Contents

1 Tree

2 Tree Traversal
  2.1 Preorder
  2.2 Postorder
  2.3 Inorder

3 Binary Search Tree (BST)
  3.1 Search in BST
  3.2 Insertion in BST
  3.3 Deletion in BST

4 Minimum Spanning Tree (MST)
  4.1 Kruskal’s algorithm
  4.2 Prim’s algorithm
1 Tree

Definitions  Trees are special types of graphs. A tree is a connected graph without cycles. To make the analysis easier, we define depths/levels for the trees, and the node at the top level is called the root, and the nodes at the bottom levels are called leaves. The non-leaf nodes are called internal nodes as well. Every node's level is decided by its distance (i.e., number of edges) to the root node. Because of the way the depths/levels are defined, any tree only has one root, and leaves are not necessarily at the same depth/level. In this course, we will uniformly call ‘depths’ instead of ‘levels’ for the sake of simplicity. Here are some examples. ‘Height’ is also defined for the nodes, which is the distance to the deepest leaf.

![Diagram of a tree with labeled nodes and edges]

Notably, depending on which node is selected as the root, the tree looks differently (the arrows are there to highlight the depths of the nodes, and trees are always undirected), but in fact they are isomorphic to each other.

Node relationship  The way the relationship between nodes is defined is similar to that of the family relationship.

- Parent of $v$: the node right above $v$.
- Children of $v$: the nodes right below $v$.
- Siblings of $v$: the nodes having the same parent.
- Cousins of $v$: the nodes have the same grandparent.
- Ancestors of $v$: $v$’s parent, grandparent, grand-grandparent, …, and the root.
- Descendants (Offspring) of $v$: $v$’s children, grandchildren, grand-grandchildren, …, and the leaves.

Subtree  Given a node $v$ in a tree $T$, the subtree $T'$ rooted by $v$ is the subgraph of $T$ who consists $v$ as the root and all of $a$’s descendants and edges among the descendants. One example is:
**m-ary tree**  A tree is called $m$-ary if every internal node has no more than $m$ children. It is called a full $m$-ary tree if every internal node has exactly $m$ children, and it is called a complete $m$-ary tree if it is full and all leaves are at the same level. A frequently used $m$-ary tree is the binary tree.

* Suppose $S(h)$ is the number of subtrees of a complete binary tree of height $h$. Identify the recurrence formula for $S(h)$ and get the exact number of subtrees of a complete binary tree of height $h$. +20

**Implementation of a tree**  There are tons of different ways to implement a tree, and here is one of them. Every node is implemented as a data structure containing several pointers, which looks like:

![Figure from http://www.openbookproject.net/thinkcs/archive/java/english/chap17.htm](http://www.openbookproject.net/thinkcs/archive/java/english/chap17.htm)

In the above example each node has two pointers because it is an implementation of a binary tree, but a node can have arbitrarily many pointers pointing to children in theory. Implementation will be discussed with more details in CS331.

## 2 Tree Traversal

Three types of tree traversal exist for a binary tree: pre-order, post-order, and in-order.

### 2.1 Preorder

In the preorder traversal, the tree is visited recursively in the following way. Note that the node will be visited before any recursion is called.

1. Visit the node $v$.
2. Traverse $v$’s left subtree recursively with the preorder if $v$ has a left child.
3. Traverse $v$’s right subtree recursively with the preorder if $v$ has a right child.

If we print the characters in the following tree by preorder traversal:

![Tree](image)

What we will print is: $++a*bc*++defg$
2.2 Postorder

In the postorder traversal, the tree is visited recursively in the following way. Note that the node will be visited after all recursions are already called.

1. Traverse $v$’s left subtree recursively with the preorder if $v$ has a left child.
2. Traverse $v$’s right subtree recursively with the preorder if $v$ has a right child.
3. Visit the node $v$.

If we print the characters in the same tree above by postorder traversal, we will print: abc*+de*f+g*+

2.3 Inorder

In the inorder traversal, the tree is visited recursively in the following way. Note that the node will be visited right after the recursion on the left subtree is called and right before the recursion on the right subtree is called.

1. Traverse $v$’s left subtree recursively with the preorder if $v$ has a left child.
2. Visit the node $v$.
3. Traverse $v$’s right subtree recursively with the preorder if $v$ has a right child.

If we print the characters in the same tree above by inorder traversal, we will print: a+b*c+d*e+f*g

3 Binary Search Tree (BST)

Binary search tree is a special tree, and it is extremely important in many areas in computer science. It has become the key component of many important systems including, but not limited to, database index, file system, and cloud deduplication (finding out duplicate files).

It has only one property: for any node $v$, all the keys in its left subtree are smaller than $v$’s key, and all the keys in its right subtree are larger than $v$’s key. One example is shown below:
For the same set of keys, there might be multiple BSTs.

Some have low height (or depth) and some have high height (or depth). For a set of \( n \) keys, the lowest possible height is \( \Theta(\log n) \) and the highest possible height is \( \Theta(n) \).

### 3.1 Search in BST

Usually, only the pointer to the root is available for a tree. Therefore, searching for a key in a BST starts from the root and traverses the tree all the way down to the leaf until the desired key is found. Whether the desired key is less or greater than the currently visiting node’s key determines whether the left/right subtree is searched. In the worst case, the number of visits involved in the searching is as many as the height of the tree, therefore the complexity is \( O(\text{height}) \).

* What is the algorithm to find the minimum key and maximum key in a BST? +2

### 3.2 Insertion in BST

Suppose the key being inserted does not exist in the BST. Then, when a key \( k \) needs to be inserted, the searching process for \( k \) is performed in the tree. At the end, an empty spot must be discovered since \( k \) does not exist. The key \( k \) is inserted at the discovered empty spot at the end. Similar to the searching, the complexity is \( O(\text{height}) \).

### 3.3 Deletion in BST

When deleting a key \( k \) from the BST, the node containing \( k \) is first searched. Then, since the deletion cannot break the BST property, we do a little complicated procedure.

If the node to be deleted is a leaf node, remove it simply. Otherwise, repeat the following procedure.

1. Find out the immediate successor of \( k \) in the BST, which is the left-most node in the right subtree. Let’s say the left-most node’s key is \( k' \). Update \( k \)’s value to \( k' \).
   - We can discover the left-most node in the right subtree by visiting the left-child over and over until we see a node having no left child, then this node is the immediate successor. For example in the following graph, 3 is the immediate successor of 2.
2. Now, instead of deleting the node used to have $k$, delete the node used to have $k'$.

3. The above two are repeated until the node to be deleted is the leaf and therefore can be removed immediately.

4 Minimum Spanning Tree (MST)

With in a graph $G$, a tree is a spanning tree if it contains all of the $|V|$ vertices and $|V| - 1$ edges to connect all of them (and therefore spans the entire graph). Obviously, there can be more than one spanning tree, and in the weighted graph, the minimum spanning tree (MST) is the spanning tree whose sum of weights of the edges is the smallest. For example,

![Graph with weights](image)

The blue-line spanning tree is the minimum spanning tree with the total weight $3600$, and there is no other spanning tree whose total weight is smaller than that.

The essence of many problems boils down to finding MST.

- Network design: telephone, internet, TV cable etc.
- Approximating NP-hard problems: TSP
- Other applications: Learning features for face verification, throughput maximization in networks with bottleneck links

Two algorithms are famous: Kruskal’s algorithm and Prim’s algorithm

4.1 Kruskal’s algorithm

The basic idea of Kruskal’s algorithm is to repeatedly find the minimum-weight edge in the graph and try to fit it in the tree $T$. If the edge forms a cycle in $T$, then discard it; otherwise, add the edge to $T$. No matter whether the edge is added or not, it is removed from the graph, and the procedure is repeated until $T$ has $|V| - 1$ edges and therefore becomes a spanning tree.
A rough pseudo-code is available in the textbook. Note that this pseudo-code only gives a sketch of the idea, and it is not complete.

**procedure Kruskal(G: weighted connected undirected graph with n vertices)**

\[ T := \text{empty graph} \]

\[ \text{for } i := 1 \text{ to } n - 1 \]

\[ e := \text{any edge in } G \text{ with smallest weight that does not form a simple circuit} \]

\[ T := T \text{ with } e \text{ added} \]

\[ \text{return } T \{ T \text{ is a minimum spanning tree of } G \} \]

Here is an example of the Kruscal’s algorithm.

**4.2 Prim’s algorithm**

The basic idea of Prim’s algorithm is to find the minimum-weight edge in the graph first and add it to an empty tree \( T \). Then, the minimum-weight edge among all the edges that are incident to \( T \) is added to \( T \) if it does not form a cycle; if it does, the next minimum-weight edge is tried. The procedure is repeated until \( T \) contains \( |V| - 1 \) edges.

A rough pseudo-code is available in the textbook as well, but this pseudo-code only gives a sketch of the idea too, and it is not complete either.

**procedure Prim(G: weighted connected undirected graph with n vertices)**

\[ T := \text{a minimum-weight edge} \]

\[ \text{for } i := 1 \text{ to } n - 2 \]

\[ e := \text{an edge of minimum weight incident to a vertex in } T \text{ and not forming a simple circuit in } T \text{ if added to } T \]

\[ T := T \text{ with } e \text{ added} \]

\[ \text{return } T \{ T \text{ is a minimum spanning tree of } G \} \]

Here is an example of the Prim’s algorithm.
(a) 

(b) 

<table>
<thead>
<tr>
<th>Choice</th>
<th>Edge</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{b, f}</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>{a, b}</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>{f, j}</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>{a, e}</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>{i, j}</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>{f, g}</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>{c, g}</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>{c, d}</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>{g, h}</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>{h, l}</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>{k, l}</td>
<td>1</td>
</tr>
</tbody>
</table>

Total: 24