Greedy Algorithms

Greedy Technique

- What is it?
  - Quick and dirty approach to solving optimization problems
  - Not necessarily optimal
- Problems explored
  - Coin changing problem
  - Minimum spanning tree algorithms
  - Dijkstra's algorithm for single source shortest paths
  - Knapsack problem

Optimization problems

- An **optimization** problem:
  - For a problem to solve, there are an **objective function** and a set of **constraints**
  - Find a **feasible** solution for the given instance for which the objective function has an optimal value (either maximum or minimum depending on the problem being solved)
    - A **feasible** solution satisfies the problem's constraints
    - The **constraints** specify the limitations on the required solutions

- **An example in the next slide**

Coin changing problem

- Problem: Return correct change using a minimum number of US coins.
- Greedy choice: Pick the coin with the highest value
- A greedy solution: next slide
- The amount owed = 37 cents.
  - The change is: 1 quarter, 1 dime, 2 cents.
- Solution is optimal when US coins are used. Why is it optimal?

A greedy solution:

```
Input: Set of coins, amount-owed
change = {}
while (more coin-sizes && valueof(change) < amount-owed)
   {
        // Selection
        Choose the largest remaining coin

        // feasibility check
         If (adding the coin makes the valueof(change) exceed the amount-owed )
             then reject the coin
         else add coin to change

        // check if solved
        if ( valueof(change) == amount-owed)
        then return change
       }

return "failed to compute change"
```

Elements of the Greedy Strategy

- Cast problem as one in which you make a greedy choice and are left with one sub-problem to solve.

- Cost-benefit analysis for a greedy choice, e.g., the number of the coins used vs. the remaining amount of the change you owe

A greedy solution is not always optimal!

- Reconstider the Coin Changing problem
  - Suppose you live in Alice's Wonderland where you have 12 cent coins in addition to US coins
  - Suppose you owe 16 cents
    - The greedy solution chooses a 12 cent coin and four 1 cent coins → 5 coins
    - An optimal solution is one dime, one nickel, and one cent → 3 coins

- Some greedy algorithms provides optimal solutions
  - A proof is needed
  - A counter example shows that a greedy algorithm does not provide an optimal solution

Greedy vs. Dynamic Programming

- Greedy algorithms make **good local choices** in the hope that they result in an optimal solution
  - In each step, just make a choice that seems best at the moment
  - Solve the remaining sub-problem in the next step by making another greedy choice

➔ Produces a feasible solution but does not necessarily end up with an optimal solution

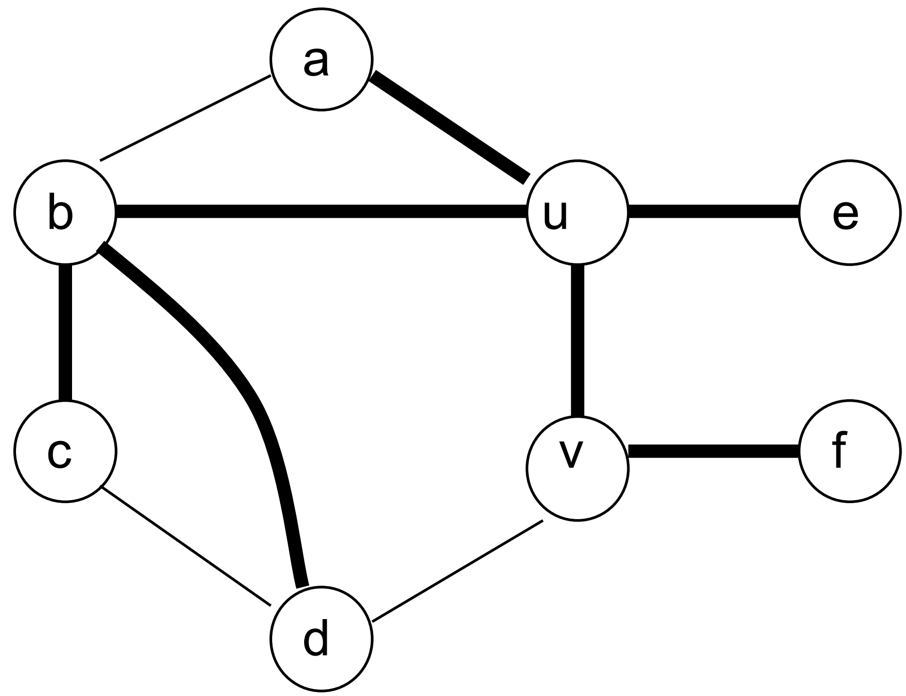
- A greedy algorithm never reconsiders its choices
    - Dynamic programming is exhaustive, and makes decisions based on all the previous decisions, potentially reconsidering previous choices (look up previous solutions)
- In an earlier lecture on dynamic programming, we saw both greedy and dynamic programming approaches for finding an Optimal BST (Binary Search Tree)
    - A greedy approach locating the highest probability node at the root or trying to minimize the tree depth does not necessarily gives you an optimal solution

Greedy Minimum Spanning Tree Algorithms

- Prim's Algorithm
- Kruskal's Algorithm

What is A Spanning Tree?

- A *spanning* tree for an undirected graph G=(V,E) is a subgraph of G that is a tree and contains all the vertices of G



- Can a graph have more than one spanning tree?

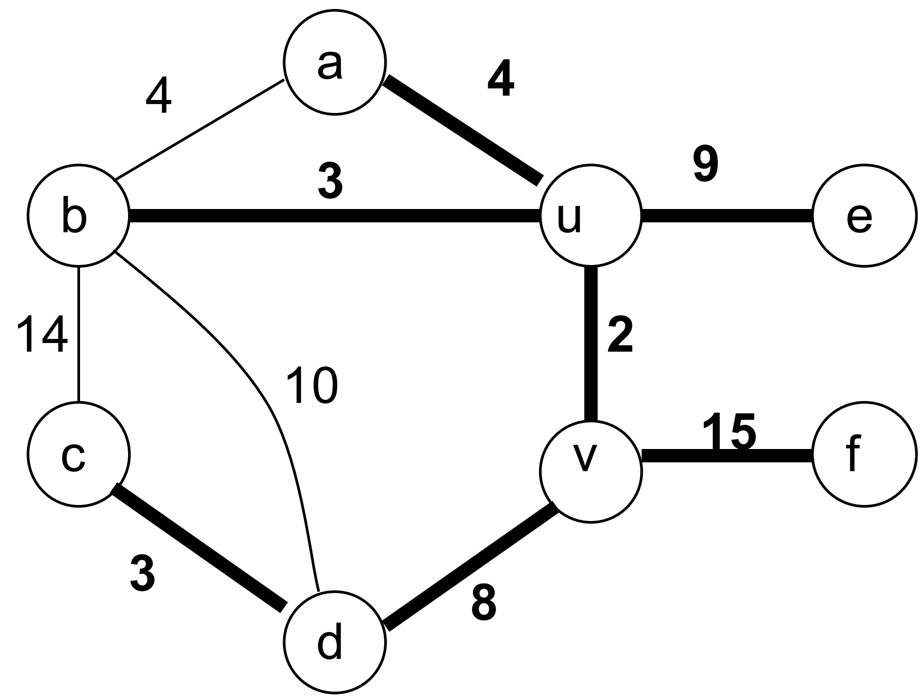
- Can a disconnected graph have a spanning tree?

Minimum Spanning Tree

- The *weight* of a subgraph is the sum of the weights of its edges.

- A *minimum spanning tree* for a weighted graph is a spanning tree with the minimum weight

- Can a graph have more than one minimum spanning tree?



MST T: $w(T) = \sum_{(u,v) \in T} w(u,v)$ is minimized

Example of a Problem that translates into a MST

The Problem

- Several pins of an electronic circuit must be connected using the least amount of wire.

Modeling the Problem

- The graph is a complete, undirected graph $G = (V, E ,W)$, where V is the set of pins, E is the set of all possible interconnections between the pairs of pins and $w(e)$ is the length of the wire needed to connect the pair of vertices.
- Find a minimum spanning tree.

Greedy Choice

We will show two ways to build a minimum spanning tree.

- *Prim's algorithm*
  - *A MST can be grown from the current spanning tree by adding the nearest vertex and the edge connecting the nearest vertex to the MST*

- *Kruskal's algorithm*
  - *A MST can be grown from a forest of spanning trees by adding the smallest edge connecting two spanning trees*

Notation

- Tree-vertices: in the tree constructed so far
- Non-tree vertices: rest of vertices

Prim's Selection rule

- Select the minimum weight edge between a tree-node and a non-tree node and add it to the tree

Key idea of Prim's algorithm

Select a vertex to be a tree-node

while (there are non-tree vertices)

{

    if (there is no edge connecting a tree node with a non-tree node)

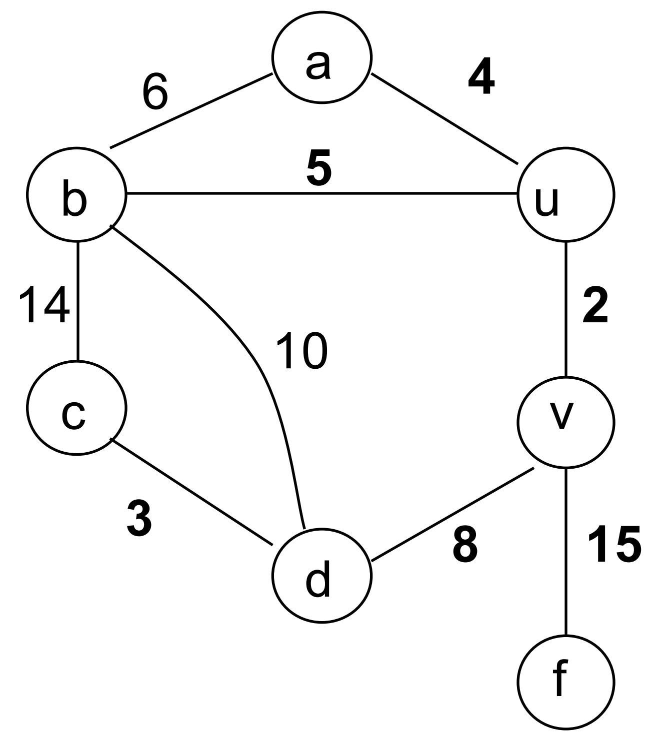
    return "no spanning tree"

    select an edge of minimum weight between a tree node and a non-tree node

    add the selected edge and its new vertex to the tree

}

return tree



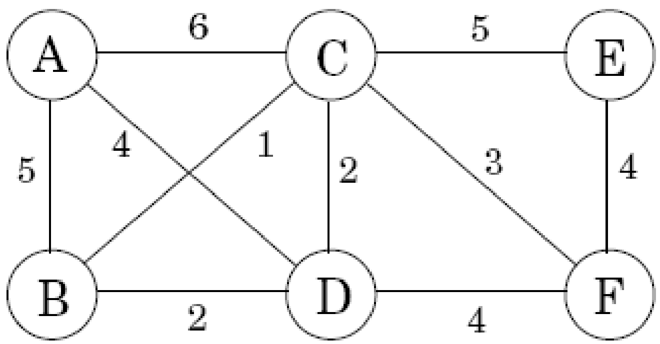
Prim's algorithm

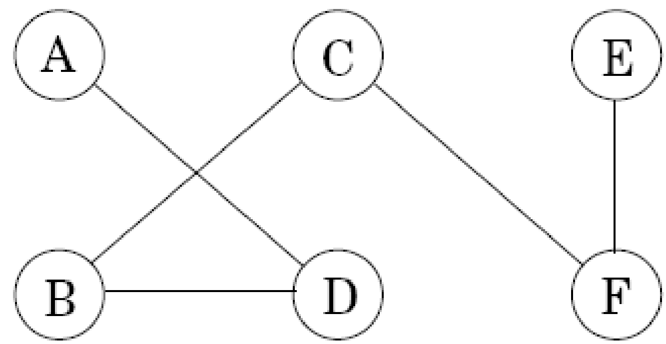
source: **Algorithms** by Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani, McGraw Hill, 2006

```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
   cost(u) = ∞
   prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)     (priority queue, using cost-values as keys)
while H is not empty:
   v = deletemin(H)
   for each {v, z} ∈ E:
      if cost(z) > w(v, z):
         cost(z) = w(v, z)
         prev(z) = v
         decreasekey(H, z)
```

w[i][j] = 0 if i=j;
edge weight if there is an edge (i,j); or
infinity if no edge (i, j) exists

Example





S: set of tree vertices

Set S	A	B	C	D	E	F
$\{\}$	$0/\text{nil}$	∞/nil	∞/nil	∞/nil	∞/nil	∞/nil
A		$5/A$	$6/A$	$4/A$	∞/nil	∞/nil
A, D		$2/D$	$2/D$		∞/nil	$4/D$
A, D, B			$1/B$		∞/nil	$4/D$
A, D, B, C					$5/C$	$3/C$
A, D, B, C, F					$4/F$	

cost/prev

Implementation

- Queue can be implemented as an array or heap
- Time complexity changes based on the implementation of the queue
  - More details next

Prim's algorithm

Source: A free online algorithms book available at http://www.cs.berkeley.edu/~vazirani/algorithms.html

```
procedure prim(G, w)
Input:    A connected undirected graph G = (V, E) with edge weights w_e
Output:   A minimum spanning tree defined by the array prev

for all u ∈ V:
    cost(u) = ∞
    prev(u) = nil
Pick any initial node u_0
cost(u_0) = 0

H = makequeue(V)    (priority queue, using cost-values as keys)
while H is not empty:
    v = deletemin(H)
    for each {v, z} ∈ E:
        if cost(z) > w(v, z):
            cost(z) = w(v, z)
            prev(z) = v
            decreasekey(H, z)
```

Θ(V)

Θ(1)

Θ(V)

So, the total time complexity is O(V*deletemin) + O(E * decreasekey)

Note that decreasekey is executed maximum E times to find an MST

Prim's algorithm

- Time complexity
  - O(V*deletemin) + O(E * decreasekey)
  - Array: deletemin is O(V) and decreaskey is O(1) → $O(V^2)$
  - Heap: deletemin is O(lg V) and decrease key is O(lg V) → O(E lg V)
  - Which is better?

Lemma 1

- Let $G = (V, E)$ be a connected, weighted undirected graph. Let T be a promising subset of E. Let Y be the set of vertices connected by the edges in T.
- **<u>If e is a minimum weight edge that connects a vertex in Y to a vertex in $V - Y$, then $T \cup \{ e \}$ is promising.</u>**

Note: A feasible set is *promising* if it can be extended to produce not only a solution but an optimal solution.

In this algorithm, a feasible set of edges is *promising* if it is a subset of a MST for the connected graph G.

Outline of Proof of Correctness of Lemma 1

- T is the promising subset and e is the minimum cost edge of Lemma 1
- Let T' be the MST such that $T \subset T'$
- We will show that if $e \notin T'$ then there must be another MST T'' such that $T \cup \{e\} \subseteq T''$.

Proof has 4 stages (shown in the following slides):

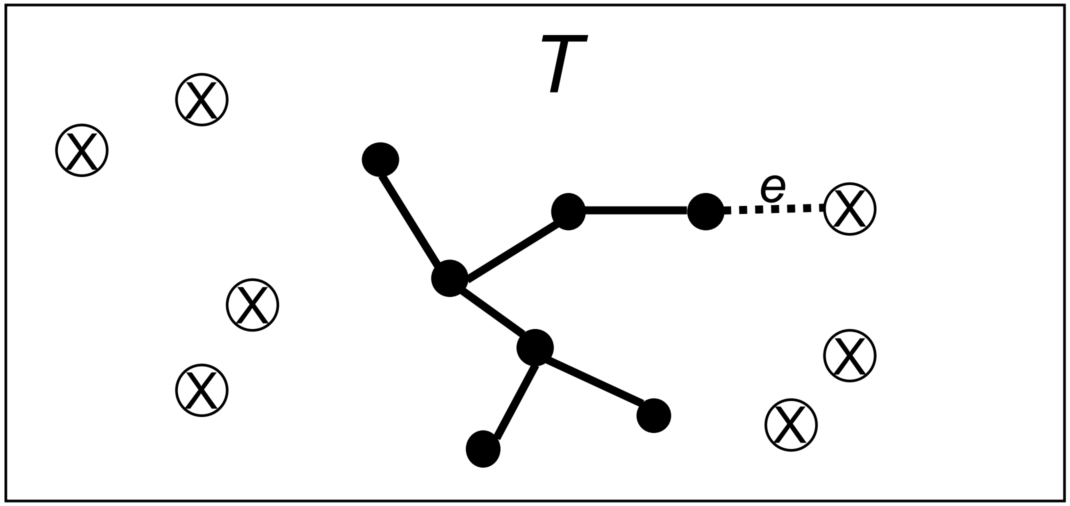
1. Adding e to T', creates a cycle in $T' \cup \{e\}$.
2. Cycle contains another edge $e' \in T'$ but $e' \notin T$
3. $T'' = T' \cup \{e\} - \{e'\}$ is a spanning tree
4. T'' is a MST

Lemma 1

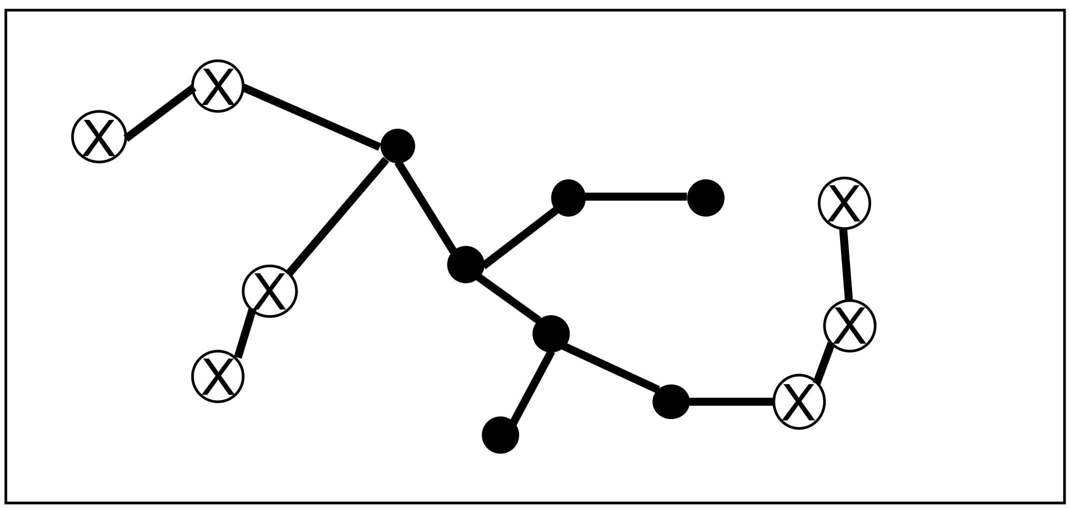
The Promising Set of Edges Selected by Prim

$\bullet \in Y$

$\otimes \in V - Y$



MST T' but $e \notin T'$

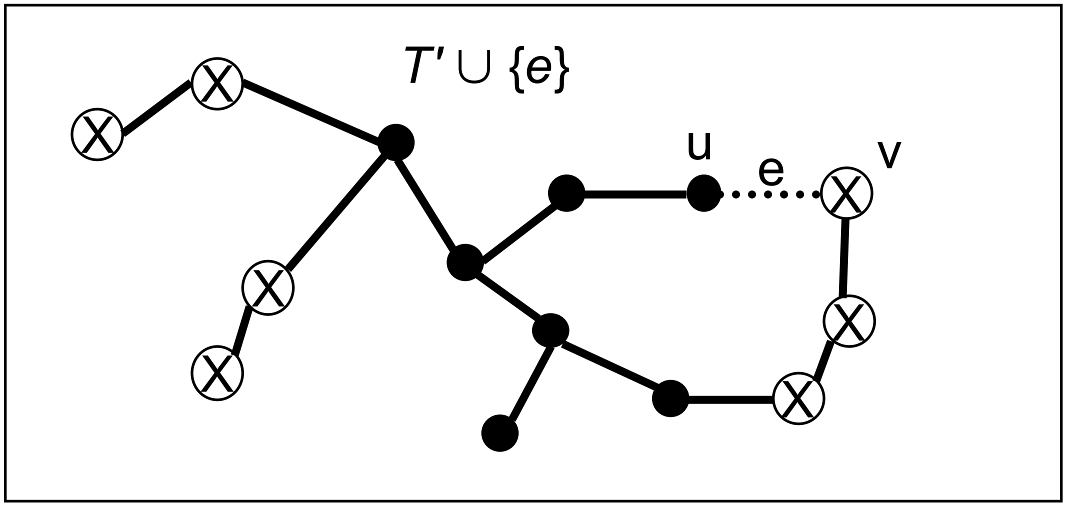


Lemma 1

Since T' is a spanning tree, adding e creates a cycle.

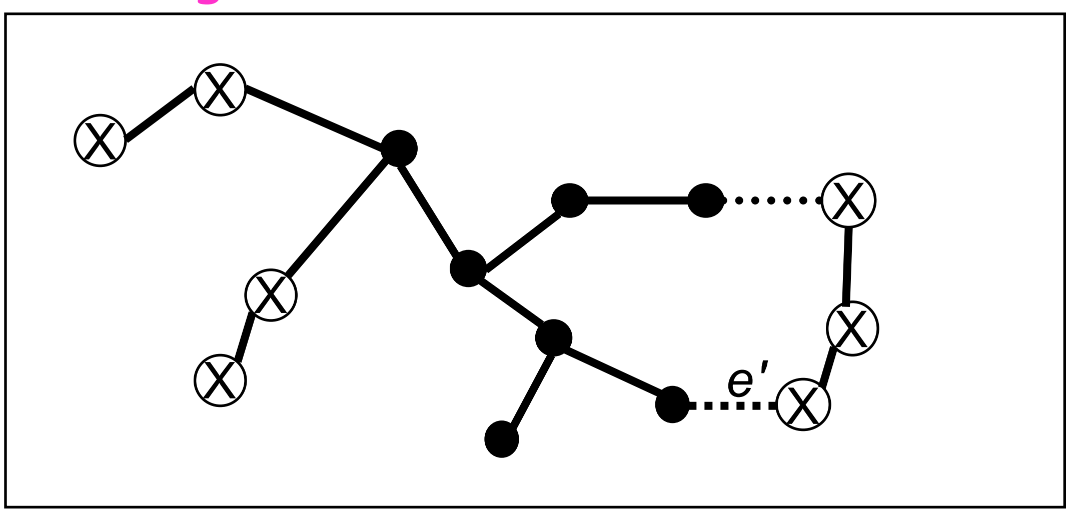
$\bullet \in Y$

$\otimes \in V - Y$



Stage 1

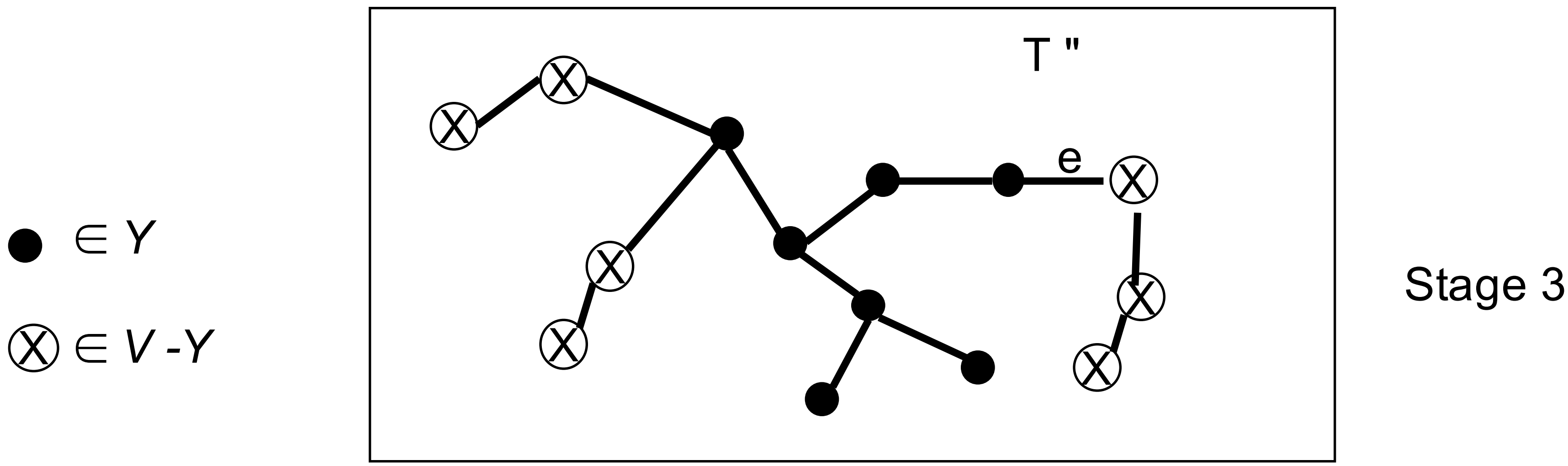
In T' there is a path from $u \in Y$ to $v \in V - Y$. Therefore the path must include another edge e' with one vertex in Y and the other in $V - Y$.



Stage 2

Lemma 1

- If we remove *e'* from *T'* ∪ **{** *e* } the cycle disappears.
- *T "* = *T '* ∪ { *e* } - {*e'*} is connected.



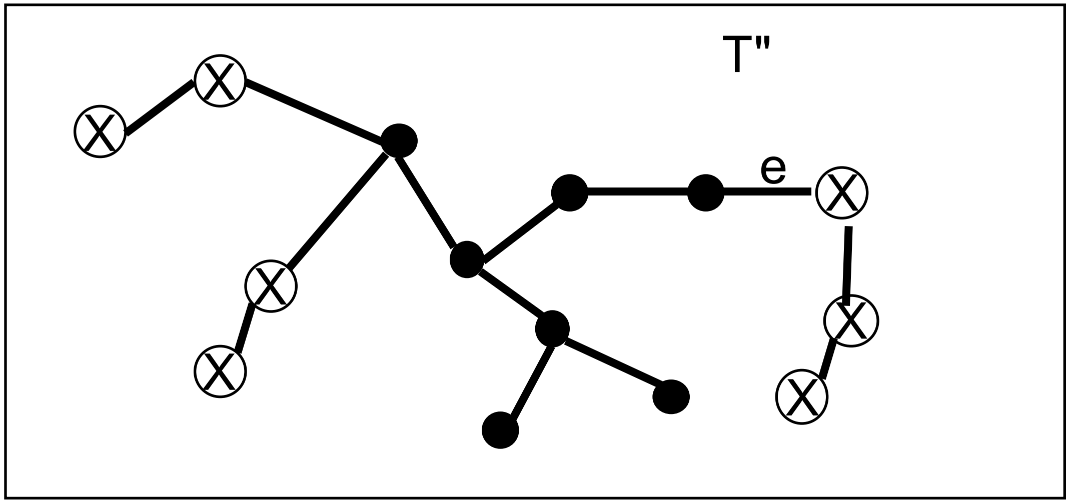
$w(e) \leq w(e')$ due to the way *Prim* picks the next edge

Lemma 1

- $w(e) \leq w(e')$ because of the way *Prim* picks the next edge
- The weight of T'', $w(T'') = w(T') + w(e) - w(e') \leq w(T')$.
- But $w(T') \leq w(T'')$ because T' is a MST.
- So $w(T') = w(T'')$ and T'' is a MST

● $\in Y$

⊗ $\in V - Y$



Stage 4

Conclusion $T \cup \{ e \}$ **is promising**

Theorem: Prim's Algorithm always produces a minimum spanning tree.

Proof by induction on the set T of promising edges.

1. Base case: Initially, $T = \emptyset$ is promising.

2. Induction hypothesis: The current set of edges T selected by Prim is promising.

3. Induction step: After Prim adds the edge e, $T \cup \{ e \}$ is promising.

   Proof: $T \cup \{ e \}$ is promising by Lemma 1.

Conclusion: When G is connected, *T* produced by Prim is a MST.