Assignment 5

Jiadao Zou --- jxz172230

Q1

> Consider the activity selection problem discussed before but with profits. Activity i $(= 1, 2, ..., n)$ has three values associated with it. s_i the start time, f_i the finish time and p_i the profit. We want to select a subset of nonoverlapping activities whose total profit is maximum. Show how to use dynamic programming to solve this problem.

First, let the $R[i] = \{j : s_j > s_i\} = i, i+1...n$ for the increasing sorting set s_i, then we define an set P, with $P[i]$ is the maximum profit of $R[i]$.

1. Sort the activity set in s_i values to fit the requirement of set R. Complexity: $\Theta(n \lg n)$
2. Also we suppose $N[i] = \min[j : s_j > f_i]$.. Do binary search to find $N[i]$ by comparing each f_i with the s_i set. Complexity: $\Theta(n \cdot \lg n)$
3. Recursion process to update $P[j]$ by $P[i]$ from nth to the beginning.:

$$P[n] = p_n$$
$$P[i] = \max\{p_i + P[N[i]], P[i+1]\}, i < n$$

   Complexity: $\Theta(n)$
4. The overall complexity is $\Theta(n \lg n + n \lg n + n) = \Theta(n \lg n)$

Q2

> Given a string $A[1, 2, ..., n]$ of numbers, find a subsequence $B[1, 2, ..., m]$ with $B[i] < B[i+1]$ for $i = 1, 2, ..., m-1$ such that the value of m is maximum. Show how to use dynamic programming to solve this problem.

- The question is asking us to finding the longest monotonically increasing subsequence of set A and it call the length be m.
- Let $L[i] =$ the length of a longest increasing subsequence which ends in the element $A[i]$. Thus

$$L[1] = 1$$
$$L[j] = \{1 + \max_{i<j:A[i]<A[j]} L[i]\}, j \geq 2$$

```
function LongestIncreasingSubsequence(A):
    n <- A.length
    for j <- 1 to n:
        do L[j] <- 1
        for i <- 1 to j-1
            do if A[i] < A[j] and 1 + L[i] > L[j]
                then L[j] <- 1+L[i]
        return L
```

The complexity is $\Theta(n^2)$.

Q3, P15-6

> Professor Stewart is consulting for the president of a corporation that is planning a company party. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. The personnel office has ranked each employee with a conviviality rating, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.
>
> Professor Stewart is given the tree that describes the structure of the corporation, using the left-child, right-sibling representation described in Section 10.4. Each node of the tree holds, in addition to the pointers, the name of an employee and that employee's conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality ratings of the guests. Analyze the running time of your algorithm.

- If the root r is included in an optimal solution, then dealing with the subproblem rooted at grandchildren of r. (root node is direct supervisor of his children node)
- If the root r is excluded in an optimal solution, then dealing with the subproblem rooted at children of r.
- Algorithms:
  - Make a set C indexed by vertices which tells us the optimal conviviality of a guest list obtained from the subtree with the root at that vertex. Also, create a set G that $G[i]$ tells the guest list we use with root i. Let T be the tree of guests.
    - So, the problem begins with checking the $G[T.root]$ or $G[T.root.grandchildren]$.
    - Look at the leaf L, if the conviviality ranking at L is positive, $G[L] = L$

and $C[L] = L.con$.Otherwise $G[L] = \emptyset$ and $C[L] = 0$. Iteratively solve the subproblems located at parents of nodes at which the subproblem has been solved. In general, for a node x:

$$C[x] = \min(\sum_{y\ is\ a\ child\ of\ x} C[y], \sum_{y\ is\ a\ grandchild\ of\ x} C[y])$$

- Since n is the number of leaves which is the number of subproblems. Solve each problem is in constant time, but the tree traversal to find the appropriate next node takes linear time. The complexity is $O(n^2)$.

Q4, P16.2-2

> Give a dynamic-programming solution to the 0-1 knapsack problem that runs in $O(nW)$ time, where n is the number of items and W is the maximum weight of items that the thief can put in his knapsack.

- Let i be the highest numbered item in an optimal solution S for W pounds and item $1...n$. Then $S' = S - i$ must be an optimal solution for $W - w_i$ pounds and items $1...i-1$, and the value of the solution S is v_i plus the value of the subproblem solution S'.
  $c[i, w]$ is the value of solution for item $1...i$ and maximum weight w.Then

$$c[i,w] = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0, \\ c[i-1,w] & \text{if } w_i > w, \\ \max(v_i + c[i-1, w-w_i], c[i-1,w]) & \text{if } i > 0 \text{ and } w \geq w_i. \end{cases}$$

- The algorithms takes the maximum weight W as the inputs, the number of item n, and two sequences $v = v_1, v_2 \ldots v_n$ and $w = w_1, w_2 \ldots w_n$. It stores the $c[i, j]$ values in $c[0..n, 0..W]$ whose entries are computed in row–major order. So, $c[n, W]$ is the maximum value:

```
function 01KNAPSACK(v,w,n,W)
    let c[0..n, 0..W] be a new array
    for w <- 0 to W
        c[0, w] = 0
    for i <- i to n
        c[i, 0] = 0
        for w <- 1 to W
            if w[i] ≤ w
                if v[i] + c[i-1, w-w[i]] > c[i-1, w]
```

```
                    c[i, w] = v[i] + c[i-1, w-w[i]]
                else c[i, w] = c[i-1, w]
            else c[i, w] = c[i-1, w]
```

- Time complexity:
  - $\Theta(nW)$ to fill in the c table: $(n+1) \cdot (W+1)$ entries, each requiring $\Theta(1)$ to compute.
  - $O(n)$ to tracing from nth row and moves up one each step.

Q5

> This problem is taken from the book by Tardos and Kleinberg: Consider the following problem faced by an advertising company. Billboards may be placed on a highway at any of n possible locations that are known to you along a highway. The highway is M miles long and the locations are at $x_1, x_2, ..., x_n$ miles from one end of the highway. Regulations of the highway department prohibit any two billboards from being 5 miles or less apart. Placing a billboard at location i is worth $r_i > 0$ to the the advertising agency. The input to the problem are the vectors x and r and the value of M and the separation 5. Give a dynamic programming algorithm to find the optimal set of locations to maximize total revenue.

- For each x_i, we either put a bill board or not. If not, the problem reduced one size down. For each x_i, let $p(i) = \max\{j : x_j < x_i - 5\}$. Time complexity is $\Theta(n)$.

```
function FilterBySepearte(A, sep):
    // A index starts from 1 to n
    let n <- A.length
    let BottomArray, TopArray be new arrays
    for i <- 1 to A.length
        BottomArray[i] = A[1] + sep * (i-1)
        TopArray[n-i+1] = A[n] - seq * (i-1)
    for i <- 1 to A.length
        if A[i] ≤ BottomArray[i] or A[i] ≥ TopArray[i]
            drop A[i]
```

- Let the total revenue be P_i at x_i. The recursion is

$$R_0 = 0; \quad R_1 = r_1$$
$$R_i = \max[R_{i-1}, r_i + R_{p(i)}], \ i \geq 2$$

- The time complexity is $O(n)$.