1 Independent Set

Previously we showed that finding a greedy maximal independent set in a graph $G = (V, E)$ is ¶-complete. Today, we show that we can indeed find some maximal independent set.

We use the following framework. Start with S being the empty set.

1. Find some independent set (not necessarily maximum) X. Add X in set S.
2. Remove the neighbors of S.
3. Repeat until graph is empty.

The set S is a maximal independent set.

Were we able to remove a constant fraction of the graph, in each time through steps 1 and 2, we would finish in $O(\log n)$ iterations.

The following randomized parallel algorithm for step 1 ensures such that steps 1 and 2 remove a constant fraction of the graph with decent probability.

1. For each node u, place it in S with probability $1/(2d(u))$, where $d(u)$ is the degree of u.
2. For any edge (u, v), remove the smaller degree node from S breaking ties randomly.

Lemma 1 *The expected number of edges incident to $X \cup N(X)$ is $\Omega(|V|)$.*

Proof:

A "bad vertice" is a vertice that has degree less than 2/3 of its neighbors. A "bad edge" is an edge where both its neighbors are bad.

Question 1: Prove that the number of bad edges is at most $|E|/2$.

An edge is in S

□

Note that we have removed a constant fraction of the graph on expectation.

To bound the work, we let C be the expected number of iterations for the number of edges to reduce by some constant factor, say 1/2. We can bound this by a constant using Markov's inequality to arge that the probability that we reduce by a constant is at least some constant, and then noting that we need some constant number of "successes" to reduce by any fixed constant factor. With this, we get the following recurrence on the work.

$$W(n) = W(|E|/2) + C|E|.$$

To bound the time, we consider that the number of successful (reducing) iterations to reduce the number of edges to a constant is $O(\log |E|)$. By a chernoff bound and Markov's inequality, this many successes will happen with high probability in $O(\log |E|)$ iterations.

2 Sorting

2.1 Quicksort

The quicksort algorithm is quite easy to parallelize. A random element is chosen, then each element compares itself to this random element and marks itself as in the left or right list. They can be compacted using two parallel prefixes.

This algorithm terminates in $O(\log n)$ iterations with high probability. Each iteration can be done in $O(\log n)$ time. Thus, the time is $O(\log^2 n)$. The work can be made optimal $O(n \log n)$ by using $n/\log n$ processors in each iteration.

In the CRCW model, one can develop an $O(\log n)$ algorithm for this problem by avoiding the parallel prefix.

Question 1. How do you do this using the CRCW Common model?

For the EREW model, it is quite difficult but still possible.

2.2 Radix Sort

Quicksort had to proceed only by comparing elements. If instead, we are allowed to examine keys one digit at a time, we can implement radix sort. That is, we sort digit by digit, starting with the least significant bit.

Essentially, one just needs to implement a stable sort (i.e., break ties by preserving the current order.)

In each iteration, one can implement a stable sort using a prefix sums for each possible value of a digit. The prefix sum tells us where to go in the array.