Algorithmic Strategies 2024/25

Week 11 – Graph Algorithms

· U C ·

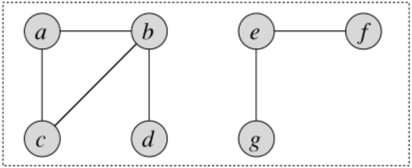
Universidade de Coimbra

Outline

1. Introduction
2. Kruskal Algorithm
3. Union-Find Algorithm

Introduction

- Union-find performs operations on a set of elements that is partitioned into a number of disjoint subsets.
- It allows to add and to merge sets as well as to identify whether elements are in the same set, in almost constant time.
- It is mostly used within graph applications, for instance, in Kruskal Algorithm to find the minimum spanning tree.

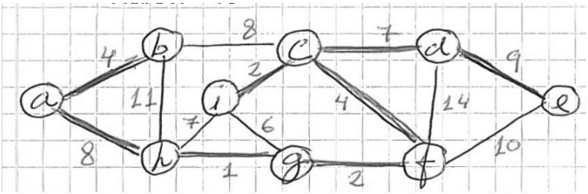


How to merge the two graphs?

Kruskal Algorithm for the minimum spanning tree problem

- A spanning tree of graph $G = (V, E)$ is a tree $T = (V, E^*)$ where the edge set $E^* \subseteq E$ defines a tree that connects all vertices in V. Note that $|E^*| = |V| - 1$
- A minimum spanning tree in G is a spanning tree such that the total sum of the weights of its edges is minimal.
- Kruskal Algorithm is a greedy algorithm that finds a minimum spanning tree with the following greedy choice: choose the next edge with the least weight that does not generate a cycle.
- In order to be efficient, it requires Union-Find operations.

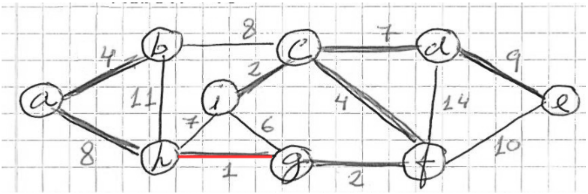
Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



Sort edges in nondecreasing order of the weights

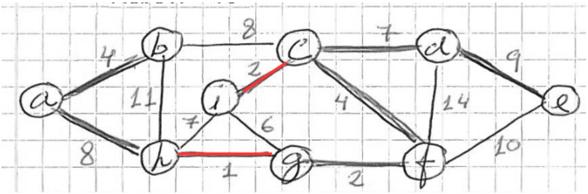
Kruskal Algorithm

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



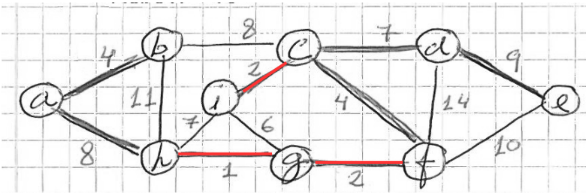
Select edge $\{h, g\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



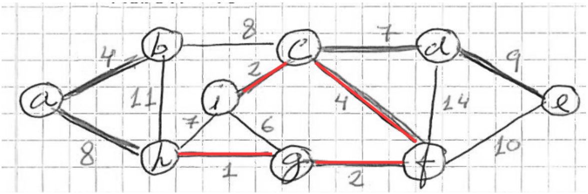
Select edge $\{i, c\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



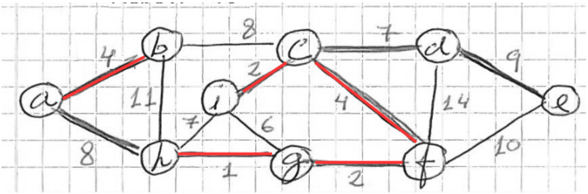
Select edge $\{g, f\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



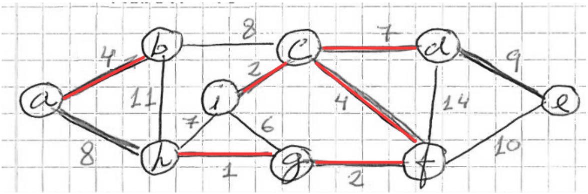
Select edge $\{c, f\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



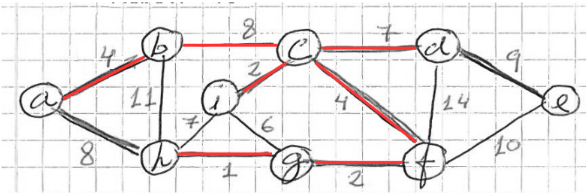
Select edge $\{a, b\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



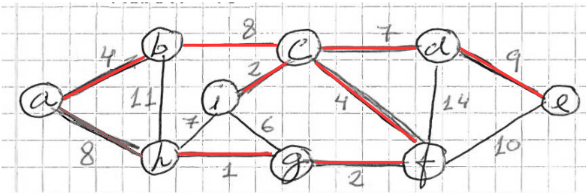
Select edge $\{c, d\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



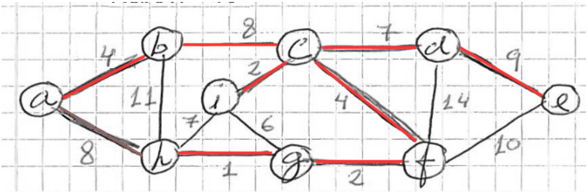
Select edge $\{b, c\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



Select edge $\{d, e\}$

Kruskal Algorithm to find a minimum spanning tree in $G = (E, V)$



Maintain each component as a set of edges.
Edges in the same set cannot be linked.

Kruskal Algorithm with Union-Find

Function *Kruskal*(G)

```
T = ∅
for each vertex v ∈ V do
    make_set(v)
sort edges in E into nondecreasing order by weight
for each edge {u, v} ∈ E  do
    if find_set(u) ≠ find_set(v)  then
        T = T ∪ {u, v}
        union(u, v)
return  T
```

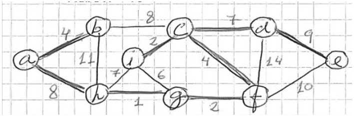
Union-Find for Kruskal Algorithm

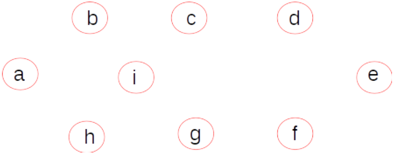
Kruskal Algorithm with Union-Find

- Operation make_set(v) creates a tree with v, which is also its root. In the subsequent operations, the tree id is the vertex root of the tree
- Operation find_set(v) checks the tree id of vertex v by traversing the tree up to the root; if two vertices belong to the same tree, they share the same tree id and cannot be directly connected
- Operation union(u,v) merges the roots of two trees to which u and v belong

Union-Find for Kruskal Algorithm

Kruskal Algorithm with Union-Find

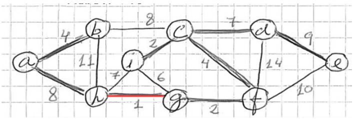


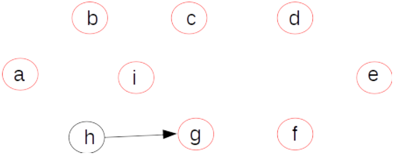


Operation make_set to all vertices

Union-Find for Kruskal Algorithm

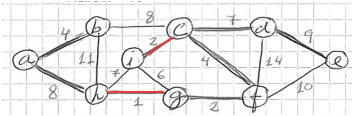
Kruskal Algorithm with Union-Find

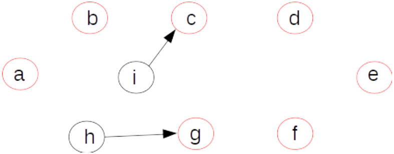




union(h, g)

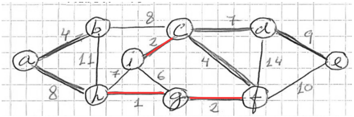
Kruskal Algorithm with Union-Find

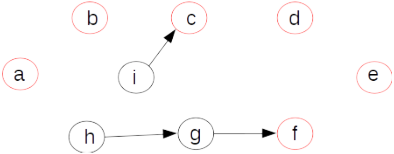




union(i, c)

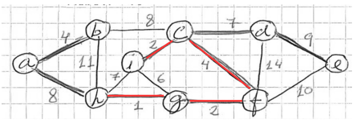
Kruskal Algorithm with Union-Find

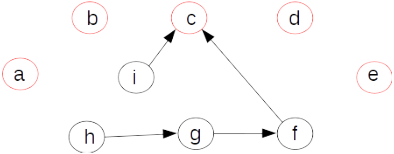




union(g, f)

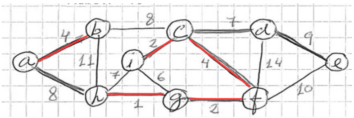
Kruskal Algorithm with Union-Find

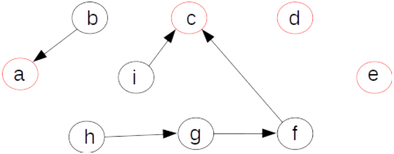




union(c, f)

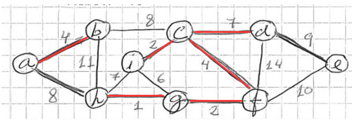
Kruskal Algorithm with Union-Find

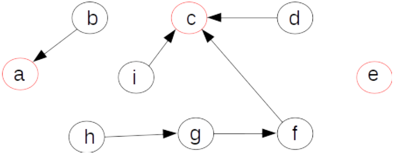




union(b, a)

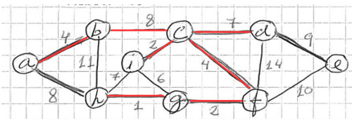
Kruskal Algorithm with Union-Find

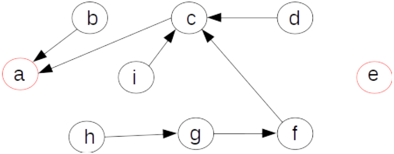




union(c, d)

Kruskal Algorithm with Union-Find

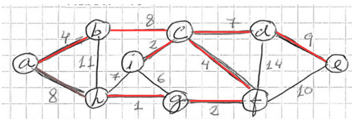


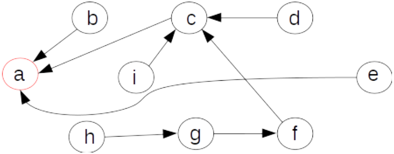


union(b, c)

Union-Find for Kruskal Algorithm

Kruskal Algorithm with Union-Find





union(e, d)

Kruskal Algorithm with Union-Find

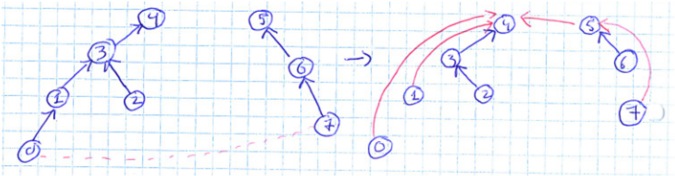
Kruskal Algorithm takes $O(|E| \cdot |V|)$:

- Sort $|E|$ edges in $O(|E| \log |V|)$
- Operation make_set on all vertices in $O(|V|)$
- Operation find_set for all edges in $O(|E| \cdot |V|)$
- Operation union for all edges in a spanning tree in $O(|V|)$

Operation find_set is the bottleneck. Can it be improved?

Improvements on Union-Find Algorithm

1. Connect the root of the smaller tree to the root of the larger tree, which shortner the time to reach the root.
2. *Path compression*: Connect all descendents to the root of the new tree. The height of the (uncompressed) tree is updated as in 1). Since this value does not correspond to the height of compressed tree, its name is *rank*.



Improvements on Union-Find Algorithm

1. Connect the root of the smaller tree to the root of the larger tree: Since the maximum tree height is $\log|V|$, find_set takes $O(|E|\log|V|)$ over all edges
2. Path compression: $|E|$ tree updates takes $O(|E|\log^*|V|)$

$$\log^* n = \begin{cases} 0 & \text{if } n \leq 1 \\ 1 + \log^*(\log n) & \text{otherwise} \end{cases}$$

After sorting, the time complexity for $|V| = 2^{2^{16}}$ is $|E| \cdot \log^*(2^{2^{16}})$ $\approx |E| \cdot 5$, that is, almost linear in the number of edges

Improvements on Union-Find Algorithm

make_set

for each vertex $i \in V$ **do**

  $set[i] = i$

  $rank[i] = 0$

union(a,b)

  $link(find(a), find(b))$

find(a)

if $set[a] \neq a$ **then**

  $set[a] = find(set[a])$

return $set[a]$

link(a,b)

if $rank[a] > rank[b]$ **then**

  $set[b] = a$

else

  $set[a] = b$

if $rank[a] = rank[b]$ **then**

  $rank[b] + +$