Homework 7 Solutions  
Fundamental Algorithms, Fall 2011, Professor Yap  
Due: Tuesday Dec 6, in class.  
HOMEWORK with SOLUTION, prepared by Instructor and T.A.s  

INSTRUCTIONS:  
• Please write clearly, and economically. Use notations we introduce in lecture notes and in lectures.  
• If you cannot write clearly, then consider type-setting the solutions (not recommended in general because this takes too much time).  
• AGAIN: you get 2 bonus points just for writing your name in the correct format: [LAST-NAME-in-CAPS] [comma] [First-Name]. E.g., “YAP, Chee K.” or “YAP, Chee” are good, but “Yap, Chee” or “Chee Yap” or “Yap Chee K.” are no good.

1. (1 Points, Dijkstra)  
Carry out the hand-simulation of Dijkstra’s algorithm for the graph in Figure 2 (p.9) of Lecture XIV. But please use the modified edge cost function $C_9$ defined as follows: $C_9(e) := C(e) + 9$ if $C(e) \leq 9$, and $C_9(e) := C(e) - 9$ if $C(e) > 9$. Please present your solution using the convention of the Lecture Notes.

SOLUTION

<table>
<thead>
<tr>
<th>VERTICES</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAGE 0</td>
<td>0</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>STAGE 1</td>
<td>0</td>
<td>16</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAGE 2</td>
<td></td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAGE 3</td>
<td></td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAGE 4</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAGE 5</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAGE 6</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STAGE 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

Comments:

2. (4 Points, Dijkstra)  
Suppose we have a vertex-costed graph $G = (V, E; C, s_0)$ where $C : V \to \mathbb{R}$. The cost of a path is just the sum of the costs $C(v)$ of vertices along the path. Note that vertex-costed graphs is different from the standard graphs in min-cost path problems: in the standard problems, the graph is edge-costed, $C : E \to \mathbb{R}$.

(a) In page 9 of Lecture XIV, we gave a detailed version of Dijkstra’s algorithm. We now want you to modify this algorithm for our vertex-costed graphs. As in the Dijkstra algorithm, assume that that cost function is non-negative, $C(v) \geq 0$ for each $v \in V$.

(b) Prove that your algorithm in (a) is correct.

SOLUTION  
This is just repeating the algorithm in the text, and checking that the proof of correctness in the text also works in the present case!

Comments:
3. (3 Points, Floyd-Warshall)
Run the Floyd Warshall Algorithm on the bigraph in Figure 1. Note that there is a negative cycle. HINT: exploit symmetry in your matrices by just filling in the upper triangular entries. The original cost matrix is shown in Figure 1(c). Show us 6 more matrices.

SOLUTION After we update using $a$.

\[
\begin{array}{cccccc}
C: & a & b & c & x & y & z \\
\hline
a & 0 & 4 & 2 & 5 & 0 & 0 \\
b & 0 & 2 & 2 & 6 & 0 & 0 \\
c & 0 & 1 & 2 & 0 & 0 & 0 \\
x & 0 & 2 & 5 & 0 & 0 & 0 \\
y & 0 & 2 & 6 & 0 & 0 & 0 \\
z & 0 & 2 & 6 & 0 & 0 & 0 \\
\end{array}
\]

INCOMPLETE: I plan to write a program to automate this... let me know if you have such a program.

We know that eventually, all the entries must contain $-\infty$. But this may take sometime to propagate through the system.

Comments: NOTE: the proper extension to Floyd-Warshall for bigraphs with negative edges ought to use the concept of irreducible paths. But we need to develop a different algorithm for this purpose.

4. (2+3+3 Points, External BST)
In Lecture III.§5, we discussed the concept of “external binary trees”. The leaves of an external binary trees are called external nodes (the others are internal nodes, of course). We now develop the concept of External BST where we store items only in external nodes, and store keys (no data) in internal nodes. The set of keys in all the nodes (whether internal or external nodes) must obey the usual BST Property. This allows us to do Lookup on items. But to insert and delete properly, we must discuss how the internal keys are generated from the keys in the items. Note that an External BST $T$ has $n$ internal nodes iff it has $n + 1$ external nodes. If $u$ is external and $v$ is internal, we require the following External BST Property:

$$u.\text{key} = v.\text{key} \iff u \text{ is the successor of } v.$$ (1)

A consequence of (1) that the only item whose key is not duplicated in an internal node is the minimum item. In order to maintain (1) efficiently under insertion and deletion, we need to be be able to get from an external node to its predecessor. Here is how: We assume that each node $u$, whether internal or external, has the usual pointers, $u.\text{parent}, u.\text{left}, u.\text{right}$. For an internal node, $u.\text{left}, u.\text{right}$ are non-nil, and points to its two children. For an external node, $u.\text{right} = \text{nil}$ and $u.\text{left}$ points to its predecessor. However, if $u$ is the leftmost leaf, then $u.\text{left} = \text{nil}$.

(a) Draw the external BST that stores the following set of items (represented by their keys only):
Assume your BST has the shape of Figure 1(b).

(b) Describe an insertion algorithm for external BST. Illustrate your algorithm by inserting the letter \( j \), and also \( n \) into the external BST in part(a). (The two insertions should be done independently, not sequentially.)

(c) Describe deletion algorithm for external BST. Illustrate your algorithm by deleting the letter \( h \), and also \( l \) from the external BST in part(a). (The two deletions should be done independently, not sequentially.)

**SOLUTION**

(a) See Figure 2.

(b) To insert a node \( x \) containing item with key \( k \), we first Lookup \( k \) to find an external node \( y \). If \( y.key = k \), we fail. Otherwise, note that \( y.key < k \). In this case, we create a new internal node \( u \) to replace \( y \). Make \( y \) and \( x \) (resp.) the left and right child of \( u \). Finally, let \( x.left \leftarrow u \) and \( u.key \leftarrow k \).

When we insert \( j \) into the tree in (a), it will become the right sibling of \( h \). When we insert \( m \) into the tree in (a), it will become the right sibling of \( l \).

(c) To delete an item with key \( k \), we also start with Lookup of \( k \) to find an external node \( x \) containing \( k \) (otherwise deletion fail). Let \( p \) be the parent of \( x \) and \( y \) the sibling of \( x \). First, we replace \( p \) by \( y \). There are two cases:

(CASE A) If \( x \) is the right child of \( p \), there is nothing else to do.

(CASE B) If \( x \) is the left child of \( p \), the (former) predecessor of \( y \) is now the predecessor of \( x \). So we need to do \( x.left \leftarrow y.left \) and \( x.left.key \leftarrow k \).

When we delete \( l \) from the tree in (a), this is CASE A, and the external node \( h \) will become a new child of the internal node \( o \). When we delete \( h \) from the tree in (a), this is CASE B. The external node \( l \) will become a new child of the internal node \( o \), and the previous internal node with \( h \) is now the predecessor of the external node \( l \), and we must replace the key \( h \) by \( l \).

**Comments:**

5. (2+2+2+2+4 Points, Splay Trees and Dynamic String Compression)

We consider the “dynamic string compression” problem – this is an easy and fun problem. I have made this problem independent, but if you like, you can read Lecture V. §4 for insights. Let \( X \) be a string over the ASCII alphabet \( \Sigma \). We may assume \( \Sigma = \{0, 1\}^\mathbb{N} \). E.g., the ASCII code for \( A \) is 0x41 or \((0100, 0001)_2\), and for \( \cup \) (space) is 0x20 or \((0010, 0000)_2\). Here, \( 0x... \) and \((0...)_2 \) indicate hexadecimal and binary notations, respectively. The comma in binary notation is just to help us parse the string. Imagine a communication protocol involving two parties: the transmitter and the receiver. The transmitter has string \( X \) and wants to transmit a bit stream, denoted \( E(X) \), to the receiver. The receiver should be able to reconstruct the string \( X \) from \( E(X) \). Of course, we would like \( E(X) \) to be as short as possible. A trivial solution is to transmit \( X \) directly, using a total of \( 8|X| \) bits where \( |X| \) is the length of \( X \). But we want to design a better protocol, with the help of external BST trees (developed in the last problem). Let \( T \) be an external BST storing a subset \( S \subseteq \Sigma \), using the natural sorting order on \( \Sigma \). For any \( x \in S \), there is a unique path from the root of \( T \) to the external node containing \( x \). This path corresponds to a binary string, denoted \( E(x) \), where 0 is a left-child and 1 is a right-child in the path. Thus, \( E(x) \) is an encoding of \( x \). If \( X = x_1 x_2 \cdots x_n \ (x_i \in S) \) then \( E(X) = E(x_1) E(x_2) \cdots E(x_n) \). We assume that
the transmitter (resp., receiver) has an external BST $T$ (resp., $R$). The idea is for both parties to keep $T$ and $R$ synchronized (identical).

(a) Explain how the transmitter can transmit $X$ to the receiver provided $X \in S^*$ and $T = R$ initially. Be sure to explain how $T$ and $R$ are used.

(b) Suppose $T$ and $R$ are now external splay trees. These are just external BSTs in which we always do a SPLAY after each Lookup, Insert or Delete. If we do Lookup/Insert/Delete, we always end up at some external node $u$. Then we splay $u$.parent (we must NOT splay $u$ itself). Explain how we can modify part(a) to using external splay trees, and say why this is expected to be an improvement.

(c) Suppose $S = \{a, b, c, d, e, f, g, h\}$ is the set of items stored in the external nodes of $T$ and $R$, and these trees have the shape of a complete binary tree. Show the sequence of binary bits transmitted if $X = \text{chaff}e\text{g}h$ $T$, $R$ are external BST. PLEASE show your bit strings in groups of 4 (use hexadecimal if you like) and annotate your string to help the graders (if you make a small mistake, we can even ignore them if you show that you know what you are doing).

(d) Repeat part(c) but now, we assume that $T$, $R$ are external splay trees.

(e) We now make one final modification: suppose the transmitter does not know the set $S$ in advance, and the external splay tree $T$ initially contains only the special letter * . Assume that the character * is the least character in $\Sigma$. Each time the transmitter wants to transmit a letter $x$, he does a Lookup($x$). If $x$ is not found, he will transmit the bit string $E(*)$, followed by the ASCII code of $x$ (this is 8 bits). Then he will insert $x$ into $T$. It is clear that the receiver can detect this situation and output $x$, and at the same time, insert $x$ into $R$. Please show the bit string that is transmitted when the message is $\text{hello}...\text{world}$. Please look up the ASCII code for these letters on the web.

| SOLUTION | (a) NOTE: CASE (a) DO NOT NEED INSERTION INTO $T$. If $X = x_1x_2, \ldots, x_n$, then the transmitter takes each $x_i$ ($i = 1, \ldots, n$) in turn, and do Lookup on $x_i$ in $T$. This produces a string $E(x_i)$ which he transmits. The receiver, on receiving $E(x_i)$ will use it to travel from the root of $R$ to the external node containing $x_i$, and will be able output the encoding of $x_i$.
(b) In case $T$, $R$ are splay trees, the transmitter will also do a Lookup, and transmit the string $E(x_i)$ representing the Lookup path. However, after the Lookup, the parent node of $x_i$ will be splayed. The receiver, upon receiving $E(x_i)$, will be able to local $x_i$, and can output $x_i$. Why do we expect this to improve on (a)? Because if a character $x$ occurs often in $X$, its depth is likely to be shall and so $E(x)$ (on average) will be a short binary sequence.
(c) Here are the first few bits: $E(c) = 010, E(h) = 111, E(a) = 000, E(f) = 101, E(f) = 101$, etc. So $E(X) = 0101, 1100, 0101, 101 \cdots$.
(d) Here are the first few bits: $E(c) = 010, E(h) = 1111, E(a) = 0000, E(f) = 101101, E(f) = 100$, etc. So $E(X) = 0101, 1110, 0001, 0110, 1110, \cdots$.
(e) This has been explained in class.

Comments:

6. (3 Points, Amortized Analysis)

Give an amortized analysis for incrementing a 3-ary counter $C$. The cost model is exactly as for binary counters. You must define a charge, a potential function $\Phi(C)$ and show the credit-potential invariant. Discuss whether your analysis is tight, and speculate on how to extend this analysis to $k$-ary counters for any $k \geq 2$.

| SOLUTION | Suppose the counter $C$ has a suffix of the form $c2^n$ where $c \in \{0, 1\}$ and $n \geq 0$. Then after incrementing, the suffix changes to $d0^n$ where $d \in \{1, 2\}$. The cost is $n + 1$.
We will charge two units for each increment. Let us define potential $\Phi(C)$ to be the number of 2’s in $C$. Suppose $n = 0$. Then the cost is 1 and the charge of 2 units can pay for any increase in potential.
It is clear that this analysis is not tight in a certain sense. In general, if you extend this kind of analysis to a $k$-ary, you can charge each operation only $1 + \frac{1}{k-1}$ instead of 2.

Comments: |