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

Answers to Examination November 15, 2021

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 210 points.

1. Fill in the blanks.

   (a) [5 points] A strongly connected directed graph with n vertices and m arcs cannot have more than **one** strong component. (Exact answer please. No partial credit.)

   (b) [15 points] Name three kinds of search structure that we've covered in lecture this semester.

   binary search tree

   hash table

   treap

   ordered list

   unordered list

2. [20 points] Explain how to implement a sparse array as a search structure of ordered pairs.

   It is fully explained in the answers to the practice test.

3. [20 points] Find the Levenshtein edit distance from the string "bcbaca" to the string "abacbca." Show the matrix.

   You can let the rows correspond to the symbols of the first string or the second, the problem can be worked either way. I will let the rows correspond to the symbols of the second string.

		b	c	b	a	c	a
	0	1	2	3	4	5	6
a	1	1	2	3	3	4	5
b	2	1	2	2	3	4	5
a	3	2	2	3	2	3	4
c	4	3	2	3	3	2	3
b	5	4	3	2	3	3	3
c	6	5	4	3	3	3	4
a	7	6	5	4	3	4	3

   The edit distance is 3.

4. [20 points] Write pseudocode for the Floyd-Warshall algorithm. Assume that a weighted directed graph G is given, whose vertices are the integers in the range $1, \ldots n$. Weights of the arcs are given – the weight of the arc from i to j is $W[i, j]$; if there is no such arc, $W[i, j] = \infty$. Your code must compute back pointers.

   This code is given in the answers to the practice test.

5. Consider the following code for a recursive C++ function.

```
int f(int n)
 {
  if(n > 0) return f(n/4)+f(n/4+1)+f(n/4+2)+f(n/4+3)+n*n;
  else return 0;
 }
```

   (a) [10 points] What is the asymptotic complexity of f as a function of n, using Θ notation?

   The recurrence is $f(n) = 4f(n/4) + n^2$. By the master theorem, the solution is $\Theta(n^2)$.

   (b) [10 points] What is the asymptotic time complexity of the code as a function of n, using Θ notation?

   The recurrence is $T(n) = 4T(n/4) + 1$. By the master theorem, the solution is $\Theta(n)$.

(c) [10 points] Write pseudo-code for a dynamic programming algorithm to compute $f(n)$ for a given n. What is the asymptotic time complexity of your code as a function of n, using Θ notation?

We compute f[i] for all $0 \le i \le n$, in increasing order.

```
f[0] = 0
for(int i = 1; i <= n; i++)
  f[i] = f[i/4]+f[i/4+1]+f[i/4+2]+f[i/4+3] + i*i;
cout << f[n]
```

We computed $n+1$ values, each of which takes $\Theta(1)$ time, thus the time complexity is $\Theta(n)$.

(d) [20 points] Write pseudo-code for a memoization algorithm to compute $f(n)$ for a given n. What is the asymptotic complexity of your code, in terms of n, using Θ notation?

Initialize an empty search structure whose items are ordered pairs of integers. Store the memo (0,0) in the search structure. Let f be the recursive function given below.

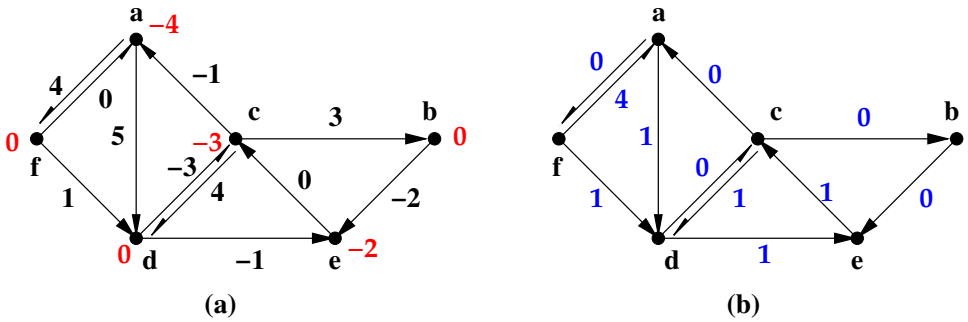
```
int f(int i)
  {
   If there is a memo (i,x)) in the search structure, return x.

   Else, let result = f(i/4)+f(i/4+1)+f(i/4+2)+f(i/4+3) + i*i,
   then store the memo (i,result) in the search structure,
   then return result.
  }
```

Print the value of f(n).

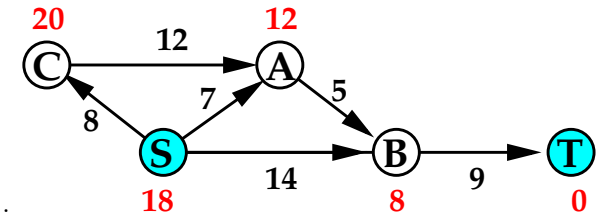
You can prove that there are no more than $(\log_4 n + 1)$ predecessors of n, although it's tricky. Thus the time complexity is $\Theta(\log n)$.

6. [20 points]

The first step of Johnson's algorithm is to compute the heuristic function. On the weighted directed graph (a) below, label each node of (a) with the correct heuristic. (You do not have to show the steps of the algorithm for this. The example is small enough that you can simply compute the values in your head.) The next step is to compute adjusted arc weights. Label the arcs of (b) with the adjusted weights.



7. [20 points] Walk through the A^* algorithm for the weighted directed graph shown below, where the pair is (S, T). The heuristic is shown as red numerals. Show the heap and the arrays at each step.



.

Heap: S

	S	A	B	C	T
h	18	12	8	20	0
f	0				
g	18				
back					

Heap: ABC

	S	A	B	C	T
h	18	12	8	20	0
f	0	7	14	8	
g	18	19	22	28	
back		S	S	S	

Heap: BC

	S	A	B	C	T
h	18	12	8	20	0
f	0	7	12	8	
g	18	19	20	28	
back		S	A	S	

Heap: TC

	S	A	B	C	T
h	18	12	8	20	0
f	0	7	12	8	21
g	18	19	20	28	21
back		S	A	S	B

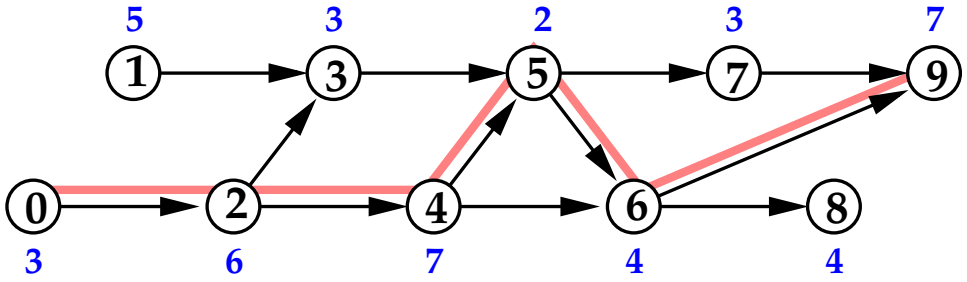
Heap: C

	S	A	B	C	T
h	18	12	8	20	0
f	0	7	12	8	21
g	18	19	20	28	21
back		S	A	S	B

T is fully processed, and we are done. The shortest path from S to T is (S,A,B,T) obtained by following the back pointers.

8. [40 points] Write pseudocode for a DP algorithm which finds the maximum weight directed path in a vertex-weighted acyclic directed graph. Assume that the vertices are the integers $0, \ldots n-1$, and all arcs are of the form (i, j) where $i < j$. Arcs are not weighted.

Let **int W[n]** be the array of vertex weights. Here is an instance of such a graph for n = 10, where **W[i]** is in blue for each vertex i. The maximum weight of a directed path is 29, and the maximum weight path is indicated in color.



Assume the input includes a triangular 2-dimensional Boolean array **arc** where **arc[i,j]** is true if and only if there is an arc from i to j. In the instance shown above, for example, **arc[2.3] = true**, while **arc[3,4] = false**.

This dynamic program should be correct. V[i] = maximum weight of any path beginning at i.

```
for(int i = n; i >= 0; i--)
 {
  s = n;
  maxforw[i] = 0;
  for(int j = i+1; j <= n; j++)
   {
    if(arc[i,j])
     if(V[j] > maxforw[i])
      {
       forw[i] = j;
       maxforw[i] = V[j];
      }
    V[i] = W[i] + maxforw[i];
   if(V[i] > V[s]) s = i;
  }
  cout << "The maximum weight of any directed path is " << V[s] << endl;
  cout << "Here is the path." << endl;
  while(V[s] > 0)
   {
    cout << s;
    s = forw[s];
   }
  cout << endl;
 }
```