Two Mergeable Data Structures

Disjoint-Set 并查集 & Leftist-Tree 左偏树

Disjoint-Set(Union-Find Set) 并查集

- N distinct elements into a collection of disjoint sets.
  - Op1: Find which set a given element belong in I.e. Judge if two elements are in the same set
  - Op2: Unite two sets
- N个不同的元素组成不相交集合
  - 操作1: 寻找给定元素所属集合 (判断两个元素是否在同一集合)
  - 操作2: 合并两个集合

An Example of Disjoint-Set

Operation	**Disjoint sets**					
Initialization	{a}	{b}	{c}	{d}	{e}	{f}
Merge(a,b)	{a,b}		{c}	{d}	{e}	{f}
Query(a,c)	False					
Query(a,b)	True					
Merge(b,e)	{a,b,e}		{c}	{d}		{f}
Merge(c,f)	{a,b,e}		{c,f}	{d}		
Query(a,e)	True					
Query(c,b)	False					
Merge(b,f)	{a,b,c,e,f}			{d}		
Query(a,e)	True					
Query(d,e)	False					

Naive Algorithm

- Assign each set a label. 给集合编号

Op / *Element*	**{a}**	**{b}**	**{c}**	**{d}**	**{e}**	**{f}**
	1	2	3	4	5	6
Merge(a,b)	1	1	3	4	5	6
Merge(b,e)	1	1	3	4	1	6
Merge(c,f)	1	1	3	4	1	3
Merge(b,f)	1	1	1	4	1	1

Naive Algorithm

- Assign each set a label. 给集合编号

Op \ *Element*	**{a}**	**{b}**	**{c}**	**{d}**	**{e}**	**{f}**
	1	2	3	4	5	6
Merge(a,b)	1	1	3	4	5	6
Merge(b,e)	1	1	3	4	1	6
Merge(c,f)	1	1	3	4	1	3
Merge(b,f)	1	1	1	4	1	1

Query(a,e)

Naive Algorithm

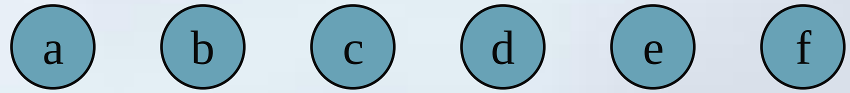
- Assign each set a label. 给集合编号

Op \ *Element*	{a}	{b}	{c}	{d}	{e}	{f}
	1	2	3	4	5	6
Merge(a,b)	1	1	3	4	5	6
Merge(b,e)	1	1	3	4	1	6
Merge(c,f)	1	1	3	4	1	3
Merge(b,f)	1	1	1	4	1	1

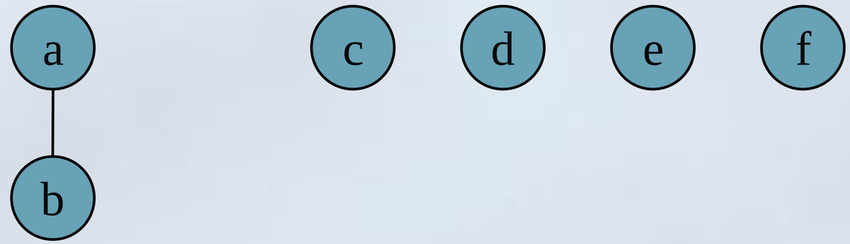
- Query – O(1); Merge – O(N)

First Look – Tree Structure

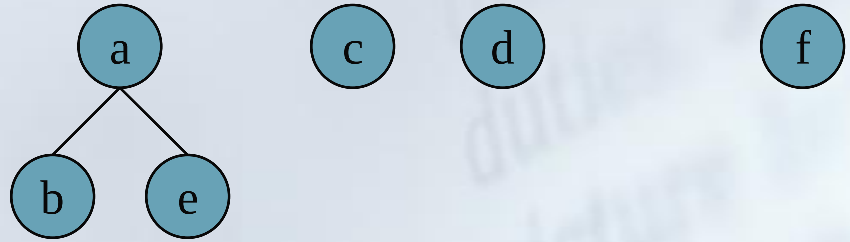
- Init:



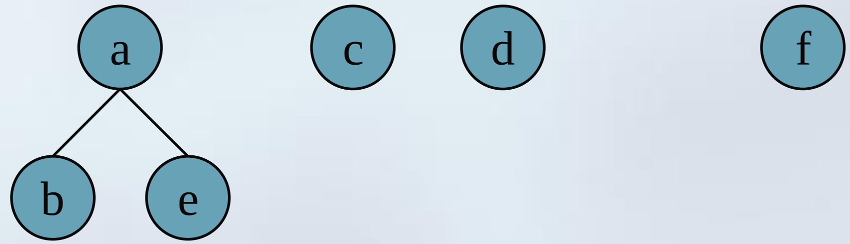
- *Merge(a,b)*



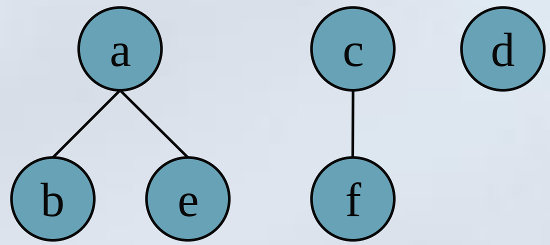
- *Merge(b,e)*



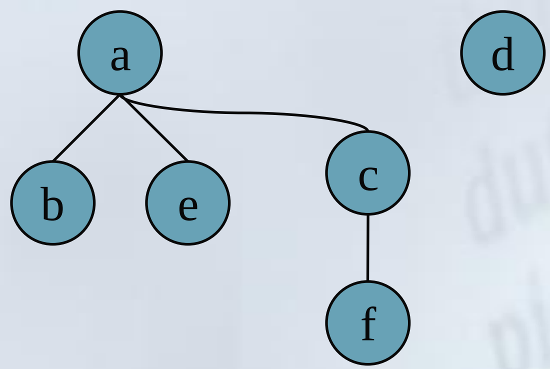
First Look – Tree Structure



- *Merge(c,f)*



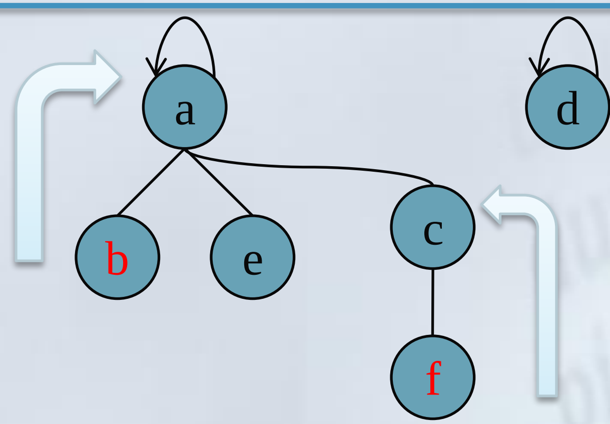
- *Merge(b,f)*



First Look – Tree Structure

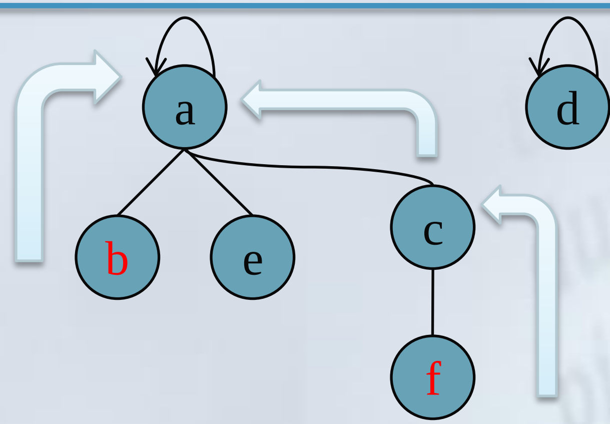
- *Merge(b,f)*
  - Attach f's tree as the direct subtree of b's
  - 将f所在树挂为b所在树的直接子树

  - Par[i] indicates i's father node; Par[i]=i for roots



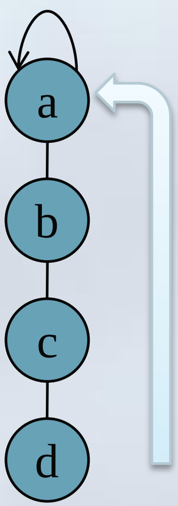
First Look – Tree Structure

- *Query(b,f)*
  - Simply compare the roots of b's tree and f's tree
  - 简单比较b和f所在树的根节点是否相同



First Look – Tree Structure

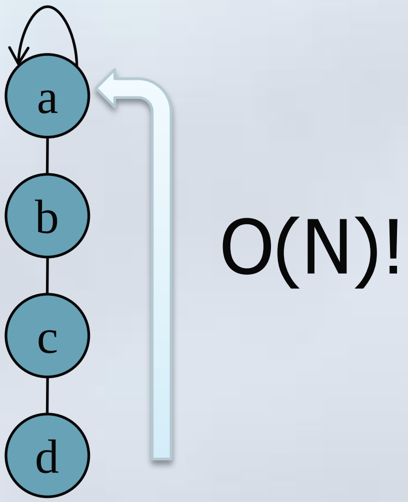
- *Weakness*
  - *Merge(c,d), Merge(b,c), Merge(a,b)*
  - *Query(d,*)*



O(N)!

First Look – Tree Structure

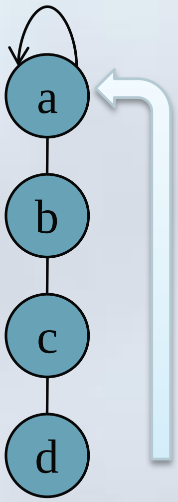
- *Weakness*
  - *Merge(c,d), Merge(b,c), Merge(a,b)*
  - *Query(d,*)*



Merge – O(1); Query – O(N)

First Look – Tree Structure

- *Weakness*
  - *Merge(c,d), Merge(b,c), Merge(a,b)*
  - *Query(d,*)*



O(N)!

Merge – O(N); Query – O(N)

Improve One – Union by Rank

- For each node, maintain a *Rank* that is an upper bound on the height of that subtree
- 每个点维护一个*Rank*表示子树最大可能高度
- Root with smaller rank is made to point to root with larger rank in Merge operation.
- 较小*Rank*的树连到较大*Rank*树的根部。

Improve One – Union by Rank

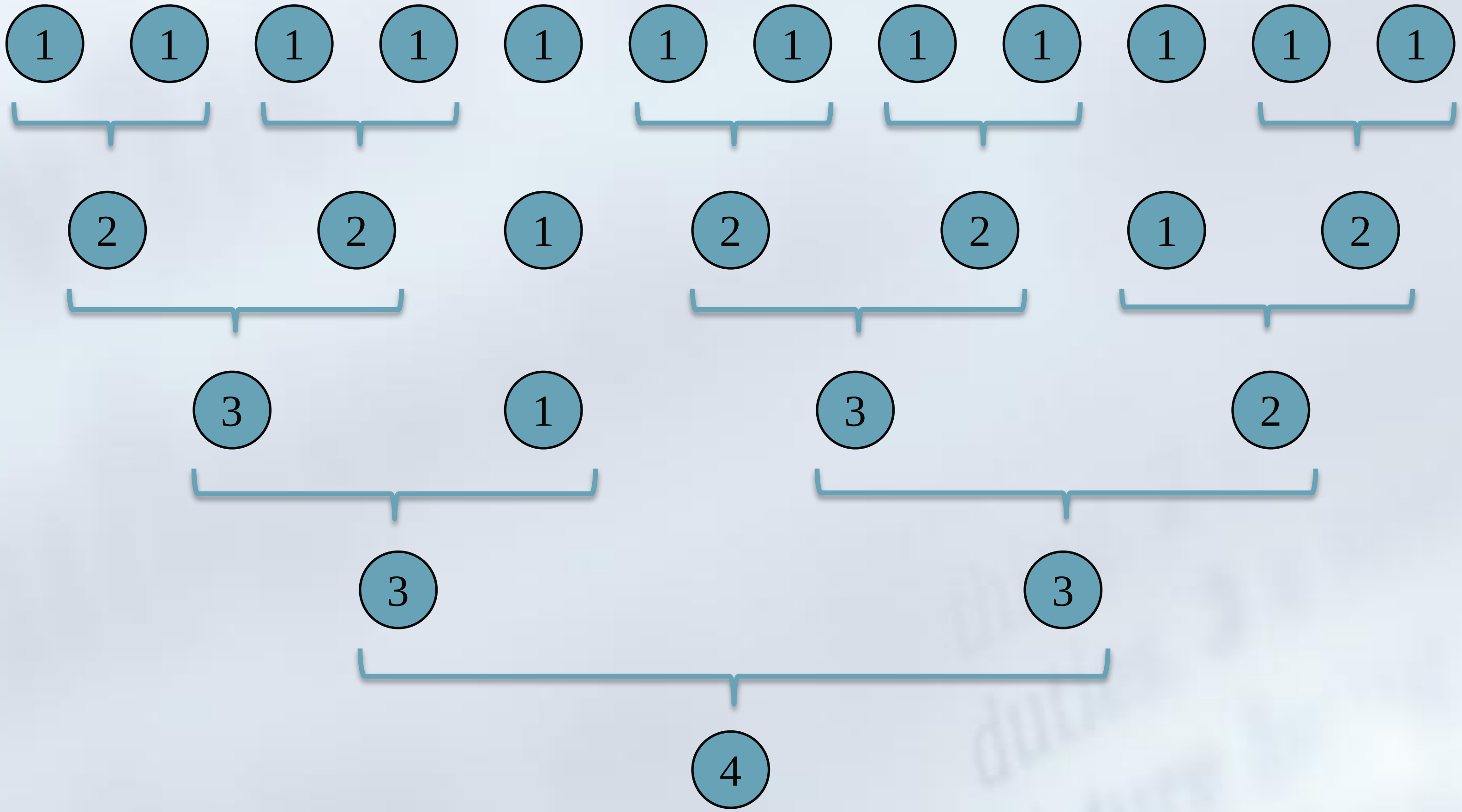
New One

- LINK(x, y)
  - If Rank[x]>Rank[y]
    - par[y] ← x
  - Else
    - Par[x] ← y
    - If Rank[x]=Rank[y]
      - Rank[y]++

Old One

- LINK(x, y)
  - par[y] ← x

Improve One – Union by Rank



Improve One – Union by Rank

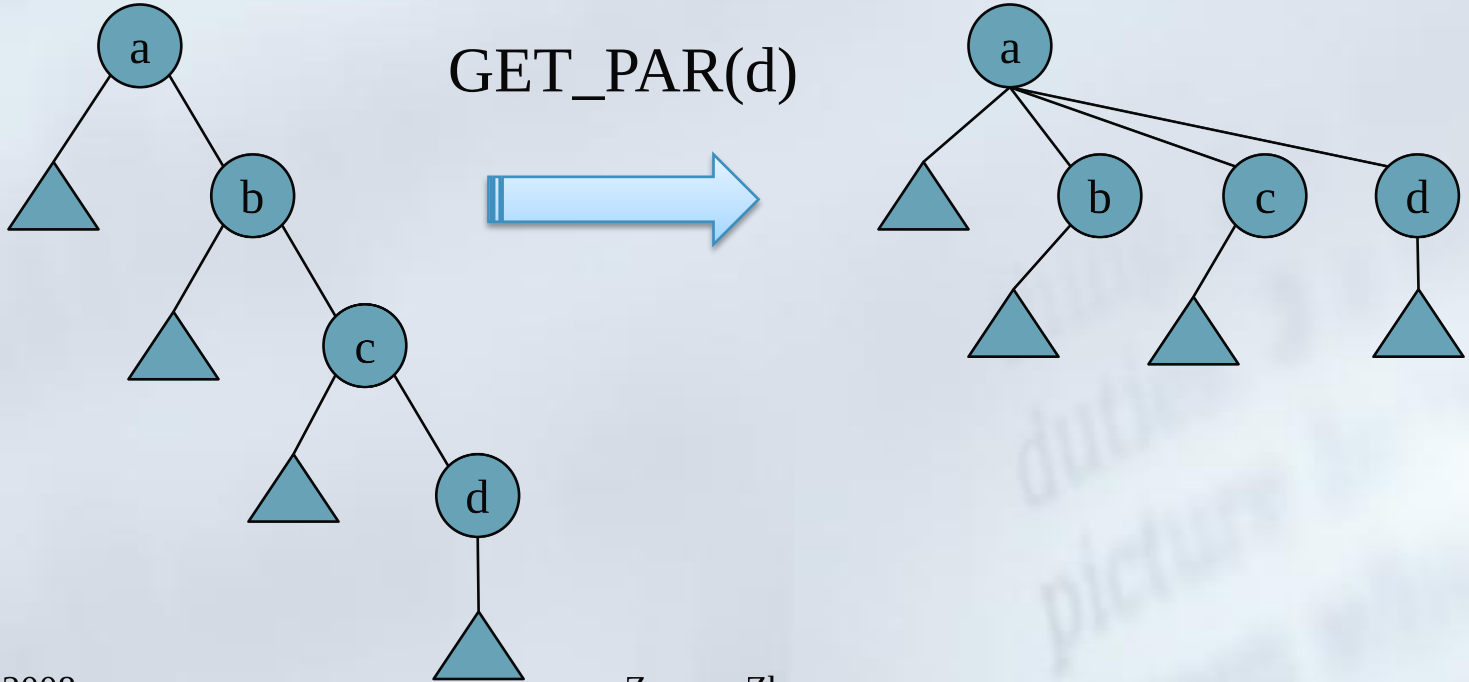
- GET_PAR(a)
  - If Par[a]=a
    - Return a
  - Else
    - Return GET_PAR(par[a])
- Query(a,b)
  - Return GET_PAR(a)==GET_PAR(b)
- Merge(a,b)
  - LINK(GET_PAR(a), GET_PAR(b))

Improve One – Union by Rank

- GET_PAR(a) – $O(\log_2 N)$
  - If Par[a]=a
    - Return a
  - Else
    - Return GET_PAR(par[a])
- Query(a,b) – $O(\log_2 N)$
  - Return GET_PAR(a)==GET_PAR(b)
- Merge(a,b) – $O(\log_2 N)$
  - LINK(GET_PAR(a), GET_PAR(b))

Improve Two – Path Compression

- In GET_PAR method, make each node on the find path directly point to the root
- 将GET_PAR中查找路径上的节点直接指向根



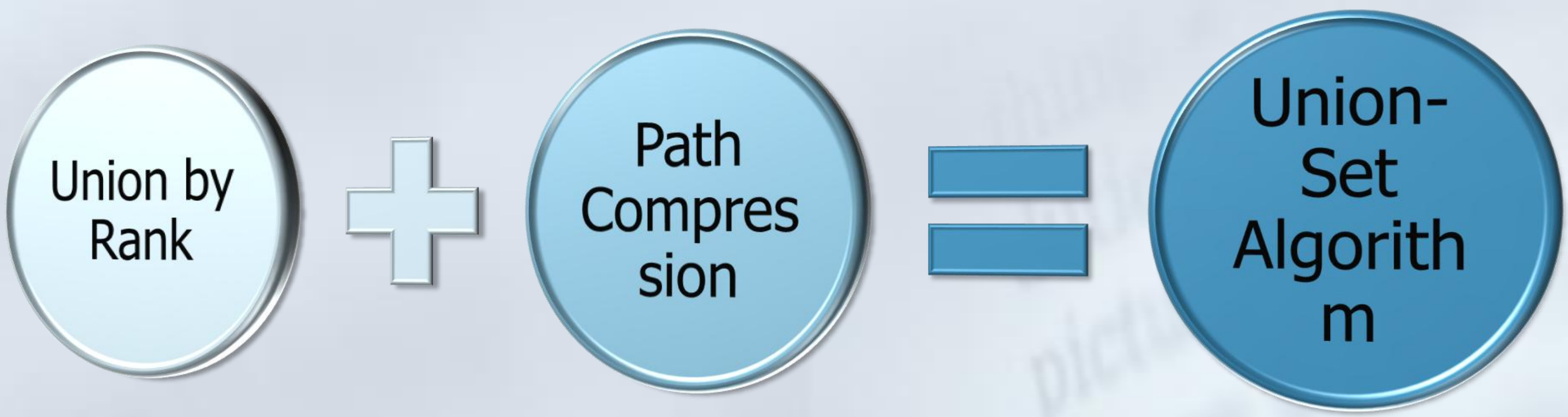
Improve Two – Path Compression

New Code

- GET_PAR(a)
  - If Par[a]!=a
    - Par[a] =GET_PAR(par[a])
  - Return par[a]

Old Code

- GET_PAR(a)
  - If Par[a]=a
    - Return a
  - Else
    - Return GET_PAR(par[a])



Complexity

Amortized cost of GET_PAR operation $O(\alpha(n))$

GET_PAR函数的平摊复杂度为$O(\alpha(n))$

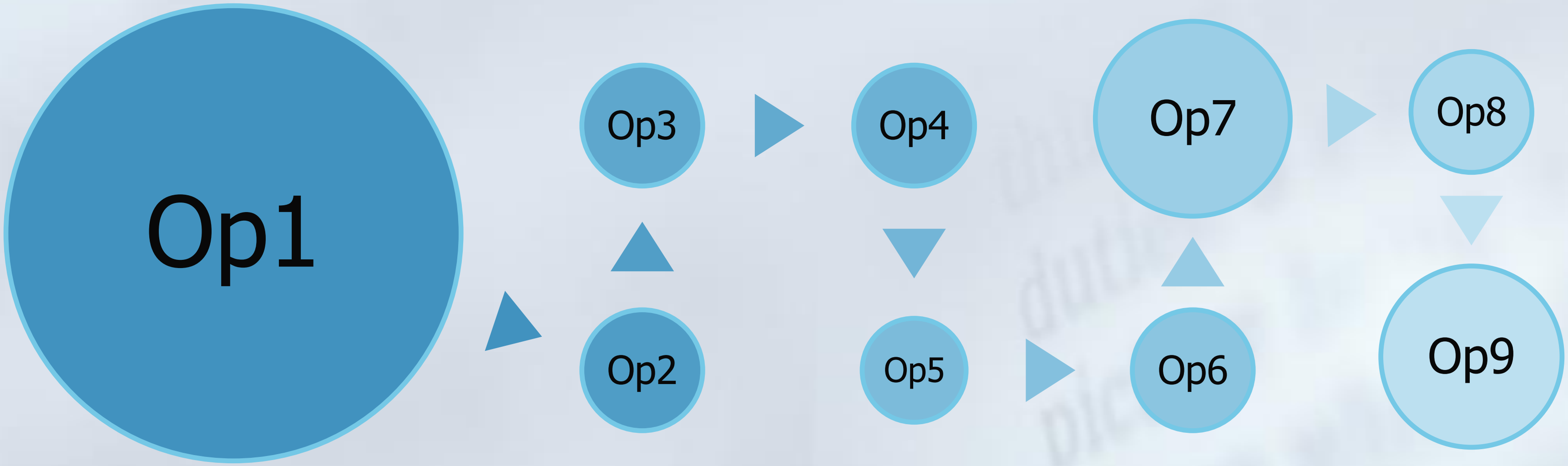
- $\alpha(n)$ $=0$, if $0<=n<=2$
- $=1$, if $n=3$
- $=2$, if $4<=n<=7$
- $=3$, if $8<=n<=2047$
- $=4$, if $2048<=n<=A_4(1)\approx \left.2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}\right\}2048$

Complexity

Amortized cost of GET_PAR operation O(α(n))

GET_PAR函数的平摊复杂度为O(α(n))

- Amortized analysis is a tool for analyzing algorithms that perform a sequence of **similar operations**.
- 平摊分析是一种分析一串**类似操作**的总体效率的思想



Practical Use

Practical Use

```
int get_par(int u) {
  if (par[a]!=a)
    par[a] = get_par(par[a]);
  return par[a];
}
```

```
int link(int x, int y) {
  if (rank[x]>rank[y]) par[y]=x;
  else par[x]=y;
  if (rank[x]==rank[y])
    rank[y]++;
}
```

```
int par[];
int rank[];
```

```
int query(int a,int b) {
  return get_par(a)==get_par(b);
}
```

```
void merge(int a,int b) {
  link(get_par(a), get_par(b)
}
```

Practical Use

```
int get_par(int u) {
  if (par[a]!=a)
    par[a] = get_par(par[a]);
  return par[a];
}
```

```
int link(int x, int y) {
  par[y]=x;
}
```

```
int par[];
```

```
int query(int a,int b) {
  return get_par(a)==get_par(b);
}
```

```
void merge(int a,int b) {
  link(get_par(a), get_par(b)
}
```

Practical Use

```
int get_par(int u) {
  if (par[a]!=a)
    par[a] = get_par(par[a]);
  return par[a];
}
```

```
int par[];
```

```
int query(int a,int b) {
  return get_par(a)==get_par(b);
}
```

```
void merge(int a,int b) {
  par[get_par(a)] = get_par(b);
}
```

Practical Use

```
int par[];
```

```
int get_par(int u) {
  return par[a]==a ? a : par[a]=get_par(par[a]);
}
```

```
int query(int a,int b) {
  return get_par(a)==get_par(b);
}
```

```
void merge(int a,int b) {
  par[get_par(a)] = get_par(b);
}
```

Exercise 银河英雄传说

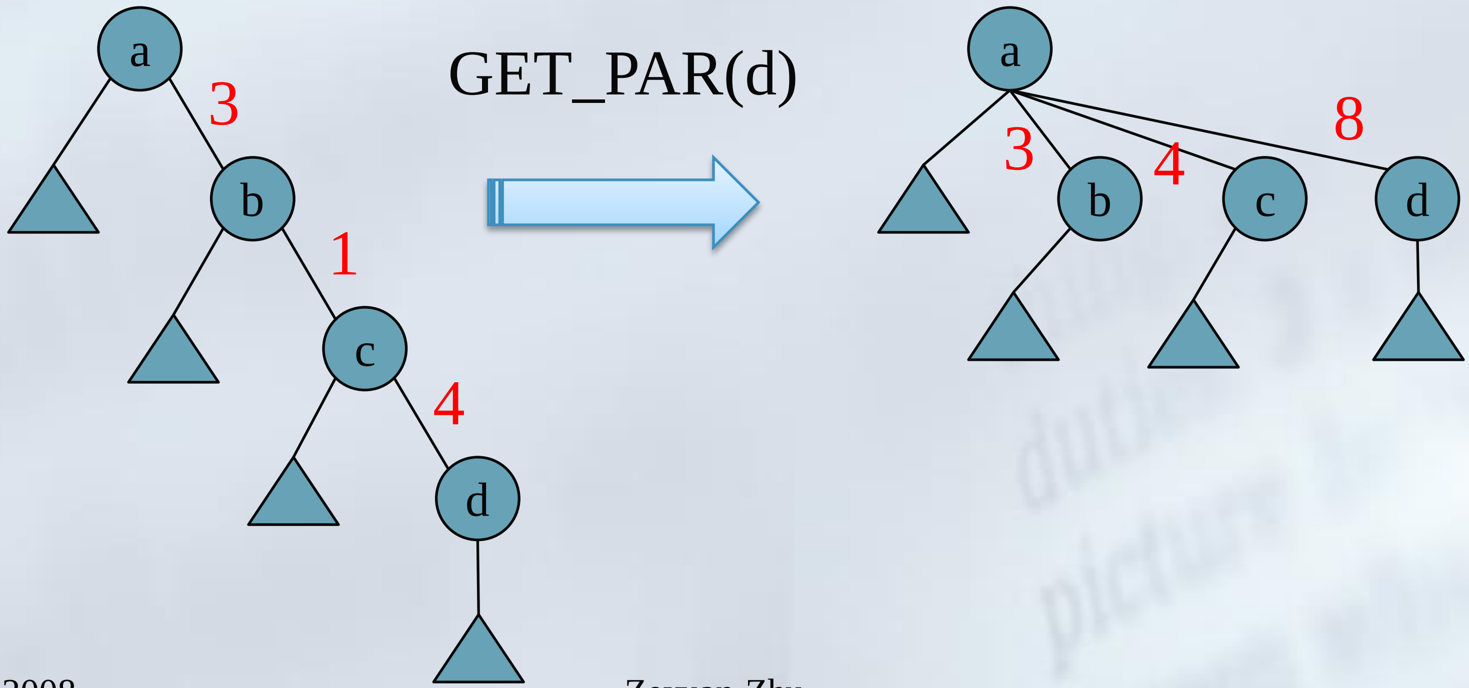
- 题目大意:
- M i j：让第i号战舰所在的整个战舰队列，作为一个整体(头在前尾在后)接至第j号战舰所在的战舰队列的尾部。
- C i j：询问电脑，杨威利的第i号战舰与第j号战舰当前是否在同一列中，如果在同一列中，那么它们之间布置有多少战舰。
- National Olympiad in Informatics 2002 天津

Exercise 银河英雄传说

- 可以把每列划分成一个集合，那么，舰队的合并、查询就是对集合的合并和查询。这样就是一个很典型的并查集算法的模型。
- 与普通并查集的区别是，此处需要记录每个点相对当前父节点的相对位置，用来回答查询操作中，两艘之间布置有多少战舰的问题。

Improve Two – Path Compression

- In GET_PAR method, make each node on the find path directly point to the root
- 将GET_PAR中查找路径上的节点直接指向根



Leftist-Tree 左偏树

——是一个二叉堆

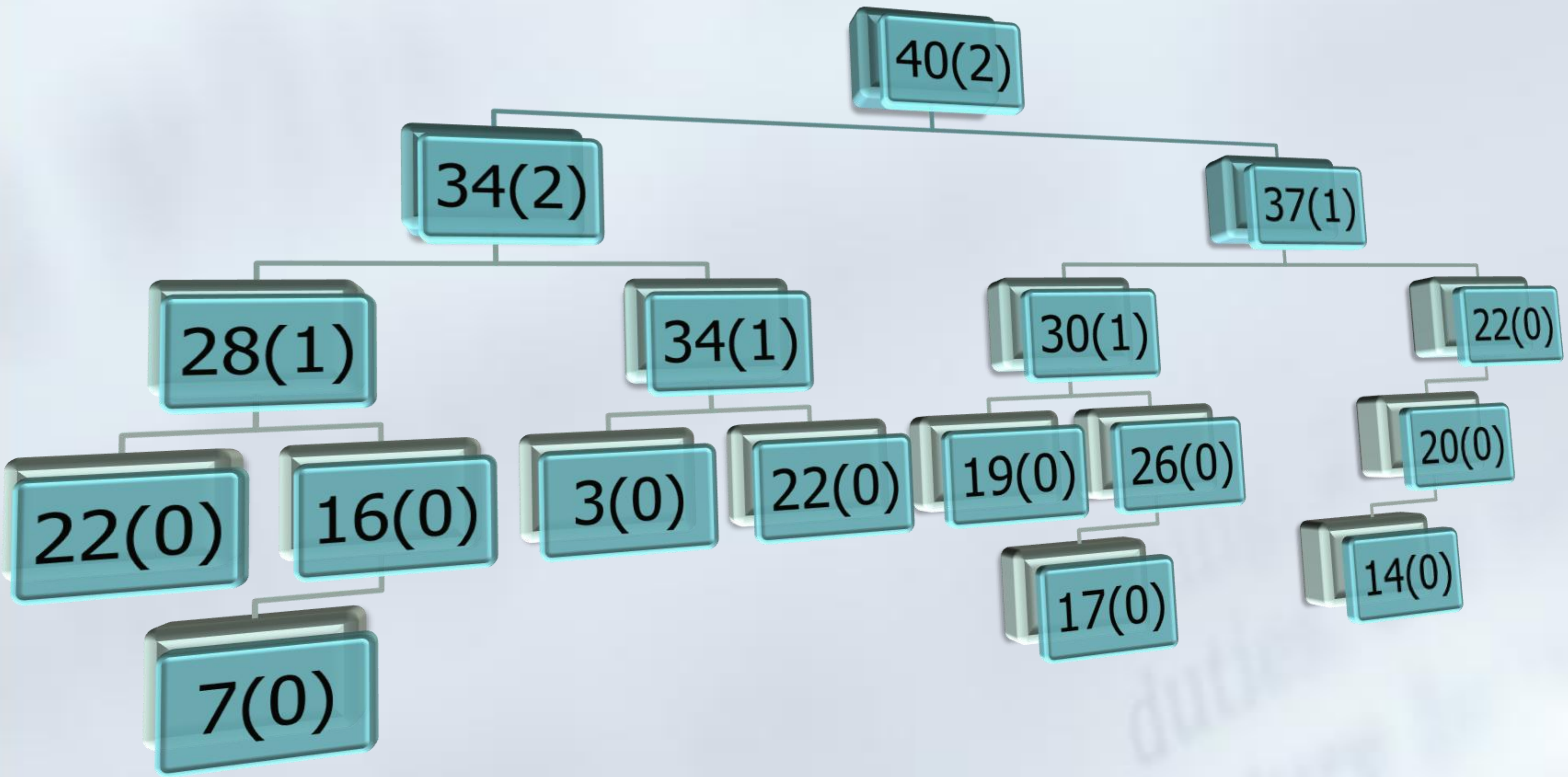
Lestist Tree 左偏树

	Classical Heap	Leftist Tree	Binomial Heap	Fibonacci Heap
Initialization	O(n)	O(n)	O(n)	O(n)
Insert	O(logn)	O(logn)	O(logn)	O(1)
Get Top	O(1)	O(1)	O(logn)	O(1)
Remove Top	O(logn)	O(logn)	O(logn)	O(logn)
Remove Any	O(logn)	O(logn)	O(logn)	O(logn)
Merge	O(n)	O(logn)	O(logn)	O(1)
Coding Difficulty	Low	Medium	High	Very High

Definition

- Every node has a count *dist* on the distance to the nearest **external** node(on its own subtree). In addition to the heap property, leftist trees are kept so the right descendant of each node has shorter distance to a leaf.
- 每个结点记录自身子树上到达最近外结点距离*dist*。除了堆所具有性质以外，左偏树保证右孩子的dist小于左孩子

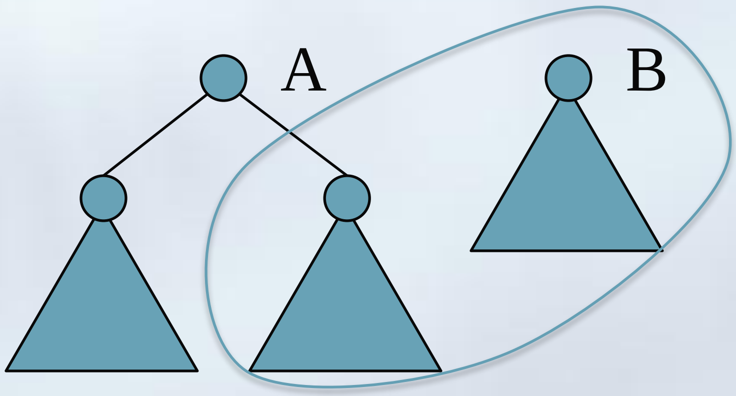
Definition



Merge Operation: Merge(A, B)

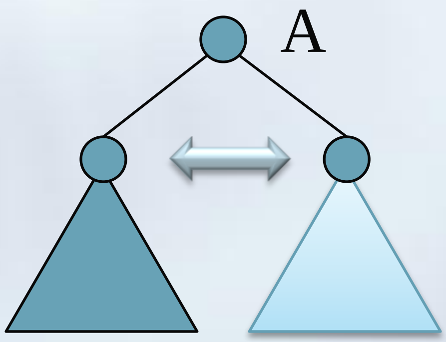
- Simplest case: either tree is empty (A=NULL or B=NULL). Just return the other tree.
- 如果其中一棵树为空，直接返回另一棵。
- If A==NULL Return B
- If B==NULL Return A

Merge Operation: Merge(A, B)



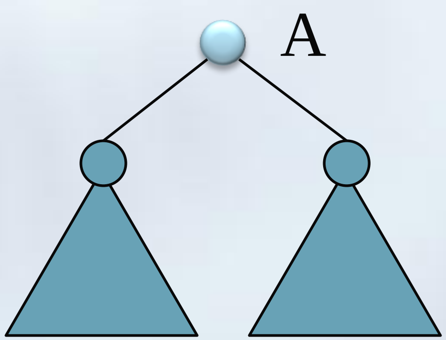
- Suppose A's root has larger *key*. Simply merge B and the right subtree of A.
- 设A根节点键值更大，将A右子树和B合并
- If Key[A] < Key[B] Swap(A,B)
- Right[A] ← Merge(Right[A], B)

Merge Operation: Merge(A, B)



- Swap Right(A) and Left(A) when necessary
- 当需要时交换Right(A)及Left(A)
- If dist[left[A]] < dist[Right[A]]
  - Swap(left[A],right[A])

Merge Operation: Merge(A, B)



- Update dist(A)
- If Right[A]==NULL
  - dist[A] ← 0
- Else
  - Dist[A] ← dist[Right[A]] + 1

Other Operations

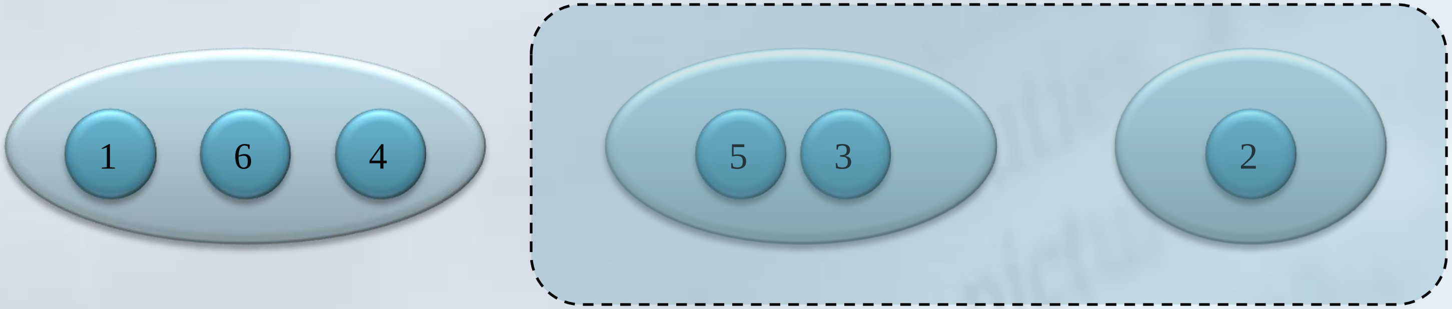
- Insert(A, x)
  - Merge(A, tree of x)
- RemoveTop(A)
  - Merge(Left[A], Right[A])

Lestist Tree 左偏树

	Classical Heap	Leftist Tree	Binomial Heap	Fibonacci Heap
Initialization	O(n)	O(n)	O(n)	O(n)
Insert	O(logn)	O(logn)	O(logn)	O(1)
Get Top	O(1)	O(1)	O(logn)	O(1)
Remove Top	O(logn)	O(logn)	O(logn)	O(logn)
Remove Any	O(logn)	O(logn)	O(logn)	O(logn)
Merge	O(n)	O(logn)	O(logn)	O(1)
Coding Difficulty	Low	Medium	High	Very High

Mixture of Disj-Set & Leftist Tree

- Merge(Node a, Node b)
  - Merge two heaps containing a/b respectively
  - 将包含a/b的两个堆合并
- FindMax (Node a)
  - Acquire the maximum element in the heap of a
  - 求a所在堆中的最大元素



Applications

- Medical Science: 疾病监控
- Biology: 细菌扩散
- Math: 等价类
- ……

References

- http://www.dgp.toronto.edu/people/JamesStewart/378notes/10leftist/
- Introduction to Algorithms (2nd Edition)

- Thanks!
- zhuzeyuan@hotmail.com
- 朱泽园 基科62