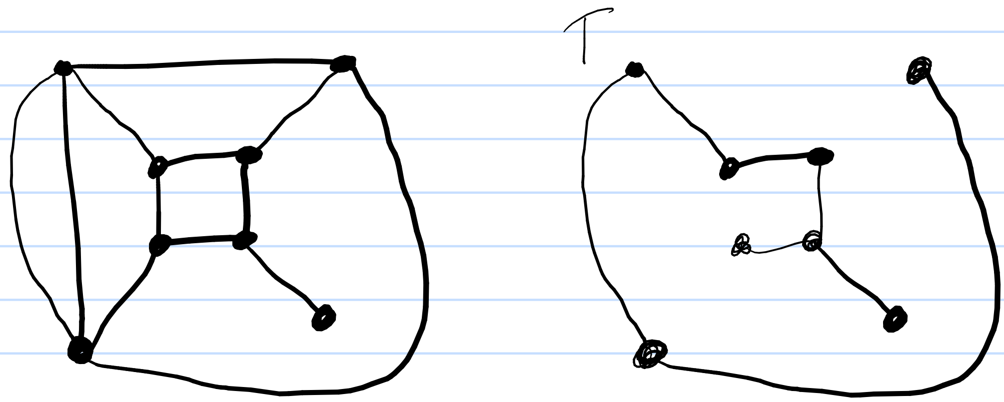
9.3 Spanning Trees

3, 4, 5

Def A subgraph T of G such that T is a tree where $V(T) = V(G)$ is called a <u>spanning tree</u> of G.

Ex G



is a spanning tree of G

Thm A graph G has a spanning tree $\iff$ G is connected

Q: How to find a spanning tree?

Option 1: Breadth-First search:

Algorithm: We input connected G with $V(G) = \{v_1, v_2, \dots, v_n\}$ + output T.
We will build V', E' so that the algorithm outputs $V' = V(T)$, $E' = E(T)$.
We will keep an ordered list S:

```
bfs(V,E): initilize S=(v1), V'={v1}, E'=∅.
    while (true)
        for each x ∈ S,
            for each y ∈ V−V'
                if (x,y) ∈ E(G) and (x,y)∪T is acyclic:
                    add (x,y) to E' and y to V'
        if (no edges added), output T
        Update S to be the children of S
```

Option 2: Depth-First search

Algorithm: We input connected G with $V(G)=\{v_1, v_2, \ldots, v_n\}$ + output T

We will build V', E' so that the algorithm outputs $V' = V(T)$, $E' = E(T)$, and v_1 the root of spanning tree

dfs(V, E): initialize $V' = \{v_1\}$, $E' = \emptyset$, $w = v_1$.

while (true):

while (there is some (w, v) where $T \cup (w, v)$ is acyclic)

add (w, v_k) to E'

add v_k to V'

update $w = v_k$.

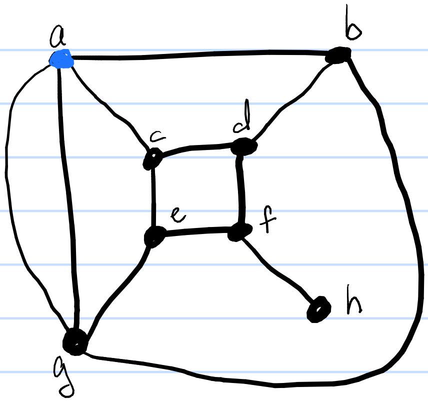
if $(w = v_1)$

output T

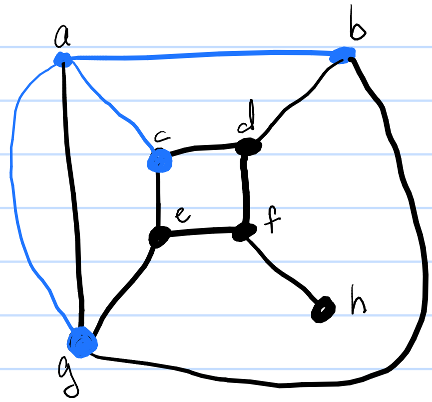
update w to be the parent of w in T (if we have reached a dead end)

Ex) BFS : choose ordering (a, b, c, d, e, f, g, h) of vertices

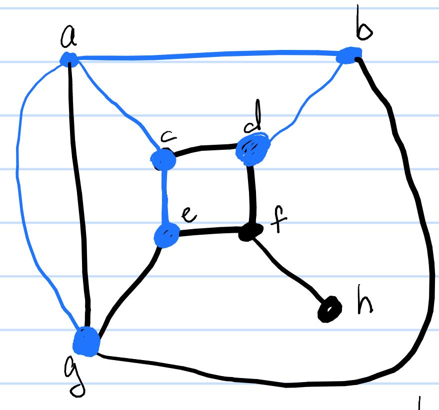
will highlight E', V' in blue as go



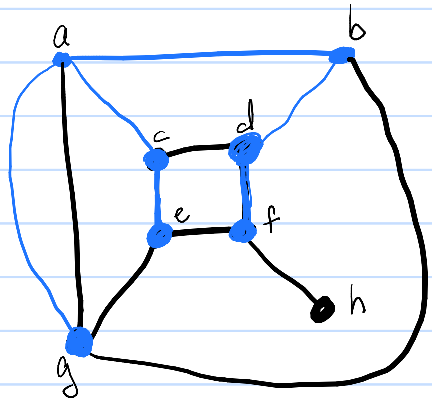
$S = (a)$



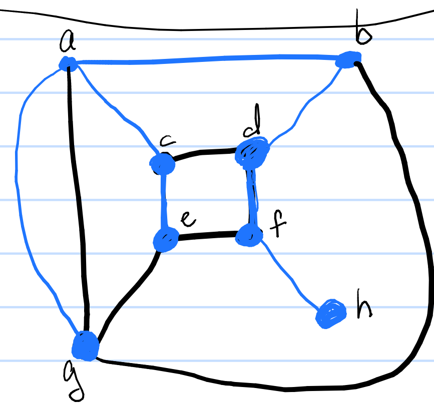
$S = (b, c, g)$



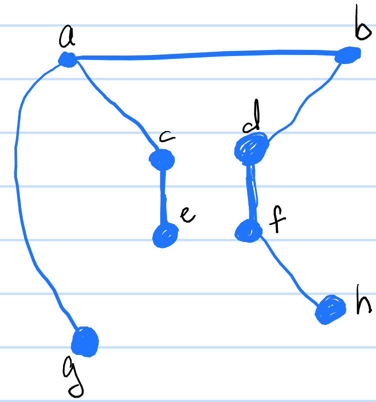
$S = (d, e)$



$S = (f)$



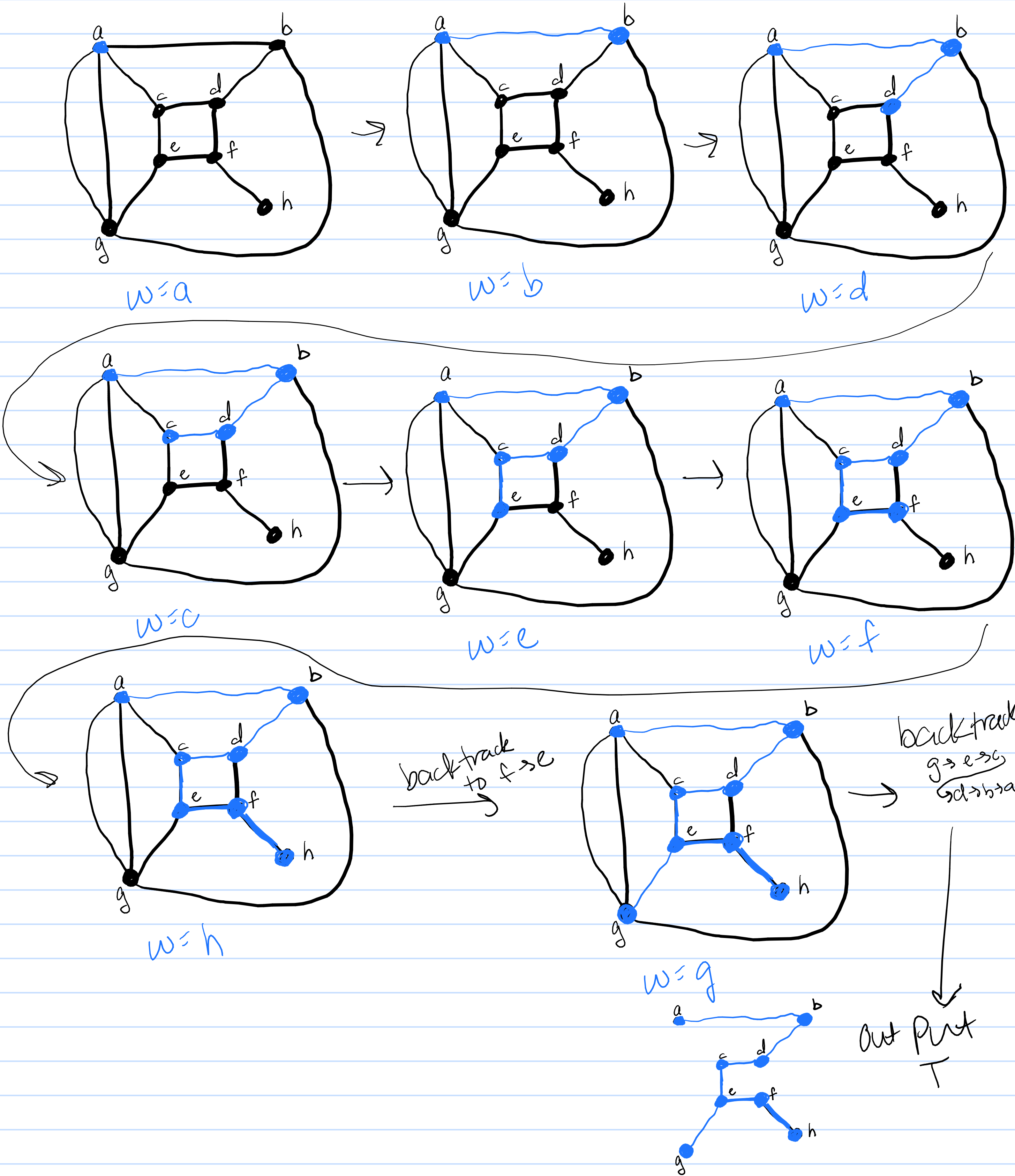
↓ outputs



T

Ex) DFS : choose ordering (a, b, c, d, e, f, g, h) of vertices

will highlight E', V' in blue as go



$\boxed{9.4}$ Minimal Spanning Trees

$\boxed{\text{Def}}$ For a weighted graph G, a spanning tree of G whose sum of weights is minimal is called a <u>minimal spanning tree</u> (MST)

One algorithm to compute a minimal spanning tree is Prim's algorithm

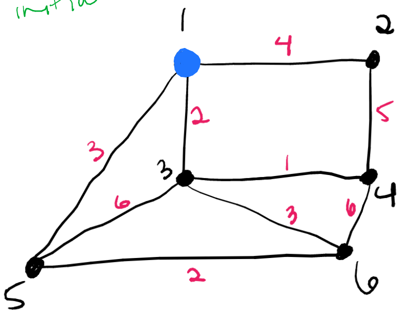
we input G, a connected weighted graph with $V(G) = \{1, \dots, n\}$ & start at $s \in V(G)$.
For $(i,j) \in E(G)$ $w(i,j)$ is the weight of (i,j), otherwise $w(i,j) = \infty$.
This outputs the MST. We update $V(i) = \begin{cases} 1 & i \in V(MST) \\ 0 & i \notin V(MST) \end{cases}$ & E throughout

```
prim(w, n, s):
    initialize all v(i)=0, E=∅.
    update v(s)=1.
    for i=1 to n-1:
        min = ∞
        for j=1 to n:
            if (v(j)=1):
                for k=1 to n:
                    if v(k)=0 and w(j,k)<min:
                        set k to be addable vertex
                        e = (j,k)
                        min = w(j,k)
        Update v(addable vertex) = 1
        Update E = E ∪ e
    return E
```

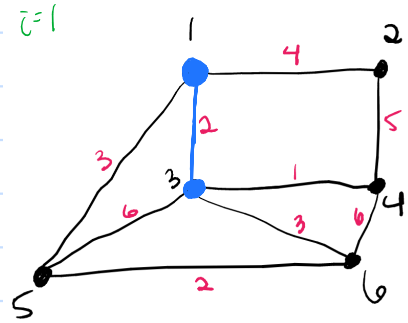
edges in MST

Ex] Take G + start at vertex 1.

initialize

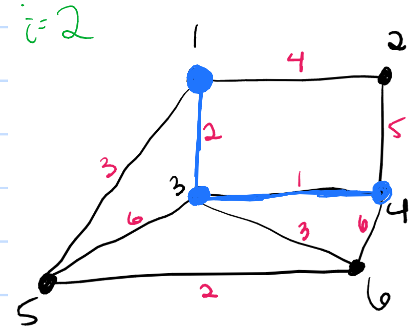


i=1



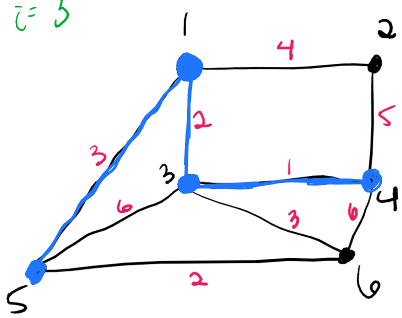
selects 3 to add

i=2



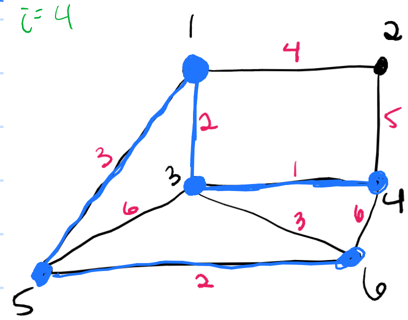
selects 4 to add

i=3



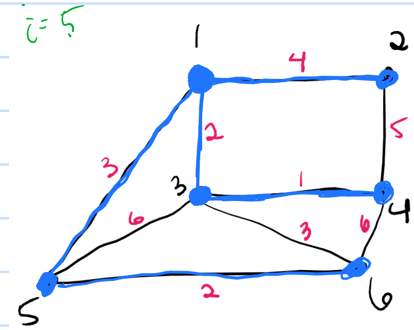
selects 5 to add

i=4



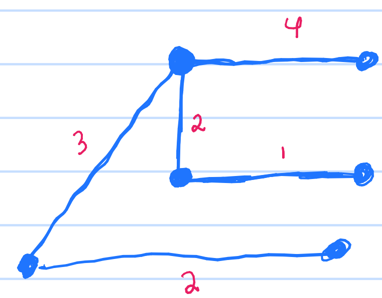
selects 6 to add

i=5



selects 2 to add

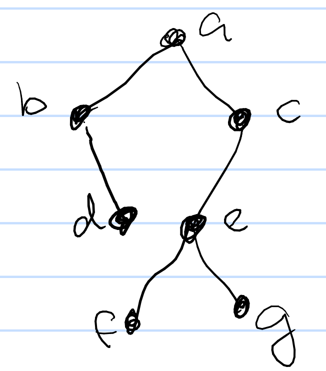
outputs



9.5 Binary Trees

Def A binary tree is a rooted tree where each vertex has either 0, 1, or 2 children. If 1, we designate it as either "right" or "left". If 2, one is designated right + the other as left.

Ex



is a binary tree

d, f, g have 0 children

e is left child of c
d is right child of b
b, c are left + right children of a, respectively

Def A full binary tree is a binary tree in which each vertex has 0 or 2 children.

Thm If T is a full binary tree with i internal vertices, then T has $i+1$ terminal vertices + $2i+1$ total vertices.

Pf

$$V(T) = \{v \mid v \text{ is a child of some } w \in V(T)\} \cup \{\text{root vertex}\}$$

Since there are i internal vertices + each must have 2 children $\Rightarrow$ there are $2i$ children total

$$\Rightarrow V(T) = 2i+1$$