Randomization in Computing Notes - 10/27/11
Written up by: Michael Fagan

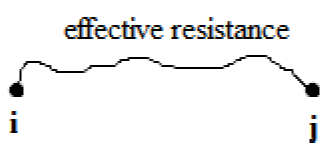
Help for Homework Problem #9

Let G(V,E) be any undirected graph

We want to calculate the travel time across the graph.

Think of each edge as one resistor of 1 Ohm.

Say we have two nodes: i and j



Let the effective resistance between i and j = $R_{ij} = \frac{1}{\sum \frac{1}{r_x}}$ where r_x is the resistance of a path between i and j. Therefore R_{ij} is one over the sum of one over the resistance of each path from i to j.

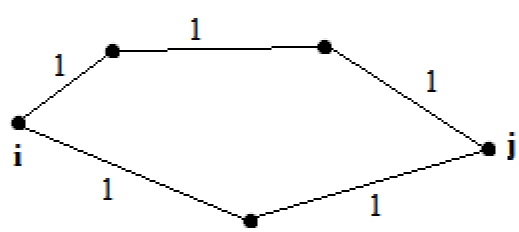
Fact: The commute time between i and j is C_{ij}

$C_{ij} = 2mR_{ij}$ where $m = |E|$

Let the resistance of the graph be $R = \text{Max } R_{ij}$

Fact: The expected cover time $C(G) = O(mR\log n)$

Fact: The R_{ij} for any 2 nodes $\leq$ the length of the shortest path between i and j. For example:



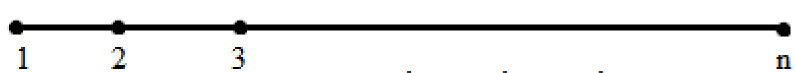
$R_{ij} = \frac{1}{\frac{1}{3}+\frac{1}{2}} = \frac{6}{5} \leq 2$

Fact: For a d-regular graph with n nodes, the diameter $\leq \frac{n}{d}$

Using all of these facts together, you can solve problem number 9.

Example: Using the facts for a 2-SAT

Modeled as a random walk on a graph



$m = n$
$R = n$
Thus $C(G) = O(n^2\log n)$ Note: this is worse than with Markov chains, but this method will result in a better result with d-regular graphs, as in the HW

Prefix Calculations

Input: $k_1, k_2, \ldots k_n \in \Sigma$

Output: $k_1, k_1 \oplus k_2, k_1 \oplus k_2 \oplus k_3, \ldots, k_1 \oplus k_2 \oplus k_3 \ldots \oplus k_n$

Here $\oplus$ is any binary, associative, and unit time operation. Recall that if $\oplus$ is associative then, for any 3 elements x,y,z it should be true that $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$.

Examples:

1. $\Sigma = \mathbb{R}$ and $\oplus$ is addition
2. $\Sigma = \mathbb{R}$ and $\oplus$ is multiplication
3. $\Sigma =$ 2x2 matrices and $\oplus$ is matrix multiplication
4. $\Sigma = \mathbb{R}$ and $\oplus$ is Min

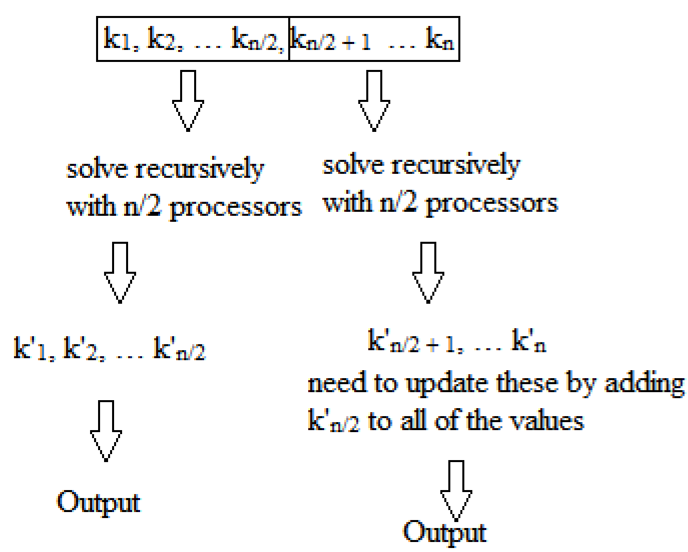
The best sequential run time = S = n – 1

Algorithms for logarithmic time prefix calculations

Algorithm 1

P = n CREW processors

Split your input in 2. Then recursively perform prefix computations on each half. Overall the process works like:



Note: $k'_2 = k_1 + k_2$
$k'_3 = k_1 + k_2 + k_3$
and so on for all to k'_n

Analysis of this algorithm:

Let T(n) be the runtime on any input of size n using n processors.

Then $T(n) = T\left(\frac{n}{2}\right) + O(1) = O(\log n)$
$PT = O(n\log n) \neq O(n)$ therefore the algorithm is not work optimal.

A work optimal algorithm:

$P = \frac{n}{\log n}$ CREW PRAM processors

Assign log n elements to each processor such that the 1st log n elements go to p_1, 2nd log n elements go to p_2, etc.

1. For $1 \leq i \leq \frac{n}{\log n}$ in || do
   Processor i computes the prefix sums (note: here sum refers to the $\oplus$ operator) of its elements

   Let the results be $k'_1, k'_2, \ldots k'_{\log n} \ldots k'_n$

2. Perform the prefix calculation on $k'_{\log n}, \ldots k'_{2\log n}, \ldots k'_n$

   Let the results be $k''_{\log n}, \ldots, k''_{2\log n}, \ldots k''_n$

3. For $1 \leq i \leq n$ in || do
   Processor i pre-adds $k''_{(i-1)\log n}$ to every value it computed in step 1

Analysis:

Step 1 takes log n time
Step 2 takes $O\left(\log \frac{n}{\log n}\right) = O(\log n)$ time
Step 3 takes log n time
Thus total time = $O(\log n)$
Thus, this algorithm is asymptotically optimal.

Example:

Input: $X= k_1, k_2, \ldots k_n$ (a sequence of elements) and an integer y

Output: a rearrangement of X where all the elements $\leq$ y appear first, followed by all other elements

An example:
y = 10

8	11	7	3	15	9	16	2	4

Output:

8	7	3	9	2	4	11	15	16

The algorithm:

Use a Boolean array A[1:n] such that
A[i] = 1 if $k_i \leq y$
= 0 otherwise
for all $1 \leq i \leq n$

for the example:

1	0	1	1	0	1	0	1	1

Now perform a prefix sums (where sum refers to addition) on the array

1	1	2	3	3	4	4	5	6

Now you can use these prefix values as unique addresses for the elements that are $\leq$ y and place them in successive cells. We can use another similar prefix computation to place the remaining elements in successive cells.

Can we do prefix calculations in O(1)? Nope!

(Beam and Hastäd 1985)
Theorem: Computing the parity of n bits needs $\Omega(\frac{\log n}{\log\log n})$ time on the CRCW PRAM given only a polynomial number of processors.

(Cole and Vishkin 1983)
We can solve the prefix additions problem in $O(\frac{\log n}{\log\log n})$ time using $\frac{n\log\log n}{\log n}$ arbitrary CRCW PRAM processors, provided the elements are integers in the range $[1, n^c]$ for any constant c.

Example uses:

Sorting:

$k_1, k_2, \ldots k_n = X$

We can sort these in O(log n) time using $\frac{n^2}{\log n}$ CREW PRAM Processors.

This is done by giving each element n processors and letting them calculate its rank, and then outputting the elements in the order of their ranks.

Selection:

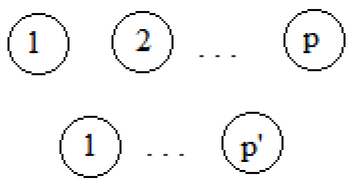
Input: $k_1, k_2, \ldots k_n = X$ and an i such that $1 \leq i \leq n$

Output: The i^{th} smallest element of X

Slow-down Lemma

Lemma: Let A be a || algorithm that uses p processors and runs in time T. The same algorithm can be run on p' processors in time $O(\frac{pT}{p'})$ provided p' $\le$ p.

Proof:



Old Machine

New Machine

Assign $\frac{p}{p'}$ processors of the old machine to each of the processors in the new machine. Each step of the old machine can be simulated in $\le \frac{p}{p'}$ steps in the new machine.

The runtime on this new machine is $\le \frac{p}{p'}\ T \le T\ \frac{p}{p'} + 1 = O\ \frac{pT}{p'}$.