FinalRC_Meng Zi :)

Date: 2024/12/13

AVL Tree

First you need to know that all of the operations (insert, remove, search) on AVL tree are O(logn)

Balance condition

- Empty tree
- or both its left and right subtrees are AVL balanced && the height diff between them at most 1

Height limitation

$$log_2(n+1)-1 \leq h \leq 1.44log_2(n+2)$$

Please try to prove it by yourself

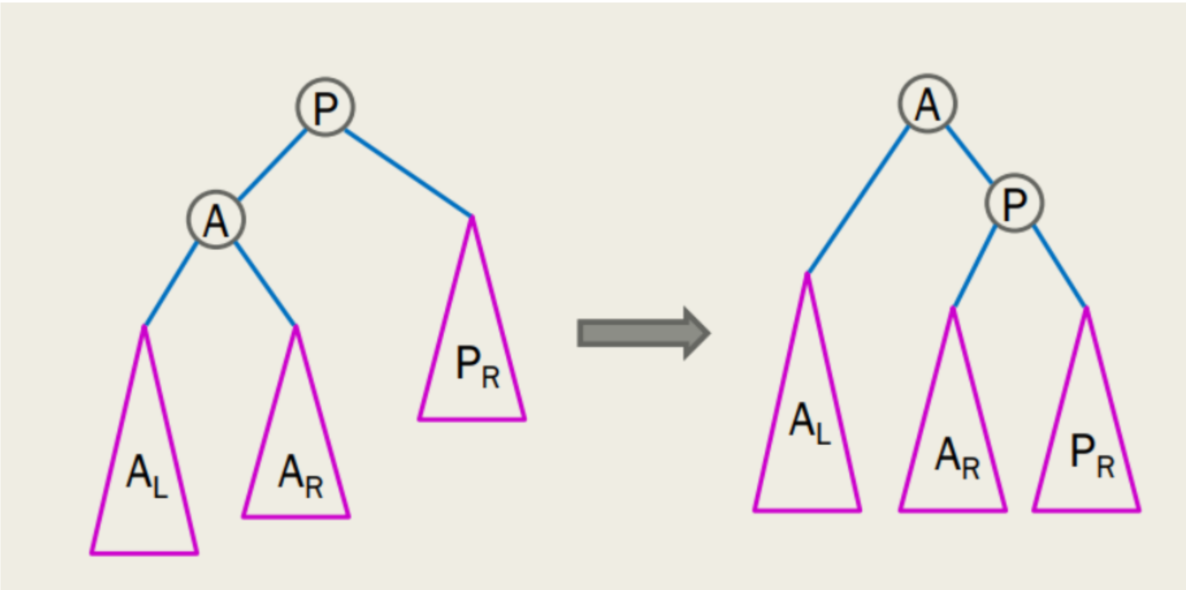
Re-balance

When to re-balance?

- You can check the def of **balance factor**. Also, if the height diff between two subtrees is greater than 1, you need to re-balance it.
- every time you do insertion or removal, you need to check whether you need to do a re-balance

Rotation

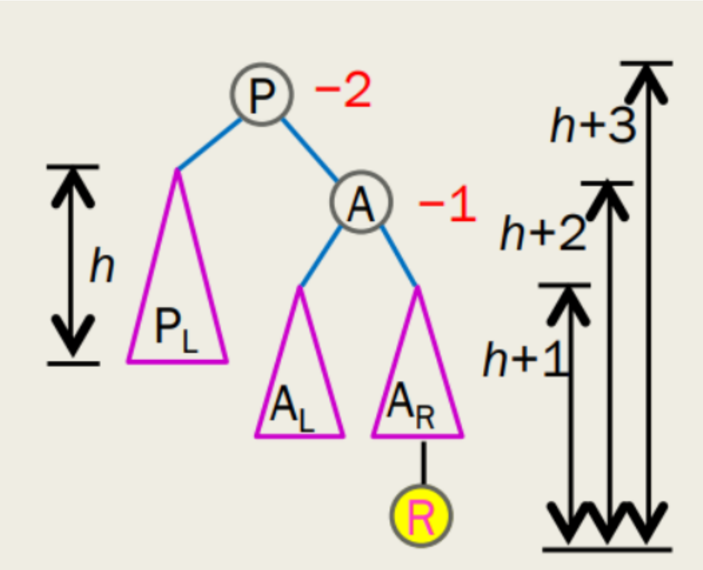
Right Rotation:

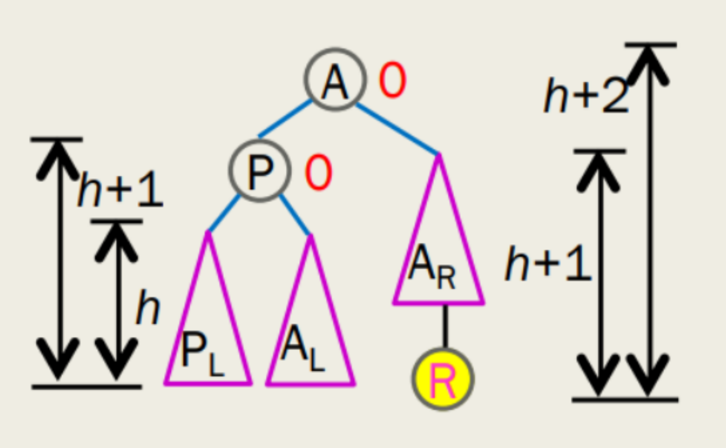


```
void rightRotation(Nood *& P){
    Node* A = P->left;
    Node* A_right = P->left->right;
    A->right = P;
    P->left = A_right;
    renewHeight(P);
    renewHeight(A);
}

//But...
```

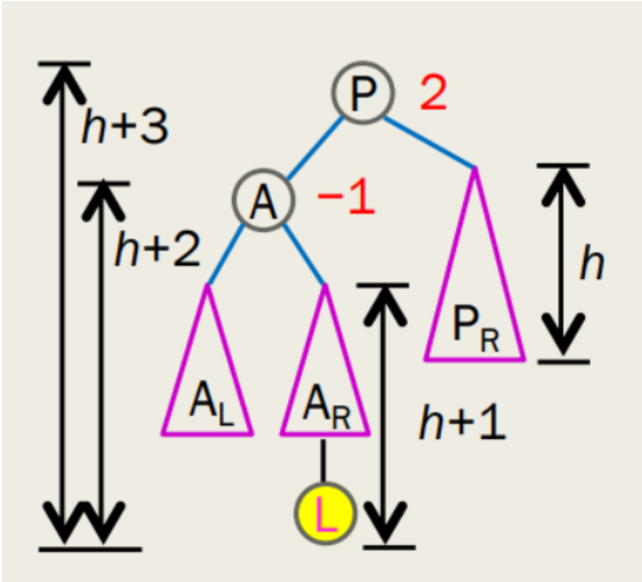
RR Rotation:

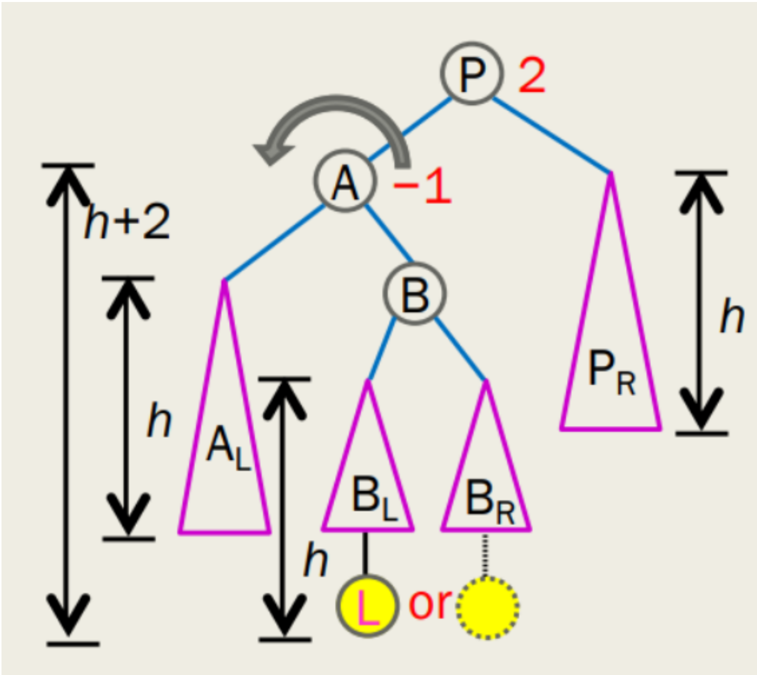


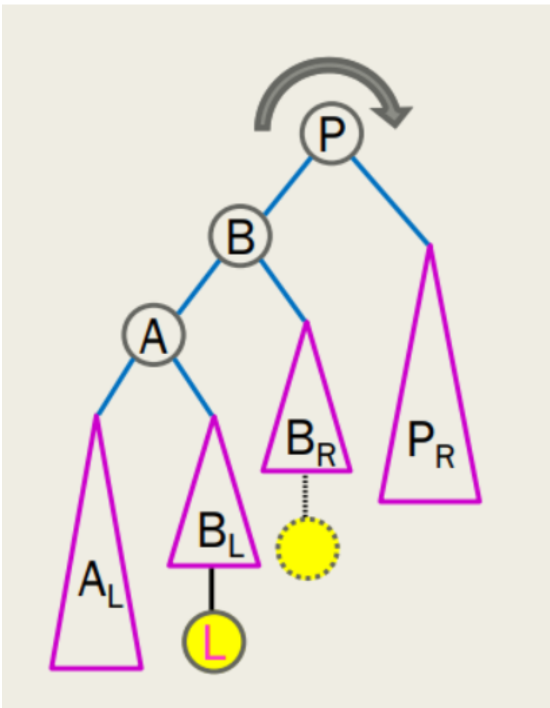


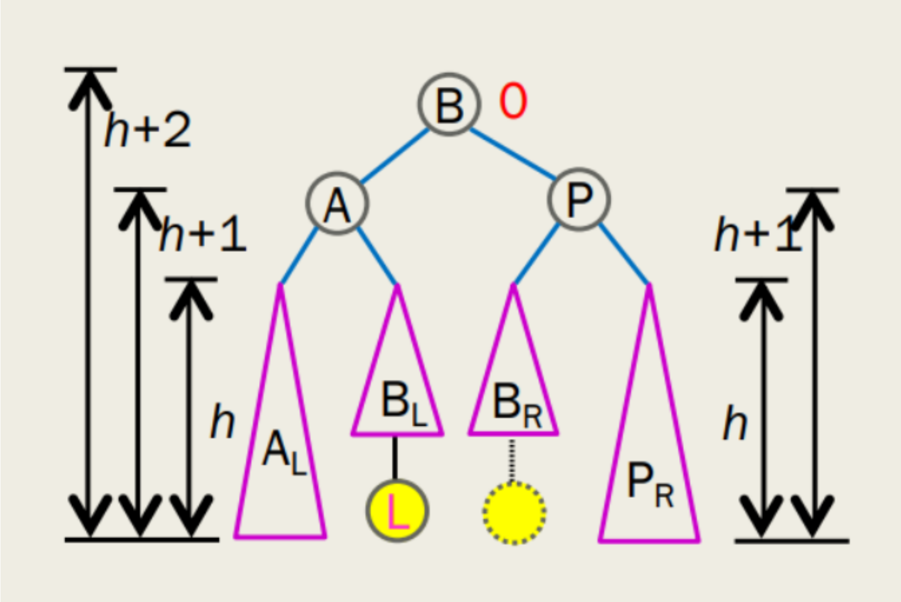
Final height is the same as the height before insertion.

LR Rotation:


P
2
A
−1
h+3
h+2
h+1
h
A_L
A_R
P_R
L


P
2
A
−1
B
h+2
h
h
h
A_L
B_L
B_R
P_R
L
or


P
B
A
A_L
B_L
B_R
P_R
L



Final height is the same as the height before insertion.

How to memorize them?

Implementation

Preparation

```
struct{
    Item item;
    int height;
    node *left;
    node *right;
}
```

A quick question: when we do insertion or removal, how to renew the height and balance factor?

```
int getHeight(node *n){
    if(!n) return -1;
    return n->height;
}

void RenewHeight(node *n){
    if(!n) return;
    n->height = max(getHeight(n->left),getHeight(n->right)) + 1;
}

int BalFactor(node*n){
    if(!n) return 0;
    return (getHeight(n->left) - getHeight(n->right));
}
void Balance(node *&n) {
    if(BalFactor(n) > 1) {
        if(BalFactor(n->left) > 0) LLRotation(n);
        else LRRotation(n);
    }
```

```
    else if(BalFactor(n) < -1) {
        if(BalFactor(n->right) < 0) RRRotation(n);
        else RLRotation(n);
    }
}
```

```
void insert(node *&root, Item item){
    if(root == NULL){
        root = new node(item);
        return;
    }
    if(item.key < root->item.key) insert(root->left, item);
    else insert(root->right, item);

    //why we don't need to renewheight here?
    Balance(root);
    RenewHeight(root);
}
```

Remove is similar, try it as a practice by yourselves!

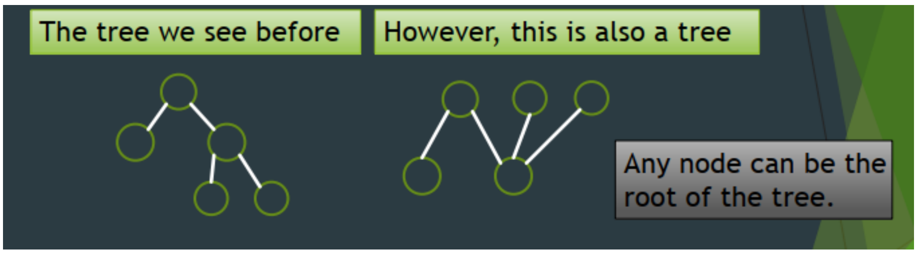
Minimum Spanning Tree

Definition Review

Tree: Connected acyclic undirected graph with num(E) = num(V) - 1

Spanning Tree: Subgraph of a graph that is a tree and contains all vertices of the graph.

Minimum Spanning Tree: Spanning tree with the smallest sum of edge weights.



Prim's Algorithm (greedy)

Basic idea:

1. Divide **V** into **T** and **T'**
2. Arbitrarily pick one node into **T**
3. while **T'** is not empty, find the shortest link between **T** and **T'** and move the node on **T'** side to **T**

We store a value $D(v)$ for every v in T' to represent the min distance between v and all the vertices in T. Every time, we choose $min(D(v))$ to move it from T' to T.

After we move v into T, for every neighbor vertices (linked with an edge) u, we compare $D(u)$ with $w(v, u)$ and choose the smaller one to be the new $D(u)$.

For those who need the code:

```
// V: number of vertices, adj: adjacency list
// weight(u, v): weight of edge (u, v)
vector<int> prim() {
    // D: shortest distance to the MST
    vector<int> D(V, INT_MAX);
    // in_tree: whether in the MST
    vector<bool> in_tree(V, false);
    D[0] = 0;
    vector<int> parent(V, -1);
    for (int i = 0; i < V; i++) {
        int v = -1;
        // Find the smallest D[v] for v not in the tree
        for (int u = 0; u < V; u++) {
            if (!in_tree[u] && (v == -1 || D[u] < D[v])) {
                v = u;
            }
        }
        if (v == -1) {
            // MST does not exist
            return vector<int>();
        }
        in_tree[v] = true;
        for (int u : adj[v]) {
            int w = weight(v, u);
            // Update D[u] for all u not in the tree
            if (!in_tree[u] && w < D[u]) {
                D[u] = w;
                parent[u] = v;
            }
        }
    }
    return parent;
}
```

Time complexity analysis:

- linear scan (T'): $O(V^2 + E)$
- binary heap (T'): $O(VlogV + ElogE)$
- Fibonacci heap (T'): $O(VlogV + E)$

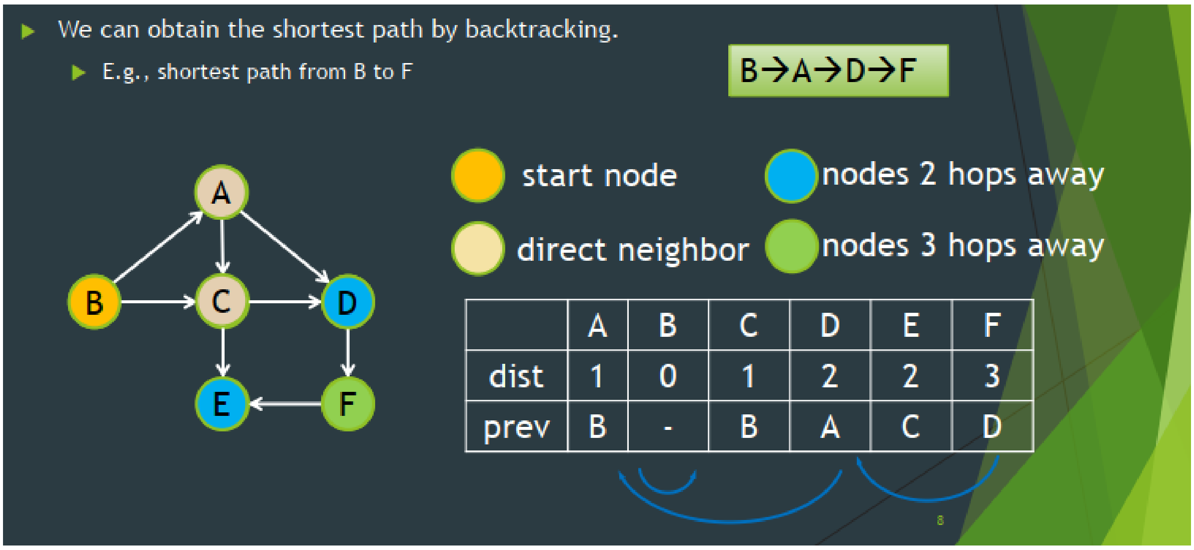
Shortest Path

To find the shortest path between two vertices in a graph.

Remember that we need to solve a **single source all destinations problem**

Unweighted graph:

Use BFS + store the predecessor, for example:



Weighted Graph:

We need to first make sure all the weights less than zero. **(why?)**

Then we use Dijkstra's Algorithm:

Denote

- distance between the vertice v and the starting vertice as $D(v)$
- the predecessor as $P(v)$

1. Initially, $D(s) = 0$, $D(v) = inf$, $P(v) = NULL$ for all $v \neq s$
2. Store all the nodes in a set R
3. while R is not empty
    1. Find the v_m that make $D(v_m)$ the min of $D(v)$, remove v_m from R
    2. The shortest path for v_m is $D(v_m)$, store it.
    3. for all the neighbor nodes u of v_m in R. If $D(v_m) + w(v_m, u) < D(u)$, we renew $D(u)$ and record $P(u) = v_m$

Strongly encourage you to go through the whole process with an example shown in the slides.

For those who need the code:

```
// V: number of vertices, adj: adjacency list
vector<int> bfs(int start) {
    // visited is substituted by parent to record the path
    vector<int> parent(V, -1);
    // level: distance from start
    vector<int> level(V, -1);
    queue<int> q;
    q.push(start);
    level[start] = 0;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (int u : adj[v]) {
            if (parent[u] == -1) { // Not visited
                q.push(u);
                level[u] = level[v] + 1;
                parent[u] = v;
            }
        }
    }
    return parent; // or level, depending on the requirement
}
```

Ending

Some exam tips?

Thank all of you for finishing the course ECE2810J!

Actually, I have thought about a lot of conclusion remarks and tried to find some fancy sentences to work as the ending. Finally, I decide to say something in plain words "I wish all of you all the best in the future, achieve all your goals and realize all your dreams."