Week04

> DFS，BFS ，贪心算法，二分查找

DFS代码模板

- **递归写法**

```
map<int,int>visited;
void dfs(Node* root){
  //terminator
  if(!root) return;

  if (visited.count(root->val)){
    //already visited
    return;
  }

  visited[root->val] =1;

  //process current node here.
  //...
  for (int i=0; i<root->children.size();++i){
    dfs(root->children[i]);
  }
  return;
}
```

- **非递归写法**

```
void dfs(Node* root){
   map<int,int> visited;
   if(!root) return;

   stack<Node*> stackNode;
   stackNode.push(root);

   while (!stackNode.empty){
     stackNode.pop();
     if (visited.count(node->val)) continue;
     visited[node->val] = 1;

```

```
      for (intc i=node->children.size()-1; i>=0; --i{
         stackNode.push(node->children[i]);
      }
    }

    return;
}
```

BFS代码模板

```
void bfs(Node* root){
  map<int,int> visited;
  if(!root) return;

  queue<Node*> queueNode;
  queueNode.push(root);

  while(!queueNode.empt()){
    Node* node=queueNodetop();
    queueNode.pop();
    if(visited.count(node->val)) continue;
    visited[node->val] =1;

    for (int i=0(); i< node->children.size();++i){
      queueNode.push(node->children[i]);
    }
  }

  return;
}
```

贪心算法

1. 特点

- 在每一步选择中都采取在当前状态下最或最优（即最有利）的选择，从而希望导致结构是全局最好或最优的算法

- 与动态规划的区别：对每个子问题的解决方案都做出选择，不能回退
- 解决一些最优化问题： 求图中的最小生成树，求哈夫曼编码等

2.适用场景

能够分解子问题来解决， 子问题的最优解能递推到最终问题的最优解。这种子问题最优解称为最优子结构

二分查找

1.前提

- 目标函数单调性（单调递增或者递减）
- 存在上下界（bounded）
-

```
int binarySearch (const vector <int>& nums, int target){
    int left =0, right=(int)nums.size()-1;

    while (left<=right){
      int mid = left+(right-left)/2;
      if (nums[mid]==target) return mid;
      else if (nums[mid]<target) left=mid+1;
      else right=mid-1;
    }

    return -1;
}

```