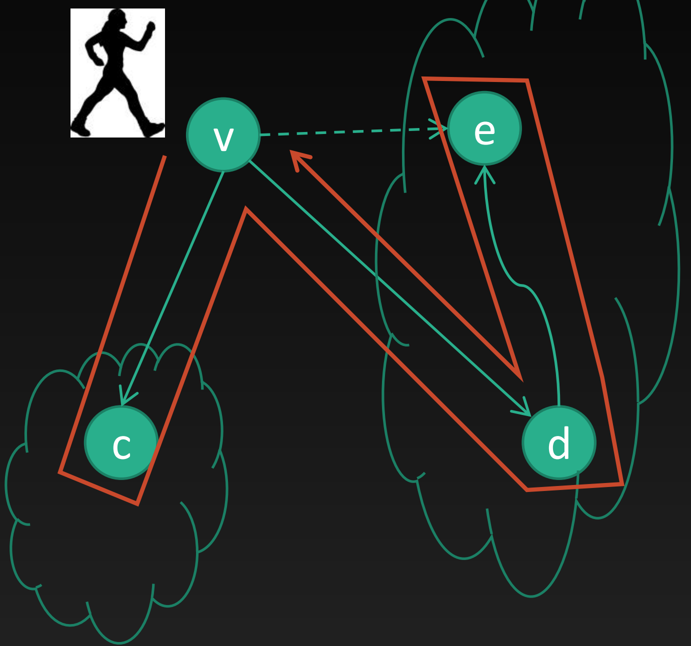
CS 4800: Algorithms & Data

Lecture 15

March 2, 2018

(Depth-First) Search in Graph

- Search(vertex v)
  - $explored[v] \leftarrow 1$
  - For $(v, w) \in E$
    - If $explored[w] \; = \; 0$ then
      - $parent[w] \leftarrow v$
      - search(w)
  - post-visit(v)

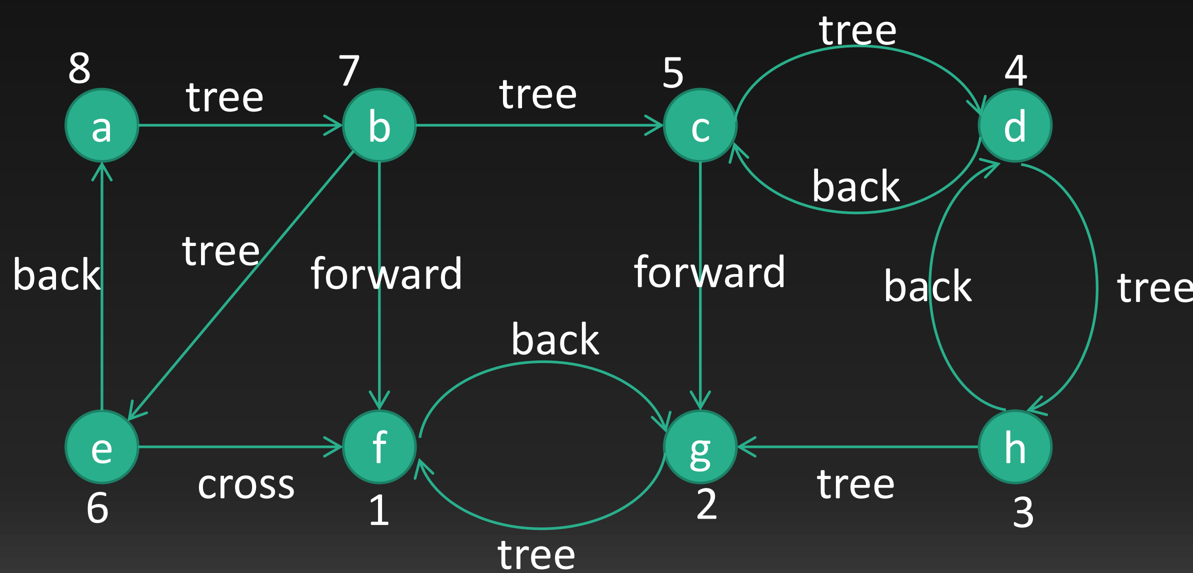


v	v	v	v	v	v	v
	c		d	d	d	
				e		

Observations

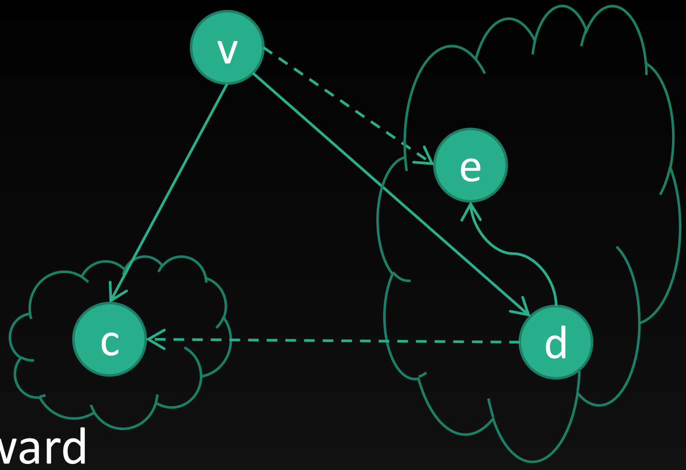
```
Search(vertex v)
    explored[v] ← 1
    For (v,w) ∈ E
        If explored[w] = 0 then
            parent[w] ← v
            search(w)
    post-visit(v)
```

- If $(u, v) \in E$ then $postorder[u] < postorder[v] \leftrightarrow (u, v)$ is backward



Vertex	Post-order
a	8
b	7
c	5
d	4
e	6
f	1
g	2
h	3

Observations



- If $(u, v) \in E$ then $postorder[u] < postorder[v] \leftrightarrow (u, v)$ is backward

Proof:

- search(v) finishes after searches for its children finish
  - If (u,v) is tree edge then postorder[u] > postorder[v]
  - If (u,v) is forward edge then postorder[u] > postorder[v]
  - If (u,v) is backward then postorder[u] < postorder[v]
- If $postorder[u] < postorder[v]$ then search(u) finishes before search(v).
- Thus, search(v) is not called by search(u)
- explored[v]=1 when running search(u) i.e. search(v) started before search(u)
- Search(v) starts before and ends after search(u)
  - Can only happen for backward edge
  - Cannot happen for cross edge

Topological sort

- Directed graph G=(V,E)
- Scheduling
  - Vertices: tasks
  - Edges: Precedence constraints: edge (u,v) implies u must finish before v can start
- Compiling large programs (e.g. in Go)
  - Vertices: modules
  - Edges: dependencies: edge (u,v) implies module v depends on module u
- Goal: figure out an ordering to satisfy all precedence constraints
- Observation: impossible if there are cyclic constraints
- Directed acyclic graph (DAG)

Topological sort

- Claim: Scheduling by decreasing postorder satisfies all constraints.

Proof.

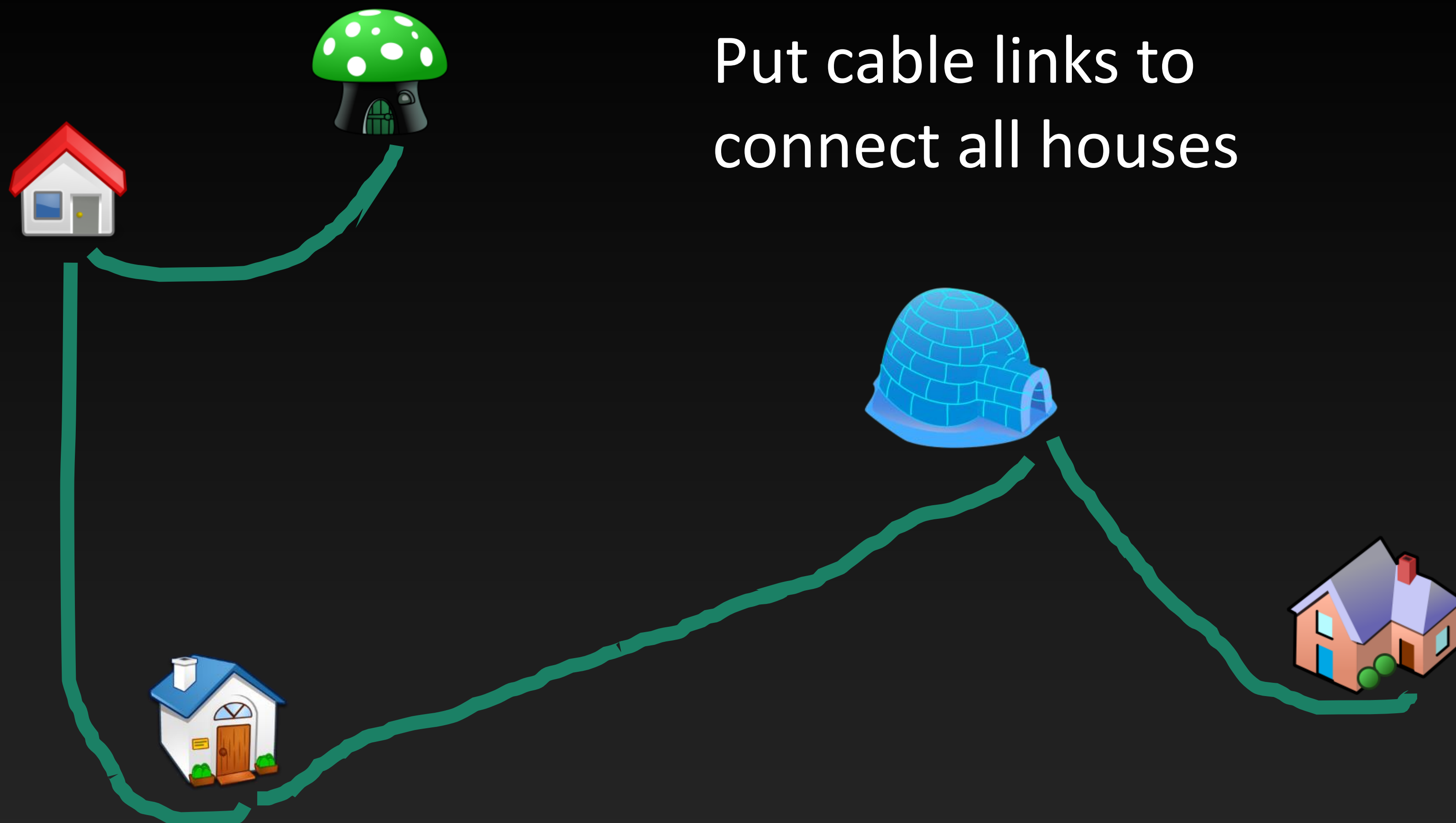
If G is acyclic then there is no backward edge.

Thus, for all edge (u,v), we have postorder[u] > postorder[v].

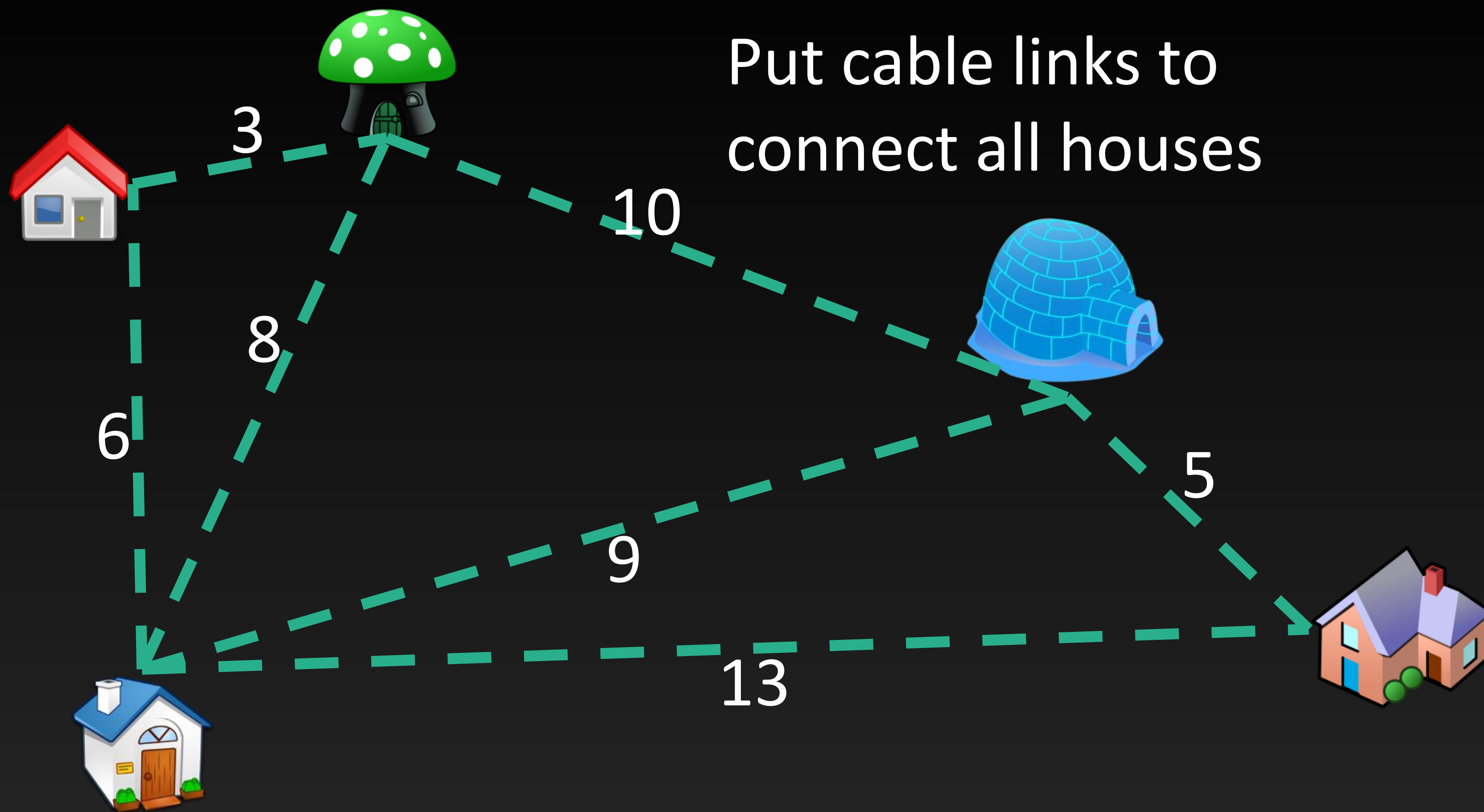
If schedule by decreasing postorder, when v is processed, all prerequisites for v are already processed.

Minimum spanning trees

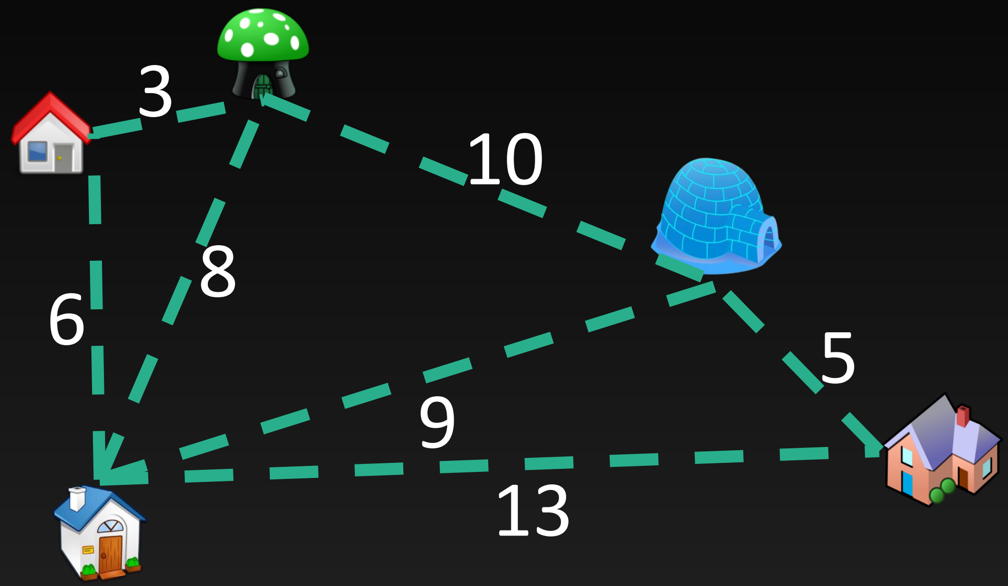
Put cable links to connect all houses



Put cable links to connect all houses



Minimum spanning tree (MST)

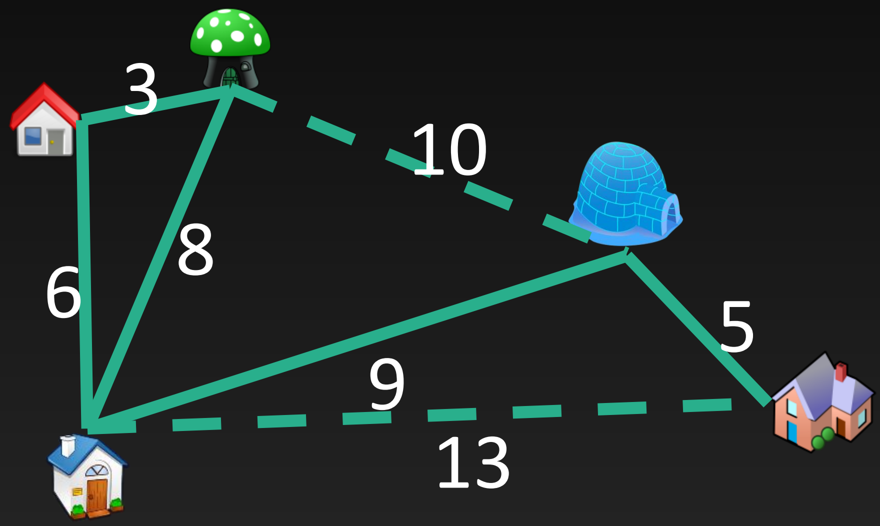


- G = (V, E, w), w positive
- Want a set of edges that connects all V and has minimum cost
- For simplicity, assume all weights are distinct

Minimum spanning tree (MST)

Looking for a set T of edges that

- Connect all vertices
- Has minimum total cost



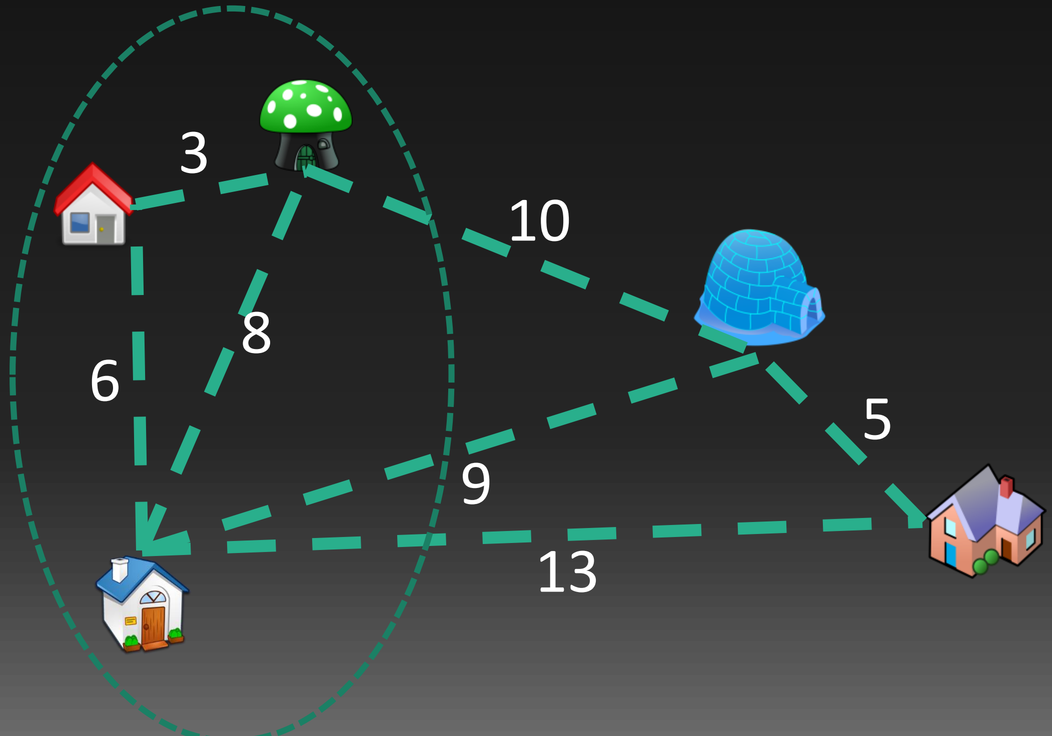
Does T have cycles? **NO**

Can remove 1 cycle edge to reduce cost

How many edges does T have? **V-1**

Blue rule

- Pick a set of nodes S
- Color minimum weight edge in cut induced by S **blue**



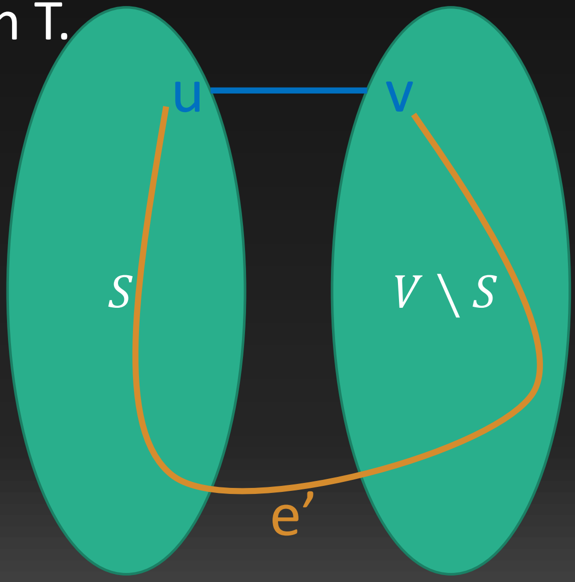
All blue edges are essential

Lemma. MST contains every blue edge.

Proof. Let S be arbitrary subset of nodes and e=(u,v) be the minimum weight edge with one end point in S.

Let T be MST that does not contain e.

T connects u and v so there is a path from u to v in T.



The path must have an edge e’ with exactly one end point in S.

Consider $T' = T \cup \{e\} \setminus \{e'\}$

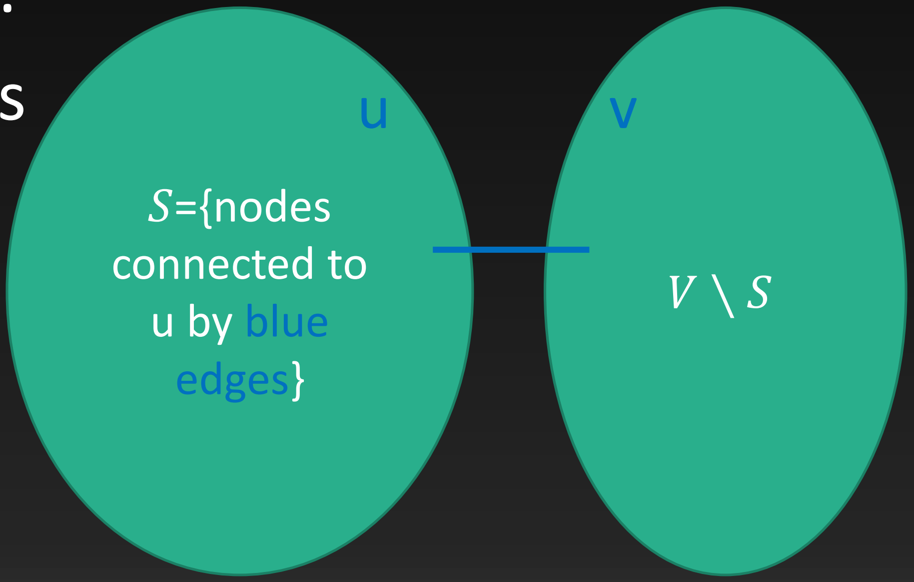
T' connects 2 ends of e’ so T' still connects all nodes.

$$w(T') = w(T) + w(e) - w(e')$$

But $w(e) < w(e')$ so $w(T') < w(T)$ i.e. T cannot be minimum.

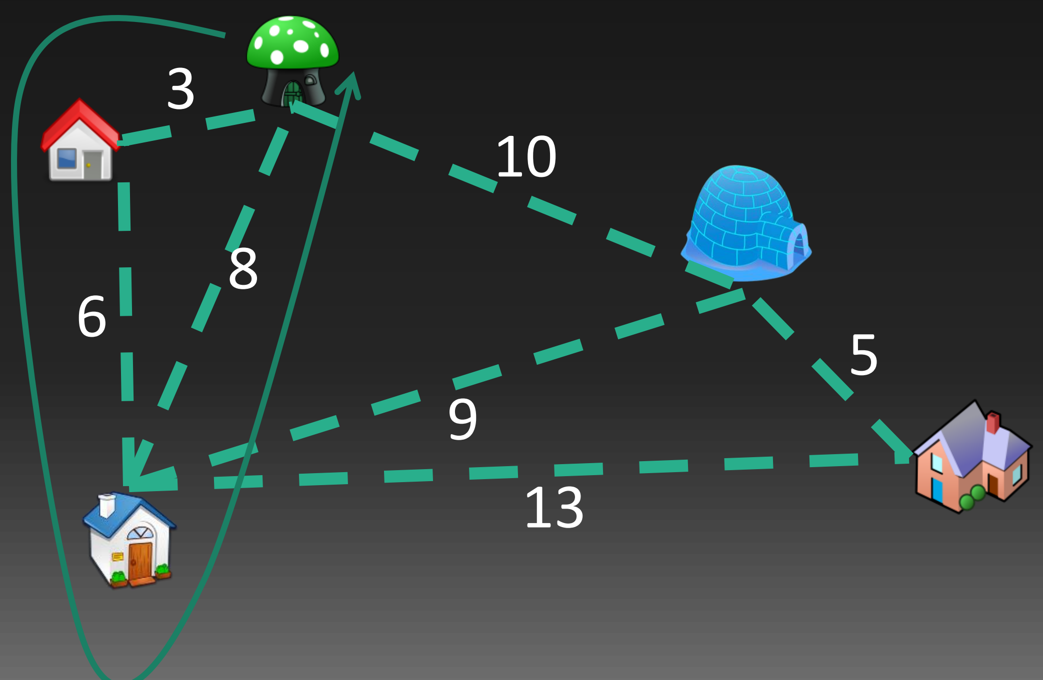
Blue edges connect all nodes

- Assume for contradiction that some u & v are not connected by blue edges.
- Apply blue rule to S yields another blue edge
- MST = set of blue edges



Red rule

- Pick a cycle C
- Color the maximum weight edge in C **red**



All red edges are useless

Lemma. MST contains no red edges.

Proof. Let C be a cycle and e=(u,v) be corresponding red edge.

Let T be MST containing e.

$T \setminus \{e\}$ has 2 connected components S and $V \setminus S$

C is a cycle so $C \setminus \{e\}$ is a path connecting u & v.

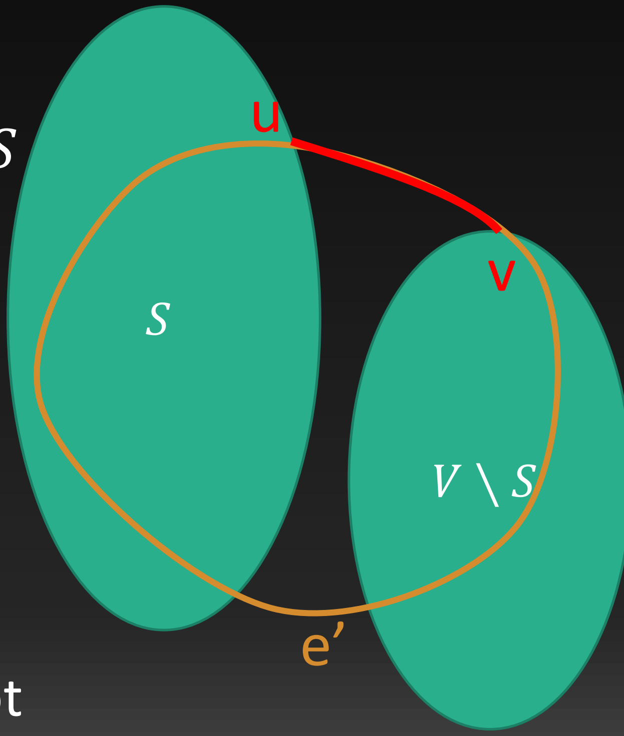
There must be an edge e' on this path with exactly one end point in S.

Consider $T' = T \setminus \{e\} \cup \{e'\}$

e' connects S and $V \setminus S$ so T' connects all nodes.

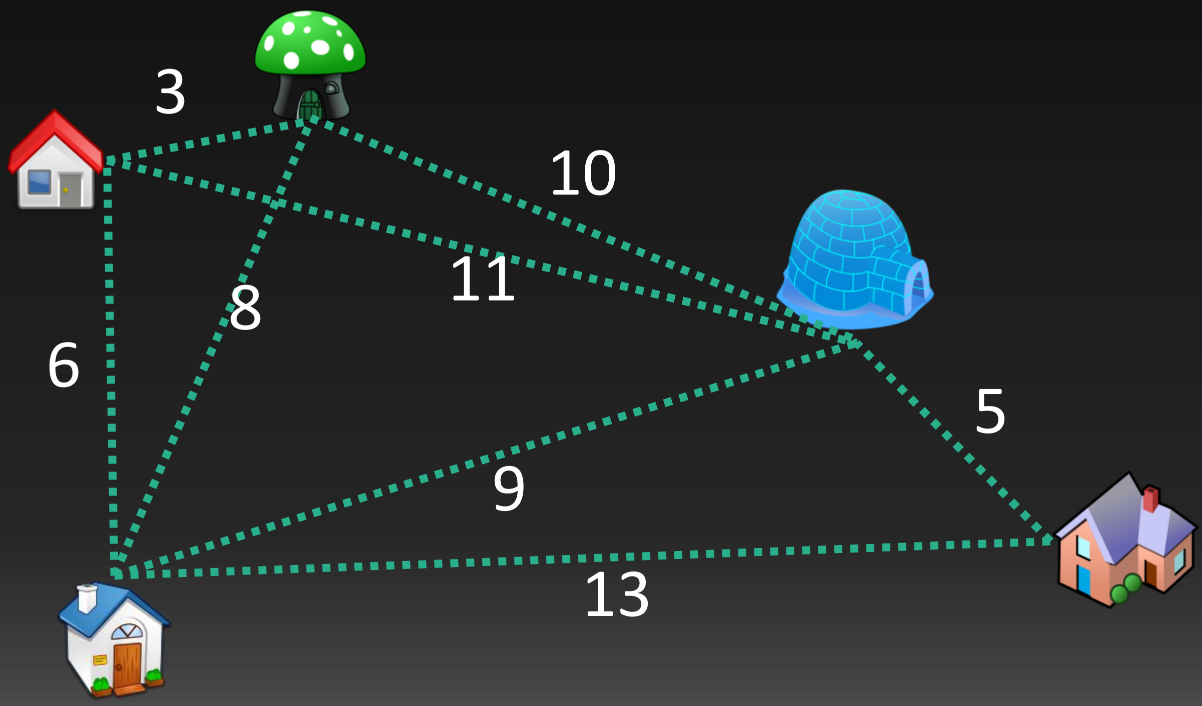
$w(T') = w(T) + w(e') - w(e)$

But $w(e') < w(e)$ so $w(T') < w(T)$ i.e. T cannot be minimum.



Exercise

- Color as many edges red or blue as you can

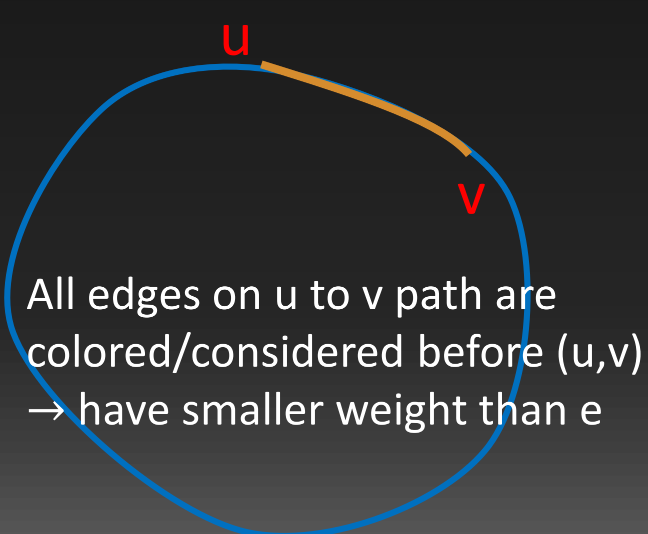


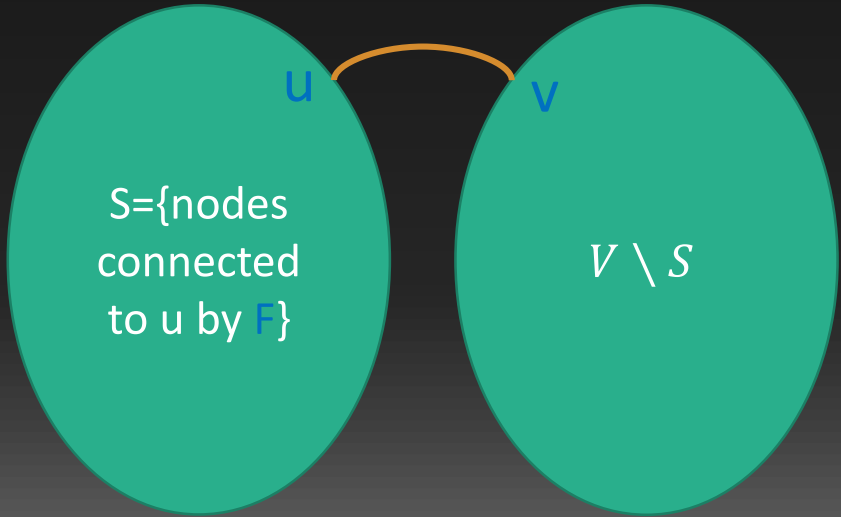
Generic algorithm

- Maintain an acyclic set of blue edges F
- Initially no edge is colored, $F = \emptyset$
- Repeat the following in arbitrary order
  - Consider a cut with no blue edge. Color the minimum weight edge in the cut blue.
  - Consider a cycle with no red edge. Color the maximum weight edge in the cycle red.
  - Terminate when n-1 edges colored blue.

Kruskal’s algorithm

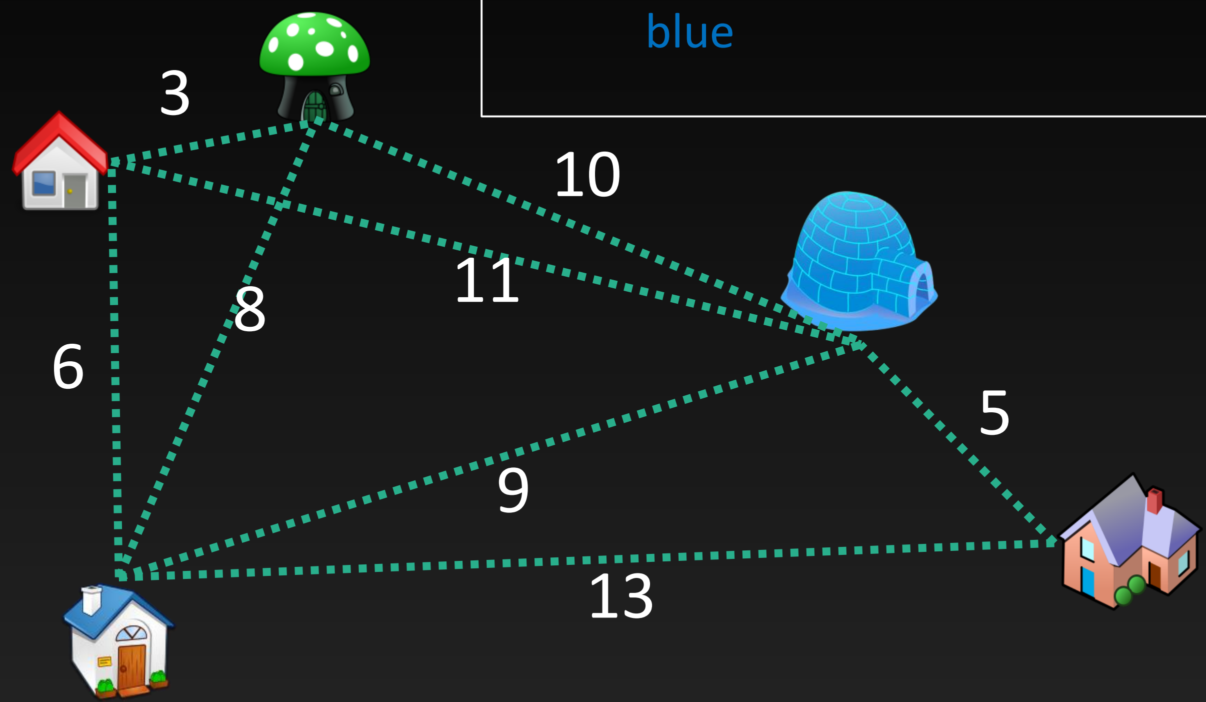
- Consider edges in order of increasing weights
- When considering e=(u,v)
  - If u and v are connected by F, color e red
  - If u and v are not connected by F, color e blue





Example

- Consider edges in order of increasing weights
- When considering e=(u,v)
  - If u and v are connected by F, color e red
  - If u and v are not connected by F, color e blue



3	5	6	8	9	10	11	13
✓	✓	✓	✗	✓	✗	✗	✗