Now remove e from H to obtain a subgraph R, which is connected since e belongs to a cycle in H.

\[ R = H - e = S + e_k - e \]

Since R is connected and has n-1 edges, it is another spanning tree of G.

By treeless theorem:

The nature of Kruskal's algorithm guarantees that \( w(e_k) \leq w(e) \).

**Proof:** e does not form a cycle with e_1, ..., e_{k-1} since \( e, e_k, e_j \in E(S) \). Thus if \( w(e) < w(e_k) \), then Kruskal would have chosen e on the kth iteration of the greedy loop instead of \( e_k \). \( \square \)

Thus R is a spanning tree of G with one more edge in common with H than S, and satisfies \( w(R) \leq w(S) \).

If \( R = T \) we are done, otherwise we may perform this same construction with \( e_k \) in place of e.
i.e. construct another spanning tree
$R_2$ with one more edge in common
with $T$ than $R$, and satisfying $W(R_1) \leq W(R)$.
Continuing in this fashion we construct
a sequence of spanning trees without
must eventually reach $T$:

$W(T) \leq \ldots \leq W(R_1) \leq W(R) \leq W(S)$,
so $W(T) \leq W(S)$ as required.
Consider the set of all algorithms (known & unknown) which solve some problem \( P \) in all its instances.

Our goals are twofold:

1.) Find an algorithm which solves \( P \) in (worst case) time \( O(f(n)) \) for some function \( f(n) \) which we aim to reduce as far as possible.

2.) Prove that any algorithm which solves \( P \) must run in (worst case) time \( \Omega(g(n)) \) for some function \( g(n) \) which we aim to increase as far as possible.

Here \( n \) denotes the 'size' of an instance of problem \( P \).

We are happy when \( f(n) = \Theta(g(n)) \) for then we know we have the best possible algorithm to solve \( P \) (apart from improvements in hidden constants.)
(1) is called algorithms by some authors, while (2) is the theory of computational complexity. The function \( g(n) \) is in \( (2) \) is called a lower bound on the complexity of Problem \( P \).

**Decision Trees / Information Theoretic Lower Bounds**

**Example:** Let \( m \) be an integer in the range \( 1 \leq m \leq 6 \). Problem: Determine the value of \( m \) by asking a sequence of yes/no questions.

This problem is similar to binary search!
Apparently the answer can be obtained by asking no more than 3 questions.
(will 2 suffice?)

Review: Graph - Tree - Rooted Tree.

A **Graph** $G = (V, E)$ is a pair of sets called **Vertices** $(V)$ and **Edges** $(E)$. Each edge joins a (unique) pair of (distinct) vertices. Two vertices which are joined by an edge are called **Adjacent**.

![Graph](image1)

A **Path** in $G$ is a sequence of consecutively adjacent vertices. $G$ is called **Connected** if every pair of vertices in $G$ are joined by a path. A **Cycle** is a closed path, i.e. a path in which the initial and terminal vertices are identical. $G$ is called **Acyclic** if it contains no cycle. A **Tree** is a connected acyclic graph.

![Tree](image2)
A **rooted tree** is a tree in which one vertex has been distinguished as the **root**. The vertices in such a tree are often called **nodes**. The depth of a node is its distance from the root. (Distance means shortest path length.)

![Tree Diagram]

The children of a node $x$ are those nodes adjacent to $x$ whose depth is one more than that of $x$. The **parent** of $x$ is the (unique) node which is adjacent to $x$ and has depth one less than that of $x$.

The root is the only node which has no parent (since the root has depth 0). If a node $y$ has no children, it is called a **leaf**. A non-leaf node is called an **internal node**.
The height of a rooted tree is its maximum node depth, i.e., the length of a longest downward path from the root to a leaf. The height of a node is the height of the subtree rooted at that node.

A binary tree is a rooted tree in which each node has at most 2 children. More generally, a k-ary tree is a rooted tree in which each node has at most k children.

A complete binary tree (CBT) is a binary tree in which all leaves are at the same depth, and each internal node has exactly 2 children.

\[ h = 2 \]

\[
\begin{array}{c|c}
\text{Depth} & \# \text{Nodes} \\
0 & 1 \\
1 & 2 \\
2 & 4 \\
3 & 8 \\
\end{array}
\]

\# nodes at depth \( d \) = \( 2^d \)

\# leaves = \# nodes at depth \( h \) = \( 2^h \)

The height of a CBT with \( n \) leaves is \( h = \lg(n) \).
An Almost Complete Binary Tree (ACBT) is a binary tree which has the maximum possible number of nodes at each depth, except possibly the last, which is filled from left to right.

\[ h = 3 \]

(An ACBT is the basis of the heap data structure.)

Exercise

Prove that an ACBT with \( n \) leaves and height \( h \) satisfies \( \lceil \log n \rceil \leq h \leq \lceil \log n \rceil + 1 \).

Theorem

The height \( h \) of any binary tree with \( n \) leaves satisfies

\[ h \geq \lceil \log n \rceil \]

Notation:

Let \( L(T) \) and \( H(T) \) denote the number of leaves and the height of a