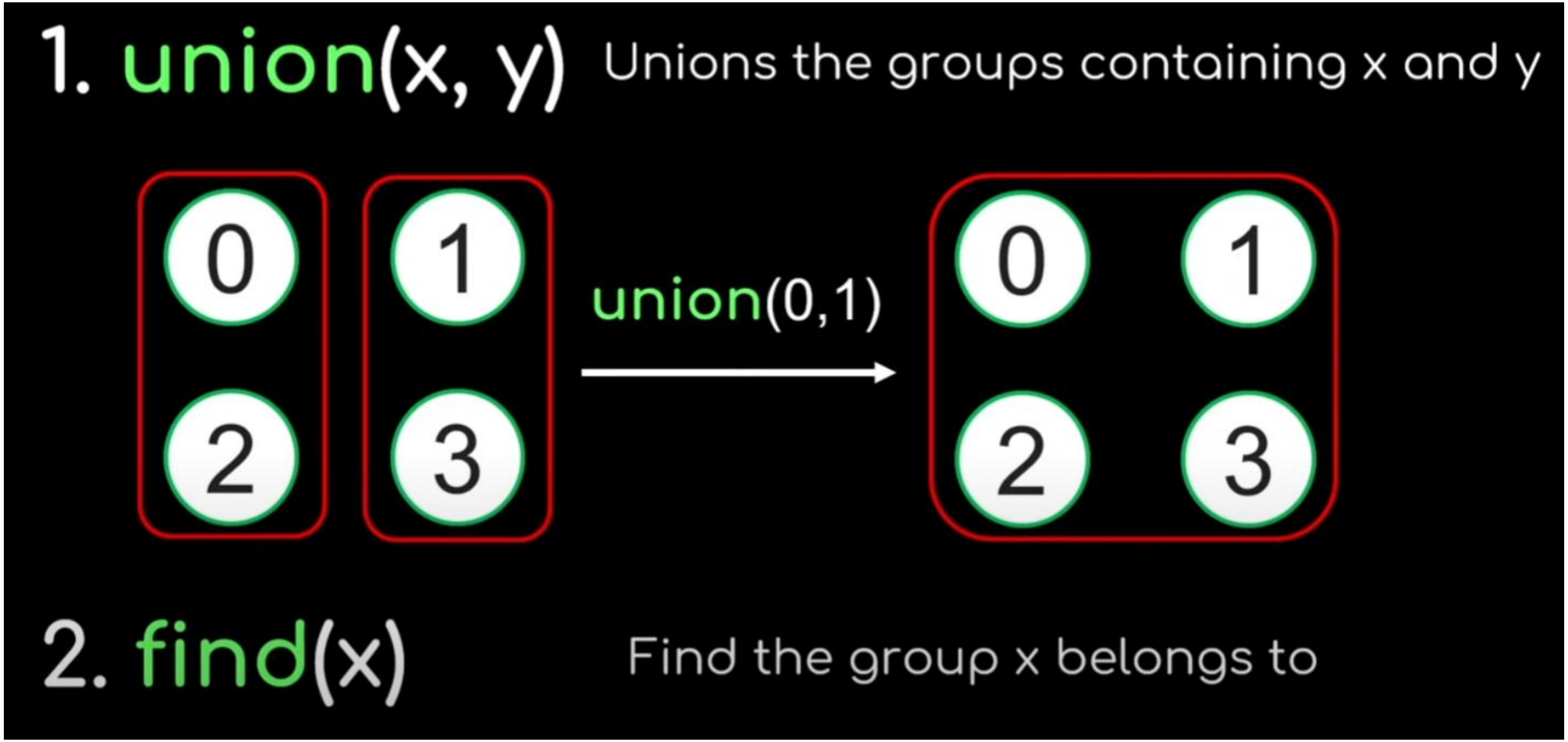
Minimal Spanning Tree and applications in clustering

Mustafa Hajij

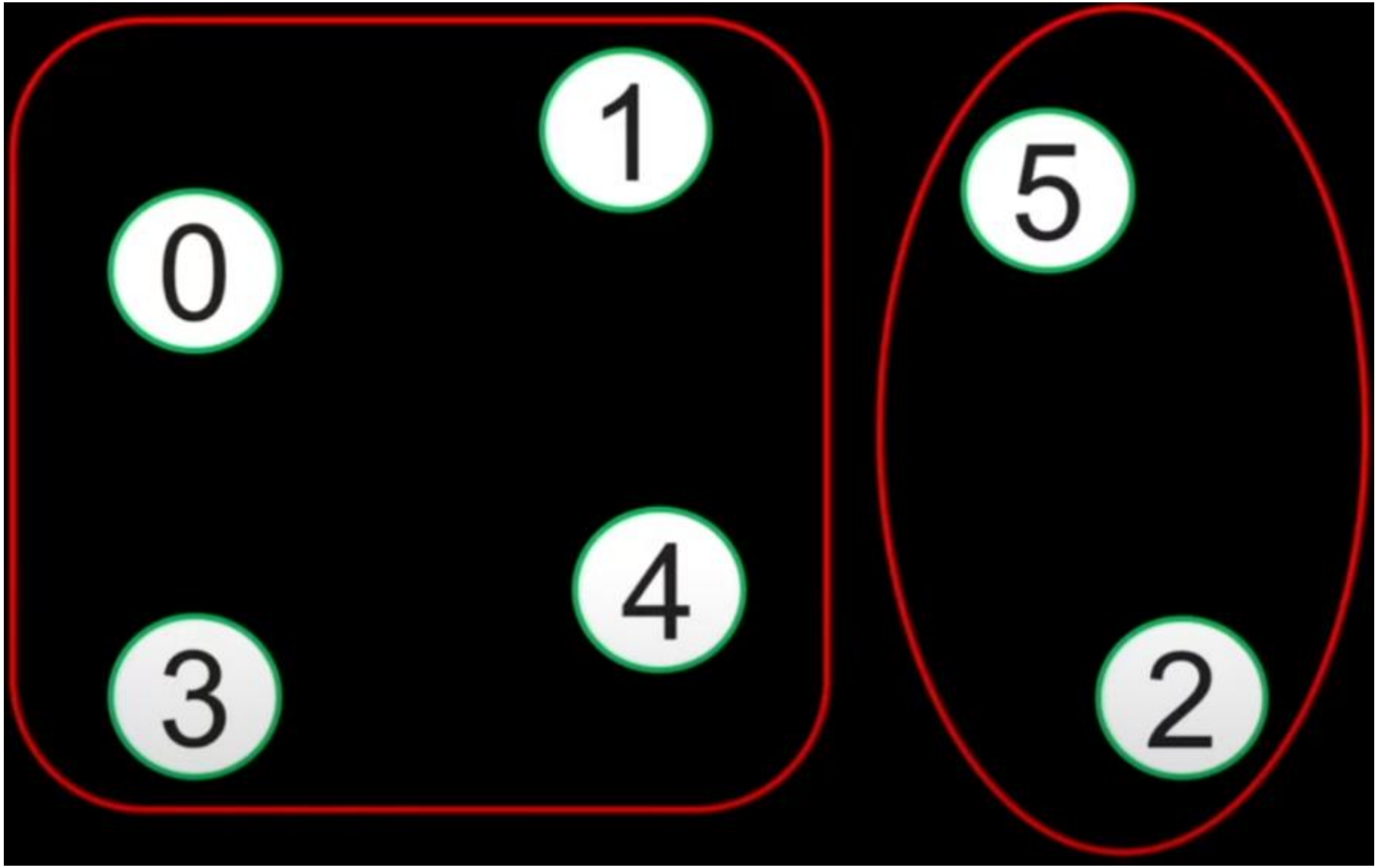
A quick introduction to Union-Find



We want a data structure that can do the above two operations easily and quickly (Union and Find).

Ref :https://www.youtube.com/watch?v=ayW5B2W9hfo

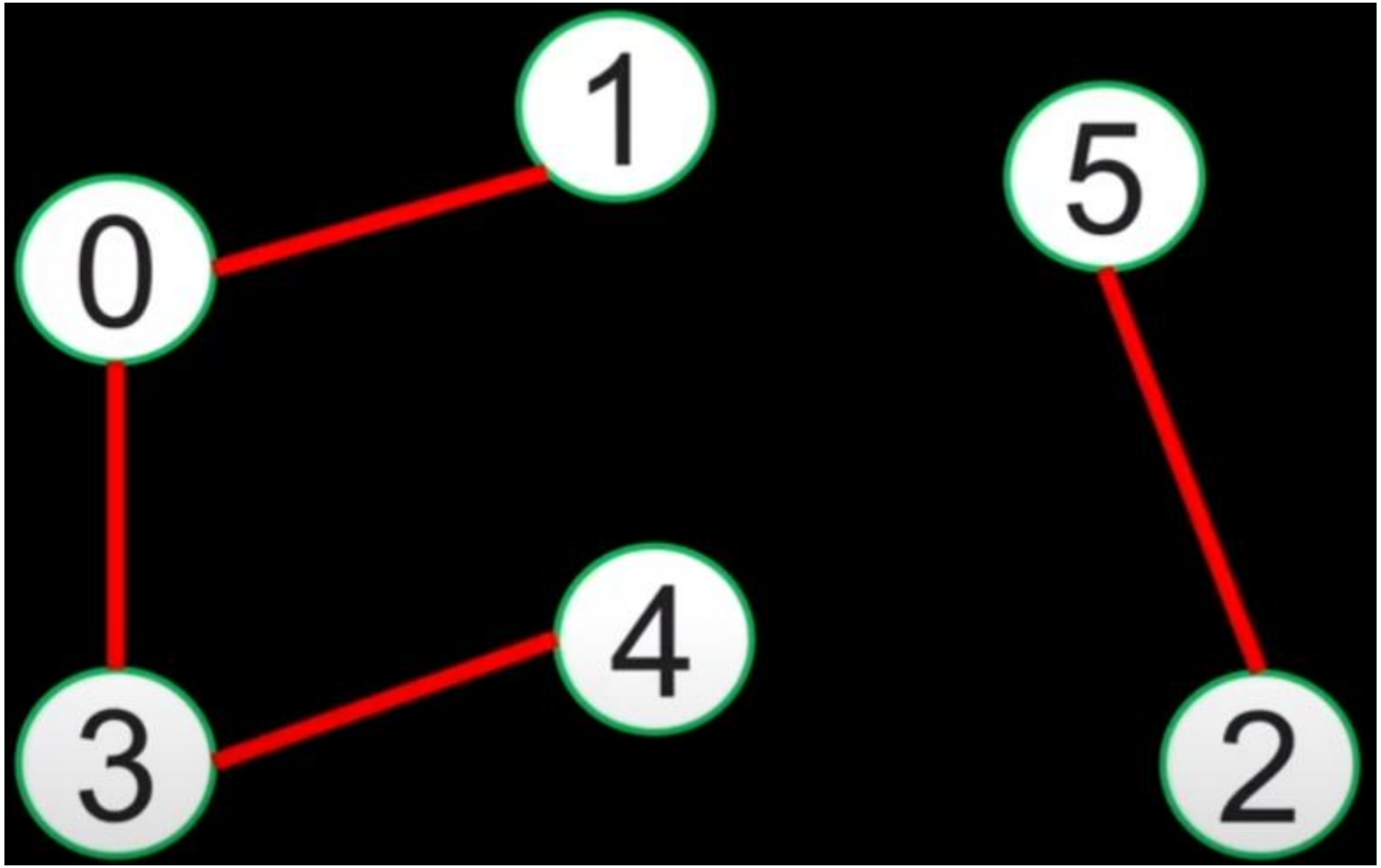
A quick introduction to Union-Find



Lets discuss find first.

Ref :https://www.youtube.com/watch?v=ayW5B2W9hfo

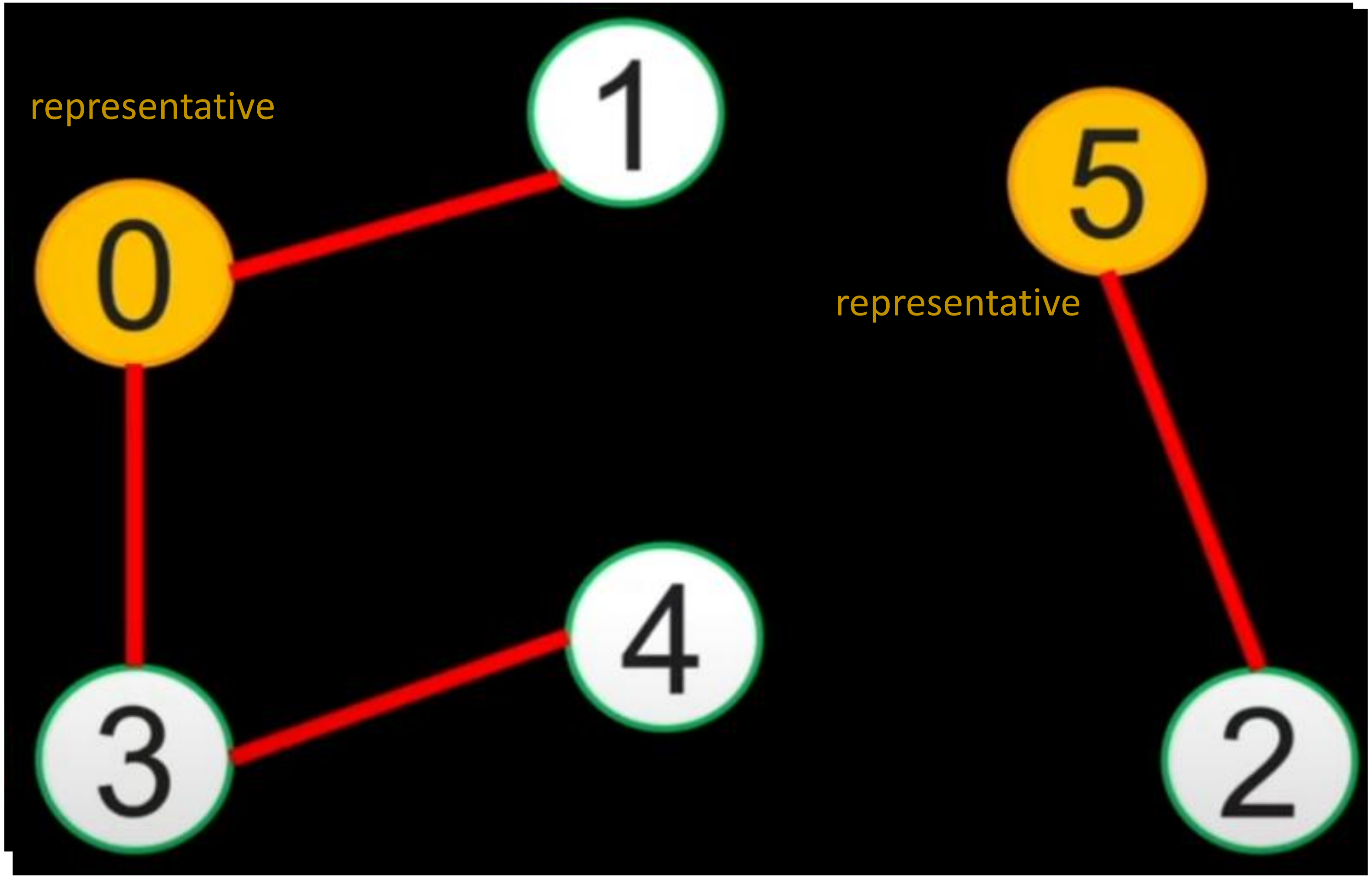
A quick introduction to Union-Find



Connect all elements that belong to the same group by edges

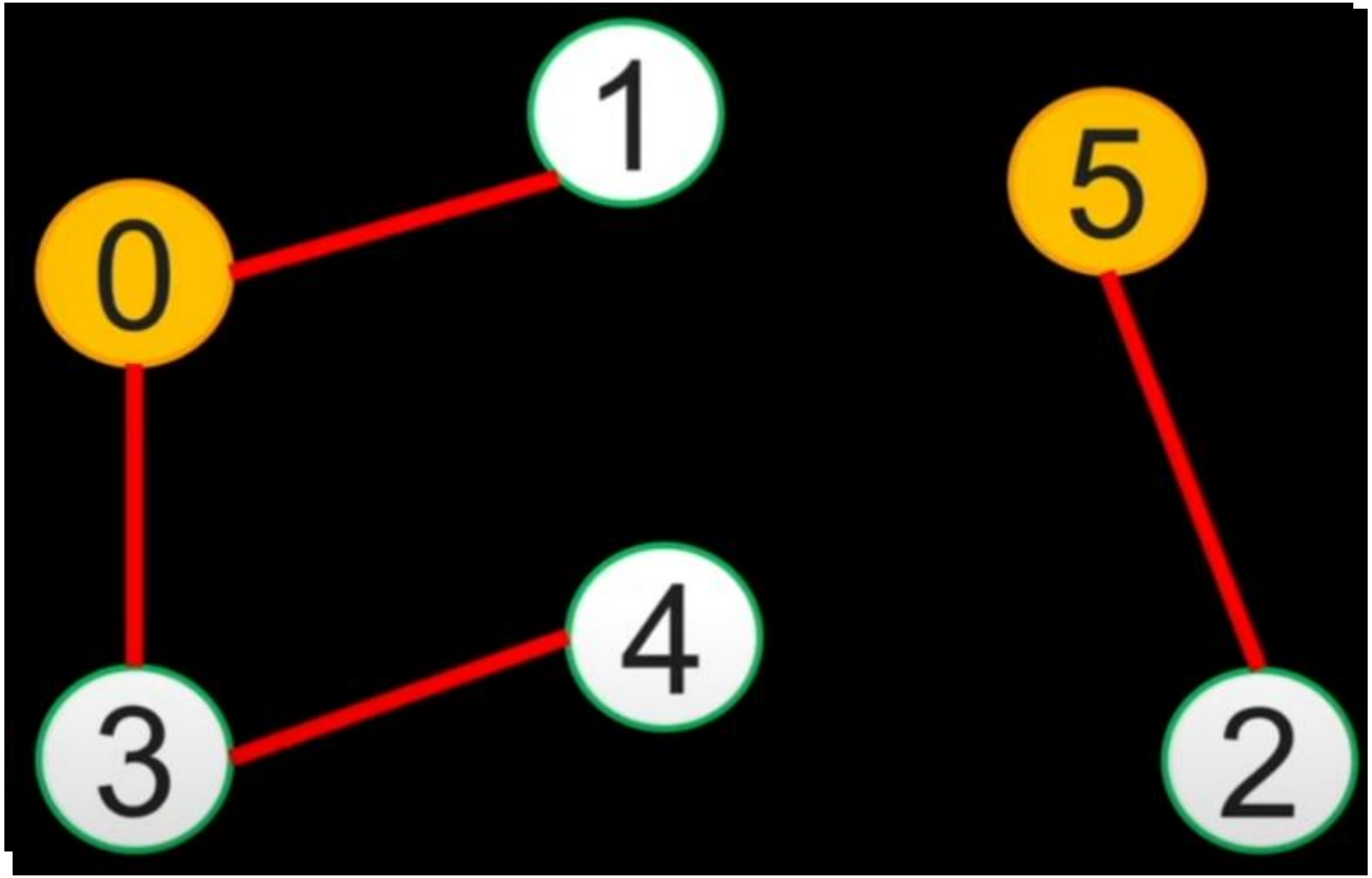
Ref :https://www.youtube.com/watch?v=ayW5B2W9hfo

A quick introduction to Union-Find



- Next we set a representative to each group.
- We design the data structure such that find(4)=0 and find(2)=5   Ref :https://www.youtube.com/watch?v=ayW5B2W9hfo

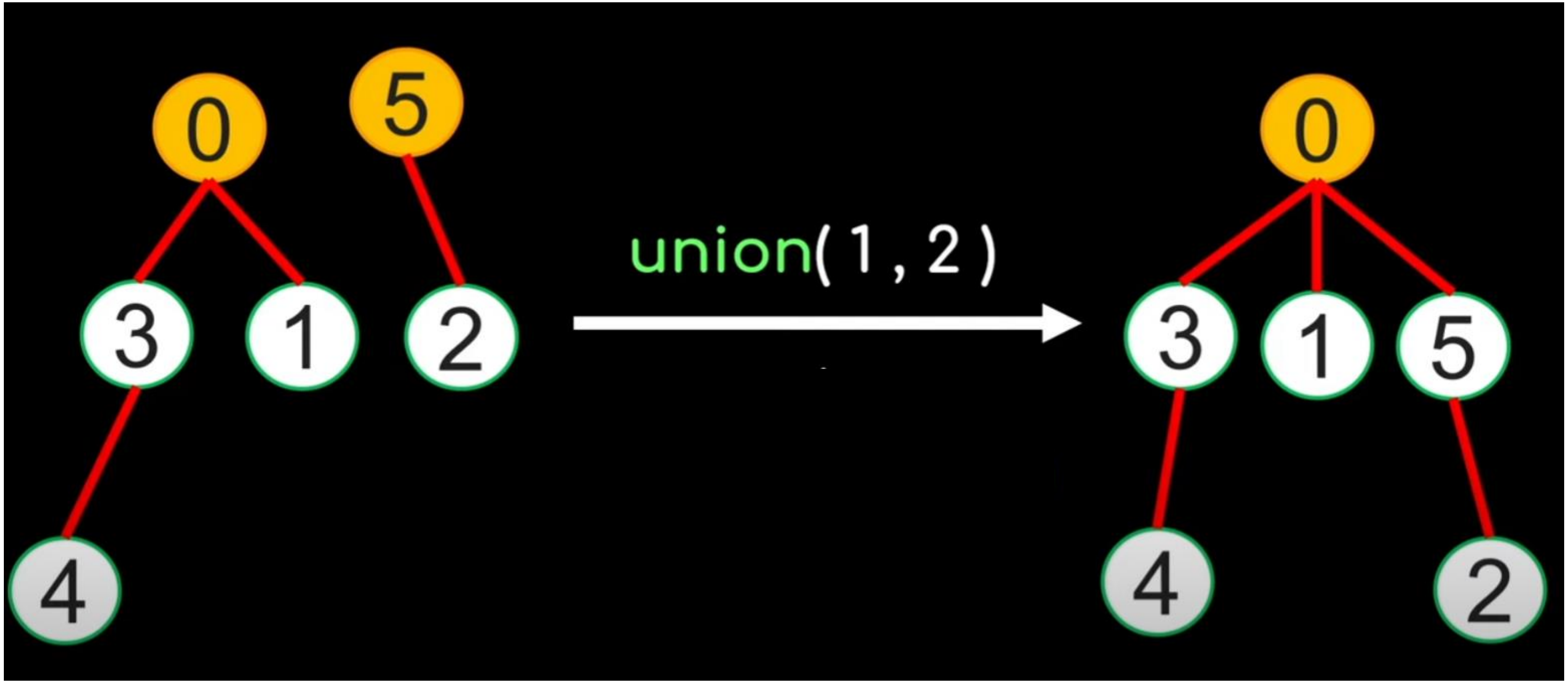
A quick introduction to Union-Find



- Two elements u and v belong to the same group iff find(u)=find(v)

Ref :https://www.youtube.com/watch?v=ayW5B2W9hfo

A quick introduction to Union-Find



- Union (say union(2,1)) is a matter of merging the trees together!

Ref :https://www.youtube.com/watch?v=ayW5B2W9hfo

A quick introduction to Union-Find

In short a union find data structure allows for three operations :

(1) MAKE-SET(v) : Make a connected component from the node. $O(1)$

(2) FIND-SET(u) :given a node u, returns a pointer to the connected component it belongs to. $O(\log(n))$*

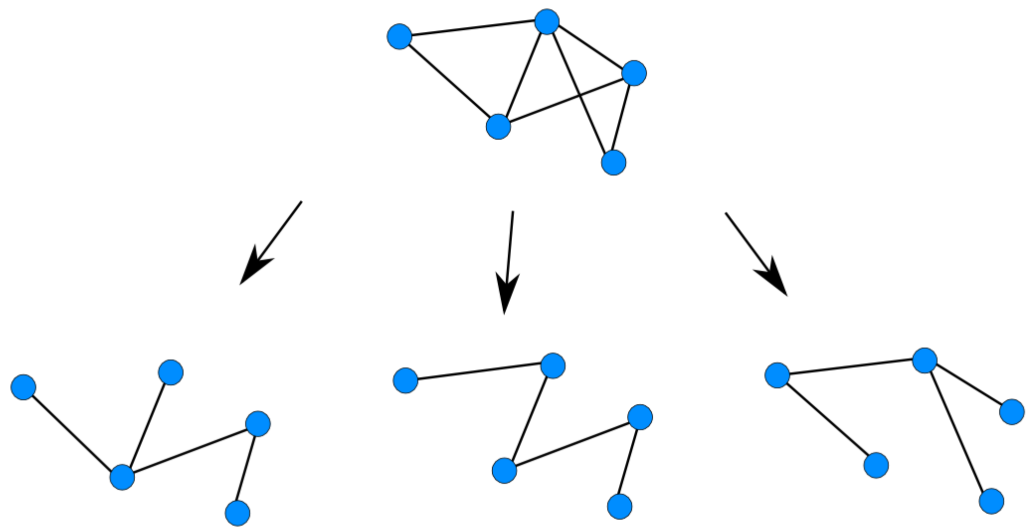
(3) UNION(u, v): Given two nodes u, v that may belong to two separate connected components, merge these two separate connected components into a single one. $O(\log(n))$*

* The optimal complexity is in fact $O(\alpha(n))$, $\alpha(n)$ is the extremely slow-growing inverse Ackermann function.

Spanning Tree

Let $G = (V, E)$ be a connected weighted graph. A spanning tree for G is a subgraph of G which includes all of the vertices of G and is a tree.

A graph might have more than one spanning tree

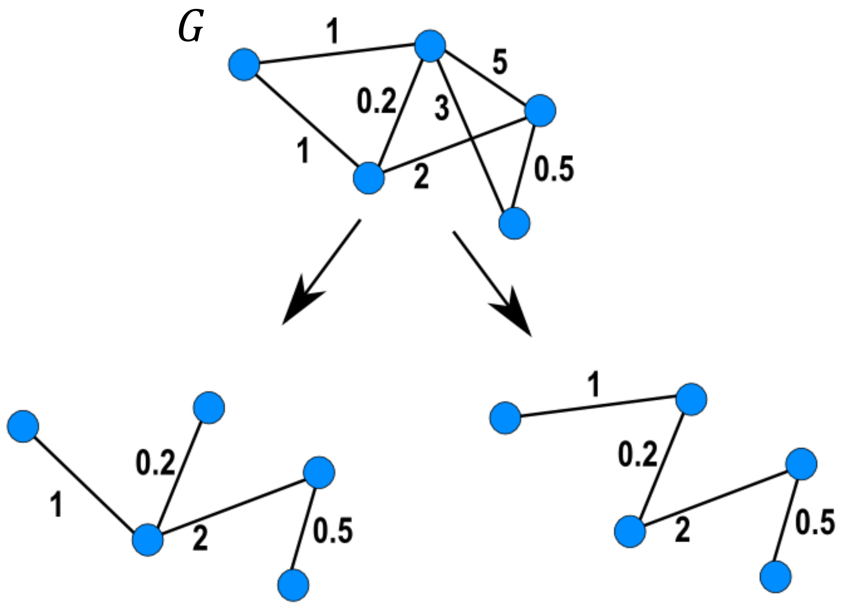


Spanning trees for G

Minimal Spanning Tree

Let $G = (V, E, w)$ be a connected weighted graph. A minimal spanning tree for G is a spanning tree whose sum of edge weights is as small as possible.

A graph might have more than one minimal spanning tree. However, if all edges in the graph have unique weights then the minimal spanning tree is unique.



Minimal spanning trees for G

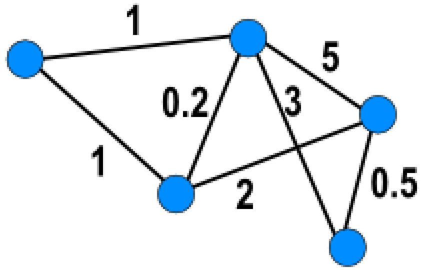
Kruskal's Algorithm

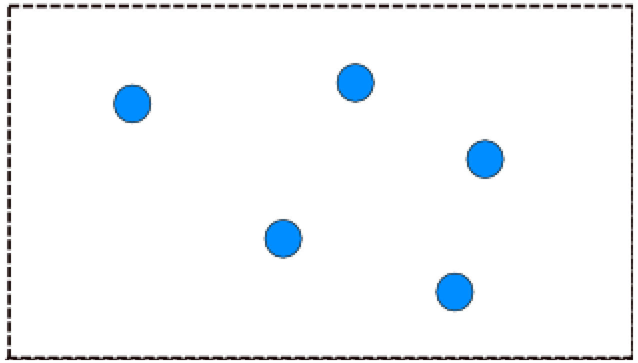
Let $G = (V, E, w)$ be a connected weighted graph. The Kruskal's algorithm is a greedy algorithm.

Informally, the algorithm can be given by the following three steps :

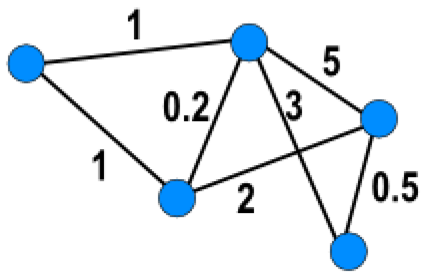
1. Set V_T $to\ be\ V, Set\ E_T = \{\}$. Let $S = E$
2. While S is not empty and T is not a spanning tree
   1. Select an edge e from S with the minimum weight and delete e from S.
   2. If e connects two separate trees of T then add e to E_T

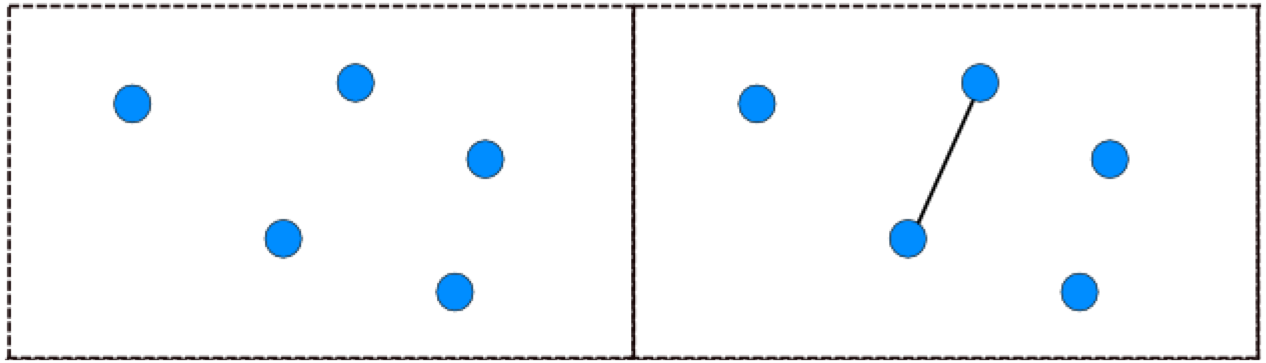
Kruskal’s Algorithm Example



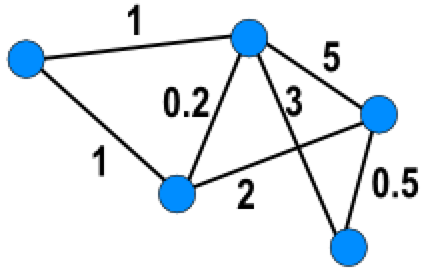


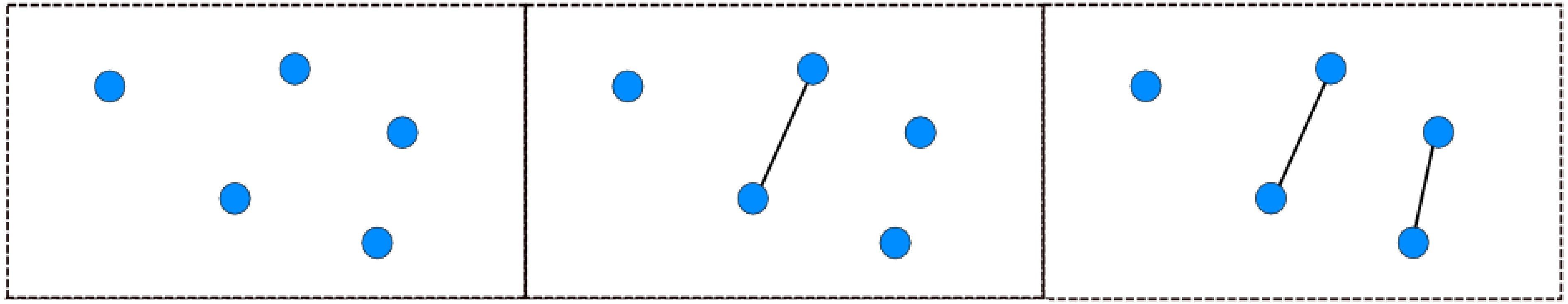
Kruskal's Algorithm Example



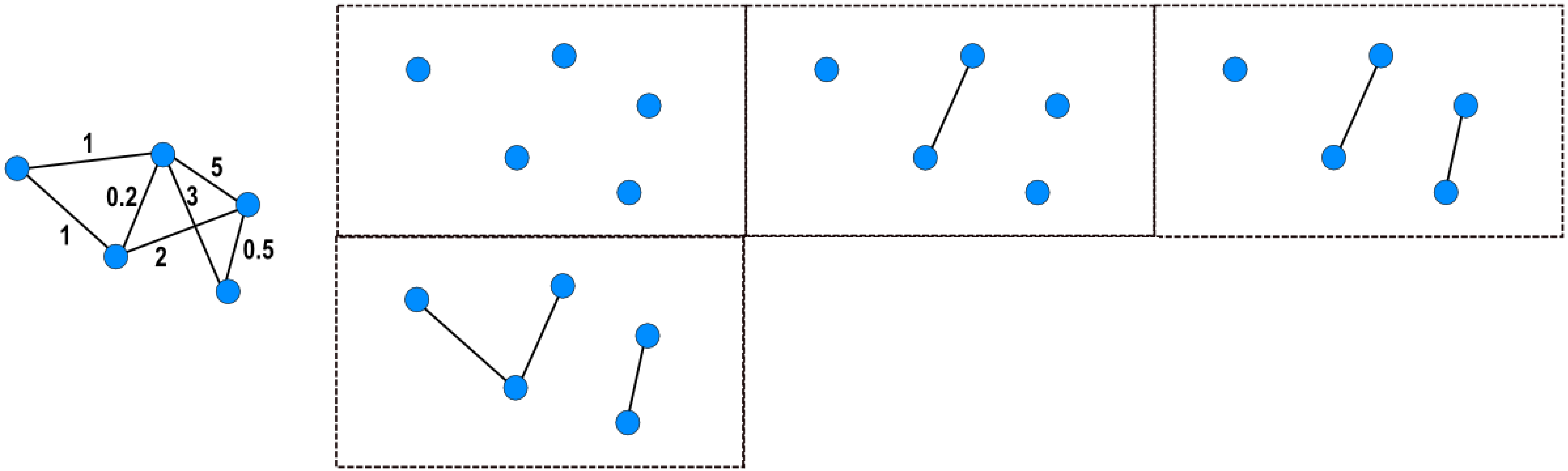


Kruskal’s Algorithm Example

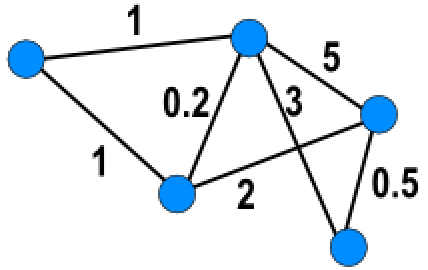


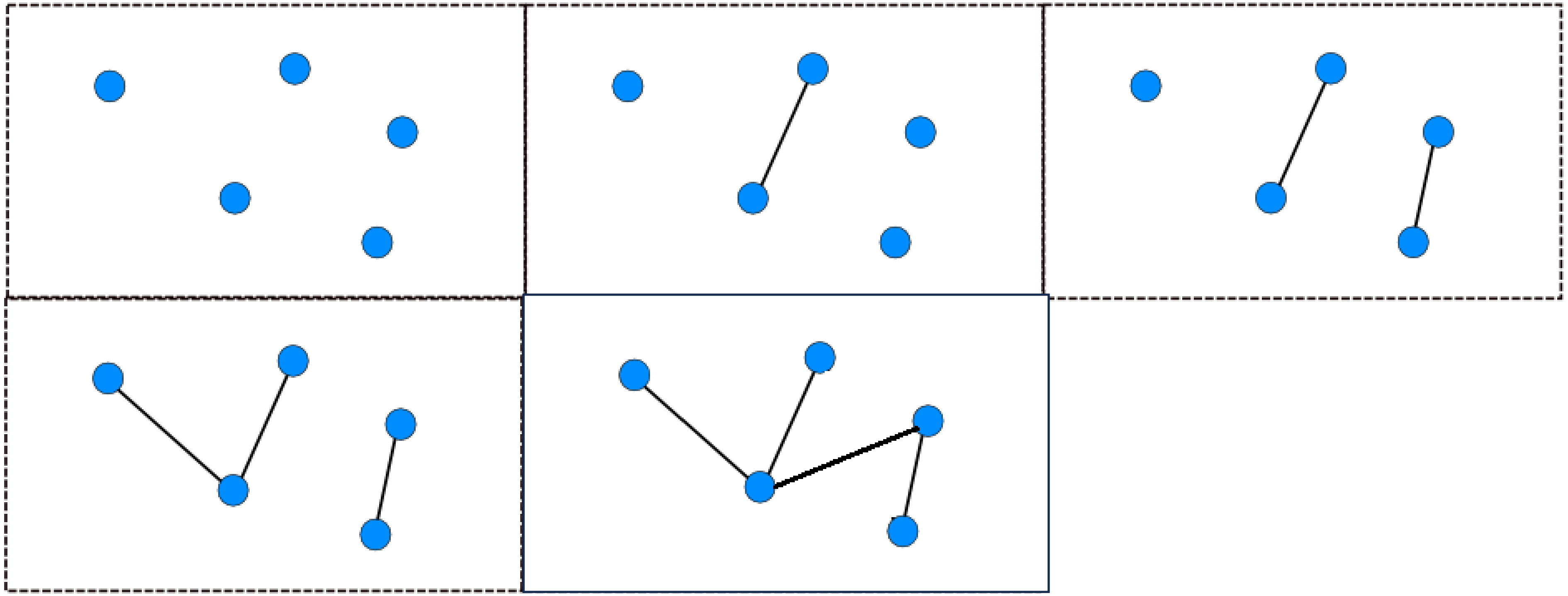


Kruskal’s Algorithm Example



Kruskal's Algorithm Example





Kruskal's Algorithm

Let $G = (V, E, w)$ be a connected weighted graph. The Kruskal's algorithm is a greedy algorithm

This can be implemented using union-find data-structure

1- A= {}
2-foreach $v \in V$:
3- MAKE-SET(v)
4-foreach (u, v) in E ordered by weight(u, v), increasing:
5- if FIND-SET(u) ≠ FIND-SET(v):
6- $A = A \cup \{(u, v)\}$
7- UNION(u, v)
8-return A

Prim's Algorithm

Let $G = (V, E, w)$ be a connected weighted graph. The Prim's algorithm is a greedy algorithm

Informally, the algorithm can be given by the following three steps :

1. Select an arbitrary vertex v from V. Set $V_T = \{v\}\ and\ E_T = \{\quad\}$
2. Grow the tree by one edge : choose an edge e(u,v) from the set E with the lowest cost such that u in V_T and v is in $V \backslash V_T$ then add v to V_T and add e to E_T
3. If $V_T = V$ break, otherwise go to step 2.

Application to Clustering : Zahn's algorithm

Suppose that we are given a set of a weighted graph G.

1. Construct the MST of G (using say Kruskal's algorithm).

2. Remove the inconsistent edges to obtain a collection of connected components (clusters).

3. Repeat step (2) as long as the termination condition is not satisfied.

In this case, an edge in the tree is called inconsistent if it has a length more than a certain given length L
Zahn's algorithm that we used to obtain a clustering algorithm on point cloud can be simply used to obtain a clustering algorithm on graphs as follows. The connected components of the remaining forest are the clusters of the graph

Question : how can you apply this algorithm to point cloud?