Computational Complexity

EECS 477

Lecture 20, 11/21/2002

Today

- Finish up with Fast Fourier Transform
- Start computational complexity
  - Chapter 12
    - Today everything before P and NP and such…
    - P and NP and such next week
  - We do not cover chapter 10 and 11 at all
    - No probabilistic algorithms
    - No parallel algorithms

DFT

- $w := \exp(-2\pi i/N)$
  - N-th roots of unity $(w^k)^N = 1$
  - Complex number $\exp(iu) = \cos(u) + i \sin(u)$
  - Taking squares get n/2 roots of unity
  - Fourier transform: what applications

  $a\hat{}[k] = \Sigma_{j=0..N-1}\ a[j]\ w^{kj}$, for all k=0..N-1
- Simple algorithm

  N times N-term sums = $\Theta(N^2)$

FFT

- Use the fact that

$\Sigma_{j=0..N-1}\ a[j]\ x^j = \Sigma_{j=0..N/2-1}\ a[2*j]\ x^{2j} + x\ \Sigma_{j=0..N/2-1}\ a[2*j+1]\ x^{2j}$

- Recursive procedure

```
fft(a[0..n-1]) {
    ye = fft((a[0],a[2],..,a[n-2]));
    yo = fft((a[1],a[3],..,a[n-1]));
    for(k=0; k<n/2; ++k) {
        y[k] = ye[k] + w^k yo[k];
        y[k+n/2] = ye[k] - w^k yo[k];
    }
    return y;
}
```

FFT

- $T(N) = 2\,T(N/2) + \Theta(N)$
  – Case 2 of the Master Theorem
- Fast algorithm
  $T(N) = \Theta(N \log N)$

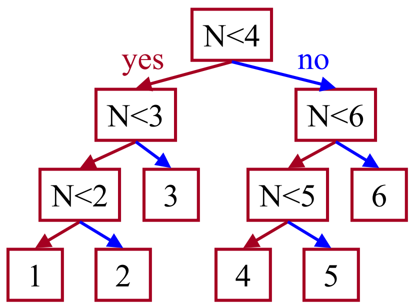
[0,1,2,3,4,5,6,7,8]							
[0,2,4,6]				[1,3,5,7]			
[0,4]		[2,6]		[1,5]		[3,7]	
[0]	[4]	[2]	[6]	[1]	[5]	[3]	[7]

Information-theoretic arguments

- Game of five questions
  – A person chooses an integer 1..32
    - We can always guess with five yes-no questions what the number is.
    - Prove it!
  – Can we do better than this?
    - No, when our friend cheats
    - Prove it!
  – Cheating = giving a counter-example

Game of two questions

- Decision tree
  - All possible data as leaves of the tree
    - Outputs = verdicts



Trees of height two would have no more than four leaves so we cannot use less than three questions.

Worst-case analysis

Average case analysis

- Runtime >= # of questions on a path
  - Average height of T := average *depth* of all the leaves in that tree
- Any binary tree with *k* leaves has an average height of at least *log k*

  h(k) := "the smallest possible sum of leaf depths"   *Need: h(k) >= k log k*
- $h(k) = \min_{0<i<k} [\ h(i) + h(k-i) + k\ ],\ k>1$
- $h(0) = 0,\ h(1) = 0$

Proof by induction

Sorting complexity

- List of N elements
  - Number of leaves
    - N! permutations
  - Tree with N! leaves has minimal depth of $\log(N!) = \log 1 + \log 2 + .. + \log N = \Theta(N \log N)$
    - See midterm solutions
  - Minimal average tree height is $\Theta(N \log N)$
    - Quicksort has optimal performance
  - Insertion sort, heapsort decision tree (book)

Adversary arguments (12.3)

- Finding maximum
  - O(N) easy
  - What about a lower bound?
    - Information-theoretic argument gives log N
  - At least (N-1) comparisons via adversary argument
    - Smaller in comparison loses a comparison
    - If less then there two that did not lose
      - By contradiction

Graph connectivity

- Graph with N vertices
  - Is it connected?
    - N^2 tests is enough
    - What is the lower bound?
    - Information-theoretic gives lower bound of 1 ☹
  - Daemon splits graph into two equal parts
  - There are $\Omega(N^2)$ edges in between these
    - Have to test all of them

Median

- Adversary argument for finding median
  - We know an algorithm in O(N)
  - Proof that less than 3(N-1)/2 comparisons is not enough
  - Daemon and an array T[1..N], N is odd
    - Assigns values
      - 1..N are low
      - 3N+1..4N are high
      - Follow the rules on the next slide
      - T is uninitialized at first

Median II

- Upon comparison T[x] and T[y]
  - If both uninitialized set T[x]=x, T[y]=3N+y
  - If one of T[x] and T[y] uninitialized
    - If it's the last uninitialized element set it to 2N
    - If T[x] is low set T[y] to high 3N+y, balance
  - If both initialized
    - Low-low, low-median -- lower lost a comparison
    - High-high, med-high – higher lost a comparison

    (N-1)/2 to init, less than N-1 is left for losing
    at least one non-median that has not lost

Linear reductions

- A is linearly reducible to B (A<=B) if the existence of a O(t(n)) algorithm for B implies the existence of O(t(n)) algorithm for A
- When both ways we get linear equivalence
  - Ex: SQR and MULT
    - $x^2 = x*x$
    - $x*y = ((x+y)^2 - (x+y)^2)/4$

Smoothness matters: see the book
f(bN) = O(f(N)),
for all integer b>=2

Decision problems

- TSP
  - Find the tour of the minumum cost
- Decision problem
  - For K, is there a tour of cost <=K
- P – class of decision problems that can be solved by a polynomial-time algorithm
- NP – non-deterministic polynomial time
  - Given a solution, it can be checked in polynomial time (like given a tour, check?)