Homework 5 Solutions
Fundamental Algorithms, Fall 2004, Professor Yap

Due: Thu December 9, in class
SOLUTION PREPARED BY Instructor and T.A.s Ariel Cohen and Vikram Sharma

INSTRUCTIONS:

- NOTICE: In the last homework, some students copied programs from each other. This is *not* acceptable for any part of your homework. While group discussions are encouraged, each submitted homework *must* represent your individual work. It must *not* be copied from anywhere, including group members.

---

1. (10+15+10 Points) MST
You must know how to do hand-simulations of the algorithms of Prim and Kruskal.
(a) Please carry out the hand simulations of these two algorithms on the graph of Figure 7 (Lecture V). The edge weights are described in Exercise 5.1.
(b) Let $S \subseteq E$ be a set of edges, and S is acyclic (i.e., S is a forest). To implement Kruskal's algorithm, we need to test whether a given edge e will create a cycle when added to S. We want you to implement this test by using a simple data structure, denoted $D(S)$: it is essentially a list of linked lists. Each linked list of $D(S)$ corresponds to the set of nodes in a tree of S. Assume that the vertex set is $V = \{1, 2, \ldots, n\}$. If $i \in V$, let $L(i)$ denote the linked list corresponding to i. Show how to implement the following queries on $D(S)$ efficiently:

   - EQUAL(i, j): Given $i, j \in V$, is $L(i) = L(j)$?
   - MERGE(i, j): Given $i, j \in V$, merge the list of $L(i)$ and $L(j)$. After merging, we have $L(i) = L(j)$.

   NOTE: you may introduce auxilliary data structures as needed.
   (c) Implement Kruskal's algorithm using $D(S)$, and give a complexity analysis of your implementation.
   SOLUTION:

   (a) The simulations in these two solutions are important to master! Please make sure you understand how to do this.

   Prim's Algorithm. Let the nodes be $V = \{1, 2, \ldots, 11, 12\}$. We must have a starting node, which we may pick to be node 1. Let us organize the array $d[i]$ ($i \in V$) as rows of a $n \times (n+1)$ matrix, where the original row is initially $d[1] = 0$ and $d[i] = \infty$ for $i > 1$.

   Let $U \in V$ be the current set of picked nodes. At the beginning of state 1, $U = \{1\}$. At stage $i \geq 1$, suppose we pick a new node $v_i \in V \setminus U$ where $d[v_i] = \{d[j] : j \in V \setminus U\}$. Then we update all the values of $d[u]$ for all $u \in V \setminus U$ that is adjacent to v_i. The update rule is this:

$$d[u] = \min\{d[u], COST(v_i, u)\}.$$

   The resulting array is written as row $i + 1$ in our matrix:

Stage	1	2	3	4	5	6	7	8	9	10	11	12
0	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	X	3	1	7	∞	∞	∞	∞	∞	∞	∞	∞
2			X	6				3				
3		X			6							
4								X	8			
5				X		7						
6					X		6					6
7						3	X			3	2	
8						1				1	X	2
9						X						
10									6	X		
11												X
12									X			

We mark the newly picked node in each stage with an 'X'. Also, any value that is unchanged from the previous row may be left blank. Thus, in stage 2, the node 3 is picked and we update $d[4]$ using $d[4] = \min\{d[4], COST[3,4]\} = \min\{7,6\} = 6$.

The final cost of the MST is 37. To see this, each X corresponds to a vertex v that was picked, and the last value of $d[v]$ contributes to the cost of the MST. E.g., the X corresponding to vertex 1 has cost 0, the X corresponding to vertex 2 has cost 3, etc. Summing up over all X's, we get 37.

Kruskal's Algorithm: The edges in sorted order are shown in the table below. Using this order, we now consider each edge in turn: edge 1−3, edge 6−11, edge 10−11, etc.

We maintain a partition of $V = \{1, \ldots, n\}$ into disjoint sets. Let $L(i)$ denote the set containing vertex i. Initially, each node is in its own set, i.e., $L(i) = \{i\}$. Whenever an edge $i{-}j$ is added to the MST, we merge the corresponding sets $L(i) \cup L(j)$. E.g., in the first step, we add edge 1−3. Thus the lists $L(1)$ and $L(3)$ are merged, and we get $L(1) = L(3) = \{1,3\}$. To show the computation of Kruskal's algorithm, for each edge, if the edge is "rejected", we mark it with an "X". Otherwise, we indicate the merged list resulting from the union of $L(i)$ and $L(j)$:

sorting order	edge	weight	merged list
1	1-3:	1	$\{1,3\}$
2	6-11:	1	$\{6,11\}$
3	10-11:	1	$\{6,10,11\}$
4	6-10:	2	X
5	7-11:	2	$\{6,7,10,11\}$
6	11-12:	2	$\{6,7,10,11,12\}$
7	1-2:	3	$\{1,2,3\}$
8	3-8:	3	$\{1,2,3,8\}$
9	6-7:	3	X
10	7-10:	3	X
11	2-5:	6	$\{1,2,3,5,8\}$
12	3-4:	6	$\{1,2,3,4,5,8\}$
13	5-7:	6	$\{1,2,3,4,5,6,7,8,10,11,12\}$
14	5-12:	6	X
15	9-10:	6	$\{1,2,3,4,5,6,7,8,9,10,11,12\}$
16	1-4:	7	X
17	4-6:	7	X
18	8-9:	8	X
19	4-5:	10	X
20	4-9:	11	X

REMARK: if we kept track of the number of edges added to the MST, we could stop after this number is equal to $n-1$. In particular, we could have stopped after stage 15, when we added the 11th edge.

(b) We have asked you to represent each set $L(i)$ as a linked list. In order to implement the two queries in this question, called EQUAL(i,j) and MERGE(i,j), we need to maintain some other information. In particular, let us assume an array $FIND[1..n]$ where $FIND[i]$ tells us which list vertex i belows to. There are two ways to do this, and we will explore both solutions to see their tradeoffs:

SOLUTION (A): Let $FIND[i]$ point to the head of the linked list representing $L(i)$. In this case, we can implement the EQUAL(i,j) quite easily: EQUAL(i,j) returns true iff $FIND[i] = FIND[j]$. This takes $O(1)$ time. To implement $MERGE(i,j)$, we must first concatenate the two lists $L(i)$ and $L(j)$ – this is easy, and takes $O(1)$ time, assuming we also keep a pointer to the end of each list. To be specific about this concatenation, let us suppose $L(j)$ is appended to the end of list $L(i)$. However, we must also update the array $FIND$. This means we need to go to each vertex k in list $L(j)$ and update $FIND[k]$ to point to the beginning of list $L(i)$. To allow this, we may assume that each node in $L(i)$

stores the index of the node that it represents. In any case, this process takes $\Theta(n)$ time in the worst case.

SOLUTION (B): Let $FIND[i]$ point to the node that represents i in the linked list $L(i)$. To implement EQUAL(i,j), we need to go to the node $FIND[i]$, and follow the links until we get to the last node of the linked list. We do the same for node $FIND[j]$. If both of them end up with the same node, we return true, else return false. This takes time $\Theta(n)$ in the worst case. To implement UNION(i,j), we just have merge the two linked list as in SOLUTION (A). But unlike SOLUTION (A), we need not modify the $FIND$ array. So UNION takes $O(1)$ time.

REMARK: It is important to understand the above solution because this is the beginning of a series of development that lead to the ultimate and optimal "UNION-FIND" algorithms.

2. (15+15 Points) Convex Hull
Let $S_n = (v_1, \ldots, v_n)$ be an input sequence of points in the plane. Assume that the points are sorted by x-coordinates and satisfy $v_1 <_x v_2 <_x \cdots <_x v_n$. Note that "$a <_x b$" means that $a.x < b.x$ where a, b are points. Our goal is to compute the upper hull of S_n. In stage i ($i = 1, \ldots, n$), we have processed the sequence S_i comprising the first i points in S_n. Let H_i be the upper hull of S_i. The vertices of H_i are stored in a push-down stack data structure, D. Initially, D contain just the point v_1.
(a) Describe a subroutine $Update(v_{i+1})$ which modifies D so that it next represents the upper hull H_{i+1} upon the addition of the new point v_{i+1}. HINT: Assume D contains the sequence of points $(u_1, \ldots, u_h)$ where $h \geq 1$ and u_1 is at the top of stack, with $u_1 >_x u_2 >_x \cdots >_x u_h$. For any point p, let $LT(p)$ denote the predicate $\mathtt{LeftTurn}(p, u_1, u_2)$. If $h = 1$, $LT(p)$ is defined to be true. Implement $Update(v_{i+1})$ using the predicate $LT(p)$ and the (ordinary) operations of push and pop of D.
(b) Using part (a), describe an algorithm for computing the convex hull of a set of n points. Analyze the complexity of your algorithm.

SOLUTION:

(a) The update routine is a simple while loop:

```
UPDATE(p, D)
    Input: p is the newly inserted point, D contains current upper hull.
        Assume p.x > q.x for all q ∈ D.
    Output: Void. But D is updated to contain the new upper hull.
    0. Assert(D is non-empty).
    1. While LT(p) = false
        D.pop().
    2. D.push(p).
```

(b) Here is the algorithm to compute the convex hull of a set of points:

```
CONVEX HULL
    1. Sort the input points by their x-coordinates.
        Assume the sorted points are v_1, ..., v_n, with v_1 the leftmost point, n ≥ 1.
    2. Initialize push-down stacks D_a and D_b, representing the upper and lower hulls, respectively.
    3. D_a.push(v_1) and D_b.push(v_1).
    4. For i = 2, ..., n do
        Update(p, D_a) and Update(p, D_b).
    5. "Merge" the lists represented by D_a, D_b into a listing of the convex hull (details omitted).
```

Complexity Analysis: the sorting takes $O(n \log n)$ time. There are $O(n)$ pushes and pops, to the stack operations. Hence the overall complexity is $O(n \log n)$ time.

3. (5+15 Points) Splay Analysis
In this question, we define the potential of node u to be $\Phi(u) = \lg(\textsc{Size}(u))$, instead of $\Phi(u) =$

$\lfloor \lg(\text{SIZE}(u)) \rfloor$.

(a) How does this modification affect the validity of our Key Lemma about how to charge `splayStep`? HINT: In fact, we now have $\Phi'(u) - \Phi(u)$ is *strictly positive*. This appears to make our proof easier, but what could go wrong?

(b) Cosider Case I in the proof of the Key Lemma. Show that if $\Phi'(u) - \Phi(u) \le \lg(6/5)$ then $\Delta\Phi = \Phi'(w, v) - \Phi(u, v) \le -\lg(6/5)$. HINT: the hypothesis implies $a + b \ge 9 + 5c + 5d$.

SOLUTION:

(a) In our original proof, we had 2 cases: $\Phi'(u) - \Phi(u)$ is 0 or positive. But now, we have $\Phi'(u) - \Phi(u)$ is always positive. But just because this quantity is positive does not mean that it is "large enough" to pay for the cost of the SplayStep. We do not care how positive it must be, but is MUST be larger than some FIXED constant $C > 0$. Hence, we STILL have two cases to consider: when $\Phi'(u) - \Phi(u) \ge C$ and when $\Phi'(u) - \Phi(u) < C$. In the second part of this question, we choose $C = \lg(6/5)$.

(b) Since $\Phi'(u) - \Phi(u) = \lg((3+a+b+c+d)(1+a+b)) \le \lg(6/5)$, we get $5(3+a+b+c+d) \le 6(1+a+b)$, i.e,

$$9 + 5(c + d) \le a + b. \tag{1}$$

We need to show that $\Phi'(v, w) - \Phi(u, v) \le \lg(5/6)$. I.e., $\lg((2 + b + c + d)(1 + c + d)) - \lg((1 + a + b)(2 + a + b + c)) \le \lg(5/6)$, i.e.,

$$6(2 + b + c + d)(1 + c + d) \le 5(1 + a + b)(2 + a + b + c).$$

This last inequality follows from the following:

$$\begin{array}{rcll} 6(2 + b + c + d)(1 + c + d) & = & [3(1 + c + d)][2(2 + b + c + d)] & \\ & \le & [1 + a + b][2(2 + b + c + d)] & \text{(By (1), } 3(1 + c + d) \le 1 + a + b) \\ & \le & [1 + a + b][3(2 + a + b + c)] & \text{(By (1), } 2(b + c + d) \le 2b + (a + b)) \\ & = & 3(1 + a + b)(2 + a + b + c). & \end{array}$$

REMARK: It is clear that this inequality is rather lose. Case II can similarly be proved.

4. (15+10+10 Points)

(a) Compute the length of the longest common subsequence, the edit distance and the alignment distance of the following pair of strings: $X =$ `agacgttcgtta` and $Y =$ `cgactgctgt`. These are the functions $L(X, Y), D(X, Y), A(X, Y)$. You must organize your 3 computations in the form of matrices, as in the Lecture Notes.

(b) Suppose you want to compute $L(X, Y, Z)$, the length of the longest common subsequence for three strings X, Y, Z. State and prove the dynamic programming principle analogous equation (2) in Lecture VII.

(c) Describe an algorithm for $L(X, Y, Z)$ in pseudo code. Analyze its complexity as a function of $m = |X|, n = |Y|, p = |Z|$.

SOLUTION:

(a) OMITTED.

(b) Assume $m = |X|, n = |Y|, p = |Z|$.

$$L(X, Y, Z) = \begin{cases} 0 & \text{if } mnp = 0 \\ 1 + L(X', Y', Z') & \text{if } x_m = y_n = z_p \\ \max\{L(X, Y, Z'), L(X, Y', Z), L(X', Y, Z)\} & \text{else.} \end{cases}$$

Let us prove that the 3 cases in this formula are correct.

The base case $mnp = 0$ is clearly correct.

If $x_m = y_n = z_p$, and if $W \in LCS(X, Y, Z)$, then the formula is verified if we can show that the last character of W is x_m. Suppose not. In that case, $|W| = L(X', Y', Z')$, but surely the string Wx_m is a longer common subsequence of X, Y, Z. This is a contradiction.

The last case is by exhaustive analysis: if $W \in LCS(X, Y, Z)$, and the last character of W is w then w must be different from x_m, y_n or z_p. This means $W \in LCS(X', Y, Z)$ or $W \in LCS(X, Y', Z)$ or $W \in LCS(X, Y, Z/)$. This proves our formula.

(c)

```
L(X,Y,Z):
    ▷ Initialization
    1. Let m = |X|, n = |Y| and p = |Z|.
          Create a matrix M[0..m, 0..n, 0..p].
    2. For i = m downto 1 do
          For j = n downto 1 do
               For k = p downto 1 do
                    M[0, j, k] = M[i, 0, k] = M[i, j, 0] = 0.
    ▷ Main Loop
    3. For i = m downto 1 do
          For j = n downto 1 do
               For k = p downto 1 do
                    If X[i] = Y[j] = Z[k], M[i, j, k] = 1 + M[i − 1, j − 1, k − 1]
                    Else M[i, j, k] = max{M[i − 1, j, k], M[i, j − 1, k], M[i, j, k − 1]}
    4. Return M[m, n, p].
```

5. (10+20 Points) Dynamic Programming
The following exercises are found in Lecture VII.
(a) Exercises 4.2, the optimal order for multiplying matrices. (Read section 5 for more information about optimal multiplication of matrices.)
(b) Exercises 4.3, a wavelet computation problem.

SOLUTION:

a This takes $T(n) = \Theta(3^n)$ since the recurrence is $T(n) = 1 + 3T(n-1)$. MORE precisely: Using range transform, $t(n) = T(n)/3^n$ so that $t(n) = t(n-1) + 1/3^n$. Telescoping, $t(n) = \sum_{i=0}^{n} 1/3^i$ and $T(n) = \sum_{i=0}^{n} 3^i = (3^{n+1} - 1)/2$.

b The support of $f(x, n)$ is contained in $(-1, 1)$ for all n. This is true for $n = 0$. For $n \geq 1$, we use induction: Note that $h(x, n)$ depends on $h(y, n-1)$ where $y = 2x + 1, 2x, 2x - 1$. But if $x \leq -1$ then $y \leq -1$ and so $h(y, n-1) = 0$. Similarly, if $x \geq 1$ then $y \geq 1$ and gain $h(y, n-1) = 0$. In either case, this implies $h(x, n) = 0$.

c We note that the number of subproblems in evaluating $f(\frac{i}{m}, k)$, for $i \in \{-m, -m+1, \ldots, m-1, m\}$ and $k \in \{0, \ldots, n\}$ is $(n+1)(2m+1)$. This is because:

- When we expand an evaluation of $f(-, k)$ into evaluations of $f(-, k-1)$, the grid (i.e., the spacing between the values between the first arguments for the function at the new level) is not refined further (the spacing remain at $1/m$).
- The first argument can be restricted to the interval $(-1, 1)$ by part (b).

Let $D_m = \{i/m : i = -m, -m+1, \ldots, 0, 1, \ldots, m\}$. Hence, if we assume that the set $f(D_m, n-1) = \{f(x, n-1) : x \in D_m\}$ have been evaluated and stored for easy access, then each $f(x, n)$ for $x \in D_m$ can be evaluated in constant time. Thus $f(D_m, n)$ can be evaluated in $O(m)$ time assuming $f(D_m, n-1)$ is available. By induction, the time to evaluate $f(D_m, n)$ is $\Theta(nm)$.

A stronger observation can be made: even if we evaluate just $f(x, n)$ at ONE pair (x, n), the number of subproblems generated at every level $i < n$ is at most 2. This automatically implies the result of (iii).

6. (0 Points)
The problems below carry no credit. Please do not hand such problems; they are for your own practice. We strongly recommend that you do them.

7. (a) Show that the minimum spanning tree T of an undirected graph G with distinct weights must contain the edge of smallest weight. (b) Must it contain the edge of second smallest weight? (c) Must it contain the edge of third smallest weight?

8. Our cost model and analysis of binary counters can yield the exact cost (not just an upper bound) of incrementing from any initial counter value to any final counter value. Show that the exact cost to increment a counter from 68 to 125 is 190.

   SOLUTION:

   (a) It costs $2n-\Phi(D_n$ units to count from 0 to n, where D_n denotes the counter holding the binary value of n. If $\Delta\Phi$ is the increase in potential as we count from 68 to 125, we have $CHARGE = COST + \Delta\Phi$. We charge two units per operation, so $CHARGE = 2(125-68) = 114$. Since $125 = (1,111,101)_2$ and $68 = (1,000,100)_2$, we have $\Delta\Phi = \Phi(D_{125}) - \Phi(D_{68}) = 6-2 = 4$ Thus, $114 = COST + 4$, or COST=110.

9. Perform the following splay tree operations, starting from an initially empty tree.

$$\texttt{insert}(3,2,1,6,5,4,9,8,7), \texttt{lookUp}(3), \texttt{delete}(7), \texttt{insert}(12,15,14,13), \texttt{split}(8).$$

10. Suppose we want to compute the edit distance $D(X,Y)$ in which each delete and insert operations has cost 1, but each replace operation has a cost depending a and b, where we want to replace symbol a by symbol b. Let $r(a,b)$ denote this cost. We assume $0 < r(a,b) < 2$. Design an efficient algorithm for this problem.