Computer Science 477/677 Fall 2019
University of Nevada, Las Vegas
Answers to Second Examination October 14, 2019

1. Fill in the blanks.

   (a) Every comparison-based sorting algorithm makes $\Omega(n \log n)$ comparisons in the worst case.

   (b) The Las Vegas algorithm for finding the median of a sequence of numbers makes $\Theta(n)$ comparisons on the average, but $\Theta(n^2)$ comparisons in the worst case.

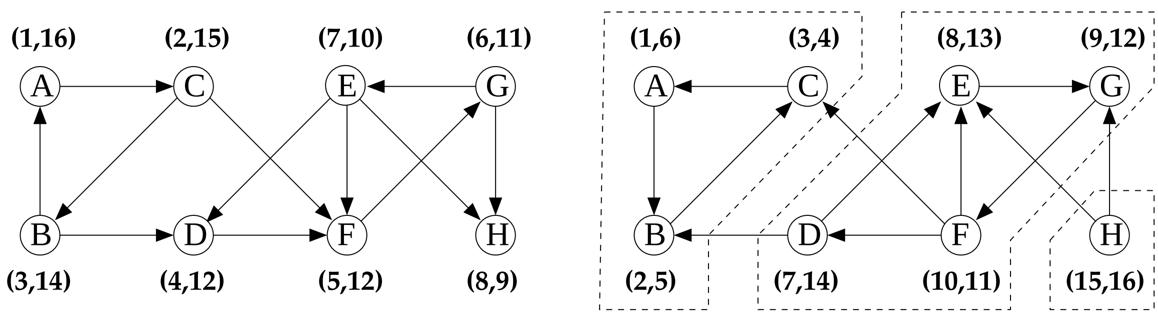
   (c) An undirected graph of n nodes cannot have more than $\binom{n}{2} = \frac{n(n-1)}{2}$ edges, while a directed graph of n nodes cannot have more than n^2 directed edges. (Exact answers are required, not asymptotic notaion.)

   (d) A directed graph has a topological order if and only if it is **acyclic**.

   (e) To save space, a sparse graph should be represented as **an array of lists of edges**.

   (f) For a digraph G with n vertices and m edges, it takes $n+m$ time to construct G^R, provided G is implemented as an array of lists.

2. Let G be the directed graph G given below. Use the DFS algorithm we have given in class to find the strong components of G.



G $\qquad\qquad$ G^R

   (a) Label the pre and post numbers of the vertices of G using depth first search.

   (b) Construct G^R by reversing the edges of G.

   (c) Compute pre and post of G^R, always starting with the vertex with the highest possible post number from the previous step.

   (d) During the DFS search of G^R, each time the pre and post number of a vertex v differ by one, v is the last vertex of the current strong component.

   (e) The strong components are $\{\mathrm{A, B, C}\}$, $\{\mathrm{D, E, G, F}\}$, and $\{\mathrm{H}\}$.

3. Values of the 2-parameter array $C[n,k]$ for $n \geq 0$ and $0 \leq k \leq n$ can be computed using dynamic program, using:

   (a) $C[n,0] = C[n,n] = 1$ for all n.

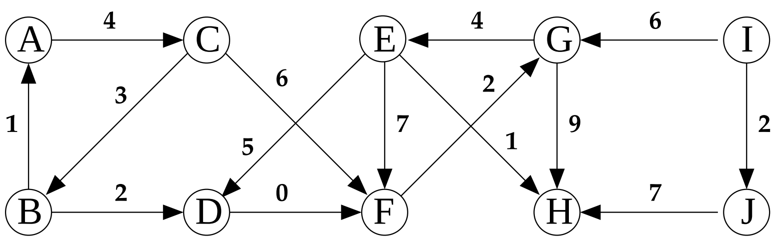
   (b) If $0 < k < n$, then $C[n,k] = C[n-1,k-1] + C[n-1,k]$.

   Compute the values of $C[n,k]$ for all $0 \leq k \leq n \leq 4$.

   Each $C[n,k]$ is a subproblem. The problems $C[n,0]$ and $C[n,n]$, for any n, are base cases. The dynamic program is executed as follows.

   - $C[0,0] = 1$. Base case.
   - $C[1,0] = 1$. Base case.
   - $C[1,1] = 1$. Base case.
   - $C[2,0] = 1$. Base case.
   - $C[2,2] = 1$. Base case.
   - $C[3,0] = 1$. Base case.
   - $C[3,3] = 1$. Base case.
   - $C[4,0] = 1$. Base case.
   - $C[4,4] = 1$. Base case.
   - $C[2,1] = C[1,0] + C[1,1] = 1 + 1 = 2$
   - $C[3,1] = C[2,0] + C[2,1] = 1 + 2 = 3$
   - $C[3,2] = C[2,1] + C[2,2] = 2 + 1 = 3$
   - $C[4,1] = C[3,0] + C[3,1] = 1 + 3 = 4$
   - $C[4,2] = C[3,1] + C[3,2] = 3 + 3 = 6$
   - $C[4,3] = C[3,2] + C[3,3] = 3 + 1 = 4$

4. Let G be the directed graph given below. Use Dijkstra's algorithm to solve the single source shortest path problem on G with start vertex A. Show your work.



I will try to use the notation used in the code on page 110 of our textbook, Thus, the backpointer is called prev. The second array is the minqueue, Q, where minimum dist node is on the left. Initially, A is the only member of the queue, and its backpointer prev[A] is undefined.

	A	B	C	D	E	F	G	H	I	J
dist	0	∞	∞	∞	∞	∞	∞	∞	∞	∞
prev	$\star$									

A
0

At each step, we execute u = deletemin(Q). Thus u = A. We insert C into Q, since it is the only outneighbor of A.

	A	B	C	D	E	F	G	H	I	J
dist	0	∞	4	∞	∞	∞	∞	∞	∞	∞
prev	$\star$		A							

C
4

Now we let u = deletemin(Q) = C, and we insert B and F into Q.

	A	B	C	D	E	F	G	H	I	J
dist	0	7	4	∞	∞	10	∞	∞	∞	∞
prev	$\star$	C	A			C				

B	F
7	10

Now we let u = deletemin(Q) = B, and we insert D into Q. Note that D is ahead of F in the priority.

	A	B	C	D	E	F	G	H	I	J
dist	0	7	4	9	∞	10	∞	∞	∞	∞
prev	$\star$	C	A	B		C				

D	F
9	10

Now we let u = deletemin(Q) = D, and we update dist(F) and prev(F).

	A	B	C	D	E	F	G	H	I	J
dist	0	7	4	9	∞	9	∞	∞	∞	∞
prev	$\star$	C	A	B		D				

F
9

Now we let u = deletemin(Q) = F, and we insert G into Q. Now we let u = deletemin(Q) = F, and we insert G into Q.

	A	B	C	D	E	F	G	H	I	J
dist	0	7	4	9	∞	9	11	∞	∞	∞
prev	$\star$	C	A	B		D	F			

G
11

Now we let u = deletemin(Q) = G, and we insert E and H into Q.

	A	B	C	D	E	F	G	H	I	J
dist	0	7	4	9	15	9	11	20	∞	∞
prev	$\star$	C	A	B	G	D	F	G		

E	H
15	20

Now we let u = deletemin(E) = G, and we update dist[H] and prev[H]

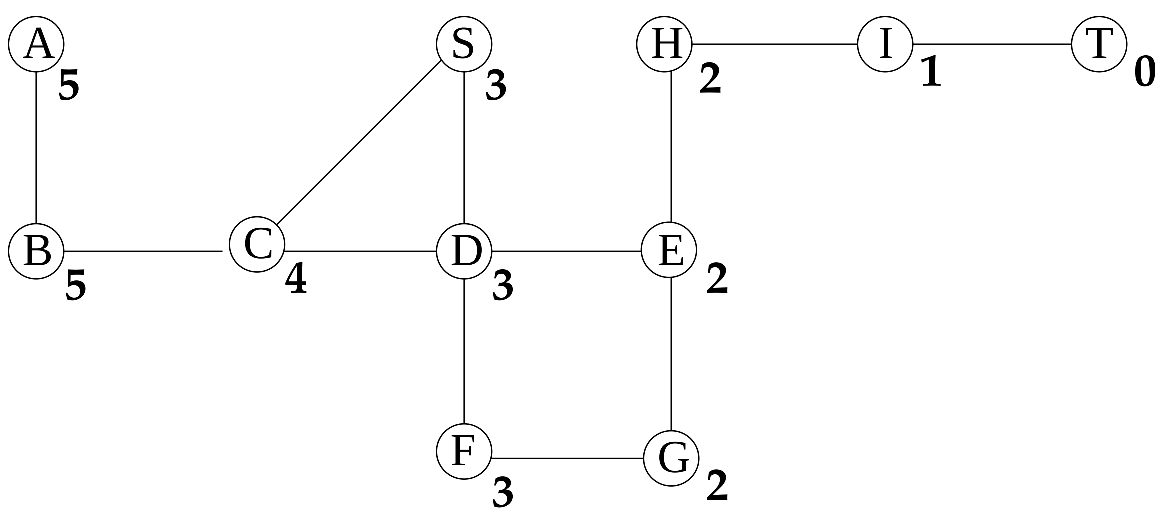
	A	B	C	D	E	F	G	H	I	J
dist	0	7	4	9	15	9	11	16	∞	∞
prev	$\star$	C	A	B	G	D	F	E		

H
16

Now we let u = deletemin(E) = H. Since H has no outneighbors, we do not insert anything into Q. Since Q is empty, we are done. I and J are unreachable from A.

	A	B	C	D	E	F	G	H	I	J
dist	0	7	4	9	15	9	11	16	∞	∞
prev	$\star$	C	A	B	G	D	F	E		

5. Use the A^* algorithm to find the shortest path from S to T. All edges have weight 1. The value of the heuristic at each node is indicated in the figure.



We write Q after each step. The node with minimum g = h+f is the first line.

	h	f	g	prev
S	3	0	3	$\star$

Delete S, and insert C and E into Q.

	h	f	g	prev
D	3	1	4	S
C	4	1	5	S

Delete D and insert C and F into Q.

	h	f	g	prev
E	2	2	4	D
C	4	1	5	S
F	3	2	5	D

Delete E and insert G and H into Q.

	h	f	g	prev
C	4	1	5	S
F	3	2	5	D
G	2	3	5	E
H	2	3	5	E

Delete C and insert B into Q.

	h	f	g	prev
F	3	2	5	D
G	2	3	5	E
H	2	3	5	E
B	5	2	7	C

We do two steps at once. Delete F, then G, from Q.

	h	f	g	prev
H	2	3	5	E
B	5	2	7	C

Delete H and insert I into Q.

	h	f	g	prev
I	1	4	5	H
B	5	2	7	C

Delete I and insert T into Q. Since we have reached T, we are done.

	h	f	g	prev
T	0	5	5	I
B	5	2	7	C

Following the back pointers, we obtain the shortest path: S,D,E,H,I,T.

6. Write the loop invariant of the main partition loop of quicksort, as given in the code below.

$$\text{first}+1 \leq \text{ i } \leq \text{lo} \Rightarrow \text{A[i]} \leq \text{pivot and hi} < \text{i} \leq \text{last} \Rightarrow \text{A[i]} \geq \text{pivot}$$

```
    void quicksort(int first,int last) // sorts the subarray A[first .. last]
     {
      if(first < last) // if first >= last, we are done
       {
        int mid = (first+last)/2;
        int pivot = A[mid];
        swap(A[mid],A[first]); // move pivot to first position
        int lo = first;
        int hi = last;
  // LOOP inveriant holds HERE
        while(lo < hi) // MAIN PARTITION LOOP
         {
  // LOOP inveriant holds HERE
          if(A[lo+1] > pivot and A[hi] < pivot)
           {
            swap(A[lo+1],A[hi]);
            lo++;
            hi--;
           }
          if(A[lo+1] <= pivot) lo++;
          if(A[hi] >= pivot) hi--;
  // LOOP inveriant holds HERE
         } END MAIN PARTITION LOOP
  // LOOP inveriant holds HERE
        //assert(lo == hi);
        swap(A[first],A[lo]); // place pivot between subarrays
        quicksort(first,lo-1);  // sort left subarray
        quicksort(lo+1,last);  // sort right subarray
       }
     }
```