Lec03-ADT-union-find

KT 2.5, 4.6

Tuesday, August 27, 2024 1:58 PM

We just finished proving correctness of MST algorithms.
Let's circle back around to implementation

Data structures are design specs

- how to store data in memory — storage
- what ops (operations) we can perform on data — ops
- the algs for those ops — algs
- time & space efficiency of algs — complexity

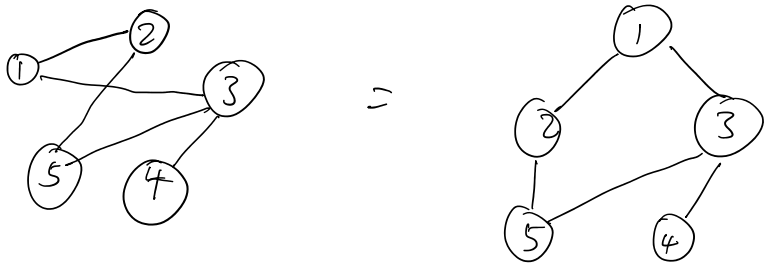
} organization of data can lead to speed

ADT (abstract data type) omits implementation, but says what should happen

Ex. Graph ADT G

G.add_vertex(u) – adds a vertex to G with label u.
G.add_edge(u, v, d) – adds edge {u,v} with weight d
G.has_edge(u,v) – returns True iff $\{u,v\} \in E_G$
G.neighbors(u) – returns list of vertices adjacent to u in G
G.weight(u,v)

Multiple drawings can represent same graph

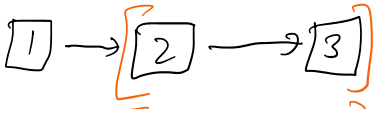


Option 1: Adjacency matrix

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ \end{bmatrix}$$

$A_{ij} = 1$ iff $\{i,j\} \in E_G$

Option 2: Adj list

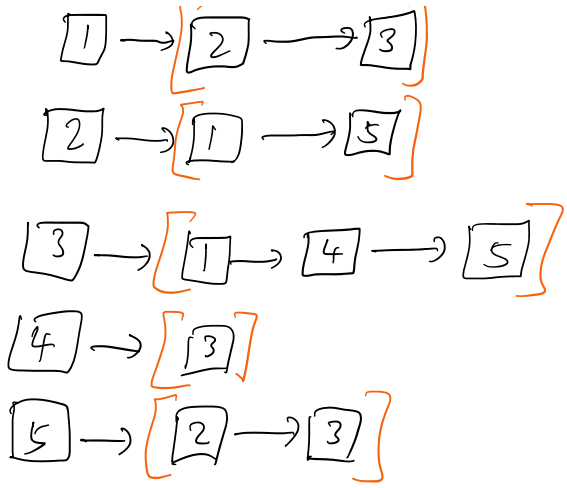


$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$A_{ij} = 1$ iff $\{i,j\} \in E_G$

or

$A_{ij} = w(\{i,j\})$



Option 3: Edge list

{1, 2}
{1, 3}
{2, 5}
{3, 4}
{3, 5}

How to impl weights?

How long to find neighbors?

What about adding or deleting an edge or node?

How do we check in Kruskal if adding an edge (u, v) would create a cycle!

- Would create cycle if u, v in same connected component already.
- We start with a component for each node.
- Components merge when we add an edge.
- Need way to check if u, v are in same component & to merge two components into one.

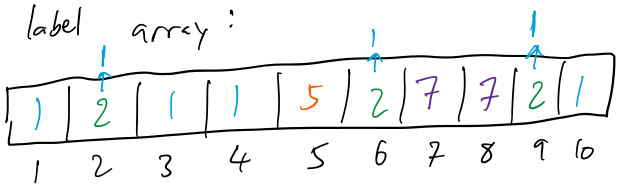
Union-Find Abstract Data Type (KT 4.6)

Operations:

- MakeUnionFind(S) - create data structure containing |S| sets, each containing one item from S.
- Find(i) - return label of set containing i.
- Union(a, b) - merge sets a, b into single set.

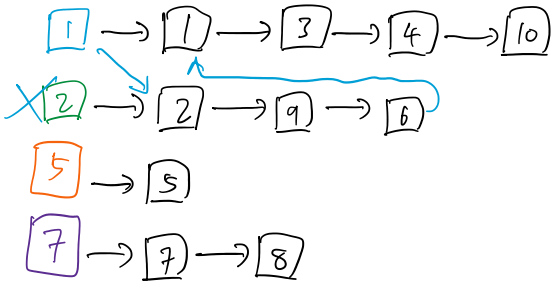
Array-based Union-Find

Set label array:



← says which set an item is in

Items lists:



Sizes

1	4
2	3
5	1
7	2

} 1 7

Make Union Find (S): create data structures, initialize to |S| sets

$O(|S|)$ time (proportional to |S| time)

Find(i): Return U.F. sets[i]

$O(1)$ time (constant time)

Union(x,y): Use size array to determine which set is smaller

WLOG, assume $|x| < |y|$

Walk down elements i in set x, setting sets[i] = y

Set size[y] = size[y] + size[x]

Make y point to start of x list and end of y list point to old start of y list

prepend smaller list to larger list

Let's compute runtime of array-based union-find

Thm Any seq of k union operations on a collection of n items takes $O(k \log k)$ time

Proof. After k unions, at most $2k$ items have been involved in a union

(each union can touch at most 2 new items if it was a union of two singleton sets, often only 0 or 1 new items touched)

For any item v, set[v] changes at most $\log_2(2k)$ times ← total number of items possible in largest set

because each time set[v] changes, |set[v]| at least doubles

(since we update labels of smaller set)

At most $2k$ items updated at most $\log_2(2k)$ times each

$\Rightarrow 2k \log_2 2k$ work

$\Rightarrow O(k \log k)$ work.

(Size of item list updates constant per update so $O(k)$)

Running time of Kruskal using arrays

Sorting edges $\approx m \log m$ for m edges

$m \le n^2$, so $\log m \le \log n^2 = 2 \log n$

$\approx m \log n$

At most $2m$ "find" operations $\approx 2m$ time

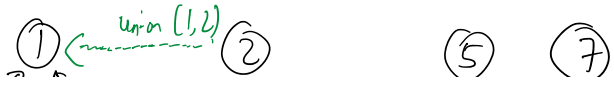
(to check if u, v in same component)

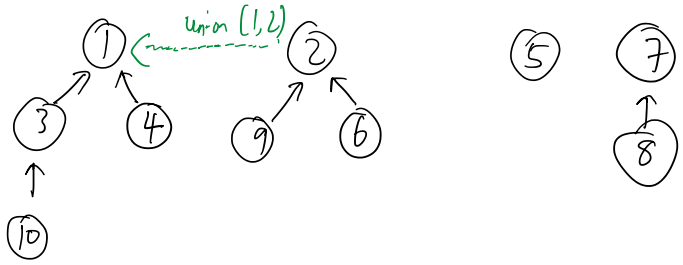
At most $n-1$ union ops $\approx n \log n$ time

$\Rightarrow$ total runtime $\approx m \log n + 2m + n \log n$ $O(m \log n)$

largest term since $m \ge n$ if graph is connected and not already a tree)

Tree-based union-find





MakeUnionFind(S): create $|S|$ single-node trees. $\Theta(S)$ time

Find(i): Follow pointers from i to root of tree.

Union(x,y): If $|x| < |y|$

Thm Find(i) takes $\Theta(\log n)$

proof. set[i] is renamed at most $\log_2(n)$ times

because each renaming doubles the size.

The depth of a tree is the number of renamings for an item.

Runtime of Kruskal using trees:

Sorting edges $\Theta(m \log n)$

$\leq 2m$ "find" ops: $\log n$ time each $\rightarrow 2m \log n$

$\leq n-1$ union ops: $\Theta(n)$ time

Total running time $\approx m \log n + 2m \log n + n = 3m \log n + m$

$= \Theta(m \log n)$