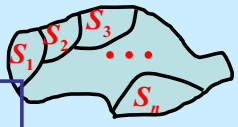
Data Structures for Disjoint Sets

$S = \{S_1, S_2, S_3, \ldots, S_n\}$ Collection of disjoint sets

Each set is an "equiv. Class" identified by a rep.



Make_Set(x) … Create the singleton set $S_x = \{x\}$ with x as rep.

Union(x,y) … Create $S_x \bigcup S_y$ with set rep and destroy S_x & S_y

Find_Set(x) Return pointer to set rep of

Data Structures for Disjoint Sets

Parameters

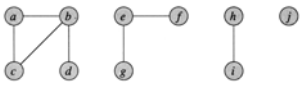
- n = # of Make-Set ops
- m = # of all ops
  {Make_Set, Union, Find_Set}
- # Unions $\leq n-1$
- $m \geq n$

Data Structures for Disjoint Sets

```
CONNECTED-COMPONENTS(G)
1  for each vertex v ∈ V[G]
2       do MAKE-SET(v)
3  for each edge (u, v) ∈ E[G]
4       do if FIND-SET(u) ≠ FIND-SET(v)
5            then UNION(u, v)

SAME-COMPONENT(u, v)
1  if FIND-SET(u) = FIND-SET(v)
2     then return TRUE
3     else return FALSE
```

Data Structures for Disjoint Sets

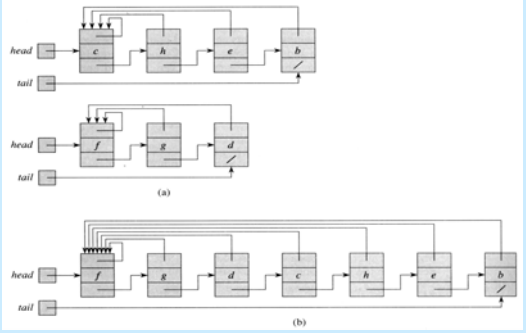


(a)

Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

(b)

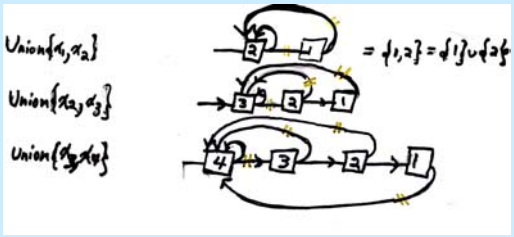
Data Structures for Disjoint Sets



Data Structures for Disjoint Sets

Operation	Number of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
UNION(x_3, x_4)	3
⋮	⋮
UNION(x_{q-1}, x_q)	$q-1$

Data Structures for Disjoint Sets



Data Structures for Disjoint Sets

Operation	Number of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
UNION(x_3, x_4)	3
⋮	⋮
UNION(x_{n-1}, x_n)	$n-1$

$$1+2+3+\ldots+(n-1)=\sum_{j=1}^{n-1} j=\frac{(n-1)n}{2}=O(n^2)$$

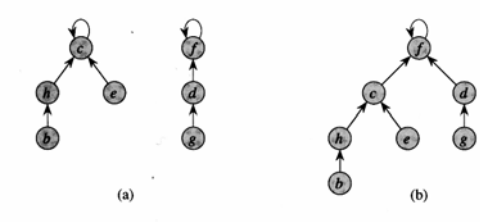
Weighted-Union Heuristic

- Each rep has a Length-of-List field
- Use this to append smaller sets to larger ones

Theorem. If the Linked-List representation of sets and if the weighted-union heuristic are used, and if a sequence of m {Make_Set, Union, Find_Set} ops are used (with n Make-Set ops), then the computational time taken is

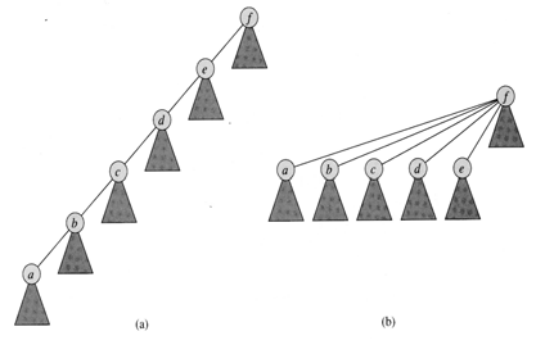
$$O(m+n\lg n)$$

Disjoint-Set Forest Representation



Union

A Problem



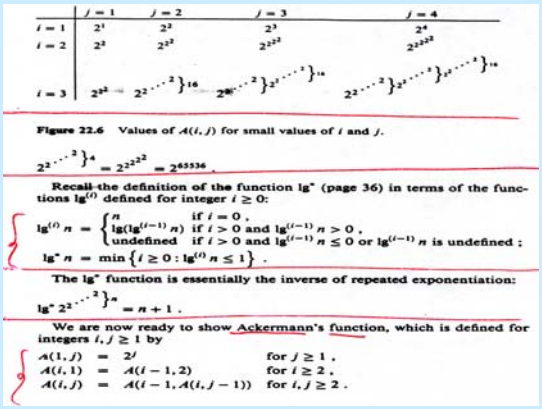
Data Structures for Disjoint Sets

```
MAKE-SET(x)
1  p[x] ← x
2  rank[x] ← 0
```

```
UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))
```

```
LINK(x, y)
1  if rank[x] > rank[y]
2     then p[y] ← x
3     else p[x] ← y
4          if rank[x] = rank[y]
5             then rank[y] ← rank[y] + 1
```

```
FIND-SET(x)
1  if x ≠ p[x]
2     then p[x] ← FIND-SET(p[x])
3  return p[x]
```



	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	2^1	2^2	2^3	2^4
$i=2$	2^2	2^{2^2}	$2^{2^{2^2}}$	$2^{2^{2^{2^2}}}$
$i=3$	$2^{2^{2^2}}$	$2^{2^{\cdot^{\cdot^{\cdot^2}}}\}16}$	$2^{2^{\cdot^{\cdot^{\cdot^2}}}\}2^{2^{\cdot^{\cdot^{\cdot^2}}}\}16}}$	$2^{2^{\cdot^{\cdot^{\cdot^2}}}\}2^{2^{\cdot^{\cdot^{\cdot^2}}}\}2^{2^{\cdot^{\cdot^{\cdot^2}}}\}16}}}$

Figure 22.6 Values of $A(i, j)$ for small values of i and j.

$2^{2^{\cdot^{\cdot^{\cdot^2}}}\}4} = 2^{2^{2^{2^2}}} = 2^{65536}$.

Recall the definition of the function $\lg^*$ (page 36) in terms of the functions $\lg^{(i)}$ defined for integer $i \geq 0$:

$$\lg^{(i)} n = \begin{cases} n & \text{if } i = 0, \\ \lg(\lg^{(i-1)} n) & \text{if } i > 0 \text{ and } \lg^{(i-1)} n > 0, \\ \text{undefined} & \text{if } i > 0 \text{ and } \lg^{(i-1)} n \leq 0 \text{ or } \lg^{(i-1)} n \text{ is undefined;} \end{cases}$$

$$\lg^* n = \min \{ i \geq 0 : \lg^{(i)} n \leq 1 \}.$$

The $\lg^*$ function is essentially the inverse of repeated exponentiation:

$$\lg^* 2^{2^{\cdot^{\cdot^{\cdot^2}}}\}n} = n + 1.$$

We are now ready to show Ackermann's function, which is defined for integers $i, j \geq 1$ by

$$\begin{aligned} A(1, j) &= 2^j && \text{for } j \geq 1, \\ A(i, 1) &= A(i-1, 2) && \text{for } i \geq 2, \\ A(i, j) &= A(i-1, A(i, j-1)) && \text{for } i, j \geq 2. \end{aligned}$$

Heuristics to Improve Running Time

- Union by Rank
- Path Compression

Each node has a field containing Rank, which is an upper bound on the height of the node