Homework 6
Fundamental Algorithms, Spring 2008, Professor Yap

Due: Mon May 5, during the last class.

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.
- Please write succinctly, to the point. Every sentence must be an complete English sentence.
- You may post general questions to the homework discussion forum in class website. Also, bring your questions to recitation on Monday.

---

1. (2 Points)
   Exercise 6.5, Lect.V.

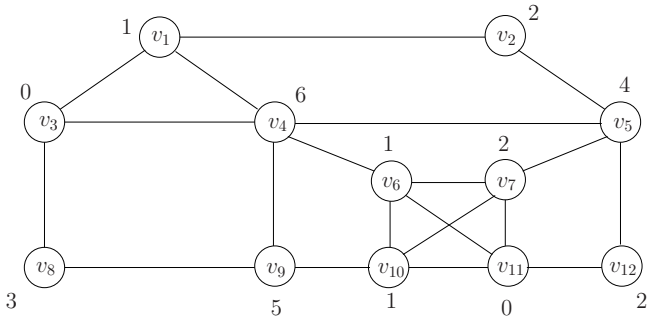

Figure 1: The house graph: The cost of edge v_i−v_j is defined as $C(v_i) + C(v_j)$, where $C(v)$ is the value indicated next to v. E.g. $C(v_1$−$v_4) = 1 + 6 = 7$.

Do hand simulation of Kruskal's Algorithm on the graph of Figure 1. We consider each edge in turn, maintaining a partition of $V = \{1, \ldots, 12\}$ 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 $L(1) = \{1\}$ and $L(3) = \{1\}$ 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 $L(j)$: Please fill in the last two columns of the table (we have filled in the first 4 rows for you).

Sorting Order	Edge	Weight	Merged List	Cumulative Weight
1	1-3:	1	$\{1,3\}$	1
2	6-11:	1	$\{6,11\}$	2
3	10-11:	1	$\{6,10,11\}$	3
4	6-10:	2	X	3
5	7-11:	2		
6	11-12:	2		
7	1-2:	3		
8	3-8:	3		
9	6-7:	3		
10	7-10:	3		
11	2-5:	6		
12	3-4:	6		
13	5-7:	6		
14	5-12:	6		
15	9-10:	6		
16	1-4:	7		
17	4-6:	7		
18	8-9:	8		
19	4-5:	10		
20	4-9:	11		

2. (15 Points)
Exercise 6.6, Lect.V.

This question considers two concrete ways to implement Kruskal's algorithm. Let $V = \{1, 2, \ldots, n\}$ and $D[1..n]$ be an array of size n that represents a **forest** $G(D)$ with vertex set V and edge set $E = \{(i, D[i]) : i \in V\}$. More precisely, $G(D)$ is an directed graph that has no cycles except for self-loops (i.e., edges of the form (i, i)). A vertex i such that $D[i] = i$ is called a **root**. The set V is thereby partitioned into disjoint subsets $V = V_1 \cup V_2 \cup \cdots \cup V_k$ (for some $k \geq 1$) such that each V_i has a unique root r_i, and from every $j \in V_i$ there is a path from j to r_i. For example, with $n = 7$, $D[1] = D[2] = D[3] = 3$, $D[4] = 4$, $D[5] = D[6] = 5$ and $D[7] = 6$ (see Figure 2). We call V_i a **component** of the graph $G(D)$ (this terminology is justified because V_i is a component in the usual sense if we view $G(D)$ as an *undirected* graph).

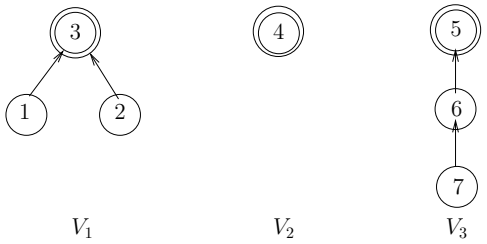

Figure 2: Directed graph $G(D)$ with three components (V_1, V_2, V_3)

(i) Consider two restrictions on our data structure: Say D is **list type** if each component is a linear list. Say D is **star type** if each component is a star (i.e., each vertex in the component points to the root). E.g., in Figure 2, V_2 and V_3 are linear lists, while V_1 and V_2 are stars. Let ROOT(i) denote the root r of the component containing i. Give a pseudo-code for computing ROOT(i), and give its complexity in the 2 cases: (1) D is list type, (2) D is star type.
(ii) Let COMP(i) $\subseteq V$ denote the component that contains i. Define the operation $MERGE(i, j)$ that transforms D so that COMP(i) and COMP(j) are combined into a new component (but all the other components are unchanged). E.g., the components in Figure 2 are $\{1, 2, 3\}, \{4\}$ and $\{5, 6, 7\}$. After $MERGE(1, 4)$, we have two componets, $\{1, 2, 3, 4\}$ and $\{5, 6, 7\}$. Give a pseudo-code that implements $MERGE(i, j)$ under the assumption that i, j are roots and D is list type which you must preserve.

Your algorithm *must* have complexity $O(1)$. To achieve this complexity, you need to maintain some additional information (perhaps by a simple modification of D).
(iii) Similarly to part (ii), implement $MERGE(i, j)$ when D is star type. Give the complexity of your algorithm.
(iv) Describe how to use ROOT(i) and $MERGE(i, j)$ to implement Kruskal's algorithm for computing the minimum spanning tree (MST) of a weighted connected undirected graph H.
(v) What is the complexity of Kruskal's in part (iv) if (1) D is list type, and if (2) D is star type. Assume H has n vertices and m edges.

3. (0 Points)
Exercise 1.1, Lect.VI.

Our model and analysis of counters can yield the exact cost to increment from any initial counter value to any final counter value. Show that the exact number of work units to increment a counter from 68 to 125 is 190.

4. (4 Points)
Exercise 1.2, Lect.VI.

A simple example of amortized analysis is the cost of operating a special kind of pushdown stack. Our stack S supports the following two operations: $S.\texttt{push}(K)$ simply add the key K to the top of the current stack. But $S.\texttt{pop}(K)$ will keep popping the stack as long at the current top of stack has a key smaller than K (the bottom of the stack is assume to have the key value ∞). The cost for push operation is 1 and the cost for popping $m \geq 0$ items is $m + 1$.
(a) Use our potential framework to give an amortized analysis for a sequence of such push/pop operations, starting from an initially empty stack.
(b) How tight is your analysis? E.g., can it give the *exact cost*, as in our Counter Example?

5. (10 Points)
Exercise 1.3, Lect.VI.

Let us generalize the example of incrementing binary counters. Suppose we have a collection of binary counters, all initialized to 0. We want to perform a sequence of operations, each of the type

$$\texttt{inc}(C), \quad \texttt{double}(C), \quad \texttt{add}(C, C')$$

where C, C' are names of counters. The operation $\texttt{inc}(C)$ increments the counter C by 1; $\texttt{double}(C)$ doubles the counter C; finally, $\texttt{add}(C, C')$ adds the contents of C' to C while simultaneously set the counter C' to zero. Show that this problem has amortized constant cost per operation.

We must define the cost model. The length of a counter is the number of bits used to store its value. The cost to double a counter C is just 1 (you only need to prepend a single bit to C). The cost of $\texttt{add}(C, C')$ is the number of bits that the standard algorithm needs to look at (and possibly modify) when adding C and C'. E.g., if $C = 11,1001,1101$ and $C' = 110$, then $C + C' = 11,1010,0011$ and the cost is 9. This is because the algorithm only has to look at 6 bits of C and 3 bits of C'. Note that the 4 high-order bits of C are not looked at: think of them as simply being "linked" to the output. Here is where the linked list representation of counters is exploited. After this operation, C has the value $11,1010,0011$ and C' has the value 0.
HINT: The potential of a counter C should take into account the number of 1's as well as the bit-length of the counter.

You couldn' do this witl arrays!

6. (0 Points)
Exercise 2.1 and 2.2, Lect.VI.

7. (5 Points)
Fix a key K and a splay tree T_0. Let $T_{i+1} \leftarrow \texttt{splay}(K, T_i)$ for $i = 0, 1, \ldots$.
(a) Under what conditions will the T_i's stabilize (become a constant)?
(b) What must be the case if the T_i's do not stabilize.

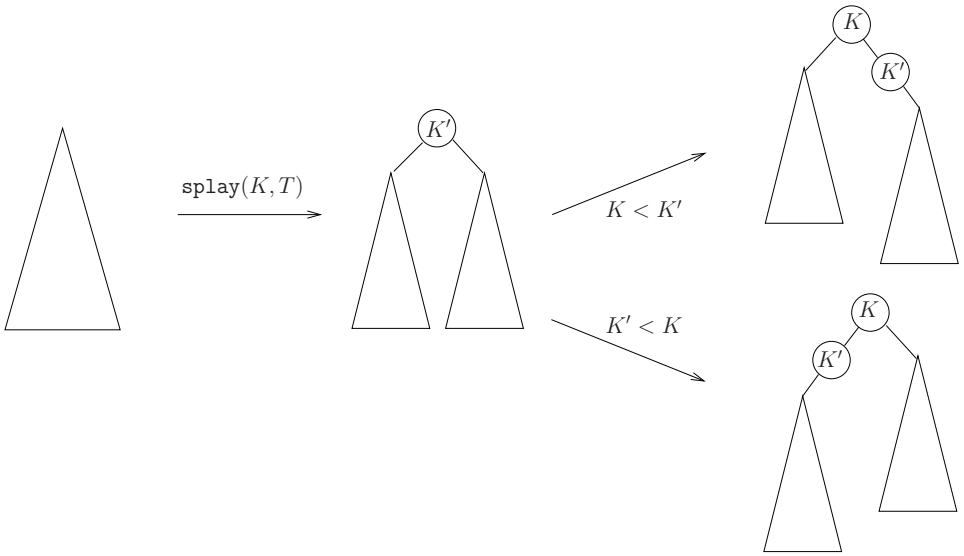

Figure 3: Alternative method to insert a key K.

8. (8 Points)
Exercise 2.7, Lect.VI.

In our operations on splay trees, we usually begin by performing a splay. This is not the case in our insertion algorithm. But consider the following variant insertion that follows this paradigm. To insert an item X into T:
1. Perform $\texttt{splay}(X.\texttt{key}, T)$ to give us an equivalent tree T'.
2. Now examine the key K' at root of T': if $K' = X.\texttt{key}$, we declare an error (recall that keys must be distinct).
3. If $K' > X.\texttt{key}$, we install a new root containing X, and K' becomes the right child of X; the case $K' < X.\texttt{key}$ is symmetrical. In either case, the new root has key equal to $X.\texttt{key}$. See Figure 3.
(a) Prove that the amortize complexity this insertion algorithm remains $O(\log n)$. NOTE: To do this, you must understand Theorem 1 (VI.§1) and might be helpful to look at the proof of Theorem 5 as well.
(b) (Ignore this part)

9. (15 Points)
Recall the convex hull problem in Lect.VI, §4. Consider an array-based representation of upper hulls. An upper hull $U = (v_0, \ldots, v_k)$ is represented by an array $U[0..k]$, where $U[i] = v_i$. We want to build U incrementally using METHOD ONE in the text. Assume the set of input points is non-degenerate (no 3 points are collinear).
(a) Describe in detail how to carry out the operation of finding LeftTangent of a given point p in U.
(b) Describe how to implement the insertion of a new point p.
(c) Carry out a complexity analysis of your algorithm.
(d) Show how to modify your algorithms if the input can be degenerate. You must discuss how to implement the changes using the LeftTurn(p,q,r) predicate. NOTE: For this problem, ASSUME that an input point p is in U if it does NOT lie in the interior of the convex hull.

10. (2 Points)
Exercises 1.3, Lect.VII.

11. (10 Points)
Exercises 1.10, Lect.VII. Let X, Y be strings.
(a) Prove that $L(XX, Y) \leq 2L(X, Y)$.

(b) Give examples where the inequality is strict, and where it is not strict.
(c) Prove that $L(XX, YY) \leq 3L(X, Y)$. How tight is this upper bound?

12. (12 Points)
Exercise 1.18, Lect.VII. Researchers are using LCS computation to fight computer viruses. A virus that is attacking a machine has a predictable pattern of messages it sends to the machine. We view the concatenation of all these messages that a potential virus sends as a single string. Call the first 1000 bytes than from any source (i.e., potential virus) the **signature** of that source. Let X be the signature of an unknown source and Y is the signature of a known virus. To test the source is the Y-virus, we compute $L(X, Y)$. Empirically, suppose it is shown that if $L(X, Y) > 500$, then that our source is likely to be Y-virus.
(a) Design a practical and efficient algorithm for the decision problem $L(X, Y, k)$ which outputs "PROBABLY VIRUS" if $L(X, Y) > k$ and "PROBABLY NOT VIRUS" otherwise. Give the pseudo-code for an efficient practical algorithm. NOTE: The obvious algorithm is to use the standard algorithm to compute $L(X, Y)$ and then compare n to k. But we want you to do better than this. HINT: There are two ideas we want you to exploit – most students only think of one idea.
(b) Quantify the complexity of your algorithm, and compare its performance to the obvious algorithm (which first computes $L(X, Y)$). First do your analysis using the general complexity parameters of $m = |X|, n = |Y|$ and k, and also $\ell = L(X, Y)$. Also discuss this for the special case of $m = n = 1000$ and $k = 500$.

13. (2 Points)
Exercise 2.4, Lect.VII. Compute $A(X, Y)$ where X, Y are the strings `AATTCCCGA` and `GCATATT`. Assume δ has gap penalty 2, $\delta(x, x) = -2$ and $\delta(x, y) = 1$ if $x \neq y$. You must organize this computation systematically as in the LCS problem.