

## CPSC 311-501 Homework 4

- **Prob. 15-7**

Let  $A$  be a set of  $n$  jobs,  $\{a_1, \dots, a_n\}$ , that are sorted by their deadlines such that  $d_1 \leq d_2 \leq \dots \leq d_n$ .

Let  $P[i, j]$  be the maximum amount of profit obtained by a subset of  $\{a_1 \dots a_i\}$  by time  $j$ ,  $1 \leq i \leq n$  and  $0 \leq j \leq d_n$ .

MAXIMUM-PROFIT

```

1  for i ← 1 to n //initialize the maximum profit at time 0 to be zero for each job
2  P[i, 0] = 0
3  for j ← 1 to d_n //determine the maximum profit obtained by {a_1} at each time step
4    if ((j == t_1) and (j <= d_1))
5      P[1, j] = p_1
6    else
7      P[1, j] = P[1, j-1]
8  for i ← 2 to n
9    for j ← 1 to d_i
10   if (j > d_i) // case (1): the current time step j already exceeds a_i's deadline
11     then P[i, j] = P[i-1, j]
12   else if (j < t_i) // case (2): there is not enough time to finish job a_i by the current time j
13     then P[i, j] = P[i-1, j]
14   else // case (3): time j does not exceed a_i's deadline, and there is enough time to finish job a_i by time j
15     then P[i, j] = max(P[i-1, j], P[i-1, j-t_i] + p_i)

```

The running time of MAXIMUM-PROFIT is  $O(n) + O(d_n) + O(nd_n) = O(nd_n)$ .

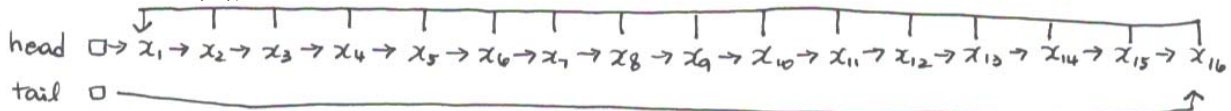
- **Ex. 21.1-3**

For each edge, FIND-SET is called for its two incident vertices. Hence, FIND-SET is called  $2|E|$  times.

For each connected component  $S_i$ ,  $1 \leq i \leq k$ , UNION is called  $|S_i| - 1$  times. Thus, UNION is called  $\sum_{i=1}^k (|S_i| - 1) = |V| - k$  times.

- **Ex. 21.2-2**

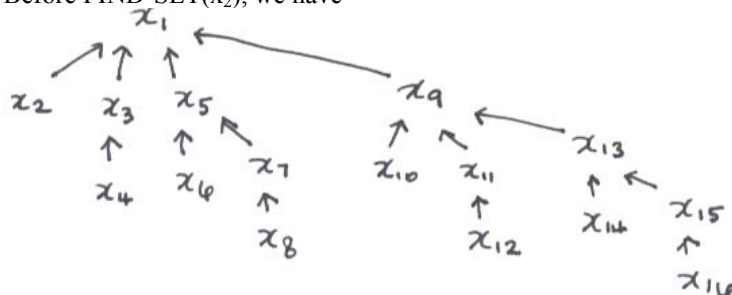
Before FIND-SET( $x_2$ ), we have



FIND-SET( $x_2$ ) returns  $x_1$ . FIND-SET( $x_9$ ) returns  $x_1$ . FIND-SET( $x_2$ ) and FIND-SET( $x_9$ ) do not change the linked list data structure.

- **Ex. 21.3-1**

Before FIND-SET( $x_2$ ), we have



FIND-SET( $x_2$ ) and FIND-SET( $x_9$ ) return  $x_1$ . FIND-SET( $x_2$ ) and FIND-SET( $x_9$ ) do not change the tree.

- **Prob. 21-1**

- (a)

4 3 2 6 8 1

- (b)

When  $i = 1$ , there are two cases to consider:

- (i)  $j \neq m+1$ :

$i$  was inserted during the insertion sequence  $I_j$ , and  $i=1$  is clearly the minimum during the  $j$ -th EXTRACT-MIN call.  $i$  is correctly placed in  $extracted[j]$ .  $K_j$  is then merged with  $K_l$ , and  $K_j$  is destroyed—this effectively ensures that if 2 were in  $I_j$ , it would be placed in  $extracted[l]$  as it would be extracted during the  $l$ -th EXTRACT-MIN call.

- (ii)  $j = m+1$ :

There is no  $(m+1)$ -th EXTRACT-MIN call, and the *extracted* array does not contain 1.

Similarly, we consider two cases for  $i > 1$ . At each  $i > 1$ , if  $j \neq m+1$ ,  $i$  is placed in  $extracted[j]$  as  $i$  would be extracted during the  $j$ -th EXTRACT-MIN call. If  $j = m+1$ , the *extracted* array does not contain 1.

At the end of the for loop, the array *extracted* is correct.

- (c)

We implement OFF-LINE-MINIMUM with disjoint-set forests with union by rank and with path compression. Let  $DSF_j$  represent the set containing the keys in  $I_j$ . Let  $DSF_j$  point to  $K_j$  and vice versa. We change lines 2 and 6.

- Line 2: do FIND-SET( $i$ )

- Line 6: call UNION( $j, l$ ). Make the resulting disjoint-set forest and  $K_l$  point to each other. Delete  $K_j$ .

Assuming that finding  $l$  takes a constant amount of time, OFF-LINE-MINIMUM takes  $O(2n \alpha(n) + n) = O(n \alpha(n))$  time.

- **Ex. 22.1-3**

- *adjacency-list*

LIST-TRANSPOSE // Given an array Adj of  $|V|$  lists, compute  $Adj^T$

```

1  for each  $u \in V$ 
2     $Adj^T[u] =$  empty list
3  for each  $u \in V$ 
4    for each  $v \in Adj[u]$ 
5      insert  $u$  into  $Adj^T[v]$ 
6  return  $Adj^T$ 
```

LIST-TRANSPOSE takes  $O(V+E)$  time.

- *adjacency matrix*

MATRIX-TRANSPOSE // Given an adjacency-matrix  $A$ , compute  $A^T$

```

1  for ( $i = 1; i \leq |V|; i++$ )
2    for ( $j = 1; j \leq |V|; j++$ )
3       $A^T[i, j] = A[j, i]$ 
4  return  $A^T$ 
```

MATRIX-TRANSPOSE takes  $O(V^2)$  time.

- **Ex. 22.2-3**

The overhead for initialization is  $O(V)$ . Each vertex is enqueued and dequeued at most once; scanning for adjacent vertices of each dequeued vertex takes  $O(V)$  time. Thus, the total time is  $O(V + VV) = O(V^2)$ .

- **Ex. 22.3-4**

- (a)

$\Rightarrow$  : Assume that  $(u, v)$  is a tree edge or forward edge.  $v$  is then a descendant of  $u$  in the depth-first forest of  $G$ . By Corollary 22.8,  $d[u] < d[v] < f[v] < f[u]$ .

$\Leftarrow$  : Assume that  $d[u] < d[v] < f[v] < f[u]$ . By Corollary 22.8,  $v$  is a descendant of  $u$ , and thus  $(u, v)$  is a tree edge or forward edge.

▪ (b)

$\Rightarrow$  : Assume that  $(u, v)$  is a back edge.  $u$  is then a descendant of  $v$ . By Corollary 22.8,  $d[v] < d[u] < f[u] < f[v]$ .

$\Leftarrow$  : Assume that  $d[v] < d[u] < f[u] < f[v]$ . By Corollary 22.8,  $u$  is a descendant of  $v$ .  $v$  is then an ancestor of  $u$ , and  $(u, v)$  is a back edge.

▪ (c)

$\Rightarrow$  : Assume that  $(u, v)$  is a cross edge. Neither  $u$  nor  $v$  is an ancestor of each other, and  $v$  is not a descendant of  $u$ . By Theorem 22.7, the intervals  $[d[u], f[u]]$  and  $[d[v], f[v]]$  are entirely disjoint. By Theorem 22.9 and Corollary 22.8,  $d[u]$  is not less than  $d[v]$ . Thus,  $d[v] < f[v] < d[u] < f[u]$ .

$\Leftarrow$  : Assume that  $d[v] < f[v] < d[u] < f[u]$ . By Theorem 22.7, neither  $u$  nor  $v$  is a descendant of the other, and  $(u, v)$  is then a cross edge.

• **Ex. 22.4-3**

DETECT-CYCLE( $G$ )

- 1 found-cycle = false
- 2 Call DFS( $G$ ) with a modification to DFS-VISIT( $u$ ) such that if color [ $v$ ] is not WHITE in line 5 (p. 541), return to DFS and set found-cycle to be true.
- 3 In DFS, as each vertex is finished, stop and return if found-cycle is true.
- 4 return found-cycle

When  $G$  is acyclic,  $|E| \leq |V| - 1$  by Theorem B.2 and the running time is  $O(V + V - 1) = O(V)$ . When we find a cycle in  $G$ , the running time is  $O(V + V) = O(V)$ . Thus, our algorithm runs in  $O(V)$  time.