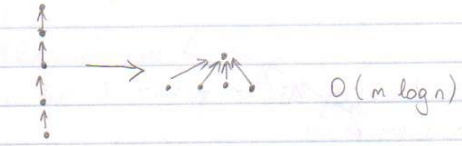
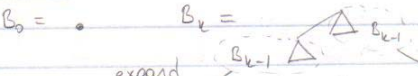
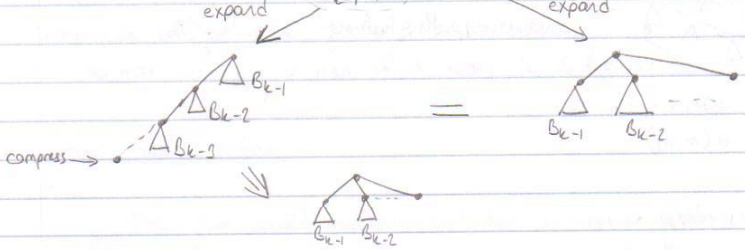
9/20/2006

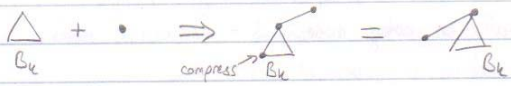
Path compression with naive union



Binomial tree

B_0 = • B_k =  ← 2^k nodes, depth k, root has k children.





$n = 2^{k+1}$

↳ B_k with 1/2 nodes repeat n times link with single node, compress long path.
$\frac{1}{2} \log n$! (lower bound)

Doing path reversal on B_k tree ends up giving B_k tree.

Amortized complexity

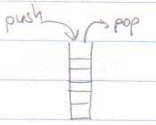
Assign a cost ("amortized cost") to each operation such that

Σ amortized costs $\geq \Sigma$ actual costs

for all possible sequences of ops.

Banker's view: Money for some amount of computation. Assign cost such that you never run out of money and finish your computation. Some operations may be fast so that you save money for other slower operations.

Physicist's view:



multipop: 0 or more pops followed by 1 push.
Start with empty stack, do m multipops.
total cost $\leq 2m$

exact cost: $2m-k$, where k = final stack height
(assign \$2 per multipop).

Assign a potential Φ to each configuration of data structure.
Define amortized time of operation i = a_i = $t_i + \Delta\Phi$ (actual time; $\Phi_i - \Phi_{i-1}$)

$$\Rightarrow a_i = t_i + \Phi_i - \Phi_{i-1}$$

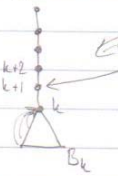
$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} t_i + \underbrace{\Phi_m}_{\geq 0 (?)} - \underbrace{\Phi_0}_{= 0 (?)} \quad \Rightarrow \sum a_i \geq \sum t_i$$

(art is finding or potential)

Path compression with naive linking

$O(\log n)$ amortized time per link, find?

Suppose we do all links first with no compression = union tree.

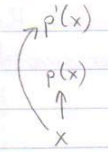


Compressions become partial compressions on union tree.

leaf has height = 0

height = 1 + max hts of its children.

$\Rightarrow$ hts strictly increasing.

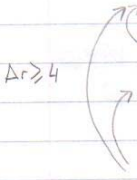


$r(p'(x)) > r(p(x))$

cost of find = 2 + # pointer changes

find a bound on this

$O(n^2)$

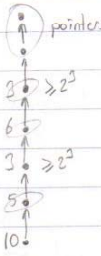


$O(m + n^2)$ — $n \log n$

$\log n$ — balance these two terms

Assign each node x to one of $O(\log n)$ classes depending on $r(p(x)) - x$

$level(x) = \lfloor \log [r(p(x)) - r(x)] \rfloor$



nodes charged to find $\leq \log n + 2 \leftarrow O(m \log n)$

Each node on find is charged either to find, or has its level increase as a result of the find.

$O(n \log n)$

charge last node in each level to find.

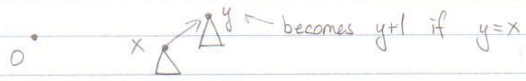
Assign $\log n$ credits per node

$\log n + 2$ credits per find.

Path compression + union by rank

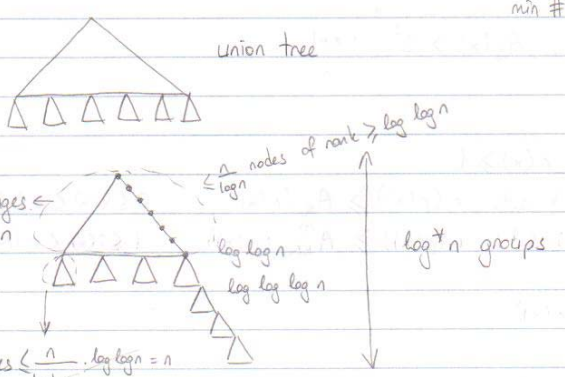
Nodes of high rank are exponentially infrequent.

With union by rank, the final rank assigned to a node is its height in the union tree.

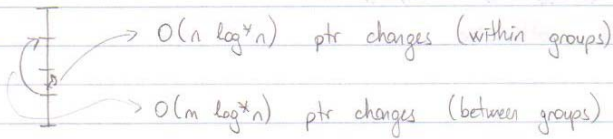


nodes of rank $k \leq n/2^k$

(*) $\log^* n$ amortized bound per op. = $\underbrace{\log \log \log \cdots \log n}_{\text{min \#}} \leq 1$



Same for all groups.



$\log^{**} n$ → # times you have to apply $\log^* n$

$\log^{***} n$

⋮

As $\frac{m}{n}$ increases

$O(m + n \log n)$ $\quad 2^n$

↓

$O(m + n \log^* n)$

↓ any constant # of *'s $\quad 2^{2^{\cdots}}$ } n 2's.

$O(m + n \log^{*\cdots*} n)$

$A_0(x) = x+1$ for $x \geq 1$

$A_{k+1}(x) = \underbrace{A_k^{(x+1)}(x)}$ for $x \geq 1$

$\underbrace{A_k(A_k(\cdots A_k(x)\cdots))}_{x+1}$

$\alpha(n)$ = smallest k s.t. $A_k(1) \geq n$

$A_1(x) = 2x+1$, $A_2(x) > 2^x$, $A_3(x) > 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}\}x+1$

Let x be a nonroot with $r(x) \geq 1$

level of $x = k(x)$ = max k s.t. $r(p(x)) \geq A_k(r(x))$ $\quad 0 \leq k(x) < \alpha(n)$

index of $x = i(x)$ = max i s.t. $r(p(x)) \geq A_{k(x)}^{(i)}(r(x))$ $\quad 1 \leq i(x) \leq r(x)$

$r(p(x)) \geq A_{k(x)}^{(r(x)+1)}(r(x))$

$\Downarrow$

$A_{k(x)+1}(r(x))$

$\phi(x) = \alpha(n)\, r(x)$ if x is a root or $r(x) = 0$

$\phi(x) = (\alpha(n) - k(n))\, r(x) - i(x)$ otherwise

$\Phi = \sum \phi(x)$