University of Nevada, Las Vegas Computer Science 477/677 Spring 2021

Answers to Assignment 6 Due Tuesday April 6, 2021

1. Consider the single source shortest path problem on a weighted digraph G. (digraph = directed graph) Fill in each blank with **one** word, or formula. Assume G has n vertices and m arcs (directed edges.) Assume that every answer (shortest path from the source to a vertex) has no more than p edges.

   (a) The simple dynamic programming algorithm requires that G be **acyclic**. In that case, the time complexity of the algorithm is $O(m)$.

   (b) The Bellman-Ford algorithm requires that G have no **negative cycle**. In that case, the time complexity of the algorithm is $O(mp)$.

   (c) Dijkstra's algorithm requires that G have no **negative arc** . In that case, the time complexity of the algorithm is $O(m \log n)$.

   (d) In order to work the simple dynamic programming algorithm for the single source shortest path problem, we must visit the vertices of G in **topological** order.

2.
   ```
   int f(int n)
   {
    if(n<7) return 1;
    else return f(n/2)+f(n/2+1)+f(n/2+2)+f(n/2+3);
   }
   ```

   The function f(n) can be computed by recursion, as given in the C++ code above. However, we could also compute f(n) using dynamic programming, or memoization.

   (a) What is the time complexity of the recursive computation of f(n)?

   Let $T(n)$ be the time complexity of the recursive computation of f(n). Then $T(n) = 4T(n/2) + 1$, hence $T(n) = \Theta(n^2)$.

   (b) What is the time complexity of the dynamic programming computation of f(n)?

   In this case the time satisfies the recurrence $T(n) = T(n-1) + 1$, hence $T(n) = \Theta(n)$

   (c) What is the time complexity of the computation of f(n) using memoization? Hint: Try a large value of n, such as n = 1024.

   In this case the time satisfies the recurrence $T(n) = T(n/2) + 1$, hence $T(n) = \Theta(\log n)$

3. Let G be a weighted directed graph with n vertices and m arcs (directed edges).

   (a) What is the time complexity of the Floyd-Warshall algorithm for the all pairs shortest path problem on G?

   $\Theta(n^3)$, because you need the backpointers as well as the minimum path distances.

   (b) What is the time complexity of Johnson's algorithm for the all pairs shortest path problem on G?

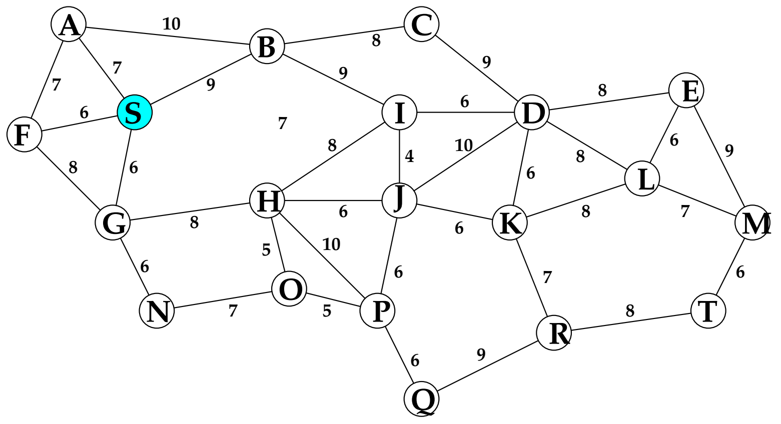
   $O(nm \log n)$

(c) (This problem was not on the homework, but you need to know it.) What is the space requirement for the solution to the all pairs shortest path problem on G?

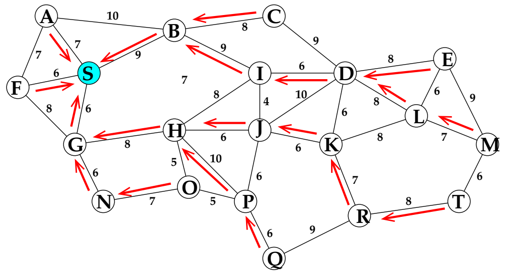
$\Theta(n^2)$

4. Walk through Dijkstra's algorithm to solve the single source shortest path problem for the weighted graph shown below, where the source vertex is S. Each undirected edge is two directed edges, one in each direction.

Show the set of vertices in the heap, as well as the backpointers, at each step.



Consider the two figures below. The left figure shows the original weighted graph, with the backpointers in red. The second figure shows the main array, with the two corrections indicated. The first row of the main array lists the vertices. The second row shows the minimum weight of any path from the source S to each vertex, and the third row shows the backpointers. That array is all you need to get full credit for this problem. Be sure to indicate the values that were changed: in this case, there are two columns of with changed values.



S	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	T
0	7	9	17	~~26~~ 24	34	6	6	14	18	20	26	34	~~41~~ 39	12	19	24	32	33	41
	S	S	B	~~C~~ I	D	S	S	G	B	H	J	K	~~E~~ L	G	H	H	F	K	R

I will now walk through the algorithm in detail, showing every step. We show the minheap at each step. The second row shows the current minimum distance known from S to the vertex, while the third row shows the current backpointer. At each step, the column whose distance value is minimum is deleted from the minheap and its successors are inserted. Distances and backpointers may be updated. As each vertex is deleted from the minheap, its column in the main array is updated. To save space, we do not show the main array after each step.

Initially, only S is in the minqueue.

S
0
*

We delete S from the minqueue and update the first column of the main array.

S	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	T
0																			
*																			

then insert the outneighbors of S. The minqueue is now:

F	G	A	B
6	6	7	9
S	S	S	S

We delete F and update the F column of the main array. The minqueue is now:

G	A	B
6	7	9
S	S	S

We delete G and update the G column of the main array. We insert H and N into the minqueue. The minqueue is now:

A	B	N	H
7	9	12	14
S	S	G	G

while the main array is now:

S	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	T
0						6	6												
*						S	S												

We delete A and update the A column of the main array. The minqueue is now:

B	N	H
9	12	14
S	G	G

We delete B and update the B column of the main array. We insert C and I into the queue. The minqueue is now:

N	H	C	I
12	14	17	18
G	G	B	B

We delete N and update the N column of the main array. We insert O into the queue. The minqueue is now:

H	C	I	O
14	17	18	19
G	B	B	N

The main array is now:

S	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	T
0	7	9				6	6							12					
∗	S	S				S	S							G					

We delete H and update the H column of the main array. We insert J and P into the queue. The minqueue is now:

C	I	O	J	P
17	18	19	20	24
B	B	N	H	H

We delete C and update the C column of the main array. We insert D into the queue. The minqueue is now:

I	O	J	P	D
18	19	20	24	26
B	N	H	H	C

We delete I and update the I column of the main array. Update the cost and backpointer of D. The minqueue is now:

O	J	P	D
19	20	24	24
N	H	H	I

We delete O and update the O column of the main array. The minqueue is now:

J	P	D
20	24	24
H	H	I

We delete J and update the J column of the main array. We insert K into the queue. The minqueue is now:

P	D	K
24	24	26
H	I	J

We delete P and update the P column of the main array. We insert Q into the queue. The minqueue is now:

D	K	Q
24	26	30
I	J	P

The main array is now:

S	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	T
0	7	9	17			6	6	14	18	20				12	19	24			
∗	S	S	B			S	S	G	B	H				G	N	H			

We delete D and update the D column of the main array. We insert E and L into the queue. The minqueue is now:

K	Q	E	L
26	30	32	32
J	P	D	D

We delete K and update the K column of the main array. We insert R into the queue. The minqueue is now:

Q	R	E	L
30	31	32	32
P	K	D	D

We delete Q and update the Q column of the main array. The minqueue is now:

R	E	L
31	32	32
K	D	D

We delete R and update the R column of the main array. We insert T into the queue The minqueue is now:

E	L	T
32	32	39
D	D	R

We delete E and update the E column of the main array. We insert M into the queue The minqueue is now:

L	T	M
32	39	41
D	R	E

We delete L and update the L column of the main array. The cost and backpointer of M are updated. The minqueue is now:

T	M
39	39
R	L

We delete T and update the T column of the main array. The minqueue is now:

M
39
L

We delete M and update the M column of the main array. The minqueue is now empty. The final main array is:

S	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	T
0	7	9	17	24	32	6	6	14	18	20	26	32	39	12	19	24	30	31	39
$*$	S	S	B	I	D	S	S	G	B	H	J	D	L	G	N	H	P	K	R

5. Write pseudocode for the Floyd-Warshall algorithm on a weighted directed graph G. Assume that the vertices of G are the integers from 1 to n. We let $W(i,j)$ be the weight of the edge from i to j. If there is no such edge, we let $W(i,j) = \infty$ by default. Your output consists of two 2-dimensional arrays. $V(i,j)$ is the least cost of any path from i to j, and **back**(i,j) is the next-to-the-last vertex on the least cost path from i to j.

```
for(int i = 1; i <= n; i++)
  for(int j = 1; j <= n; j++)
    {
     V(i,j) = W(i,j); // possibly infinity
     back(i,j) = i;
    }
for(int i = 1; i <= n; i++)
  V(i,i) = 0;
for(int j = 1; j <= n; j++)
  for(int i = 1; i <= n; i++)
    for(int k = 1; k <= n; k++)
     {
      temp = V(i,j)+V(j,k);
      if(temp < V(i,k)
       {
        // relaxation step
        V(i,k) = temp;
        back(i,k) = back(j,k);
       }
     }
```

6. Compute the Levenshtein edit distance between $u =$"thorough" and $v =$"thourow." Show the matrix.

We fill in the values of the matrix $L[\,,\,]$ by dynamic programming. Recall that $L[i,0] = i$, $L[0,j] = j$, and for $i > 0, j > 0$, $L[i,j] = min\{L[i-1,j]+1, L[i,j-1]+1, L[i-1,j-1]\}$ if $u_i = v_j$; and $L[i,j] = min\{L[i-1,j]+1, L[i,j-1]+1, L[i-1,j-1]+1\}$ otherwise.

		t	h	o	u	r	o	w
	0	1	2	3	4	5	6	7
t	1	0	1	2	3	4	5	6
h	2	1	0	1	2	3	4	5
o	3	2	1	0	1	2	3	4
r	4	3	2	1	1	1	2	3
o	5	4	3	2	2	2	1	2
u	6	5	4	3	3	3	2	2
g	7	6	5	4	4	4	3	3
h	8	7	6	5	5	5	4	4

The Levenstein distance is $L[8,7] = 4$.
The edit sequence is:

`thourow`
`thorow` (delete)
`thorou` (replace)
`thoroug` (insert)
`thorough` (insert)

7.