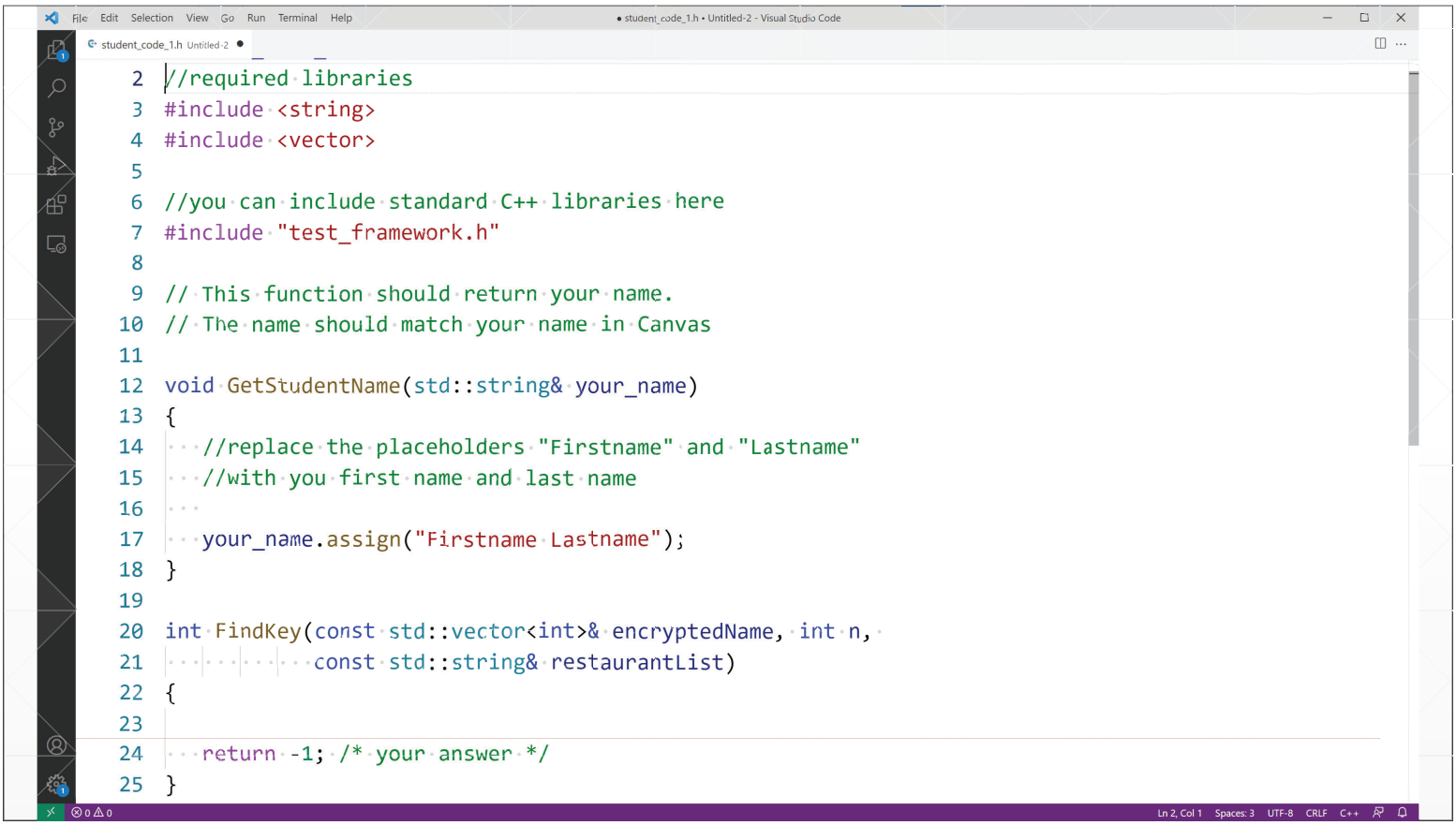
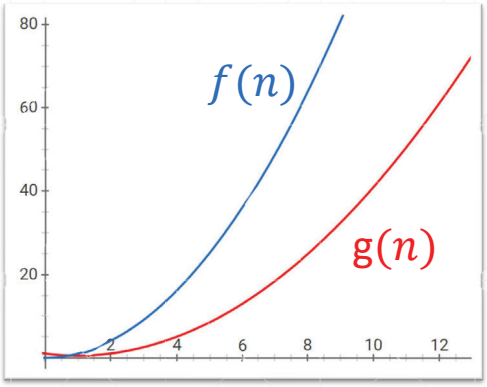


Big O Notation

Big O Notation



Suppose $f(n), g(n) > 0$.

- What does $f(n) = O(g(n))$ mean?
- Think of it: $f(n) \leq O\big(g(n)\big)$.
- $f(n) = O(g(n))$ if there exists a positive number C such that

$$f(n) \leq C \cdot g(n)$$

for all n.

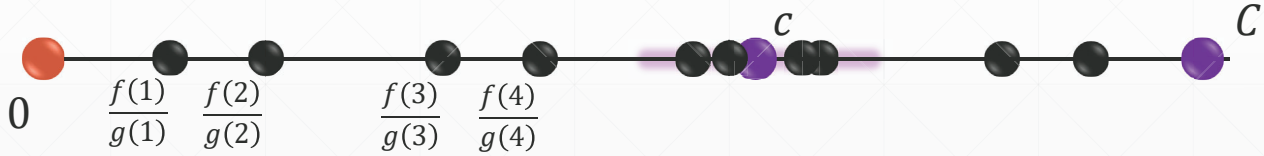
Sufficient Condition

- If

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$$

- Then, $f(n) = O(g(n))$.

Proof.



Examples

- True or False: $n^3 = O(n^4)$.

$$\lim_{n\to\infty} \frac{n^3}{n^4} = \lim_{n\to\infty} \frac{1}{n} = 0$$

- True or False: $5n^3 + 2n^2 + 1 = O(n^3)$.

$$\lim_{n\to\infty} \frac{5n^3 + 2n^2 + 1}{n^3} = 5$$

Sufficient Condition

- If

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$$

- Then, $f(n) = O(g(n))$.
- If

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

- Then, $f(n) \neq O(g(n))$.

Examples

- True or False: $n^4 = O(n^3)$.

$$\lim_{n\to\infty} \frac{n^4}{n^3} = \lim_{n\to\infty} n = \infty$$

Big O Notation & Running Time

- Running time is $O(n \log n)$ if the running time is bounded by

$$C\ n \log n$$


ILLINOIS
200

Greedy Algorithms

CS 336: Design and Analysis of Algorithms

What are Greedy Algorithms?

- How to spend your vacation?

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Movies	Movies	Movies	Movies	Movies	Movies	Movies
Theater	Theater	Theater	Theater	Theater	Theater	Theater
Museum	Museum	Museum	Museum	Museum	Museum	Museum
Party	Party	Party	Party	Party	Party	Party
Restaurant	Restaurant	Restaurant	Restaurant	Restaurant	Restaurant	Restaurant

What are Greedy Algorithms?

- How to spend your vacation?

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Movies	Movies	Movies	Movies	Movies	Movies	Movies
Theater	Theater	Theater	Theater	Theater	Theater	Theater
Museum	Museum	Museum	Museum	Museum	Museum	Museum
Party	Party	Party	Party	Party	Party	Party
Restaurant	Restaurant	Restaurant	Restaurant	Restaurant	Restaurant	Restaurant

What are Greedy Algorithms?

- How to spend your vacation?

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Movies	**Movies**	Movies	Movies	Movies	Movies	Movies
Theater	Theater	Theater	Theater	Theater	Theater	Theater
Museum	Museum	Museum	Museum	Museum	Museum	Museum
Party	Party	Party	Party	Party	Party	Party
Restaurant	Restaurant	Restaurant	Restaurant	Restaurant	Restaurant	Restaurant

What are Greedy Algorithms?

- How to spend your vacation?

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Movies	**Movies**	Movies	Movies	Movies	Movies	Movies
Theater	Theater	Theater	Theater	Theater	Theater	Theater
Museum	Museum	**Museum**	Museum	Museum	Museum	Museum
Party	Party	Party	Party	Party	**Party**	**Party**
Restaurant	Restaurant	Restaurant	**Restaurant**	**Restaurant**	Restaurant	Restaurant

What are Greedy Algorithms?

- How to spend your vacation? Given your happiness scores.

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
10	11	10	12	11	22	12
12	13	9	15	14	11	14
15	16	3	25	15	70	18
9	8	15	15	12	28	11
10	20	12	15	1	23	100

What are Greedy Algorithms?

- How to spend your vacation? Given your happiness scores.

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
10	11	10	12	11	22	12
12	13	9	15	14	11	14
15	16	3	**25**	**15**	**70**	18
9	8	**15**	15	12	28	11
10	**20**	12	15	1	23	**100**

What are Greedy Algorithms?

- How to spend your vacation? Given your happiness scores.

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
10	11	10	12	11	22	12
12	13	9	15	14	11	14
15	16	3	**25**	**15**	**70**	18
9	8	**15**	15	12	28	11
10	**20**	12	15	1	23	**100**

Informal Definition: Greedy algorithms make decisions to maximize "immediate profit" or minimize "immediate loss".

Why Greedy Algorithms?

- **Informal Definition:** Greedy algorithms
  - Solve a problem in multiple steps;
  - Each time maximizing profit or minimizing loss.
- Greedy algorithms are usually fast & simple.
- Use them when that's possible!

Basic Examples

Problem: You are given n numbers $x_1, \dots, x_n$ pick k numbers with the maximum sum ($k \leq n$).

Basic Examples

Problem: You are given n numbers $x_1, \dots, x_n$ pick k numbers with the maximum sum ($k \leq n$).

Solution:

Let $X = \{x_1, \dots, x_n\}$.
Sort X in decreasing order.
return the first k elements in X.

What do we need to do?

- Analyze the running time.
- Show that the algorithm is correct.
  - The algorithm must output a ***feasible solution***.
  - The algorithm must output the ***optimal solution*** *i.e.* the best solution among all ***feasible solution***.

Basic Examples

Problem: You are given n numbers $x_1, \dots, x_n$ pick k numbers with the maximum sum.

Solution:

> Let $X = \{x_1, \dots, x_n\}$.
> Sort X in decreasing order.
> return the first k elements in X.

Running time: $O(n \log n)$. Can implement in $O(n)$ time.

Correctness

- The solution is feasible: It contains exactly k numbers from $x_1, \dots, x_n$.

Correctness

- Consider the optimal and algorithmic solutions:

$$OPT = \{o_1, \dots, o_k\}$$
$$ALG = \{a_1, \dots, a_k\}$$

- Sort $o_1 \geq o_2 \geq \cdots \geq o_k$ and $a_1 \geq a_2 \geq \cdots \geq a_k$.

Proof: Prove by induction that the prefixes of length i in OPT and ALG are the same for $i = 0, \dots, k$.

- Consider the optimal and algorithmic solutions:

$$OPT = \{o_1, \dots, o_k\}$$
$$ALG = \{a_1, \dots, a_k\}$$

- Sort $o_1 \geq o_2 \geq \cdots \geq o_k$ and $a_1 \geq a_2 \geq \cdots \geq a_k$.

Proof: Prove by induction that the prefixes of length i in OPT and ALG are the same for $i = 0, \dots, k$.

- Consider the optimal and algorithmic solutions:

$$OPT = \{o_1, \dots, o_k\}$$
$$ALG = \{a_1, \dots, a_k\}$$

- Sort $o_1 \geq o_2 \geq \cdots \geq o_k$ and $a_1 \geq a_2 \geq \cdots \geq a_k$.

Proof: Prove by induction that the prefixes of length i in OPT and ALG are the same for $i = 0, \dots, k$.

Base case: Prefixes of length 0 are empty and, hence, are equal.

Induction Step: Suppose that $a_j = o_j$ for all $j < i$ and $i \leq k$. We show that $a_i = o_i$.

- Consider $OPT = \{o_1, \dots, o_k\}$ and $ALG = \{a_1, \dots, a_k\}$
- Sort $o_1 \geq o_2 \geq \cdots \geq o_k$ and $a_1 \geq a_2 \geq \cdots \geq a_k$.

Proof: Prove by induction that the prefixes of length i in OPT and ALG are the same for $i = 0, \dots, k$.

Induction Step: Suppose that $a_j = o_j$ for all $j < i$. We show that $a_i = o_i$. Observe that a_i is the largest element in

$$\{x_1, \dots, x_n\} \setminus \{a_1, \dots, a_{i-1}\}$$

Particularly, $a_i > o_i$ and $a_i > o_i \geq o_{i+1} \geq \cdots \geq o_k$.

Consider $O' = \{o_1, \dots o_{i-1}, a_i, o_{i+1}, \dots o_k\}$. It is a *feasible solution,* and its value is greater than the value of OPT.

- Consider $OPT = \{o_1, \dots, o_k\}$ and $ALG = \{a_1, \dots, a_k\}$
- Sort $o_1 \geq o_2 \geq \cdots \geq o_k$ and $a_1 \geq a_2 \geq \cdots \geq a_k$.

Particularly, $a_i > o_i$ and $a_i > o_i \geq o_{i+1} \geq \cdots \geq o_k$.

$$O' = \{o_1, \dots o_{i-1}, \boldsymbol{a_i}, o_{i+1}, \dots o_k\}$$
$$OPT = \{o_1, \dots o_{i-1}, \boldsymbol{o_i}, o_{i+1}, \dots, o_k\}$$

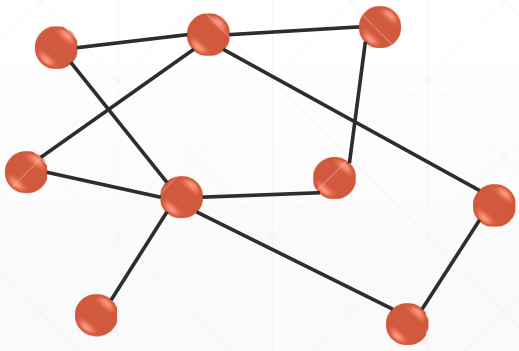
Minimum Spanning Tree

CS 336: Design and Analysis of Algorithms

Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

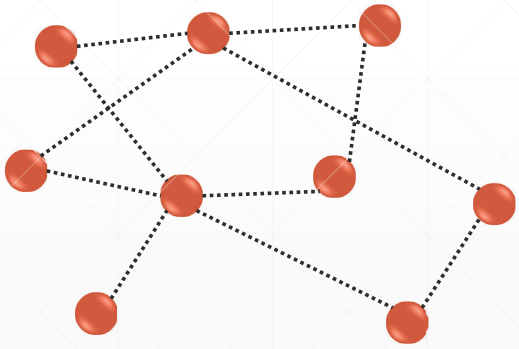
- Weighted graph:
  - V – vertices:
  - E – edges:
  - w_{uv} – weights of edges.



Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

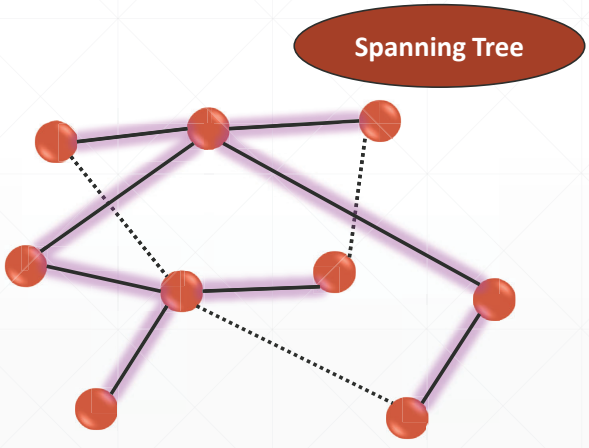
- Weighted graph:
  - V – vertices:
  - E – edges:
  - w_{uv} – weights of edges.



Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

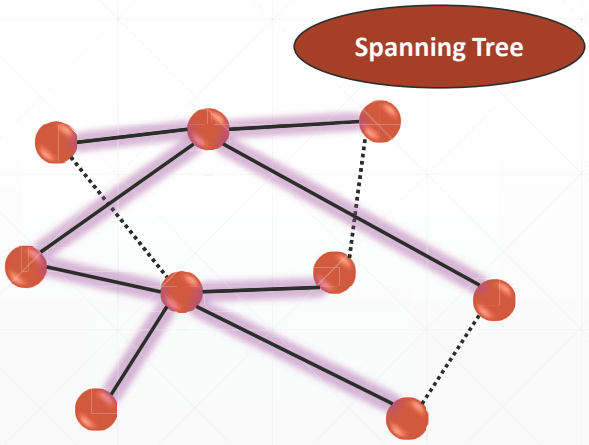
- T is a spanning tree of G if
  - T is a subgraph of G;
  - T covers all vertices of G;
  - T is a tree.



Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

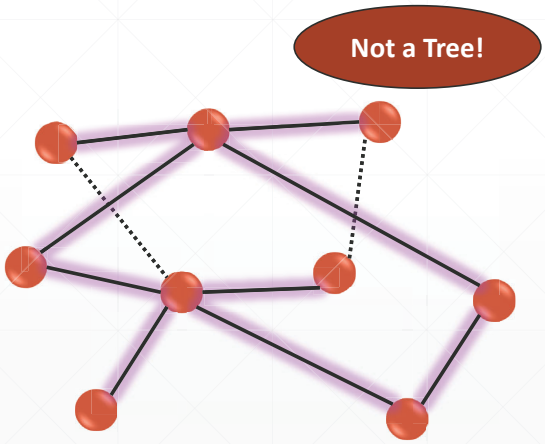
- T is a spanning tree of G if
  - T is a subgraph of G;
  - T covers all vertices of G;
  - T is a tree.



Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

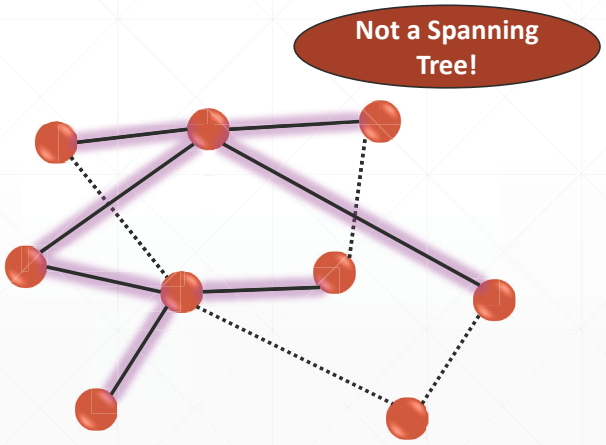
- T is a spanning tree of G if
  - T is a subgraph of G;
  - T covers all vertices of G;
  - T is a tree.



Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

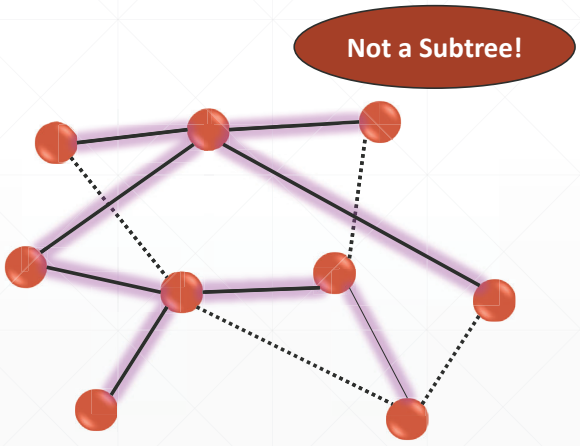
- T is a spanning tree of G if
  - T is a subgraph of G;
  - T covers all vertices of G;
  - T is a tree.



Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

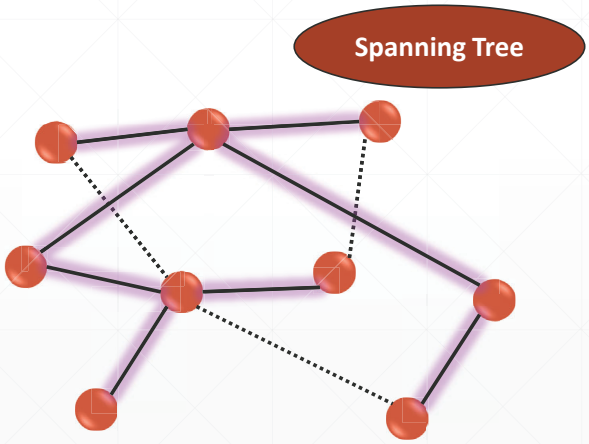
- T is a spanning tree of G if
  - T is a subgraph of G;
  - T covers all vertices of G;
  - T is a tree.



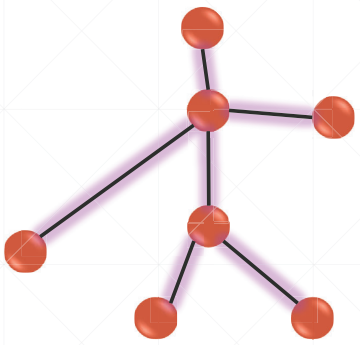
Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

- T is a spanning tree of G if
  - T is a subgraph of G;
  - T covers all vertices of G;
  - T is a tree.



Trees



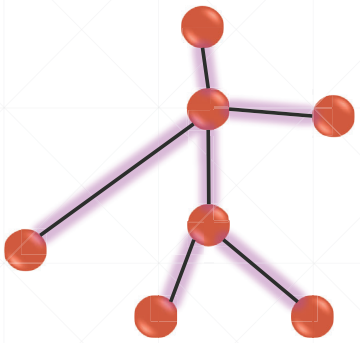
- Tree is a connected graph without cycles.
- In a tree: $\#edges = \#vertices - 1$

- We assume that $G = (V, E, w)$ is a connected graph.
- Denote $n = |V|; m = |E|$.

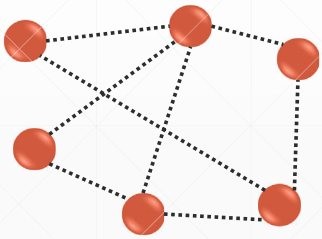
- We need to pick $(n - 1)$ edges in G.

Trees

- Tree is a connected graph without cycles.
- In a tree: $\#edges = \#vertices - 1$



- We assume that $G = (V, E, w)$ is a connected graph.
- Denote $n = |V|; m = |E|$.

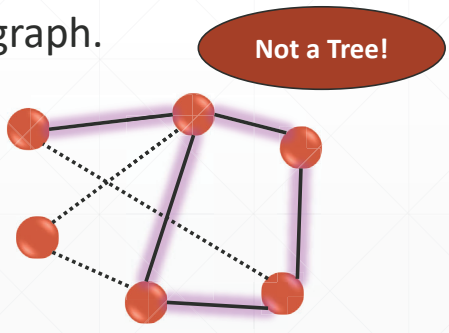


- We need to pick $(n - 1)$ edges in G.
- Can we pick $(n - 1)$ lightest edges in G?

Trees

- Tree is a connected graph without cycles.
- In a tree: $\#edges = \#vertices - 1$

- We assume that $G = (V, E, w)$ is a connected graph.
- Denote $n = |V|; m = |E|$.

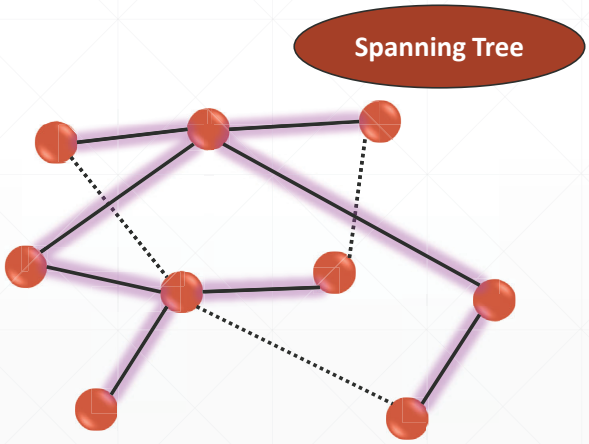


- We need to pick $(n - 1)$ edges in G.
- Can we pick $(n - 1)$ lightest edges in G? **No** ☹

Minimum Spanning Tree (MST)

- **Problem:** Given a weighted graph $G = (V, E, w)$ find a minimum weight spanning tree of G.

$$\min \sum_{(u,v) \text{ in } T} w_{uv}$$



Feasibility and Optimality

- A ***feasible*** solution to the Minimum Spanning Tree problem is a spanning tree.
- An ***optimal*** solution is a feasible solution of minimum cost.

$$cost(S) = \sum_{(u,v) \in E} w_{uv}$$

Kruskal's Algorithm

S – set of edges. Let $S = \emptyset$.
Sort all edges in E by weight w_e

while ($|S| < n - 1$):
  Get the lightest edge e from the list
  If adding e to S doesn't create a cycle,
    then add e to S
    else discard e

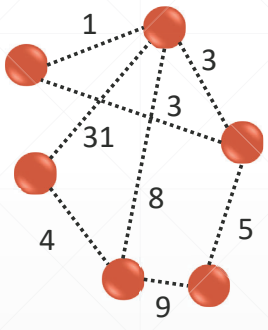
return S

Kruskal's Algorithm

S – set of edges. Let $S = \emptyset$.
Sort all edges in E by weight w_e

while ($|S| < n - 1$):
  Get the lightest edge e from the list
  If adding e to S doesn't create a cycle,
    then add e to S
    else discard e

return S

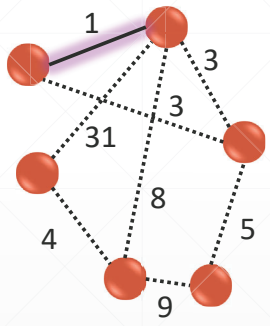


Kruskal's Algorithm

S – set of edges. Let $S = \emptyset$.
Sort all edges in E by weight w_e

while ($|S| < n - 1$):
  Get the lightest edge e from the list
  If adding e to S doesn't create a cycle,
    then add e to S
    else discard e

return S

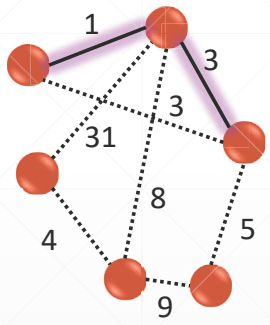


Kruskal's Algorithm

S – set of edges. Let $S = \emptyset$.
Sort all edges in E by weight w_e

while ($|S| < n - 1$):
  Get the lightest edge e from the list
  If adding e to S doesn't create a cycle,
    then add e to S
    else discard e

return S

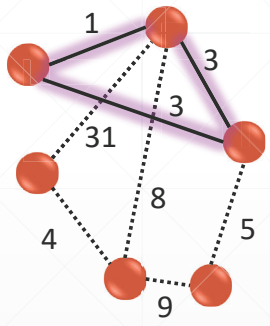


Kruskal's Algorithm

S – set of edges. Let $S = \emptyset$.
Sort all edges in E by weight w_e

while ($|S| < n - 1$):
  Get the lightest edge e from the list
  If adding e to S doesn't create a cycle,
    then add e to S
    else discard e

return S

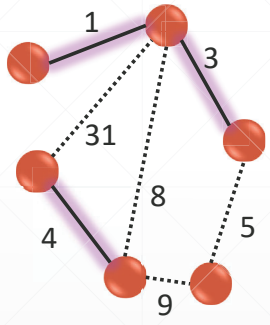


Kruskal's Algorithm

S – set of edges. Let $S = \emptyset$.
Sort all edges in E by weight w_e

while ($|S| < n - 1$):
  Get the lightest edge e from the list
  If adding e to S doesn't create a cycle,
    then add e to S
    else discard e

return S

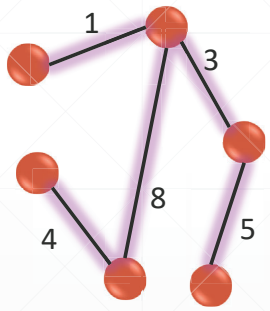


Kruskal's Algorithm

S – set of edges. Let $S = \emptyset$.
Sort all edges in E by weight w_e

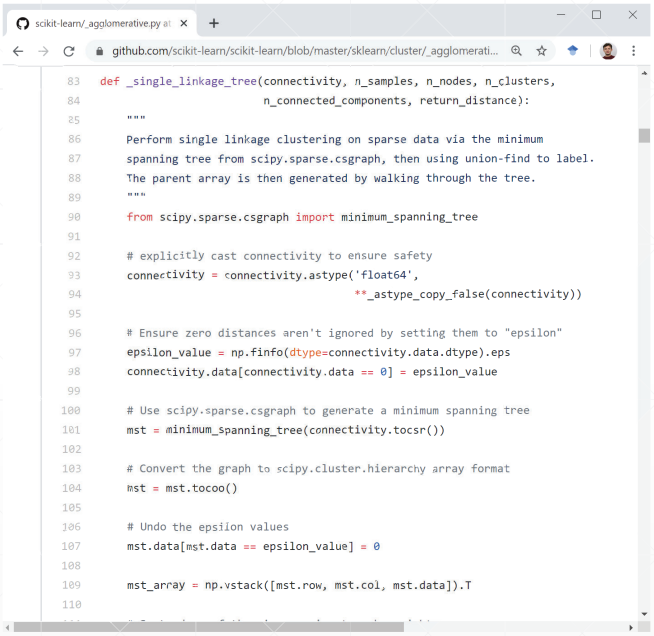
while ($|S| < n - 1$):
  Get the lightest edge e from the list
  If adding e to S doesn't create a cycle,
    then add e to S
    else discard e

return S



Performance Analysis

- Sorting edges by weight: $O(m \log m)$
- For each edge, checking if adding e to the solution S creates a cycle in S: $O(\log^* m)$.
  - Use *Union-Find* or *Disjoint-Set* data structure.
- Total running time is $O(m \log m)$



Scikit-learn Library

Applications to ML

- **Single-linkage agglomerative clustering** is based on Kruskal's algorithm.
- **Single-linkage clustering:** Run Kruskal's algorithm. Stop when we have k disjoint sets. Output these sets as clusters.

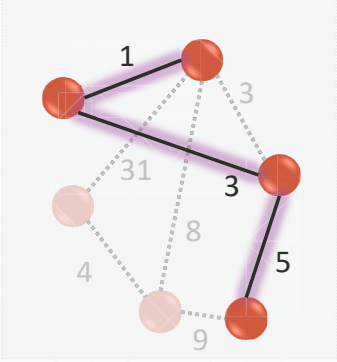
Minimum Spanning Tree II

CS 336: Design and Analysis of Algorithms

Prim's Algorithm

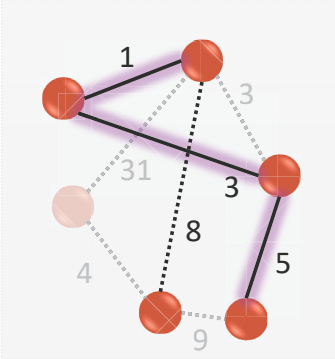
Prim's Algorithm

Idea: Maintain a subtree of G. At every step of the algorithm, attach a vertex outside of the tree with the minimal connection cost.



Prim's Algorithm

Idea: Maintain a subtree of G. At every step of the algorithm, attach a vertex outside of the tree with the minimal connection cost.

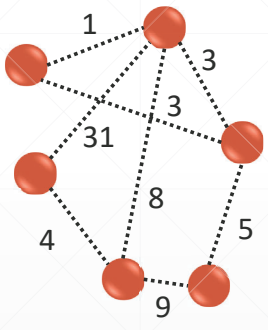


Prim's Algorithm

Pick an arbitrary vertex $r \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S

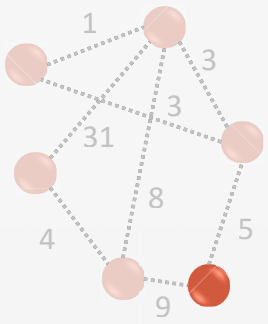


Prim's Algorithm

Pick an arbitrary vertex $r \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S

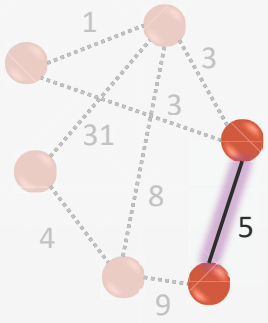


Prim's Algorithm

Pick an arbitrary vertex $r \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S

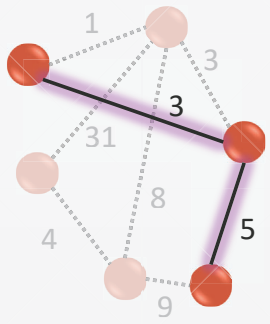


Prim's Algorithm

Pick an arbitrary vertex $r \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S

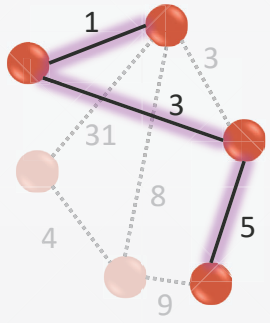


Prim's Algorithm

Pick an arbitrary vertex $r \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S

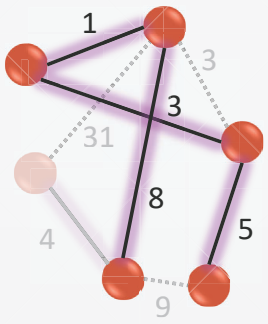


Prim's Algorithm

Pick an arbitrary vertex $r \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S

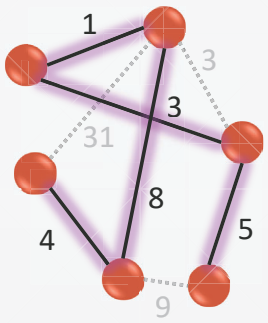


Prim's Algorithm

Pick an arbitrary vertex $r \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S



We need to

- Prove that the algorithm returns a correct solution.
  - Solution is ***feasible*** – it is a spanning tree.
  - Solution is ***optimal*** – it has the minimum cost.
- Analyze the running time.
  - Specify implementation details.

Feasibility and Optimality

- A ***feasible*** solution to the Minimum Spanning Tree problem is a spanning tree.
- An ***optimal*** solution is a feasible solution of minimum cost.

$$cost(S) = \sum_{(u,v)\in E} w_{uv}$$

Why does the algorithm terminate?

- **Assumption:** G is a connected graph.
- Hence, at every step of the algorithm, there is at least one edge leaving X (otherwise, X would be disconnected from $V \backslash X$).
- Consequently, at every step the algorithm adds one vertex to X.
- Thus, in $(n-1)$ steps the algorithm will terminate.
- **Corollary:** edges in S form a spanning tree. Why?

Why does the algorithm terminate?

- **Assumption:** G is a connected graph.
- Hence, at every step of the algorithm, there is at least one edge leaving X (otherwise, X would be disconnected from $V \backslash X$).
- Consequently, at every step the algorithm adds one vertex to X.
- Thus, in $(n-1)$ steps the algorithm will terminate.
- **Corollary:** edges in S form a spanning tree. Why?
- … because S contains $(n-1)$ edges; S is connected.

Optimality

- **Assume:** All weights w_{uv} are distinct.
- Let S_{opt} be the optimal set of edges.
- Let e_i be the edge chosen by the algorithm at step i.

Claim: $\{e_1, \dots, e_i\} \subset S_{opt}$.

Optimality

- **Assume:** All weights w_{uv} are distinct.
- Let S_{opt} be the optimal set of edges.
- Let e_i be the edge chosen by the algorithm at step i.

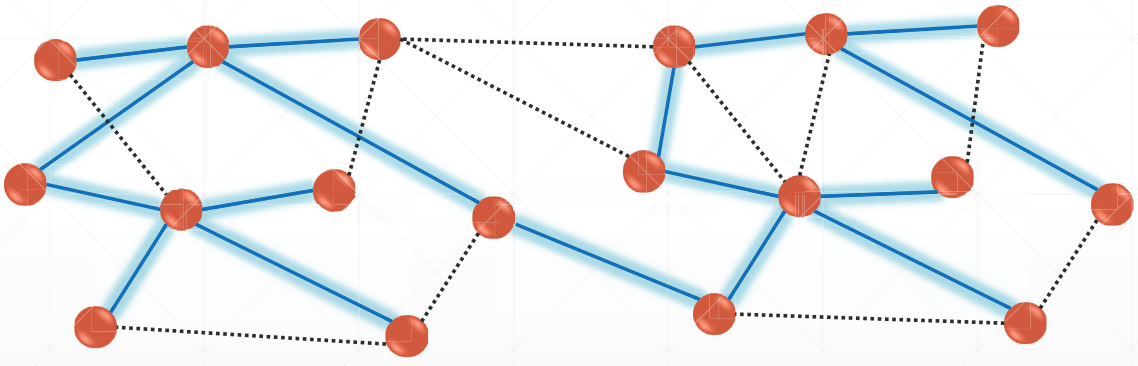
Claim: $\{e_1, \dots, e_i\} \subset S_{opt}$.

If this claim is correct, then $S \subset S_{opt}$ and $|S_{opt}| = |S| = n - 1$

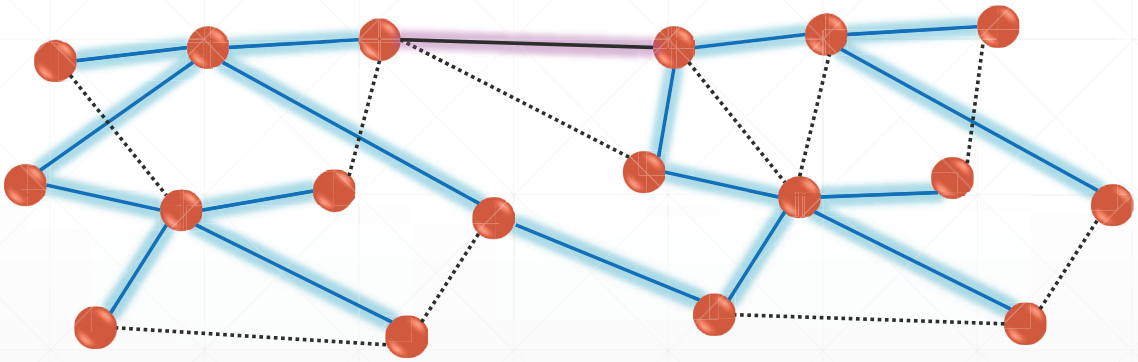
Proof by induction

- **Inductive hypothesis:** $\{e_1, \dots, e_i\} \subset S_{opt}$.
- **Base case:** For $i = 0$, the claim holds.
- **Inductive step:** Assume that inductive hypothesis holds for $i - 1$. We prove it for i.

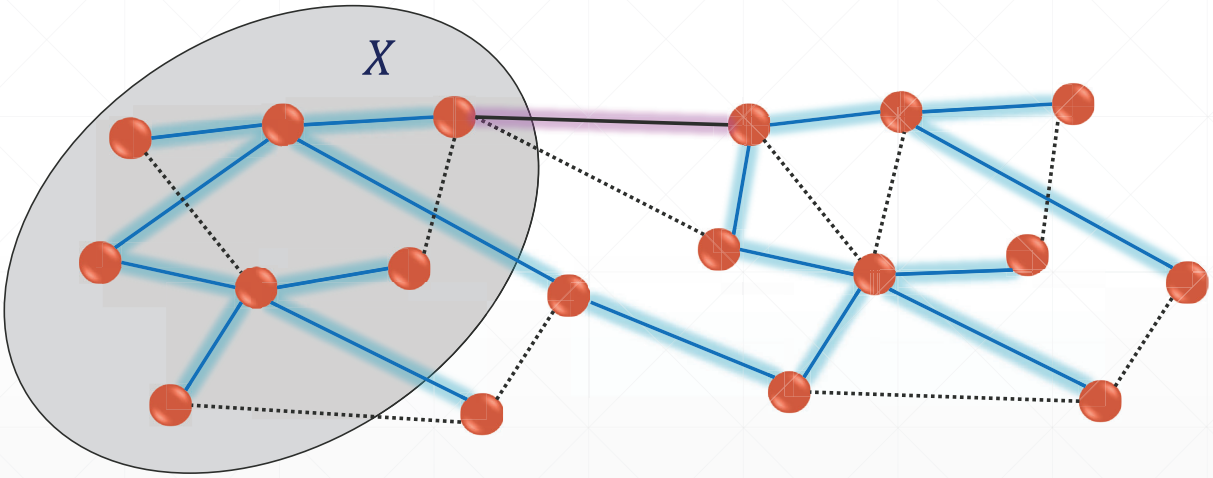
Optimal Solution



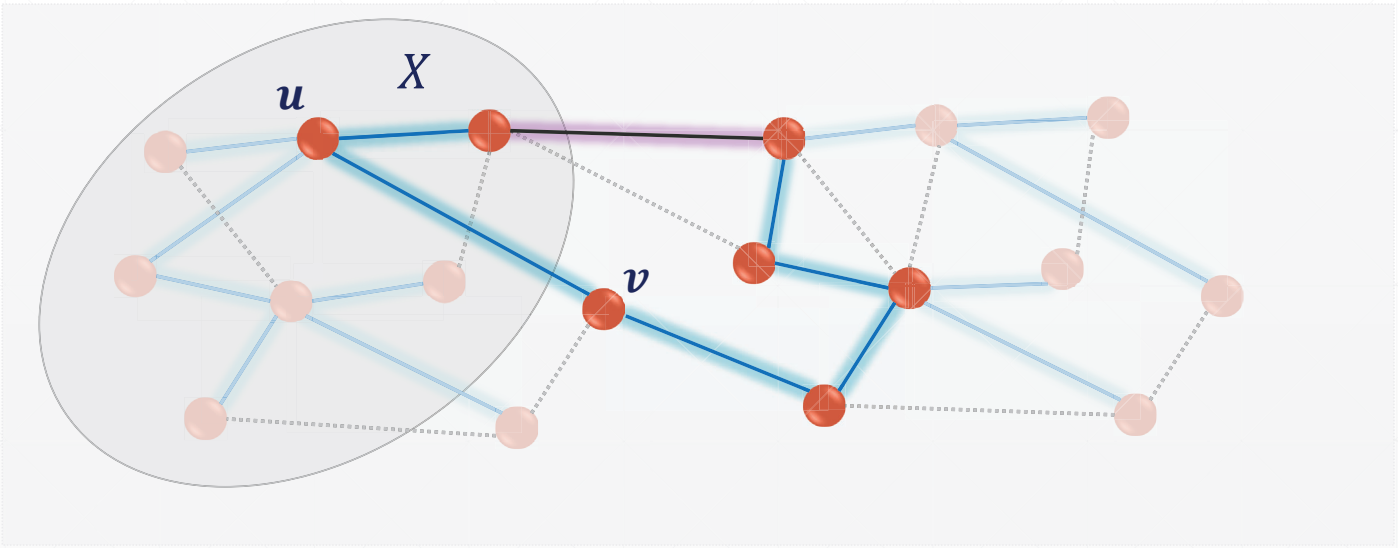
Optimal Solution + e_i



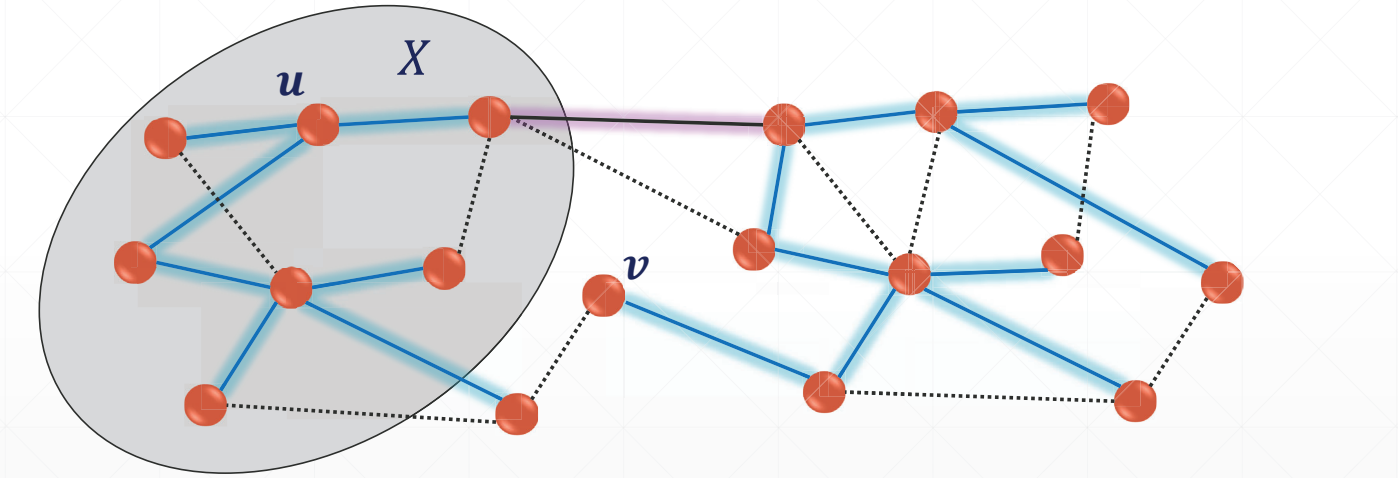
Optimal Solution + e_i at Step i



Optimal Solution + e_i at Step i



Optimal Solution + e_i − (u, v)



The new tree is connected & has $(n - 1)$ edges!

New Solution

- $S_{new} = S_{opt} + e_i - (u, v)$
- Edges in S_{new} form a spanning tree.
- $cost(S_{new}) = cost(S_{opt}) + w_{e_i} - w_{uv} < cost(S_{opt})$.

➢ We get a contradiction! Hence, $e_i \in S_{opt}$.

➢ This finishes the proof of correctness.

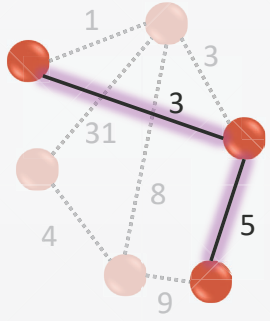
➢ Let's analyze the running time.

Implementation Details

Pick an arbitrary vertex $u \in V$.
Let $X = \{r\}$; $S = \emptyset$;

while ($X \neq V$):
  Get the lightest edge $(u, v) \in E$ leaving set X.
  That is, $u \in X$ and $v \notin X$.
  Add v to X and (u, v) to S.

Return S



Easy to implement in $O(nm)$ time.

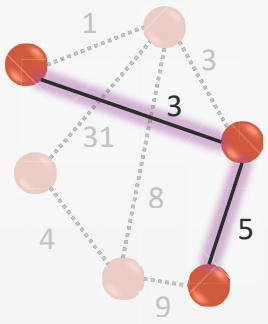
Implementation Details

Pick an arbitrary vertex $u \in V$.
Let $X = \{r\}$; $S = \emptyset$;
Priority queue Q – insert all edges incident on r.

while ($X \neq V$):
  Remove the lightest edge (u, v) from Q.

  if $u, v \in X$, discard (u, v).
  else
    Add v to X; (u, v) to S;
    Insert all edges (v, v') incident on v in Q

Return S



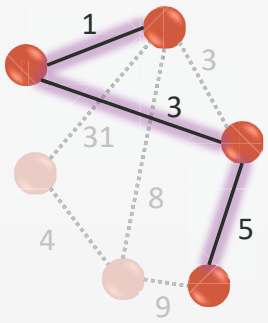
Implementation Details

Pick an arbitrary vertex $u \in V$.
Let $X = \{r\}$; $S = \emptyset$;
Priority queue Q – insert all edges incident on r.

while ($X \neq V$):
  Remove the lightest edge (u, v) from Q.

  if $u, v \in X$, discard (u, v).
  else
    Add v to X; (u, v) to S;
    Insert all edges (v, v') incident on v in Q

Return S



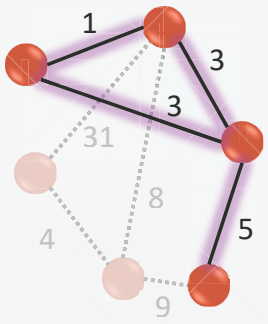
Implementation Details

Pick an arbitrary vertex $u \in V$.
Let $X = \{r\}$; $S = \emptyset$;
Priority queue Q – insert all edges incident on r.

while ($X \neq V$):
  Remove the lightest edge (u, v) from Q.

  if $u, v \in X$, discard (u, v).
  else
    Add v to X; (u, v) to S;
    Insert all edges (v, v') incident on v in Q

Return S



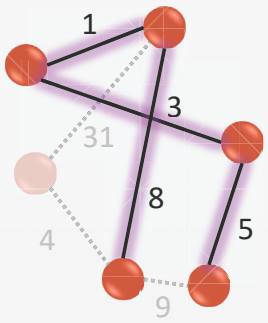
Implementation Details

Pick an arbitrary vertex $u \in V$.
Let $X = \{r\}$; $S = \emptyset$;
Priority queue Q – insert all edges incident on r.

while ($X \neq V$):
  Remove the lightest edge (u, v) from Q.

  if $u, v \in X$, discard (u, v).
  else
    Add v to X; (u, v) to S;
    Insert all edges (v, v') incident on v in Q

Return S



Running Time

- We add every edge to Q at most 2 times.
- We remove every edge from Q at most 2 times.
- So, the total number of operations is at most:

$$4m \cdot O(\log m) = m \cdot O(\log m)$$

where m is the number of edges in the graph.

Conclusion

- Discussed Kruskal's and Prim's algorithms for finding MST.
- Proved correctness of Prim's algorithms.
- Analyzed the running time of Prim's algorithm.