Cameron Palmer - CSCE 5150
November 1, 2007

Homework 4

Page 364 15-1

Intuition - Scanning from left to right you connect each point to its closest neighbor. As the scan of the graph occurs you pick up a new point and connect it to the vertex closest to the point that is already in the bitonic tour.

If Z represents the cost of the shortest bitonic path from i, through all verticies, and $1 \leq i \leq n$, $j > i$ then

$$Z_{i,j} = d(i, i+1) + Z_{i+1,j} \text{ where } j \neq i+1$$

which represents the addition of a new point to the tour, and

$$Z_{i,i+1} = \min_{k>i+1} d(i,k) + Z_{i+1,k}$$

It takes O(i) to compute $Z_{i,j}$. So time complexity is $\Theta(\sum_{i=1}^{n} i)$ which is $\Theta(n^2)$.

Page 367 15-4

For any $x \in T$. Let W_x be the value of an optimal solution if we force x in the the solution. Let W'_x be the value of the optimal solution for subtree rooted at x if we force x out of the solution. Let O_x be a value of an optimal solution for a subtree rooted at x. We want to compute O.

If x is not a leaf,

$$O_x = \max(W_x, W'_x) \tag{1}$$

Let $C(x)$ be the children of x,

$$W_x = w(x) + \sum_{y \in C(x)} W'_y \tag{2}$$

$$W'_x = \sum_{y \in C(x)} O_y \tag{3}$$

If x is a leaf,

$$W_x = W(x)$$

$$W'_x = O$$

The algorithm is post order in which recurrence relations (1), (2), (3) are used to compute O_x, W_x, W'_x for all $x \in V$. Time complexity is O(n) where $n = |v|$.

Page 369 15-7

You could order the jobs a by deadline d. From here you place the maximum profit p in the matrix and optimize your path through the jobs. Which would yield O(n^3) running time based upon the size of the table n^2 and then the recovery of the optimal path n.

In table T,

$$T[i,t] = \max \begin{cases} T[i-1,t] & \\ T[i-1,t-t_i] + p_i & \text{if } t \leq d_i \\ T[i-1,t-t_i] & \text{if } t > d_i \end{cases}$$

Problem 4a

Using Floyd-Warshall as a basis simply recursively follow the backpointers for all the shortest paths that had the same weight.

```
sub count(i, j)
if edge(i, j) = d(i, j) then
  return 1
else
  for edge(i, k): where d(i, k) + d(k, j) = d(i, j) do
    sum ← sum + count(i, k)
  end for
  return sum
end if
```

Problem 4b

Since a fixed edge means that there is some edge e(i, j) we can easily lookup the j-th row and determine how many backpointers refer back to that cell. That can be lookup in linear time.