

CS 5321:
Advanced Algorithms –
Data Structures for Disjoint Sets

Ali Ebneenasir
Department of Computer Science
Michigan Technological University

Acknowledgement

- Eric Torng
- Moon Jung Chung
- Charles Ofria

Dynamic Sets and Data Structures

- Over the course of an algorithm's execution, an algorithm may maintain a dynamic set of objects
- The algorithm will perform operations on this set
 - Queries
 - Modifying operations
- We must choose a data structure to implement the dynamic set efficiently
- The "correct" data structure to choose is based on
 - Which operations need to be supported
 - How frequently each operation will be executed

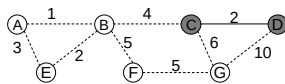
Case Study: Minimum Spanning Trees

- Input
 - Weighted, connected undirected graph $G=(V,E)$
 - Weight (length) function w on each edge e in E
- Task
 - Compute a spanning tree of G of minimum total weight
- Spanning tree
 - If there are n nodes in G , a spanning tree consists of $n-1$ edges such that no cycles are formed

Prim's algorithm

- A greedy approach to edge selection
 - Initialize connected component N to be any node v
 - Select the minimum weight edge connecting N to $V-N$
 - Update N and repeat
- Dynamic set in Prim's algorithm
 - An item is a node in $V-N$
 - The value of a node is its minimum distance to any node in N
 - A minimum weight edge connecting N to $V-N$ corresponds to the node with minimum value in $V-N$ (Extract minimum)
 - When v is added to N , we need to update the value of the neighbors of v in $V-N$ if they are closer to v than other nodes in N (Decrease key)

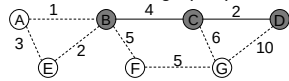
Illustration



- Maintain dynamic set of nodes in $V-N$
- If we started with node D , N is now $\{C,D\}$
- Dynamic set values of other nodes:
 - A, E, F : infinity
 - B : 4
 - G : 6
- Extract-min: Node B is added next to N

Updating Dynamic Set

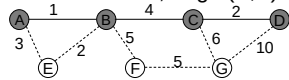
- Node B is added to N; edge (B,C) is added to T



- Need to update dynamic set values of A, E, F
 - Decrease-key operation
- Dynamic set values of other nodes:
 - A: 1
 - E: 2
 - F: 5
 - G: 6
- Extract-min: Node A is added next to N

Updating Dynamic Set Again

- Node A is added to N; edge (A,B) is added to T



- Need to update dynamic set values of E
 - Decrease-key operation
- Dynamic set values of other nodes:
 - E: 2 (unchanged because 2 is smaller than 3)
 - F: 5
 - G: 6

Dynamic Set Analysis

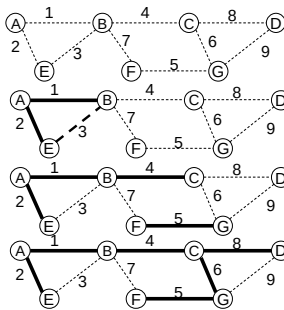
- How many objects in initial dynamic set representation of V-N?
- How many extract-min operations need to happen?
- How many decrease-key operations may occur?
- Given all of the above, choose a data structure and tell me the implementation cost.
 - Time to build initial dynamic set
 - Time to implement all extract-min operations
 - Time to implement all decrease-key operations

Kruskal's Algorithm

- A greedy approach to edge selection
 - Initialize tree T to have no edges
 - Iterate through the edges starting with the minimum weight one
 - Add the edge (u,v) to tree T if this does not create a cycle

Example

- (A,B)
- (A,E)
- (B,E): cycle
- (B,C)
- (F,G)
- (C,G)
- (B,F): cycle
- (C,D)
- (D,G): cycle



Better data structures

- We need mergeable data structures that still support fast searches
 - Binomial heaps (ch. 19 - CLR's book)
 - Fibonacci heaps (ch. 20 - CLR's book)
 - Disjoint set data structures (ch. 21 - CLR's book)
 - linked lists
 - forests

Disjoint Set Data Structures

- Given a universe U of objects (nodes V)
 - Maintain a collection of disjoint sets S_i that partition U
 - Each set S_i has a representative
- Basic operations:
 - $\text{Make-Set}(x)$: create a set whose only member is x
 - $\text{Find-set}(x)$: Returns set S_i that contains x
 - $\text{Merge}(x, y)$ or $\text{Union}(x, y)$:
 - Returns new set $S_k = (S_i \text{ union } S_j)$ where S_i is the set represented by x and S_j is the set represented by y
 - Sets S_i and S_j are destroyed
 - Representative of S_k becomes a member S_i or S_j

Example

- Disjoint Sets and Kruskal's algorithm
 - Universe U is the set of vertices V
 - The sets are the current connected components
 - When an edge (u,v) is considered, we check for a cycle by determining if u and v belong to the same set
 - 2 calls to $\text{Find-set}(x)$
 - If we add (u,v) to T , we need to merge the 2 sets represented by u and v .
 - $\text{Merge}(S_u, S_v)$

Analysis

- How do we initialize the universe?
- How many calls to find-set do we perform?
- How many calls to merge-set do we perform?

An Application

- Finding the connected components of dynamic graphs

CONNECTED-COMPONENTS($G = \langle V, E \rangle$){

- For each vertex v in V ,
 - Make-Set(v)
- For each edge (u, v) in E ,
 - if (Find-Set(u) $\neq$ Find-Set(v)) then Merge(u, v)

}

Analysis

- While DFS is more efficient to find the connected components of a static graph, this algorithm may be more efficient in cases where the edges of a graph change over time
- If G has k components, then
 - How many times do we call Find-Set?
 - How many times do we call UNION?

Linked List (LL) Disjoint Sets

- Each set represented by a linked list with *head* and *tail* pointers
- Representative: the first object of a list
- Structure of each node:
 - Next pointer
 - Data/set member
 - Pointer to the representative
- What is the complexity of Make-Set(x)?
- How about Find-Set(x)?
 - Find-Set returns a pointer to the representative

UNION in LL Disjoint Sets

- What is the semantics of UNION(x, y)?
 - Append x 's list to the end of y 's list
- Who is the representative now?
 - The repres. of y
- How about the complexity?
 - $|S_x|$, where S_x is the set containing x
 - All rep. pointers in x 's list should be revised
 - Costly!

Amortized Cost

- Make a list from n objects
- n Make-Set operations followed by $n-1$ UNION ops.
- What is the worst-case cost?
- What is the aggregate amortized cost of each UNION?
- Can we improve this cost?
- How?

Weighted Union

- Keep a length field in each list
- Always append the shorter list to the longer one
- Does this improve the complexity?
- m operations Make-Set, Find-Set, and Union, where n of those ops are Make-Set ops
- $O(m + (n \cdot \log n))$

Disjoint-set forests

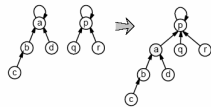
- Representation
 - Each set is represented as a tree, nodes point to parent
 - Root element is the representative for the set, points to self or has null parent pointer
 - Height: maintain height of tree as an integer

Naïve implementation

```
MAKESET(x):  
  parent(x) ← x
```

```
FIND(x):  
  while x ≠ parent(x)  
    x ← parent(x)  
  return x
```

```
UNION(x, y):  
   $\bar{x} \leftarrow \text{FIND}(x)$   
   $\bar{y} \leftarrow \text{FIND}(y)$   
  parent( $\bar{y}$ ) ←  $\bar{x}$ 
```



Merging two sets stored as trees. Arrows point to parents. The shaded node has a new parent.

Figure copied from Jeff Erickson's slides at UIUC.

Disjoint-set forests

- Operations
 - Make-Set: make a tree with one node
 - Find-Set: progress from current element to root element following links
 - Union: connect root of lower height tree to point to root of larger height tree
- Figures copied from Jeff Erickson, UIUC

union-by-rank or union-by-depth

```
MAKESET(x);
parent(x) ← x
depth(x) ← 0
```

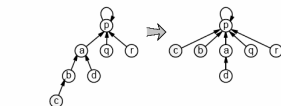
```
FIND(x):
while x ≠ parent(x)
  x ← parent(x)
return x
```

```
UNION(x, y)
x̄ ← FIND(x)
ȳ ← FIND(y)
if depth(x) > depth(y)
  parent(ȳ) ← x̄
else
  parent(x̄) ← ȳ
  if depth(x) = depth(ȳ)
    depth(ȳ) ← depth(ȳ) + 1
```

Leads to height of any tree of n nodes being at most $O(\lg n)$.

Figure copied from Jeff Erickson's slides at UIUC.

Path Compression



Path compression during $\text{FIND}(c)$. Shaded nodes have a new parent.

```
FIND(x)
if x ≠ parent(x)
  parent(x) ← FIND(parent(x))
return parent(x)
```

Leads to amortized cost of $\alpha(n)$, the inverse ackerman function.
For all practical purposes, $\alpha(n) \leq 4$.

Figure copied from Jeff Erickson's slides at UIUC.

Amortized Analysis

- Running time of m operations on n elements with union by rank and path compression is $O(m \alpha(n))$, where $0 \leq \alpha(n) < 4$ for all practical purposes

– $\alpha(n)$ is a function that grows very slowly

n	$\alpha(n)$
0–2	0
3	1
4–7	2
8–2047	3
2048– $A_4(1)$	4

- What's $A_4(1)$?
 - See Section 21.4, if you dare! It's $\gg 10^{80} \approx$ # of atoms in observable universe!
- $A_k(j) = j+1$; if $k = 0$
- $A_k(j) = A_{k-1}^{(j+1)}(j)$ if $k \geq 1$

Amortized Analysis - Continued

- Given n elements x in a forest disjoint set;
 $p[x]$ is the parent of x
- Lemma 1:** For all nodes x , we have
 - $\text{rank}(x) \leq \text{rank}(p[x])$;
 - if $x \neq p[x]$ then $\text{rank}(x) < \text{rank}(p[x])$
 - $\text{rank}[x]$ is initially 0 and increases until $x \neq p[x]$
 - When $x \neq p[x]$ holds, $\text{rank}(x)$ does not change anymore
 - $\text{rank}(p[x])$ monotonically increases
 - Ranks strictly increase from leaves to root

Amortized Analysis - Continued

- Lemma 2:** Every node has rank at most $n-1$
 - In fact, a stronger upper bound $\lfloor \log n \rfloor$
 - Prove by induction
- Lemma 3:**
 - If
 - UNION is replaced by two FIND-SET followed by a LINK in a sequence of m operations,
 - Then
 - the upper bound $O(m \alpha(n))$ changes only by a constant factor.

Amortized Analysis – Continued

- Potential function: assign a potential $\phi_k(x)$ to node x after k operations
- Potential of the forest after k operations:
 - $\phi_k = \sum_x \phi_k(x)$; $\phi_0 = 0$
- Define ϕ_k as:
 - If x is the root or $\text{rank}(x) = 0$ then
 - $\phi_k = \alpha(n) \cdot \text{rank}(x)$
 - If x is non-root and $\text{rank}(x) \geq 1$ then
 - $\phi_k(x) = (\alpha(n) - \text{level}(x)) \cdot \text{rank}(x) - \text{iter}(x)$
 - $\text{level}(x)$ and $\text{iter}(x)$ are two parameters that maximize the value of the Akerman function applied on $\text{rank}(x)$ while bounded by $\text{rank}(p[x])$
 - $\text{level}(x) = \max\{i: A_i(\text{rank}(x)) \leq \text{rank}(p[x])\}$
 - $\text{iter}(x) = \max\{i: A_{\text{level}(x)}^i(\text{rank}(x)) \leq \text{rank}(p[x])\}$
