

1

□□□□□□□□□□□□□□□□

--	--	--	--	--	--

- [illegible]

```
void bfs(Node *root) //广度优先遍历
{
    head = tail = 0;
    Queue[tail++] = root;
    while(head<tail)
    {
        Node *node = Queue[head];
        cout << node->key << endl;
        if(node->lchild) Queue[tail++] = node->lchild;
        if(node->rchild) Queue[tail++] = node->rchild;
        head++;
    }
    return;
}
```

□□□□□□

- [illegible]

```
void dfs(Node *root) //递归
{
    if(root == NULL) return;
    int start,end;
    tot += 1;
    start = tot;
    if(root->lchild) dfs(root->lchild); //递归左子树
    if(root->rchild) dfs(root->rchild); //递归右子树
    tot += 1; //右子树遍历结束
    end = tot;
    cout << root->key << endl;
    return;
}
```

□□□□□

- [illegible]

- 访问第 $2*i$ 个元素
- 访问第 $2*i+1$ 个元素

## 2. 二叉树的遍历

- 前序遍历
- 中序遍历

- 二叉树

根节点	左孩子	右孩子
左孩子	左孩子	右孩子

- 二叉树的遍历
- 二叉树的遍历

二叉树

二叉树的遍历

- 二叉树的遍历
- 二叉树的遍历
  - 二叉树的遍历
  - 二叉树的遍历

```
#include<bits/stdc++.h>
using namespace std;
void build_inorder_thread(Node *root)
{
    if(root == NULL) return ;
    if(root->ltag == 0) build_inorder_thread(root->lchild);
    if(inorder_root == NULL) inorder_root = root; //root
    访问第inorder_root个元素
    if(root->lchild == NULL) //空
    {
        root->lchild = prenode;
        root->ltag = 1;
    }
    if(prenode && prenode->rchild == NULL) //空
    {
        prenode->rchild = root;
        prenode->rtag = 1;
    }
    prenode = root; //prenode指向当前节点
    访问
    if(root->rtag == 0) build_inorder_thread(root->rchild);
    return;
}

void __build_inorder_thread(Node *root)
{
    build_inorder_thread(root);
    prenode->rchild = NULL; //prenode指向当前节点
    prenode->rtag = 1;
```

```

        return;
    }

    Node *getnext(Node *root)
    {
        if(root->rtag == 1) return root->rchild; //右孩子
        //右孩子不存在
        root = root->rchild; //右孩子为root
        //右孩子为root
        //右孩子为root
        while(root->ltag == 0 && root->lchild) //左孩子存在
        {
            root = root->lchild;
        }
        return root;
    }

    int main()
    {
        Node *node = inorder_root; //中序遍历的根节点
        while(node)
        {
            cout << node->key << " ";
            node = getnext(node);
        }
        clear(root);
        return 0;
    }

```

中序遍历的根节点为root，左孩子为root->lchild，右孩子为root->rchild

中序遍历

中序遍历

```

#include<bits/stdc++.h>
using namespace std;
#define KEY(n) (n ? n->key : -1) //中序遍历

typedef struct Node
{
    int key;
    struct Node *lchild,*rchild;
}Node;

Node *getnewnode(int key)
{
    Node *p = new Node;
    p->key = key;
    p->lchild = p->rchild = NULL;
    return p;
}

```

```

}

void clear(Node *root)
{
    if(root == NULL) return ;
    clear(root->lchild);
    clear(root->rchild);
    delete root;
    return ;
}

Node *insert(Node *root,int key)
{
    if(root == NULL) return getnewnode(key);
    if(rand()%2) root->lchild = insert(root->lchild,key);
    else root->rchild = insert(root->rchild,key);
    return root;
}

Node *getrandbinarytree(int n)
{
    Node *root =NULL;
    for(int i = 0;i < n;i++)
    {
        root = insert(root,rand() % 100);
    }
    return root;
}

char buff[1000];
int len = 0; //[]

void __serialize(Node *root) //[]
{
    if(root == NULL) return;
    len += sprintf(buff + len ,"%d",root->key);//sprintf[]
    if(root->lchild == NULL && root->rchild == NULL) return;
    len += sprintf(buff + len , "(");
    __serialize(root->lchild);
    if(root->rchild)
    {
        len += sprintf(buff + len , ",");
        __serialize(root->rchild);
    }
    len += sprintf(buff + len , ")");
    return ;
}

void serialize(Node *root)
{
    memset(buff,0,sizeof(buff));
    len = 0;

```



```

Node *getnewnode(int key)
{
    Node *p = new Node;
    p->key = key;
    p->lchild = p->rchild = NULL;
    return p;
}

void clear(Node *root)
{
    if(root == NULL) return ;
    clear(root->lchild);
    clear(root->rchild);
    delete root;
    return ;
}

Node *insert(Node *root,int key)
{
    if(root == NULL) return getnewnode(key);
    if(rand()%2) root->lchild = insert(root->lchild,key);
    else root->rchild = insert(root->rchild,key);
    return root;
}

Node *getrandrinarytree(int n)
{
    Node *root =NULL;
    for(int i = 0;i < n;i++)
    {
        root = insert(root,rand() % 100);
    }
    return root;
}

char buff[1000];
int len = 0; //[]

void __serialize(Node *root) //[]
{
    if(root == NULL) return;
    len += sprintf(buff + len ,"%d",root->key);//sprintf[]
    if(root->lchild == NULL && root->rchild == NULL) return;
    len += sprintf(buff + len , "(");
    __serialize(root->lchild);
    if(root->rchild)
    {
        len += sprintf(buff + len , ",");
        __serialize(root->rchild);
    }
    len += sprintf(buff + len , ")");
    return ;
}

```

```
}
```

```
void serialize(Node *root)
{
    memset(buff,0,sizeof(buff));
    len = 0;
    __serialize(root);
    return;
}
```

```
Node *deserialize(char *buff,int n)
{
    Node **s = (Node **)malloc(sizeof(Node *) * 100);
    int top = -1,flag = 0,scode = 0; //[] [] [] [] [] [] [] [] [] []
    Node *p = NULL , *root =NULL;
    for(int i = 0 ; buff[i];i++)
    {
        switch(scode)
        {
            case 0:
            {
                if(buff[i] >= '0' && buff[i] <= '9') scode = 1;
                else if(buff[i] == '(') scode = 2;
                else if(buff[i] == ',') scode = 3;
                else scode = 4;
                i -= 1;
            }
            break;
            case 1:
            {
                int num = 0;
                while(buff[i] <= '9' && buff[i] >= '0')
                {
                    num = num * 10 +(buff[i] - '0');
                    i += 1;
                }
                p = getnewnode(num);
                if(top >= 0 && flag == 0) s[top]->lchild = p;
                if(top >= 0 && flag == 1) s[top]->rchild = p;
                i -= 1; //[] num[] i [] [] [] [] [] [] [] [] [] i+1 [] [] [] [] [] [] i-1
                scode = 0;
            }
            break;
            case 2:
            {
                s[++top] = p;
                flag = 0;
                scode = 0;
            }
        }
    }
}
```

```

        break;
    case 3:
    {
        flag = 1;
        scode = 0;
    }
    break;
    case 4:
    {
        root = s[top--];
        scode = 0;
    }
    break;
    }
}
return root;
}

void print(Node *node)
{
    printf("%d(%d,%d)\n",KEY(node),KEY(node->lchild),KEY(node->rchild));
    return ;
}

void output(Node *root)
{
    if(root == NULL) return;
    print(root);
    output(root->lchild);
    output(root->rchild);
    return;
}

int main()
{
    srand((unsigned)time(NULL));
    #define n 10
    Node *root = getrandrbinarytree(n);
    serialize(root);
    output(root);
    cout << buff << " " <<"□□□";
    Node *new_root = deserialize(buff,len);
    output(new_root);
    clear(root);
    return 0;
}

```

□□□□

 alt text □□□□□□□□□□□□

```

#include<bits/stdc++.h>
using namespace std;
typedef struct Node
{
    int freq;
    char ch;
    struct Node *lchild,*rchild;
}Node;

Node *getnewnode(int freq,char ch)
{
    Node *p = new Node;
    p->ch = ch;
    p->freq = freq;
    p->lchild = p->rchild = NULL;
    return p;
}

void swap_node(Node **node_arr,int i,int j)
{
    Node *temp = node_arr[i];
    node_arr[i] = node_arr[j];
    node_arr[j] = temp;
    return;
}

int find_min_node(Node **node_arr,int n)
{
    int ind = 0;
    for(int j = 1;j <= n;j++)
    {
        if(node_arr[ind]->freq > node_arr[j]->freq) ind = j;
    }
    return ind;
}

// Huffman Tree Construction
Node *buildhaffmantree(Node **node_arr,int n)
{
    for(int i = 1;i < n;i++)
    {
        int ind1 = find_min_node(node_arr,n - i);
        swap_node(node_arr,ind1,n-i); // Huffman Tree Construction
        int ind2 = find_min_node(node_arr,n - i - 1);
        swap_node(node_arr ,ind2,n - i - 1);
        int freq = node_arr[n - i]->freq + node_arr[n - i - 1]->freq;
        Node *node = getnewnode(freq , 0);
        node->lchild = node_arr[n - i];
        node->rchild = node_arr[n - i - 1];
        node_arr[n - i - 1] = node;
    }
}

```

```

        return node_arr[0];
    }


void extracthaffmancode(Node *root ,char buff[],int k)
{
    buff[k] = 0;
    if(root->lchild == NULL && root->rchild == NULL)
    {
        cout << root->ch << buff << endl;
        return ;
    }
    buff[k] = '0';
    extracthaffmancode(root->lchild,buff,k+1);
    buff[k] = '1';
    extracthaffmancode(root->rchild,buff,k+1);
    return ;
}

void clear(Node *root)
{
    if(root == NULL) return ;
    clear(root->lchild);
    clear(root->rchild);
    delete root;
    return ;
}

int main()
{
    int n,freq;
    char s[10];
    cin >> n;
    Node **node_arr = new Node *[n];
    for(int i = 0;i < n;i++)
    {
        cin >> s >> freq;
        node_arr[i] = getnewnode(freq,s[0]);
    }
    Node *root = buildhaffmantree(node_arr,n);
    char buff[1000];
    extracthaffmancode(root,buff,0);
    clear(root);
    return 0;
}

```

**n**□□□□□

 alt text /\* // Definition for a Node. class Node { public: int val; vector<Node\*> children;

```

Node() {}

Node(int _val) {
    val = _val;
}

Node(int _val, vector<Node*> _children) {
    val = _val;
    children = _children;
}

};
*/

class Solution {
public:
    vector<int> preorder(Node* root) {
        if(root == NULL) return vector<int>();
        vector<int> ans;
        ans.push_back(root->val);
        for(auto x: root->children)
        {
            vector<int> temp = preorder(x);
            for(auto y : temp) ans.push_back(y);
        }
        return ans;
    }
};

```

□□□□□□□□□□□□□□

 alt text

- □□□□□□
- □□□□□□□□□□□□□□□□□□
  - □□□□□□□□□□□□□□□□
  - □□□□□□

```

class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        if(preorder.size() == 0) return NULL;
        int pos = 0;
        while(inorder[pos] != preorder[0]) pos += 1;
        TreeNode *root = new TreeNode(preorder[0]);
        vector<int> pre , in;
        for(int i = 1; i <= pos; i++) pre.push_back(preorder[i]);
        for(int i = 0; i <= pos - 1; i++) in.push_back(inorder[i]);
        root->left = buildTree(pre, in);
        pre.clear();
        in.clear();
    }
};

```

```

        for(int i = pos + 1; i < preorder.size(); i++)
        {
            pre.push_back(preorder[i]);
            in.push_back(inorder[i]);
        }
        root->right = buildTree(pre, in);
        return root;
    }
};

```

□□□□□□□□

 alt text

11

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        if(root == NULL) return vector<vector<int>>();
        TreeNode *node;
        queue<TreeNode *> q;
        q.push(root);
        vector<vector<int>> ans;
        while(!q.empty())
        {
            int cnt = q.size();
            vector<int> temp;
            for(int i = 0;i < cnt;i++)
            {
                node = q.front();
                temp.push_back(node->val);
                if(node->left) q.push(node->left);
                if(node->right) q.push(node->right);
                q.pop();
            }
            ans.push_back(temp);
        }
        return ans;
    }
};
```

11

- 
- 

```
class Solution {
    void dfs(TreeNode *root,int k,vector<vector<int>> &ans)
    {
        if(root == NULL) return ;
    }
};
```

```

        if(k == ans.size()) ans.push_back(vector<int>());
        ans[k].push_back(root->val);
        dfs(root->left,k+1,ans);
        dfs(root->right,k+1,ans);
        return ;
    }
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        dfs(root,0,ans);
        return ans;
    }
};

```

□□□□□

 alt text


- □□□□□□ □□□□□□□□□□□□ □□□□□□□□□□
- □□□□c++□□□swap□□□□□□□□□□□□□□

```

class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        if(root == NULL) return NULL;
        swap(root->right,root->left);
        invertTree(root->left);
        invertTree(root->right);
        return root;
    }
};

```

□□□□□□□

 alt text

```

class Solution {
public:
    void dfs(TreeNode *root,int k,vector<vector<int>> &ans)
    {
        if(root == NULL) return;
        if(k == ans.size()) ans.push_back(vector<int>());
        ans[k].push_back(root->val);
        dfs(root->left,k+1,ans);
        dfs(root->right,k+1,ans);
        return ;
    }
    vector<vector<int>> levelOrderBottom(TreeNode* root){
        vector<vector<int>> ans;
        dfs(root,0,ans);
    }
};


```

```

        for(int i = 0,j = ans.size()-1;i < j;i++,j--)
        {
            swap(ans[i],ans[j]);
        }
        return ans;
    }
};

```

□□□□□□□□□□

alt text

```
class Solution {
public:
    void dfs(TreeNode *root,int k,vector<vector<int>> &ans)
    {
        if(root == NULL) return;
        if(k == ans.size()) ans.push_back(vector<int>());
        ans[k].push_back(root->val);
        dfs(root->left,k+1,ans);
        dfs(root->right,k+1,ans);
        return ;
    }
    vector<vector<int>> zigzagLevelOrder(TreeNode* root){
        vector<vector<int>> ans;
        dfs(root,0,ans);
        for(int k = 1;k < ans.size();k+=2)
        {
            for(int i = 0,j = ans[k].size()-1;i < j;i++,j--)
                swap(ans[k][i],ans[k][j]);
        }
        return ans;
    }
};
```

1111

 alt text   alt text

- 000000000000000000000000
- 000000000000
- 00000set[0] 000pair[0]000000
- 00n-1

```
#include<bits/stdc++.h> //oj287
using namespace std;
typedef pair<int,int> PII;
int main()
```



```
    return 0;  
}
```