§4 Fast&Slowly Growing Functions

2^n exponential $\lceil \log N \rceil = \min\{ n : 2^n \geq N \}$

2^{2^n} doubly exponential $\lceil \log\log N \rceil = \min\{ n : 2^{2^n} \geq N \}$

…

$2^{2^{\cdot^{\cdot^{2}}}}$ tower of height n

aka tetration $2\uparrow\uparrow n = 2^{2\uparrow\uparrow(n-1)}$

$\log^*(N)$ = #iterations of log before argument ≤ 1 $= 1 + \log^*(\log N),\ N>1$

Example: $\log(2^{64})=?$ $\log\log(2^{128})=?$ $\log^*(2^{256})=?$

Ackermann function

$A_0(n)=n+2,\quad A_{k+1}(0)=A_k(1),\quad A_{k+1}(n+1) = A_k(\,A_{k+1}(n))$

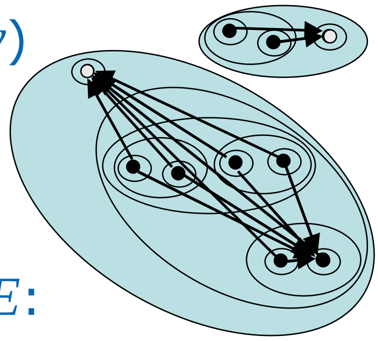
$A_1(n) = 2n+3,\quad A_2(n) = 2^{n+3}-3,\quad A_3(n) = 2\uparrow\uparrow(n+3)-3$

Inverse Ackermann $\alpha(N) = \min\{ n : A_n(n) \geq N \}$

§4 Disjoint-Set Data Structure

`MakeSet`(x) **`FindSet`**(x) **`Union`**(x,y)

return "handle" **Goal:** amortized $O^*(1)$



Example Application: graph $G=(V,E)$ connected components with 'growing' E:

`sameComponent`(u,v): return **`FindSet`**(u)=**`FindSet`**(v)

`addEdge`(u,v): **`Union`**(u,v)

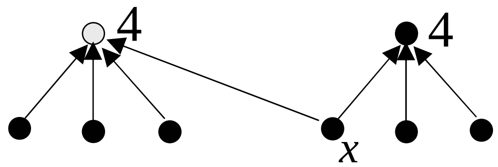
Naive implementation as forest of depth 1: n calls, $m:=n/2$ of which are **`MakeSet`** $\rightarrow$ runtime quadratic in n

Weighted union heuristic: attach smaller to larger tree

Theorem: This yields total time $O(n+m\cdot\log m)$ for any sequence of n calls, m of which are **`MakeSet`**

§4 Analysis of Union-by-Weight

MakeSet(x) **FindSet**(x) **Union**(x,y)



Observation: An element's link is updated only when its set is combined with one of more or equal weight.

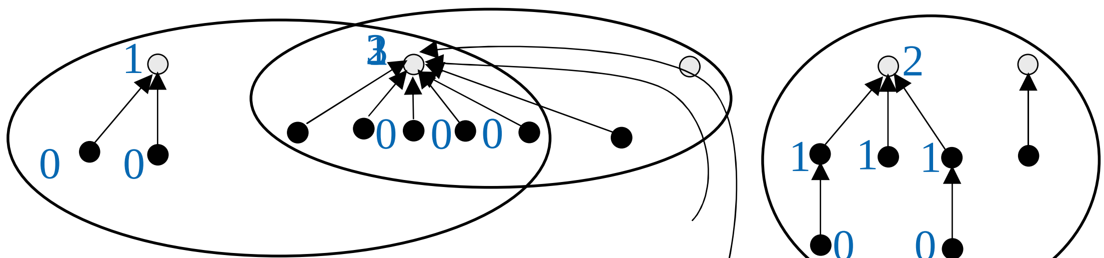
So to any of the $\leq m$ possible elements x, k updates occur only <u>after</u> having made $m \geq 2^k$ calls to **MakeSet**

MakeSet, **FindSet**: $O(1)$, **Union** ?

Weighted union heuristic: attach smaller to larger tree

Theorem: This yields total time $O(n+m \cdot \log m)$ for any sequence of n calls, m of which are **MakeSet** ■

Lazy Union by Rank, Path Compression




```
Union(x,y) //x,y handles
if y.rank > x.rank
 then attach x to y
else attach y to x;
 if x.rank = y.rank
  then x.rank++; fi; fi;
```

```
function FindSet(x): if x.parent≠NIL
 then x.parent := FindSet(x.parent);
return(x.parent);
```

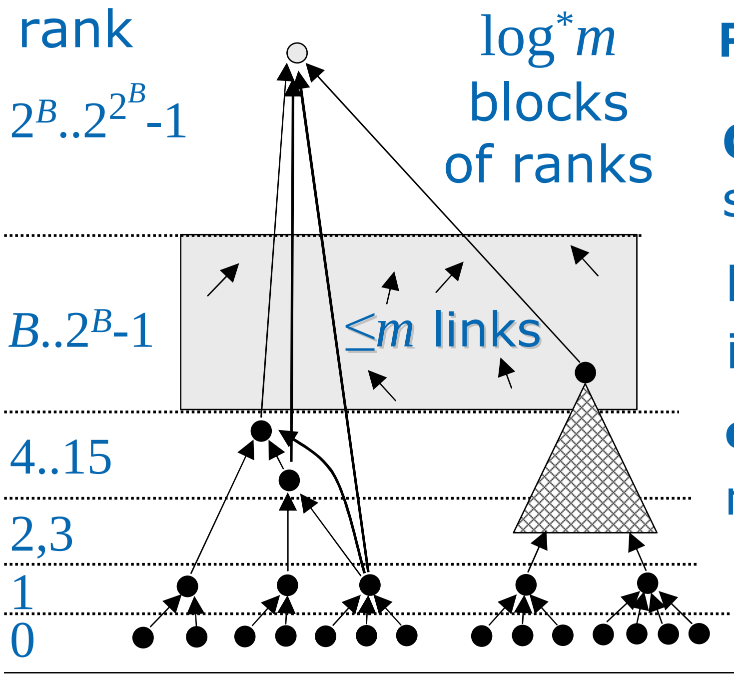
~~Naive~~ implementation as forest of <u>unbounded</u> depth:

MakeSet, **Union**: $O(1)$, **FindSet** ? *Path compression*

Lazy Union-by-rank: attach shallower to deeper tree

Theorem: This algorithm makes m **MakeSet** and n-m **FindSet** and **Union** calls run in total time $O(n \cdot \alpha(m))$.

Amortized Analysis of Union-Find



FindSet(x) **Union**(x,y)

Claim a) Ranks increase strictly along each path.

b) Any node of rank r is ancester to $\ge 2^r$ nodes.

c) No more than $m/2^r$ nodes can have rank$\ge r$

Total #steps from all nodes to their root(s)

= #links transcending block(s) + #links within a block

Theorem: This algorithm makes m **MakeSet** and n-m **FindSet** and **Union** calls run in total time $O(n\cdot\log^* m)$.

Review of Chapters 1 to 4

- *Virtues of Theoretical Computer Science:*
  - full and unambiguous problem specification
  - formal semantics of primitive operations
  - algorithm design (as opposed to 'programming')
  - and analysis (correctness, asymptotic cost)
  - optimality proof
- Stable Matchings and Gale-Shapley Algorithm
- AVL-Trees, Binomial Heaps
- Amortized Analysis, Potential Method
- Fibonacci Heaps, Relaxed Binomial Trees
- Fast/Slowly Growing Functions
- Union-Find Algorithm&Analysis