Binary Search Trees Implementation

Tom Chao Zhou

CSC2100B Data Structures Tutorial 5

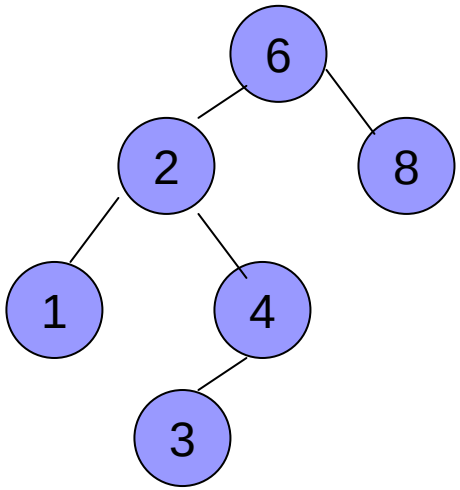
Outline

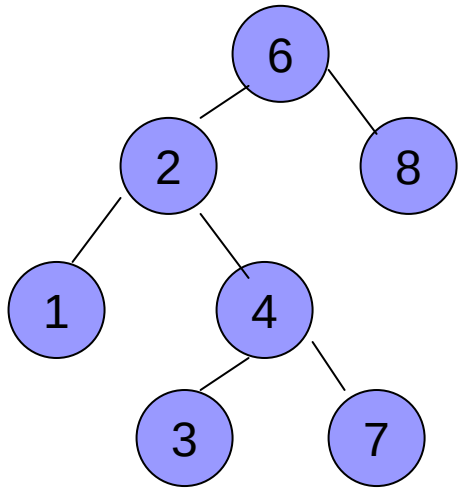
- Binary Search Trees Properties
- Implementation
  - Declaration
  - MakeEmpty
  - Find
  - FindMin
  - FindMax
  - Insert
  - Delete
  - Retrieve
- Test Example
- Appendix
  - BinarySearchTree.h
  - BinarySearchTree.c
  - TestBinarySearchTree.c

Binary Search Tree Properties

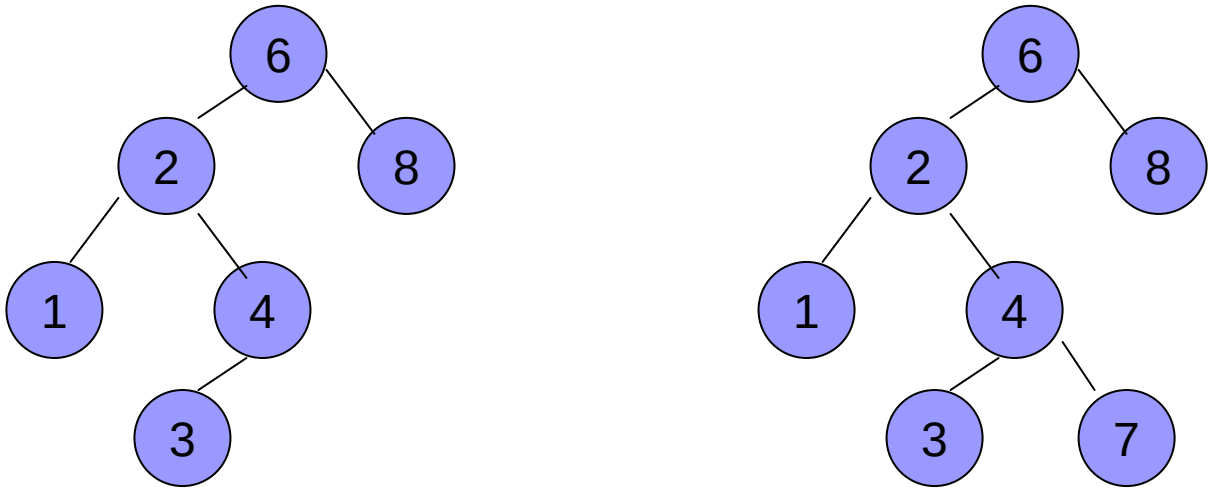
- Binary Tree -> Binary Search Tree
  - For every node T in the tree, the values of all the items in its left subtree are smaller than the item in T
  - The values of all the items in its right subtree are larger than the item in T.
  - Average depth of a binary search tree is O(logN).

Binary Search Tree Properties





Binary Search Tree Properties



Two binary trees (only the left tree is a search tree)

Implementation: Declaration (.h header file)

BinarySearchTree.h

```
typedef int ElementType;

#ifndef _BINARY_SEARCH_TREE_H
#define _BINARY_SEARCH_TREE_H
struct TreeNode;
typedef struct TreeNode* Position;
typedef struct TreeNode* SearchTree;

SearchTree MakeEmpty(SearchTree T);
Position Find(ElementType X,SearchTree T);
Position FindMin(SearchTree T);
Position FindMax(SearchTree T);
SearchTree Insert(ElementType X,SearchTree T);
SearchTree Delete(ElementType X,SearchTree T);
ElementType Retrieve(Position P);
#endif
```

Implementation: .c file

BinarySearchTree.c

```
#include<stdio.h>
#include "BinarySearchTree.h“
struct TreeNode
{
    ElementType data;
     SearchTree Left;
     SearchTree Right;
};
…//implementation of all the functions
```

Implementation: MakeEmpty

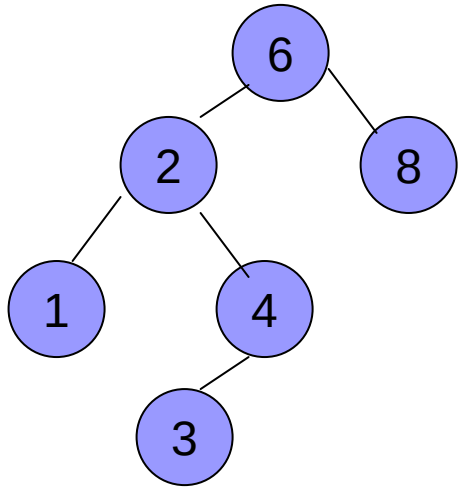
- Function:
  - Make a tree become empty.

```
//use the idea of recursion
SearchTree MakeEmpty(SearchTree T)
{
    if(T!=NULL)
     {
            MakeEmpty(T->Left);
            MakeEmpty(T->Right);
            free(T);
     }
     return NULL;
}
```

Implementation: Find

- Function:
  - find the tree node which equals to certain value

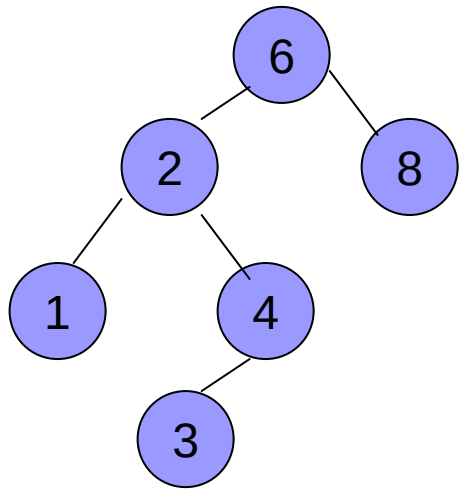
```
Position Find(ElementType X,SearchTree T)
{
    if(T==NULL)
            return NULL;
     else if(X<T->data)
            return Find(X,T->Left);
     else if(X>T->data)
            return Find(X,T->Right);
     else
            return T;
}
```



Implementation: FindMin

- Function:
  - find the minimum tree node if available.

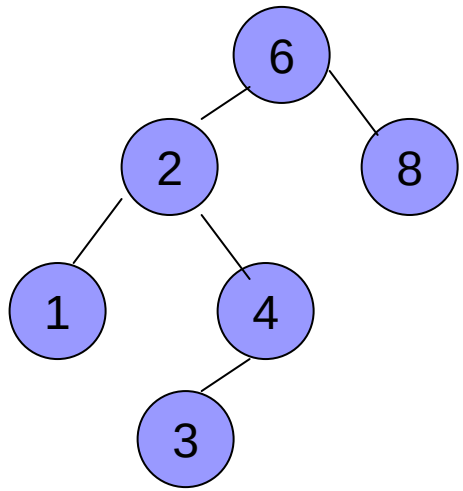
```
//use while loop
Position FindMin(SearchTree T)
{
   if(T!=NULL)
    {
        while(T->Left!=NULL)
                        T=T->Left;
    }
    return T;
}
```



Implementation: FindMax

- Function:
  - find the maximum tree node if available.

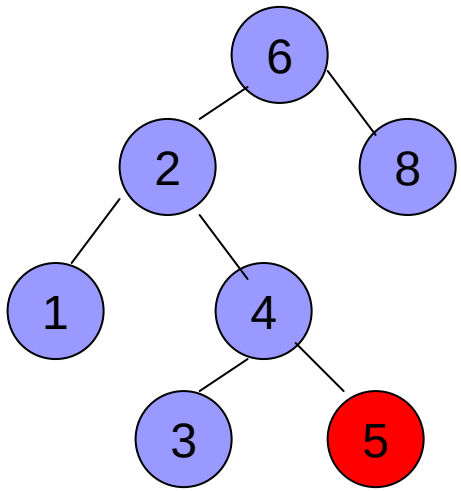
```
//use recursion
Position FindMax(SearchTree T)
{
   if(T==NULL)
            return NULL;
    else if(T->Right==NULL)
            return T;
    else
            return FindMax(T->Right);
}
```



Implementation: Insert

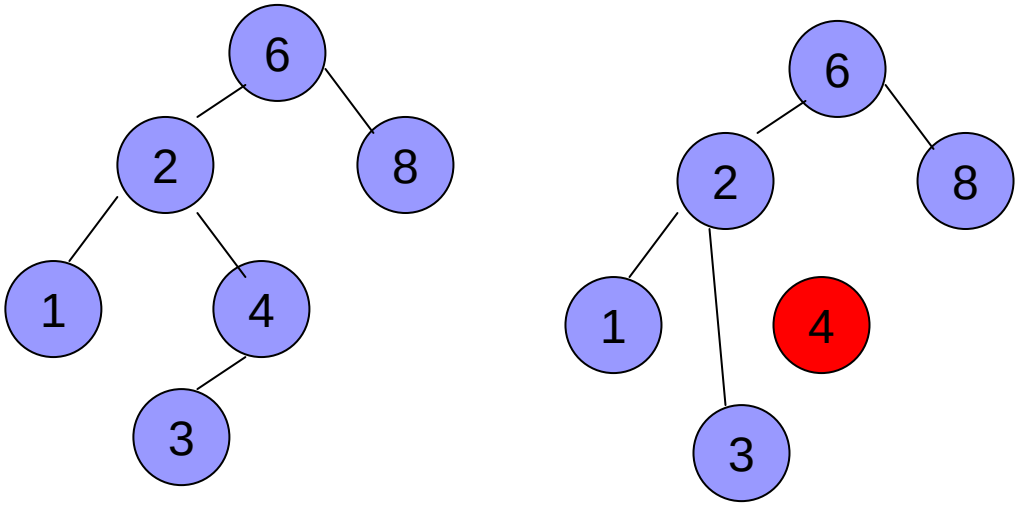
- Function:
  - Insert a data into the tree and return the new tree.

```
SearchTree Insert(ElementType X,SearchTree T)
{
   if(T==NULL)
    {
       T = (SearchTree)malloc(sizeof(struct TreeNode));
            T->data = X;
            T->Left = NULL;
            T->Right = NULL;
    }
    else if(X<T->data)
            T->Left = Insert(X,T->Left);
    else if(X>T->data)
            T->Right = Insert(X,T->Right);
    else
            ;
    return T;
}
```



Implementation: Delete

- Function:
  - Delete a tree node with certain value.



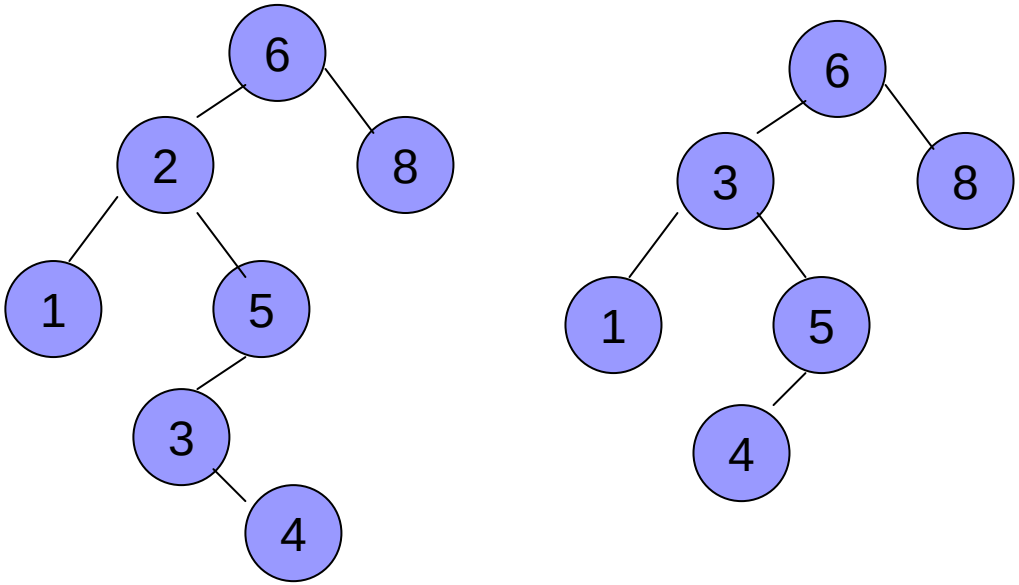
delete
node 4

•If the node is a leaf, it can be deleted directly.

•If the node has one child, the node can be deleted after its parent adjusts a link to bypass the node.

Implementation: Delete

- Function:
  - Delete a tree node with certain value.



delete
node 2

•If the node has two children, replace the data of this node with the smallest data of the right subtree and recursively delete that node.

Implementation: Delete

```
SearchTree Delete(ElementType X,SearchTree T)
{
      Position Tmp;
      if(T==NULL)
            ;
      else if(X<T->data)
            T->Left = Delete(X,T->Left);
      else if(X>T->data)
            T->Right = Delete(X,T->Right);
      else if(T->Left!=NULL && T->Right!=NULL)
      {
         Tmp = FindMin(T->Right);
            T->data = Tmp->data;
            T->Right = Delete(T->data,T->Right);
      }
      else
      {
         Tmp = T;
            if(T->Left==NULL)
                  T = T->Right;
            else if(T->Right==NULL)
                  T = T->Left;
            free(Tmp);
      }
      return T;
}
```

Implementation: Retrieve

- Function:
  - Get the data of a tree node.

```
ElementType Retrieve(Position P)
{
   return P->data;
}
```

Test Example

- TestBinarySearchTree.c

```c
#include <stdio.h>
#include "BinarySearchTree.h"

int main()
{
    SearchTree T;
    Position P;
    int i;
    int j = 0;
    T = MakeEmpty( NULL );
    for(i=0;i<10;i++)
       T = Insert(i,T);
    printf("Min: %d\n",Retrieve(FindMin(T)));
    printf("Max: %d\n",Retrieve(FindMax(T)));
    for(i=0;i<15;i++)
    {
        if(Find(i,T)!=NULL)
            printf("Find: %d\n",i);
        else
            printf("Not Found: %d\n",i);
    }
    for(i=0;i<5;i++)
        T=Delete(i,T);
    for(i=0;i<10;i++)
    {
        if(Find(i,T)!=NULL)
            printf("Find: %d\n",i);
        else
            printf("Not Found: %d\n",i);
    }
    return 0;
}
```

Appendix

- BinarySearchTree.h

```
typedef int ElementType;

#ifndef _BINARY_SEARCH_TREE_H
#define _BINARY_SEARCH_TREE_H

struct TreeNode;
typedef struct TreeNode* Position;
typedef struct TreeNode* SearchTree;

SearchTree MakeEmpty(SearchTree T);
Position Find(ElementType X,SearchTree T);
Position FindMin(SearchTree T);
Position FindMax(SearchTree T);
SearchTree Insert(ElementType X,SearchTree T);
SearchTree Delete(ElementType X,SearchTree T);
ElementType Retrieve(Position P);

#endif
```

```c
#include<stdio.h>
#include "BinarySearchTree.h"

struct TreeNode
{
    ElementType data;
    SearchTree Left;
    SearchTree Right;
};

SearchTree MakeEmpty(SearchTree T)
{
    if(T!=NULL)
    {
        MakeEmpty(T->Left);
        MakeEmpty(T->Right);
        free(T);
    }
    return NULL;
}
Position Find(ElementType X,SearchTree T)
{
    if(T==NULL)
        return NULL;
    else if(X<T->data)
        return Find(X,T->Left);
    else if(X>T->data)
        return Find(X,T->Right);
    else
        return T;
}
Position FindMin(SearchTree T)
{
    if(T!=NULL)
    {
        while(T->Left!=NULL)
            T=T->Left;
    }
    return T;
}
```

```c
Position FindMax(SearchTree T)
{
    if(T==NULL)
        return NULL;
    else if(T->Right==NULL)
        return T;
    else
        return FindMax(T->Right);
}
SearchTree Insert(ElementType X,SearchTree T)
{
    if(T==NULL)
    {
        T = (SearchTree)malloc(sizeof(struct TreeNode));
        T->data = X;
        T->Left = NULL;
        T->Right = NULL;
    }
    else if(X<T->data)
        T->Left = Insert(X,T->Left);
    else if(X>T->data)
        T->Right = Insert(X,T->Right);
    else
        ;
    return T;
}
```

```c
SearchTree Delete(ElementType X,SearchTree T)
{
    Position Tmp;
    if(T==NULL)
        ;
    else if(X<T->data)
        T->Left = Delete(X,T->Left);
    else if(X>T->data)
        T->Right = Delete(X,T->Right);
    else if(T->Left!=NULL && T->Right!=NULL)
    {
        Tmp = FindMin(T->Right);
        T->data = Tmp->data;
        T->Right = Delete(T->data,T->Right);
    }
    else
    {
        Tmp = T;
        if(T->Left==NULL)
            T = T->Right;
        else if(T->Right==NULL)
            T = T->Left;
        free(Tmp);
    }
    return T;
}
ElementType Retrieve(Position P)
{
    return P->data;
}
```

BinarySearchTree.c

TestBinarySearchTree.c

```c
#include <stdio.h>
#include "BinarySearchTree.h"

int main()
{
    SearchTree T;
    Position P;
    int i;
    int j = 0;
    T = MakeEmpty( NULL );
    for(i=0;i<10;i++)
       T = Insert(i,T);
    printf("Min: %d\n",Retrieve(FindMin(T)));
    printf("Max: %d\n",Retrieve(FindMax(T)));
    for(i=0;i<15;i++)
    {
        if(Find(i,T)!=NULL)
            printf("Find: %d\n",i);
        else
            printf("Not Found: %d\n",i);
    }
    for(i=0;i<5;i++)
        T=Delete(i,T);
    for(i=0;i<10;i++)
    {
        if(Find(i,T)!=NULL)
            printf("Find: %d\n",i);
        else
            printf("Not Found: %d\n",i);
    }
    return 0;
}
```

?