University of Nevada, Las Vegas Computer Science 477/677 Fall 2026

Third Examination April 15, 2026

Name:_____

No books, notes, scratch paper, or calculators. Use pen or pencil, any color. Use the rest of this page and the backs of the pages for scratch paper. If you need more scratch paper, it will be provided. If you want anything on extra pages to be graded, staple those pages to your test and write, "Please grade this page."

The entire examination is 355 points.

1. [10 points] What are the important properties of a good hash function?

   Deterministic.
   Easy to calculate.
   No locality.
   Equally distribued.
   Should appear random, even though it isn't.

2. [10 points] Name two greedy algorithms introduced in class this semester.

   Huffman's and **Kruskal's** .

3. [10 points] In closed hashing, collisions can resolved by the use of probe sequences. However, probe sequences are unnecessary if a **perfect** hash function is used.

4. [5 points] In a **cuckoo** hash table, an item can eject a previously stored item, which then has to use an aternative hash value.

5. [5 points] The worst case time complexity of quicksort on a list of length n is $O(n^2)$.

6. [10 points] The height of a treap with n items is $O(n)$ in the worst case, but $O(\log n)$ in the average case.

7. [5 points] In an open hash table, the items at each index of the table must be held in a **search structure** which is frequently shown as a list.

8. [5 points] The asymptotic complexity of the Floyd/Warshall algorithm is $\Theta(n^3)$

9. [5 points] The asymptotic complexity of Dijkstra's algorithm algorithm is $O(m \log n)$

10. [10 points] A directed graph is **strongly connected** if there is a directed path from any vertex to any other vertex.

11. [5 points] **Huffman's** algorithm finds a binary code so that the code for one symbol is never a prefix of the code for another symbol.

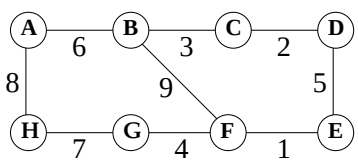
12. [5 points] An acyclic directed graph with 8 vertices must have at least **8** strong components. (Must be exact answer.)

13. [5 points] In **open hashing** or **separate chaining** there can be any number of items at a given index of the hash table.

14. [10 points] Write the prefix expression equivalent to the infix epression $a * -(b \wedge c \wedge d) + e$ (Don't forget that $\wedge$ means exponentiation.)

$+ * a \sim \wedge b \wedge cde$

15. [20 points] Walk through Kruskal's algorithm to find the minimum spanning tree of the connected weighted graph G shown below. Show the evolution of the union/find structure. Whenever there is choice between two vertices of equal weight, choose the alphabetically larger vertex. Indicate path compression when it occurs.



To work Kruskal's algorithm, we first sort the edges of G in order of weight. We then execute union(p, q) for each edge (p, q), in that weight order. After that step, p and q will be in the same set. At some point all vertices will be in the same set, and we halt.

Execution of union(p, q) is as follows.

$r =$ find(p)
$s =$ find(q)
If $(r \neq s)$
  union(r, s)
  Add the edge (p, q) to the minimum spanning tree.

Path compression only occurs during find. Here is recursive pseudocode for find, which also executes path compression if indicated.

find(x):
z = parent(x)
if$(z \neq x)$
  $w =$ find(z)
  parent$(x) = w$ (This is the compression step.)
return w

Here is a list of the edges of G is weight order.

(E, F) of weight 1
(C, D) of weight 2
(B, C) of weight 3
(F, G) of weight 4
(D, E) of weight 5
(A, B) of weight 6

(G, H) of weight 7
(A, H) of weight 8
(B, F) of weight 9

We will halt after processing the first 7 edges, because they will form a minimal spanning tree, and the vertices will all be in the same set.

Process (E, F): find$(E) = E$ and find$(F) = F$, both of which have weight 1. Set parent$(E) = F$. The weight of F will now be 2. (E, F) is added to the MST.

Process (C, D): find$(C) = C$ and find$(D) = D$, both of which have weight 1. Set parent$(C) = D$. The weight of D will now be 2. (C, D) is added to the MST.

Process (B, C): find$(B) = B$, which has weight 1, and find$(C) = D$, which has weight 2. Then excute union(B, D). Then parent$(B) = D$. The weight of D will now be 3. (B, C) is added to the MST.

Process (F, G): find$(F) = F$, of weight 2, and find$(G) = G$, of weight 1., Then parent$(G) = F$. The weight of F is now 3. (F, G) is added to the MST.

Process (D, E): find$(D) = D$ and find$(E) = F$, both of which have weight 3. Then parent$(D) = F$. The weight of F is now 6. (D, E) is added to the MST.

Process (A, B): find$(A) = A$. parent$(B) = D$ and thus find$(B) =$ find$(D) = F$. According to the code of find, we now have path compression, hence parent$D = F$, and we execute union(A, F), setting parent$(A) = F$. The weight of F is now 7.

Process (G, H): find$(G) = F$ of weight 7, and find$(H) = H$, of weight 1. Thus parent$(H) = F$, which now has weight 8. (G, H) is added to the MST. All vertices are in the same set, and we halt.

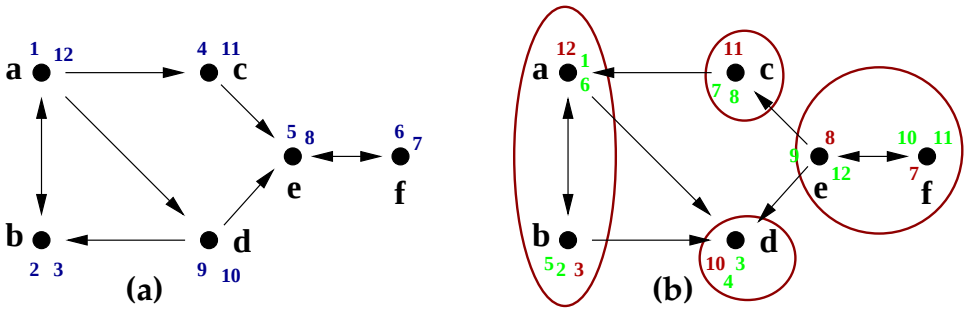
16. [10 points] You are trying to construct a cuckoo hash table of size 8 holding 8 names. Each of the names listed below has two possible hash values, as indicated in the array. Insert the items in alphabetic order. Instead of erasing ejected items, simply strike them out, so that I can see your steps.

Abe	2	7
Ben	3	6
Deb	0	6
Eli	1	7
Kat	1	2
Mel	0	4
Ron	1	5
Sue	5	7

0	
1	
2	
3	
4	
5	
6	
7	

If you execute the algorithm correctly, it will never converge. But how can you prove that no assignment is possible? Use Hall's Marriage Theorem. The set of 5 data $\{Abe, Eli, Kat, Ron, Sue\}$ have only 4 possible hash values, namely 1,2,5,7. Thus, it is impossible for all 5 of these data to fit into the hash table.

17. [20 points] Use the DFS method to compute the strong components of the digraph shown in (a) below. Use (a) and (b) for your work. Circle the strong components in (b).



We use Kosaraju's algorithm. First execute DFS on the graph, marking each vertex with its pre-order and post-order. We then reverse the graph and label each node with the post-order we just computed, shown in red. We now run DFS again, where whenever the stack is empty, we begin DFS with the unvisited vertex of maximum red number. Each time the stack is empty, the set of nodes popped off since the last time it was empty is a strong component.

18. [20 points] Explain how to implement a sparse array using a search structure.

Let A be the sparse virtual array. Let S be a search structure which contains ordered pairs of the form (i, x), where $A[i] = x$. To fetch the value of $A[i]$, search S for a pair (i, x). If that pair is found, return x, otherwise return a default value, such as 0. To store a value x for $A[i]$, search S for a pair (i, y). If that pair is found, replace y by x. That pair is not found, insert the pair (i, x) into S.

19. Consider the following recursive C++ subprogram.

```
int f(int n)
  {
   if(n <= 2)
    return 1;
   else
    return f(n/2)+f(1+n/2)+n*n
    return 0;
  }
```

(a) [10 points] What is the asymptotic value of the function $f(n)$ computed by the recursive code?

$\Theta(n^2)$

(b) [10 points] What is the asymptotic time complexity of the computation of $f(n)$ using the recursive code? (Hint: not the same answer as above.) $\Theta(n)$

(c) [10 points] What is the asymtotic time complexity of a computation of f(n) using memoization?

$\Theta(\log n)$

20. True or False.

(a) [5 points] **T** In SHA256 hashing, collisions are so unlikely that if that algorithm were used 10 times a day by every person on the Earth for a thousand years, there would probably not be a collision.

(b) [5 points] **F** Open hashing uses open addressing.

(c) [5 points] **T** False overflow for a queue can be avoided by circular list implementation.

(d) [5 points] **T** If a stack is implemented as a linked list, the head of the linked list should hold the top item of the stack.

(e) [5 points] **F** Kruskal's algorithm uses dynamic programming.

(f) [5 points] **F** There will be no collisions if the size of a hash table is at least the square of the number of data items.

21. Solve each recurrence, expressing each answer in terms of n, using Θ notation.

(a) [5 points] $F(n) = 3F(n/3) + 3F(2n/3) + n^2$

$\Theta(n^2 \log n)$

(b) [5 points] $F(n) = F(n/3) + F(2n/3) + 1$

$F(n) = \Theta(n)$

(c) [5 points] $G(n) = 2G(n/4) + \sqrt{n}$

$G(n) = \Theta(\sqrt{n} \log n)$

(d) [5 points] $H(n) = \log n + 1$

$H(n) = \Theta(\log^* n)$

(e) [5 points] $H(n) = 4H(2n/5) + H(3n/5) + 2n^2$

$4(2/5)^2 + (3/5)^2 = 1$. Therefore $H(n) = \Theta(n \log n)$.

(f) [5 points] $G(n) = 4(G(n/2) + 5n^2$

$4(1/2)^2 = 1$, therefore $G(n) = \Theta(n \log n)$.

(g) [5 points] $F(n) = F(n - \log n) + \log^2 n$

$$\frac{F(n) - F(n - \log n)}{\log n} = \frac{\log^2 n}{\log n}$$

$F'(n) = \Theta(\log n)$

$F(n) = \Theta(n \log n)$

22. Find the time complexity of each of these code fragments in terms of n, using Θ notation.

(a) [5 points]

```
for(int i = 1; i < n; i = 2*i)
 cout < "Hello world!";
```

$\Theta(\log n)$

(b) [5 points]

```
for(int i = n; i > 1; i = sqrt(i))
 cout < "Hello world!";
```

Assume that `sqrt(i)` returns the floor of the square root of i.

$\Theta(\log \log n)$

(c) [5 points]

```
for(int i = 1; i < n; i++)
 for(int j = i; j < n; j=2*j)
  cout < "Hello world!";
```

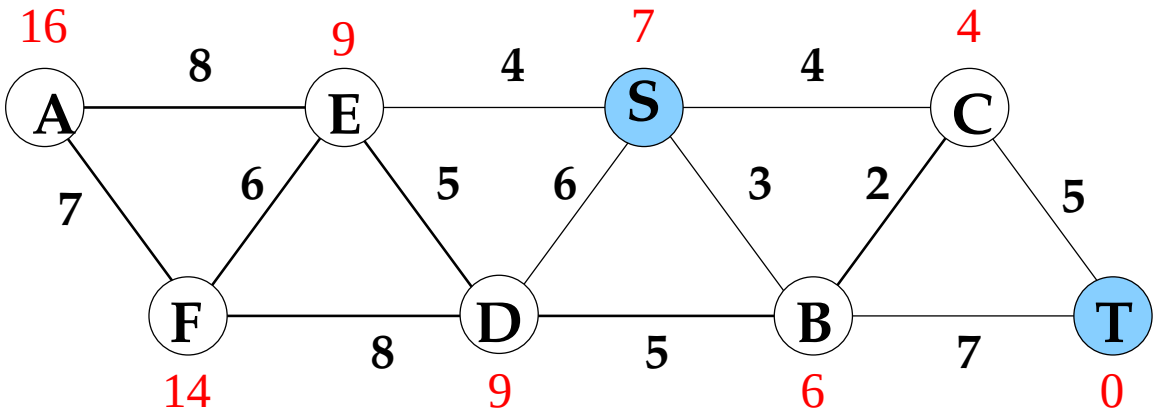
$\Theta(n)$

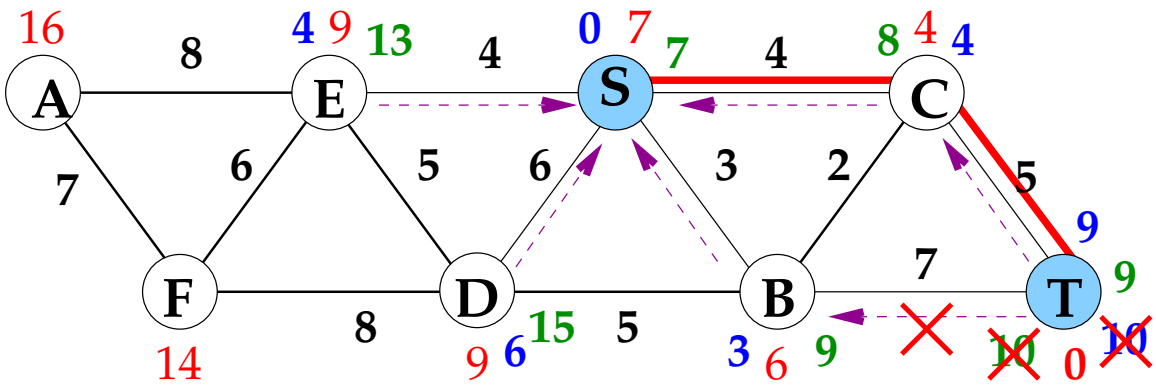
23. [20 points]j Find the Levenshtein edit distance from the word "mennoover" to the word "maneuver." Show the matrix.

		m	a	n	e	u	v	e	r
	0	1	2	3	4	5	6	7	8
m	1	0	1	2	3	4	5	6	7
e	2	1	1	2	2	3	4	5	6
n	3	2	2	1	2	3	4	5	6
n	4	3	3	2	2	3	4	5	6
o	5	4	4	3	3	3	4	5	6
o	6	5	5	4	4	4	4	5	6
v	7	6	6	5	5	5	4	5	6
e	8	7	7	6	5	6	5	4	5
r	9	8	8	7	6	7	6	5	4

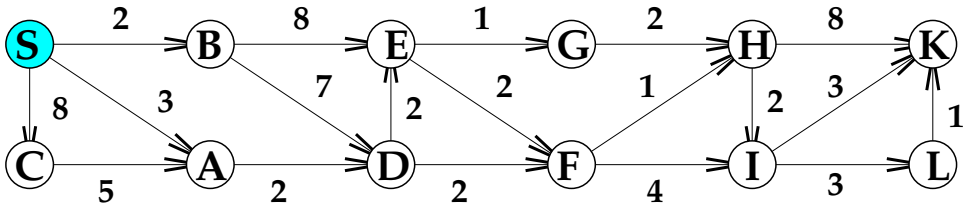
The edit distance is 4.

24. [20 points] Work the A* algorithm for the following weighted digraph, where the heuristic values are printed in red. Draw back pointers and write values of g and f in pencil or any color pen. I'll be able read it if you try to be neat. To make neatness easier, I have made the diagram large.



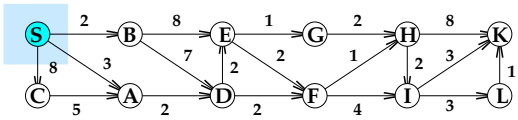


25. [15 points] Walk through Dijkstra's algorithm for the following graph.

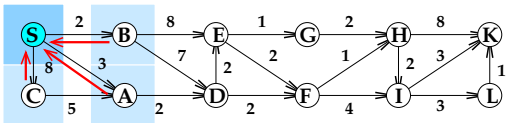


	S	A	B	C	D	E	F	G	H	I	K	L
V	0											
back												

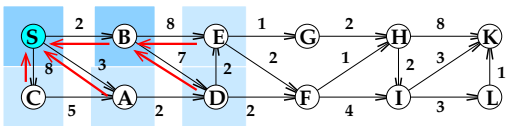
Fully processed nodes have a blue background.



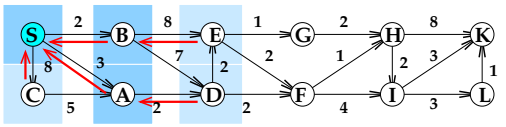
	S	A	B	C	D	E	F	G	H	I	K	L
V	0											
back												



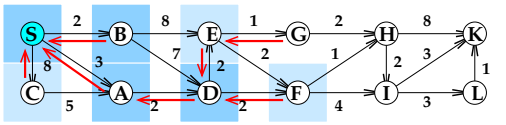
	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8								
back		S	S	S								



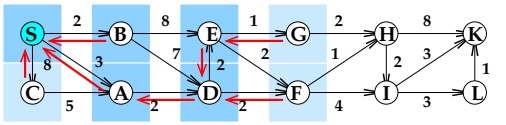
	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8	9	10						
back		S	S	S	B	B						



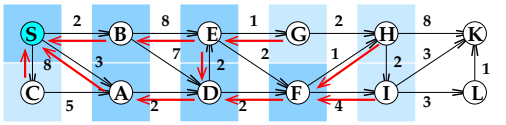
	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8	~~9~~ 5	10						
back		S	S	S	~~B~~ A	B						



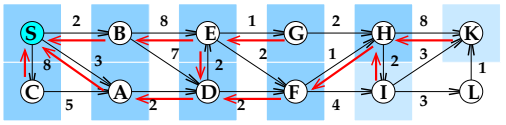
	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8	~~9~~ 5	~~10~~ 7	7					
back		S	S	S	~~B~~ A	~~B~~ D	D					



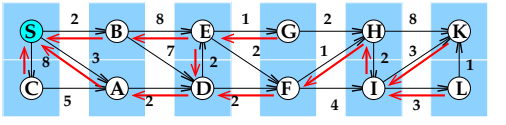
	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8	~~9~~ 5	~~10~~ 7	7	8				
back		S	S	S	~~B~~ A	~~B~~ D	D	E				



	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8	~~9~~ 5	~~10~~ 7	7	8	8	11		
back		S	S	S	~~B~~ A	~~B~~ D	D	E	F	F		



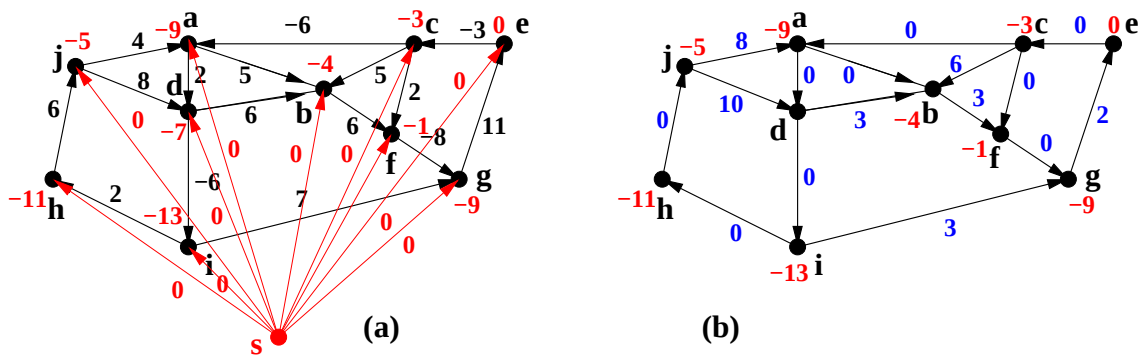
	S	A	B	C	D	E	F	G	H	I	K	L
V	0	3	2	8	~~9~~ 5	~~10~~ 7	7	8	8	~~11~~ 10	16	
back		S	S	S	~~B~~ A	~~B~~ D	D	E	F	~~F~~ H	H	



	S	A	B	C	D	E	F	G	H	I		K	L
V	0	3	2	8	~~9~~ 5	~~10~~ 7	7	8	8	~~11~~ 10		~~16~~ 13	13
back		S	S	S	~~B~~ A	~~B~~ D	D	E	F	~~F~~ H		~~H~~ I	I

26. [20 points] Figure (a) below shows a weighted digraph. Work the first part of Johnson's algorithm using the copies of the graph shown. Make sure that you show the adjusted weights on one of the graphs.

Do not complete the computation of Johnson's algorithm.

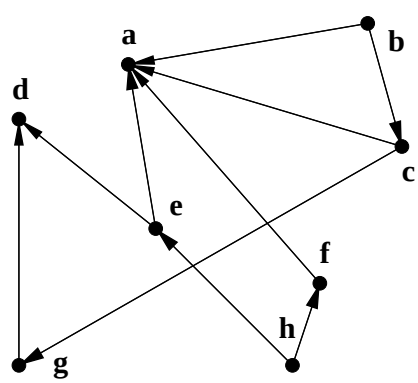


27. [10 points] If the array $A[5][7]$ is stored in column-major order, how many predecessors does $A[3][4]$ have?

4*5 + 3 = 23

28. [10 points]

Write the array of in-neighbor lists and the array of out-neighbor lists for the directed graph shown below.



a	
b	a,c
c	a,g
d	
e	a,d
f	a
g	d
h	e,f

a	b,c,e,f
b	
c	b
d	e,g
e	h
f	h
g	c
h	

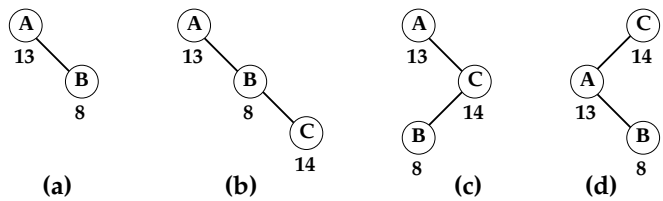
29. [20 points] Let G be a weighted acyclic diagraph, with vertices $v_1, \ldots v_n$, indexed in topological order, where $w(v_i, v_j)$ is the weight of the arc (v_i, v_j) if that arc exists. Write pseudocode for an algorithm which computes the maximum weight directed path in G. Compute backpointers.

Let in-neighbors(i) be the set of in-neighbors of i. Define $W(i)$ to be the maximum weight of any directed path ending at v_i, and let back(i) be the backpointer of that path if it is non-trivial. We compute $W(i)$, back(i), in increasing order.

For i from 1 to n
  $W(i) = 0$
  For j from 1 to $i - 1$
    If(j is an in-neighbor of i)

If$(w(j,i) + W(j) > W(i))$
$\quad W(i) = w(j,i) + W(j)$
$\quad \text{back}(i) = j$

30. [10 points] You need to store the items A, B, and C, in that order, in a treap. The priority for A is 13, for B is 8, and for C is 14. Use maxheap order. Draw the resulting treap after each insertion, and show each rotation.



We first insert A, then B. Heap order is preserved, so no rotation is necessary. We next insert C. To restore heap order, we do a left roatation at B. Heap order is still not restored. We do a left rotation at A. Heap order is then restored.

31. [15 points] Here is another coin-row problem. You have a row of coins of various values, where the value of the ith coin is $V[i] > 0$. Write pseudocode which finds the maximum value of a subset of coins, where the set may not contain coins which are either adjacent or just one apart in the row. That is, if the set contains the ith coin, it may not contain either the $(i+1)^{\text{st}}$ coin or the $(i+2)^{\text{nd}}$ coin. For example, if the coins are ⓐ ⓑ ⓒ ⓓ ⓔ ⓕ ⓖ ⓗ in that order, the subset may be {ⓐ, ⓓ, ⓗ}, but not {ⓑ, ⓓ, ⓖ}.

We have two dynamic programs for this problem.

- Let $A[i]$ be the maximum sum of any legal sequence ending at i. Then the answer is $max(A[n-2], A[n-1], A[n])$ where the values of A are computed as follows.

```
A[1] = V[1]
A[2] = V[2]
A[3] = V[3]
A[4] = V[4] + A[1]
A[5] = V[5] + max(A[1],A[2])
for i from 6 to n
  A[i] = V[i] + max(A[i-5],A[i-4],A[i-3])
```

- Let A[i] be the maximum sum of any legal subsequence of the first i coins. The answer is then $A[n]$ where the values of A are computed as follows.

```
A[1] = V[1]
A[2] = max(V[2],A[1])
A[3] = max(V[3],A[2])
for i from 4 to n
  A[i] = max(A[i-1],V[i] + A[i-3])
```