W3U2: Kruskal's Algorithm Part 2

C Pandu Rangan

Recap

- Partition ADT
- Name Array Representation

Outcomes

- Inverted FOREST data structure
- Complexity analysis of Kruskal's algorithm

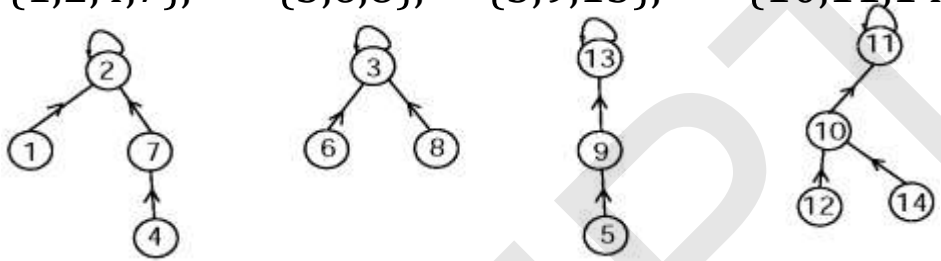
Inverted FOREST data structure

We will now discuss another data structure to implement partition ADT.
This is called in-forest representation.
In-tree is a directed rooted tree where each edge is directed 'towards' the root. We assume that there is a self-loop at the root of an in-tree.
A k-partition is represented by an in-forest with k trees where each set is represented by one tree in the forest.
The shape of the tree is arbitrary and each node of a tree stores one element of the corresponding set. The element stored in the root of a tree serves as the name for the corresponding set.

Example

A 4-partition of $\{1,2,3,\cdots,14\}$, its in-forest representation and the corresponding parent-array representation are shown below:

$\{1,2,4,7\}$, $\{3,6,8\}$, $\{5,9,13\}$, $\{10,11,14,12\}$



$p(1) = 2$ $p(2) = 2$ $p(3) = 3$ $p(4) = 7$

$p(5) = 9$ $p(6) = 3$ $p(7) = 2$ $p(8) = 3$

$p(9) = 13$ $p(10) = 11$ $p(11) = 11$ $p(12) = 10$

$p(13) = 13$ $p(14) = 10$

Union (A, B)

\\ A and B are values at the roots of two different sets

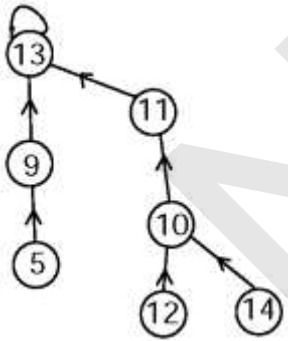
\\ union by "Tree hooking"

Example (contd)

Example (contd)

Note that root has self loop. Hence, $p(x) = x$ iff x is at the root. The value at the root serves as the name of the set represented by the tree.

Union (13,11)



Find (x)

\\ "ancestor chasing"

\\ returns the name of the set containing x.

That is, returning the value at the root of the tree containing x

$y = p(x)$

while $(y \neq p(y))$

$y = p(y)$

Return (y)

Example (contd)

Example (contd)

Complexity

Union - $O(1)$

Find - $O(\text{depth of } x)$

- $O(n)$

Worst case example

Complexity (contd)

More efficient implementation

(weighted merge for union)
We hook the smaller tree with larger tree. We maintain the information on the sizes of the trees we have at any given time.
Size (x), where x is a root value, is the number of nodes in the tree.
Initially, for the partition
$\{1\}, \{2\}, \ldots \{n\}$, we set $p(i) = i \forall 1 \leq i \leq n$
and size $(i) = 1 \forall 1 \leq i \leq n$

Union Algorithm

Union (A, B)

If size $(A) \leq$ size (B)

$p(A) = B$

size $(B) =$ size $(B) +$ size (A)

else

$p(B) = A$

size $(A) =$ size $(a) +$ size (B)

Complexity

No change in Find procedure

Union (A, B) ----------$O(1)$

Find (x) ----------$O(\log n)$

This is because:

- Initially depth $(x) = 0$
- For every increase in depth of x by 1, the size of the set containing x atleast DOUBLES!!
- Thus, x can not go deeper than $\log n$ levels
- Since x is arbitrary, the complexity of Find(x) is $(\log n)$

Pseudocode

Initialize:

$A = \emptyset$,

$e =$ First edge of L;

Initialize the names of the connected components of

$T = (V, A)$

Process:

While (NOT END OF L)

Let $e = (x, y)$

$X =$ Find (x);

$Y =$ Find (y);

If $(X \neq Y)$

Add e to A

Union (X, Y)

$e =$ Next (e, L)

Output:

$T = (V, A)$

\\ T is a MST of G

Complexity (contd)

Complexity of Kruskal's algorithm

- Kruskal's algorithm requires a sorted list of edges and this sorted list can be generated in $O(mlogm) = O(mlogn)$ time.
- During the extension, Kruskal's algorithm performs $(n-1)$ union operations and $2m$ find operations.

Summary

If we NAME array to implement partitions the complexity is

$O\big(n(n-1)\big) + O\big(1 \cdot (2m)\big) = O(n^2 + m).$

If we use Parent array (weighted merge) the complexity is

$O\big(1 \cdot (1m)\big) + O(2m \cdot logx) = O(n + mlogn)$

We may treat the cost of generating L as a preprocessing cost that is independent of the data structures and may ignore the same for comparison purposes.

Name array base implementation	Better for dense graphs
Parent array base implementation	Better for sparce graphs

This is because of n^2 term found in first analysis and $mlogn$ term found in the second analysis.

Thank You