15499-Spring 2009
Homework 4 solutions

March 17, 2009

1 List prefix sums

1. We will show that the size of S is $O(n/\log n)$ with high probability using Chernoff bounds. Assoicate indicator variable X_i with the i-th element of the initial list to indicate whether or not it has been selected in to S. Clearly, X_i are all independent. Also $E[X_i] = 1/\log n$, therefore $E[X] = E[\sum_i X_i] = \sum_i E[X_i] = n/\log n$ by linearity of expectations. Using Chernoff bounds,

$$Pr[X > (1+1)(n/\log n)] < e^{(-n/\log n)/4} < 1/n^3. \quad (1)$$

   The amount of work needed for step 4 is $|S| \log |S| < (2n/\log n) \log (2n/\log n) < 4n$ with high probablilty. It is evident that the work in steps $1, 2, 3, 5$ is $O(n)$ in the worst case. Therefore, the total work is $O(n)$ with high probability bound.

2. Steps $1, 3$ can be done in depth $O(1)$. Step 4 can be computed in depth $O(\log n)$ with high probability (we saw this in class). We will look at step 2 (the analysis for step 5 is the same). View the list of n elements as a collection of $n/(2\log^2 n)$ chunks of size $2\log^2 n$ each. The probability that none of the elements in a particular chunk are selected in to S is

$$(1 - 1/\log n)^{2\log^2 n} = ((1 - 1/\log n)^{\log n})^{2\log n} \leq (1/e)^{2\log n} \leq 1/n^2 \quad (2)$$

   The probability that at least one of the chunks has no element in S is less than $(n/(2\log^2 n)) * (1/n^2) \leq 1/n$ by union bounds. Therefore, with probability greater than $1 - 1/n$, none of the elements $s \in S$ has more than $4\log^2 n$ elements between s and $next(S)$. Therefore, the depth of steps $2, 5$ is less than $O(\log^2 n)$ with high probability. $D(n) = O(\log^2 n)$ with high probability.

Many of you have tried to show that the depth is $O(\log n)$. However, for any constant c, the probability that no element is selcted in to S from $c\log n$ consecutive elements is $(1 - 1/\log n)^{c\log n} \geq (1/4)^c, \text{for} n \geq 2$. Since $(1/4)^c$ is a constant for all constants c, it is not possible for the depth to be $c\log n$ with high probability.

2 Fast Maximum

1. Associate a bit b_i with each element x_i and set all these bits to 1 initially (work n and depth 1). For each pair of elements $x_i, x_j, i \neq j$, make a comparison in parallel (work

$(n^2 - n)/2$ and depth 1) and attempt the write $b_i = 0$ if $x_i < x_j$ and $b_j = 0$ if $x_j < x_i$. If an element is smaller than some other element, it's assoicated bit will get set to 0. All elements whose associated bits are 1 have the same value. In parallel, if b_i, then write x_i to the output word. Since all attempted writes are the same, the output is the maximum.

2. We use divide and conquer with branching factor $n^{1/3}$. Divide the array in to blocks of $n^{2/3}$ elements and recursively find their maximum (the recursion bottoms out when $n \leq 2$. From the maximum elements of each of the $n^{1/3}$ blocks, compute the maximum by using the brute force comparison described above.

$$W(n) = O((n^{1/3})^2) + n^{1/3}W(n^{2/3}) \quad (3)$$
$$D(n) = O(1) + D(n^{2/3}) \quad (4)$$

These recurrence relations solve to $W(n) = O(n), D(n) = O(\log \log n)$.

3 Treap Intersection

The split operation which a key k and treap T and splits in to two treaps, one with keys less than k and the other with keys not less than k. It is trivial to modify this algorithm (with only a constant factor increase in cost) to make it output whether or not the input k is in the treap T. Therefore, the return value of split is

```
(Tl, Tr, keyIn),
```

where `keyIn` indicates the presence or absence of key k in T. Pseudocode for intersection:

```
intersect (treap T1, treap T2)
   if (empty(T1) OR empty(T2)) return NULL
   else
      if (T1.root.prioity < T2.root.prioirity) return intersect (T2, T1)
      else (Tl, Tr, keyIn) <- split (T2, T1.root.key);
           left_output = intersect (T1.left, Tl);
           right_output = intersect (T1.right, Tr);

           if (keyIn) return (left_output, T1.root, right_output)
           else return join (left_output, right_output)
```

4 Random mate on graphs

A vertex is relabelled if, and only if, it's corresponding coin toss is T (probability 1/2), and at least one of it's neighboring vertex gets a H (probability $1 - (1/2)^d$). Therefore, a vertex is relabelled with probability $1/2(1 - (1/2)^d)$, which means the expected number of vertices after one contraction step is $n - n(1/2(1 - (1/2)^d)) = (n/2)(1 + (1/2)^d))$.

5 MST(extra credit)

Assume that the initial graph is connected. Each iteration to find a minimum weight edge (in case there are several min-weight edges at each vertex, one of them is arbitrarily chosen) incident on each vetex will take at most m work and can be done in $O(\log \log n)$ depth (problem 2). Once the set of minimum weight edges incident on each vertex is computed, the forest (note that these edges can not form a cycle) corresponding to these edges can be contracted using $O(n)$ work and $O(\log n)$ depth. Relabelling the edges accordingly and discarding self-loops will not involve more than m work and can be done in $O(1)$ depth. Therefore, each iteration takes not more than $O(m)$ work and $O(\log n)$ depth.

In each iteration, the vertex set shrinks by half because a forest constructed by selecting the minimum weight edge for each vertex of a connected graph can not have a disconnected component that consists of a single vertex. This means that each connected component of such a forest contains at least 2 vertices, and the forest of x vertices contains not more than $x/2$ connected components, which after tree contraction yields a graph on less than $x/2$ vertices.

Therefore, the maximum number of iterations is $\log n$, which means that the total work is $O(m \log n)$ and the depth is $O(\log^2 n)$.