25. Minimum Spanning Trees

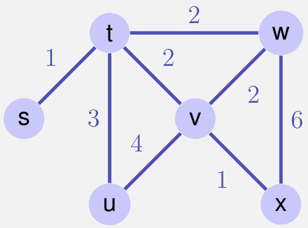
Motivation, Greedy, Algorithm Kruskal, General Rules, ADT Union-Find, Algorithm Jarnik, Prim, Dijkstra, Fibonacci Heaps

[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 subgraph $E' \subset E$, such that $\sum_{e \in E'} c(e)$ minimal.



Application: cheapest / shortest cable network

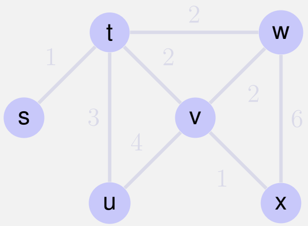
Greedy Procedure

Recall:

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

Greedy Idea

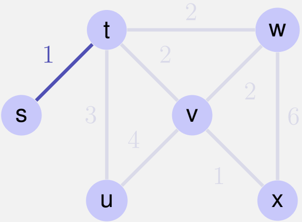
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea

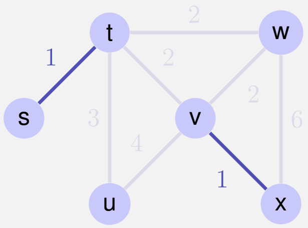
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea

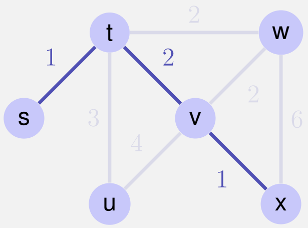
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea

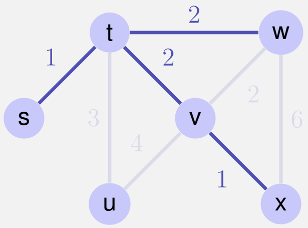
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea

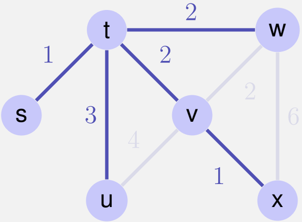
Construct T by adding the cheapest edge that does not generate a cycle.



(Solution is not unique.)

Greedy Idea

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)

Correctness

At each point in the algorithm (V, A) is a forest, a set of trees.

MST-Kruskal considers each edge e_k exactly once and either chooses or rejects e_k

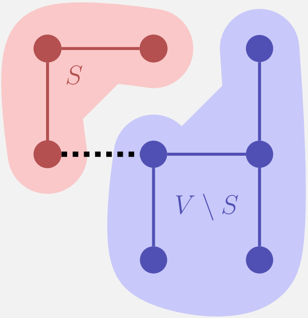
Notation (snapshot of the state in the running algorithm)

- A: Set of selected edges
- R: Set of rejected edges
- U: Set of yet undecided edges

Cut

A cut of G is a partition $S, V - S$ of V. ($S \subseteq V$).

An edge crosses a cut when one of its endpoints is in S and the other is in $V \setminus S$.



Rules

1. Selection rule: choose a cut that is not crossed by a selected edge. Of all undecided edges that cross the cut, select the one with minimal weight.
2. Rejection rule: choose a circle without rejected edges. Of all undecided edges of the circle, reject those with minimal weight.

Rules

Kruskal applies both rules:

1. A selected e_k connects two connection components, otherwise it would generate a circle. e_k is minimal, i.e. a cut can be chosen such that e_k crosses and e_k has minimal weight.
2. A rejected e_k is contained in a circle. Within the circle e_k has minimal weight.

Correctness

Theorem

Every algorithm that applies the rules above in a step-wise manner until $U = \emptyset$ is correct.

Consequence: MST-Kruskal is correct.

Selection invariant

Invariant: At each step there is a minimal spanning tree that contains all selected and none of the rejected edges.

If both rules satisfy the invariant, then the algorithm is correct. Induction:

- At beginning: $U = E$, $R = A = \emptyset$. Invariant obviously holds.
- Invariant is preserved.
- At the end: $U = \emptyset$, $R \cup A = E \Rightarrow (V, A)$ is a spanning tree.

Proof of the theorem: show that both rules preserve the invariant.

Selection rule preserves the invariant

At each step there is a minimal spanning tree T that contains all selected and none of the rejected edges.

Choose a cut that is not crossed by a selected edge. Of all undecided edges that cross the cut, select the egde e with minimal weight.

- Case 1: $e \in T$ (done)
- Case 2: $e \notin T$. Then $T \cup \{e\}$ contains a circle that contains e Circle must have a second edge e' that also crosses the cut.[38] Because $e' \notin R$, $e' \in U$. Thus $c(e) \leq c(e')$ and $T' = T \setminus \{e'\} \cup \{e\}$ is also a minimal spanning tree (and $c(e) = c(e')$).

[38] Such a circle contains at least one node in S and one node in $V \setminus S$ and therefore at lease to edges between S and $V \setminus S$.

Rejection rule preserves the invariant

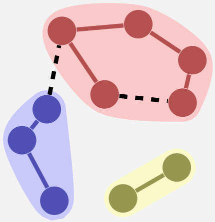
At each step there is a minimal spanning tree T that contains all selected and none of the rejected edges.

Choose a circle without rejected edges. Of all undecided edges of the circle, reject an edge e with minimal weight.

- Case 1: $e \notin T$ (done)
- Case 2: $e \in T$. Remove e from T, This yields a cut. This cut must be crossed by another edge e' of the circle. Because $c(e') \leq c(e)$, $T' = T \setminus \{e\} \cup \{e'\}$ is also minimal (and $c(e) = c(e')$).

Implementation Issues

Consider a set of sets $i \equiv A_i \subset V$. To identify cuts and circles: 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: ADT (Union-Find-Structure) 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.

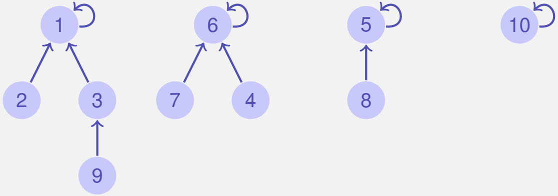
Sort edges by weight $c(e_1) \leq ... \leq c(e_m)$
$A \leftarrow \emptyset$
for $k = 1$ **to** $|V|$ **do**
    MakeSet(k)

for $k = 1$ **to** $|E|$ **do**
    $(u, v) \leftarrow e_k$
    if Find(u) $\neq$ Find(v) **then**
        Union(Find(u), Find(v))
        $A \leftarrow A \cup e_k$

return (V, A, c)

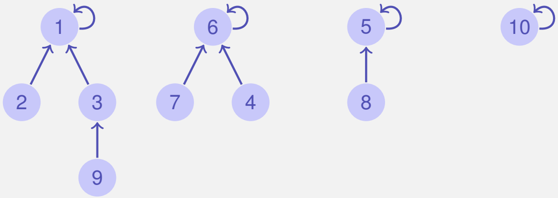
Implementation Union-Find

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



roots = names 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**

Operations:

- 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): [39] $p[j] \leftarrow i$; **return** i

[39] i and j need to be names (roots) of the sets. typically: Union(Find(a),Find(b))

Optimisation of the runtime for Find

Tree may degenerate. Example: Union$(1, 2)$, Union$(2, 3)$, Union$(3, 4)$, ...

Idea: always append smaller tree to larger tree. Additionally required: size information g

Operations:

- 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)
  $p[j] \leftarrow i$
  $g[i] \leftarrow g[i] + g[j]$
  return i

Observation

Theorem

The method above (union by size) preserves the following property of the trees: a tree of height h has at least 2^h nodes.

Immediate consequence: runtime Find = $\mathcal{O}(\log n)$.

Proof

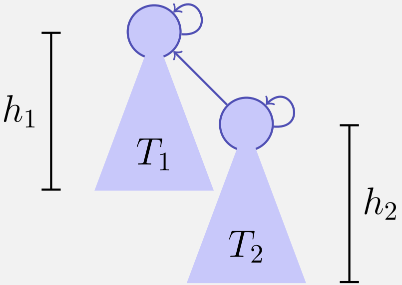
Induction: by assumption, sub-trees have at least 2^{h_i} nodes. WLOG: $h_2 \leq h_1$

- $h_2 < h_1$:

$$h(T_1 \oplus T_2) = h_1 \Rightarrow g(T_1 \oplus T_2) \geq 2^h$$

- $h_2 = h_1$:

$$g(T_1) \geq g(T_2) \geq 2^{h_2}$$
$$\Rightarrow g(T_1 \oplus T_2) = g(T_1) + g(T_2) \geq 2 \cdot 2^{h_2} = 2^{h(T_1 \oplus T_2)}$$



Further improvement

Link all nodes to the root when Find is called.

Find(i):

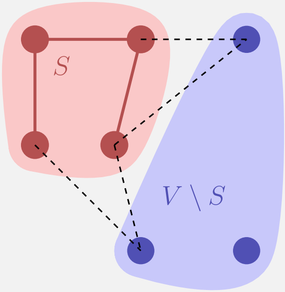
$j \leftarrow i$
while $(p[i] \neq i)$ **do** $i \leftarrow p[i]$
while $(j \neq i)$ **do**
    $t \leftarrow j$
    $j \leftarrow p[j]$
    $p[t] \leftarrow i$
return i

Amortised cost: amortised *nearly* constant (inverse of the Ackermann-function).

MST algorithm of Jarnik, Prim, Dijkstra

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

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



Running time

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

Improvements (like with Dijkstra's ShortestPath)

- Memorize cheapest edge to S: for each $v \in V \setminus S$. $\deg^+(v)$ many updates for each new $v \in S$. Costs: $|V|$ many minima and updates: $\mathcal{O}(|V|^2 + \sum_{v \in V} \deg^+(v)) = \mathcal{O}(|V|^2 + |E|)$
- With Minheap: costs $|V|$ many minima = $\mathcal{O}(|V| \log |V|)$, $|E|$ Updates: $\mathcal{O}(|E| \log |V|)$, Initialization $\mathcal{O}(|V|)$: $\mathcal{O}(|E| \cdot \log |V|.)$
- With a Fibonacci-Heap: $\mathcal{O}(|E| + |V| \cdot \log |V|)$.

Fibonacci Heaps

Data structure for elements with key with operations

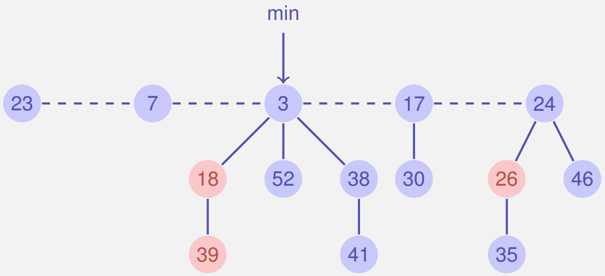
- MakeHeap(): Return new heap without elements
- Insert(H, x): Add x to H
- Minimum(H): return a pointer to element m with minimal key
- ExtractMin(H): return and remove (from H) pointer to the element m
- Union(H_1, H_2): return a heap merged from H_1 and H_2
- DecreaseKey(H, x, k): decrease the key of x in H to k
- Delete (H, x): remove element x from H

Advantage over binary heap?

	Binary Heap (worst-Case)	Fibonacci Heap (amortized)
MakeHeap	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(\log n)$	$\Theta(1)$
Minimum	$\Theta(1)$	$\Theta(1)$
ExtractMin	$\Theta(\log n)$	$\Theta(\log n)$
Union	$\Theta(n)$	$\Theta(1)$
DecreaseKey	$\Theta(\log n)$	$\Theta(1)$
Delete	$\Theta(\log n)$	$\Theta(\log n)$

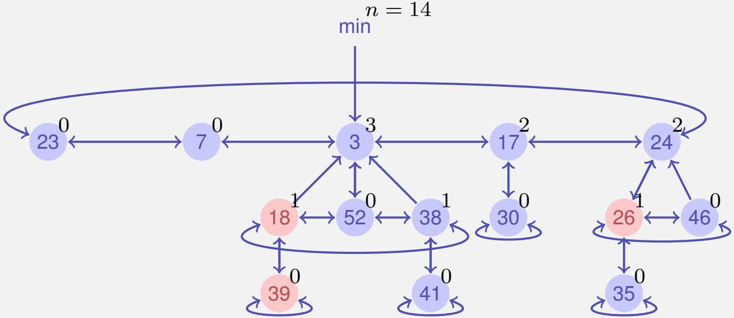
Structure

Set of trees that respect the Min-Heap property. Nodes that can be marked.



Implementation

Doubly linked lists of nodes with a marked-flag and number of children. Pointer to minimal Element and number nodes.



Simple Operations

- MakeHeap (trivial)
- Minimum (trivial)
- Insert(H, e)
  1. Insert new element into root-list
  2. If key is smaller than minimum, reset min-pointer.
- Union (H_1, H_2)
  1. Concatenate root-lists of H_1 and H_2
  2. Reset min-pointer.
- Delete(H, e)
  1. DecreaseKey($H, e, -\infty$)
  2. ExtractMin(H)

ExtractMin

1. Remove minimal node m from the root list
2. Insert children of m into the root list
3. Merge heap-ordered trees with the same degrees until all trees have a different degree:
   Array of degrees $a[0, \ldots, n]$ of elements, empty at beginning.
   For each element e of the root list:
   a. Let g be the degree of e
   b. If $a[g] = nil$: $a[g] \leftarrow e$.
   c. If $e' := a[g] \neq nil$: Merge e with e' resutling in e'' and set $a[g] \leftarrow nil$. Set e'' unmarked. Re-iterate with $e \leftarrow e''$ having degree $g + 1$.

DecreaseKey (H, e, k)

1. Remove e from its parent node p (if existing) and decrease the degree of p by one.
2. Insert(H, e)
3. Avoid too thin trees:
   a. If $p = nil$ then done.
   b. If p is unmarked: mark p and done.
   c. If p marked: unmark p and cut p from its parent pp. Insert (H, p). Iterate with $p \leftarrow pp$.

Estimation of the degree

Theorem

Let p be a node of a F-Heap H. If child nodes of p are sorted by time of insertion (Union), then it holds that the ith child node has a degree of at least $i - 2$.

Proof: p may have had more children and lost by cutting. When the ith child p_i was linked, p and p_i must at least have had degree $i - 1$. p_i may have lost at least one child (marking!), thus at least degree $i - 2$ remains.

Estimation of the degree

Theorem

Every node p with degree k of a F-Heap is the root of a subtree with at least F_{k+1} nodes. (F: Fibonacci-Folge)

Proof: Let S_k be the minimal number of successors of a node of degree k in a F-Heap plus 1 (the node itself). Clearly $S_0 = 1$, $S_1 = 2$. With the previous theorem $S_k \geq 2 + \sum_{i=0}^{k-2} S_i, k \geq 2$ (p and nodes p_1 each 1). For Fibonacci numbers it holds that (induction) $F_k \geq 2 + \sum_{i=2}^{k} F_i, k \geq 2$ and thus (also induction) $S_k \geq F_{k+2}$.

Fibonacci numbers grow exponentially fast ($\mathcal{O}(\varphi^k)$) Consequence: maximal degree of an arbitrary node in a Fibonacci-Heap with n nodes is $\mathcal{O}(\log n)$.

Amortized worst-case analysis Fibonacci Heap

$t(H)$: number of trees in the root list of H, $m(H)$: number of marked nodes in H not within the root-list, Potential function $\Phi(H) = t(H) + 2 \cdot m(H)$. At the beginnning $\Phi(H) = 0$. Potential always non-negative.

Amortized costs:

- Insert(H, x): $t'(H) = t(H) + 1$, $m'(H) = m(H)$, Increase of the potential: 1, Amortized costs $\Theta(1) + 1 = \Theta(1)$
- Minimum(H): Amortized costs = real costs = $\Theta(1)$
- Union(H_1, H_2): Amortized costs = real costs = $\Theta(1)$

Amortized costs of ExtractMin

- Number trees in the root list $t(H)$.
- Real costs of ExtractMin operation $\mathcal{O}(\log n + t(H))$.
- When merged still $\mathcal{O}(\log n)$ nodes.
- Number of markings can only get smaller when trees are merged
- Thus maximal amortized costs of ExtractMin

$$\mathcal{O}(\log n + t(H)) + \mathcal{O}(\log n) - \mathcal{O}(t(H)) = \mathcal{O}(\log n).$$

Amortized costs of DecreaseKey

- Assumption: DecreaseKey leads to c cuts of a node from its parent node, real costs $\mathcal{O}(c)$
- c nodes are added to the root list
- Delete $(c-1)$ mark flags, addition of at most one mark flag
- Amortized costs of DecreaseKey:

$$\mathcal{O}(c) + (t(H)+c) + 2\cdot(m(H)-c+2)) - (t(H)+2m(H)) = \mathcal{O}(1)$$