

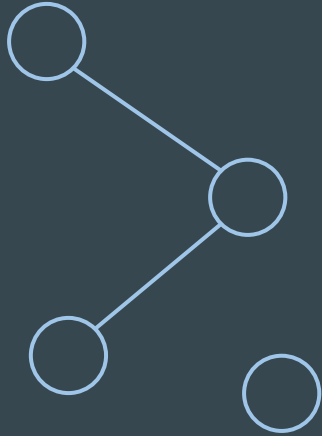


Graph Theory 3

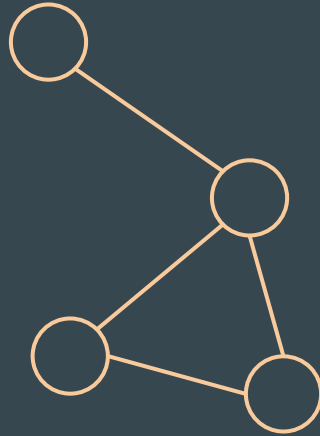
...

Minimum Spanning Tree

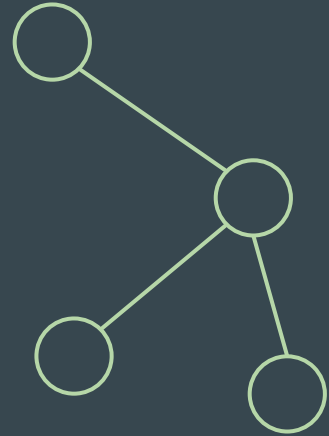
Reminder : what is a tree ?



acyclic graph



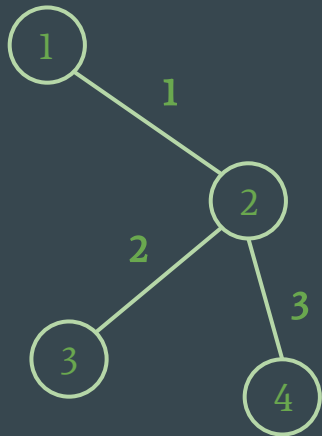
connected graph



tree = connected + acyclic

Some properties of trees...

- A tree has $N-1$ edges for N vertices
- A tree is a connected & acyclic graph
- Remove any edge from the tree and it disconnects the graph
- In the general case vertex can have any degree
 - Some trees are more specific : binary trees, k-trees



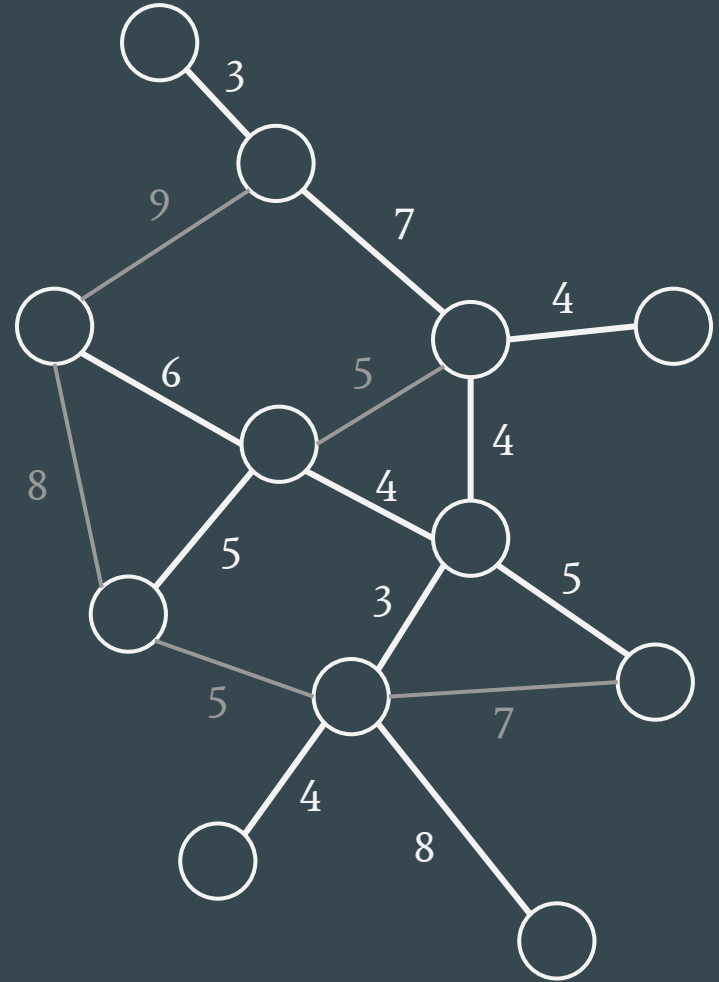
Minimum spanning tree

We are given a connected graph with weighted edges

→ what is the minimum cost to connect all the nodes?

(on this graph: 53)

/!\ Multiple MST are possible



Prim's algorithm

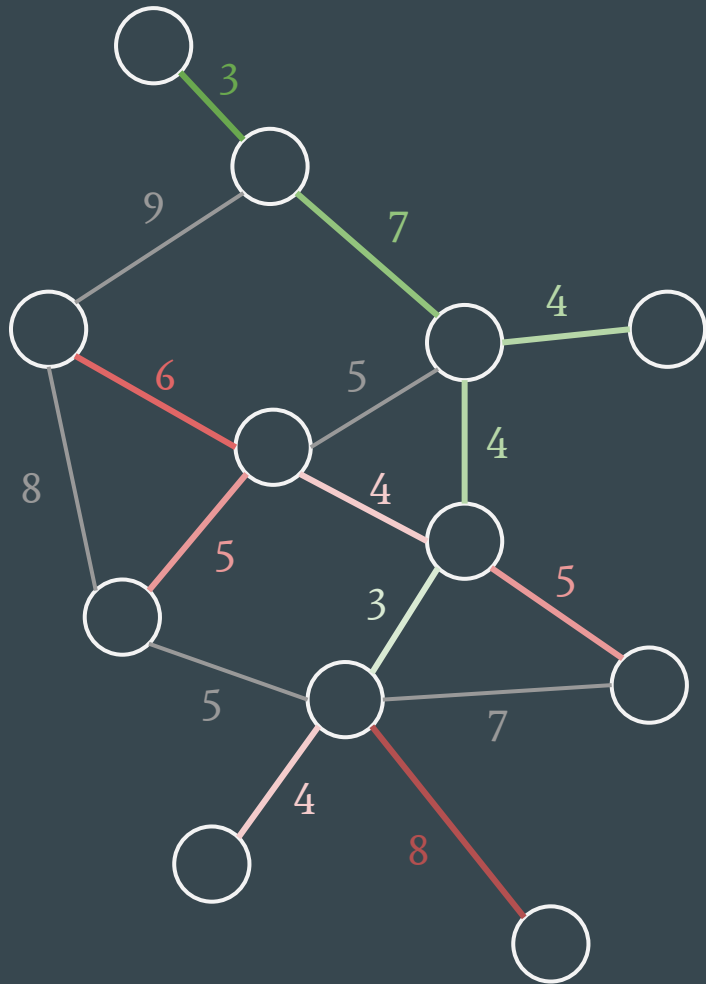
Greedy algorithm:

- put a random starting node in the tree
- while the tree doesn't have all the nodes:
 - select the node closest to the tree
 - add the corresponding edge in the tree

Implemented with a heap (remember Dijkstra last week ?)

→ Complexity : $O((|V| + |E| \log |E|))$
see hq op complexity)

(on the right, from green to red, order of addition in the tree)



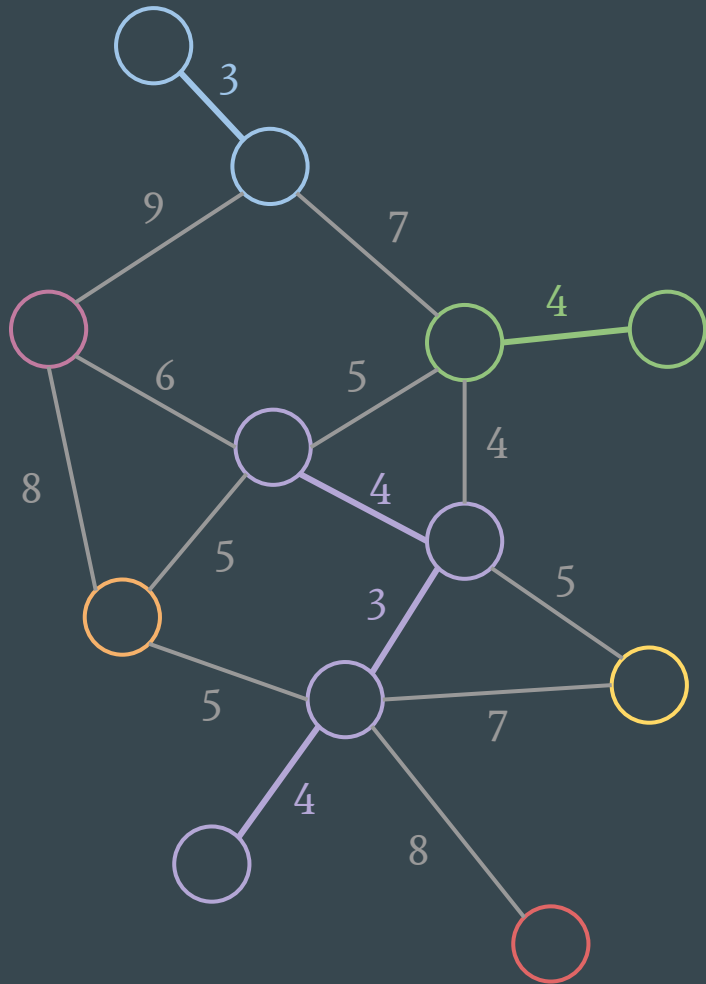
Kruskal's algorithm

Greedy algorithm:

- create a trivial forest (all nodes are alone)
- as long as we can, we use the smallest edge that creates a bridge between two trees

(on the right, example of a forest at some point during execution)

How to avoid making cycles ?

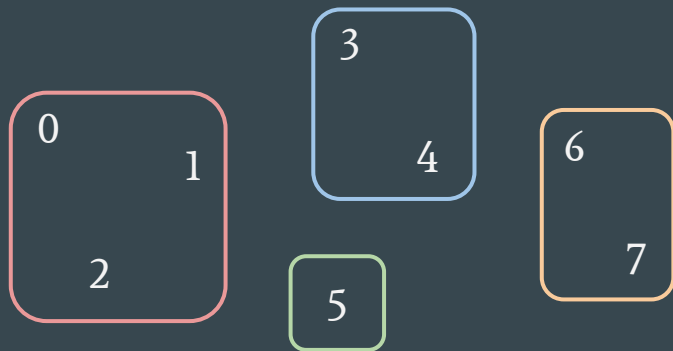


Disjoint sets (union-find d.s.)

We need a data structure to store disjoint sets with the following near-constant-time operations:

- add a new set
- merge two sets
- determine whether two elements are in the same set

Naive idea: let's use a list of sets



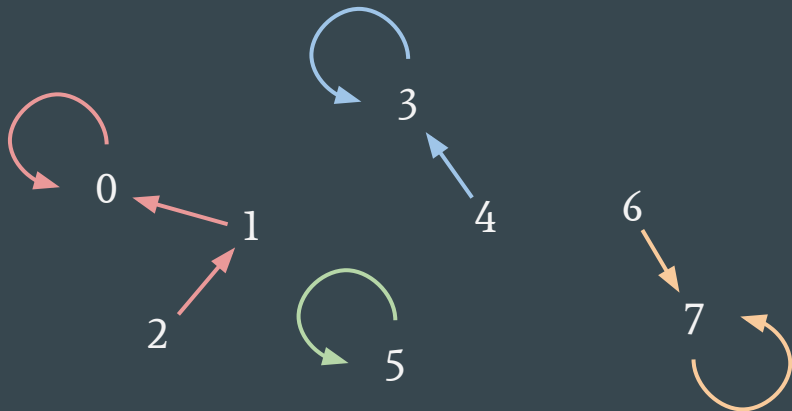
$[\{0,1,2\}, \{3,4\}, \{5\}, \{6,7\}]$

Add a new set	$O(1)$
Merge two sets	$O(n)$
Determine whether 2 elements are in the same set	$O(n)$

Disjoint sets (union-find d.s.)

Better idea: let's use a forest

- each node has a parent, the root of every tree is its own parent
- to merge two sets, point the root node of a set to any node of the other (you will usually use the *find* operation before, to check if the two sets are disjoint or the same)
- to determine whether 2 elements are in the same set, find the roots of their trees and compare them



$\{0:0, 1:0, 2:1, 3:3, 4:3, 5:5, 6:7, 7:7\}$

Add a new set	$O(1)$
Merge two sets	$O(s_1 + s_2)$
Determine whether 2 elements are in the same set	$O(s_1 + s_2)$

Disjoint sets (union-find d.s.)

How to improve complexity?

→ try to have shallow trees

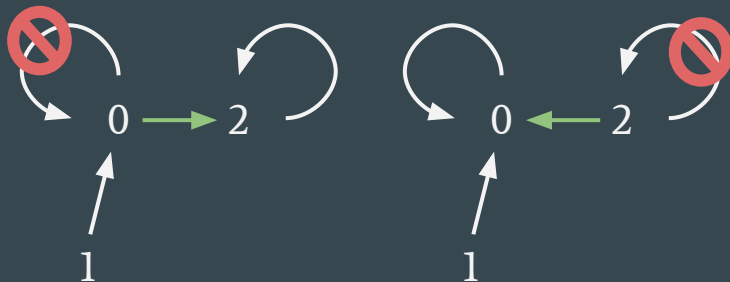
→ this is achieved with **union by rank** (or *by size*)
and **path compression** (or *path halving* or *splitting*)

Add a new set	$O(1)$
Merge two sets	$O(\alpha(n))$ *
Determine whether 2 elements are in the same set	$O(\alpha(n))$ *

* $\alpha(n) < 5$ for any n that can be written in this physical universe,
so that's basically $O(1)$ (see **inverse Ackermann function**)



path compression on the path from a node to its root during a *find* operation



arbitrary union (left) VS *union by rank* (right)

Union-find in Python

- Bad news : this structure isn't implemented by default in Python
- Good news : Louis Sugy [made a library](#) for us 2 years ago !
- Create a Union-find data structure with $O(1)$ approximate complexities
- Use `find(element)` to get the root of an element
- Use `union(e1, e2)` to merge the sets containing `e1` and `e2`
- Use `is_same_set(e1, e2)` to determine whether `e1` and `e2` are in the same set

Back to Kruskal

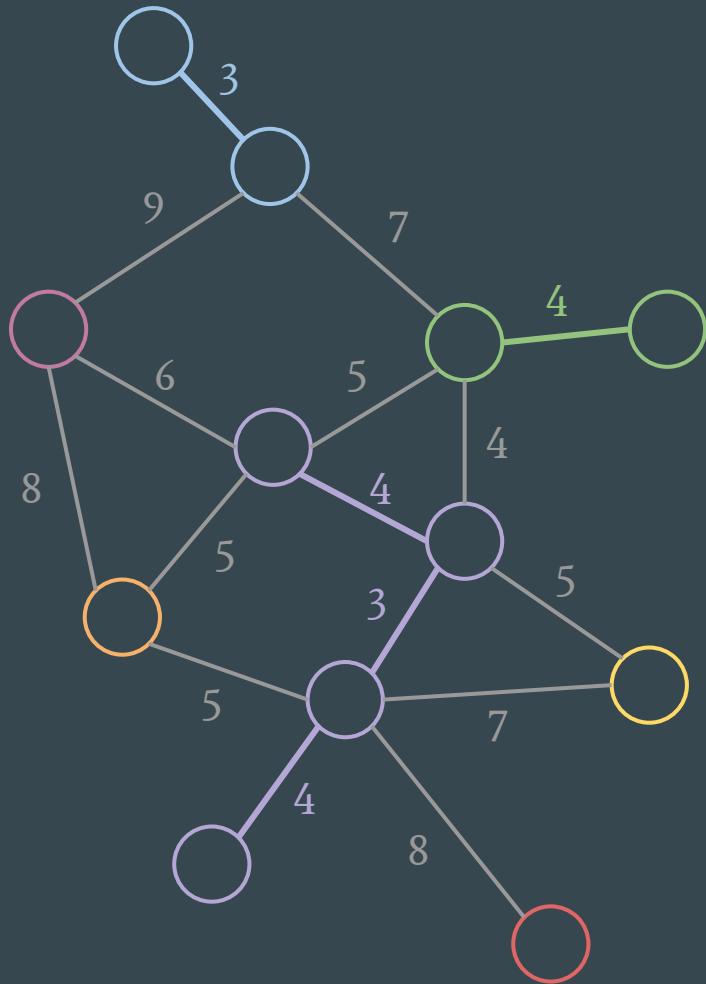
Greedy algorithm:

- create a trivial forest (all nodes are alone)
- as long as we can, we use the smallest edge that creates a bridge between two trees

Complexity?

→ $O(|E| \log |E|)$ with an optimized Union-Find data structure and a heap to select the next edge

Note: Bernard Chazelle has found a $O(|E| \alpha(m, n))$ solution based on **soft heaps**, but that's a story for another time



Your mission, should you choose to accept it...

Implement the Kruskal algorithm

You can use our implementation of the Union-Find data structure or try to implement your own (with or without optimizations). See:

- https://en.wikipedia.org/wiki/Kruskal's_algorithm
- https://en.wikipedia.org/wiki/Disjoint-set_data_structure

Templates and tests on GitHub: [INSAlgo/trainings-2018 \(W14\)](#)

(there is also my solution there)

Credits

Slides: Louis Sugy for INSAIgo

Edited in 2020

Thanks to:

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]. *Introduction to Algorithms* (3rd ed.), chapter 23
- Wikipedia