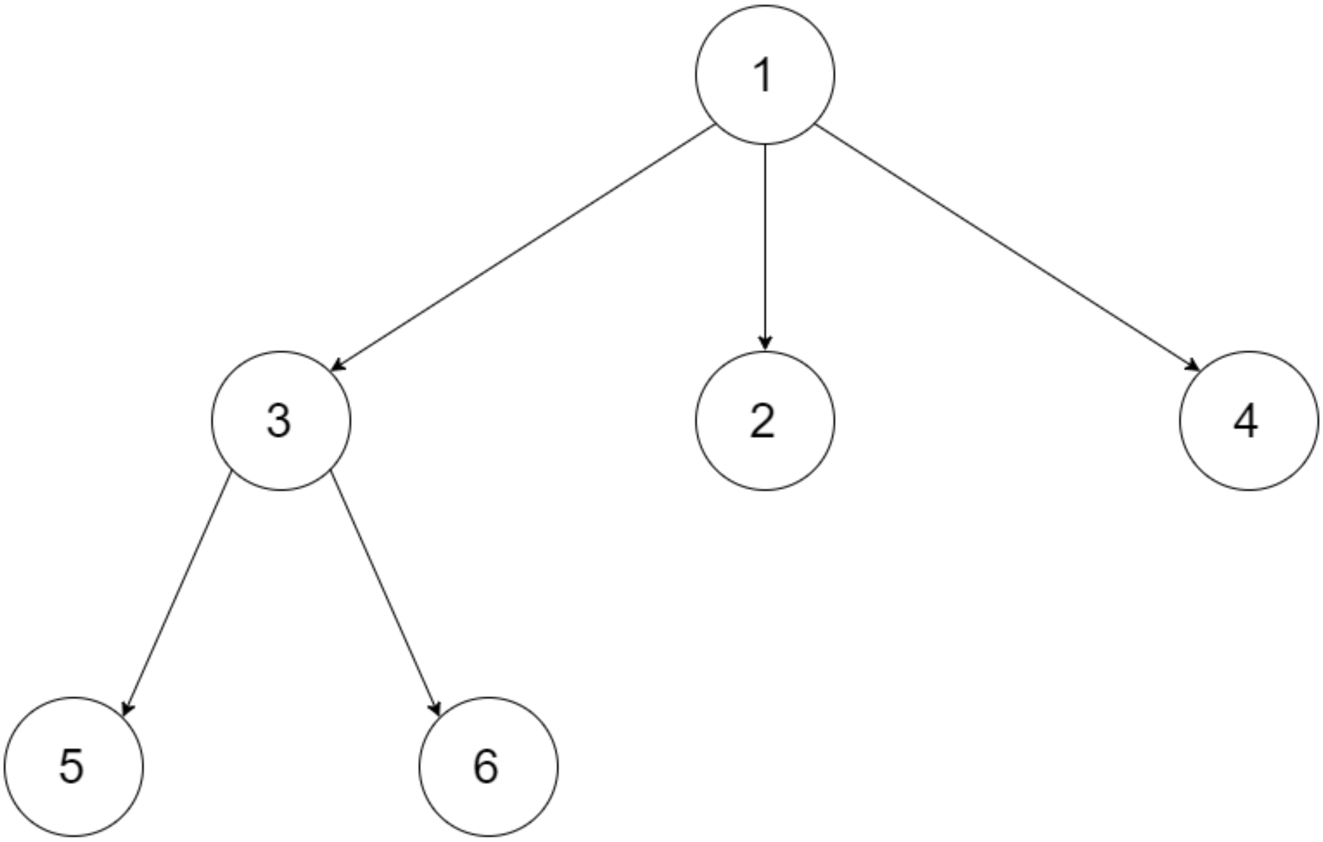
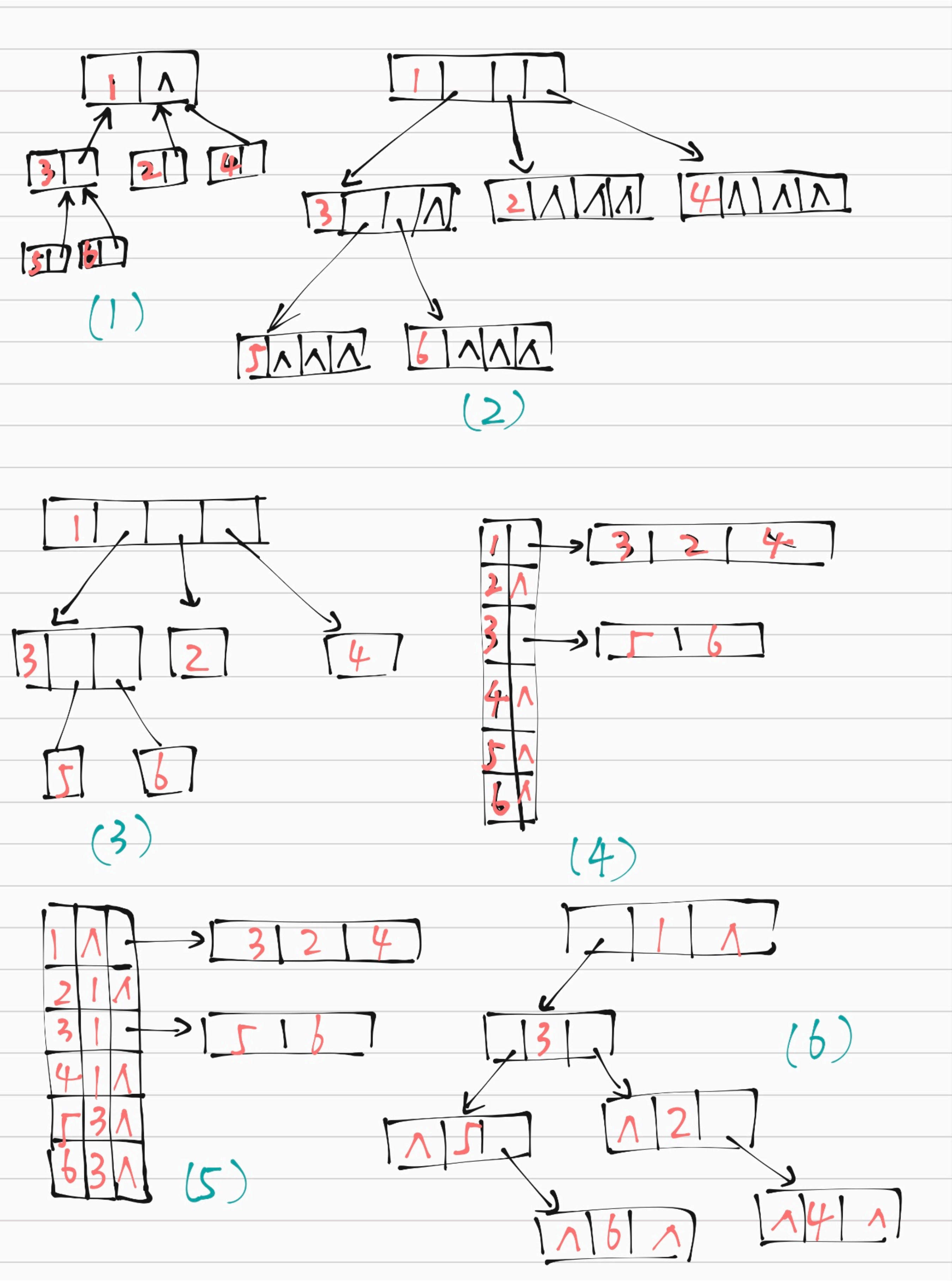
多叉树（N叉树）

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（缺点是不好访问父节点）
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 以vector的形式存储子节点
class Node {
public:
    int val;
    vector<Node*> children;
```

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

/**
 * 创建树
 * @param subtree
 * @param val
 */
void addChildren(Node*subtree,int val)
{
    subtree->children.push_back(new Node(val));
}

Node* create(int val)
{
    Node*root=new Node;
    root->val=val;
    return root;
}

int main() {
    Node* r=create(1);
    addChildren(r,3);
    addChildren(r,2);
    addChildren(r,4);
    addChildren(r->children[0],5);
    addChildren(r->children[0],6);

    vector<vector<int>> res=levelOrder(r);
    cout<<"层次遍历结果："<<endl;
    cout<<'['<<endl;
    for(int i=0;i<res.size();i++)
    {
        cout<<'[';
        for(int j=0;j<res[i].size();j++)
        {
            cout<<res[i][j]<<" ";
        }
        cout<<']'<<endl;

    }
    cout<<']'<<endl;

    vector<int> r_pos=postorder(r);
    cout<<"后序遍历结果："<<endl;
    cout<<'[';

    for(int j=0;j<r_pos.size();j++)
    {
        cout<<r_pos[j]<<" ";
    }
    cout<<']'<<endl;

    vector<int> r_pre=preorder(r);
    cout<<"前序遍历结果: "<<endl;
    cout<<'[';

    for(int j=0;j<r_pre.size();j++)
    {
        cout<<r_pre[j]<<" ";
    }
    cout<<']'<<endl;

    return 0;
}
/*
输出:
层次遍历结果:
[
[1 ]
[3 2 4 ]
[5 6 ]
]
后序遍历结果:
[5 6 3 2 4 1 ]
前序遍历结果:
[1 3 5 6 2 4 ]
 */
```

遍历多叉树

先序遍历

```
class Solution {
public:
    vector<int> preorder(Node* root) {
        vector<int>res;
        pre(root,res);
        return res;
    }
    void  pre(Node*root,vector<int>&res)
    {
        if(root==NULL)return;
        res.push_back(root->val);
        vector<Node*> t=root->children;
        for(int i=0;i<t.size();i++)
        {
            pre(t[i],res);
        }
    }
};
```

后序遍历

```cpp
class Solution {
public:
    vector<int> postorder(Node* root) {
        vector<int>res;
        pos(root,res);
        return res;
    }
    void  pos(Node*root,vector<int>&res)
    {
        if(root==NULL)return;
        vector<Node*> t=root->children;
        for(int i=0;i<t.size();i++)
        {
            pos(t[i],res);
        }
        res.push_back(root->val);
    }
};
```

层次遍历

```cpp
class Solution {
public:
//实现输出：[1,3,2,4,5,6]
    vector<int> levelOrder(Node* root) {
       vector<int> res;
        if(root==NULL)return res;
        queue<Node*>q;
        q.push(root);
        res.push_back(root->val);
        while(!q.empty())
        {
            Node*top=q.front();
            q.pop();
            res.push_back(top->val);
            vector<Node*>t=top->children;
            for(int i=0;i<t.size();i++)
            {
                q.push(t[i]);
            }
        }
        return res;
    }
};

//输出vector<vector<int> >：[[1],[3,2,4],[5,6]]
    vector<vector<int> > levelOrder(Node* root) {
       vector< vector<int> > res;
        if(root==NULL)return res;
        queue<Node*>q;
        q.push(root);
        while(!q.empty())
        {
            vector<int>temp;
            int cur_size=q.size();
            for(int i=0;i<cur_size;i++)
            {
                Node*top=q.front();
                q.pop();
                temp.push_back(top->val);
                vector<Node*>t=top->children;
                for(int i=0;i<t.size();i++)
                {
                    q.push(t[i]);
                }
            }
            res.push_back(temp);

        }
        return res;
    }
```