CSE 201 3-2-21

Theorem (Treeness)

The following statements are equivalent.

(1) G is a tree (acyclic & connected)

(2) G is connected and $|E| = |V| - 1$.

(3) G is acyclic and $|E| = |V| - 1$

(4) G is acyclic, but if a new edge is added, a unique cycle is created.

(5) G is connected, but the removal of an edge disconnects G.

(6) There is a unique x-y path in G for all $x, y \in V$.

Kruskal $\begin{cases} n = |V| \\ m = |E| \end{cases}$

- choose an edge of min weight.
- while # edges chosen $< n-1$
- amongst all edges which do not create a cycle with previously chosen edges choose one of min weight.

Rmk

- at each stage the chosen edges form an acyclic spanning subgraph of G. (spanning forest.)
- When the forest contains $n-1$ edges, it must be connected, since any any acyclic graph on n vertices & $n-1$ edges is connect, by treeness thm.

- So Kruskal's alg. builds a SP, tree, call it T.

Theorem

This tree T has min. weight amongst all spanning trees in G.

Proof.

Let S be any other SP. tree in G. we must show

$$w(T) \leq w(S).$$

Let $E(T) = \{e_1, e_2, \ldots, e_{n-1}\}$, given in the order chosen by Kruskal. since $S \neq T$, there is a first edge $e_k \in T$ which is not in S.

i.e. $\{e_1, \ldots, e_{k-1}\} \subseteq E(S)$

$e_k \notin E(S)$

Let H be the subgraph obtained by adding e_k to S:

$$H = S + e_k$$

By Treeness theorem H contains a unique cycle (which contains e_k), Call the cycle C.

Note C must contain an edge e of S that is not in T (otherwise, T would contain C, but T is acyclic.)

Now remove e from H; call resulting subgraph R

$$R = H - e = S + e_K - e$$

Note R is connected, since we removed a cycle edge. Also R has $n-1$ edges, so must be a SP. tree.

The nature of Kruskal guarantees that $w(e_K) \leq w(e)$.

why? e does not form a cycle with $\{e_1, \ldots, e_{K-1}\}$ since

$$\{e_1, \ldots, e_{K-1}, e\} \subseteq E(S).$$

If $w(e) < w(e_K)$, then Kruskal would have chosen e on K^{th} iteration, instead of e_K.

$\therefore$ $w(e_K) \leq w(e)$

Therefore: $w(R) \leq w(S)$.

Also R has one more edge in common with T than S does:

$$E(S) = \{ \underbrace{e_1, e_2, \ldots, e_{k-1}}_{\text{in } T}, \underbrace{e_{kj}}_{\text{not in } T} \ldots \}$$

$$E(R) = \{ \underbrace{e_1, e_2, \ldots, e_{k-1}}_{\text{in } T}, \underbrace{e}_{\text{in } T}, \ldots \}$$

If $R = T$, we're done. If not we can do the same construction with R in place of S. i.e. Construct a sp. tree R_2 with one more edge in common with T, and $w(R_2) \leq w(R)$.

Continuing, we obtain a sequence

$$S, R, R_2, \ldots, T$$
$$\uparrow \quad \uparrow$$
$$R_0 \quad R_1$$

eventually, we must reach T.

we have

$$w(T) \leq \ldots \leq w(R_2) \leq w(R_1) \leq w(R_0)$$
$$\uparrow \qquad \uparrow$$
$$R \qquad S$$

hence $w(T) \leq w(S)$. ∎

Lower bounds

Consider some problem P, in all its instances. Two goals

1. find an algorithm that solves P, and find an asymp. upper bound for its runtime: $O(f(n))$ for some fcn. $f(n)$. We aim to reduce $f(n)$ by finding better algorithms.

2. Prove that any algorithm solving P runs in time $\Omega(g(n))$ for some fcn. $g(n)$, we aim to increase $g(n)$ by discovering better proofs.

we're happy when $f(n) = \Theta(g(n))$
(or better yet, $f(n) \sim g(n)$.)
Then we have the best possible algorithm.

(1) is called algorithmics.

(2) is called computational complexity.
$g(n)$ is called a lower bound for P.

Two types of lower bounds

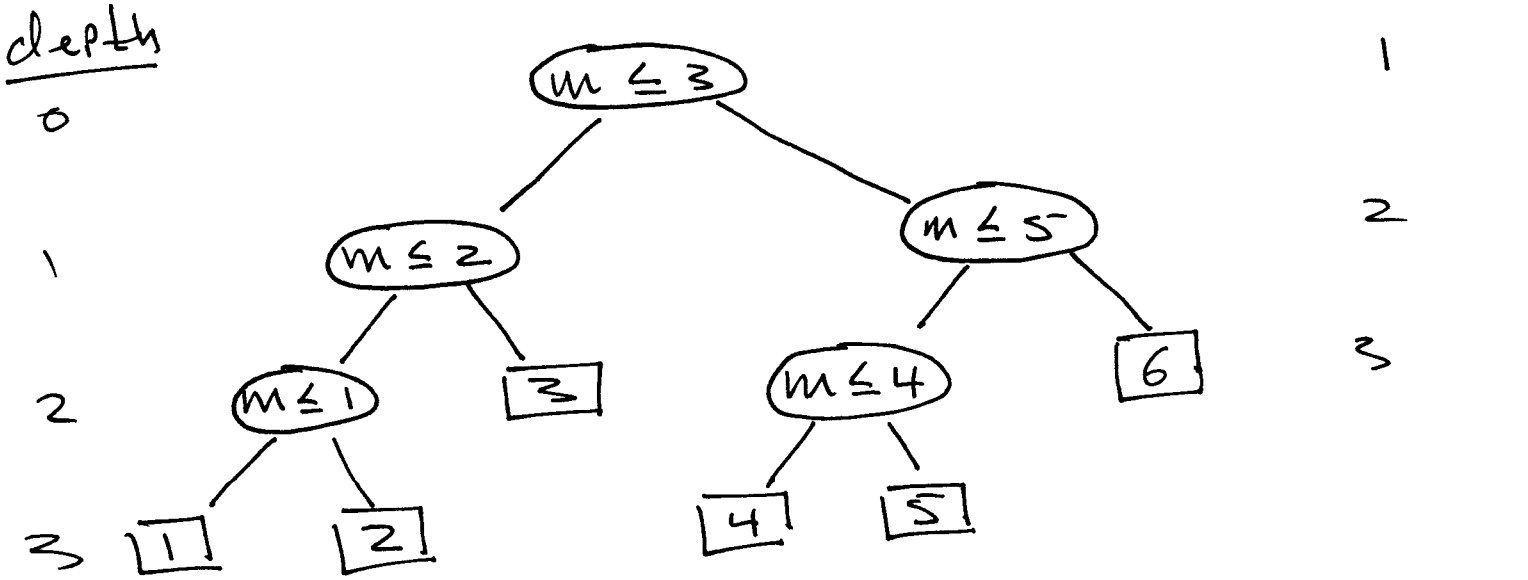
- Decision tree lower bounds
  (also information theoretic L.B.)
- Adversary argument. L.B.

Decision Trees

Ex. Let $m \in \{1, 2, 3, 4, 5, 6\}$.

Problem: determine m by asking a seq. of yes/no questions.

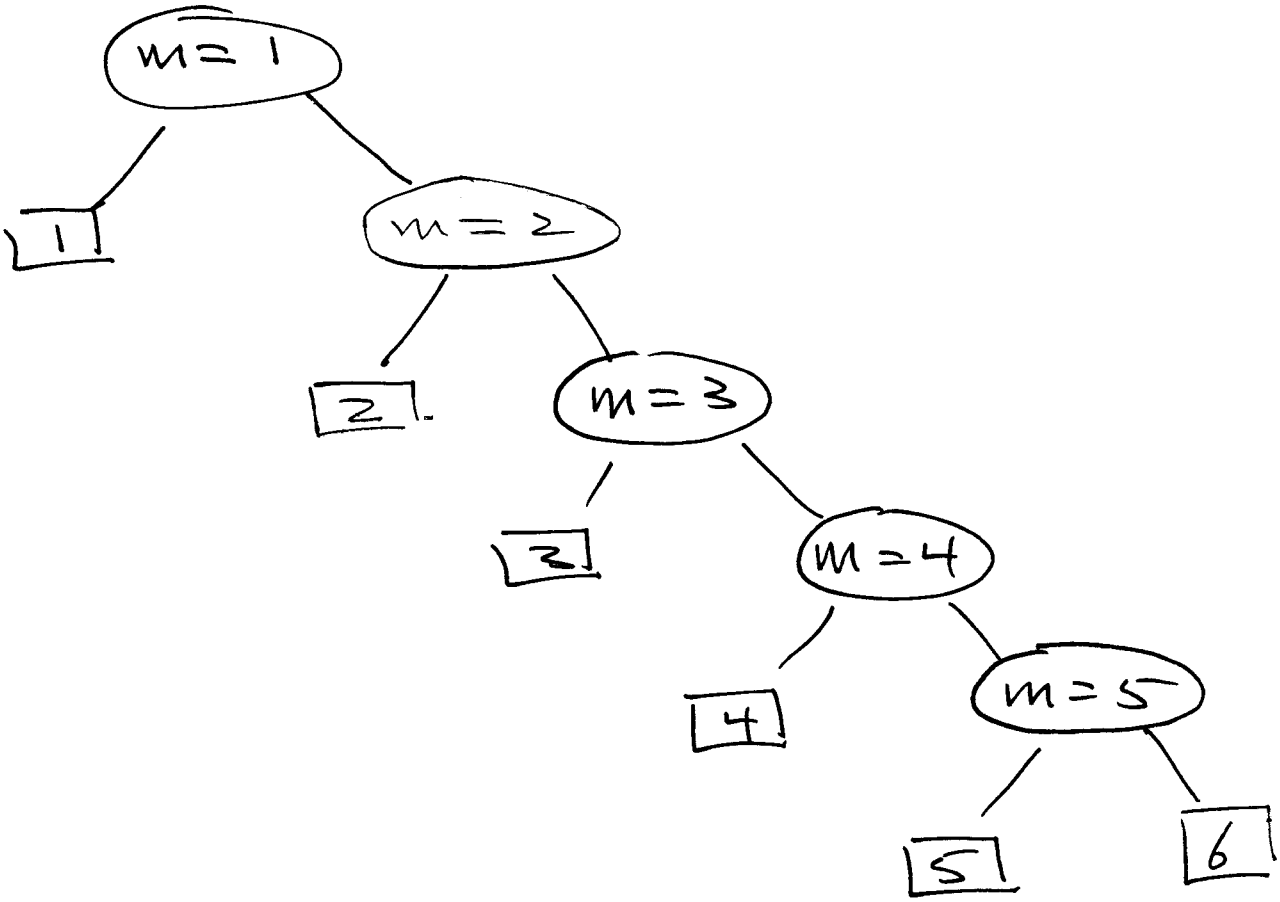
Variation of binary search:



(worst case # ques.) = 3

(avg case # ques.) = $\frac{4 \cdot 3 + 2 \cdot 2}{6} = \frac{8}{3} = 2.66$

Linear search as a decision tree:



worst case = 5 questions

$$\text{avg case} = \frac{1 \cdot 1 + 1 \cdot 2 + 1 \cdot 3 + 1 \cdot 4 + 2 \cdot 5}{6}$$

$$= \frac{20}{6} = 3.33\ldots$$