Eric Tjon
Yulan Jin
Scott Fang
April 8th, 2019

CS255 HW3

1. Devise a CREW PRAM algorithm that generates a random permutation in O(logn) steps. Give a Θ-bound on the number of bits randomness it uses and number of processors needed by your algorithm.

Permute-By-Sorting(A)
n :=length[A]
Parallel for i :=1 to n
    P[i] = Random(1,n3)
Box sort A, using P as sort keys
return A.

Recall that box sort from the class notes is a parallel sort that's likely to run in O(log n) time. The parallel for loop for assigning the random keys also occurs in O(log n) time.

Therefore, this permutation runs in O(log n) time with O(n log n) bits of randomness.

2. Give a concrete example with 16 processors where the faulty processor succeeds in foiling a threshold choice in the Byzantine agreement algorithm.

Out of 16 processors, there are 15 good processors and 1 faulty processor.
Recall that the three different thresholds are L, H, and G, and the processors follow the algorithm below.

$L=(5n/8)+1 = 11$
$H=(3n/4)+1 = 13$
$G=7n/8 = 14$

Input: A value for b[i], our current decision choice.
Output: A decision d[i].
1. vote = b[i].
2 For each round, do
3. Broadcast vote;
4. Receive votes from all the other processors.
5. Set maj = majority (0 or 1) value among the votes cast
6. Set tally = the number of votes that maj received.
7. If coin = heads then set threshold = L; else set threshold = H
8. If tally >= threshold then set vote = maj; else vote = 0
9. If tally >= G then set d[i] = maj permanently.

A threshold choice is foiled if some good processor tallies below the threshold, and another good processor tallies at or above the threshold.

An example foiled threshold:
Group A: 10 good processors vote =1
Group B: 5 good processors vote = 0

The faulty processor says it votes 0 to group A, and says it votes 1 to group B. The coin flip is heads so threshold = L = 11.

Majority: 1
Group A Tally: 10/16 for majority
Group B Tally: 11/16 for majority

Since they are on opposite sides of the threshold, the threshold is foiled. Group A is set to 0 and group B is set to the majority which is 1.

Group A: 10 good processors vote = 0
Group B: 5 good processors vote = 1

3. Carefully give a DMRC algorithm for determining in a communications network the median incoming network traffic to a node. Assume the input consist of (key; value) pairs of the form ((i,j);ni,j) where the key is the pair (i,j) and ni,j is the number of bytes of traffic from node i to j in the network.

For each input pair, we want to map it to the accepting node j as the key, and the traffic ni,j as the value.

```
Map(k, v):
        Output(j, ni,j)
```

Next, we want to reduce all incoming traffic values ni,j as (v1, v2, ...vn) under the key of the accepting node j as k.

```
reduce(k,(v1, v2,..., vn)):
   values = sort(v1, v2,..., vn)
   if n is odd:
        median = values[ (n+1)/2 ]
   else:
        median = avg( values[n/2] + values[n/2 + 1]
   output(k, median)
```

The output would then be (key, value) where key is a node, and the value is the median of incoming traffic to that node.

Analysis:

The algorithm devised for parallel MTIS is very similar to the one proposed for performing parallel MIS search:

```
Given Graph = (Vertices V, Edges E):
    output = empty set
    While V is not empty:
        tempOutput = empty set
        Parallel for each v in V:
            If  d(v) = 0, add to output and remove v from V
            Else mark v with Pr(1/2d(v))
        Parallel for each (v1,v2) in E:
            If v1 and v2 are marked and v1 and v2 are in the same triangle,
             unmark vertice w/ lower degree or lower id
        Parallel for each v in V:
            If v is marked, add v to tempOutput
        output = output union tempOutput
        Delete tempOutput from V and all vertices that share a triangle w/ tempOutput
        Delete incident edges from E
```

We want to see if this algorithm holds similar properties to the original algorithm.

Lemma: For good vertice v with d(v) > 0, the probability of gamma(w) where w is a neighbor of v being marked is at least 1 - exp(-⅙).

Each vertice w has Pr(1/2d(w)) chance of being marked. Since v is good, it has at least d(v)/3 vertices that have at most d(v) neighbors. Those adjacent vertices have Pr(1/2d(v)) chance of being marked. The chance that none of these are marked is (1 - (1/2d(v)))^(d(v)/3) <= e^(-⅙)

Lemma: During any iteration, if a vertex w is marked, it is selected to be in S w/ probability at least 1 - P(triangle |w,v).

A vertice is unmarked if it is neighbors with a vertice that is part of the same triangle and that vertice is marked as well. Each neighbor is marked w/ probability at most 1/2d(w), and the number of such neighbors is at most d(w). Vertices are also unmarked if they share a triangle, that is, they share at least one other neighbor with each other. Without accounting for shared neighbors, the probability that the marked vertex v is selected to be in S is at least ½ (see March 4 slides). For some pair of selected vertices, let P(triangle|w,v) be the probability that the two vertices share at least one other neighbor. The probability that a vertex would be selected to be in S would be 1 - d(w) * 1/(2d(w))* P(triangle|w,v) = 1 - P(triangle|w,v). Thus, we can see that the probability is bounded by the number of triangles in a graph. More triangles mean that a vertex is less likely to be selected while less triangles means a vertex is more likely to be selected.

Lemma: The probability that a good vertex belongs to S union Gamma(S) is at least (1-exp(-⅙))/(1 - P(triangle|w,v))

Let v ge a good vertex with d(v) > 0. Consider vertex w, a neighbor of v with the lowest degree of those neighbors is marked. We know that it was marked with probability of at least 1 - P(triangle|w,v). If w is S, then we know v is a neighbor of w. From the first lemma, E happens with chance 1 - exp(-⅙). Thus, the probability v is in S or a neighbor of a vertice in S is (1 - exp(-⅙)) / (1 - P(triangle|w,v))

Theorem: The parallel MTIS algorithm runs in O(logxlogn) using O(n + m) processors where x is the number of triangles in the graph.

Each round runs on O(logn) time using O(n + m) processors. Since vertices are eliminated if either marked or as a part of a triangle, that means the expected number of edges eliminated during the iteration is a constant fraction of current number of triangles. If a marked vertex is part of a triangle, the triangle will be eliminated; if not, only the marked vertex will be eliminated. Thus, the algorithm eliminates more vertices more quickly if there are more triangles while non-triangle vertices are eliminated at a constant rate.

https://www.dropbox.com/sh/38iv1piy8we3llk/AAD7TCXgF_iySb6rr3uy9mXda?dl=0