Divide & Conquer

May 6, 2014 10:30 AM

Master Theorem

Let $a \geq 1, b \geq 1, c \geq 0$ be constants, $f(n) = n^c$ and $T(n)$ be defined on nonnegative integers by the recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Then

$$T(n) \in \begin{cases} \Theta(n^c) & \text{if } b^c > a \\ \Theta(n^c \lg n) & \text{if } b^c = a \\ \Theta\left(n^{\log_b a}\right) & \text{if } b^c < a \end{cases}$$

Linear Median Algorithm (BFPRT)

Runtime:

$$T(n) = \frac{n}{c}\text{Sort}(c) + T\left(\frac{n}{c}\right) + \frac{n}{c}\left(\frac{c-1}{2}\right) + T\left(\frac{n}{2c}\left(c + \frac{c-1}{2}\right)\right)$$

For $c = 3$:

$$T(n) = \frac{n}{3}\text{Sort}(3) + T\left(\frac{n}{3}\right) + \frac{n}{3} + T\left(\frac{2n}{3}\right) = n + T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) \in O(n \log n)$$

For $c = 5$:

$$T(n) = \frac{n}{5} \cdot 7 + T\left(\frac{n}{5}\right) + \frac{2n}{5} + T\left(\frac{7n}{10}\right) = \frac{9n}{5} + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right)$$

Induction hypothesis: $T(n) \leq \alpha n$

$$T(n) \leq \frac{9}{5}n + \alpha\frac{n}{5} + \alpha\frac{7n}{10} = \frac{(18 + 9\alpha)}{10}n$$

Need $\frac{18+9\alpha}{10} \leq \alpha \Rightarrow 18 + 9\alpha \leq 10\alpha \Rightarrow \alpha \geq 18$

$\therefore T(n) \in O(n)$

General

$$T(n) = \frac{n}{c}\left(\lceil \log_2 c! \rceil + \left(\frac{c-1}{2}\right)\right) + T\left(\frac{n}{c}\right) + T\left(\frac{n}{2c}\left(c + \frac{c-1}{2}\right)\right)$$

Induction hypothesis: $T(n) \leq \alpha n$

$$T(n) \leq Cn + \alpha\left(\frac{n}{c} + \frac{n}{2c}\left(c + \frac{c-1}{2}\right)\right) \leq \alpha n$$

$$C + \alpha\left(\frac{1}{c} + \frac{1}{2} + \frac{1}{4} - \frac{1}{4c}\right) = C + \frac{3}{4}\alpha\left(1 + \frac{1}{c}\right) \leq \alpha$$

$$C \leq \alpha\left(\frac{1}{4} - \frac{3}{4c}\right)$$

$$\alpha \geq \frac{\frac{1}{c}\left(\lceil \log_2 c! \rceil + \frac{c-1}{2}\right)}{\frac{1}{4} - \frac{3}{4c}}$$

$$\alpha \geq \frac{4\lceil \log_2 c! \rceil + 2(c-1)}{c-3}$$

$c = 5: \alpha \geq 18$
$c = 7: \alpha \geq 16$
$c = 9: \alpha \geq 15\frac{1}{3}$
$c = 11: \alpha \geq 15\frac{1}{2}$

With better median selection:

$$\alpha \geq \frac{4\,\text{SelectMed}(c) + 2(c-1)}{c-3}$$

Knuth: Minimum comparison selection

$$V_t(n) = n - t + (t-1)\lceil \log_2(n + 2 - t) \rceil$$

$$\text{SelectMed}(c) \leq V_{\frac{c+1}{2}}(c) = c - \frac{c+1}{2} + \frac{c-1}{2}\left\lceil \log_2\left(c + 2 - \frac{c+1}{2}\right)\right\rceil$$

$c = 5: \alpha \geq 16$
$c = 13: \alpha \geq 12$

Optimal Binary Search Tree

Use dynamic programming to compute $w(i,j)$

$w(i,j) = w(i,j-1) + p_j + q_j$

$\frac{1}{2}n^2(2) = n^2$ additions

A Faster Greedy Algorithm

May 15, 2014 10:27 AM

$T(n) = \log n + T(a) + T(n-a-1)$

Consider Case

$T(n) = 2T\left(\frac{n}{2}\right) + \lg n$

Try $T(n) = \alpha n - \lg n + 2$

$T(n) = \alpha n - 2\lg\frac{n}{2} + \lg n = \alpha n - 2\lg n + 2 + \lg n = \alpha n\ - \lg n + 2$

Case

If $a = 0$ each time

$$T(n) = \lg n + T(n-1) = \sum_{i=1}^{n} \lg(n-i) \in O(n \lg n)$$

Doubling (Galloping) Binary Search

Want the cost to be cheap if the split is near the ends, and $\lg n$ if the split is near the middle.

1) Look in the middle

    So we discover if solution is in the first half or in the second half.

2) WLOG assume solution is in the first half: start at 1 and keep doubling guess until overshoot

    Takes $\lg a$ steps

3) Finish with a binary search between the last 2 elements considered

    $\lg\frac{a}{2} = \lg a - 1$

$T(n) = 2\lg(a+1) + T(a) + T(n-a-1), \qquad a < \frac{n}{2}$

Self-Adjusting Data Structures: Linear Linked List

May 20, 2014 10:23 AM

Amortized Cost of MTF (Move to Front)

Model

Start with empty list
Scan to element requested
if not there insert at end (has cost $n+1$)
Apply heuristic (no cost)

Theorem

$S_{\text{Opt}} =$ Static Optimal
Under "insert of first request" start-up model cost MTF $\leq 2S_{\text{Opt}}$ on any sequence of requests.

Proof of Request

Consider search for just 2 elements (b, c)
# of searches: $k\, b's,\ m\, c's,\ k < m$
S_{Opt} - Put c in front of b.
Cost of b's & c's in searches for b's & c's
$(m+k)+k =$ cost of S_{Opt}
What order of requests maximizes MTF cost:
$c^{m-k}(cb)^k$ with cost $(m-k)+4k = m+3k \leq 2S_{\text{Opt}} = 2m+4k$

Splay Trees

May 27, 2014 9:59 AM

Linear Search

MTF cost $< 2S_{\text{opt}}$

< 2 Offline Opt

but just by swapping adjacent values

If offline algorithm is - on request for x we are told the next time x will be requested & cost of swapping 2 ajacent segments of size r is r swaps. Then we can access all elements in $\Theta(n \lg n)$ time. ? under other model cost is $\Theta(n^2)$

For any request sequence, start with elements ordered by 1st request. Amortized cost $O(\lg r_i)$ where r_i is the number of other elements requested until this one requested again. $\Theta\left(\lg \frac{1}{p_i}\right)$ or averaging over all $\lambda\Theta(H)$

But this is "offline" and usually we don't know the future for large sets especially if there ?. Searches for which we want smallest element $\leq$ one searched for.

Binary Search Trees

Use binary search trees.

We already know about optimal binary search tree.

Average search cost $\approx H$

Swap With Parent

If all elements have same independent probability of access then all binary search trees are equally likely. This is bad - about half of BSTs have a root with one child.

Average cost is $O(\sqrt{n})$

Rotate To Root by Single Rotations

If probabilities are independent cost $\leq 1.38 \times S_{\text{opt}}$

But, it is possible to construct arbitrarily long very bad sequences $\sim O(n)$ amortized cost.

SPLAY Trees

A different move to root method

- If root accessed, do nothing
- If child of root accessed, rotate to root
- If outside grandchild accessed then zig-zig

Working Set Bound

$\leq 3\lg(r+2)$

$r = \#$ of other different elements accessed since last time we accessed this

Operations

- Insert in usual way and splay new value to root
- Delete - remove in "usual" way - then splay parent of the node removed

Notation

$n(v)$ "size of node" $=$ number of nodes in subtree rooted at v

Includes v and all external nodes

So $n(root) = 2n + 1$

$r(v) = \text{rank}(v) = \lg(n(v))$

Full splay to root $=$ **step**

zig-zig, zig-zag, zig $=$ **substep**

General approach

"banking analogy"

Keep "virtual account" at each node.

Account not really kept in data structure.

In doing a splay we pay a certain number of units (cyberdollars) to be determined later.
3 cases:

- payment = splay work
- payment > work ⇒ deposit excess at nodes
- payment < work ⇒ make withdrawals

Invariant: Each node v has $r(v)$ cyber$ (before and after each step)

$$r(T) = \sum_{v \in T} r(v)$$

To preserve invariant we must make payment = splay work + $\Delta r(T)$

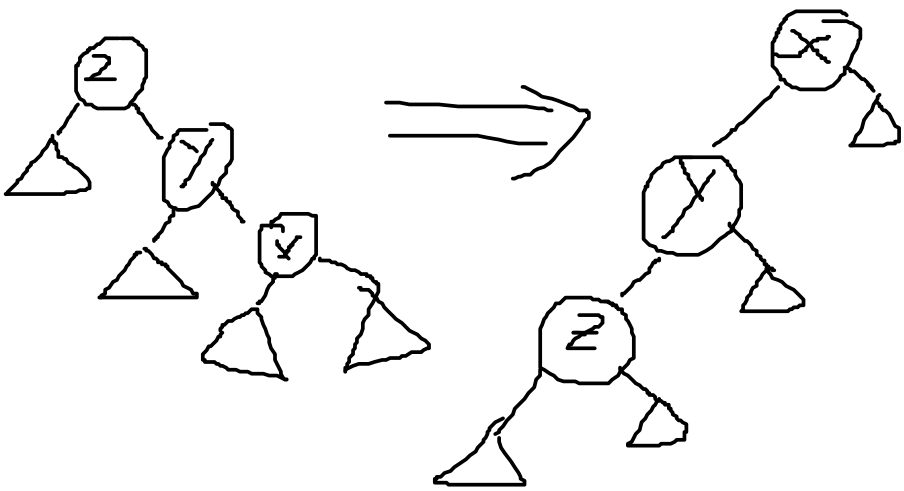
Lemma

δ-variation of $r(T)$ by a single subsetp is
$\delta \leq 3\big(r'(x) - r(x)\big) - 2$ for zig-zig or zig-zag
$\delta \leq 3(r'(x) - r(x)$ for a zig

Proof

Fact: $a > 0; b > 0, c = a + b \Rightarrow \lg a + \lg b \leq 2 \lg c - 2$

We do zig-zig step otherwise similar
size of each node = 1+ sum of sizes of children.
Only rank changes in zig-zig are to x, y, z



$r'(x) = r(z)$
$r'(y) \leq r'(x)$
$r(y) \geq r(x)$
So (1):
$\delta = r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) \leq r'(y) + r'(x) - r(x) - r(y) \leq r'(x) + r'(z) - 2r(x)$
$n(x) + n(z') \leq n'(x)$
So $r(x) + r'(z) \leq 2r'(x) - 2$ or $r'(z) \leq 2r'(x) - r(x) - 2$
This + (1) gives
$\delta \leq r(x') + (2r'(x) - r(x) - 2) - 2r(x) = 3\big(r'(x) - r(x)\big) - 2$

Theorem

T is a splay tree, root t, Δ total variation in $r(T)$ splaying x at depth d to root
$\Delta \leq 3\big(r(t) - r(x)\big) - d + 2$

Proof

Splaying x has $p = \left\lceil \frac{d}{2} \right\rceil$ substeps
Let $r_0(x) = r(x)$ =initial rank of x
  $r_i(x)$ = rank of x after ith substep
  δ_i = variation of $r(T)$ caused by ith substep
From lemma

$$\Delta = \sum_{i=1}^{p} \delta_i \le \left(\sum_{i=1}^{p} 3\big(r_i(x) - r_{i-1}(x)\big) - 2 \right) + 2 \le 3\big(r(t) - r(x)\big) - d + 2$$

Balance

wts. $= 1/\text{n}$
m accesses
amortized cost at most $3 \lg n + 1$ per access plus max(?account, change in entropy) → n lg n
$O(m + (m - n) \lg n)$ for n accesses

Static Optmial

$m = \sum q_i$
$\sum q_i \lg \left(\frac{m}{q_i}\right)$

Proof of Working Set

Give element i weight q_i/m (then $w = 1$)
amortized access time is $O\left(\lg\left(\frac{m}{q_i}\right)\right)$ and net change in balance/potential at most $\sum \lg \left(\frac{m}{q_i}\right)$

2 issues maybe 3

- Comparison based problems
- Lower bounds

Probabilistic/randomized algorithms

Sorting Using Quicksort

Quicksort:

choose "pivot" as middle value of subarray

expected # of comparisons $(2\lg 2)n\lg n$ averaging over all input orders

modify: choose pivot at random.

expected sort cost $(2\lg 2)n\lg n$ for any input sequence

Lower bound on sorting

$\lg(n!)$ comparisons is $n\lg n - n\lg e$ comparisons

Finding max by comparisons need at least $n-1$ comparisons

$n-1$ elements must be "disqualified" as max and each comparison disqualifies at most 1 element.

Obvious algorithm does this optimally.

Aside

Scan elements and compare with largest seen so far. $n-1$ comparisons

But now many "replacements" of max do we see?

Perhaps only 1

Perhaps n

How many if all input orders equally likely?

Expected number of new maxes

$$1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}$$

$$\sum_{i=1}^{n}\frac{1}{i}=H_i\approx\ln n+\gamma+o(1)$$

Finding 2nd largest - worst case

Pair elements

Keep pair winners till we have the max at top of balanced tree $n-1$ comparisons.

2nd largest lost directly to max. So $\lg n$ candidates - scan for max of those

So worst case $n+\lg n-1$ comparisons

But, how many comparisons necessary in worst case?

Lower bound: $n+\lg n-1$ comparisons. Necessary in worst case for any algorithm.

Finding k^th largest / Median Finding

Give a lower bound ~ worst case

*

|

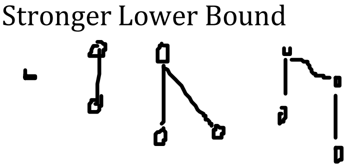
*

"Declare" this element is larger than any other still in system

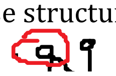
1 up for 2 comparisons - can do $\frac{n}{2}-1$ times then must ? max of the rest $\frac{n}{2}-1$

$$\frac{3}{2}n-2$$

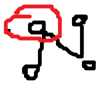
Stronger Lower Bound



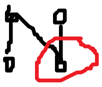
These structures can be formed as the alg likes

1) 

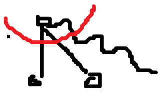
These structures can be formed as the alg likes

1) 

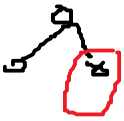
1 up for 2 & next time we get



1 don for 2
⇒ 2n bound

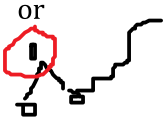
2) 

1 up for 3
Next time get

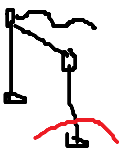


1 down for 1
⇒ 1 up 1 down - 4 comparisons
⇒ 2n bound

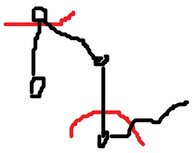
or



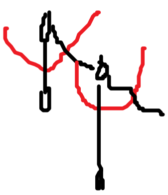
1 down for 2
next time
1 up for 2

3) 

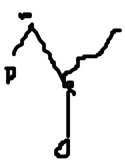
1 up for 3, 1 down for 1
⇒ 1 up 1 down for 4

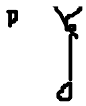


1 up 1 down for 4

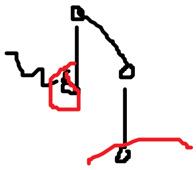


2 up for 4 and next time

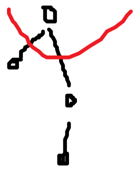




2 down for 3
2 up, 2 down for 1
$\Rightarrow \frac{7}{4}n$



2 down for 3
then next time



2 up, 2 down for 7

Any of these situations could happen. All lead to lower bound of at least $\frac{7}{4}n = 1.75n$
worst case median finding takes αn comparisons $\alpha \in (2,3)$

How about expected case?

Method you know
"one armed quicksort"
$(2 + 2\lg 2)n$ comparisons
$3.4n$ comparisons
But how do we do better
Floyd & Rivest

Take a sample size $(n^{\frac{7}{8}})$
Sort it and find 2 elements
- one above median (high)
- one below median(low)

So that
high - almost certainly $>$ median of entire set
low - almost certainly $<$ median of entire set
& expect # of elements in the entire set between high & low is not too large

Then scan through rest of values. compare with high
If $<$ high, compare with low
Count # above high (below low) & keep values that are in between
So if true median lies between high & low we have a selection problem on these. sort and find it.

Choosing numbers
Take sample size (say) $n^{\frac{7}{8}}$
Take high/low ? sample of rank $\frac{n^{\frac{7}{8}}}{2} \pm n^{\frac{1}{2}}$
Then we expect about $\frac{3}{2}n$ comparisons & some number between
we expect $n^{\frac{1}{8}} \times n^{\frac{1}{2}} = n^{\frac{5}{8}}$ in between
unlikely more than $n^{\frac{7}{8}}$ between high & low
if so ? & repeat answer

$\frac{3}{2}n$ or so comparisons

Either of 2 problems could occur

i. High & low don't bracket median
ii. Too many in between

Discrepancy Minimization

June 10, 2014 10:06 AM

Discrepancy Minimization

Given a set system $(X, \mathcal{S})$

X a set, $\mathcal{S} \in 2^{2^X}$,

$\mathcal{S} = \{S_1, \ldots, S_n\}, S_i \subseteq X$

Formally, let $\chi: X \to \{-1, 1\}$ (a colouring of the elements of X)

$$\text{Disc}_\chi(X, \mathcal{S}) = \max_{S \in \mathcal{S}} \left| \sum_{i \in S} \chi(i) \right|$$

$$\text{Disc}(X, \mathcal{S}) = \min_\chi \text{Disc}_\chi(X, \mathcal{S})$$

Applications:

The cell probe complexity of dynamic range counting

Examples

1) $S_i \cap S_j = \emptyset \ \forall i, j$

   $\text{Disc} = 0$ or 1

2) A complicated set system with discrepancy 0

   $X = \{x_1, \ldots, x_{\frac{n}{2}}\} \cup \{y_1, \ldots, y_{\frac{n}{2}}\}$

   $\mathcal{S}_X =$ all subsets of $\{x_1, \ldots, x_{\frac{n}{2}}\}$

   $S_i \in \mathcal{S}_X$, let S_i' be obtained by replacing x's with y's.

   $\mathcal{S} = \{S_i \cup S_i' | S_i \in \mathcal{S}_X\}$

   $\text{Disc}(X, \mathcal{S}) = 0$

Goal

Prove $\text{Disc}(X, \mathcal{S}) \le f(m, n)$, $|X| = n, |\mathcal{S}| = m$

Fact

There exists $(X, \mathcal{S})$ with $m = n$ such that Disc $> \Omega(\sqrt{n})$

Trivial upper bound : Color all elements in X with the same color. $\forall S$, #red − #blue $= |S| \le n$

Idea

Colour randomly

$\forall\, i \in X, \quad \Pr[\chi(i) = 1] = \Pr[\chi(i) = -1] = 0.5$

Analysis

$$\text{Let } \chi(S_j) = \left| \sum_{i \in S_j} \chi(i) \right|$$

Let $P_j = \Pr[X(S_j) > c]$

$$\Pr[\exists j \text{ s.t. } \chi(S_j) > c] \le \sum_{j=1}^{m} P_j \overset{\text{want}}{\le} \epsilon$$

ϵ is some small probability. Want $P_j \ll \frac{1}{m}$

Aside: Random Walk on a Line

$$\text{At each } t, X_t = \begin{cases} -1 & \text{with probability } \frac{1}{2} \\ 1 & \text{with probability } \frac{1}{2} \end{cases}$$

Position X after n steps

$$= \sum_{t=1}^{n} X_t$$

What is $\Pr[|X| > c]$?

Attempt 1: Markov's Inequality

$$P(X \geq c) = \frac{E(X)}{c}, \qquad X > 0$$

Attempt 2: Chebyshev's Inequality

$$P(|X - E(X)| \geq c) \leq \mathrm{Var}(X)$$

$$X = \sum_{t=1}^{n} X_t, \qquad \mathrm{Var}(X) = \sum_{t=1}^{n} \mathrm{Var}(X_t)$$

$$\mathrm{Var}(X_t) = E(X_t^2) - \left(E(X_t)\right)^2 = 1 - 0 = 1$$

$$\Rightarrow \mathrm{Var}(X) = n \Rightarrow P(|X| \geq c) \leq \frac{n}{c^2}$$

Can model $X(S_j)$ as random walk with $|S_j|$ steps since $\chi(i)$ is uniform random.

So $P_j \leq \frac{|S_j|}{c^2} \leq \frac{n}{c^2}$

Want $\frac{n}{c^2} = \frac{1}{m^2} \Rightarrow c = m\sqrt{n}$

Attempt 3: Chernoff bound

$$\text{If } X = \sum X_i, \qquad X_i = \{-1, +1\}$$

$$P(|X| \geq c) \leq 2 \cdot e^{-\frac{c^2}{2n}}$$

Want $2 \cdot e^{-\frac{c^2}{2n}} = \frac{1}{m^2} \Rightarrow -\frac{c^2}{2n} = \ln\left(\frac{1}{2m^2}\right) \Rightarrow c = \sqrt{2n \ln 2m^2} = O\left(\sqrt{n \log m}\right)$

Remarks

- Disc $\leq 2t$, t is the degree, the maximum number of sets an element is part of
  - Beck-Fiala Theorem
  - Conjecture: Disc $\leq O\left(\sqrt{t}\right)$
- Disc $\leq O\left(\sqrt{t} \log m \log n\right)$
- More general Chernoff bounds available

NP-Completeness

June 12, 2014 10:12 AM

Recursive Functions

Allowed operations:

- Increment variable
- Set to 0
- Test if zero
- Branch on condition - or - make procedure calls

NP-hard problems

June 19, 2014 10:07 AM

- Our case is a special case which we can solve quickly
  Similar notion - we may have a 2 parameter problem taking time exponential in one but polynomial in the other
- Parameterized complexity
- Randomization (& derandomization)
- "Standard" approach have a ply time algorithm guaranteed to find a solution "close" to optimal.

2 SAT

Keep 2 copies of the expression & auxiliary data
For each variable x_i and its negation $\neg x_i$, keep doubly linked list of clauses where x_i occurs and one where $\neg x_i$ occurs.
Also, for each variable, keep "flag" (T, F, ?)

Run "natural algorithm" with
    $x_i \equiv T$ on one copy
    $x_i \equiv F$ on the other
each process grows at same "speed"
Continue until one:
    successful - makes its decisions permanent, set the other the same and recurse
    unsuccessful - make permanent the decisions of the one still running

MAX SAT

3 literals per clause
Try to satisfy as many clauses as you can
|Take a random guess assignment of all variables
How many clauses do we expect to be true?
Probability of any one clause being true is 7/8
Expected # true 7/8 * n

P Reductions

June 26, 2014 10:11 AM

To prove L is NP-complete

- Show it is in NP
- Show solving arbitrary instance of known NP-complete problem K can be done in poly time if L is in P

Usually

Reduction

Given arbitrary instance of K, in poly time transform instance of K to one of L (which is in L iff it is in K).

Non-deterministic Turing Machine - Poly Time
↓
SAT
↓
3-CNF-SAT
↓ ↓
Clique Subset Sum

Clique

Does graph G on n nodes have a clique of size k?
Clique: complete subgraph
In NP: Guess k nodes. Show there is an edge between each pair of them.
Reduce 3-CNF to Clique:
Given

$$\phi = \prod_{r=1}^{k} \phi_r \quad \text{- A 3CNF formula}$$

$$c_r = l_1^r \vee l_2^r \vee l_3^r$$

Create a graph $G = (V, E)$
  l_i^r is vertex v_i^r in G
  edge $v_i^r v_j^s$ $(r \neq s)$ iff l_i^r and l_j^s are compatible (i..e. NOT negations of each other)

Claim: Graph has a k-clique iff ϕ is satisfiable.

Subset Sum: $\langle S, t \rangle$

Given set of positive integer S, is there a subset S' of S whose sum is t?

i) Clearly Subset Sum is in NP.
  Guess a subset of S and show its sum is t

ii) "Translate" an arbitrary* 3-CNF problem to a Subset Sum problem
  * Arbitrary with the following restrictions:
    Consider a 3-SAT ϕ with n variables
    - Each which actually occurs as x_i and $\neg x_i$ in ϕ_i
    - No clause with both x_i and $\neg x_i$

  Now we will create a Subset Sum Problem from ϕ.
    This will give S and t
  Elements in S will be $n + |C|$ digit numbers. The elements of S:
    $\forall x_i$ of ϕ, 2 integers - one for x_i, one for $\neg x_i$
    $\forall c_i$ of ϕ - 2 integers to handle issue of more than one literal in a cluase being true
    WLG n variable digits come first $t = 1^n 4^c$
    $\forall$ variable x_i:
      v_i, v_i' - 1 digit in position x_i
        1 digit in position c_j: in v_i if x_i in c_j, in v_i' if $\neg x_i$ in c_j

$\forall c_i, S$ has s_j, s_j' with 0 everywhere except position of c_i is 1 in s_j and 2 in s_j'

Without the s_i numbers, a satisfying assiment correspondes to subset with sum $1^n\{1,2,3\}^c$

With s_i number we can add 1 by taking s_j, 2 by taking s_j', or 3 by taking both.

Looking carefully we see a subset sum of size $t = 1^n 4^{|C|}$ iff formula is satisfiable. Do arithmetic base 10

Knapsack Problem

Given n objects each with size s_i and weight w_i choose a subset with $\sum_{\text{subset}} s_i \leq k$ having maximum value (weight).

Optimizing problem but NP-hard.

Proof: Let $s_i = w_i \forall i$. Let $t =$ knapsack size.

The question can we achieve value t is subset sum.

Parameterized Complexity

July 3, 2014 9:56 AM

Some Problems

Vertex Cover

Input: An undirected graph $G = (V, E)$ and integer $0 \leq k \leq |V|$
Question: Is there a subset $V' \subseteq V$ such that $|V'| \leq k$ and for each edge $\{u, v\} \in E$, at least one of u, v is in V'?

Independent Set

Input as above.
Question: Is there a subset $V' \subseteq V$, $|V'| \geq k$ such that for all $u, v \in V$, $\{u, v\} \notin E$

Clique

Question: Is there a $V' \subseteq V$, $|V'| \geq k$ such that for all $u, v \in V'$, $\{u, v\} \in E$

Parameterized Problems

A parameterized (decision) problem is a language $L \subseteq \Sigma^* \times N$ for finite alphabet Σ. We call the second component the **parameter**.

Fixed-Parameter Tractable (FPT)

A problem L is fixed-parameter tractable if there exists an algorithm determining if $(x, k) \in L$ with worst-case running time $f(k) \cdot n^{O(1)}$ where

- f is computable
- n is the length of the binary string encoding of x and k

Parameterized Reduction

For two parameterized problems L_1, L_2, L_1 reduces to L_2 by a parameterized reduction if $\exists$ functions f and g mapping k to N and function h mapping $\Sigma \times N$ to Σ^* such that h is computable in time $g(k) \cdot n^c$ for some constant c and $(x, k) \in L_1$ iff $\big(h(x, k), f(k)\big) \in L_2$

Reductions

Independent Set $\leftrightarrow$ Clique
Keep k the same, invert the edges

Vertex Cover $\overset{?}{\rightarrow}$ Independent Set
Transform $k \rightarrow |V| - k$, but this depends on $|V|$, not just k so is not a valid parameterized reduction.
Vertex Cover is fixed-parameter tractable (FPT = W[0])
Independent Set is W[1]

- $W[1] \supseteq W[0]$

Parameterized Algorithms

July 3, 2014 10:31 AM

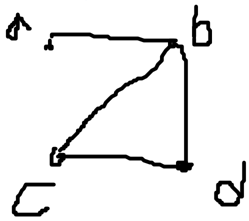
1. Search tree exploration
2. Data reduction (kernelization)
3. Treewidth
4. Colour-coding

Bounded Search Tree

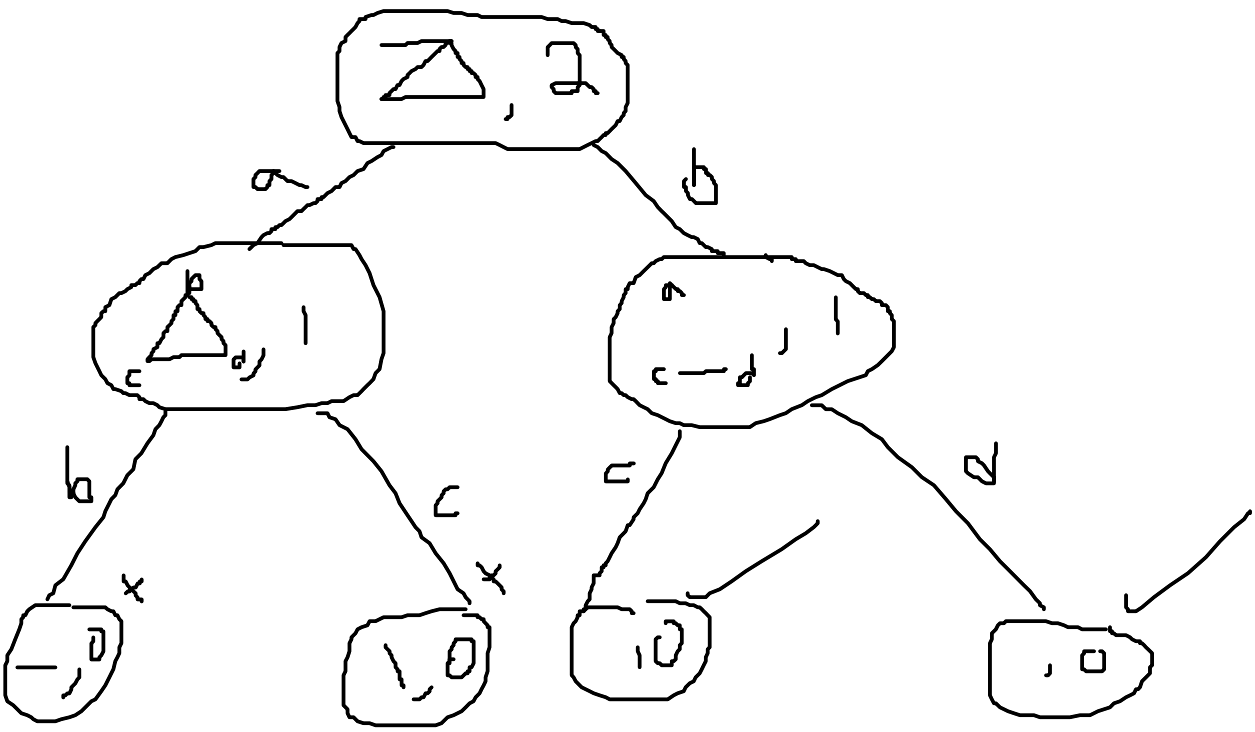
Bounded height and fan-out: $O\left(\text{fan-out}^{\text{weight}} \cdot \text{cost of processing a node}\right)$
fan-out: A function of k only
weight: A function of k only
cost of processing a node: $n^{O(1)}f(k)$

Vertex Cover

Example



At each node in the tree, pick an edge and branch on which vertex on that edge is in the cover.



In this case, fan-out=2, depth=k

Independent Set

Try: At a node, pick a vertex and branch on picking that vertex or any of its neighbours. But the branching factor isn't bounded as a function of k. If the degree was bounded this would work. With planar graphs, there is always a vertex with degree ≤ 6, so keep picking one with that property.

Kernelization

Max Sat

Can at least k out of m clauses be satisfied?

If $k \leq \frac{|m|}{2}$, answer yes.

We know $\frac{|m|}{2} < k$

Vertex Cover

Rule 1: Remove isolated vertices, keep k
Rule 2: Re degree-one vertex v with neighbour w, remove w and all incident edges, reduce k by 1
Rule 3: For every vertex v of degree greater than k, remove v and all incident edges, reduce k by 1

Claim: If the reduced graph is a yes-instance, it is small.
If big ⇒ answer "no". If small ⇒ brute force FPT.
There are at most $k^2 + k$ vertices in a "yes" instance (k vertices in cover, each with degree ≤ k)

Knapsack Problem

July 8, 2014 10:11 AM

0/1 Knapsack

Given n objects of the integer weights w_i and values v_i, choose a subset of total weight $\leq$ w that maximizes the value.
or as a decision problem - can we get total value $\geq$ v ?

- This problem is in NP-Complete (decision version)
- let $w_i = v_i$ then it is subset sum
- similarly, optimization problem is NP-Hard

Fractional Knapsack

Can take any fraction of any "object".
Take values per unit weight, i.e. $x_i = \frac{v_i}{w_i}$
Sort x_i's into decreasing order.
for i = 1 till done
    take as much of object i as you can.
This runs in $O(n \log n)$

Better Algorithm

Find median x_{med}

- Can I take all of substances of relative value x_{med} or higher?
- If NO then will not take any that are less valuable then x_{med}
  - so recurse on valuable half
- If YES then take all of the valuable $\frac{n}{2}$ subsets, can take more
  - so recurse on less valuable half

So with either case runtime $T(n)$
$T(n) = \Theta(n) + T\left(\frac{n}{2}\right) = \Theta(n)$

Solving 0/1 Knapsack Algorithm

Clearly can do this in $O(n2^n)$ time
We are given the objects in arbitrary order
$B[k, w] =$ max value on first k items with weight exactly w (if this does not exist, NaN)
So $B[0, w] = 0$

$$B[k, w] = \begin{cases} B(k-1, k) & \text{if } w_k > w \\ \max(B(k-1, w), B(k-1, w-w_k) + v_k) & \text{otherwise} \end{cases}$$

We are going to consider the objects in order, so when we come to element k we only need $B(w)$, best value with weight = w choosing only from elements $1, \dots, k-1$
Assume max weight is $\mathcal{W}$.

```
for w ← 0 to 𝒲 do
    B[w] ← 0
for k ← 1 to n do
    for w ← 𝒲 down to w_k do
        if B[w − w_k] + v_k > B[w]
            then B[w] ← B[w − w_k] + v_k
```

This method takes $O(n\mathcal{W})$ time and $O(\mathcal{W})$ space.
Note: we can get the optimal choice. Each time we update $B[w]$ keep track of what we added and which $B[w - w_k]$ we used.
This is NOT a poly time algorithm in the # of bits of input as encoding $\mathcal{W}$ takes $\theta(\lg \mathcal{W})$ bits.
Such a solution is called **pseudopolynomial**.

Optimization

July 8, 2014 11:02 AM

New coping method for optimization problems: do as well as you can and have some bounds you can guarantee.
Approximation algorithm with guaranteed approximate.

$\rho(n) \geq \max\left(\frac{c}{c^*}, \frac{c^*}{c}\right)$

where $c =$ cost of our solution, $c^* =$ cost of optimal solution to this instance. We don't know c^*.

Ideal Solution

Approximation Scheme: taking into account a parameter ϵ and getting $(1+\epsilon)$ ratio
Poly time approximation scheme:

> For fixed ϵ scheme runs in polytime (e.g. $n^{\frac{1}{\epsilon}}$) Fully polynomial time approximation scheme has runtime polynomial in n and also in $\frac{1}{\epsilon}$
>
> e.g $O\left(\frac{1}{\epsilon^2}n^3\right)$