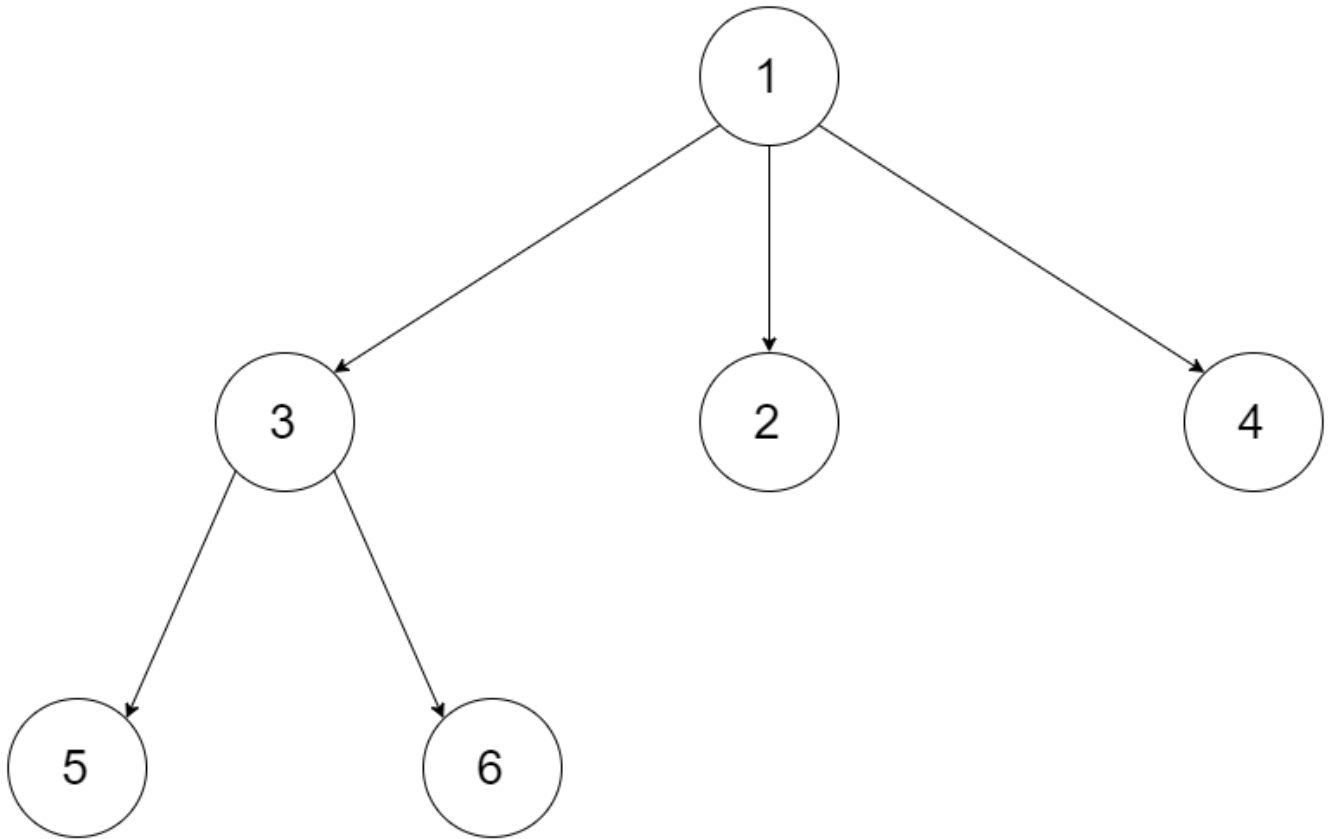


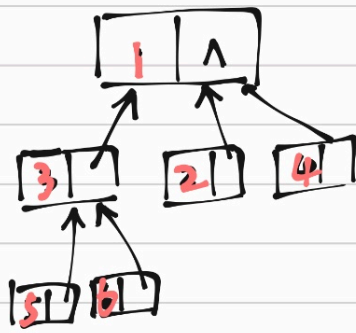
多叉树 (N叉树)

创建多叉树

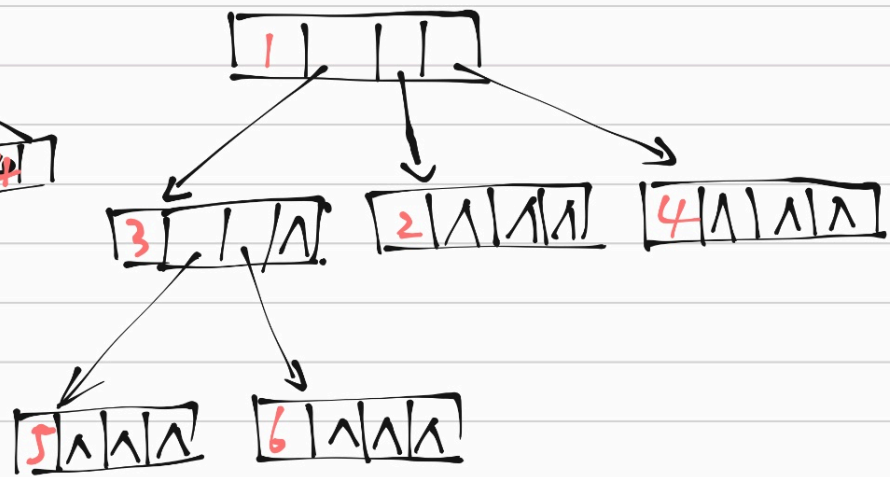


多叉树的存储方式

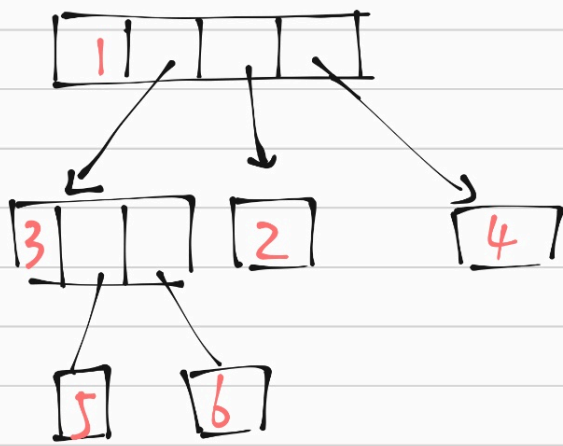
1. Father 链接结构 (缺点是不好遍历)
2. 节点大小规定的链接结构
3. 节点大小不固定的链接结构
4. 孩子链结构 (缺点是访问父节点)
5. 父亲-孩子链结构 (4的改进)
6. 左孩子右兄弟链接结构 (优点是跟二叉树的结构一样, 将节点的最左儿子当左节点, 最大右兄弟当右节点)



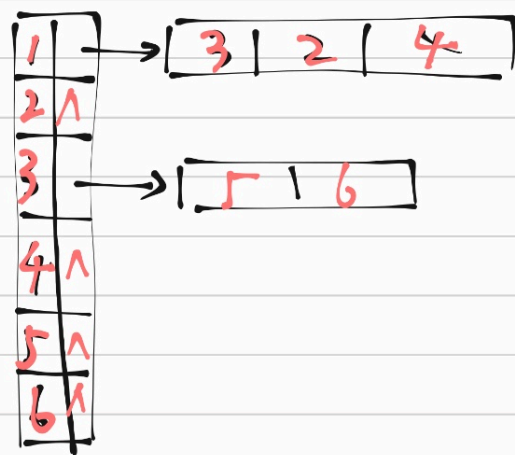
(1)



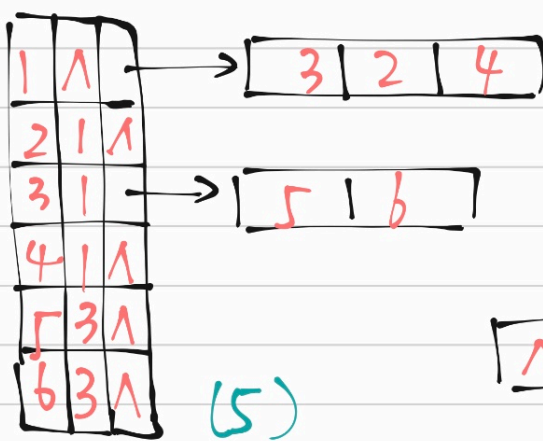
(2)



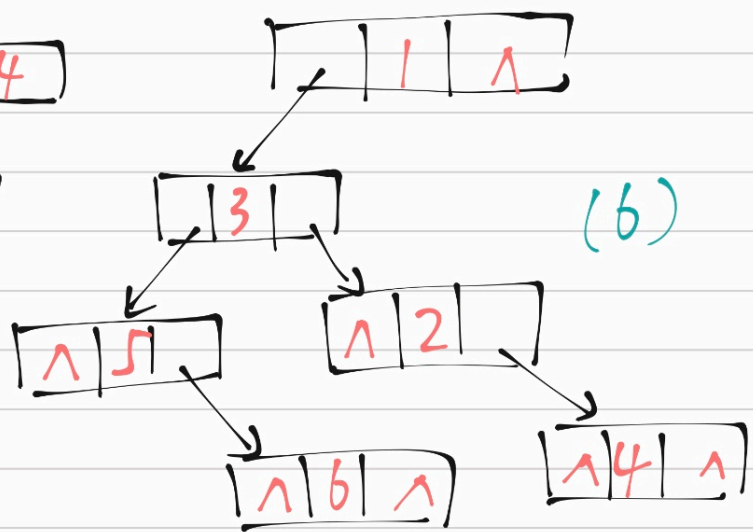
(3)



(4)



(5)



(6)

```

1 // 1 以vector的形式存储子节点
2 class Node {
3 public:
4     int val;
5     vector<Node*> children;
  
```

```

6     Node() {}
7     Node(int _val) {
8         val = _val;
9     }
10    Node(int _val, vector<Node*> _children) {
11        val = _val;
12        children = _children;
13    }
14 };
15
16 /**
17  * 创建树
18  * @param subtree
19  * @param val
20  */
21 void addChildren(Node*subtree,int val)
22 {
23     subtree->children.push_back(new Node(val));
24 }
25
26 Node* create(int val)
27 {
28     Node*root=new Node;
29     root->val=val;
30     return root;
31 }
32
33 int main() {
34     Node* r=create(1);
35     addChildren(r,3);
36     addChildren(r,2);
37     addChildren(r,4);
38     addChildren(r->children[0],5);
39     addChildren(r->children[0],6);
40
41     vector<vector<int>> res=levelOrder(r);
42     cout<<"层次遍历结果: "<<endl;
43     cout<< '['<<endl;
44     for(int i=0;i<res.size();i++)
45     {
46         cout<< '[';
47         for(int j=0;j<res[i].size();j++)
48         {
49             cout<<res[i][j]<<" ";
50         }
51         cout<<']'<<endl;
52     }
53     cout<<']'<<endl;
54
55     vector<int> r_pos=postorder(r);
56     cout<<"后序遍历结果: "<<endl;
57     cout<< '[';
58
59     for(int j=0;j<r_pos.size();j++)
60     {
61

```

```

62         cout<<r_pos[j]<<" ";
63     }
64     cout<<']'<<endl;
65
66     vector<int> r_pre=preorder(r);
67     cout<<"前序遍历结果: "<<endl;
68     cout<< '[';
69
70     for(int j=0;j<r_pre.size();j++)
71     {
72         cout<<r_pre[j]<<" ";
73     }
74     cout<<']'<<endl;
75
76     return 0;
77 }
78 /*
79 输出:
80 层次遍历结果:
81 [
82 [1 ]
83 [3 2 4 ]
84 [5 6 ]
85 ]
86 后序遍历结果:
87 [5 6 3 2 4 1 ]
88 前序遍历结果:
89 [1 3 5 6 2 4 ]
90 */

```

遍历多叉树

先序遍历

```

1 class Solution {
2 public:
3     vector<int> preorder(Node* root) {
4         vector<int>res;
5         pre(root,res);
6         return res;
7     }
8     void pre(Node*root,vector<int>&res)
9     {
10         if(root==NULL)return;
11         res.push_back(root->val);
12         vector<Node*> t=root->children;
13         for(int i=0;i<t.size();i++)
14         {
15             pre(t[i],res);
16         }
17     }
18 };

```

后序遍历

```
1 class Solution {
2 public:
3     vector<int> postorder(Node* root) {
4         vector<int> res;
5         pos(root, res);
6         return res;
7     }
8     void pos(Node* root, vector<int> &res)
9     {
10         if(root == NULL) return;
11         vector<Node*> t = root->children;
12         for(int i = 0; i < t.size(); i++)
13         {
14             pos(t[i], res);
15         }
16         res.push_back(root->val);
17     }
18 };
```

层次遍历

```
1 class Solution {
2 public:
3     //实现输出: [1,3,2,4,5,6]
4     vector<int> levelOrder(Node* root) {
5         vector<int> res;
6         if(root == NULL) return res;
7         queue<Node*> q;
8         q.push(root);
9         res.push_back(root->val);
10        while(!q.empty())
11        {
12            Node* top = q.front();
13            q.pop();
14            res.push_back(top->val);
15            vector<Node*> t = top->children;
16            for(int i = 0; i < t.size(); i++)
17            {
18                q.push(t[i]);
19            }
20        }
21        return res;
22    }
23 };
24
25 //输出vector<vector<int> >: [[1],[3,2,4],[5,6]]
26 vector<vector<int> > levelOrder(Node* root) {
27     vector<vector<int> > res;
28     if(root == NULL) return res;
29     queue<Node*> q;
30     q.push(root);
```

```
31 while(!q.empty())
32 {
33     vector<int>temp;
34     int cur_size=q.size();
35     for(int i=0;i<cur_size;i++)
36     {
37         Node*top=q.front();
38         q.pop();
39         temp.push_back(top->val);
40         vector<Node*>t=top->children;
41         for(int i=0;i<t.size();i++)
42         {
43             q.push(t[i]);
44         }
45     }
46     res.push_back(temp);
47 }
48
49 return res;
50 }
```