CS 341: Algorithms

Lecture 13: Minimum spanning trees

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Fall 2024

Spanning trees

Input and output:

- $G = (V, E)$ is a **weighted, connected undirected graph**
- edges have **weights** $w(e_i)$
- a **spanning tree** is a tree with edges from E that covers all vertices
- examples: BFS tree, DFS tree

Remark: will assume $w(e_i)$ distinct, using $W(e_i) = [w(e_i), i]$ to break ties if needed

Goal:

- a spanning tree with **minimal weight**
- notation: $w(T) = \sum_{e \text{ edge in } T} w(e)$
- all weights fit in a word, as usual

Exercise

what about maximal weight spanning trees?

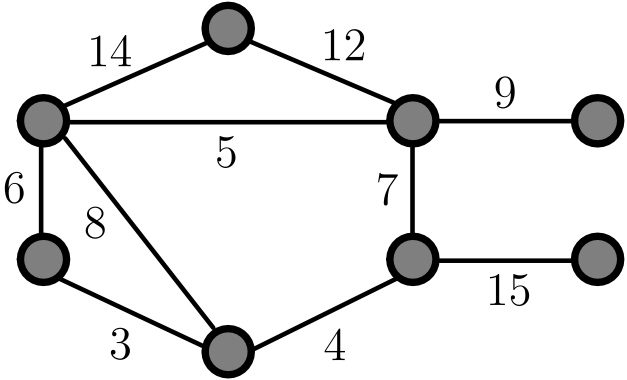
Kruskal's algorithm

Kruskal's algorithm

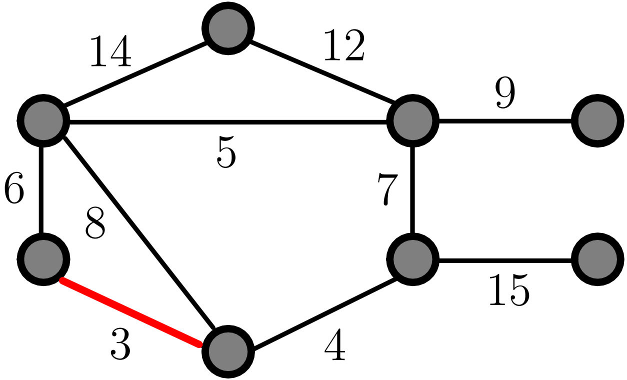
GreedyMST(G)

1. $F \leftarrow [\ ]$
2. sort edges by non-decreasing weight
3. **for** $k = 1, \ldots, m$ **do**
4.   **if** e_k does not create a cycle in (V, F) **then**
5.   append e_k to F
6. **return** $A = (V, F)$

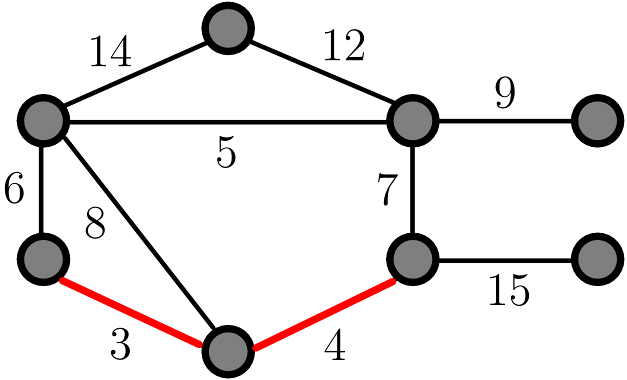
Example



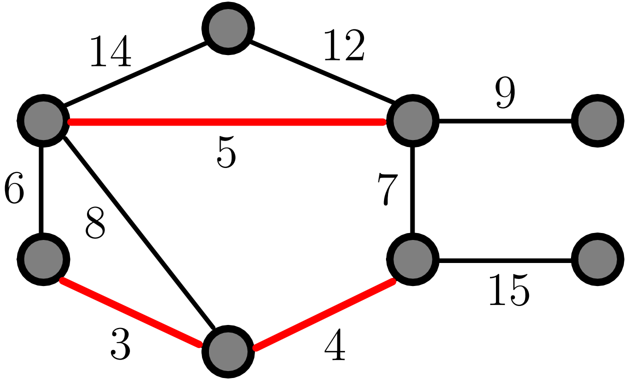
Example



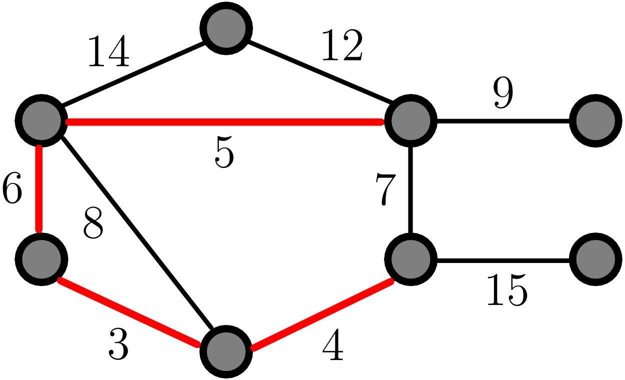
Example



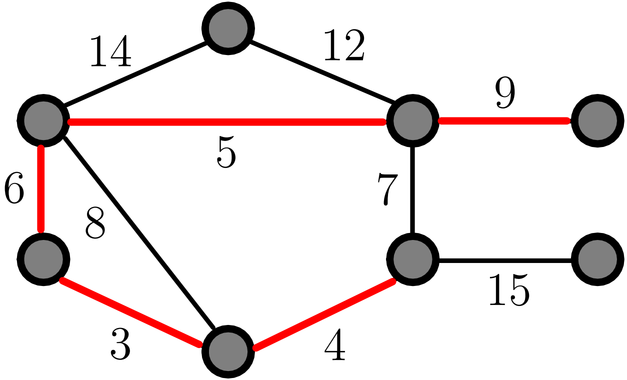
Example



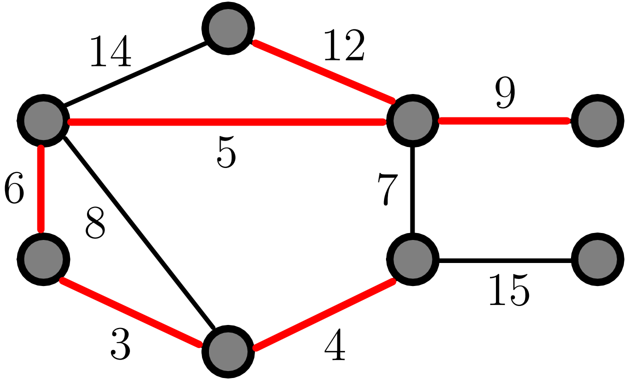
Example



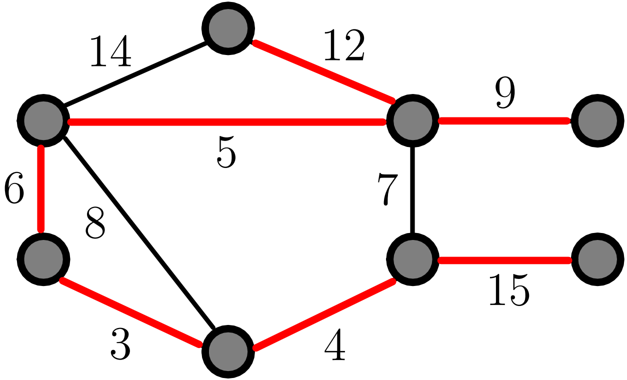
Example



Example



Example



Properties of the output

Claim

The output $\boldsymbol{A = (V, F)}$ **is a spanning tree**

Proof:

- of course, A has no cycle: it is a **forest**
- suppose A is **not connected**. Then, there exists an edge e not in F, such that $(V, F \cup \{e\})$ still has no cycle (join two connected components)
- when we checked e, we did not include it
- that's because that it created a cycle with some edges already in F: **impossible**.

The cut property

Definition

cut: a partition of the vertices into sets $\boldsymbol{S}$ and $\boldsymbol{V} - \boldsymbol{S}$
cutset: the edges between $\boldsymbol{S}$ and $\boldsymbol{V} - \boldsymbol{S}$

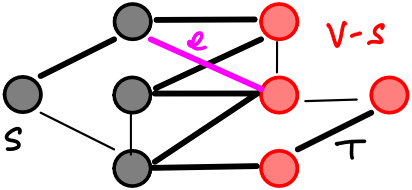
Claim

For **any** cut, the minimal weight edge in the cutset is in **any** minimum spanning tree.

Proof

For any cut, the minimal weight edge e in the cutset is in any minimum spanning tree.

- let T be a minimum spanning tree **that does not contain e**
- adding e to T creates a cycle C, and there must be an edge $e' \neq e$ in C connecting S and $V - S$



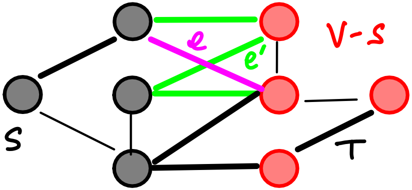
consider $\boldsymbol{T' = T - \{e'\} \cup \{e\}}$

- $w(T') < w(T)$
- but T' is still a spanning tree
  - $n - 1$ edges
  - connected: can replace edge e' by $C - \{e'\}$ to connect vertices
- contradiction

Proof

For any cut, the minimal weight edge e in the cutset is in any minimum spanning tree.

- let T be a minimum spanning tree **that does not contain e**
- adding e to T creates a cycle C, and there must be an edge $e' \neq e$ in C connecting S and $V - S$



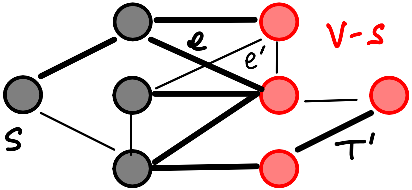
consider $\boldsymbol{T' = T - \{e'\} \cup \{e\}}$

- $w(T') < w(T)$
- but T' is still a spanning tree
  - $n - 1$ edges
  - connected: can replace edge e' by $C - \{e'\}$ to connect vertices
- contradiction

Proof

For any cut, the minimal weight edge e in the cutset is in any minimum spanning tree.

- let T be a minimum spanning tree **that does not contain e**
- adding e to T creates a cycle C, and there must be an edge $e' \neq e$ in C connecting S and $V - S$



consider $\boldsymbol{T' = T - \{e'\} \cup \{e\}}$

- $w(T') < w(T)$
- but T' is still a spanning tree
  - $n - 1$ edges
  - connected: can replace edge e' by $C - \{e'\}$ to connect vertices
- contradiction

Kruskal is optimal

Claim: every edge we add to the output is in every minimal spanning tree

Proof: consider $A = (V, F)$ the forest just before inserting $e = \{v, w\}$, let S be the vertices in the tree containing v

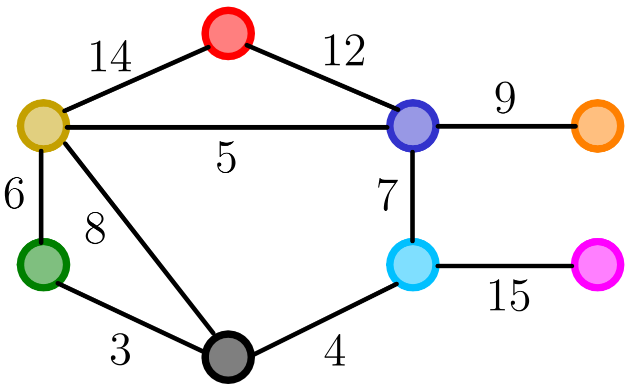
fact 1: w is in $V - S$ (otherwise, cycle)
fact 2: the other edges in the cutset have not been considered yet
(they do not create cycles, so they would have been put in F)

so e is has minimal weight in the cutset, and it is in every minimal spanning tree

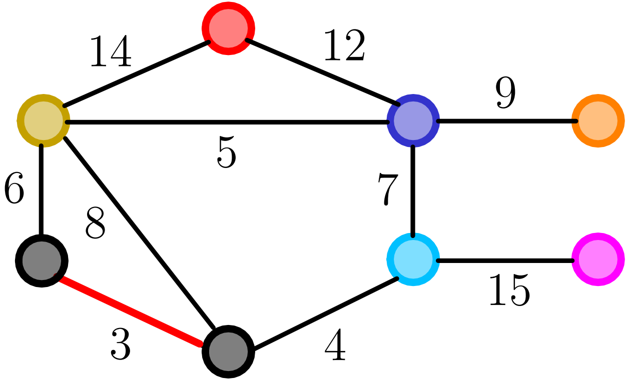
Remark 1: this proves that the minimum spanning tree is unique

Remark 2: proof by exchange argument doable as well

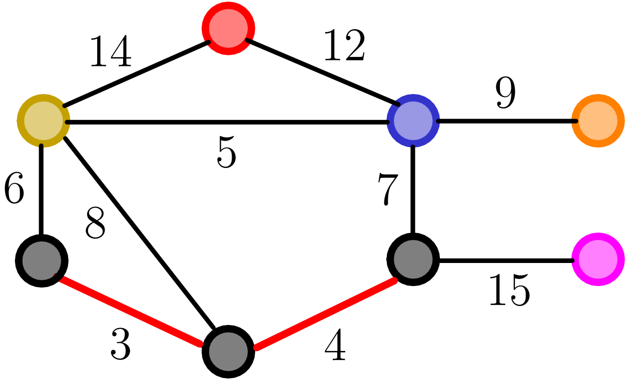
Merging connected sets of vertices



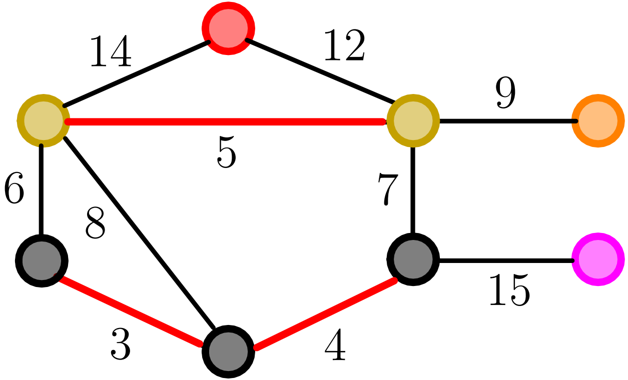
Merging connected sets of vertices



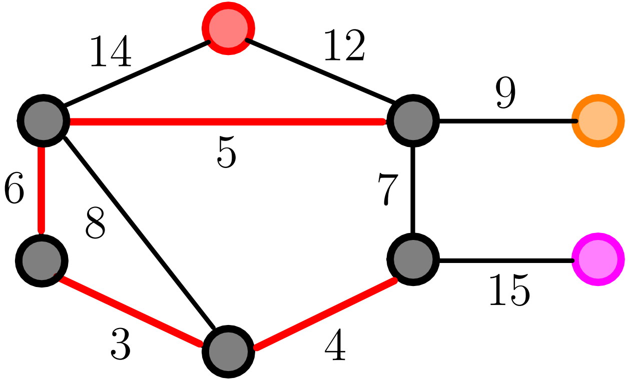
Merging connected sets of vertices



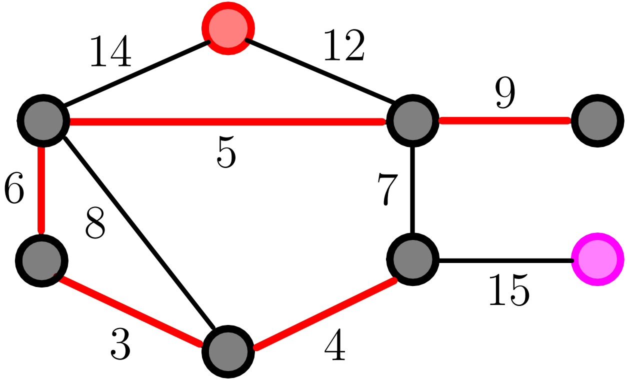
Merging connected sets of vertices



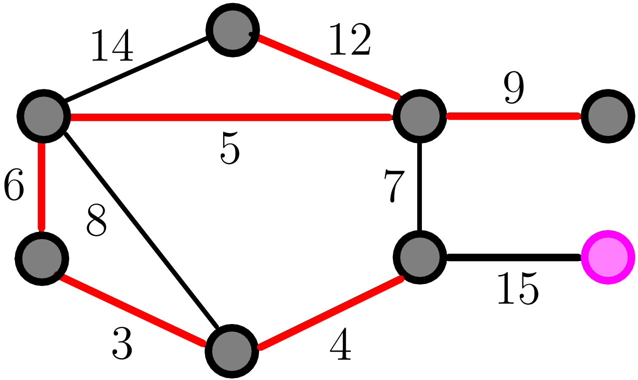
Merging connected sets of vertices



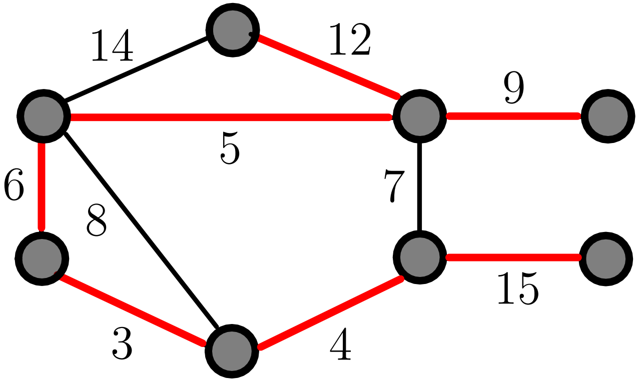
Merging connected sets of vertices



Merging connected sets of vertices



Merging connected sets of vertices



Data structures

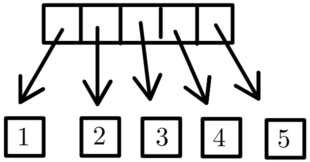
Operations on **disjoint sets of vertices:**

- **Find:** identify which set contains a given vertex
- **Union:** replace two sets by their union

GreedyMST_UnionFind(G)
1. $T \leftarrow [\,]$
2. $S \leftarrow \{\{v_1\}, \ldots, \{v_n\}\}$
3. sort edges by non-decreasing weight
4. **for** $k = 1, \ldots, m$ **do**
5. **if find**$(S, e_k.1) \neq$ **find**$(S, e_k.2)$ **then**
6. **union**$(S,$ **find**$(S, e_k.1),$ **find**$(S, e_k.2))$
7. append e_k to T

An OK solution

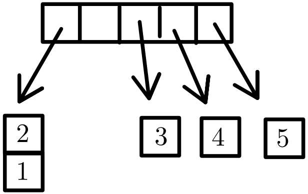
a data structure for union: an array U of **linked lists**



union_v1(U, s, t)
1. **while** $U[s]$ not **NULL do**
2. $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$
3. $U[s] \leftarrow U[s]$.next

An OK solution

a data structure for union: an array U of **linked lists**

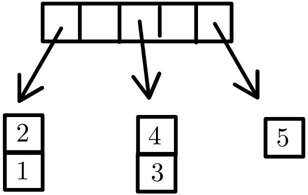


union_v1(U, s, t)

1. **while** $U[s]$ not **NULL do**
2. $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$
3. $U[s] \leftarrow U[s]$.next

An OK solution

a data structure for union: an array U of **linked lists**

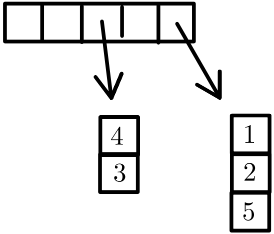


union_v1(U, s, t)

1. **while** $U[s]$ not **NULL do**
2. $U[t] \leftarrow$ new list$(U[s].\text{value}, U[t])$
3. $U[s] \leftarrow U[s].\text{next}$

An OK solution

a data structure for union: an array U of **linked lists**



union_v1(U, s, t)

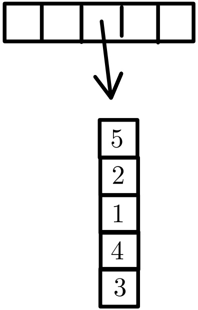
while $U[s]$ not **NULL do**

$\quad U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$

$\quad U[s] \leftarrow U[s]$.next

An OK solution

a data structure for union: an array U of **linked lists**



union_v1(U, s, t)

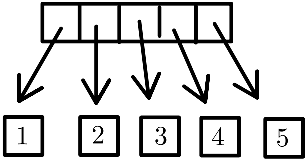
while $U[s]$ not **NULL do**

  $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$

  $U[s] \leftarrow U[s]$.next

An OK solution

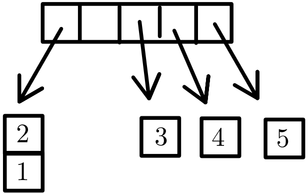
for find, use an **array of indices**, $X[i]$ = index of the set that contains i (find returns $X[i]$)



$$X = [1, 2, 3, 4, 5]$$

An OK solution

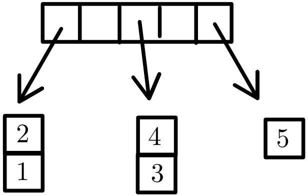
for find, use an **array of indices**, $X[i]$ = index of the set that contains i (find returns $X[i]$)



$$X = [1, 1, 3, 4, 5]$$

An OK solution

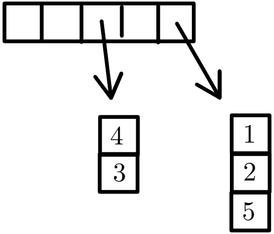
for find, use an **array of indices**, $X[i]$ = index of the set that contains i (find returns $X[i]$)



$$X = [1, 1, 3, 3, 5]$$

An OK solution

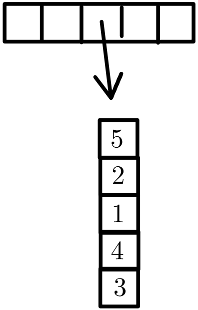
for find, use an **array of indices**, $X[i]$ = index of the set that contains i (find returns $X[i]$)



$$X = [5, 5, 3, 3, 5]$$

An OK solution

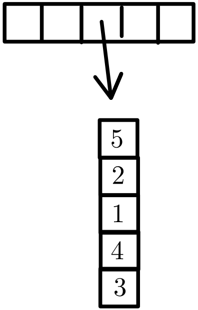
for find, use an **array of indices**, $X[i]$ = index of the set that contains i (find returns $X[i]$)



$$X = [3, 3, 3, 3, 3]$$

An OK solution

for find, use an **array of indices**, $X[i]$ = index of the set that contains i (find returns $X[i]$)



union_v2(X, U, s, t)

while $U[s]$ not **NULL do**

  $U[t] \leftarrow$ new list$(U[s]$.value$, U[t])$

  $X[U[s]$.value$] \leftarrow t$

  $U[s] \leftarrow U[s]$.next

Analysis

Worst case:

- **Find** is $O(1)$
- **Union** **traverses** one of the linked lists and **updates** corresponding entries of X.
  Worst case $\Theta(n)$

Analysis

Worst case:

- **Find** is $O(1)$
- **Union** **traverses** one of the linked lists and **updates** corresponding entries of X. Worst case $\Theta(n)$

Kruskal's algorithm:

- sorting edges $\boldsymbol{O(m\log(m))}$
- $O(m)$ **Find**
- $O(n)$ **Union**

Worst case $\boldsymbol{O(m\log(m)+n^2)}$

A simple heuristics for Union

Modified Union

- each list in U keeps track of its size
- merge the **smaller list** into the **larger list**

A simple heuristics for Union

Modified Union

- each list in U keeps track of its size
- merge the **smaller list** into the **larger list**

Key observation: worst case for **one** union **still** $\Theta(n)$, but the amortized cost is better.

- for any vertex v, the size of the set containing v **at least doubles** when we update $X[v]$
- so $X[v]$ updated at most $\log(n)$ times
- so the **total** cost of union **per vertex** is $O(\log(n))$

A simple heuristics for Union

Modified Union

- each list in U keeps track of its size
- merge the **smaller list** into the **larger list**

Key observation: worst case for **one** union **still** $\Theta(n)$, but the amortized cost is better.

- for any vertex v, the size of the set containing v **at least doubles** when we update $X[v]$
- so $X[v]$ updated at most $\log(n)$ times
- so the **total** cost of union **per vertex** is $O(\log(n))$

Conclusion: $\boldsymbol{O(n\log(n))}$ for all unions and $\boldsymbol{O(m\log(m))}$ total

Prim's algorithm

The idea

Goal

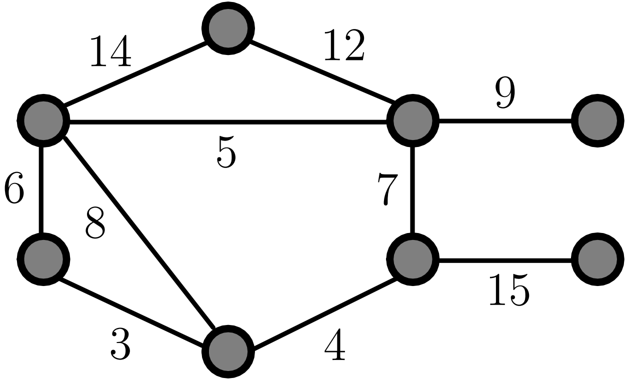
- G is an undirected graph
- $w : E \to R$ a weight function
- as before, want a **minimum weight spanning tree**

The idea:

- start from an **arbitrary source**
- **grow a tree** (connected, no cycle) edge-by-edge
- new edges chosen in a greedy manner

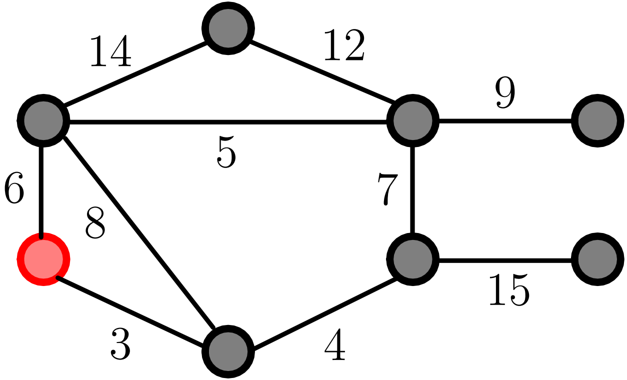
Growing a tree

we grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



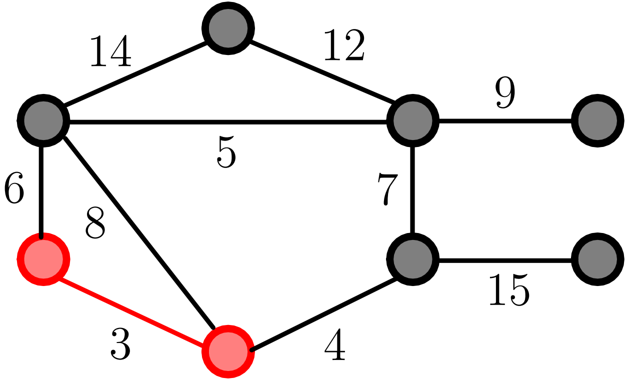
Growing a tree

we grow $\boldsymbol{A = (S, F)}$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow \boldsymbol{(V - S)}$



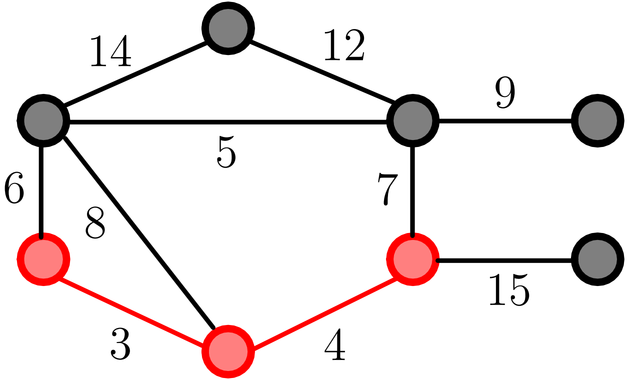
Growing a tree

we grow $\boldsymbol{A} = (\boldsymbol{S}, \boldsymbol{F})$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow (\boldsymbol{V} - \boldsymbol{S})$



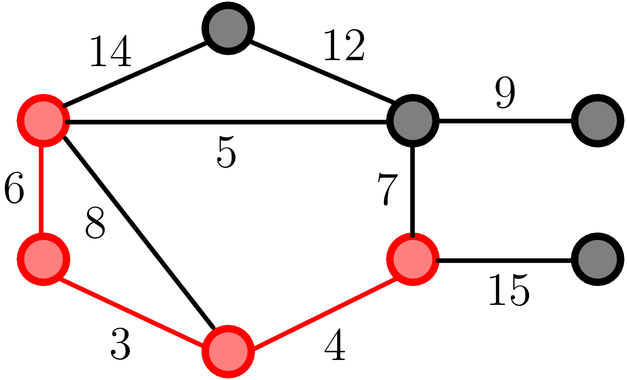
Growing a tree

we grow $\boldsymbol{A = (S, F)}$ by adding the **minimal weight edge** $\boldsymbol{S} \leftrightarrow \boldsymbol{(V - S)}$



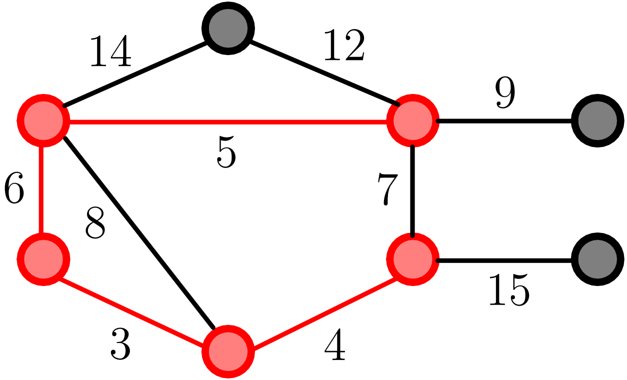
Growing a tree

we grow $A = (S, F)$ by adding the **minimal weight edge** $S \leftrightarrow (V - S)$



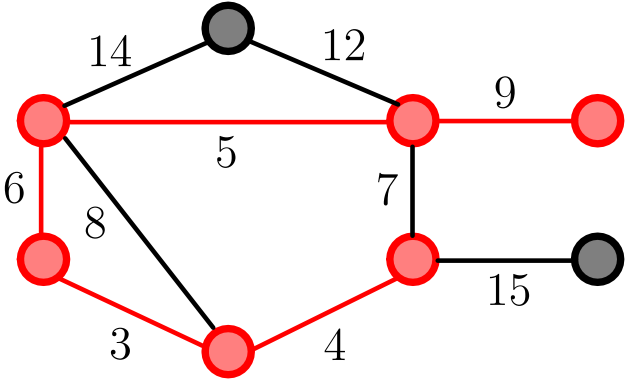
Growing a tree

we grow $A = (S, F)$ by adding the **minimal weight edge** $S \leftrightarrow (V - S)$



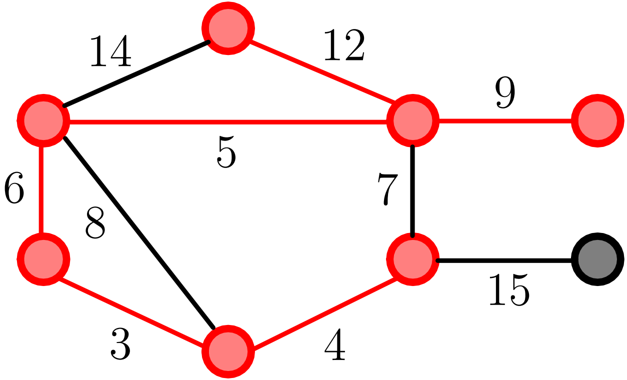
Growing a tree

we grow $A = (S, F)$ by adding the **minimal weight edge** $S \leftrightarrow (V - S)$



Growing a tree

we grow $A = (S, F)$ by adding the **minimal weight edge** $S \leftrightarrow (V - S)$



Growing a tree

we grow $A = (S, F)$ by adding the **minimal weight edge** $S \leftrightarrow (V - S)$