CS250 Assignment 5 solutions

April 28, 2003

Question 1. [Gregory]

Part 1:

Given a chain of $n$ matrices $A_1 \times A_2 \times \ldots \times A_n$, let $P(n)$ equal the number of different ways to multiply these matrices together. We note that $P(n)$ can be defined recursively: we can choose to multiply the subchain $(A_1 \times A_2 \times \ldots \times A_k)$ and the subchain $(A_{k+1} \times \ldots \times A_n)$ for some $1 \leq k < n$ independently and then multiply their results to produce a final answer. Then $P(n)$ is simply equal to $P(k) \times P(n - k)$, since we need to look at every possible way to multiply the $k$ matrices in the first subchain and the $n - k$ matrices in the second subchain. Since any $k$ between 1 and $n - 1$ will work, our final equation becomes:

\[
P(n) = \sum_{k=1}^{n-1} P(k)P(n - k) \text{ for } n \geq 2 \\
P(1) = 1
\]

We can inductively show that, for $n \geq 2$, $P(n) \geq 2^{n-2}$: For our base case, let $n = 2$. There is only one way to multiply two matrices together, so $P(2) = 1 = 2^{2-2}$. Now, assume that the $P(n)$ case is satisfied. Examine the $P(n + 1)$ case:
\[ P(n + 1) = \sum_{k=1}^{n+1-1} P(k)P(n + 1 - k) \]
\[ = P(n)P(1) + \sum_{k=1}^{n-1} P(k)P(n + 1 - k) \]
\[ \geq P(n)P(1) + \sum_{k=1}^{n-1} P(k)P(n - k) \]
\[ = P(n)P(1) + P(n) \]
\[ = 2P(n) \]
\[ \geq 2\cdot2^{n-2} \text{ (by the inductive hypothesis)} \]
\[ = 2^{n-1} \]
\[ = 2^{(n+1)-2} \]

Now, we can easily show that \(2^{n-2}\) is \(\Omega(2^n)\): just let our constant \(c = 1/4\). Then \(2^{n-2} = (1/(2^2)) \cdot (2^n) = (1/4) \cdot (2^n) \geq c \cdot 2^n\) for all \(n\). And since we’ve established that \(P(n) \geq 2^{n-2}\), it follows that \(P(n) \geq c \cdot 2^n\) as well, so \(P(n)\) is \(\Omega(2^n)\), and thus so is any algorithm that calculates \(P(n)\) combinations.

**Part 2:**

One answer is the following:

```
OptimalMatrixMultiplicationCount(d1,d2,...dn):

create two-dimensional array N (of size n x n)

for i= n downto 1 do:
    N[i,i] = 0
    for j= i to n do:
        min = infinity;
        for k = i to j-1 do:
            count = N[i,k] + N[k+1,j] + d[i]d[k+1]d[j+1]
            if (count < min) then
                min = count
        N[i,j] = min

return N[1,n]
```
Question 2. [Gregory]

The following example Heap implementation implements the pseudocode algorithms described in the class lecture notes for Week 9, Lecture 3:

// (in file Heap.java):

// Inserts the value ‘val’ into the heap. We implement this
// routine from the pseudo-code algorithm described in the
// class slides from lecture 9-3.
public void insert(int val) {
    T.insert();
    int v = T.lastIndex();
    T.set(v, val);

    while ((v != 1) && (T.get(v/2) > T.get(v))) {
        int temp = T.get(v);
        T.set(v, T.get(v/2));
        T.set(v/2, temp);
        v = v/2;
    }
}

// Returns the minimum value in the heap. If the heap
// is empty, return 0. We implement this routine from
// the pseudo-code algorithm described in the class
// slides from lecture 9-3.
public int extractMin() {
    if (T.lastIndex() == 0) // if the array is empty?
        return 0;

    int min = T.get(1); // get the root node value

    T.set(1, T.get(T.lastIndex()));
    T.delete();
    int v = 1;

    while (2*v <= T.lastIndex()) {
        }
if (2*v+1 > T.lastIndex()) {
    if (T.get(2*v) < T.get(v)) {
        swap(v, 2*v);
        v = 2*v;
    } else
        break;  // This line is important!
} else {
    // Find Min(T[v], T[2v], T[2v+1]):
    int minvertex = T.get(v);
    if (T.get(2*v) < minvertex) minvertex = T.get(2*v);
    if (T.get(2*v+1) < minvertex) minvertex = T.get(2*v+1);

    if (minvertex == T.get(v))
        break;
    else if (minvertex == T.get(2*v)) {
        swap(v, 2*v);
        v = 2*v;
    } else {
        swap(v, 2*v+1);
        v = 2*v+1;
    }
}

return min;
}

// This is a new method used to swap two heap values
private void swap(int index1, int index2)
    throws IndexOutOfBoundsException {

    int temp = T.get(index1);
    T.set(index1, T.get(index2));
    T.set(index2, temp);
}

Notice the “else break” line in the extractMin() method. It's very important to have this line (or something similar), otherwise the method might loop forever under certain
conditions. Also, notice that extractMin() returns 0 if the heap is empty (instead of just throwing an IndexOutOfBoundsException exception).

The testing routine that was used for grading heap implementations did the following:

1. Insert the following sequence of numbers into an initially empty heap: 4, 9, 1, 8, 5, 10, 3, 7, 2, 6
2. Call heap.extractMin() 10 times (should return this sequence: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
3. Call heap.extractMin one more time (should return zero, since the heap is now empty).

**Question 3. []**

**Question 4. [Irina]**

Show that the language of regular expressions over \( \Sigma = \{a, b, (, ), *, | \} \) is a context-free language by generating a context-free grammar for it.

Here’s one of many possible solutions. To avoid ambiguity between ‘|’ that is used to define a grammar and ‘|’ that is part of the \( \Sigma \) alphabet for regular expressions, let’s substitute the latter by ‘or’, defining \( \Sigma \) as \( \{a, b, (, ), *, or\} \)

CF grammar:

\[
\begin{align*}
\alpha & \rightarrow a | b \\
t & \rightarrow * | \epsilon \\
E & \rightarrow \alpha t | \alpha t E | (\alpha t E) t | (E or E) | \alpha t or \alpha t \\
S & \rightarrow \epsilon | E | E or E
\end{align*}
\]