# 2IL50 Data Structures

2022-23 Q3

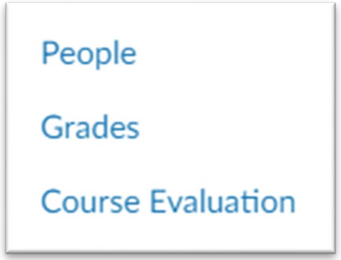
Lecture 12: Minimum Spanning Trees

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

# Announcements

| | |
|---|---|
| Practice exams | four previous exams with solutions available in Ans |
| Wednesday April 3 | recap lecture |
| Interim Test 4 | grades will be posted on Wednesday April 3<br>discussions until 12:00 on Friday April 5<br>*interim grades finalized by evening* |
| Exam | Ans + Safe Exam Browser<br>Do the SEB check!<br>*We need to know* ***now*** *if you need a loan laptop* |

# Course evaluation surveys are open



People

Grades

Course Evaluation

portal.evalytics.nl

evalytics

Be specific and focused. Use examples and **suggestions** to avoid vague statements.

Always be **respectful** when giving feedback.

Give positive feedback as well as areas for improvement. Stay **solution-oriented**.

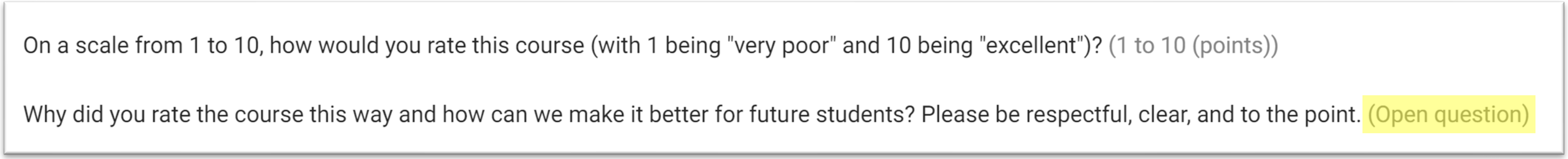
When giving feedback, focus on the **1 or 2** most important points that apply to you.

## What happens with the results?

**We read the anonymous results** and make changes for next year.

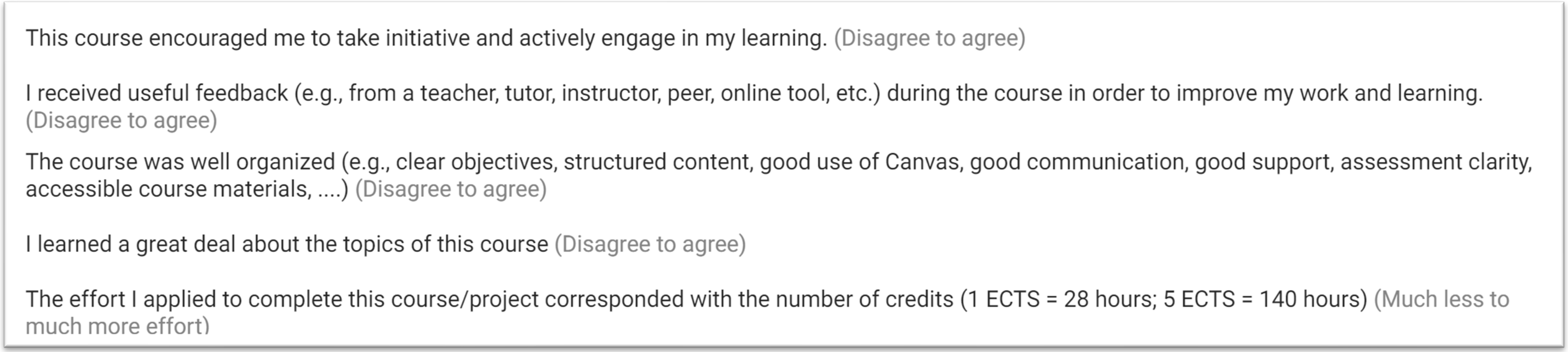
Results and and proposed changes are discussed during committee meetings to **improve the courses and programs!**

# General questions



On a scale from 1 to 10, how would you rate this course (with 1 being "very poor" and 10 being "excellent")? (1 to 10 (points))

Why did you rate the course this way and how can we make it better for future students? Please be respectful, clear, and to the point. (Open question)



This course encouraged me to take initiative and actively engage in my learning. (Disagree to agree)

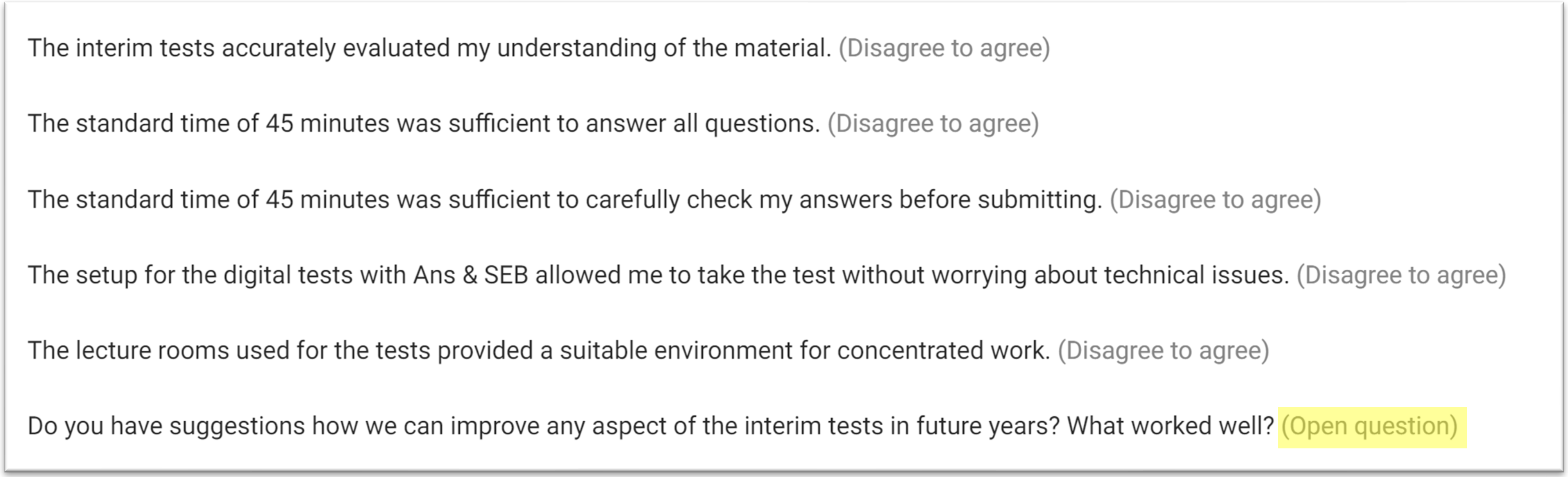
I received useful feedback (e.g., from a teacher, tutor, instructor, peer, online tool, etc.) during the course in order to improve my work and learning. (Disagree to agree)

The course was well organized (e.g., clear objectives, structured content, good use of Canvas, good communication, good support, assessment clarity, accessible course materials, ....) (Disagree to agree)

I learned a great deal about the topics of this course (Disagree to agree)

The effort I applied to complete this course/project corresponded with the number of credits (1 ECTS = 28 hours; 5 ECTS = 140 hours) (Much less to much more effort)

# Interim test questions



The interim tests accurately evaluated my understanding of the material. (Disagree to agree)

The standard time of 45 minutes was sufficient to answer all questions. (Disagree to agree)

The standard time of 45 minutes was sufficient to carefully check my answers before submitting. (Disagree to agree)

The setup for the digital tests with Ans & SEB allowed me to take the test without worrying about technical issues. (Disagree to agree)

The lecture rooms used for the tests provided a suitable environment for concentrated work. (Disagree to agree)

Do you have suggestions how we can improve any aspect of the interim tests in future years? What worked well? (Open question)

Logistics (rooms & invigilators) are responsible for full rooms and 45 min max

*do not have space for students to leave early or visit bathroom*

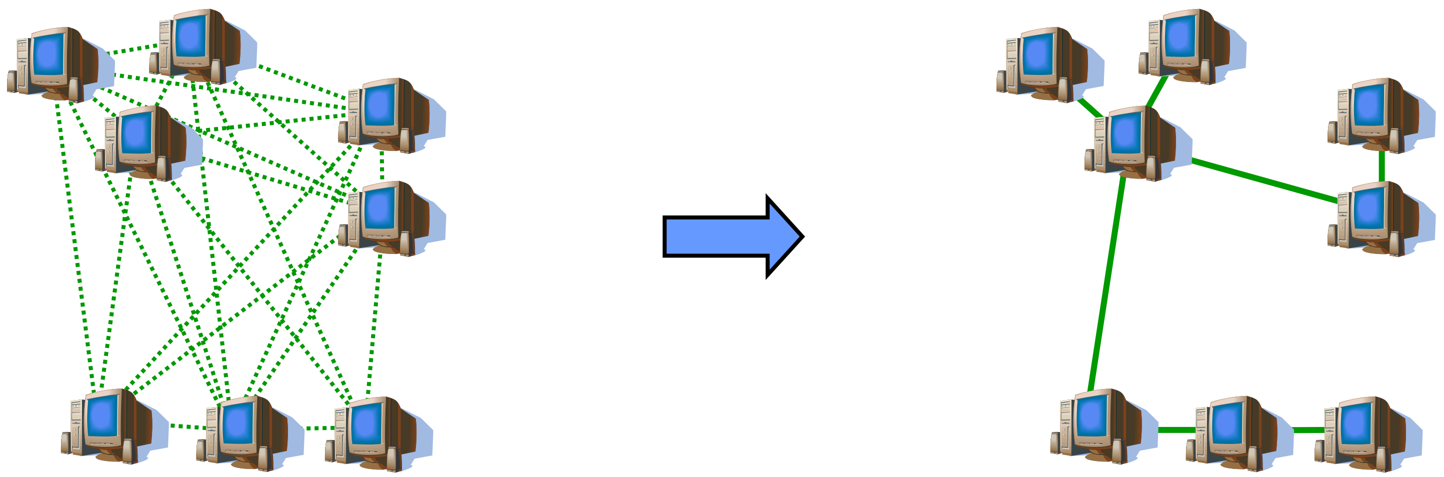
*want to be able to test full segment, so test cannot contain less material*

For better situation next year we need clear signals from students:

1. Interim tests per se are fine (well-organized, prepare for exam, help assess knowledge)
2. Logistics are a problem and hamper performance (induce more stress than needed)

# Motivation

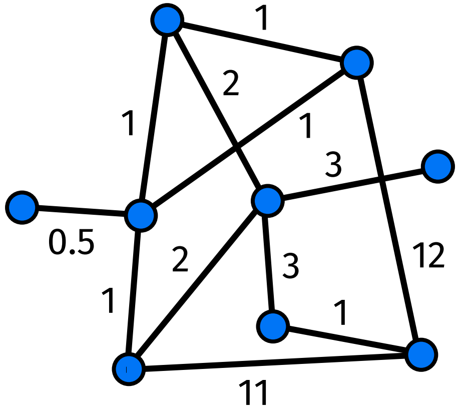
What is the cheapest network that connects a group of computers?



# Minimum spanning graph

Input undirected, weighted graph $G = (V, E)$

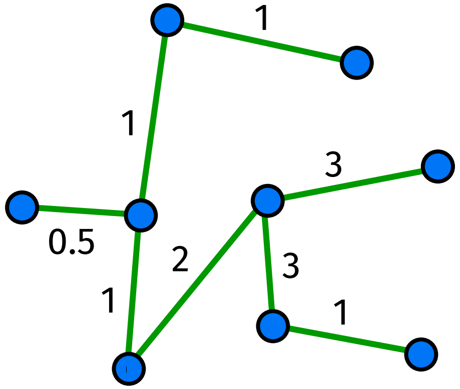
weighted graph = each edge $(u, v)$ has a weight $w(u, v)$



Output a set of edges $T \subset E$ such that

1. $T$ connects all vertices, and
2. $w(T) \ = \sum_{(u,v) \in T} w(u, v)$ is minimized

➡ $T$ is a minimum spanning graph of $G$



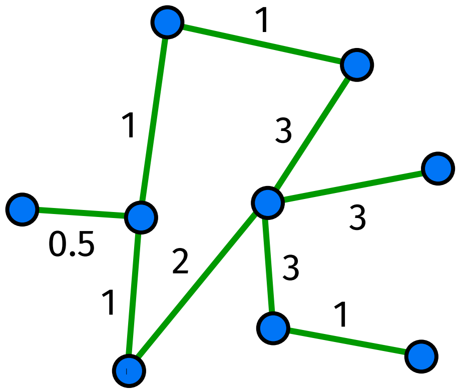
$w(T) = 12.5$

# Minimum spanning tree

Lemma If all edge weights are positive, then the minimum spanning graph is a tree.

Tree = connected, undirected, acyclic graph

Proof:



Such a tree is called a minimum spanning tree (MST)

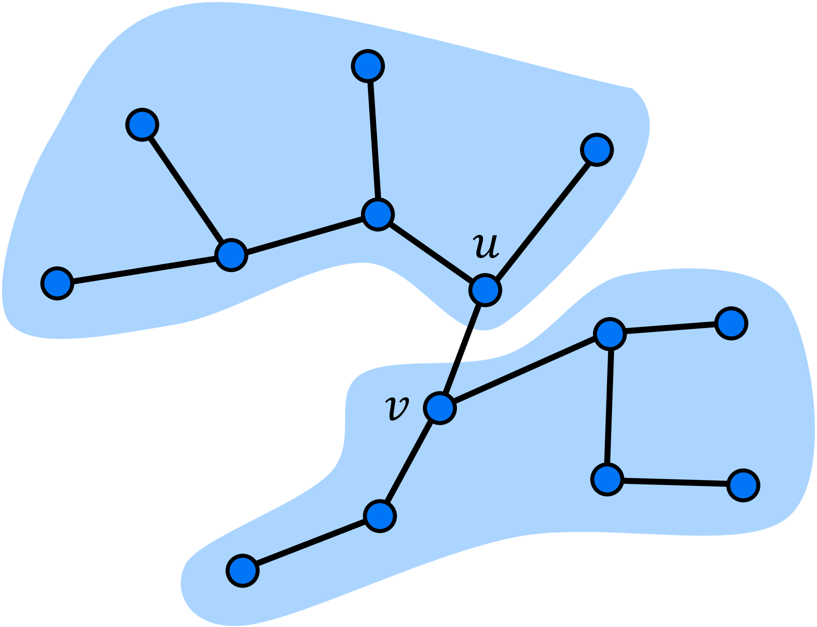
| | |
|---|---|
| Minimum spanning tree | spanning tree whose weight is minimum over all spanning trees |

# Minimum spanning tree

A minimum spanning tree

- has $V - 1$ edges
- has no cycles
- might not be unique

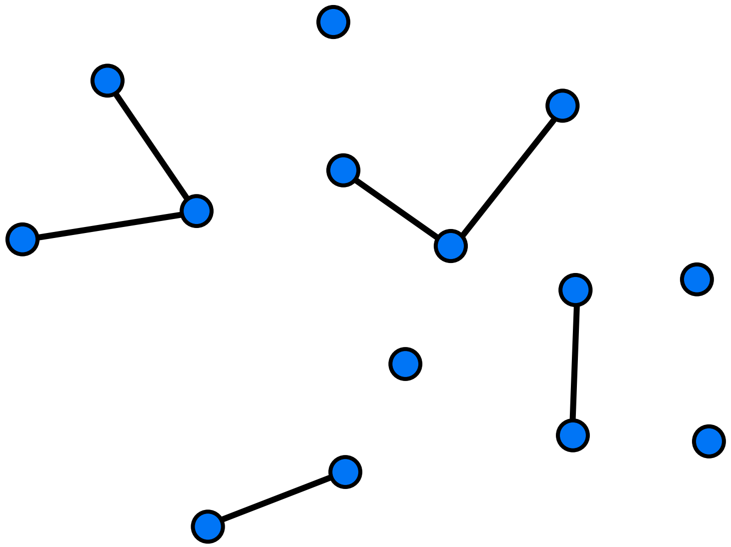
The edge $(u, v)$ is the cheapest way to connect the two components that contain $u$ and $v$, respectively.



# Constructing an MST

We will use an incremental approach

Sub-problem: Building an MST for a collection of components



8 components

Greedy approach

Can we find a (locally) optimal edge that we can add without jeopardizing an optimal solution?

*You'll learn all about greedy algorithms in* Algorithms …

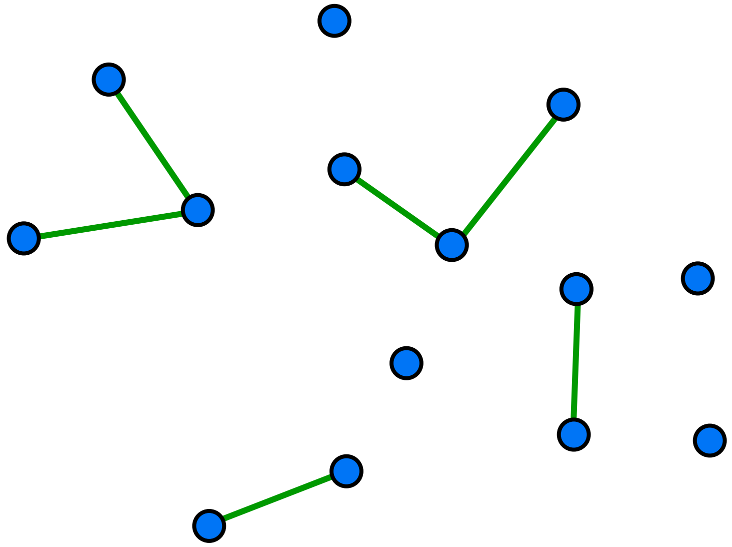
# Constructing an MST

Greedy approach

Can find we find a (locally) optimal edge that we can add without jeopardizing an optimal solution?

$A$ set of edges that have already been added

Invariant $A$ is a subset of some MST



Definition

An edge $(u, v)$ is safe for $A$ if and only if $A \cup \{(u, v)\}$ is also a subset of some MST.

➡ the greedy choice has to be safe

# Generic MST algorithm

```
Generic-MST($G, w$)
$A = \emptyset$
while $A$ is not a spanning tree
    find an edge $(u, v)$ that is safe for $A$
    $A = A \cup \{(u, v)\}$
return $A$
```

## Correctness Proof

Loop Invariant — $A$ is a subset of some MST

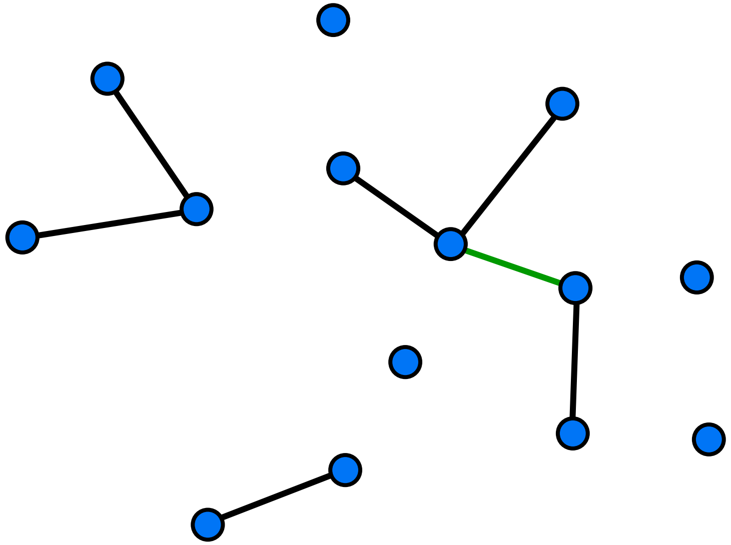
Initialization — The empty set trivially satisfies the loop invariant.

Maintenance — Since we add only safe edges, $A$ remains a subset of some MST.

Termination — All edges added to $A$ are in an MST, so when we stop,
$A$ is a spanning tree that is also an MST.

# Finding a safe edge

Idea    Add the lightest edge that does not introduce a cycle.



# Some definitions …

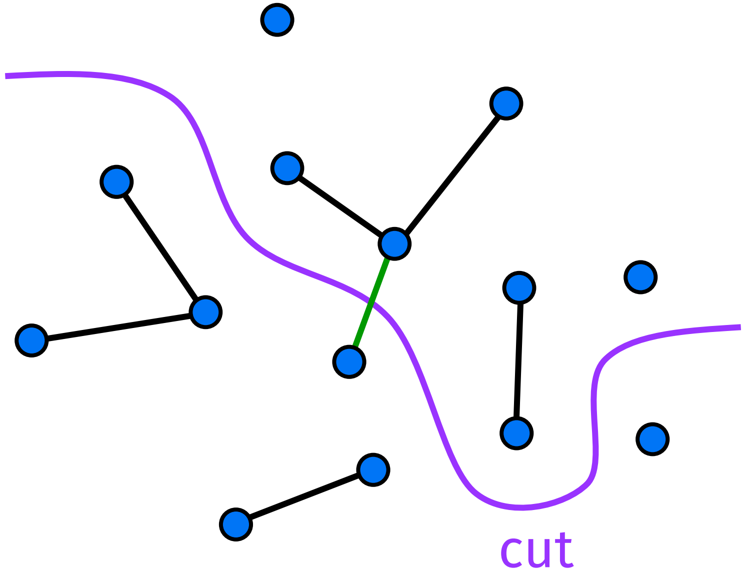
A cut $(S, V - S)$ is a partition of the vertices into disjoint sets $S$ and $V - S$

Edge $(u, v) \in E$ crosses cut $(S, V - S)$ if one endpoint is in $S$ and the other in $V - S$.

A cut respects $A$ if and only if no edge in $A$ crosses the cut.

An edge is a light edge crossing a cut if and only if its weight is minimum over all edges crossing the cut.

*For a given cut, there can be $> 1$ light edge crossing it …*



# Finding a safe edge

Theorem

Let $A$ be a subset of some MST, $(S, V - S)$ be a cut that respects $A$, and $(u, v)$ be a light edge crossing $(S, V - S)$. Then $(u, v)$ is safe for $A$.

# Finding a safe edge

Theorem Let $A$ be a subset of some MST, $(S, V - S)$ be a cut that respects $A$, and $(u, v)$ be a light edge crossing $(S, V - S)$. Then $(u, v)$ is safe for $A$.

Proof Let $T$ be a MST that includes $A$

If $T$ contains $(u, v)$ ➡ done

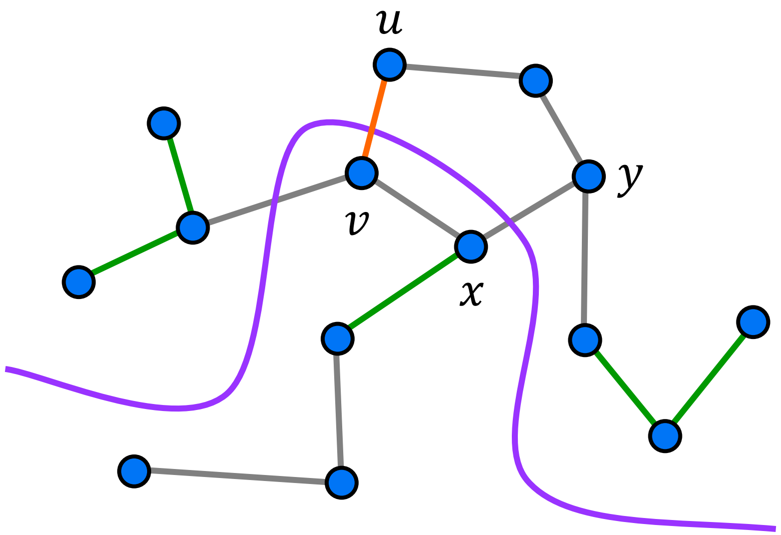
else add $(u, v)$ to $T$ ➡ cycle $c$

$c$ contains at least one edge $(x, y)$ that crosses cut

we have $w(u, v) \leq w(x, y)$ and $(x, y) \notin A$

can construct new tree $T^* = T - (x, y) + (u, v)$
with $w(T^*) = w(T) - w(x, y) + w(u, v) \leq w(T)$

$w(T) \leq w(T^*)$ by definition ➡ $T^*$ is also an MST ■

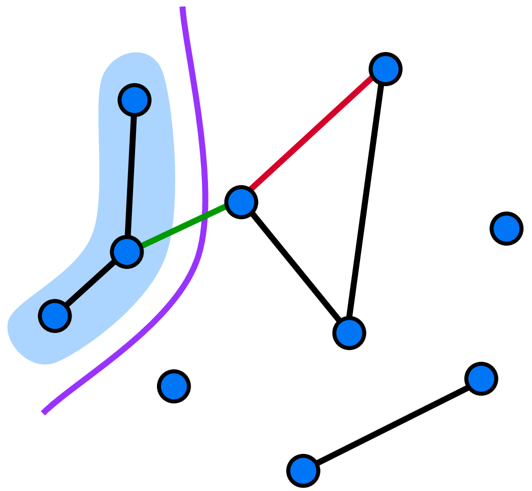


# Two greedy algorithms for MST

## Kruskal's algorithm

A is a forest

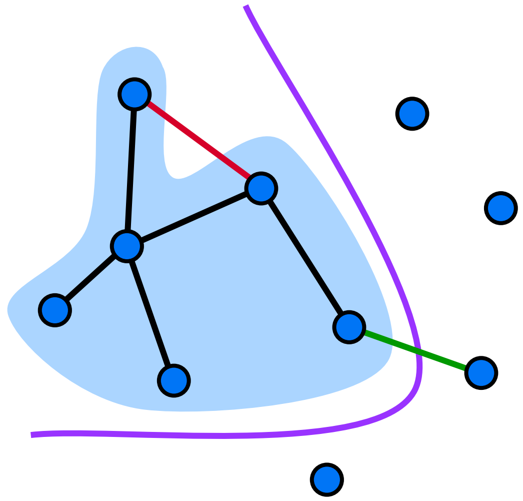
safe edge is lightest edge that connects two components



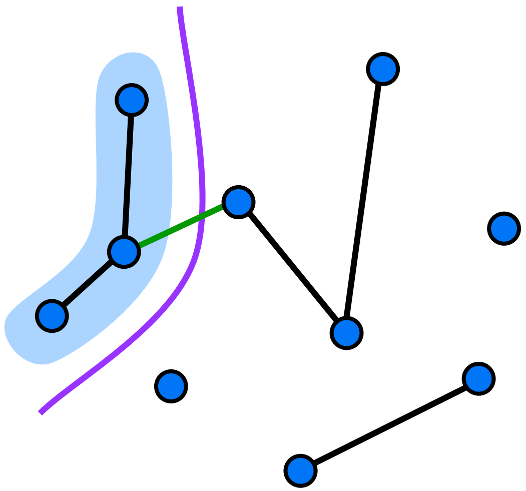
## Prim's algorithm

A is a tree

safe edge is lightest edge that connects new vertex to tree



# Kruskal's algorithm



starts with each vertex being its own component

repeatedly merges two components by
choosing light edge that connects them

scans the set of edges in monotonically increasing order by weight

uses a union-find data structure to determine
whether an edge connects vertices in different components

# Dynamic sets

**Dynamic sets**

Sets that can grow, shrink, or otherwise change over time.

Two types of operations:

| | |
|---|---|
| queries | return information about the set |
| modifying operations | change the set |

**Common queries**

Search, Minimum, Maximum, Successor, Predecessor

**Common modifying operations**

Insert, Delete

# Union-find structure

## Union-Find Structure

Stores a collection of disjoint dynamic sets.

## Operations

Make-Set($x$) creates a new set whose only member is $x$

Union($x$, $y$) unites the dynamic sets that contain $x$ and $y$

Find-Set($x$) finds the set that contains $x$

# Analysis of union-find structures

Union-find structures are often used as an auxiliary data structure by algorithms

➡ total running time over all operations is more important than worst case running time for each operation

Analysis in terms of

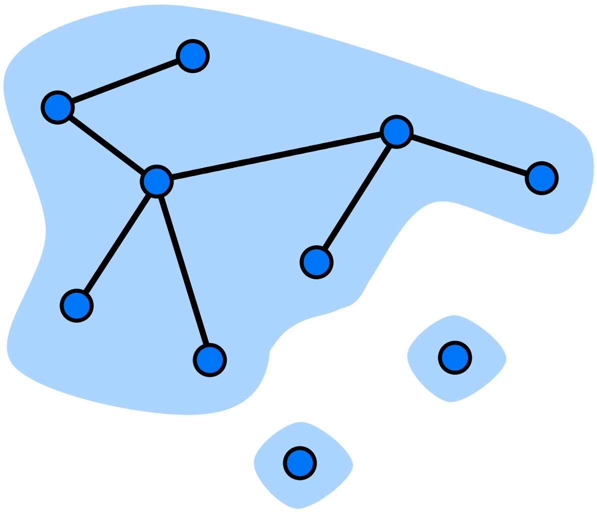
$n$ = # of elements = # Make-Set operations

$m$ = total # of operations (incl. Make-Set)

Theorem A sequence of $m$ operations, of which $n$ are Make-Set, takes $O(m\, \alpha(n))$ time in the worst case.

# Example application: connected components

Maintain the connected components of a graph $G = (V, E)$ under edge insertions.



```
Same-Component($u$, $v$)
  if Find-Set($u$) == Find-Set($v$)
      return true
  else
      return false
```

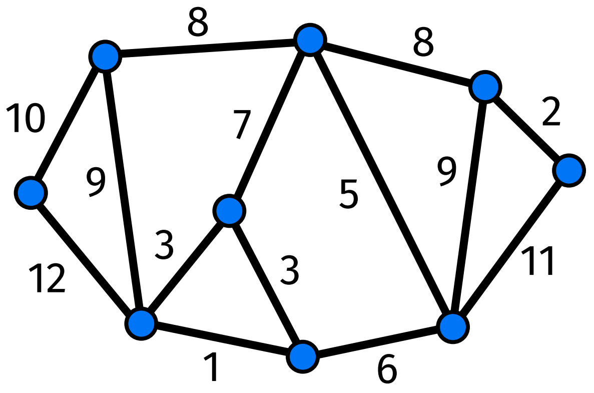
```
Connected-Components($V$, $E$)
  for each vertex $v \in V$
      Make-Set($v$)
  for each edge $(u, v) \in E$
      Insert-Edge($u, v$)
```

```
Insert-Edge($u$, $v$)
  if Find-Set($u$) $\neq$ Find-Set($v$)
      Union($u$, $v$)
```

# Kruskal’s algorithm

## Kruskal’s algorithm

$G = (V, E)$ is a connected, undirected, weighted graph.



```
Kruskal($V$, $E$, $w$)
  $A = \emptyset$
  for each vertex $v \in V$
      Make-Set($v$)
  sort $E$ into non-decreasing order by weight $w$
  for each $(u, v)$ taken from the sorted list
      if Find-Set($u$) $\neq$ Find-Set($v$)
          $A = A \cup \{(u, v)\}$
          Union($u$, $v$)
  return $A$
```

# Kruskal’s algorithm: Analysis

```
Kruskal($V$, $E$, $w$)
1  $A = \emptyset$
2  for each vertex $v \in V$
3          Make-Set($v$)
4  sort $E$ into non-decreasing order by weight $w$
5  for each $(u, v)$ taken from the sorted list
6          if Find-Set($u$) $\neq$ Find-Set($v$)
7                  $A = A \cup \{(u, v)\}$
8                  Union($u$, $v$)
9  return $A$
```

| | |
|---|---|
| 1 | $O(1)$ |
| 2 | $V$ Make-Set |
| 3 | |
| 4 | $O(E \log E)$ |
| 5 | $O(E)$ Find-Set and Union |
| 6 | |
| 7 | |
| 8 | |
| 9 | $O(1)$ |

Use union-find data structure with union-by-rank and path compression

**Theorem** A sequence of $m$ operations, of which $n$ are Make-Set, takes $O(m\,\alpha(n))$ time in the worst case.

**Running time** $O((V + E)\,\alpha(V)) + O(E \log E)$

# Kruskal’s algorithm: Analysis

```
Kruskal($V$, $E$, $w$)
1  $A = \emptyset$
2  for each vertex $v \in V$
3          Make-Set($v$)
4  sort $E$ into non-decreasing order by weight $w$
5  for each $(u, v)$ taken from the sorted list
6          if Find-Set($u$) $\neq$ Find-Set($v$)
7                  $A = A \cup \{(u, v)\}$
8                  Union($u$, $v$)
9  return $A$
```

|   |   |
|---|---|
| 1 | $O(1)$ |
| 2 | $V$ Make-Set |
| 3 |   |
| 4 | $O(E \log E)$ |
| 5 | $O(E)$ Find-Set and Union |
| 6 |   |
| 7 |   |
| 8 |   |
| 9 | $O(1)$ |

**Running time** $O((V + E)\,\alpha(V)) + O(E \log E)$

$G$ is connected ➡ $E \geq V - 1$ ➡ $O\big(E\,\alpha(V)\big) + O(E \log E)$

$E \leq V^2$ ➡ $\log E = O(2 \log V) = O(\log V)$

$\alpha(V) = O(\log V)$ ➡ total time is $O(E \log V)$

*If edges are already sorted* ➡ $O(E\,\alpha(V))$ *… almost linear …*

# Prim’s algorithm

## Prim’s algorithm

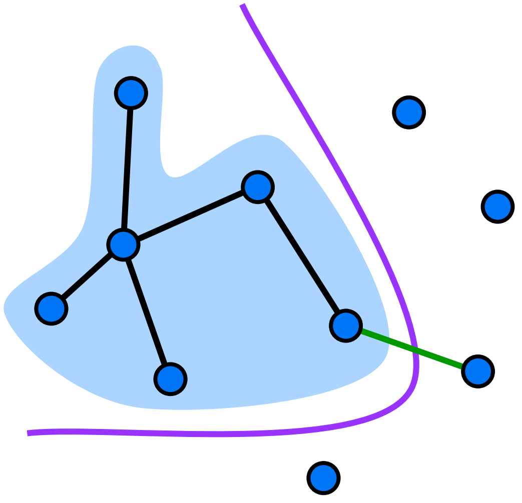
builds one tree, so $A$ is always a tree

starts from an arbitrary root $r$

at each step, finds a light edge crossing cut $(V_A, V - V_A)$, where $V_A =$ vertices $A$ is incident on

Q How can we find this light edge quickly?

A Use a priority queue $Q$



# Priority queue

Min-priority queue

abstract data type (ADT) that stores a set $S$ of elements,
each with an associated $\mathrm{key}$ (integer value).

Operations

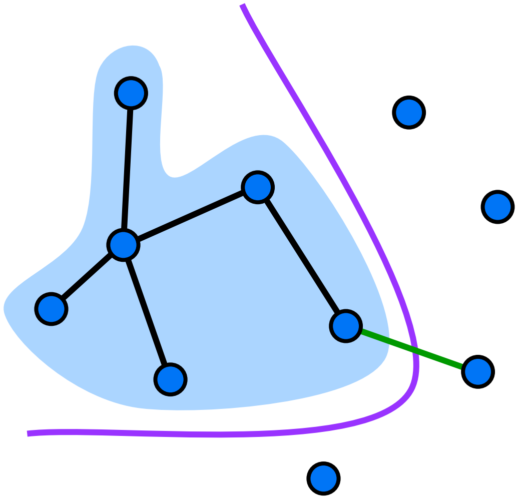
Insert($S$, $x$): inserts element $x$ into $S$, that is, $S \leftarrow S \cup \{x\}$

Minimum($S$): returns the element of $S$ with the smallest $\mathrm{key}$

Extract-Min($S$): removes and returns the element of $S$ with the smallest $\mathrm{key}$

Decrease-Key($S$, $x$, $k$): gives $x.\mathrm{key}$ the value $k$
condition: $k$ is smaller than the current value of $x.\mathrm{key}$

# Prim's algorithm



## Prim's algorithm

builds one tree, so $A$ is always a tree

starts from an arbitrary "root" $r$

at each step, finds a light edge crossing cut $(V_A, V - V_A)$, where $V_A =$ vertices $A$ is incident on

Q How can we find this light edge quickly?

A Use a priority queue $Q$

Elements are vertices in $V - V_A$

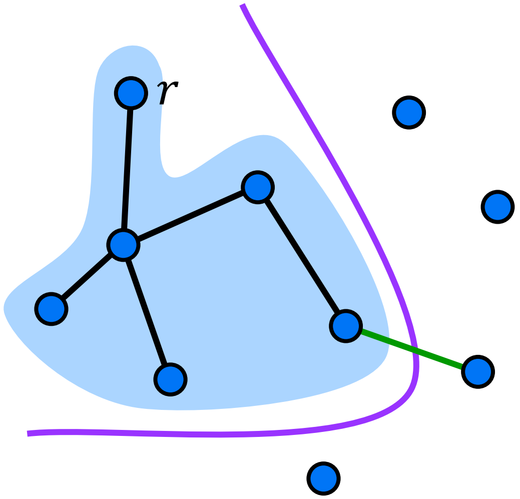
Key of vertex $v$ is minimum weight of any edge $(u, v)$, where $u \in V_A$

Extract-Min returns the vertex $v$ such there exists $u \in V_A$ and $(u, v)$ is light edge crossing $(V_A, V - V_A)$

Key of $v$ is $\infty$ if $v$ is not adjacent to any vertices in $V_A$

# Prim’s algorithm

## Prim’s algorithm



the edges of $A$ form a rooted tree with root $r$

$r$ is given as input, but can be any vertex

each vertex knows its parent in the tree, parent of $v$ is $v.\pi$

$v.\pi = NIL$ if $v = r$ or $v$ has no parent (yet)

as algorithm progresses, $A = \{(v, v.\pi): v \in V - \{r\} - Q\}$

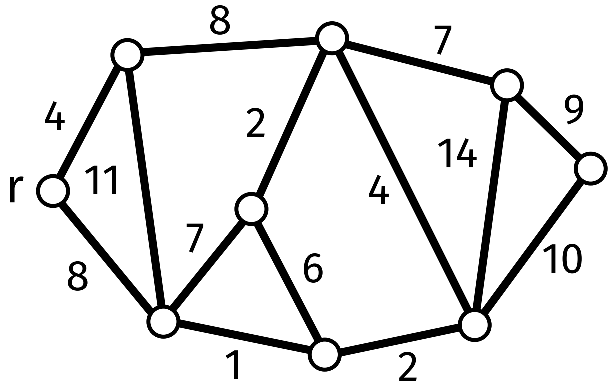
at termination, $V_A = V$ ➡ $Q = \emptyset$, so MST is $A = \{(v, v.\pi):\ v \in V - \{r\}\}$

# Prim’s algorithm

## Prim’s algorithm

$G = (V, E)$ is a connected, undirected, weighted graph

```
Prim($V, E, w, r$)
$Q = \emptyset$
for each vertex $u \in V$
    $u.\mathrm{key} = \infty$
    $u.\pi = NIL$
    Insert($Q, u$)
Decrease-Key($Q, r, 0$)  // $r.key = 0$
while $Q \neq \emptyset$
    $u =$ Extract-Min($Q$)
    for each vertex $v \in \mathrm{Adj}[u]$
        if $v \in Q$ and $w(u, v) < v.\mathrm{key}$
            $v.\pi = u$
            Decrease-Key($Q, v, w(u, v)$)
```



# Prim’s algorithm: Analysis

```
Prim($V, E, w, r$)
1   $Q = \emptyset$
2   for each vertex $u \in V$
3       $u.\mathrm{key} = \infty$
4       $u.\pi = NIL$
5       Insert($Q, u$)
6   Decrease-Key($Q, r, 0$)   // $r.key = 0$
7   while $Q \neq \emptyset$
8       $u$ = Extract-Min($Q$)
9       for each vertex $v \in \mathrm{Adj}[u]$
10          if $v \in Q$ and $w(u, v) < v.\mathrm{key}$
11              $v.\pi = u$
12              Decrease-Key($Q, v, w(u, v)$)
```

$$1 \quad O(1)$$

$$\left.\begin{array}{l}2\\3\\4\\5\end{array}\right\} O(V \log V)$$

$$6 \quad O(\log V)$$

$$\left.\begin{array}{l}7\\8\\9\\10\\11\\12\end{array}\right\} \begin{array}{l} V \text{ Extract-Min} \;➡\; O(V \log V) \\ \leq E \text{ Decrease-Key} \;➡\; O(E \log V) \end{array}$$

Total: $O(E \log V)$

Implement the priority queue with a binary heap or a balanced binary search tree

➡ Insert, Decrease-Key, and Extract-Min in $O(\log V)$ time

*Decrease-Key can be done in $O(1)$ amortized time with a Fibonacci heap (see textbook Chapter 20)*

# Data Structures

That’s it!