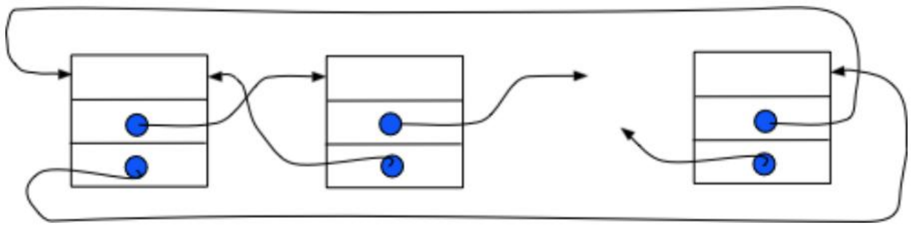
CS225 Review

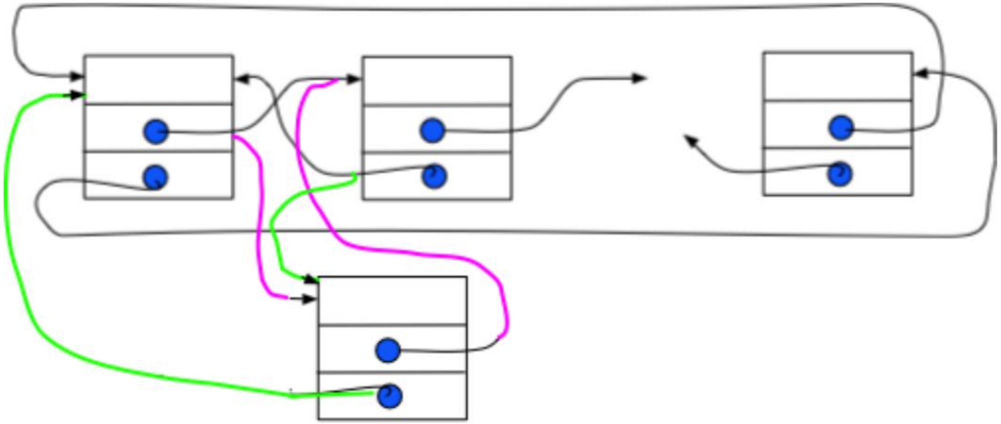
2023/05/29

List

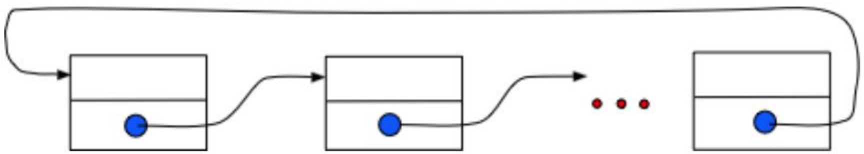
constructor default value 只在头文件
出现 default value custome constuctor 就不能有 default constructor

Doubly linked list:





Singly linked list:



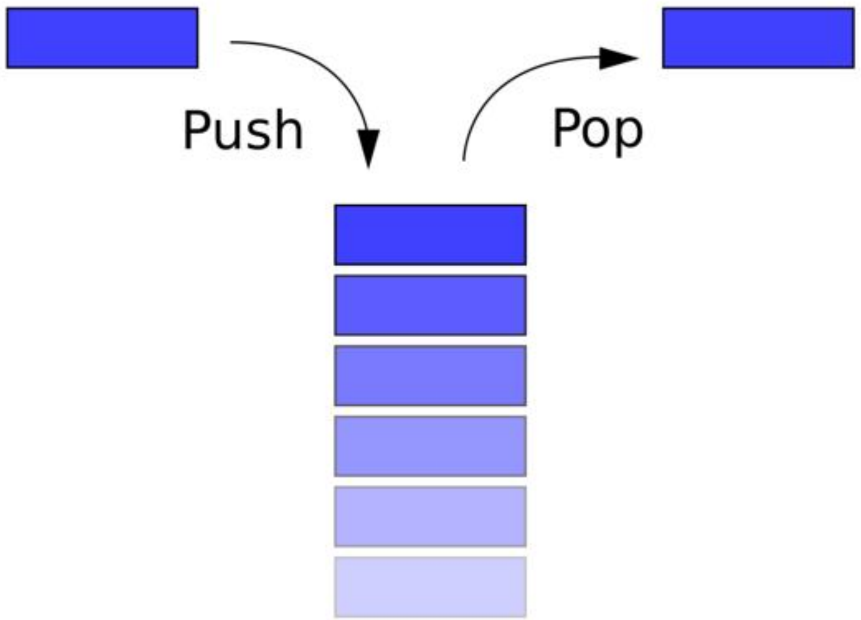
List

```
template<class T>
class LinkedList {
private:
   class listNode{
    public:
       listNode():next(NULL){}
       T data;
       listNode * next;
    };
   listNode * head;
public:
   // constructors, destructor, and other member functions
   ...
    void reverse();
};
```

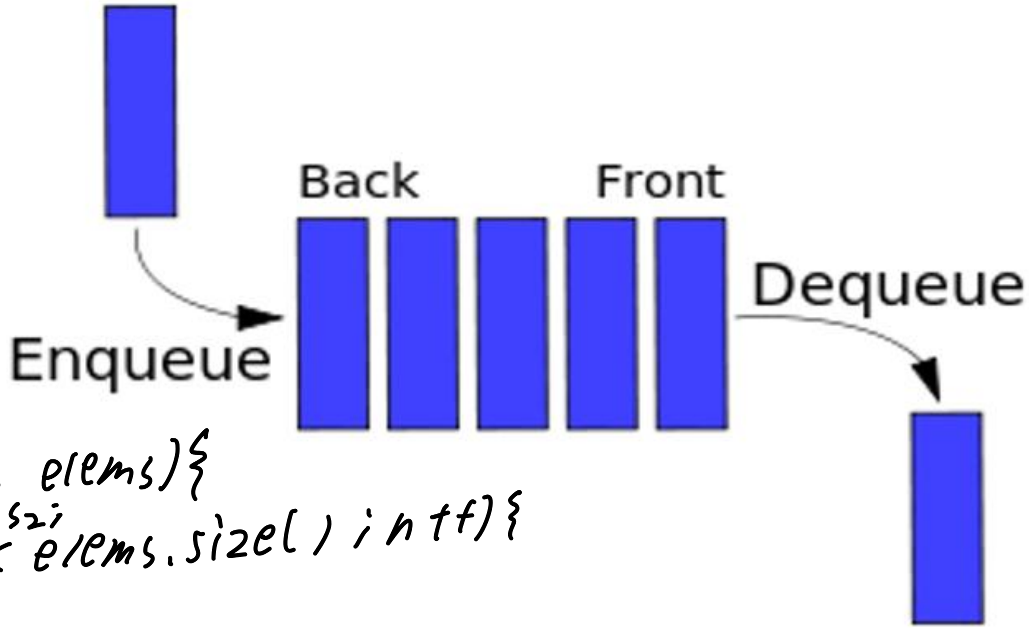
Analyze the running time of the function reverse(). You should state the **worst-case time complexity** of your code when operating on a list with n nodes and briefly explain why it would take that much time.

O(n)

Stack



Queue




```
FIFO(vector<int> elems){
    stack<int> s1, s2;
    for (n=0; n< elems.size(); n++){
        s1.push
    }
}   n019

FIFO_helper(stack<int>& s1, stack<int>& s2){
    if( empty()) return;
```

Stack and Queue

if (s1.empty()) ...

e1 = s1.top();

s2.push(e1)

n

s1.pop();

FIFO_helper(s1, s2);

s1.push(e1);

}

Explain how to implement a FIFO queue using two stacks so that each FIFO operation takes amortised constant time.

Tree Definition

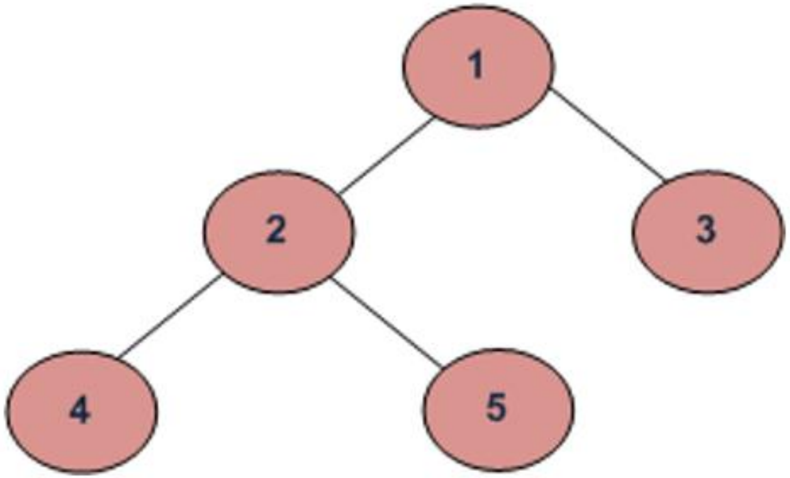
Tree Traversals (Inorder, Preorder and Postorder)

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways.

(a) Inorder (Left, root, Right) : 4 2 5 1 3

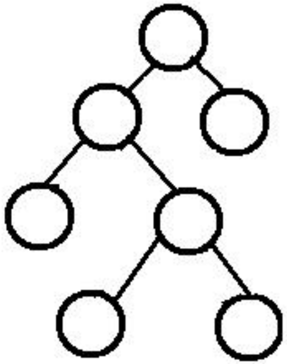
(b) Preorder (root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

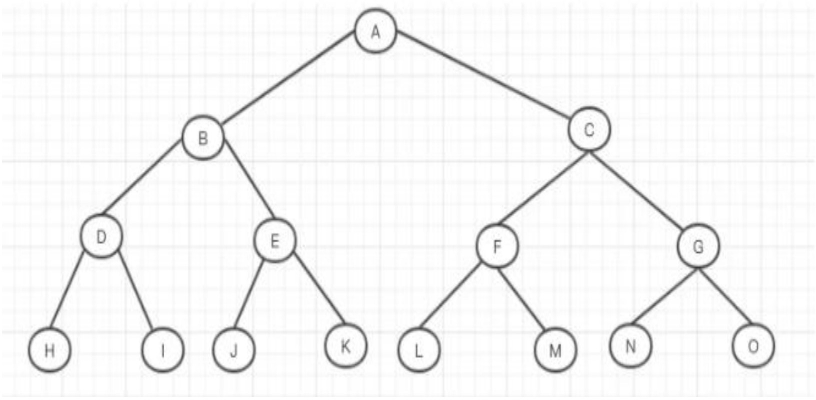


Tree Definition

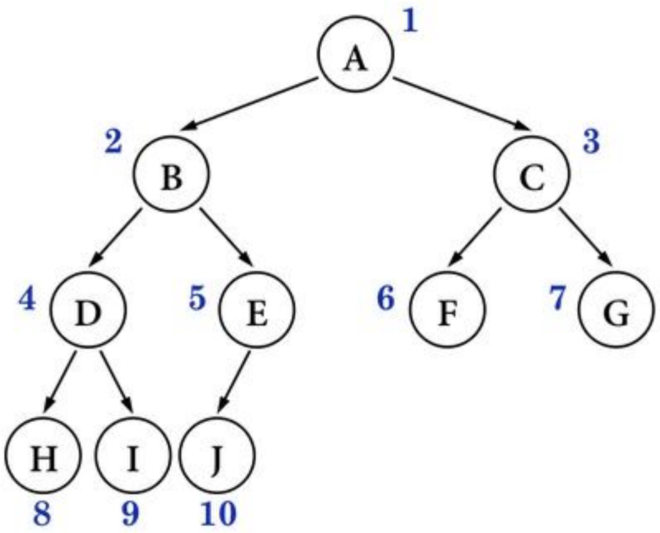
full binary tree



perfect binary tree

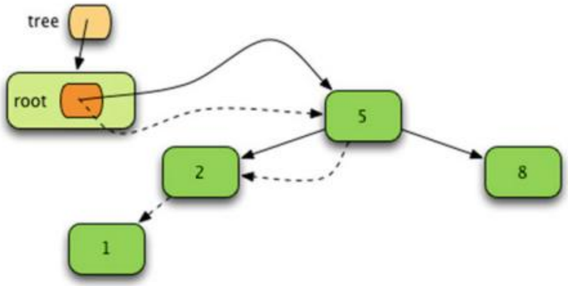


complete binary tree

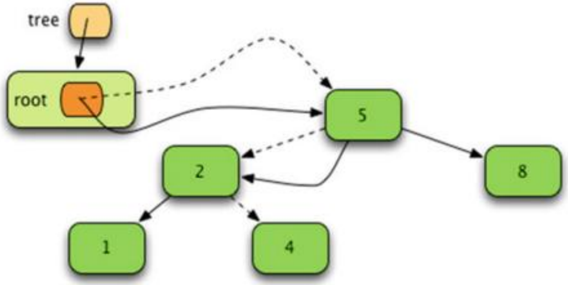


BST

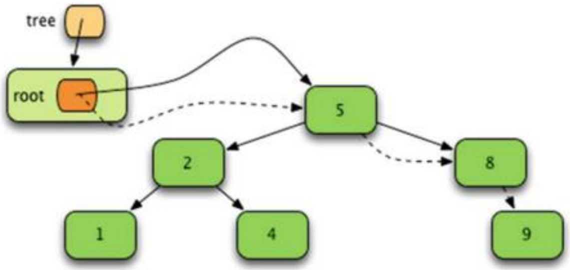
BST after insertion of 1



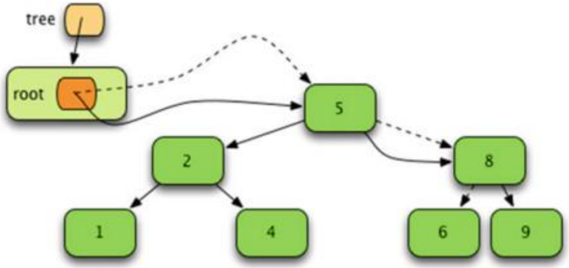
BST after insertion of 4



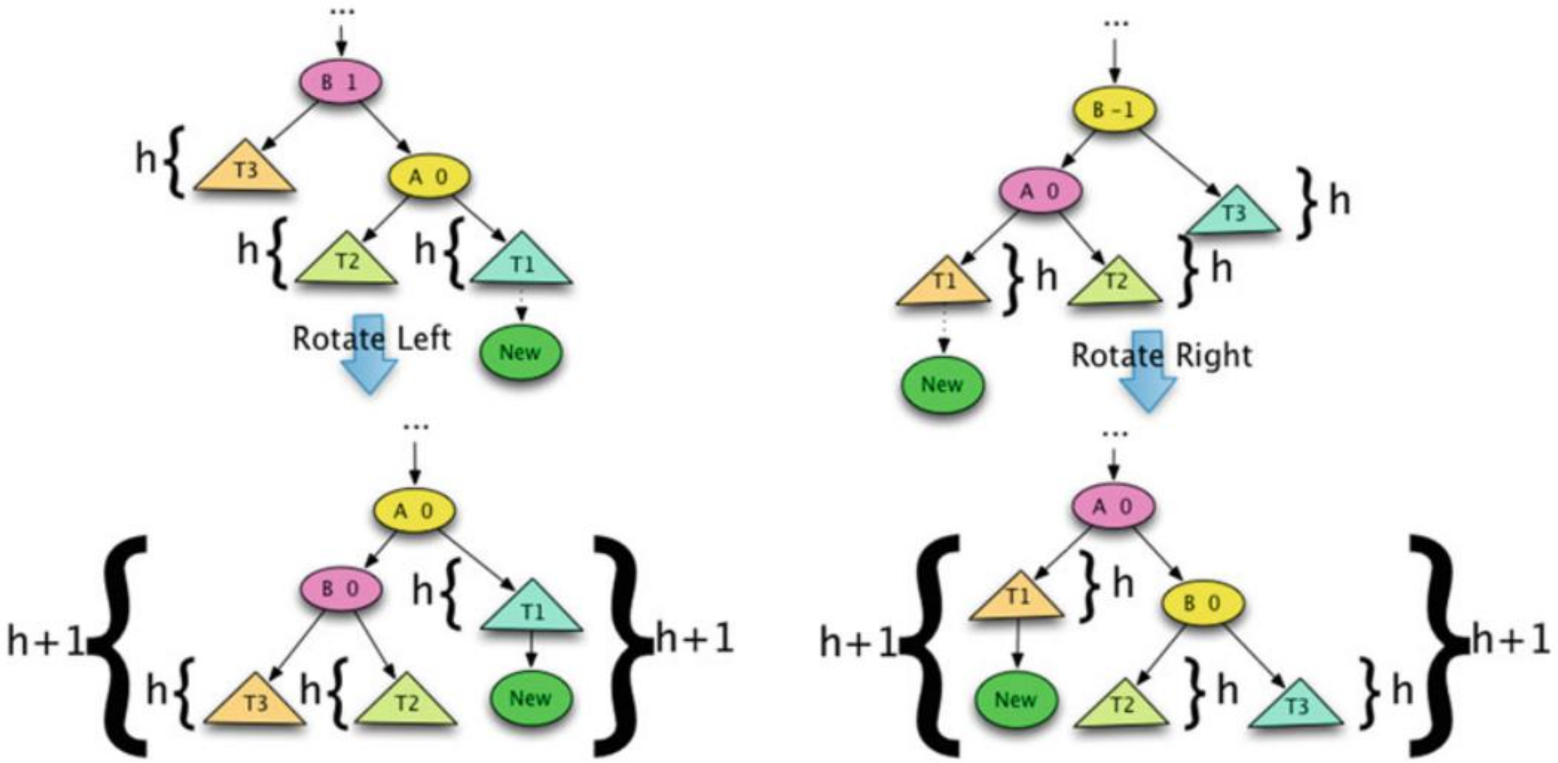
BST after insertion of 9



BST after insertion of 6



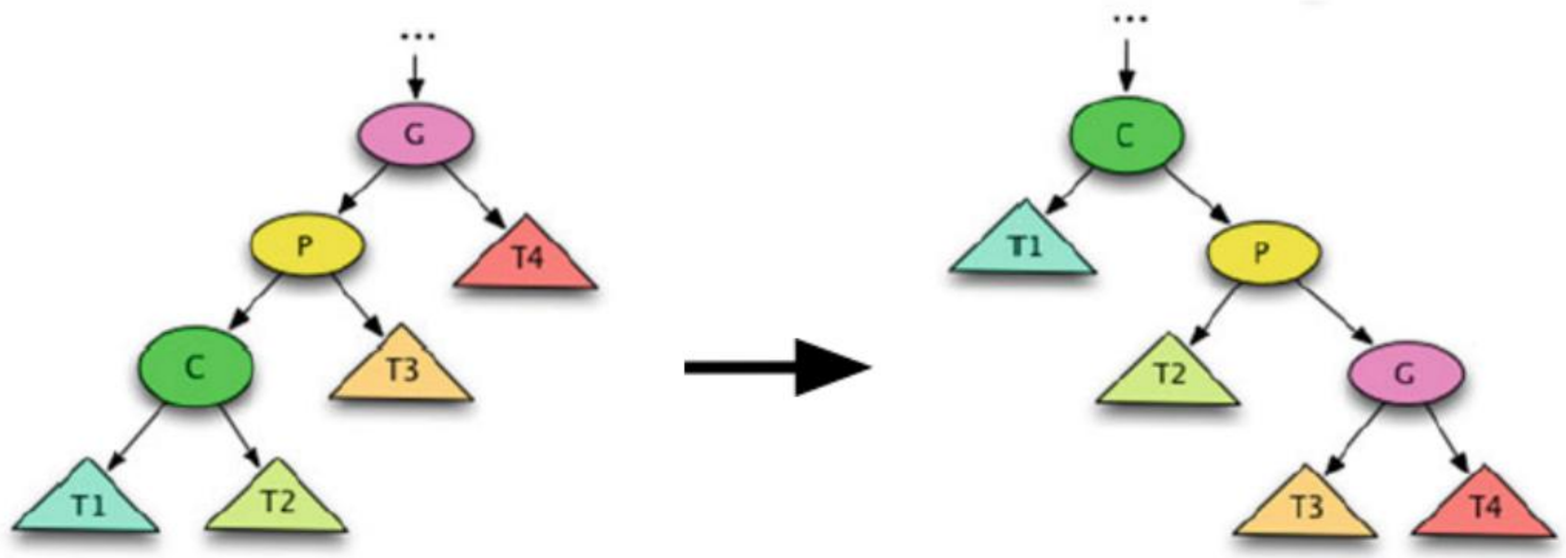
AVL Tree



B 1
h
T3
A 0
h
T2
h
T1
Rotate Left
New
A 0
B 0
h
T1
h+1
h
T3
h
T2
New
h+1
B -1
A 0
T3
h
T1
h
T2
h
New
Rotate Right
A 0
T1
h
B 0
h+1
New
T2
h
T3
h
h+1

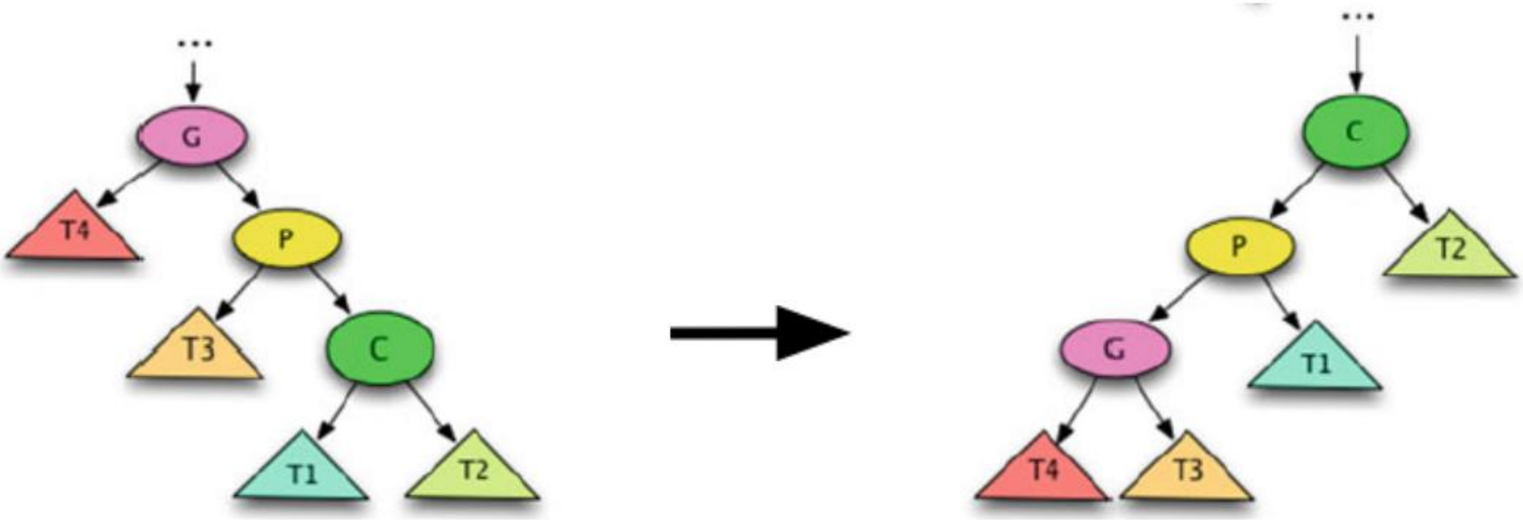
Right-Right Splaying Operation

The ***right-right splaying*** operation cyclically swaps a node C, its parent P, and its grandparent G



Left-Left Splaying Operation

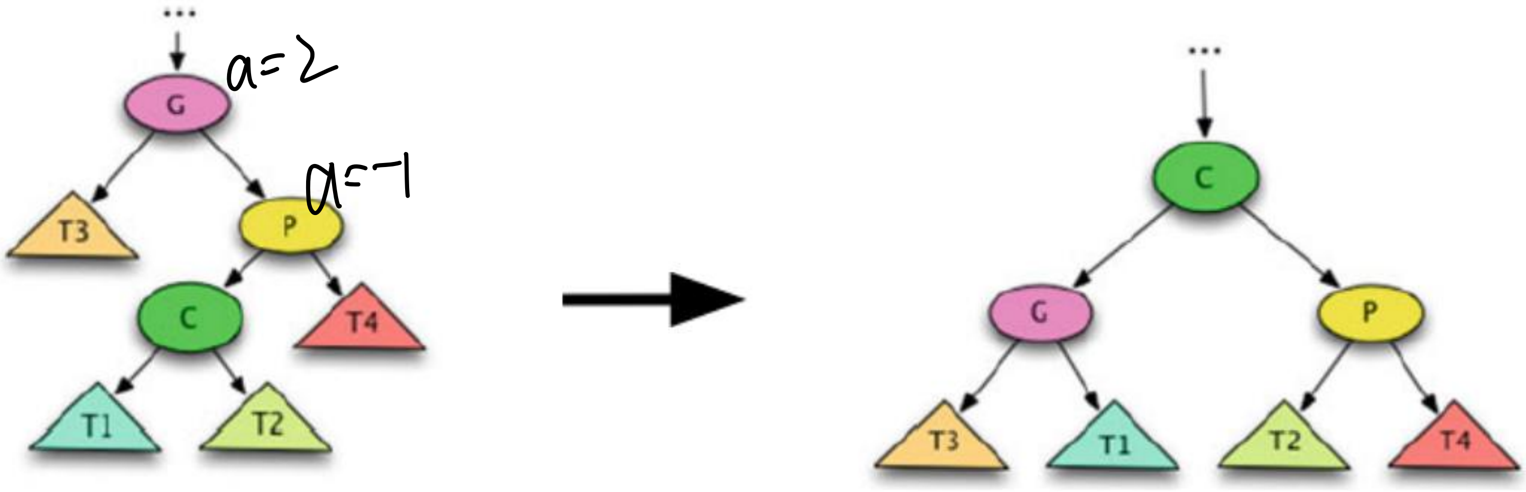
The ***left-left splaying*** operation is the inverse of the right-right splaying operation



AVL Tree

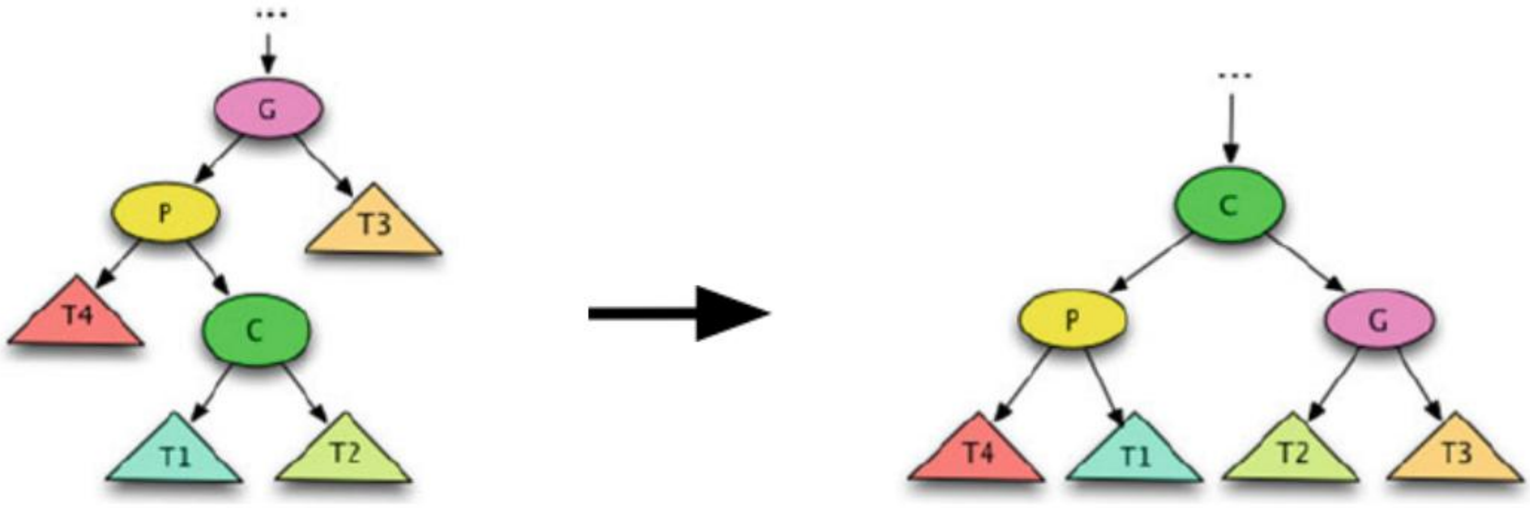
Right-Left Splaying Operation

The ***right-left splaying*** operation swaps a node C to the top and make its grandparent G and its parent P its left and right successors, respectively



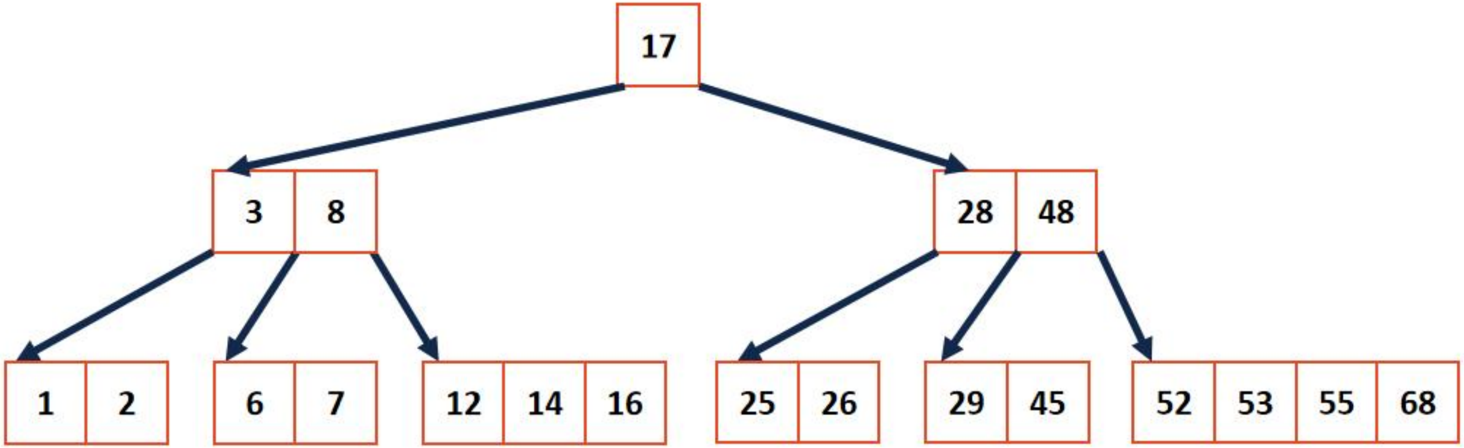
Left-Right Splaying Operation

The ***left-right splaying*** operation swaps a node C to the top and make its parent P and its grandparent G its left and right successors, respectively

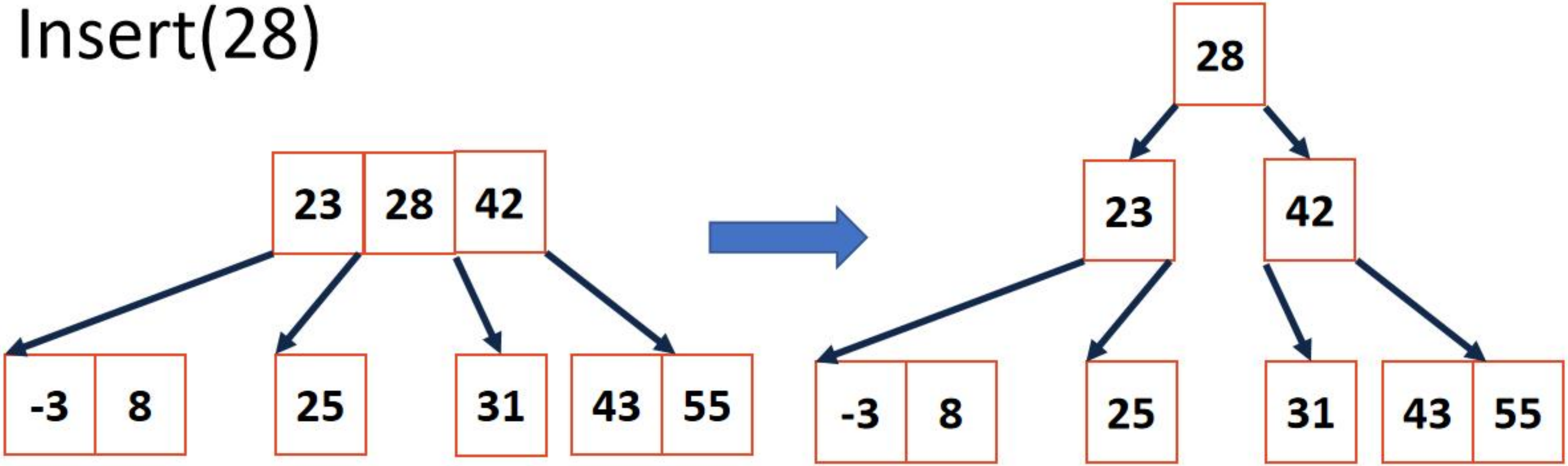


Btree Properties

- All keys within a node are ordered
- All internal nodes have exactly **one more child than keys**
- All leaves are on the same level



Insert(28)



BTree

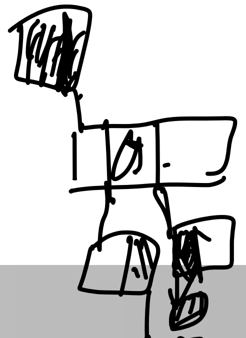
叶最左 叶最右

(i) Explain how to find the minimum and the maximum key stored in a B-tree.

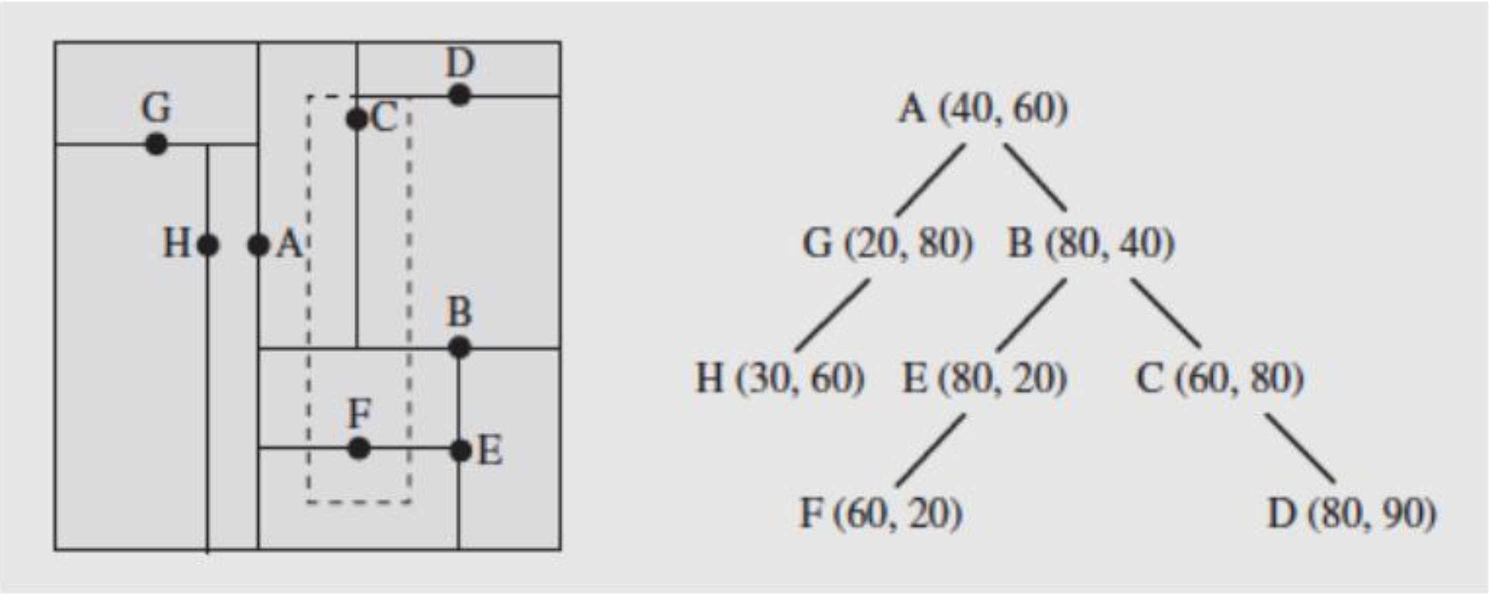
(ii) Implement operations on B-trees to return the record associated with the minimum and maximum keys, respectively. ?

(iii) Explain how to find the predecessor and the successor keys of a given key stored in a B-tree.

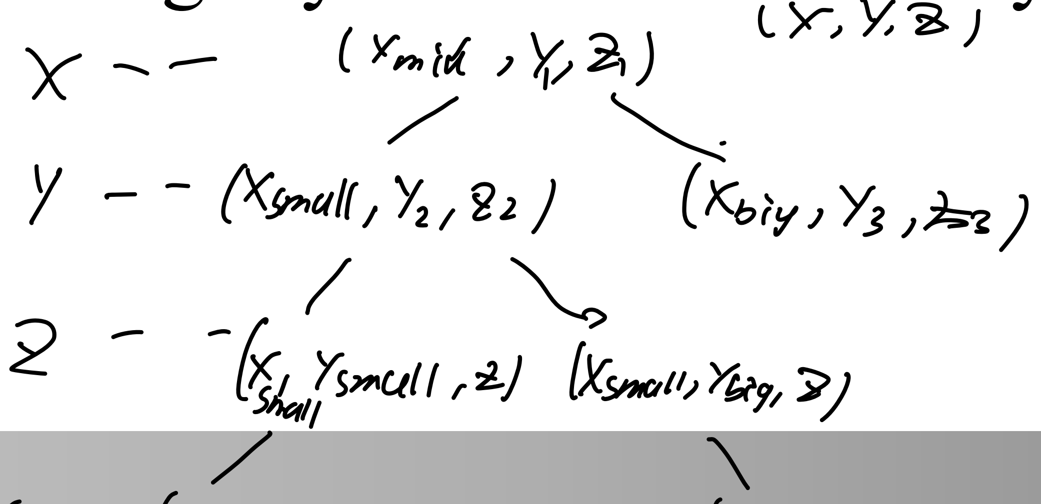
(iv) Implement operations on B-trees to return the record associated with the predecessor and the successor of a given key. ?



KD-Tree



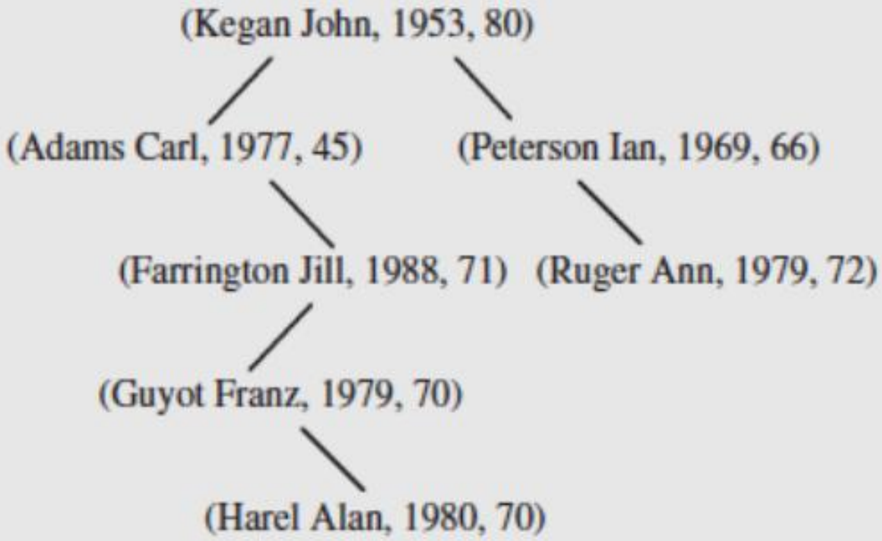
Describe in detail how to use KD tree to store a book storage system with **three** keys



KD-Tree

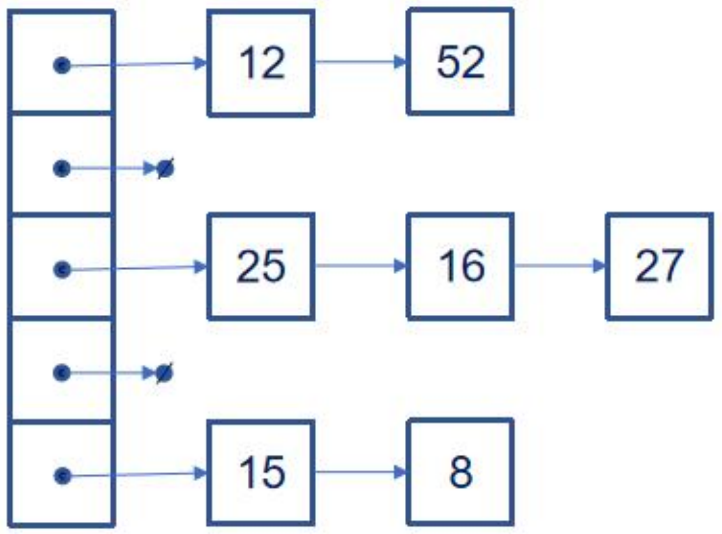
X — —(x_small, y_small, z_small (x_small, y_big, z_big)

Name	YOB	Salary (K)
Kegan, John	1953	80
Adams, Carl	1977	45
Peterson, Ian	1969	66
Farrington, Jill	1988	71
Ruger, Ann	1979	72
Guyot, Franz	1979	70
Harel, Alan	1980	70



Hashing

Open Hashing (Separate Chaining)



closed hashing

Linear Probing

Double Hashing

Analyze the advantages and disadvantages of linear probing and separate chaining in different situations

LP

advantage

~~①更容易进行序列化(serialize)操作~~

①当元素总数已知,性能更好

disadvantage

①需扩容 rehash 时间成本上升很大

②当 hash function 不良,其探到空 时间成本高

③数组存在空槽,内存浪费

④需要记录 should probe 需要空的的空间

SC advantage:

①more convenient when often add elems or delete elems as no need to rehashing

②动态分配的内存,相比于LP浪费内存更少

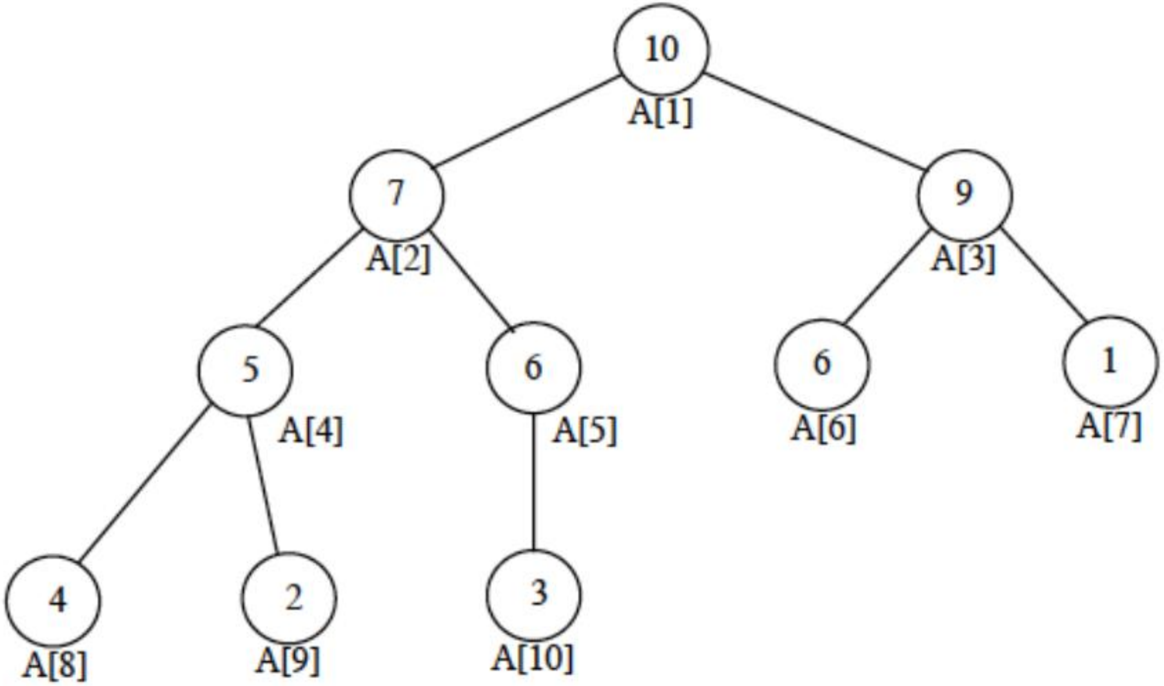
③删除元素更方便,指针操作

disadvantage:

①存储的元素动态,随机分布于内存,相比结构更紧凑的数组,hash的随机访问会带来额外的时间开销

②当不用插入删除 elems,性能更弱

Heap

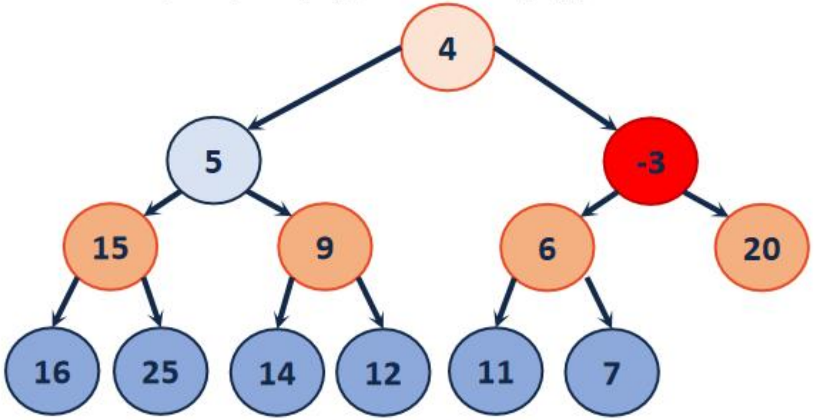


$$A = [10, 7, 9, 5, 6, 6, 1, 4, 2, 3]$$

Heap

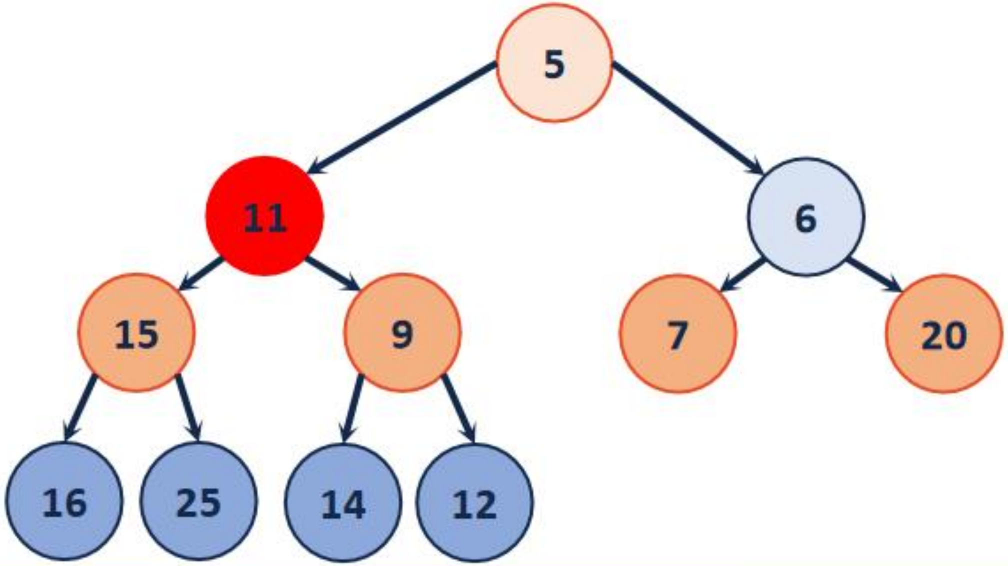
heapifyUp(i)

```
heapifyUp(i)
    if i != rootIndex && A[i] < A[parent(i)]
        swap(i, parent(i))
        heapifyUp(parent(i))
```



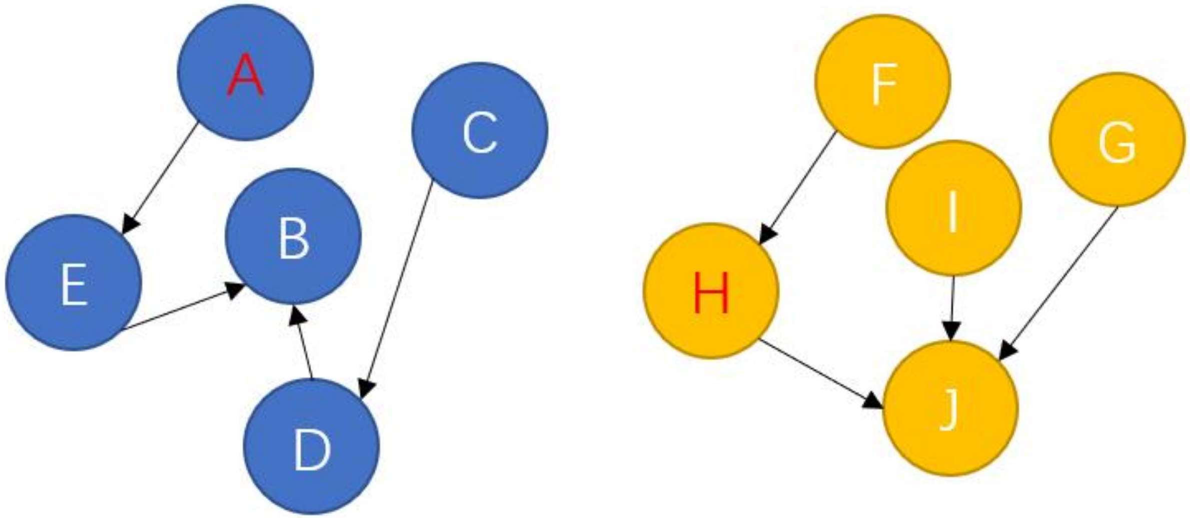
	4	5	-3	15	9	6	20	16	25	14	12	11	7

heapifyDown

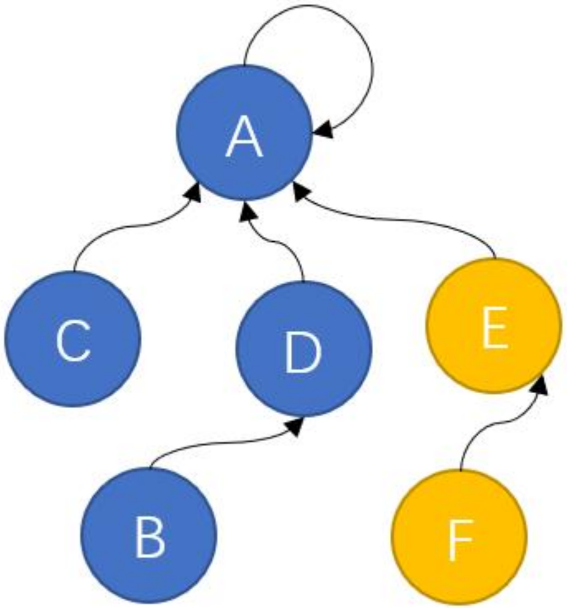


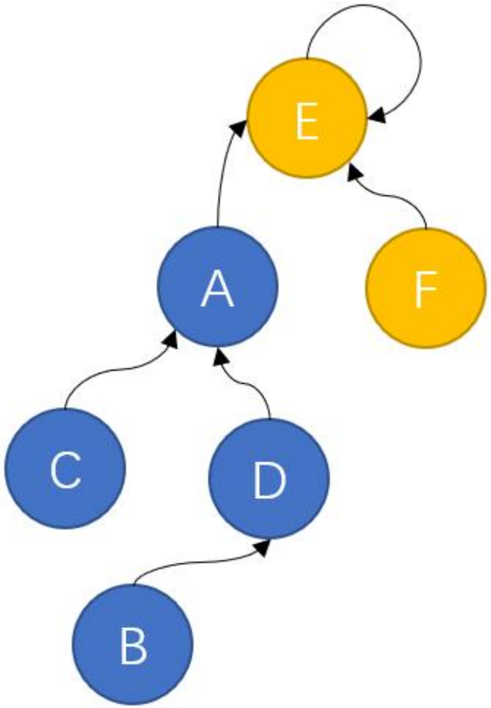
	5	11	6	15	9	7	20	16	25	14	12

Disjoint Set

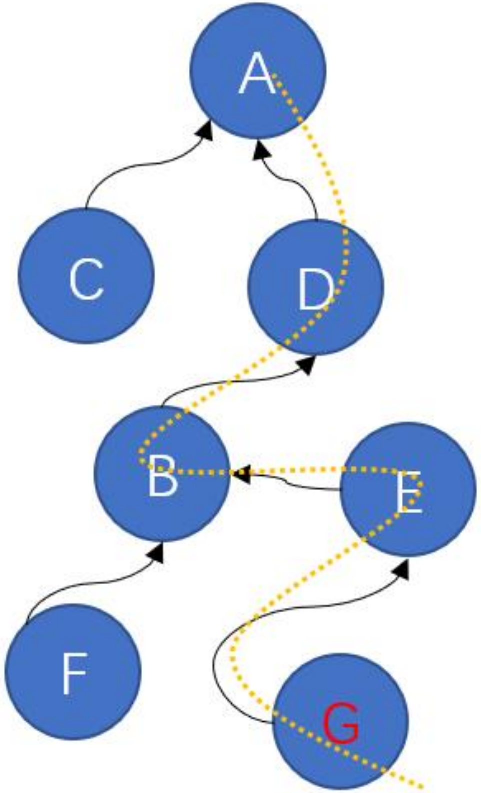


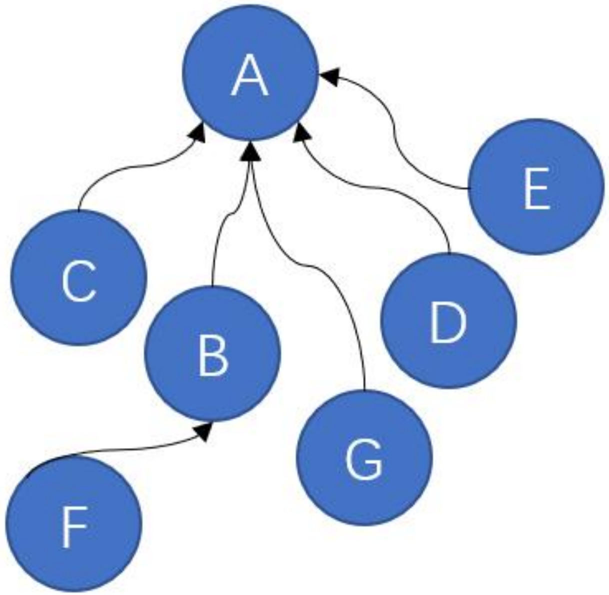
Disjoint Set





Disjoint Set





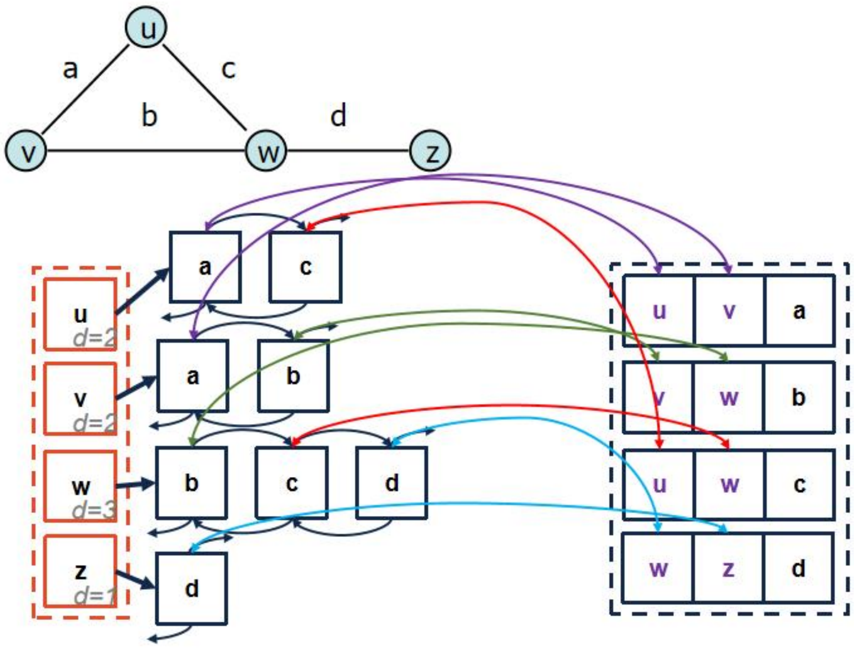
Disjoint Set

Analyze and illustrate the worst-case complexity and amortized complexity of the disjoint set when using uptree structure and path compression, respectively

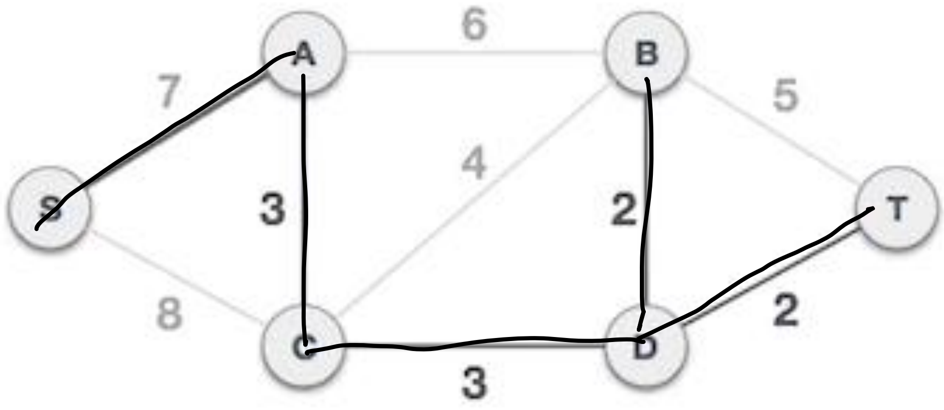
m

$\log^*$

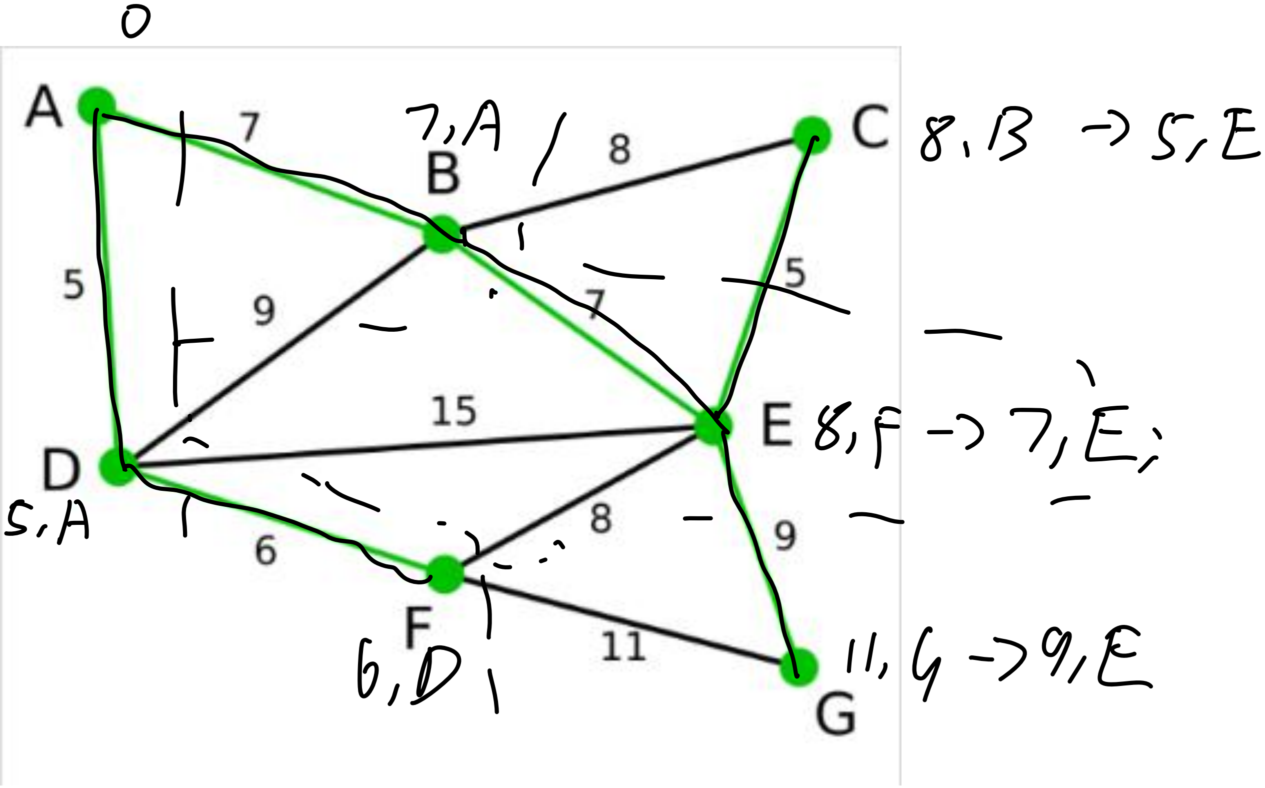
Adjacency List



MST-Kruskal



MST-Prim



MST-Prim

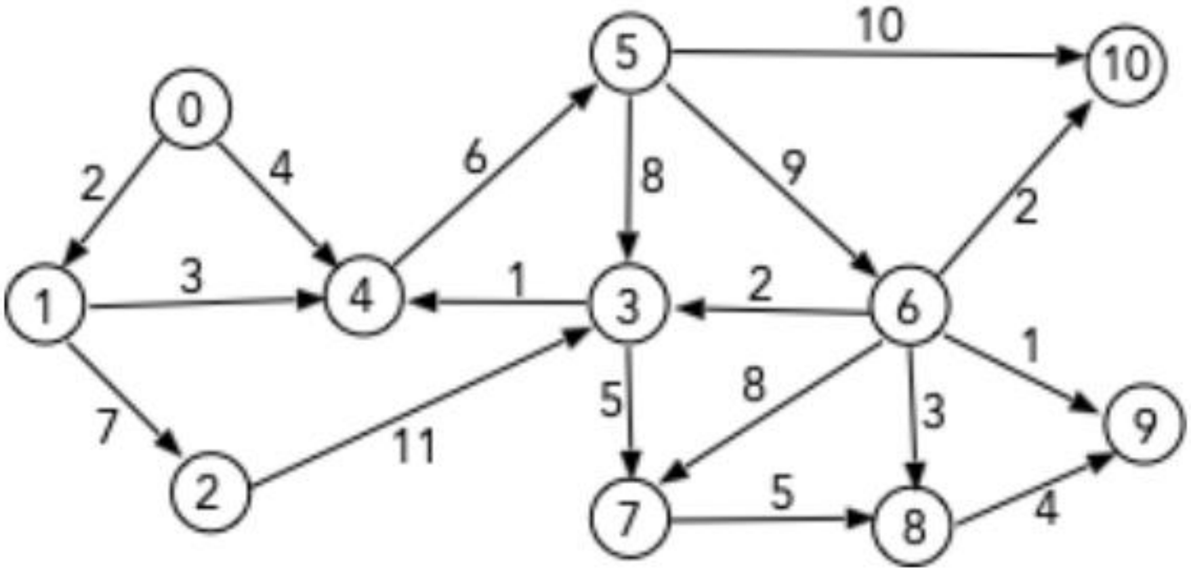
What happens in the case of Prim's and Kruskal's algorithms, if negative edge costs are permitted? Is it still sensible to talk about minimum spanning trees, if negative edge costs are permitted?

if not negative circle fine

Show that if all edge costs are pairwise different, the minimum spanning tree is uniquely defined.

Icru

Shortest Path (Dijkstra and Floyd)



Floyd:

```
For (k = 1; k <= n; k ++)
    for (I = 1; I <= n; I ++)
        for (j = 1; j <= n; j ++)
            if (d[i][k] + d[k][j] < d[i][j])
                d[i][j] = d[i][k] + d[k][j];
```

Good Luck & Have Fun

2023/05/29