CSC505 HW3

Liam Adams

March 17 2019

Problem 1

a) Let $M(k,d)$ be the max profit for the first k jobs if they have to finish at or before time d. And let the jobs be sorted in order of increasing deadline.

$$M(k,d) = \begin{cases} 0, & \text{if } k > d \text{ or } k = 0 \text{ or } d = 0 \\ max_{1 \le i \le k}\{M(k, d-1), p_i\}, & \text{if } d_i \le d \text{ and } k = 1 \\ max\{M(k-1, d-t_k) + p_k, M(k-1, d)\}, & \text{if } d_k \le d \text{ and } k > 1 \\ M(k, d-1), & otherwise \end{cases}$$

b) The algorithm to fill the table is the following

1. The input is the number of jobs k and the time they need to finish by d.
2. Sort the arrays so the d array is in increasing order.
3. Sum all the times in the t array to find D, the longest time it would take for all jobs to complete serially
4. Change any time in the d array greater than D to D
5. Create a new table m with n rows and D columns, with $n =$ number of jobs and initialize every cell to 0. m will store the max profit values. Also create a table s with n rows and D columns to store a solution for the job sequence.
6. Initialize the first row with the max profit possible for one job at each deadline.
7. Then beginning from the second row, iterate over every cell in the m table, if the deadline for the i^{th} job d_i is less than the deadline time d_j represented by the column, then set the value of $m[i,j]$ to the max between $m[i-1, j-t_i] + p_i$ and $m[i-1, j]$. If the deadline for the i^{th} job is greater than d_j then set $m[i,j] = m[i, j-1]$. If you set $m[i,j] = m[i-1, j-t_i] + p_i$ then set $s[i,j] = i$
8. return $m[k,d]$ and s
9. To recover the solution from s, use another algorithm called $Print-Jobs(s,n,D)$. It will originally be called with the full dimensions and print $s[n,D]$, the index of the job. After printing it recursively calls itself with $s[n-1, D-1]$, if that value of s is the same as the previous value it calls itself with $s[n, D-1]$ until a value changes, and it prints that value. This process repeats until $n == 0$ or $D == 0$.

c) $t = [2,4,3]$, $d = [4,6,10]$, $p = [3,4,2]$. Create a $4x10$ table and change d_2 to 9.

For row 1 we have $\{0, 3, 3, 4, 4, 4, 4, 4, 4\}$
For row 2 we have $\{0, 3, 3, 4, 4, 7, 7, 7, 7\}$
For row 3 we have $\{0, 3, 3, 4, 5, 7, 7, 7, 9\}$

The answer to M(3,9) = 9. Using the algorithm from step 9 we would get jobs [1, 2, 3]

d) The algorithm must iterate over nxD cells. Prior to that it must iterate over the t array of length n to sum the times and over the d array of length n to adjust the deadline times. However the runtime is dominated by filling out the table which is $O(nxD)$. We can write D in terms of n if we assume all the times in t are in the range [1,n]. In this case the sum D is an arithmetic series bounded by n^2. Therefore the runtime of the algorithm is $O(n^3)$

Problem 2

a) Let $s(m, n)$ be a function that returns true or false depending on whether Z_{m+n} is a shuffle of $X = x_1...x_m$ and $Y = y_1...y_n$. Let $k = m + n$

$$s(i,j) = \begin{cases} false, & \text{if } x_i \neq z_k \text{ and } y_j \neq z_k \text{ and } m \neq n \neq 0 \\ true, & \text{if } i = j = 0 \\ s[i-1, j], & \text{if } x_i = z_k \\ s[i, j-1], & \text{if } y_j = z_k \end{cases}$$

b) The algorithm to fill the table is the following

1. Create a new table s with m+1 rows and n+1 columns, with m = size of X and n = size of Y and initialize every cell to false, except s[0,0] which will be true.

2. Let i go from 0..m, j go from 0..n, and k go from 0..m+n. Also let the 0th index be the character before the start of each string, so the first characters of each string are at x[1], y[1], and z[1]. Let k=i+j and s[0,0]=true. Iterate over i and j, if $x[i] = z[k]$ or $y[j] = z[k]$ then set $c[i, j] = true$. Otherwise set $c[i, j] = false$.

c) With X = my, Y = dog and Z = domyg. We have m=2 and n=3. Our table will be 3x4.
Row 0 will be {true, true, true, false}
Row 1 will be {false, false, true, true}
Row 2 will be {false, false, true, true}

The answer will be the bottom right entry of the table

With X = my, Y = dog and Z = dmogy we have the same dimensions as above.
Row 0 will be {true, true, false, true}
Row 1 will be {false, true, true, true}
Row 2 will be {false, false, true, true}

The answer will be the bottom right entry of the table

d) Filling the table is the dominant term in the runtime of the algorithm. There are (m+1)x(n+1) cells, so the runtime is $O(mn)$.

Problem 3

a) y(k,d) returns the location of the cell tower closest to the k^{th} house. The houses are all on a line, and the x array stores their positions sorted in increasing distance from the origin of the line.

$$y(k,d) = \begin{cases} y(k-1,d), & \text{if } x_i - y(k-1,d) \leq d \\ x[k] + d, & \text{if } x_i - y(k-1,d) > d \text{ or } k = 0 \end{cases}$$

b) The greedy choice is always placing the cell tower d distance away from a house that does not have a cell tower close enough to it. This is the locally optimum solution for that particular house.
To prove the greedy choice, consider any non empty subproblem S_k and let x_m be a house without a cell tower within d units of it. Then $y_m = x_m + d$ will be included in some minimum sized subset of cell towers such that each house has a tower within d units of it.
Let Y_k be a minimum sized subset of cell towers in S_k, and let y_j be the cell tower closest to x_m. If $y_j = y_m$ then we have proven the greedy choice. If $y_j \neq y_m$, let the set $Y_k' = Y_k - \{y_j\} \cup \{y_m\}$ be Y_k but substituting y_m for y_j. All the houses in Y_k' have a cell tower at most d units away, which follows from Y_k satisfying that and $x_m + d = y_m$. Therefore $y_j \leq y_m$ and since $|Y_k'| = |Y_k|$ we can conclude that Y_k' is a minimum sized subset of cell towers, and it includes y_m

c) A greedy algorithm is described below

1. Let $n = x.length$, y be a new array of size $n-1$, and $num\_towers = 1$. Set $y[1] = x[1] + d$ and $closest\_tower = y[1]$. This is the initial greedy choice.

2. For $i = 2...n$, if $x[i] - closest\_tower > d$ then create a new tower $y[num\_towers{+}{+}] = x[i] + d$. Also set $closest\_tower$ to the tower just created. This loop is making the greedy choice whenever we encounter a house that does not have a cell tower close enough to it.

3. When the for loop completes y will contain a minimum set of cell towers.

d) Let $x = [0, 10, 15, 26]$ and $d = 5$. First set $y[1] = 5$, this will satisfy the constraint until $x[3]$ at which point we set $y[2] = 20$. This will not be close enough to $x[4]$ so we create one more $y[3] = 31$. So a minimum set of cell towers is $y = [5, 20, 31]$.

e) There is only one loop in the algorithm which goes from 2...n. Therefore the runtime is $O(n)$