CPSC 320 Sample Midterm 2
November 2011

[10] 1. Disjoint Sets Structures

[3] a. What is the Disjoint Sets data structure used for in Kruskal's minimum spanning tree algorithm?

Solution : It is used to determine the connected components that the endpoints of each edge belong to.

[3] b. Intuitively (that is, without mentioning amortized costs or potential functions) what is the purpose of performing path compression during every `find()` operation in a Disjoint Sets data structure?

Solution : It brings the nodes of the path closer to the root of the tree, and hence speeds up later `find()` operations on these nodes or their descendants.

[4] c. In our amortized analysis of the Disjoint Sets data structure, we defined a node as *close*, *far* or *very far* depending on the relationship between its rank and the rank of its parent. We then analyzed the amortized cost of a `find(N)` operation, and argued that if this operation looks at l nodes, then its amortized cost is in $O(\log^* n)$, where n is the total number of nodes in the data structure. Explain briefly how we obtained this $\log^* n$ term.

Solution : We proved that most nodes on the path from N to the root would lose one unit of potential as a result of the path compression. The exceptions where (1) a small constant number of nodes [the root, its child, and the maybe the "far" node closest to the root on the path] and (2) the "very far" nodes on the path. There is at most one "very far" node per interval, and hence at most $\log^* n$ of them in total. This means the potential decreased by $l - 3 - \log^* n$, and so the amortized cost was in $O(\log^* n)$.
Note: you did not need to provide all these details to get 4/4.

[9] 2. Recurrence relations

[4] a. Explain how one should prove a tight bound on the solution to the following recurrence relation:

$$T(n) = \begin{cases} 5T(\lceil n/4 \rceil + 8) + \Theta(n) & \text{if } n \geq 12 \\ \Theta(1) & \text{if } n \leq 11 \end{cases}$$

Do not prove this bound; simply state how one should prove it.

Solution : We would first use the Master theorem with the recurrence

$$T'(n) = \begin{cases} 5T'(\lceil n/4 \rceil) + \Theta(n) & \text{if } n \geq 12 \\ \Theta(1) & \text{if } n \leq 11 \end{cases}$$

to get a bound (in this case, $\Theta(\log_4 5)$), and then we would prove that the same bound holds for $T(n)$ using mathematical induction (that is, "guess and test" using the Master Theorem to obtain the guess).

[5] b. Write a recurrence relation that describes the running time of the following algorithm on an input array with n elements:

```
Algorithm Recursive(A, first, n)
  if (n > 1) then
    Recursive(A, first, n/2)
    Recursive(A, first + n/2, n/3)
    Recursive(A, first + n - n/4, n/4)
    LinearTimeAlgorithm(A, first, n/2, first + n/2, n - n/2 - n/4)
    LinearTimeAlgorithm(A, first, n - n/4, first + n - n/4, n/4)
```

You may assume that / performs integer division (that is, `x/y` returns the floor of x/y), and that a call to `LinearTimeAlgorithm`(A, first, n1, second, n2) runs in $O(n1 + n2)$ time. Simplify the recurrence relation you write as much as possible. You do not need to write floors and ceilings.

Solution:

$$T(n) = \begin{cases} T(n/2) + T(n/3) + T(n/4) + \Theta(n) & \text{if } n > 1 \\ \Theta(1) & \text{if } n \leq 1 \end{cases}$$

[8] 3. In class, we described an algorithm that finds both the minimum and the maximum elements of an array using at most $3\lfloor n/2 \rfloor$ comparisons (you do not need to remember the details of the algorithm for this problem; recall however that given two pairs (x_1, y_1), (x_2, y_2) where $x_1 \leq y_1$ and $x_2 \leq y_2$, we can determine the minimum and maximum elements of the two pairs, that is $(\min\{x_1, x_2\}, \max\{y_1, y_2\})$, using 2 comparisons).

Describe a divide-and-conquer algorithm that takes as input an array `A`, and two integers `first` and `last`, and returns the pair (x, y) where x is the smallest element from `A[first]` to `A[last]`, and y is the largest element from `A[first]` to `A[last]`.

Solution:

```
Algorithm MinAndMax(A, first, last)
if (first = last) then
    return (A[first], A[first])

mid ← (first + last)/2
pair1 ← MinAndMax(A, first, mid)
pair2 ← MinAndMax(A, mid + 1, last)

return (min(pair1[0], pair2[0]), max(pair1[1], pair2[1]))
```

[9] 4. Consider the following algorithm:

```
Algorithm Mysterious(array)
  accumulator ← 0
  for i ← 0 to length[array] - 1 do

    while (accumulator > 0 and compute(accumulator, array[i]) > 0)
      accumulator ← accumulator - 1

    if (array[i] is even) then
      accumulator ← accumulator + floor(log(i+1))

  return accumulator
```

Use the potential method to prove that this algorithm runs in $O(n \log n)$ time where $n =$ `length[array]`. You may assume that the function `compute()` runs in $\Theta(1)$ time. Hints:

- Think of the state of the algorithm at the end of the i$^{\text{th}}$ iteration as D_i.
- Use the value of `accumulator` at the end of the i$^{\text{th}}$ iteration as the potential of D_i.

Do not forget to show that Φ is a valid potential function.

Solution : First we observe that $\Phi(D_0) = 0$ (the initial value of the accumulator), that the value of the accumulator is always an integer, and hence that $\Phi(D_i) \geq 0$ (since we never subtract 1 from the accumulator unless it is positive).

Now let us compute the amortized cost of one loop iteration. Suppose that the body of the `while` loop executes t times. The real cost of the iteration is $t + \Theta(1)$. The potential will first go down by t, and then increase by at most $\lfloor \log(i+1) \rfloor$. Hence the potential difference is at most $\lfloor \log n \rfloor - t$. Thus the amortized cost of the iteration is $t + \Theta(1) + \lfloor \log n \rfloor - t \in O(\log n)$.

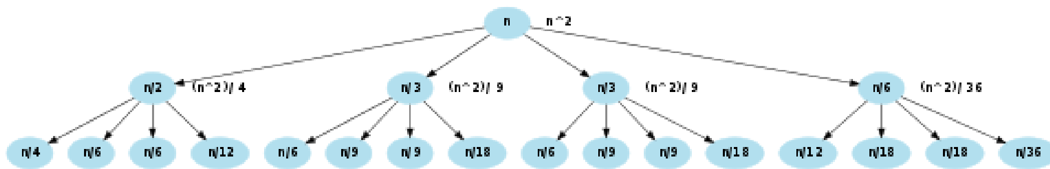
The actual cost of the n loop iterations is thus at most the sum of the amortized costs of the iterations, which is itself n times $O(\log n)$, and hence in $O(n \log n)$.

[9] 5. Prove upper and lower bounds on the function $T(n)$ defined by

$$T(n) = \begin{cases} T(n/2) + 2T(n/3) + T(n/6) + n^2 & \text{if } n \geq 6 \\ 1 & \text{if } n \leq 5 \end{cases}$$

Your grade will depend on the quality of the bounds you provide (that is, showing that $T(n) \in \Omega(1)$ and $T(n) \in O(100^n)$, while true, will not give you many marks).

Solution : Here are the first three levels of the recursion tree:



The amount of work done on the first level of the tree (level 0) is n^2. The amount of work done on level 1 is

$$(n/2)^2 + 2(n/3)^2 + (n/6)^2 = n^2/2$$

On level 2, we can observe that each node does half the work done by its parent (this can be proved by simply calling the work done by a level-1 node x, and then doing for that node's children the same computation that we did for the children of the root). Hence the total amount of work done on level 2 is half the work done on level 1, that is $n^2/4$.

The amount of work done on level i is thus $n^2/2^i$ for the first $\log_6 n$ levels, and then decreases until the last leaf on level $\log_2 n$. We thus obtain the following upper bound on $T(n)$:

$$\begin{aligned} T(n) &\leq \sum_{i=0}^{\log_2} n^2/2^i \\ &\leq \sum_{i=0}^{\infty} n^2/2^i \\ &= n^2 \sum_{i=0}^{\infty} 1/2^i \\ &= 2n^2 \end{aligned}$$

which means that $T(n) \in O(n^2)$. The top node of the tree perform n^2 work all by itself, and so $T(n) \in \Omega(n^2)$ as well.