

## CSE 361 Complexity of Sequential and Parallel Algorithms. Fall 2005

### Homework #2 Solutions

Instructor: Sanguthevar Rajasekaran

---

- Arrange the objects in the nonincreasing order of  $p_i/w_i$ 's: (1, 7, 5, 3, 6, 2, 4). Optimal solution =  $12 + 6 + 8 + 5 + 12 + 24/5 = 47.8$
  - Consider two objects:  $p_1 = 2, w_1 = w$  and  $p_2 = 1, w_2 = 1$  and  $m = 1$ . Let  $w > 2$ .  $F^*(I) = 1$  and  $F(I) = 2/w$ . As  $w \rightarrow \infty$ ,  $F^*(I)/F(I) \rightarrow \infty$ .
  - Consider the objects:  $p_1 = 1, w_1 = 1$  and  $p_2 = p, w_2 = 2$  and  $m = 1$ . Let  $p > 2$ .  $F^*(I) = p/2$  and  $F(I) = 1$ . As  $p \rightarrow \infty$ ,  $F^*(I)/F(I) \rightarrow \infty$ .
- Let  $C$  be a cycle with  $k$  vertices. Let the edge  $e$  of  $C$  with the maximum weight be a part of a minimum-cost spanning tree of  $G$ . There exists atleast one edge,  $e'$ , of  $C$  which is not a part of the minimum-cost spanning tree ( $k$  vertices can be connected by  $k - 1$  edges and  $C$  contains  $k$  edges). By replacing  $e$  in the spanning tree with  $e'$ , we can obtain a new spanning tree whose cost will be less than that of the original minimum-cost spanning tree. This is a contradiction and hence, the edge with the maximum weight of  $C$  can not be part of a minimum-cost spanning tree of  $G$ .
- Let  $v$  be the source and let the graph  $G$  be represented using an adjacency list. Create a new data structure to represent the *dist* array (refer to Dijkstra's algorithm in the text for details on *dist* array), as given below:

Create an array  $D[1..C * n]$  ( $C$  is the maximum weight of an edge in  $G$  and  $n$  is the number of vertices in the graph). Each element,  $D[i]$ , of the array is a linked list of vertices whose *dist*[ ] value is  $i$ . Maintain pointers from the head nodes in the adjacency list of the graph to the corresponding nodes (referring to the same vertex) in  $D$ . Also, in  $D$ , each non-null element maintains a pointer to the next non-null element. (i.e., if  $D[i]$  has a pointer to  $D[k]$  then that means  $D[i] \neq Null$ ,  $D[k] \neq Null$ , and  $D[i + 1] = D[i + 2] = \dots = D[k - 1] = Null$ ).

- Construct  $D$  from  $G$ .
- Apply Dijkstra's algorithm  
Repeat until  $D$  is empty
  - Find the vertex whose *dist*[ ] value is minimum. This involves looking at the first non-null element,  $D[i]$ , in  $D$ . If  $D[i]$  has more than one element, just select one out of them. Delete this element from  $D$ .
  - For the element selected, access all its adjacent vertices from the adjacency list and update their *dist*[ ] values if need be. This can be done in constant time per neighbor by following the pointers from the head nodes in the adjacency list to the

corresponding vertex nodes in  $D$ . In  $D$ , move the nodes whose  $dist[ ]$  values got modified to their proper places. Adjust the pointers (the pointers from the adjacency list to the elements in  $D$  and the non-null pointers in  $D$ ).

Complexity of (a) =  $O(|V| + |E|)$

Complexity of (b) =  $O(|V| + |E|)$

Total complexity =  $O(|V| + |E|)$

4. Refer to Algorithm *AllPaths* in the text.

Maintain arrays,  $D^k(i, j), 1 \leq k \leq n$ .

Add the following after line 10 in *AllPaths*.

If  $(A[i, j] < A[i, k] + A[k, j])$  (i.e.  $A^{k-1}(i, j) < A^{k-1}(i, k) + A^{k-1}(k, j)$ ) then  $D^k(i, j) = 0$  else  $D^k(i, j) = 1$ .

The path for each pair of vertices can be traced after *AllPaths* is run. For each pair  $(i, j)$ , if  $D^k(i, j) = 1$  then  $k$  is in the path from  $i$  to  $j$ . Obtain the complete path by applying the same procedure at  $D^{k-1}(i, k)$  and  $D^{k-1}(k, j)$ . If  $D^k(i, j) = 0$  then  $k$  is not in the path and the path is same as  $D^{k-1}(i, j)$ .

Time Complexity =  $O(n^3)$

Space Complexity =  $O(n^3)$

5. Let  $X = x_1, x_2, x_3, \dots, x_n$  and  $Y = y_1, y_2, y_3, \dots, y_m$ .

Let  $L(i, j)$  represent the length of the longest common subsequence between  $x_1, x_2, \dots, x_i$  and  $y_1, y_2, \dots, y_j$  such that the common subsequence ends at  $x_i$  and  $y_j$  in  $X$  and  $Y$  respectively.

Assuming that  $L[i-1, j-1]$  has already been calculated, compute  $L[i, j]$  as follows:

if  $x[i] = y[j]$  then  $L[i, j] = L[i-1, j-1] + 1$

else  $L[i, j] = 0$ .

Compute  $L[i, j]$  for  $0 \leq i \leq n; 0 \leq j \leq m$ . Scan through all these values and pick the largest  $L[i, j]$ . Computing  $L[i, j]$  from  $L[i-1, j-1]$  takes constant time. Thus the total run time is  $O(nm)$ .

6. (a)  $C(i, j) = \min_{i \leq k \leq j} \{C(i, k) + C(k+1, j) + D(i-1)D(k)D(j)\}$

(b) for  $i = 1$  to  $r$  do

    for  $j = i+1$  to  $r$  do

        for  $k = i$  to  $j$  do

$C[i, j] = \min(C[i, j], C[i, k] + C[k+1, j] + D(i-1)D(k)D(j))$

7. We can verify all the properties that a flow function is required to satisfy. Let  $F = \alpha f_1 + (1 - \alpha) f_2$ . Then, i) **Capacity constraint:** For all  $u, v \in V$  we have:  $F(u, v) = \alpha f_1(u, v) + (1 - \alpha) f_2(u, v) \leq \alpha c(u, v) + (1 - \alpha) c(u, v) \leq c(u, v)$ ; ii) **Skew symmetry:** For all  $u, v \in V$ ,

$F(u, v) = \alpha f_1(u, v) + (1 - \alpha) f_2(u, v) = -\alpha f_1(v, u) - (1 - \alpha) f_2(v, u) = -F(v, u)$ ; iii) **Flow conservation:** For all  $u \in V - \{s, t\}$ ,  $\sum_{v \in V} F(u, v) = \sum_{v \in V} \alpha f_1(u, v) + (1 - \alpha) f_2(u, v) = \alpha \sum_{v \in V} f_1(u, v) + (1 - \alpha) \sum_{v \in V} f_2(u, v) = 0$ .

8. Going through the steps of Ford-Fulkerson method we realize that the value of maximum flow is 19.