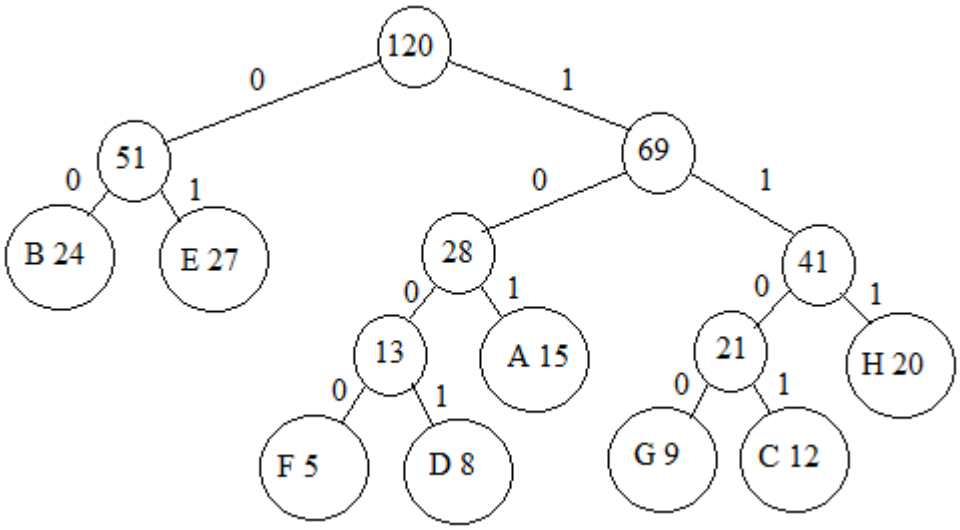
# Spring 2018 COP 3503 Section 1 Exam #2 Solutions

1) (15 pts) Huffman Coding

Consider a file that contains the following eight distinct characters (each encoded using 3 bits) with the following frequencies:

| Character | Frequency | Huffman Code |
|---|---|---|
| A | 15000 | **101** |
| B | 24000 | **00** |
| C | 12000 | **1101** |
| D | 8000 | **1001** |
| E | 27000 | **01** |
| F | 5000 | **1000** |
| G | 9000 | **1100** |
| H | 20000 | **111** |

a) (5 pts) Draw a Huffman Coding Tree for this file below.




b) (5 pts) In the chart above, fill in the corresponding Huffman codes for each character in the file based on the tree you drew in part a. Please use '0' = left, '1' = right.

c) (5 pts) How many bits would be saved using the Huffman coding on this file, assuming that the code itself took 100 bits to store?

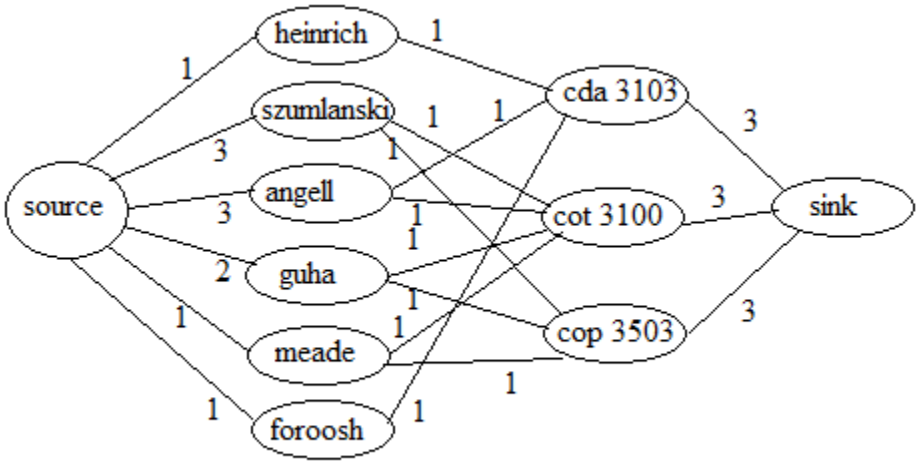
**Bits Saved = (Bits Saved for B, E) - (Extra Bits Used for F, G, D and C) - (Code Storage)**
**= 24000 + 27000 - 5000 - 9000 - 8000 - 12000 - 100**
**= 51000 - 34000 - 100**
**= 16900**

**Grading: 1 pt off per tree error (cap at 5), for codes, base on their tree, 1 pt off per error, cap at 5, for bits saved, 4 pts for a valid set up for calculation, 1 pt for arithmetic.**

2) (15 pts) Network Flow Algorithm Design

Consider the problem of assigning teachers to classes at Triple Threat University. At the university, there are three sections of each class offered, each at the same exact time. All different classes are offered at different times. Each professor the university hires has a maximum number of courses they can teach and a list of courses for which they are proficient in teaching and are available during the time period when the class is taught. (Unlike some places, Triple Threat refuses to put a teacher in a classroom for which they aren't proficient.) Given all of this data, design an efficient algorithm to determine if it is possible or not for Triple Threat University to assign professors to each of its classes. After describing your algorithm, create a small example and draw the corresponding network flow graph for that one example.

**We can solve this problem by setting up a network flow graph, and checking to see if the maximum flow in the graph is equal to the number of sections of courses offered by the university. The graph will have a source and sink vertex, as well as a vertex for each professor and a vertex for each course. The source will connect to each vertex representing a professor with a capacity equal to the maximum number of courses that professor can teach. Each professor vertex will connect to each course vertex with capacity one, if and only if the professor is proficient in teaching that course and is available at the particular time slot for that course. Finally, each course vertex will connect to the sink vertex with a capacity of 3, since we must have three different professors cover three different sections of each course at the same time. If the maximum flow of this graph is equal to three times the number of courses, then there exists a valid schedule to have each section of each course covered by a professor. Consider the following example with 6 professors and 3 courses:**




Note: Direction of all edges is left to right.

**Grading: 3 pts for saying to set up some flow network with a source and sink, 2 pts for having a vertex for each professor. 2 pts for having a vertex either for each course or each section (there are valid solutions for both…), 3 pts for the capacity from source to professor equal to the maximum number of courses they can teach, 2 pts for having edges from professors to courses only if they are available and are qualified to teach that course, 3 pts for capacities from courses to sink equal to 3. (These can be 1 in a different design…)**

3) (12 pts) Dijkstra's Algorithm Trace

Trace through Dijkstra's algorithm on the graph described below **using C as the source vertex.** The last line of your chart should display the shortest distances from C to all of the vertices in the graph. In order to receive credit, you must properly fill in the chart, which shows incremental updates to the distance array. (Note: Each edge shown below is a directed edge.)

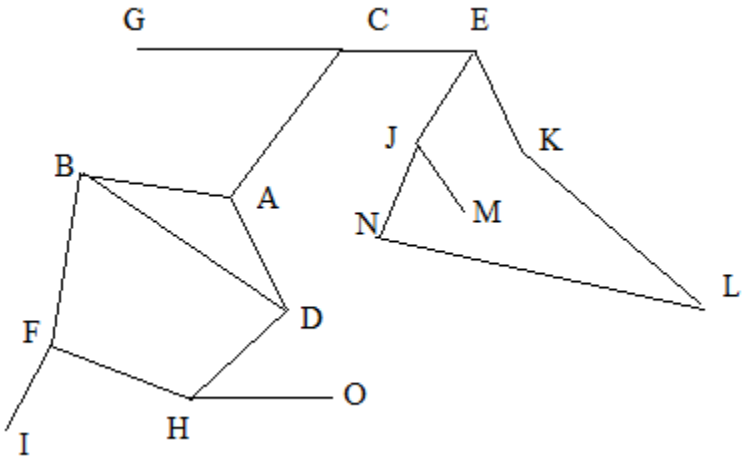
| | | | | |
|---|---|---|---|---|
| AB 4 | AD 12 | AE 10 | AF 5 | AG 3 |
| BA 5 | BD 15 | BE 20 | BG 30 | |
| CA 12 | CB 6 | CD 25 | CF 18 | |
| DA 5 | DB 9 | DE 6 | DF 12 | DG 15 |
| EA 20 | EB 18 | ED 1 | EG 6 | |
| FD 12 | FE 1 | GD 4 | GE 5 | GF 1 |

| Add to S | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| **C** | **12** | **6** | **0** | **25** | **inf** | **18** | **inf** |
| **B** | **11** | **6** | **0** | **21** | **26** | **18** | **36** |
| **A** | **11** | **6** | **0** | **21** | **21** | **16** | **14** |
| **G** | **11** | **6** | **0** | **18** | **19** | **15** | **14** |
| **F** | **11** | **6** | **0** | **18** | **16** | **15** | **14** |
| **E** | **11** | **6** | **0** | **17** | **16** | **15** | **14** |

**Grading: 2 pts per row, 1 pt off for a row if one or more values on it are wrong.**

4) (10 pts) Graphs – Depth First Search

Show the order in which the vertices get visited during a depth first search of the graph shown below, starting at vertex A. When determining the order in which to make the recursive calls from a given node, always go in alphabetical order.




**A, B, D, H, F, I, O, C, E, J, M, N, L, K, G**

**Grading: 10 pts if correct, -1 for each incorrect branch (trace future steps according to their previous steps), cap at 10 off.**

5) (8 pts) Greedy Algorithms

The 0-1 Knapsack problem is as follows: you are given a list of items, each item has an integer weight and integer value. The goal of the problem is to choose a subset of the items which have a sum of weights less than or equal to a given W with a maximal sum of values. For example, if we had the following five items (each in the form (weight, value)): $I_1$(6, 13), $I_2$(4, 10), $I_3$(1, 1), $I_4$(8, 12), and $I_5$(5, 9) and W = 13, then our maximal achievable value is 13 + 10 + 1 = 24, corresponding to the subset containing the first three items, which weigh 11 units, which is less than or equal to our weight limit of 13 units. A proposed greedy algorithm to solve the problem is as follows: (1) sort the items by per unit value. Go through the list of items in this order, taking each item as long as there is room to do so. (For this example, when we sort by per unit value, the ordering would be $I_2$, $I_1$, $I_5$, $I_4$, and $I_3$. Then, we'd take $I_2$, followed by $I_1$, skip $I_5$ and $I_4$ because enough weight wasn't left, then take $I_3$.) This algorithm is incorrect. Create a single example (a set of items with weights and values) for which this algorithm fails. In your example, show what answer the algorithm will produce and show a subset that is more valuable (but also within the weight limit.)

**Let there be three items $I_1$ has weight 5, value 10, $I_2$ has weight 5 value 9, and $I_3$ has weight 8 value 12. Let W = 9, the weight limit for our subset of items. The greedy algorithm proposed sorts the items in the order that they are presented here since their values per unit are 2 (10/5), 1.8 (9/5) and 1.5 (12/8), respectively. The greedy algorithm would then take the first item and then be unable to take either $I_2$ or $I_3$. In this case, the best solution would be to take $I_3$, which comes under the weight limit and has value 12.**

**Grading: 3 pts for listing any set of items with weights and values. 3 pts for an accurate trace through of what the greedy algorithm would do, 2 pts if their example is actually a counter-example and they give the real maximum.**

6) (15 pts) Graphs – Breadth First Search

Consider the problem of attempting to get from the top left corner to the bottom right corner of a square grid in the minimum number of moves where each square is marked as '.' (valid) or 'X' (invalid). (Both the top left and bottom right corners are always guaranteed to be valid squares.) The valid moves are up, down, left and right and the top left corner is coordinate (0, 0) and the bottom right coordinate is (n-1, n-1), where n is given in the input. Complete the code below to correctly solve the problem. All of the I/O is handled and you just have to fill in portions of the method that conducts the breadth first search. (Note: The method is designed only to return the shortest distance to the bottom right corner, not to necessarily mark all of the shortest distances from the top left to all other squares.) The method returns -1 if there is no valid path.

```
import java.util.*;

public class bfs_maze {

    public static int n;
    public static char[][] grid;
    final public static int[] DX = {-1,0,0,1};
    final public static int[] DY = {0,-1,1,0};
```

```
    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);
        int nC = stdin.nextInt();

        for (int loop=0; loop<nC; loop++) {
            n = stdin.nextInt(); // Board from (0,0) to (n-1,n-1).
            grid = new char[n][];
            for (int i=0; i<n; i++)
                  grid[i] = stdin.next().toCharArray();
            System.out.println(bfs(0, n*n-1));
        }
    }

    public static int bfs(int start, int end) {

        LinkedList<Integer> q = new LinkedList<Integer>();
        int[][] dist = new int[n][n];
        for (int i=0; i<n; i++)
            Arrays.fill(dist[i], -1);
        dist[start/n][start%n] = 0;
        q.offer(start);

        while (q.size() > 0) {

            int cur = q.poll();

            if (cur == end ) return dist[ cur/n ][ cur%n ]; // 3 pts


            for (int i=0; i<DX.length; i++) {

                int x = cur/n + DX[i];          // 3 pts

                int y = cur%n + DY[i];          // 3 pts

                if (!inbounds(x,y) || dist[x][y] != -1 || grid[x][y] == 'X')
                    continue;                           // 2 pts

                q.offer( x*n + y );                     // 2 pts

                dist[x][y] = dist[ cur/n ][ cur%n ] + 1;    // 2 pts
            }
        }

        return -1;
    }

    public static boolean inbounds(int tryX, int tryY) {
        return tryX >= 0 && tryX < n && tryY >= 0 && tryY < n;
    }
}
```

7) (15 pts) Greedy Algorithm Design

A couple wants to buy $n$ major items. They will buy one item per month. They will buy all of their items at the CrazySuperStore. Due to the current economy, it is known that at some point in the future there will be a single price hike (on all items at the same time), but due to the uncertainty of economic processes, it is not known when that price hike will occur. For each of the $n$ items the couple wants to buy, both the current price, $a_i$ and the future price, $b_i$ ($b_i > a_i$), are known. (Note that the individual price hikes on items may vary. For example, one item may go from \$100 to \$105 while another might go from \$300 to \$400. They only assumption you may make about the prices is that the second price is strictly higher than the first and that you have access to all before/after prices.) Devise a greedy strategy so that the couple is guaranteed to minimize the amount of money they spend on the items. Assume that no matter what, the couple will buy all items. Clearly describe your strategy and intuitively (or formally) prove why it's optimal.

**For each item, calculate the *difference* between the two prices. So, formally, calculate $difference_i = b_i - a_i$ for each item. Then, sort the items in order from greatest to least, based on $difference_i$. This is the order in which the couple should buy their items.**

**To show why this is optimal, assume that the price hike occurs after month k. The savings the couple gets (compared to buying all items at the higher price), is the sum of the differences of each item they bought in the first k months. Due to how they chose their items, no competing list will have a greater sum of differences, since the couple's list greedily takes the items in order based on this difference. Thus, each month, since they are buying only one item, when choosing which item to buy, they buy the one that saves them the most money. Any different choice would have saved them less money at that point in time, and ultimately could cost them down the line if the price hike occurred.**

**As an example (not necessary to earn full credit), consider these three items:**

**$I_1$ 100, 105**
**$I_2$ 300, 400**
**$I_3$ 1000, 1025**

**So our baseline price for the items at the higher costs is \$105 + \$400 + \$1025 = \$1530. If we only get one month of savings, if we get $I_2$, our cost goes down to \$1430. If we were to choose $I_1$ or $I_3$ in that slot, our costs would only go down to \$1525 and \$1505, respectively and if the price hike occurred, those worse costs would be locked in. Similarly, any choice but $I_2$ followed by $I_3$ would be suboptimal if the price hike occurred after the second month. (In this case, $I_3$ followed by $I_2$ would turn out to work, but without knowing with the price hike will happen, this schedule can't guarantee the optimal performance because it doesn't do so if the price hike occurs after one month.)**

**Grading: 12 pts for correct idea, 6 pts max for any incorrect idea, 3 pts for reasoning (can be given for incorrect algorithm case)**

8) (5 pts) What types of cakes are served at "Nothing Bundt Cakes"? **Bundt Cakes** **(Give to all)**