15-492: Parallel Algorithms **Lecturer:** Guy Blelloch
Topic: String Processing **Date:** October 16, 2007
Scribe: Kyle Comer(kcomer@)

15.1 Substring problem

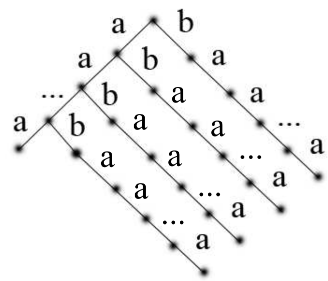
Input: String, s, of lenght n and a pattern, p, of length m
Output: All occurances of p in s

For example, the string could be Google's database of websites and the pattern could be a search phrase.

We want to preprocess s so that each search can be done in time $O(m)$ and many searches can be run in parallel. There is an algorithm that constructs a Finite State Machine from the pattern and then runs the input string through the FSM to determine all locations of the pattern. However, this runs in time $O(n)$, which is slower than we would like.

15.1.1 Trie data structure

We could construct a trie of all suffixes of the string. However, this data structure could be $O(n^2)$ large. Consider the string $s = aaa \ldots aaabaaa \ldots aaa$. This would generate a tree like the following.



15.1.2 Patricia Tree or suffix tree

To solve this problem, we can generate a Patricia Tree or suffix tree instead of a Trie. A Patricia Tree allows an edge to represent multiple letters. We can construct a Patricia Tree from a Trie by removing any node with only 1 child and merging the incoming edge and the outgoing edge. This tree has only binary nodes and leaves. With the relation that $|leaves| = |binary\ nodes| + 1$ we see that the space complexity is $2n - 1 \in O(n)$.
If we are able to build this tree, lookups will only take $O(m + \log n)$ time. How can we construct this tree? If we simply insert every prefix, this tree can be constructed is $O(n^2)$ time. However, we can do better than this.

15.1.3 Constructing Suffix Array

Instead of constructing a suffix tree, we will construct a suffix array. A suffix array is a sorted array of the suffixes. In our case, this is most easily represented as an array of points into the string. Naively, this can be constructed in $O(n^2 \log n)$ work by using quicksort on all the suffixes (since each comparison is potentially $O(n)$) or in $O(n^2)$ work using radix sort. Our goal is $O(n)$, which can be done. However, we will describe a simpler version which runs in $O(n \log n)$ work.
Input: String of length n
Output: Suffix array of pointers into the string
An interesting note about this algorithm is that the sequential version was solved using the parallel technique of contraction.

M	I	S	S	I	S	S	I	P	P	I	$
0	1	2	3	4	5	6	7	8	9	10	11

Figure 15.1.1: Sample array

Assume $ comes before all characters lexicographically.

The algorithm is:

1. Break the string into all possible (n) sets of 3 consecutive characters.

MIS	ISS	SSI	SIS	ISS	SSI	SSI	IPP	PPI	PI$	I$$	$$$
0	1	2	3	4	5	6	7	8	9	10	11

Figure 15.1.2: Suffix triplets

2. Group all triplets beginning at any index equivalent to 1 (mod 3) into an array.

3. Append all triplets begining at any index equivalent to 2 (mod 3) to the end of this array.

4. Radix sort the array and label them with their index. Now we have an array with ($2n/3$) elements sorted by their index.

Index	1	4	7	10	2	5	8	11
Triplet	ISS	ISS	IPP	I$$	SSI	SSI	PPI	$$$
Sort	4	4	3	2	6	6	5	1

Figure 15.1.3: Triplets sorted by radix

5. We do a recursive call on this array. (Numbers are treated as just characters)

6. To deal with the remaining elements (the ones equivalent to 0 (mod 3)) we can make use of the sorting that we just did. We can use radix sort on a pair of characters: the character at 0 (mod 3) and then the index of the character sequentially after it.

4	4	3	2	6	6	5	1	$

Figure 15.1.4: Recursive problem of size $2n/3$

44326651	4326651	326651	26651	6651	651	51	1
5	4	3	2	8	7	6	1

Figure 15.1.5: Solution to recursive problem

m5	s4	s3	p2
1	4	3	2

Figure 15.1.6: Sorting the 0 (mod 3) suffixes

7. Now we have a sorted array of length $2n/3$ corresponding to the suffixes of the elements equiavlent to 1 and 2 (mod 3) and a sorted array of length $n/3$ corresponding to the suffiex of the elements equivalent to 0 (mod 3). To obtain our answer, we merge these two arrays. The exact method for this will be covered in the next lecture.

M	I	S	S	I	S	S	I	P	P	I	$
-	5	8	-	4	7	-	3	6	-	2	1
1	-	-	4	-	-	3	-	-	2	-	-
5	4	11	9	3	10	8	2	7	6	1	0

Figure 15.1.7: Merging of 2 sorted suffix arrays