119 145 0.05361	119 145 0.05361	119 145 0.05361
12 198 0.05807	12 198 0.05807	12 198 0.05807
12 35 0.06079	12 35 0.06079	12 35 0.06079
121 170 0.03464	121 170 0.03464	121 170 0.03464
121 182 0.03939	121 182 0.03939	121 182 0.03939
122 139 0.01217	122 139 0.01217	122 139 0.01217
122 205 0.00647	122 205 0.00647	122 205 0.00647
124 155 0.01487	124 155 0.01487	124 155 0.01487
124 172 0.04713	124 172 0.04713	124 172 0.04713
128 239 0.01726	128 239 0.01726	128 239 0.01726
129 166 0.03392	129 166 0.03392	129 166 0.03392
13 100 0.0256	13 100 0.0256	13 100 0.0256
13 133 0.06257	13 133 0.06257	13 133 0.06257
130 194 0.05825	130 194 0.05825	130 194 0.05825
131 143 0.05854	131 143 0.05854	131 143 0.05854

131	193	0.01012	131	193	0.01012	131	193	0.01012
132	235	0.05508	132	235	0.05508	132	235	0.05508
133	166	0.06778	133	166	0.06778	133	166	0.06778
134	227	0.03986	134	227	0.03986	134	227	0.03986
135	141	0.06693	135	141	0.06693	135	141	0.06693
136	159	0.0231	136	159	0.0231	136	159	0.0231
136	234	0.03185	136	234	0.03185	136	234	0.03185
138	240	0.02076	138	240	0.02076	138	240	0.02076
14	133	0.06649	14	133	0.06649	14	133	0.06649
140	147	0.04712	140	147	0.04712	140	147	0.04712
142	213	0.03433	142	213	0.03433	142	213	0.03433
144	248	0.01527	144	248	0.01527	144	248	0.01527
146	227	0.0456	146	227	0.0456	146	227	0.0456
148	157	0.01712	148	157	0.01712	148	157	0.01712
148	184	0.03398	148	184	0.03398	148	184	0.03398
148	230	0.04043	148	230	0.04043	148	230	0.04043
15	211	0.04	15	211	0.04	15	211	0.04
150	189	0.06046	150	189	0.06046	150	189	0.06046
150	220	0.04158	150	220	0.04158	150	220	0.04158
151	208	0.00391	151	208	0.00391	151	208	0.00391
153	241	0.04247	153	241	0.04247	153	241	0.04247
154	238	0.02365	154	238	0.02365	154	238	0.02365
154	245	0.04011	154	245	0.04011	154	245	0.04011
155	180	0.03246	155	180	0.03246	155	180	0.03246
155	213	0.01559	155	213	0.01559	155	213	0.01559
156	205	0.05088	156	205	0.05088	156	205	0.05088
156	214	0.03435	156	214	0.03435	156	214	0.03435
156	219	0.00745	156	219	0.00745	156	219	0.00745
157	181	0.05473	157	181	0.05473	157	181	0.05473
158	223	0.05291	158	223	0.05291	158	223	0.05291
158	249	0.05734	158	249	0.05734	158	249	0.05734
16	117	0.05134	16	117	0.05134	16	117	0.05134
16	147	0.07171	16	147	0.07171	16	147	0.07171
160	191	0.0541	160	191	0.0541	160	191	0.0541
161	186	0.06124	161	186	0.06124	161	186	0.06124
163	211	0.01708	163	211	0.01708	163	211	0.01708
163	222	0.05489	163	222	0.05489	163	222	0.05489
165	180	0.01756	165	180	0.01756	165	180	0.01756
167	224	0.07521	167	224	0.07521	167	224	0.07521
168	187	0.0408	168	187	0.0408	168	187	0.0408
168	231	0.00268	168	231	0.00268	168	231	0.00268
169	186	0.05624	169	186	0.05624	169	186	0.05624
17	170	0.07756	17	170	0.07756	17	170	0.07756
17	229	0.06676	17	229	0.06676	17	229	0.06676
17	81	0.05763	17	81	0.05763	17	81	0.05763
170	223	0.03653	170	223	0.03653	170	223	0.03653
171	213	0.06758	171	213	0.06758	171	213	0.06758
171	235	0.03106	171	235	0.03106	171	235	0.03106
171	238	0.05664	171	238	0.05664	171	238	0.05664

175	246	0.03569	175	246	0.03569	175	246	0.03569
176	191	0.02089	176	191	0.02089	176	191	0.02089
176	202	0.04299	176	202	0.04299	176	202	0.04299
177	186	0.03537	177	186	0.03537	177	186	0.03537
177	189	0.03512	177	189	0.03512	177	189	0.03512
178	236	0.02867	178	236	0.02867	178	236	0.02867
18	86	0.02813	18	86	0.02813	18	86	0.02813
183	215	0.03017	183	215	0.03017	183	215	0.03017
185	248	0.02896	185	248	0.02896	185	248	0.02896
19	100	0.06397	19	100	0.06397	19	100	0.06397
19	174	0.01925	19	174	0.01925	19	174	0.01925
19	70	0.06872	19	70	0.06872	19	70	0.06872
190	247	0.0668	190	247	0.0668	190	247	0.0668
2	51	0.05083	2	51	0.05083	2	51	0.05083
2	86	0.0598	2	86	0.0598	2	86	0.0598
20	116	0.02095	20	116	0.02095	20	116	0.02095
200	203	0.05651	200	203	0.05651	200	203	0.05651
201	232	0.03557	201	232	0.03557	201	232	0.03557
202	204	0.04207	202	204	0.04207	202	204	0.04207
203	249	0.04148	203	249	0.04148	203	249	0.04148
207	210	0.03158	207	210	0.03158	207	210	0.03158
207	212	0.05667	207	212	0.05667	207	212	0.05667
207	221	0.02902	207	221	0.02902	207	221	0.02902
209	211	0.01269	209	211	0.01269	209	211	0.01269
21	233	0.04409	21	233	0.04409	21	233	0.04409
21	27	0.01873	21	27	0.01873	21	27	0.01873
21	71	0.03782	21	71	0.03782	21	71	0.03782
210	219	0.03728	210	219	0.03728	210	219	0.03728
22	34	0.03658	22	34	0.03658	22	34	0.03658
22	56	0.01508	22	56	0.01508	22	56	0.01508
228	241	0.01473	228	241	0.01473	228	241	0.01473
23	33	0.06032	23	33	0.06032	23	33	0.06032
23	58	0.07072	23	58	0.07072	23	58	0.07072
24	149	0.02915	24	149	0.02915	24	149	0.02915
24	209	0.02802	24	209	0.02802	24	209	0.02802
25	111	0.05309	25	111	0.05309	25	111	0.05309
25	60	0.03405	25	60	0.03405	25	60	0.03405
27	240	0.06581	27	240	0.06581	27	240	0.06581
28	198	0.00775	28	198	0.00775	28	198	0.00775
28	242	0.02503	28	242	0.02503	28	242	0.02503
29	218	0.02524	29	218	0.02524	29	218	0.02524
29	224	0.03477	29	224	0.03477	29	224	0.03477
3	153	0.04799	3	153	0.04799	3	153	0.04799
3	67	0.09725	3	67	0.09725	3	67	0.09725
30	179	0.0657	30	179	0.0657	30	179	0.0657
30	212	0.03314	30	212	0.03314	30	212	0.03314
30	244	0.01475	30	244	0.01475	30	244	0.01475
31	241	0.07871	31	241	0.07871	31	241	0.07871
32	187	0.04594	32	187	0.04594	32	187	0.04594

34	73	0.01812	34	73	0.01812	34	73	0.01812
35	94	0.06254	35	94	0.06254	35	94	0.06254
36	41	0.05329	36	41	0.05329	36	41	0.05329
36	88	0.01841	36	88	0.01841	36	88	0.01841
37	115	0.01398	37	115	0.01398	37	115	0.01398
37	153	0.03846	37	153	0.03846	37	153	0.03846
37	76	0.02674	37	76	0.02674	37	76	0.02674
38	126	0.00845	38	126	0.00845	38	126	0.00845
38	215	0.0529	38	215	0.0529	38	215	0.0529
39	149	0.05382	39	149	0.05382	39	149	0.05382
39	66	0.0559	39	66	0.0559	39	66	0.0559
4	55	0.06425	4	55	0.06425	4	55	0.06425
4	78	0.02559	4	78	0.02559	4	78	0.02559
40	164	0.03395	40	164	0.03395	40	164	0.03395
40	190	0.03649	40	190	0.03649	40	190	0.03649
40	220	0.0315	40	220	0.0315	40	220	0.0315
41	182	0.03878	41	182	0.03878	41	182	0.03878
42	108	0.06664	42	108	0.06664	42	108	0.06664
42	135	0.05008	42	135	0.05008	42	135	0.05008
42	181	0.06506	42	181	0.06506	42	181	0.06506
43	207	0.02708	43	207	0.02708	43	207	0.02708
44	204	0.01774	44	204	0.01774	44	204	0.01774
44	49	0.02107	44	49	0.02107	44	49	0.02107
45	83	0.02906	45	83	0.02906	45	83	0.02906
46	169	0.06854	46	169	0.06854	46	169	0.06854
47	218	0.02914	47	218	0.02914	47	218	0.02914
48	216	0.07531	48	216	0.07531	48	216	0.07531
48	232	0.02109	48	232	0.02109	48	232	0.02109
49	225	0.03314	49	225	0.03314	49	225	0.03314
49	97	0.03121	49	97	0.03121	49	97	0.03121
5	102	0.03834	5	102	0.03834	5	102	0.03834
5	26	0.03351	5	26	0.03351	5	26	0.03351
50	185	0.05942	50	185	0.05942	50	185	0.05942
50	59	0.04226	50	59	0.04226	50	59	0.04226
52	187	0.03447	52	187	0.03447	52	187	0.03447
52	208	0.03135	52	208	0.03135	52	208	0.03135
52	226	0.02384	52	226	0.02384	52	226	0.02384
53	120	0.02782	53	120	0.02782	53	120	0.02782
53	145	0.06902	53	145	0.06902	53	145	0.06902
54	140	0.02922	54	140	0.02922	54	140	0.02922
55	112	0.0459	55	112	0.0459	55	112	0.0459
56	120	0.05027	56	120	0.05027	56	120	0.05027
57	184	0.03368	57	184	0.03368	57	184	0.03368
57	197	0.02583	57	197	0.02583	57	197	0.02583
58	114	0.01947	58	114	0.01947	58	114	0.01947
58	68	0.04795	58	68	0.04795	58	68	0.04795
59	97	0.06442	59	97	0.06442	59	97	0.06442
6	129	0.05363	6	129	0.05363	6	129	0.05363
6	16	0.04529	6	16	0.04529	6	16	0.04529

6	236	0.05556	6	236	0.05556	6	236	0.05556
60	199	0.04868	60	199	0.04868	60	199	0.04868
61	111	0.06739	61	111	0.06739	61	111	0.06739
61	87	0.05163	61	87	0.05163	61	87	0.05163
62	233	0.03192	62	233	0.03192	62	233	0.03192
63	199	0.01821	63	199	0.01821	63	199	0.01821
63	237	0.04962	63	237	0.04962	63	237	0.04962
64	137	0.03977	64	137	0.03977	64	137	0.03977
65	184	0.0479	65	184	0.0479	65	184	0.0479
65	188	0.03552	65	188	0.03552	65	188	0.03552
66	206	0.05154	66	206	0.05154	66	206	0.05154
67	217	0.04501	67	217	0.04501	67	217	0.04501
68	176	0.04396	68	176	0.04396	68	176	0.04396
69	107	0.0564	69	107	0.0564	69	107	0.0564
69	173	0.05282	69	173	0.05282	69	173	0.05282
7	125	0.02442	7	125	0.02442	7	125	0.02442
7	157	0.00516	7	157	0.00516	7	157	0.00516
70	179	0.06528	70	179	0.06528	70	179	0.06528
70	79	0.01576	70	79	0.01576	70	79	0.01576
71	188	0.03894	71	188	0.03894	71	188	0.03894
72	189	0.03706	72	189	0.03706	72	189	0.03706
72	203	0.0347	72	203	0.0347	72	203	0.0347
74	215	0.04643	74	215	0.04643	74	215	0.04643
75	116	0.02706	75	116	0.02706	75	116	0.02706
75	89	0.03394	75	89	0.03394	75	89	0.03394
76	95	0.0737	76	95	0.0737	76	95	0.0737
77	102	0.02737	77	102	0.02737	77	102	0.02737
77	138	0.07171	77	138	0.07171	77	138	0.07171
78	239	0.02065	78	239	0.02065	78	239	0.02065
8	143	0.07437	8	143	0.07437	8	143	0.07437
8	152	0.00702	8	152	0.00702	8	152	0.00702
8	244	0.02711	8	244	0.02711	8	244	0.02711
80	225	0.05577	80	225	0.05577	80	225	0.05577
81	134	0.04508	81	134	0.04508	81	134	0.04508
82	85	0.04344	82	85	0.04344	82	85	0.04344
83	217	0.03695	83	217	0.03695	83	217	0.03695
83	232	0.06812	83	232	0.06812	83	232	0.06812
84	174	0.0277	84	174	0.0277	84	174	0.0277
86	141	0.0564	86	141	0.0564	86	141	0.0564
87	130	0.03684	87	130	0.03684	87	130	0.03684
87	234	0.08302	87	234	0.08302	87	234	0.08302
89	127	0.10682	89	127	0.10682	89	127	0.10682
89	194	0.0511	89	194	0.0511	89	194	0.0511
9	195	0.04585	9	195	0.04585	9	195	0.04585
9	23	0.03526	9	23	0.03526	9	23	0.03526
90	173	0.06973	90	173	0.06973	90	173	0.06973
91	137	0.01061	91	137	0.01061	91	137	0.01061
91	146	0.02723	91	146	0.02723	91	146	0.02723
91	218	0.04088	91	218	0.04088	91	218	0.04088

92	235	0.0604	92	235	0.0604	92	235	0.0604
93	160	0.0436	93	160	0.0436	93	160	0.0436
93	231	0.03594	93	231	0.03594	93	231	0.03594
94	141	0.0807	94	141	0.0807	94	141	0.0807
96	199	0.01569	96	199	0.01569	96	199	0.01569
98	236	0.04433	98	236	0.04433	98	236	0.04433
99	147	0.03171	99	147	0.03171	99	147	0.03171
99	162	0.06455	99	162	0.06455	99	162	0.06455

Ques3 Homework(7)

Objective → For the edge weighted Directed Acyclic Graph → we have to find the shortest Path and the Longest Path
We have Learnt → Bellman ford, Dijakstra, & Topological sort based Method

As we Know that basic method of finding the shortest ~~path~~ Path is →
We have to go through vertices & relax all edges & we need to continue this until we get stable distances.

General-way → was generalised by → Dijakstra → In case of Positive weight edges → as the → He said that (in ascendy order)
Just take the vertices with minimum distance & then just relax all the edges originated from that vertex and when all vertices are completed we will get the shortest path of each vertex w.r.t to starting source.

So, this variety appears from our general interpretation, because → Once we relax all the edges $v \to w$ ie originating from 'v' we know that D[v] is fixed it will not change as it is the minimum (when we are relaxing step by step) — and D[v] is fixed there only because any other path to D[v] will be more because of the positive weights for any $v \to w$
So, as it was relaxed once →
ie v originating edges were relaxed

once, so 'v' need not be afraid as the Relaxation inequality ie $(D[v] + \text{edge weight } v \to w) \geq D(w)$

As in an algorithm $D(w)$ will be only decreasing and $D[v]$ is constant & edge weight is constant

So, once when $v \to w$ relaxed and the above inequality hold it will always follow even after further relaxation. So, the good thing is that when we move past $v \to$ we need not look back & move to other vertices & when all vertices over, the task is over.

From that Perspective, comes the Topological Sort based Algorithm for DAGs where even the edge weights can be -ve

Here,
When we consider the Point in topological order then we know that →

$v_1 \to w_1$, $v_1 \to w_2$, w_3 } → when from $v_1 \to$ relaxations are being done as there is no back path so $v_1 \to$ need not worry that its Distance will change. So, it is quite similar to above explanation in case of Dijakstra.

Here, we have found the order where we will not have to look back the vertex again

if we follow the Topological Order (and that is possible only in case of Acyclical Graph)

So → Then → after relaxation of edges from v → (v → w) edge consider it will relax only once → [so → as explained]

as

D[w] as is always monotonously decreasing with always satisfy

$$D[w] \leq D[v] + \text{edgeweight}$$

as after first relaxation from edge from v → D[w] has to decrease & D[v] as will not be changed as it is an directed acyclic graph & we are considering the Topological sort Order :-

Now we look at the Question in Hand →

8
13
5 4 0.35
4 7 0.37

Now in the adjacency list way

0 → [2]
1 → [3]
2 → [ ]
3 → [6, 7]
4 → [0, 7]
5 → [1, 4, 7]
6 → [0, 2, 4]
7 → 2

① Topological sort → using Post order traversal & then reversing called Reverse Post Order → starting from 5 → [5 1 3 6 4 0 7 2]

So, now we will do the Relaxation in the Graph in this order only once.

Vertex	Dist	Edge to	
5	—	—	— First the edges relaxation
1	0.32	5	— second the edges relaxation
3	0.61	1	— Third " " "
6	1.13	3	— Fourth " " "
4	0.35	5	
0	0.73	4	→ Fifth " " "
7	0.28	5	→ sixth " " "
2	~~0.99~~ 0.62	~~0~~ 7	

Then, the ~~Relax~~ For the Longest Path we will make the weights -ve and then proceed with the above procedure →

8
7 3
5 4 -0.35
4 7 -0.37
5 7 -0.28
5 1 -0.32
4 0 -0.38
0 2 -0.26
3 7 -0.39
1 3 -0.29
7 2 -0.34
6 2 -0.40
3 6 -0.52
6 0 -0.58
6 4 -0.93

Graph with -ve weights

Now - we consider the Topological Order :-

Vertices	Dist	EdgeTo
5	-	-
1	-0.32	5
3	-0.61	1
6	-1.13	3
4	~~-0.35~~ ~~-2.06~~	~~5~~ 6
0	~~-1.71~~ -2.44	~~6~~ 4
7	~~-0.28~~ ~~-2.43~~	~~7~~ 4
2	~~-1.53~~ ~~-2.70~~ -2.77	~~6~~ ~~0~~ 7

Vertices	Dist	EdgeTo
5	-	-
1	-0.32	5
3	-0.61	1
6	-1.13	3
4	-2.06	6
0	-2.44	4
7	-2.43	4
2	-2.77	7

-ve is only using the shortest path algorithm to find the longest path

Ans 4 → Homework 4 Part (a) Let's compute the progress of Bellman Ford Algorithm over the graph shown on the slide :→

Bellman Ford Algorithm is a dynamic programming algorithm that consider all the possibles cases and chooses out of that. It is different from Dijakstra in the sense that in Dijkstra all weights are considered +ve. So, once we have a path to a particular point, which is the minimum we are sure that in further relaxations, the path weight to that point will only increase so, we choose that path to that point ie the shortest path that leads to that point. But when we have the -ve weights, we are not sure that even the minimum path to a particular point is actually the shortest because when we have -ve paths It is possible that some shortest path to that point comes up in further relaxations. Hence there comes the Bellmon Ford, dynamic programming algorithm for finding the shortest path

8	
15	
45	0.35
54	0.35
47	0.37
57	0.28
75	0.28
51	0.32
04	0.38
02	0.26
73	0.39
13	0.29
27	0.24
62	1.20
36	0.52
60	1.40
64	1.25

Now Let us see how the Bellmon Ford Programming algorithm works :→ (on the Graph given on the question)

Bellman Ford Algorithm

Repeat V Times :—

— Relax each edge

So, that is what we will do in our algorithm computation of the Graph.

Data Structure Used

V	distTo[ ]	edgeTo

List of Edges with weights (as given on the prev pg)

Round 1 - (as there are 8 vertices we have to take 8 rounds)

V	distTo[ ]	edgeTo
0	0.0	-
1	1.05	5→1
2	0.26	0→2
3	1.14	7→3
4	0.38	0→4
5	0.73	4→5
6	1.66	3→6
7	~~0.75~~ 0.6	~~4→7~~ 2→7

(1)	0	4	0.38 ✓
(2)	0	2	0.26 ✓
(3)	4	5	0.35 ✓
(4)	5	4	0.35 ✓
(5)	4	7	0.37 ✓
(6)	5	7	0.28 ✓
(7)	7	5	0.28 ✓
(8)	5	1	0.32 ✓
(9)	7	3	0.39 ✓
(10)	1	3	0.29 ✓
(11)	2	7	0.34 ✓
(12)	6	2	-1.20 ✓
(13)	3	6	0.52 ✓
(14)	6	0	-1.40 ✓
(15)	6	4	-1.25 ✓

So, in this round we have relaxed all the 15 edges as shown & that will be continued for other edges and computed the distances for particular vertices and the edges taking to that →

& Now we continue for other Rounds.

Round 2

v	distTo[ ]	edgeTo
0	0.0	—
1	1.05	5→1
2	0.26	0→2
3	~~1.44~~ 0.99	7→3
4	~~0.38~~ 0.26	~~0→4~~ 6→4
5	0.73	4→5
6	~~1.66~~ 1.51	3→6
7	0.6	2→7

Round 3

v	distTo[ ]	edgeTo
0	0.0	—
1	~~1.05~~ 0.93	5→1
2	0.26	0→2
3	0.99	7→3
4	0.26	6→4
5	~~0.73~~ 0.61	4→5
6	1.51	3→6
7	0.6	2→7

Round 4

v	distTo[ ]	edgeTo
0	0.0	—
1	0.93	5→1
2	0.26	0→2
3	0.99	7→3
4	0.26	6→4
5	0.61	4→5
6	1.51	3→6
7	0.6	2→7

→ So, after this round we have no more changes so, this is the solution.

① ✓ ✓
② ✓ ✓
③ ✓ ✓
④ ✓ ✓
⑤ ✓ ✓
⑥ ✓ ✓
⑦ ✓ ✓
⑧ ✓ ✓
⑨ ✓ ✓
⑩ ✓ ✓
⑪ ✓ ✓
⑫ ✓ ✓
⑬ ✓ ✓
⑭ ✓ ✓
⑮ ✓ ✓

Part2 → Bellman ford on a graph set with -ve cycle.

8
15
4 5 0.35
5 4 -0.66
4 7 0.37
5 7 0.28
7 5 0.28
5 1 0.32
0 4 0.38
0 2 0.26
7 3 0.39
1 3 0.29
2 7 0.34
6 2 0.40
3 6 0.52
6 0 0.58
6 4 0.93

Edge list for going through

① 0 4 0.38
② 0 2 0.26
③ 1 3 0.29
④ 2 7 0.34
⑤ 3 6 0.52
⑥ 4 5 0.35
⑦ 4 7 0.37
⑧ 5 1 0.32
⑨ 5 4 -0.66
⑩ 5 7 0.28
⑪ 6 0 0.58
⑫ 6 2 0.40
⑬ 6 4 0.93
⑭ 7 3 0.39
⑮ 7 5 0.28

You can take any order but I am taking in the order of vertices.

3→6 4→5
~~5→5~~ 4→7
5→1
5→4
~~5→3~~
7→3

Round1

V	Dist[]	Edge to
0	0.0	
1	1.05	5→1
2	0.26	0→2
3	~~∞~~ 0.99	7→3
4	~~0.38~~ 0.07	0→4 5→4
5	0.73	4→5
6	∞	
7	0.60	2→7

Round2

V	Dist[]	Edge to
0	0.0	
1	~~1.05~~ 0.74	5→1
2	0.26	0→2
3	~~0.99~~ 0.83	7→3
4	~~0.07~~ -0.24	5→4
5	~~0.73~~ 0.42	~~4→5~~
6	~~∞~~ 1.51	3→6
7	~~0.60~~ 0.44	~~2→7~~ 4→7

Round3

V	Dist[]	Edge to
0	0.0	
1	~~0.74~~ 0.43	5→1
2	0.26	0→2
3	~~0.83~~ 0.52	7→3
4	~~-0.24~~ -0.55	5→4
5	~~0.42~~ 0.11	4→5
6	~~1.51~~ 1.35	3→6
7	~~0.44~~ 0.13	4→7

What is being observed here is the Pattern

3 → 6 changes
4 → 5 changes
4 → 7 changes
5 → 1
5 → 4 — again decrease
7 → 3 changes

Pattern again as other
3 Changed → 6 changes
4 changed → 5 changes
4 Changed → 7 changes
5 Changed → 1 changes

4 5 again change
4 is decrease
4 due to -ve cycle
& then 4 again changes 7 changes

cycle Repeates

So, with -ve cycle the 4 is being affected
ie the distance is being decreased in every round & hence other changes & then again 4 is decreased & hence other changes & then further this cycle repeats & hence even the bellman ford fails as there is

Proposition → (Negative cycle → Then NO SPT)

(SPT → Negative cycle)
(negation of HT proposition)

By me This Pattern is clearly visible
But if we go by the Bellmann Ford Algorithm we check V-1 times ie 7 times & even if the we have changes then we have -ve cycle.

Round 4

0	0.0	—
1	0.12 ~~0.43~~	5→1
2	0.26	0→2
3	0.~~71~~ ~~0.52~~	7→3
4	-0.86 ~~0.55~~	5→4
5	-0.2 ~~0.11~~	4→5
6	~~1.35~~ 1.04	3→6
7	0.13 →	4→7
	~~-0.18~~	

So, same cycle has repeat it will continue to repeat