Lindsey Bieda and Joe Frambach
Reduction and Parallel Problems
11.04.2011

21. The input to the Fixed Hamiltonian path problem is an undirected graph G and two vertices x and y in G. The problem is to determine if there is a simple path between x and y in G that spans all the vertices in G. A path is simple if it doesn't include any vertex more than once. Show that if the Fixed Hamiltonian path problem has a polynomial time algorithm then the Hamiltonian cycle problem has a polynomial time algorithm.
HINT: I think it is easier to do this problem if you don't restrict yourself to a many-to-one reduction, that is, feel free to call the path procedure multiple times.

HAMCYCLE $\leq$ FIXED-HAMPATH

The graph has a Hamiltonian Cycle if there is a fixed Hamiltonian Path starting and ending with adjacent vertices.

```
Program HAMCYCLE:       Read Graph G.
    For each edge with vertices x and y,
        If FIXED-HAMPATH(G,x,y) returns true,
            Return true.
    Return false.
```

It should be obvious from the above that HAMCYCLE will return true if and only if for some pair of adjacent vertices, there is a FIXED-HAMPATH.

25. We consider a generalization of the Fox, goose and bag of beans puzzle
`http://en.wikipedia.org/wiki/Fox,_goose_and_bag_of_beans_puzzle`
The input is a graph G an integer k. The vertices of G are objects that the farmer has to transport over the river, there are an edge between two objects if they can not be left alone together on the same side of the river. The goal is to determine if a boat of size k is sufficient to safely transport the objects across the river. The size of the boat is the number of objects that the farmer can haul in the boat. Show that this problem is NP-hard using a reduction from one of the problems that either I showed was NP-hard in class, or that you showed was NP-hard in the homework. So I am letting you pick the problem to reduce from here. You should take some time to reflect which problem would be easiest to reduce from.

VERTEX-COVER $\leq$ FOX-GOOSE-BAG

Program VERTEX-COVER:
    Read Graph G, int k.
    Return FOX-GOOSE-BAG(G, k).

Since vertex cover decides if there is a vertex cover of size k in G, we know that our boat is able to contain all the things that are destructive to one another. If we do not put the entire vertex cover in the boat then we are leaving an edge on the shore. Therefore, VERTEX-COVER(G,k) will only return true if and only if the boat is sufficient to carry the given items.

4. Design a parallel algorithm for the parallel prefix problem that runs in time O($log\ n$) with $n/log\ n$ processors on a EREW PRAM.

Our algorithm requires three passes, each of O($log\ n$) time. Each processor is given $log\ n$ items from the input. Each processor sums these items (in $log\ n$ time). We now have $n/log\ n$ resulting partial sums. In the second phase, we calculate the prefix sums of the previously calculated partial sums. This takes O($log\ \frac{n}{log\ n}$) time since we can use the algorithm discussed in class (which I will not describe here since it is in the class notes). In the third phase, we add the sums calculated in phase two with the prefix sums of the $log\ n$ numbers assigned to each processor. Each processor requires O($log\ n$) time to perform this operation.

5. Give an algorithm that given an integer n computes $n!$, that is n factorial, in time O($log\ n$) on an EREW PRAM with n processors. Make the unrealistic assumption that a word of memory can store arbitrarily large integers.

The call to $n!$ is split into two parallel calls, $\prod_{i=1}^{n/2} i$ and $\prod_{i=n/2}^{n} i$. This split occurs $log\ n$ times until the tree has n leaves and a height of $log\ n$. We now have n processors multiplying two integers in O(1) time. The next level of the tree continues in this fashion. The total time for the factorial program is O($log\ n$).