

CH1 Time Complexity

CH2 CH4 Array & Linked List

CH3 Stack & Queue

CH5 Tree & Binary Tree

- Tree
 - ancestor=predecessor
 - descendant=successor
 - tree化成binary tree, binary tree化成tree
 - tree化成binary
 - Leftmost-child-Next-Right-sibling
 - Forest化成binary tree, binary tree化成Forest
 - 皆針對Root做操作
- Binary Tree
 - ith level max node= 2^{i-1}
 - height h max node= $2^h - 1$
 - leaf num= n_0 , degree-2= n_2 , $n_0 = n_2 + 1$
 - 不可決定唯一binary tree
 1. preorder+postorder
 2. level-order+preorder
 3. level-order+postorder
 4. BST+inorder
 - the number of different binary trees with n nodes
 - Catalan number
 - $\frac{1}{n+1} \binom{2n}{n}$
- Binary Search Tree
 - In a BST find i-th smallest data

```
struct Node {  
    Node* Lchild;  
    int data;  
    int Lsize;  
    Node* Rchild;  
};  
  
search(T:BST, i:int){//在T中找出i-th小之data  
    if(T!=Nil){
```

```

k=(T->Lsize)+1;//代表root是kth小的数据
if(i==k)
    return T->Data;
else if(i<k)
    return serach(T->Lchild,i);//去左子樹找i-th小
else
    return search(T->Rchild,i-k);//去右子樹找(i-k)th小
}
}

```

- Heap

- build a heap with n nodes

- Top-Down
 - $O(n \log n)$
 - Bottom-Up
 - $O(n)$

- Heapify[adjust(tree,i,n)]

```

void adjust(int tree[], int i, int n){
    //調整以i node no.為root之子樹成為Heap
    int j=2*i;//目前j是i之左子點No.
    int x=tree[i];
    while(j<=n){//尚有兒子
        if(j<n && tree[j]<tree[j+1])
            j=j+1;
        if(x>=tree[j])
            break;
        else{
            tree[j/2]=tree[j];//上移至父點
            j=2*j;//新的左子點位置
        }
    }
    tree[j/2]=x;//x置入正確格子中
}
void buildheap(int tree[], int n){
    for(int i=n/2;i>=1;i--)
        adjust(tree, i, n);
}

```

- Disjoin Sets

- Union
 - Find

- Thread Binary Tree

CH9 Advanced Tree

-
- Double-Ended Priority Queue
 - Min-Max Heap
 - Deap
 - SMMH
 - Extended Binary Tree
 - E=I+2N
 - Huffman Algorithm
 - AVL Tree
 - M-way search tree
 - B Tree of order m
 - B⁺ Tree of order m
 - Red-Blcak tree
 - Optimal Binay Search Tree(OBST)
 - Splay Tree
 - Leftist Heap
 - Binomail Heap
 - Fibonacci Heap

CH7 Sort

- Search
 - Linear Search
 - Binary Search
- Sort
 - Elementary/Simple Sorts
 - Insertion sort
 - Selection sort
 - Bubble sort
 - Shell sort
 - Advanced/Efficient Sorts
 - Quick sort
 - Merge sort
 - Heap sort
 - Linear-Time sorting methods
 - LSD Radix sort=Radix sort
 - MSD Radix sort=Bucket sort
 - Counting sort

CH8 Hashing

- Collision

- Overflow
- Identifier Density
- Loading Density
- Hashing 優點
- hashing function design
 - 3 design criteria
 - 計算簡單
 - 碰撞少
 - perfect hashing function
 - 不要造成hash table局部偏重儲存的情形
 - uniform hashing function
 - 常見hashing function design methods
 - Middle Square
 - Mod(Division)
 - Folding Addition
 - Digits Analysis
- Overflow Handling
 - Linear Probing
 - Quadratic Probing
 - Double Hashing
 - Chaining
 - Rehashing

CH6 Graph

- DFS
- BFS
- Topological sort
- Minimum Spanning Tree
 - Kruskal's algorithm
 - Prim's algorithm
 - Sollin's algorithm
- Shortest Path Length
 - single source to other destinations
 - Directed Acyclic Graph(DAG)
 - Dijkstra algorithm
 - Bellman-Ford Algorithm

- all pairs of vertex
 - Floyd-Warshall algorithm
 - Johnson's algorithm
- AOE network
- Articulation Point
- Biconnected Graph
- Biconnected component