CS 4040 - HW 6

Ethan Dowalter

1) $A_n = \{9, 6, 5, 3, 1\}$ $\quad C = 11$

Greedy solution: 9, 1, 1 $\quad$ Optimal solution: 6, 5

$$R[j] = \min_{1 \le m \le n} (1 + R[j - a_m])$$

$$R = [\,0 \;\; 1 \;\; 2 \;\; 1 \;\; 2 \;\; 1 \;\; 1 \;\; 2 \;\; 2 \;\; 1 \;\; 2 \;\; 2\,]$$
$$j = 0 \;\; 1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6 \;\; 7 \;\; 8 \;\; 9 \;\; 10 \;\; 11$$

$R[9] = R[6] = R[5] = R[3] = R[1] = 1$

$$R[2] = \min \begin{Bmatrix} 1 + R[2-9] = \infty \\ 1 + R[2-6] = \infty \\ 1 + R[2-5] = \infty \\ 1 + R[2-3] = \infty \\ 1 + R[2-1] = 2 \end{Bmatrix} = 2 \Rightarrow s[2] = 1$$

$$R[4] = \min \begin{Bmatrix} 1 + R[4-9] = \infty \\ 1 + R[4-6] = \infty \\ 1 + R[4-5] = \infty \\ 1 + R[4-3] = 2 \\ 1 + R[4-1] = 2 \end{Bmatrix} = 2 \Rightarrow s[4] = 3$$

$$R[7] = \min \begin{Bmatrix} 1 + R[7-9] = \infty \\ 1 + R[7-6] = 2 \\ 1 + R[7-5] = 3 \\ 1 + R[7-3] = 3 \\ 1 + R[7-1] = 2 \end{Bmatrix} = 2 \Rightarrow s[7] = 6$$

$$R[8] = \min \begin{Bmatrix} 1 + R[8-9] = \infty \\ 1 + R[8-6] = 3 \\ 1 + R[8-5] = 2 \\ 1 + R[8-3] = 2 \\ 1 + R[8-1] = 3 \end{Bmatrix} = 2 \Rightarrow s[8] = 5$$

$$R[10] = \min \begin{Bmatrix} 1 + R[10-9] = 2 \\ 1 + R[10-6] = 3 \\ 1 + R[10-5] = 2 \\ 1 + R[10-3] = 3 \\ 1 + R[10-1] = 2 \end{Bmatrix} = 2 \Rightarrow s[10] = 9$$

$$R[11] = \min \begin{Bmatrix} 1 + R[11-9] = 3 \\ 1 + R[11-6] = 2 \\ 1 + R[11-5] = 2 \\ 1 + R[11-3] = 3 \\ 1 + R[11-1] = 3 \end{Bmatrix} = 2 \Rightarrow s[11] = 6$$

$$s = [\,0 \;\; 1 \;\; 1 \;\; 3 \;\; 3 \;\; 5 \;\; 6 \;\; 6 \;\; 5 \;\; 9 \;\; 9 \;\; 6\,]$$
$$j = 0 \;\; 1 \;\; 2 \;\; 3 \;\; 4 \;\; 5 \;\; 6 \;\; 7 \;\; 8 \;\; 9 \;\; 10 \;\; 11$$

$s[1] = 1 \quad s[3] = 3 \quad s[5] = 5 \quad s[6] = 6 \quad s[9] = 9$

$s[11] = 6$, $\quad 11 - 6 = 5$

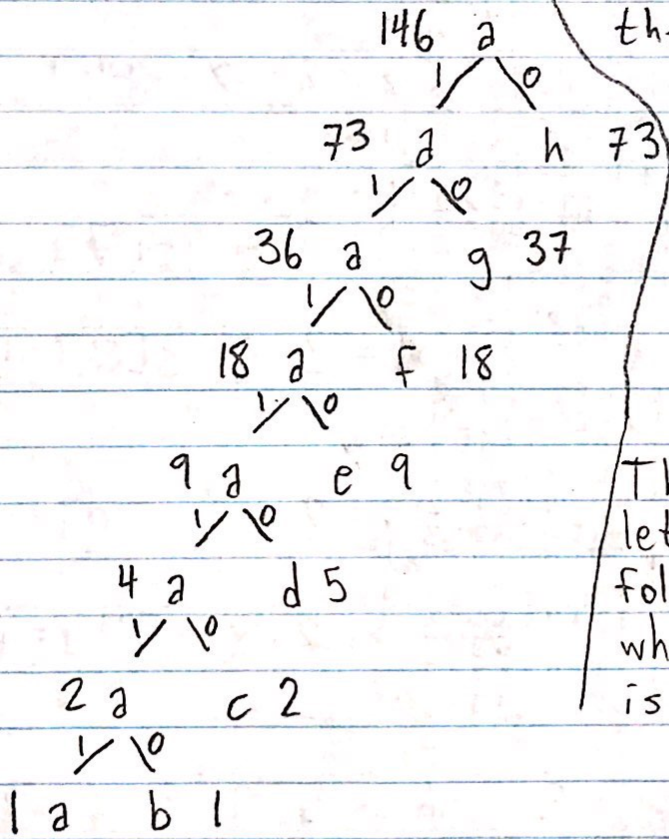
$s[5] = 5$, $\quad 5 - 5 = 0$ ✓ Done

$\therefore$ For $C = 11$, optimal solution is 6, 5

2)

Symbol	Freq	Code
a	1	1111111
b	1	1111110
c	2	111110
d	5	11110
e	9	1110
f	18	110
g	37	10
h	73	0

$\text{Freq} = f(n) = f(n-1) + f(n-2) + 2f(n-3)$



This solution <u>can</u> be generalized to the first n letters. Because the frequncy of the n'th letter is given by this recurrance relation, each successive letter roughly doubles in frequency (also due to initial conditions of $f(1)=1$, $f(2)=1$, $f(3)=2$). Thus, the tree will continue to grow upwards in the same unbalanced fasion and the codes will follow the same pattern of being all 1's with a 0 on the end (except for the letter 'a' which will have n-1 1's)

The code for the i'th letter would be n-i 1's followed by a 0, except when i=1, in which case it is only n-1 1's

Ethan Dowalter

CS 4040 - HW 6 (cont.)

3)
```
FourInARow(pair points[n]){ //array of coord pairs
    sort(points)          //sorts array by y-coord - O(nlogn)
    LineSegList = []      //list of pairs of points which
                          //are start and end points to
                          //line segs with ≥4 points on them
    for (i=0; i<n-1; i++){                          //O(n)
        AngleList = calcAngles(points[i], points) //O(n)
            //calcAngles takes a point and array of
            //points and returns a list of polar
            //angles that are formed from the given
            //point and all the rest
        sort(AngleList)      //O(nlogn)
        for (j=i+1; j<n; j++){      //O(n)
            if (AngleList[j] == AngleList[j+1] and
                AngleList[j] == AngleList[j+2]){
                K = j+2
                while (AngleList[j] == AngleList[K]){
(forgot to move j up by K)    LineSegList.add(points[i], points[K])
            j += K            K++
                } //find sets of at least 3 angles
            }     //that match and add line seg to
        }         //list, while loop is to ensure line
    }             //segs with >4 points are added
    return LineSegList
}
```

This function should run in $O(n\log n + n(n + n\log n + n))$ which is equal to $O(n^2\log n)$

4) A simple cycle in a undirected graph G is a sequence of distinct vertices $V_1, \dots, V_K$ such that $(V_i, V_{i+1 \bmod K})$ is an edge in G. Given any undirected graph G, Find a simple cycle in G such that there are no other simple cycles which contain more vertices.

Decision Problem:
Decide whether a simple cycle of size K exists in a given undirected graph G.

Language = $\{\langle G, K \rangle$: G is a undirected graph with a simple cycle of size k $\}$

CS 4040- HW 6 (cont.)

Ethan Dowalter

5) values = [v_1, v_2, ..., v_n]
weights = [w_1, w_2, ..., w_n]

```
int KnapsacK(n, P){
    K[n+1][P+1] = {0} //initialize to all 0's
    for (i=1 ; i<=n ; i++):
        for (j=1 ; j<=P ; j++):
            if weights[i-1] <= j :
                K[i][j] = max(values[i-1]+K[i-1][j-weights[i-1]],
                              K[i-1][j])
            else:
                K[i][j]= K[i-1][j]
    return K[n][P]      //max value able to be put
                        // into Knapsack
}
```

6) a) Decide whether a value K can be achieved given an array of n items, each with a designated weight and value, and a Knapsack that can hold P units of weight.

b) The dynamic programming solution can be used to solve the decision problem by simply comparing the solution that it returns, call it M for max, with K. If $M \geq K$ then we decide "Yes", otherwise we decide "No", and since this comparison is done in constant time, the time complexity of the decision problem would be $O(nP) + \theta(1) = O(nP)$

c) Yes, it is a polynomial time algorithm because $\forall nP$, $\exists c$ such that $n^c > nP$.

Ethan Dowalter

CS 4040 - HW 6 (cont.)

7) Let us define a certificate, x, as being the vertex indices $\{i_1, i_2, \ldots, i_n\}$. An algorithm can verify GRAPH-ISOMORPHISM by doing the following:

1. Check if the certificate x is a permutation of $\{1, 2, \ldots, n\}$. If not, return false, else continue
2. Permute vertices of G_1 as given by the given permutation.
3. Verify that permuted G_1 is identical to G_2

Step 1 takes at most $O(V^2)$ time
Step 2 takes at most $O(1)$ time
Step 3 takes at most $O(V+E)$ time
Therefore, the algorithm runs in $O(V^2)$ and GRAPH-ISOMORPHISM $\in$ NP

8) To prove this problem is in NP, it suffices to show that given a solution, or certificate, we can verify it in polynomial time. So, suppose we are given an n-vector Y which we know to be a inequality $AY \le c$. In order to check, we multiply A with Y, which takes $O(mn)$ time because A is an $m \times n$ matrix and Y is a $n \times 1$ matrix. Then, you take the resulting $m \times 1$ matrix/vector and compare each element with its corresponding entry in the c vector, which is also $m \times 1$. Thus, this step would take $O(m)$ time for comparing each entry to check if $AY \le c$. Therefore, to check a solution, Y, it takes $O(mn+m) = O(mn)$ which is polynomial time because for any m and n, you can choose a K such that $mn \le n^K$ (given $n > 1$). So this problem is in NP because it can be verified in polynomial time.