Introduction to Algorithms
Lecture 11 Minimum Spanning Trees

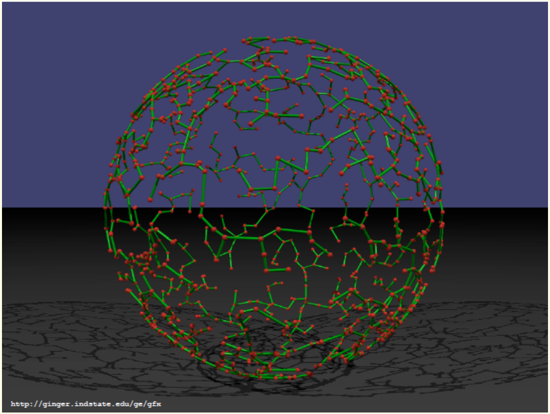
Xue Chen
xuechen1989@ustc.edu.cn
2025 spring in

Outline

1. Introduction
2. Basic Properties
3. Prim's Algorithm
4. Kruskal's Algorithm
5. Data Structures for Disjoint Sets

Overview

Spanning trees are fundamental objects in graph theory

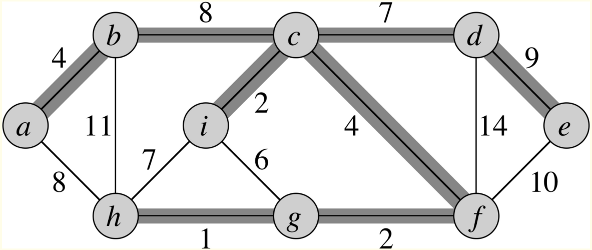


1. Connectivity: DFS/BFS trees
2. Single Source Shortest Paths Tree
3. Today: Minimum Spanning Trees
4. More: trees for cut, ...

Problem Description

Minimum Spanning Tree (MST)

Give a undirected & weighted graph G, define the weight of a tree T to be the sum of all weights on its edges. The goal is to find a Tree with minimum weight.



1. Network Design: Traffic, eletrical, ...
2. Algorithm Design: traveling salesman prob, Steiner tree, ...

Outline

1 Introduction

2 Basic Properties

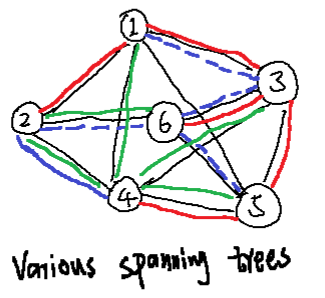
3 Prim's Algorithm

4 Kruskal's Algorithm

5 Data Structures for Disjoint Sets

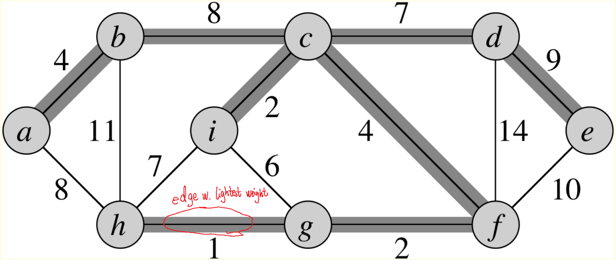
Basic Properties of Trees

1. Spanning tree is a minimal way to connect a graph
2. Remove a cycle edge can not disconnect a graph
3. A tree on n nodes has $n-1$ edges
4. Any connected graph $G=(V,E)$ with $|E|=|V|-1$ is a tree
5. A connected graph is a tree iff $\exists$ a unique path between any two nodes



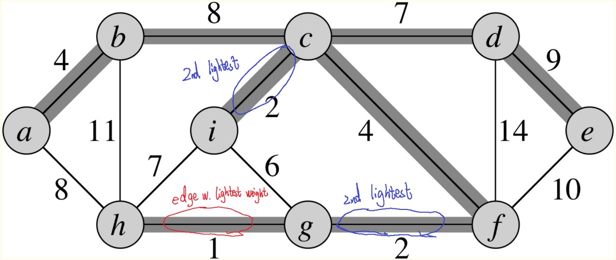
Cut Property

1. True of False: If $\exists$ a unique edge e with the lightest weight, e in any MST.



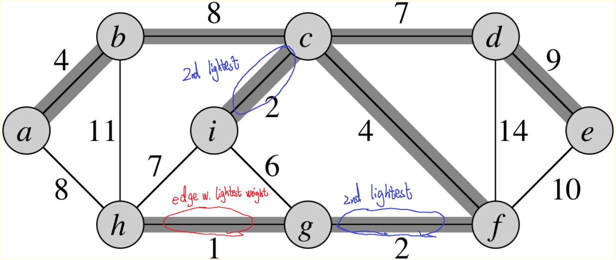
Cut Property

1. True of False: If $\exists$ a unique edge e with the lightest weight, e in any MST.
2. Question: So (h, g) is always in MST, how about 2nd lightest edges (i, c) and (g, f)?



Cut Property

1. True of False: If $\exists$ a unique edge e with the lightest weight, e in any MST.
2. Question: So (h, g) is always in MST, how about 2nd lightest edges (i, c) and (g, f)?

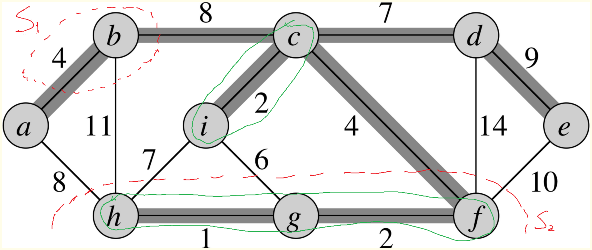


Greedy method in Theorem 23.1 of CLRS

Let A be a subset of edges that is included in some MST. Pick any cut $(S, \overline{S})$ such that S and $\overline{S}$ are NOT connected by edges in A.
Then for any lightest edge e on $S \times \overline{S}$, $\exists$ a MST that contains e.

Cut Property

1. True of False: If $\exists$ a unique edge e with the lightest weight, e in any MST.
2. Question: So (h, g) is always in MST, how about 2nd lightest edges (i, c) and (g, f)?



Greedy method in Theorem 23.1 of CLRS

Let A be a subset of edges that is included in some MST. Pick any cut $(S, \overline{S})$ such that S and $\overline{S}$ are NOT connected by edges in A.
Then for any lightest edge e on $S \times \overline{S}$, $\exists$ a MST that contains e.

Two Algorithms from opposite directions

Define $\text{cut}(S, \overline{S}) := \{(u, v) : u \in S, v \notin S\}$

Greedy method in Theorem 23.1 of CLRS

Let A be a subset of edges that is included in some MST. Pick any cut $(S, \overline{S})$ such that S and $\overline{S}$ are NOT connected by edges in A.
Then for any lightest edge e on $\text{cut}(S, \overline{S})$, $\exists$ a MST that contains e.

1. Prim's algorithm: Maintain S by adding vertices and finding the lightest edge on $E \cap (S, \overline{S})$

Two Algorithms from opposite directions

Define $\mathrm{cut}(S, \overline{S}) := \{(u, v) : u \in S, v \notin S\}$

Greedy method in Theorem 23.1 of CLRS

Let A be a subset of edges that is included in some MST. Pick any cut $(S, \overline{S})$ such that S and $\overline{S}$ are NOT connected by edges in A.
Then for any lightest edge e on $\mathrm{cut}(S, \overline{S})$, $\exists$ a MST that contains e.

1. Prim's algorithm: Maintain S by adding vertices and finding the lightest edge on $E \cap (S, \overline{S})$
2. Kruskal's Algorithm: Add an edge e and check $A \cup \{e\}$ has a cycle or not.

Outline

1 Introduction

2 Basic Properties

3 Prim's Algorithm

4 Kruskal's Algorithm

5 Data Structures for Disjoint Sets

Overview

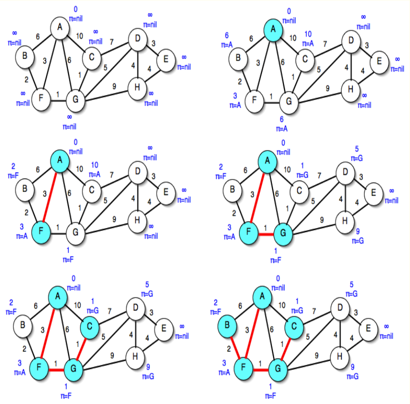
Based on the cut property, consider a greedy method to grow a tree S:

1. Start from $S = \{\text{root}\}$ and add a neighbor with the lightest crossing edge to S every time

Overview

Based on the cut property, consider a greedy method to grow a tree S:

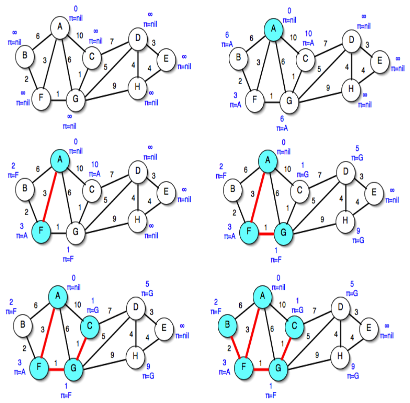
1. Start from $S = \{\text{root}\}$ and add a neighbor with the lightest crossing edge to S every time

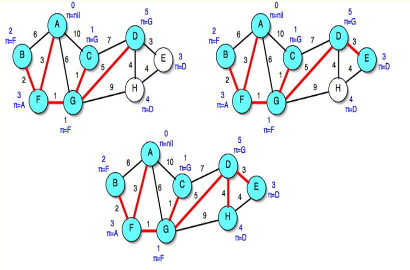


Overview

Based on the cut property, consider a greedy method to grow a tree S:

1. Start from $S = \{\text{root}\}$ and add a neighbor with the lightest crossing edge to S every time

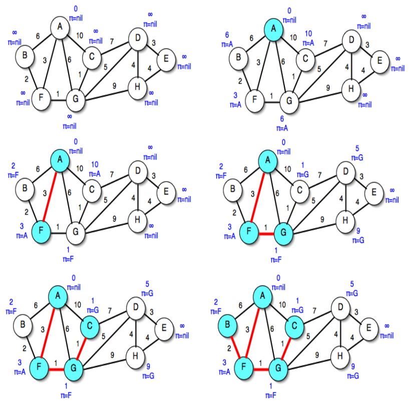


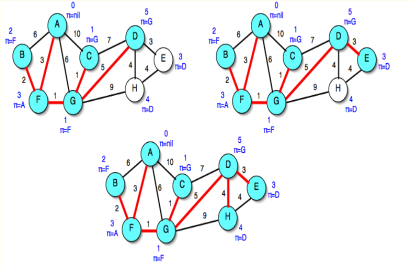


Overview

Based on the cut property, consider a greedy method to grow a tree S:

1. Start from $S = \{\text{root}\}$ and add a neighbor with the lightest crossing edge to S every time
2. Use a heap to maintain the lightest cross edge to v for every $v \notin S$





1. $u.key := \min_{v \in S} w(u, v)$ for $u \notin S$
2. $u.\pi$ denotes its argmin (as its father in MST)
3. Q maintains $\overline{S}$ according to $u.key$

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

1. $u.key := \min_{v \in S} w(u, v)$ for $u \notin S$
2. $u.\pi$ denotes its argmin (as its father in MST)
3. Q maintains $\overline{S}$ according to $u.key$

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

Time Complexity

$O(m \log n)$: Each vertex will be extracted once; so every edge will make at most one change in the heap

Outline

1. Introduction
2. Basic Properties
3. Prim's Algorithm
4. Kruskal's Algorithm
5. Data Structures for Disjoint Sets

Overview

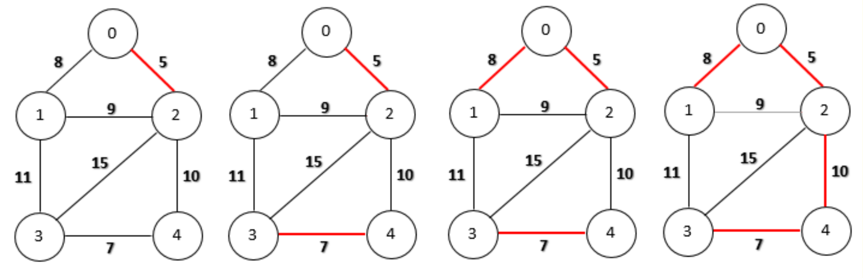
An alternate greedy approach:

1. Maintain a set of edges A (the same one in THM 23.1 in CLRS)
2. Try every edge e from the lightest to the heaviest: If $A + \{e\}$ does not constitute a cycle, $A = A + \{e\}$

Overview

An alternate greedy approach:

1. Maintain a set of edges A (the same one in THM 23.1 in CLRS)
2. Try every edge e from the lightest to the heaviest: If $A + \{e\}$ does not constitute a cycle, $A = A + \{e\}$



Description

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

Example: Pseudo-code from CLRS

Time Complexity

SORT(m edges) $+ n \times$ UNION $+ m \times$ FIND-SET

Next: Disjoint sets supports UNION and FIND-SET in almost constant time: RADIX-SORT + Disjoint sets is faster than $O(m \log n)$

Outline

1. Introduction
2. Basic Properties
3. Prim's Algorithm
4. Kruskal's Algorithm
5. Data Structures for Disjoint Sets

Introduction

A data structure for disjoint sets: For a fixed ground set, support 3 operations

1. MAKE-SET(x): create a new set $\{x\}$ for element x
2. FIND-SET(x): Find the representative element of x's set

Introduction

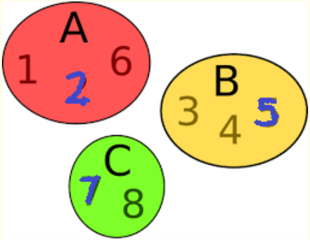
A data structure for disjoint sets: For a fixed ground set, support 3 operations

1. MAKE-SET(x): create a new set $\{x\}$ for element x
2. FIND-SET(x): Find the representative element of x's set
3. UNION(x, y): Unites two disjoint sets containing x and y separately into one set

Introduction

A data structure for disjoint sets: For a fixed ground set, support 3 operations

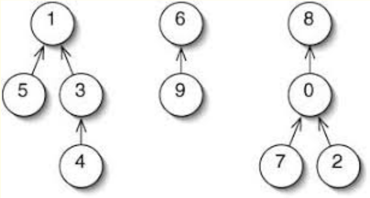
1. MAKE-SET(x): create a new set $\{x\}$ for element x
2. FIND-SET(x): Find the representative element of x's set
3. UNION(x, y): Unites two disjoint sets containing x and y separately into one set



Applications

Maintenance of connected components in graphs and maps, equivalence relation, least-common ancestors problem, ...

Implementation



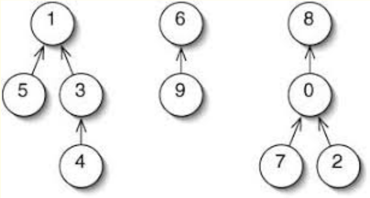
Maintain those sets as trees

1. Root is the unique representation
2. Question:

   How to implement MAKE-SET and FIND-SET?

   How about UNION operation?

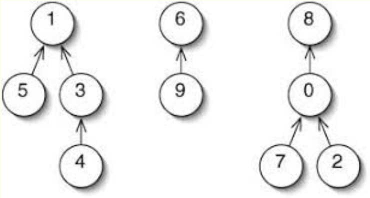
Implementation



Maintain those sets as trees

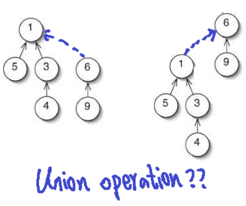
1. Root is the unique representation
2. Question:
   How to implement MAKE-SET and FIND-SET?
   How about UNION operation?
3. Recall: height is the most important parameter for time complexity

Implementation



Maintain those sets as trees

1. Root is the unique representation
2. Question:
   How to implement MAKE-SET and FIND-SET?
   How about UNION operation?
3. Recall: height is the most important parameter for time complexity

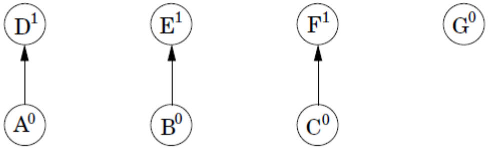


Rank of elements

To maintain heights, define a rank for each element:

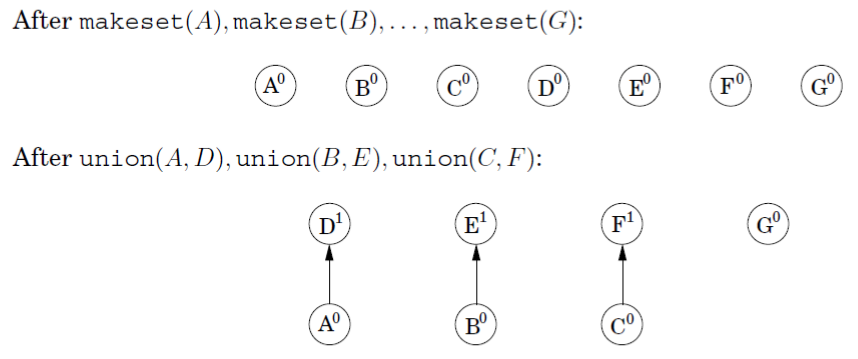
After $\texttt{makeset}(A), \texttt{makeset}(B), \ldots, \texttt{makeset}(G)$:

After $\texttt{union}(A, D), \texttt{union}(B, E), \texttt{union}(C, F)$:



Rank of elements

To maintain heights, define a rank for each element:

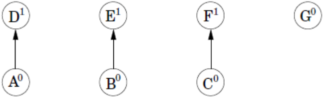


UNION(x, y)

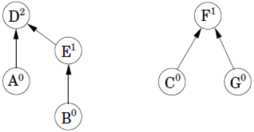
1. Let r_x and r_y be their representatives
2. If $rank(r_x) > rank(r_y)$: $\pi(r_y) = r_x$
3. Else: $\pi(r_x) = r_y$ and Update $rank(r_y)$ to $\max\{rank(r_y), rank(r_x) + 1\}$

After makeset(A), makeset(B), $\ldots$, makeset(G):

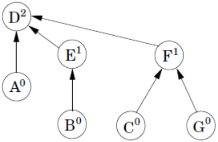
After union(A, D), union(B, E), union(C, F):



After union(C, G), union(E, A):



After union(B, G):



Basic Properties

1. For any non-root x, $rank(\pi(x)) > rank(x)$
   — Question: Is it always true $rank(\pi(x)) = rank(x) + 1$?

Basic Properties

1. For any non-root x, $rank(\pi(x)) > rank(x)$
   — Question: Is it always true $rank(\pi(x)) = rank(x) + 1$?
2. Any tree with root-rank k has at least 2^k nodes
   — proof by induction

Basic Properties

1. For any non-root x, $rank(\pi(x)) > rank(x)$
   — Question: Is it always true $rank(\pi(x)) = rank(x) + 1$?
2. Any tree with root-rank k has at least 2^k nodes
   — proof by induction
3. If there are n elements in all trees, at most $n/2^k$ nodes of rank k
   — implies all trees have height $\leqslant \log n$

Basic Properties

1. For any non-root x, $rank(\pi(x)) > rank(x)$
   — Question: Is it always true $rank(\pi(x)) = rank(x) + 1$?
2. Any tree with root-rank k has at least 2^k nodes
   — proof by induction
3. If there are n elements in all trees, at most $n/2^k$ nodes of rank k
   — implies all trees have height $\leqslant \log n$

☹ But this only implies $O(\log n)$ time for FIND-SET!

Basic Properties

1. For any non-root x, $rank(\pi(x)) > rank(x)$
   — Question: Is it always true $rank(\pi(x)) = rank(x) + 1$?
2. Any tree with root-rank k has at least 2^k nodes
   — proof by induction
3. If there are n elements in all trees, at most $n/2^k$ nodes of rank k
   — implies all trees have height $\leqslant \log n$

☹ But this only implies $O(\log n)$ time for FIND-SET!

A Small trick improves time significantly

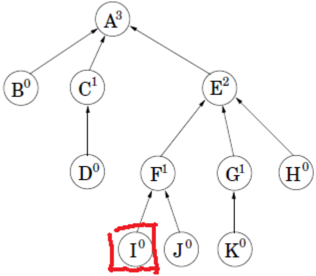
Path compression: In FIND-SET, set $\pi(x)$ to be the root for all x on the path

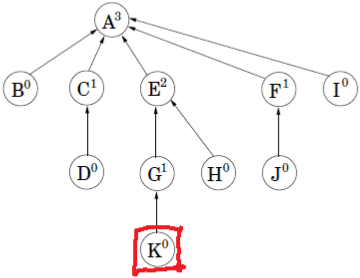
Formal Description

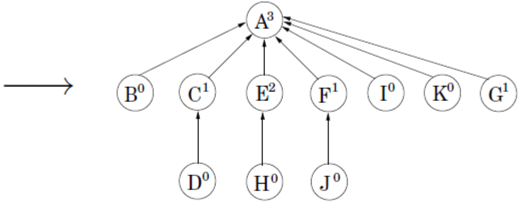
function FIND-SET(x)
  if $\pi(x) \neq x$ **then** //x is not a root
    $\pi(x) = \text{FIND-SET}(\pi(x))$ // Path Compression Trick
  return $\pi(x)$

procedure MAKE-SET(x)
  $\pi(x) = x$ and $rank(x) = 0$

procedure UNION-SET(x, y)
  $r_x = \text{FIND-SET}(x)$ and $r_y = \text{FIND-SET}(y)$
  if $rank(r_x) > rank(r_y)$ **then**
    $\pi(r_y) = r_x$
  else
    $\pi(r_x) = r_y$ and $rank(r_y) = \max\{rank(r_y), rank(r_x) + 1\}$







Find(I) followed by Find(k)

Amortized Analysis

Let $\log^* n$ be number of log operations that bring n down to 1, e.g., $\log^* 1000 = 4$ and $\log^* 2^{65536} = 5$

THM: Running time of Disjoint Sets

Amortized time of m FIND-SET operations is $O(m+n) \cdot \log^* n$.

Amortized Analysis

Let $\log^* n$ be number of log operations that bring n down to 1, e.g., $\log^* 1000 = 4$ and $\log^* 2^{65536} = 5$

THM: Running time of Disjoint Sets

Amortized time of m FIND-SET operations is $O(m+n) \cdot \log^* n$.

1. Consider those ranks— path compression does not affect their ranks
2. Divide all ranks into $\log^* n + O(1)$ intervals:

$\longleftarrow$ $\log^* n$ *intervals* $\longrightarrow$

$\{1\}, \{2\}, \{3,4\}, \{5,6,\cdots,16\}, \{17,\cdots,2^{16}=65536\}, \{65537,\cdots,\}, \cdots$

$\leq \frac{n}{2}$ nodes, $\leq \frac{n}{4}$, $\cdots$, $\leq \frac{n}{32} + \cdots + \frac{n}{2^{16}}$ nodes, $\cdots$

Amortized Analysis

Let $\log^* n$ be number of log operations that bring n down to 1, e.g., $\log^* 1000 = 4$ and $\log^* 2^{65536} = 5$

THM: Running time of Disjoint Sets

Amortized time of m FIND-SET operations is $O(m+n) \cdot \log^* n$.

1. Consider those ranks— path compression does not affect their ranks
2. Divide all ranks into $\log^* n + O(1)$ intervals:

   $\log^* n$ *intervals*

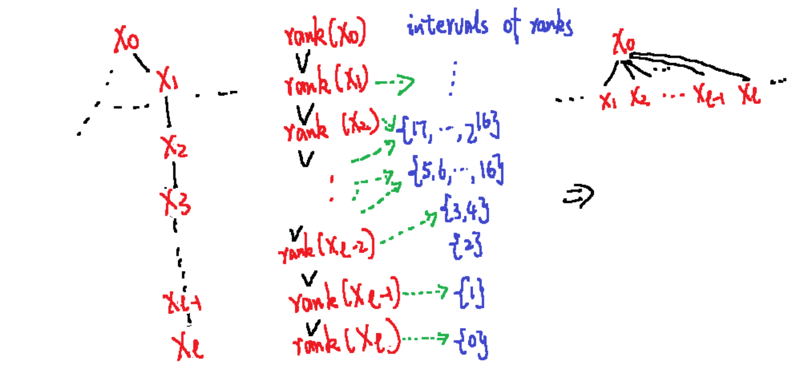
   $\{1\}, \{2\}, \{3,4\}, \{5,6,\cdots,16\}, \{17,\cdots,2^{16}=65536\}, \{65537,\cdots,\}, \cdots$

   $\le \frac{n}{2}$ nodes, $\le \frac{n}{4}$, $\cdots$, $\le \frac{n}{32} + \cdots + \frac{n}{2^{16}}$ nodes, $\cdots$

3. Amortized analysis with accounting method: For each x whose rank $\in [k+1, \ldots, 2^k]$, assign a budget 2^k — total budget $= n \cdot \log^* n$
4. Next: $m \cdot \log^* n + \textit{budgets}$ bounds time of $m \times$ FIND-SET

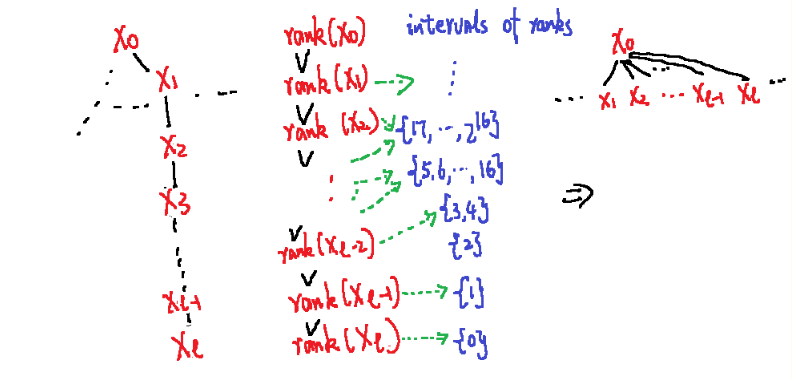
1. Consider FIND-SET of length ℓ: $x_\ell \to x_{\ell-1} \to \cdots \to x_1 \to x_0$

1. Consider FIND-SET of length ℓ: $x_\ell \to x_{\ell-1} \to \cdots \to x_1 \to x_0$
2. Fix $i \in [\ell]$: FIND-SET never changes $rank(x_i)$; but $rank(\pi(x_i))$ becomes larger



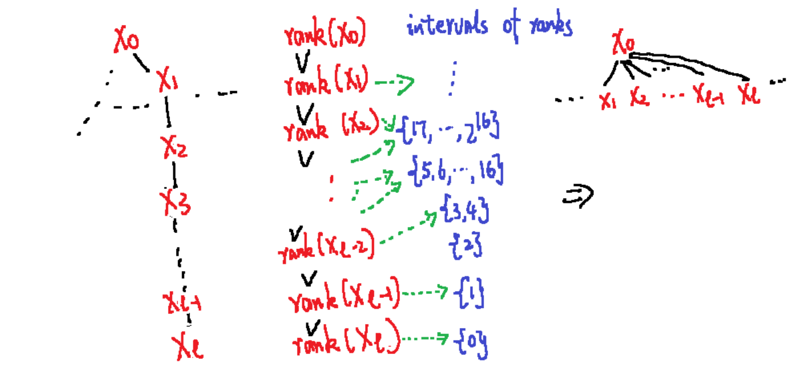
3. To bound amortized-time(FIND-SET), our focus is on $[rank(x_i), rank(\pi(x_i))]$ vs intervals of ranks

1. Consider FIND-SET of length ℓ: $x_\ell \to x_{\ell-1} \to \cdots \to x_1 \to x_0$
2. Fix $i \in [\ell]$: FIND-SET never changes $rank(x_i)$; but $rank(\pi(x_i))$ becomes larger



3. To bound amortized-time(FIND-SET), our focus is on $[rank(x_i), rank(\pi(x_i))]$ vs intervals of ranks
4. OBS: Once $[rank(x_i), rank(\pi(x_i))]$ crossed an interval, it keeps crossing those intervals in the future — this part is at most $\log^* n$

1. Consider FIND-SET of length ℓ: $x_\ell \to x_{\ell-1} \to \cdots \to x_1 \to x_0$
2. Fix $i \in [\ell]$: FIND-SET never changes $rank(x_i)$; but $rank(\pi(x_i))$ becomes larger



3. To bound amortized-time(FIND-SET), our focus is on $\left[rank(x_i), rank(\pi(x_i))\right]$ vs intervals of ranks
4. OBS: Once $\left[rank(x_i), rank(\pi(x_i))\right]$ crossed an interval, it keeps crossing those intervals in the future — this part is at most $\log^* n$
5. O.w. pay the cost by the budget of x_i — If $rank(x_i) \in [k+1, \ldots, 2^k]$, after x_i paid $\leqslant 2^k$ times, $rank(\pi(x_i))$ falls into above case and x_i stops paying

Summary

1. Two algorithms implements the greedy idea
2. Prim's ALG runs in $O(m \log n)$ via heaps (faster by Fibonacci-heap)
3. Kruskal's ALG runs in $O(m \log^* n)$ after sorting

Summary

1. Two algorithms implements the greedy idea
2. Prim's ALG runs in $O(m \log n)$ via heaps (faster by Fibonacci-heap)
3. Kruskal's ALG runs in $O(m \log^* n)$ after sorting
4. For disjoint sets, a small change makes a big difference: from $O(\log n)$ to 5!

Questions?