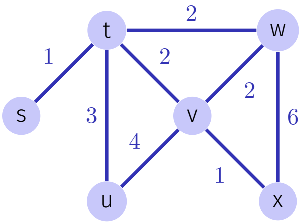
14. Minimum Spanning Trees

Motivation, Greedy, Algorithm Kruskal, General Rules, ADT Union-Find, Algorithm Jarnik, Prim, Dijkstra [Ottman/Widmayer, Kap. 9.6, 6.2, 6.1, Cormen et al, Kap. 23, 19]

Problem

Given: Undirected, weighted, connected graph $G = (V, E, c)$.

Wanted: Minimum Spanning Tree $T = (V, E')$: connected, cycle-free subgraph $E' \subset E$, such that $\sum_{e \in E'} c(e)$ minimal.



Application Examples

- Network-Design: find the cheapest / shortest network that connects all nodes.
- Approximation of a solution of the travelling salesman problem: find a round-trip, as short as possible, that visits each node once. [18]

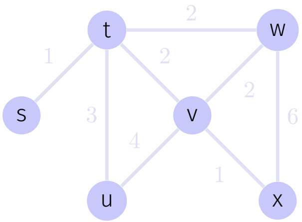
[18] The best known algorithm to solve the TS problem exactly has exponential running time.

Greedy Procedure

- Greedy algorithms compute the solution stepwise choosing locally optimal solutions.
- Most problems cannot be solved with a greedy algorithm.
- The Minimum Spanning Tree problem can be solved with a greedy strategy.

Greedy Idea (Kruskal, 1956)

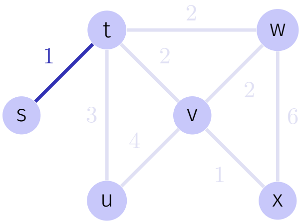
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea (Kruskal, 1956)

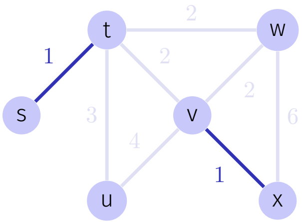
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea (Kruskal, 1956)

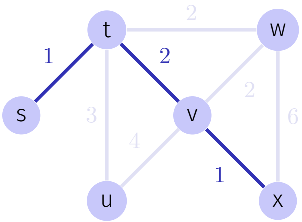
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea (Kruskal, 1956)

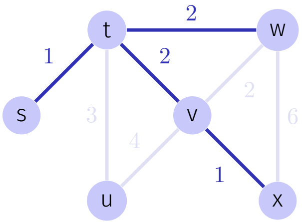
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea (Kruskal, 1956)

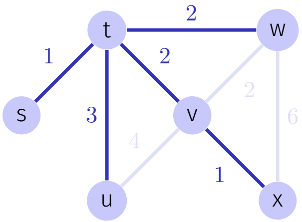
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea (Kruskal, 1956)

Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Algorithm MST-Kruskal(G)

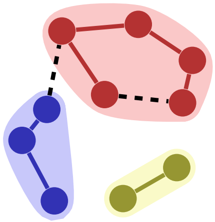
Input: Weighted Graph $G = (V, E, c)$
Output: Minimum spanning tree with edges A.

Sort edges by weight $c(e_1) \leq ... \leq c(e_m)$
$A \leftarrow \emptyset$
for $k = 1$ **to** $|E|$ **do**
    if $(V, A \cup \{e_k\})$ acyclic **then**
        $A \leftarrow A \cup \{e_k\}$

return (V, A, c)

Implementation Issues

Consider a set of sets $i \equiv A_i \subset V$. To identify cuts and cycles: membership of the both ends of an edge to sets?



Implementation Issues

General problem: partition (set of subsets) .e.g.
$\{\{1,2,3,9\},\{7,6,4\},\{5,8\},\{10\}\}$
Required: Abstract data type "Union-Find" with the following operations

- Make-Set(i): create a new set represented by i.
- Find(e): name of the set i that contains e .
- Union(i,j): union of the sets with names i and j.

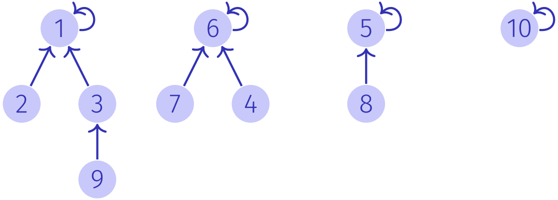
Union-Find Algorithm MST-Kruskal(G)

Input: Weighted Graph $G = (V, E, c)$
Output: Minimum spanning tree with edges A.

```
Sort edges by weight c(e_1) ≤ ... ≤ c(e_m)
A ← ∅
for k = 1 to |V| do
    MakeSet(k)
for k = 1 to m do
    (u, v) ← e_k
    if Find(u) ≠ Find(v) then
        Union(Find(u), Find(v))
        A ← A ∪ e_k
    else                                        // conceptual: R ← R ∪ e_k
return (V, A, c)
```

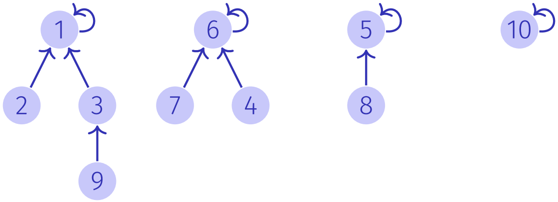
Implementation Union-Find

Idea: tree for each subset in the partition,e.g. $\{\{1, 2, 3, 9\}, \{7, 6, 4\}, \{5, 8\}, \{10\}\}$



roots = names (representatives) of the sets,
trees = elements of the sets

Implementation Union-Find



Representation as array:

Index	1	2	3	4	5	6	7	8	9	10
Parent	1	1	1	6	5	6	5	5	3	10

Implementation Union-Find

Index	1	2	3	4	5	6	7	8	9	10
Parent	1	1	1	6	5	6	5	5	3	10

Make-Set(i)	$p[i] \leftarrow i$; **return** i
Find(i)	**while** $(p[i] \neq i)$ **do** $i \leftarrow p[i]$ **return** i
Union(i, j) [19]	$p[j] \leftarrow i$;

[19] i and j need to be names (roots) of the sets. Otherwise use Union(Find(i),Find(j))

Optimisation of the runtime for Find

Tree may degenerate. Example: Union$(8, 7)$, Union$(7, 6)$, Union$(6, 5)$, ...

Index	1	2	3	4	5	6	7	8	..
Parent	1	1	2	3	4	5	6	7	..

Worst-case running time of Find in $\Theta(n)$.

Optimisation of the runtime for Find

Idea: always append smaller tree to larger tree. Requires additional size information (array) g

Make-Set(i)	$p[i] \leftarrow i$; $g[i] \leftarrow 1$; **return** i
Union(i, j)	**if** $g[j] > g[i]$ **then** swap(i, j) $p[j] \leftarrow i$ **if** $g[i] = g[j]$ **then** $g[i] \leftarrow g[i] + 1$

$\Rightarrow$ Tree depth (and worst-case running time for Find) in $\Theta(\log n)$

Further improvement

Link all nodes to the root when Find is called.

Find(i):

$j \leftarrow i$
while $(p[i] \neq i)$ **do** $i \leftarrow p[i]$
while $(j \neq i)$ **do**
    $t \leftarrow j$
    $j \leftarrow p[j]$
    $p[t] \leftarrow i$
return i

Cost: amortised *nearly* constant (inverse of the Ackermann-function).[20]

[20] We do not go into details here.

Running time of Kruskal's Algorithm

- Sorting of the edges: $\Theta(|E|\log|E|) = \Theta(|E|\log|V|)$. [21]
- Initialisation of the Union-Find data structure $\Theta(|V|)$
- $|E|\times$ Union(Find(x),Find(y)): $\mathcal{O}(|E|\log|E|) = \mathcal{O}(|E|\log|V|)$.

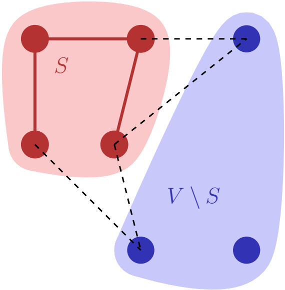
Overal $\Theta(|E|\log|V|)$.

[21] because G is connected: $|V| \leq |E| \leq |V|^2$

Algorithm of Jarnik (1930), Prim, Dijkstra (1959)

Idea: start with some $v \in V$ and grow the spanning tree from here by the acceptance rule.

$A \leftarrow \emptyset$
$S \leftarrow \{v_0\}$
for $i \leftarrow 1$ **to** $|V|$ **do**
  Choose cheapest (u, v) mit $u \in S$, $v \notin S$
  $A \leftarrow A \cup \{(u, v)\}$
  $S \leftarrow S \cup \{v\}$ // (Coloring)



Remark: a union-Find data structure is not required. It suffices to color nodes when they are added to S.

Running time

Trivially $\mathcal{O}(|V| \cdot |E|)$.

Improvement (like with Dijkstra's ShortestPath)

- With Min-Heap: costs
  - Initialization (node coloring) $\mathcal{O}(|V|)$
  - $|V|\times$ ExtractMin = $\mathcal{O}(|V| \log |V|)$,
  - $|E|\times$ Insert or DecreaseKey: $\mathcal{O}(|E| \log |V|)$,

  $\mathcal{O}(|E| \cdot \log |V|)$