APC Overview

Isabel H.

Winter 2023/2024

1 Bootstrapping

Sections

1. Computing minimum spanning trees in linear time
2. Computing minimum cuts in $\tilde{O}(n^2)$ time

2 Randomized Search Trees

Sections

1. Computing minimum spanning trees in linear time
2. Computing minimum cuts in $\tilde{O}(n^2)$ time

3 Point Location

Sections

1. Computing minimum spanning trees in linear time
2. Computing minimum cuts in $\tilde{O}(n^2)$ time

4 Linear Programming

Sections

1. Computing minimum spanning trees in linear time
2. Computing minimum cuts in $\tilde{O}(n^2)$ time

5 Randomized Algebraic Algorithms

Main topics: Probabilistic checking: find out if a something is correct probabilistically

Sections

1. Checking matrix multiplication
2. Is a polynomial identically Zero?
3. Testing for perfect bipartite matchings
4. Perfect matchings in general graphs
5. Comparisons with other matching algorithms
6. Counting perfect matchings in planar graphs

5.1 Checking matrix multiplication

Task: check $\mathbf{AB} = \mathbf{C} \Leftrightarrow \mathbf{C} - \mathbf{AB} = \mathbf{0}$
Idea: Check each entry with probability $\frac{1}{2}$. Fails to find error with probability as high as $\frac{1}{2}$, but we can repeat it in an arbitrary time to get a sufficient low probability.

5.2 Is a polynomial identically zero?

Task: blackbox evaluating polynomial is given, find out if it is 0.

Theorem 5.1 (Schwartz-Zippel theorem).

6 Parallel Algorithms

Sections

1. Warm up: add two numbers
2. Models and basic concepts
3. List and trees
4. Merging and sorting
5. Connected components
6. (Bipartite) perfect matching
7. Modern/Massively parallel computation (MPC)

6.1 Warm up: add two numbers

Basic algorithm: carry-ripple
Problematic part: preparation of carry bits
Idea: kill/propagate/generate bit, compute that in parallel $\Rightarrow$ use binary tree

6.2 Models and basic concepts

Two models: logic circuits and PRAM (parallel random access machines)

6.2.1 Circuits

Circuits: Boolean circuits with AND/OR/NOT gates, connected by wires
$NC(i)$: set of all decision problems that can be decided in boolean circuit with $poly(n)$ gates of at most two imputs and depth of at most $O(\log^i n)$ depth
$AC(i)$: set of all decision problems that can be decided in boolean circuit with $poly(n)$ gates of potentially unbound fan-in and depth of at most $O(\log^i n)$ depth

Lemma 6.1. *For any k, we have $NC(k) \subseteq AC(k) \subseteq NC(k+1)$*

Def: $NC = \cup_i NC(i) = \cup_i AC(i)$

6.2.2 Parallel random access machines (PRAM)

p number of RAM processors, each with its own local registers and access to a global (shared) memory
Four variations: Exclusive/Concurrent Read Eclusive/Concurrent Write

Lemma 6.2. *For any k, we have $CRCW(k) \subseteq EREW(k+1)$*

We also have $NC = \cup_k EREW(k)$: PRAM can simulate circuits and vice versa

6.2.3 Some basic problems

Parallel Prefix: Task: Given array A of length n, compute B with $B[j] = \sum_{i=1}^{j} A[i]$, Can be parallized with binary tree idea
List ranking: Task: Given linked list by its content array $c[1..n]$ and successor pointer array $s[1..n]$, get all suffix sums $\Rightarrow$ Use algorithm based on idea of **pointer jumping**
List ranking with pointer jumping: $\log n$ iterations, each with two steps:

1. In parallel, for each $i \in \{1, \ldots, n\}$, set $c(i) = c(i) + c(s(i))$

2. In parallel, for each $i \in \{1, \ldots, n\}$, set $s(i) = s(s(i))$

Lemma 6.3. *At the end of the above process, for each $i \in \{1, \ldots, n\}$, c(i) is equal to its suffix sum*

6.2.4 Work-efficient parallel algorithms

Computational process can be viewed as a sequence of rounds, each rounds consits of number of computations indepentend of each other, computable in parallel
Total number of rounds: **depth** of computation; Summation of number of computaions: total **work**
Primary goal: depth small as possible, second goal: small total work

Theorem 6.4 (Brent's principle)**.** *If an algorithm does x computations in total and has depth t, then using p processors, the algorithm can be run in $x/p + t$ time*

Work-efficient algorithm: total amount of work of parallel algorithm is proportional to amount of work in sequential case.

7 Lists and trees

7.1 List ranking

Goal: Obtain algorithm with $O(\log n)$ depth and $O(n)$ total work