Computer Science 477/677 Fall 2019
University of Nevada, Las Vegas
Answers to Third Examination November 20, 2019

1. True or False. Write T if the statement is known to be true, F it is known to be false, and O if it is open, meaning that science has not determined whether it is true or false.

   (a) **F** Open hashing uses open addressing.

   (b) **O** Every problem that can be worked in $O(n)$ time by one processor can be worked in $O(\log^k n)$ time with $O(n^k)$ processors, for some k.

   If $\mathcal{P} = \mathcal{NC}$, the answer is true, otherwise it is false.

2. Fill in the blanks.

   (a) In open hashing, suppose the number of data items equals the size of the table. Approximately what proportion of the buckets are empty? Give one of the following answers: less than 10%, between 10% and 20%, between 20% and 40%, more than 40%. Between 20% and 40%. The expected proportion is $1/e \approx 0.368$.

   (b) There are no collisions in a **perfect** hash table.

3. Explain how to find the sum of n numbers using $n/\log n$ processers in logarithmic time.

   partition the numbers into batches of size $\log n$. The sum of each batch is computed sequentially by one processor, resulting in $n/\log n$ subtotals. Use the tournament method to find the total in $O(\log n)$ time with $n/\log n$ processors.

4. Compute the $(\max, +)$ matrix product:

$$\begin{pmatrix} 2 & 5 & -1 \\ 5 & 0 & -\infty \\ 0 & -\infty & -\infty \end{pmatrix} \begin{pmatrix} 5 & 0 & -\infty \\ -1 & 2 & 1 \\ 4 & -\infty & 5 \end{pmatrix} = \begin{pmatrix} 7 & 7 & 6 \\ 10 & 5 & 1 \\ 5 & 0 & -\infty \end{pmatrix}$$

5. Execute the following operations, in the given order, in a union/find structure. The items are the letters from A to H, and initially each letter is the leader of its own set. Use path compression. When there is a tie, the leader will be the larger letter.

   (a) union(A, B)

   (b) union(C, D)

   (c) union(B, D)

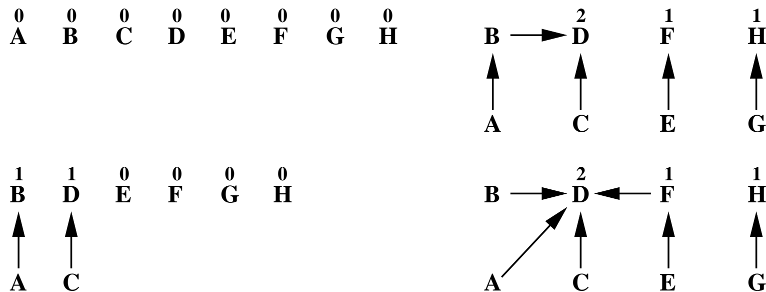
   (d) union(E, F)

   (e) union(G, H)

   (f) union(G, H)

   (g) union(A, E)

   Execute (a) and (b). Then execute (c), (d), (e), (f). Finally execute (g).

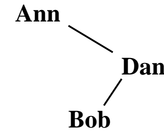


In order to execute union(X,Y), find(X) and find(Y) must be executed first. Thus, for example, union(A,B) is interprested to mean union(find(A),find(B)). We illustrate the initial configuration, then the configuration after executing (a) and (b), then after executing (c), (d), (e), and (f), and finally after executing (g).

6. Find an optimal binary search tree for the items (Ann,Bob,Dan). where out of every 10 operations:

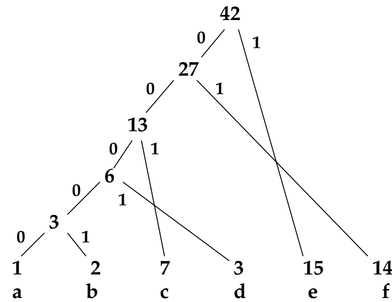
   - find(Ann) is executed 5 times,
   - find(Bob) is executed 2 times,
   - find(Dan) is executed 3 times.

   There are five possible binary search trees. This one is best.

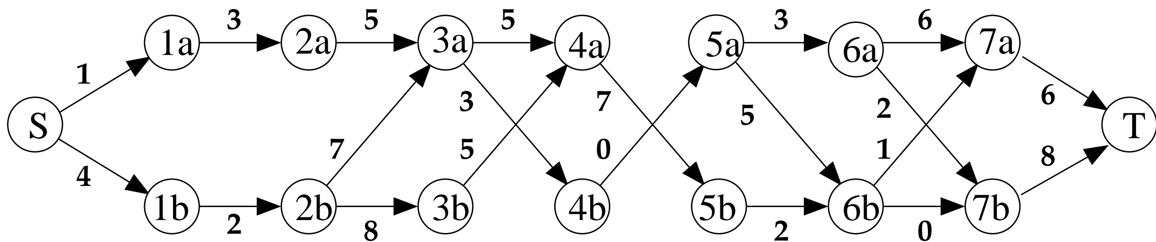


7. Construct an optimal Huffman code from the alphabet $\{a,b,c,d,e,f\}$ with the following frequencies:

a	1	00000
b	2	00001
c	7	001
d	3	0001
e	15	1
f	14	01



8. Consider the layered graph G shown below, where S is the 0th layer, $\{1a,1b\}$ is the first layer, etc. For $i<j$, let $M_{i,j}$ be the matrix corresponding to the step, or steps, between the ith layer and the jth layer. For example, $M_{6,7} = \begin{pmatrix} 6 & 2 \\ 1 & 0 \end{pmatrix}$



Write the matrices indicated below:

$$M_{0,1} = \begin{pmatrix} 1 & 4 \end{pmatrix}$$

$$M_{1,2} = \begin{pmatrix} 3 & -\infty \\ -\infty & 2 \end{pmatrix}$$

$$M_{2,3} = \begin{pmatrix} 5 & -\infty \\ 7 & 8 \end{pmatrix}$$

$$M_{3,4} = \begin{pmatrix} 5 & 3 \\ 5 & -\infty \end{pmatrix}$$

$$M_{4,5} = \begin{pmatrix} -\infty & 7 \\ 0 & -\infty \end{pmatrix}$$

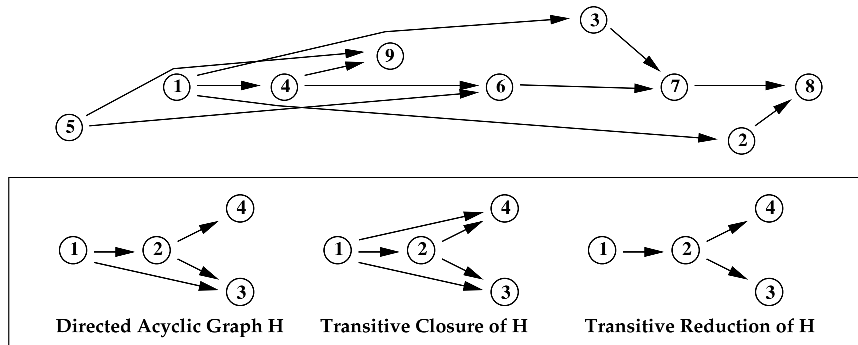
$$M_{5,6} = \begin{pmatrix} 3 & 5 \\ -\infty & 2 \end{pmatrix}$$

$$M_{6,7} = \begin{pmatrix} 6 & 2 \\ 1 & 0 \end{pmatrix}$$

$$M_{7,8} = \begin{pmatrix} 6 \\ 8 \end{pmatrix}$$

9. We say a sequence $\{x_i\}$ is *strictly monotone increasing* if $x_{i+1} > x_i$ for each i. Design an algorithm to find a maximum length strictly monotone increasing subsequence of a given sequence. Your algorithm should take $O(n)$ time. For example, if the input sequence is 5,1,4,9,6,3,7,2,8, a maximum length strictly increasing subsequence is 1,4,6,7,8.

   The problem reduces to finding the maximum path length in nn acyclic directed graph, G of n nodes There is an arc from $x[i]$ to $x[j]$ provided $i < j$ and $x[i] < x[j]$. We can replace G by its transitive reduction, since the longest path through a directed acyclic graph is equal to the longest path through its transitive reduction, and also equal to the longest path through its transitive closure. (See box.)



   The maximum length of a path in G is computed by dynamic programming. Here is a simple quadratic time algorithm for the problem.

```
int const N = 100;
int n;
int x[N]; // x[1], x[2], ... x[n] are read from infile
int length[N]; // length[i] = length of longest mono subs ending at x[i]
int back[N]; // back[i] = max{j le i | length[j] + 1 = length[i]
int lasti; // index of last term of maximum increasing subsequence so far
```

```
void readx()
 {
  cin >> n;
  assert(n > 0);
  assert(n < N);
  for(int i = 1; i <= n; i++)
   cin >> x[i];
 }

void writex()
 {
  cout << "Input sequence:" << endl;
  for(int i = 1; i <= n; i++)
  cout << " " << x[i];
  cout << endl;
 }

void writesequence(int i)
 {
  if(back[i] != 0)
   writesequence(back[i]);
  cout << " " << x[i];
 }

void writesequence()
 {
  cout << "Maximum length strictly monotone subsequence:" << endl;
  writesequence(lasti);
  cout << endl;
 }

void simple()
 {
  readx();
  writex();
  maxell = 0;
  lasti = 0;
```

```
   for(int i = 1; i <= n; i++)
    {
     length[i] = 1;
     back[i] = 0;
     for(int j = i-1; j > 0; j--) // LINEAR SEARCH for best back(i)
      if(x[j] < x[i] and length[j] >= length[i])
       {
        length[i] = length[j]+1;
        back[i] = j;
        if (length[i] > maxell)
         {
          maxell = length[i];
          lasti = i;
         }
       }
    }
   writesequence();
  }

 int main()
  {
   simple();
  }
```

Speedup.

The longest monotone increasing subsequence can be found in $O(\log n)$ time. During each iteration of the main loop, the term `x[i]` of the original sequence is processed, and the values `length[i]` and `back[i]` are computed

Instead of linear search, it is possible to use binary search to find the best choice of `back[i]` For each index i, length[i] is the length of the longest monotone subsequence ending at x[i], and back[i] is the index of the predecessor of x[i] in the subsequence. If length[i] = 0, then back[i] = 0 by default.

We say that an index j is *dominated* by i if i > j, x[i] < x[j], and length[i] = length[j]. We maintain an array consisting of all undominated indices, ordered by their values of length. The values of x[i] will then also be increasing. We can use binary search to determine which previous j that i should dominate, or whether the length[i] is a new larger value. Note that after every step, all rows of the second array are increasing.

We first initialize the arrays with i = 1, which has length 1.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1								
back[i]	0								

length[i]	1
i	1
x[i]	5
back[i]	0

We now process i = 2, which dominates and replaces i = 1.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1							
back[i]	0	0							

length[i]	1
i	2
x[i]	1
back[i]	0

We now process i = 3, which has length 2 and does not dominate.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1	2						
back[i]	0	0	2						

length[i]	1	2
i	2	3
x[i]	1	4
back[i]	0	2

We now process i = 4, which has length 3 and does not dominate.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1	2	3					
back[i]	0	0	2	3					

length[i]	1	2	3
i	2	3	4
x[i]	1	4	9
back[i]	0	2	3

We now process i = 5, which dominates and replaces i = 4.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1	2	3	3				
back[i]	0	0	2	3	3				

length[i]	1	2	3
i	2	3	5
x[i]	1	4	6
back[i]	0	2	3

We now process i = 6, which dominates and replaces i = 3.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1	2	3	3	2			
back[i]	0	0	2	3	3	2			

length[i]	1	2	3
i	2	6	5
x[i]	1	2	6
back[i]	0	1	3

We now process i = 7, which has length 4 and does not dominate.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1	2	3	3	2	4		
back[i]	0	0	2	3	3	2	5		

length[i]	1	2	3	4
i	2	6	5	7
x[i]	1	3	6	7
back[i]	0	1	3	5

We now process i = 8, which dominates and replaces i = 6.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1	2	3	3	2	4	2	
back[i]	0	0	2	3	3	2	5	1	

length[i]	1	2	3	4
i	2	8	5	7
x[i]	1	2	6	7
back[i]	0	1	3	5

We now process i = 9, which has length 5 and does not dominate.

i	1	2	3	4	5	6	7	8	9
x[i]	5	1	4	9	6	3	7	2	8
length[i]	1	1	2	3	3	2	4	2	5
back[i]	0	0	2	3	3	2	5	1	7

length[i]	1	2	3	4	5
i	2	8	5	7	9
x[i]	1	2	6	7	8
back[i]	0	1	3	5	7

We now use the backpointers to recover the maximum length monotone increasing subsequence.

- Index 9 has the greatest length.
- back[9] = 7.
- back[7] = 5.
- back[5] = 3.
- back[3] = 2.
- back[2] = 0.

The sequence is x[2], x[3], x[5], x[7], x[9], that is: 1, 4, 6, 7, 8.