



第四周（2.22-2.28）

📅 StartDate	@2021/02/22 → 2021/02/28
▼ Status	进行中
≡ 内容	1.深度、广度优先搜索 2.贪心算法与二分查找
≡ 总结	
↗ 题目	

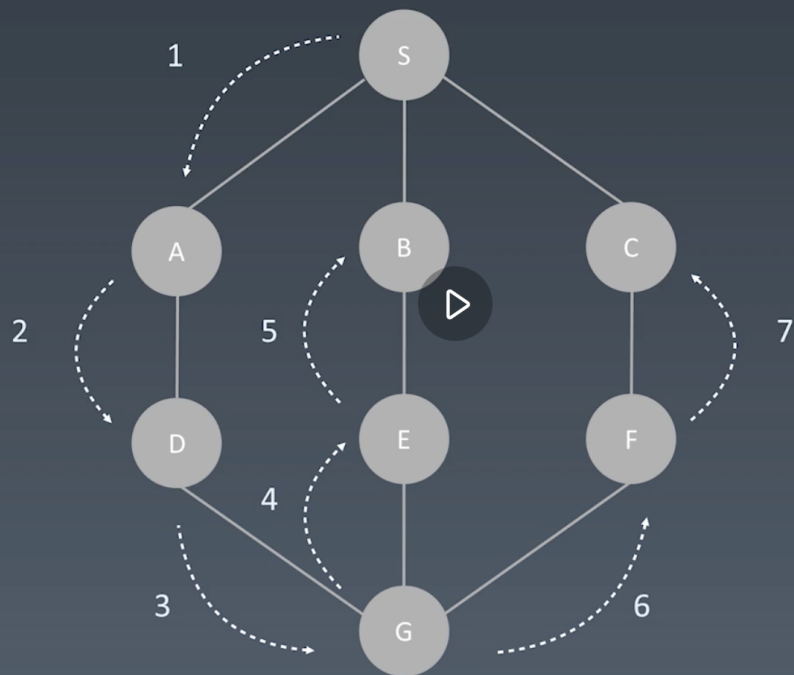
一、深度、广度优先搜索

- 搜索-遍历的特点：每个节点都要访问一次，每个节点仅访问一次。
- 根据节点访问顺序的不同，分类：
 1. 广度优先BFS：breadth first search
 2. 深度优先DFS：depth first search
 3. 其他....优先级优先（启发式搜索）

BFS和DFS有什么区别？

- DFS：不等递归调用执行完，直接进入下一层
- 图的DFS遍历示例：

遍历顺序



$S \rightarrow A$ 、 $S \rightarrow B$ 、 $S \rightarrow C$ 的概率相同。 $G \rightarrow E$ 、 $G \rightarrow F$ 的概率相同。

▼ DFS模版

```
// JavaScript 递归
// 二叉树
const visited = new Set();
const dfs = node => {
  // 节点已访问过
  if (visited.has(node)) return;
  visited.add(node);
  dfs(node.left);
  dfs(node.right);
}

// 多叉树
const visited = new Set();
const dfs = node => {
  visited.add(node);

  // process current node here
  // ...
  for (next_node in node.children()) {
    // 加入时判断是否已访问过，直接调用dfs进入下一层
  }
}
```

```

        if (!visited.has(node)) dfs(next_node);
    }
}

// 多叉树-递归
const visited = new Set();
const dfs = node => {
    // 递归结束条件
    if (visited.has(node)) return;
    visited.add(node);

    // process current node here
    // ...
    for (next_node in node.children()) {
        // 加入时判断是否已访问过，直接调用dfs进入下一层
        if (!visited.has(node)) dfs(next_node);
    }
}

```

▼ 多叉树-非递归-伪代码

```

// 多叉树-非递归
const dfs = (tree) => {
    if (!tree.root) return [];
    const visited = new Set(), stack = [], [tree.root]; // [tree.root]?

    while(stack.length) {
        node = stack.pop();
        visited.add(node);

        // process(node); // ?
        nodes = generate_related_nodes(node);
        stack.push(nodes);
    }
}

```

```

// python
def DFS(self, tree):
    if tree.root is None:
        return []

    visited, stack = [], [tree.root]

    while stack:
        node = stack.pop()
        visited.add(node)

        process (node)

```

```
nodes = generate_related_nodes(node)
stack.push(nodes)
# other processing work
```

▼ BFS模版

```
//JavaScript
const bfs = (root) => {
  let result = [], queue = [root];
  while (queue.length > 0) {
    let level = [], n = queue.length;
    for (let i = 0; i < n; i++) {
      let node = queue.pop();
      level.push(node.val);
      if (node.left) queue.unshift(node.left);
      if (node.right) queue.unshift(node.right);
    }
    result.push(level);
  }
  return result;
};
```

双向BFS原理剖析？

实战题目

- 二叉树的层序遍历（字节跳动、亚马逊、微软在半年内面试中考过）
- 最小基因变化
- 括号生成（字节跳动、亚马逊、Facebook 在半年内面试中考过）
- 在每个树行中找最大值（微软、亚马逊、Facebook 在半年内面试中考过）


课后作业

- 单词接龙（亚马逊在半年内面试常考）
- 单词接龙 II（微软、亚马逊、Facebook 在半年内面试中考过）
- 岛屿数量（近半年内，亚马逊在面试中考查此题达到 350 次）
- 扫雷游戏（亚马逊、Facebook 在半年内面试中考过）

二、贪心算法

贪心算法的原理是什么，与动态规划的不同之处：

贪心算法 Greedy



贪心算法是一种在每一步选择中都采取在当前状态下最好或最优（即最有利）的选择，从而希望导致结果是全局最好或最优的算法。

贪心算法与动态规划的不同在于它对每个子问题的解决方案都做出选择，不能回退。动态规划则会保存以前的运算结果，并根据以前的结果对当前进行选择，有回退功能。


贪心：当下做局部最优判断

回溯：能够回退

动态规划：最优判断 + 回退

贪心算法有哪些高级应用？

贪心算法 Greedy



贪心法可以解决一些最优化问题，如：求图中的最小生成树、求哈夫曼编码等。然而对于工程和生活中的问题，贪心法一般不能得到我们所要求的答案。

一旦一个问题可以通过贪心法来解决，那么贪心法一般是解决这个问题的最好办法。由于贪心法的高效性以及其所求得的答案比较接近最优结果，贪心法也可以用作辅助算法或者直接解决一些要求结果不特别精确的问题。

参考链接

- [coin change 题目](#)

- 动态规划定义

课后作业

- 柠檬水找零（亚马逊在半年内面试中考过）
- 买卖股票的最佳时机 II（亚马逊、字节跳动、微软在半年内面试中考过）
- 分发饼干（亚马逊在半年内面试中考过）
- 模拟行走机器人
- 跳跃游戏（亚马逊、华为、Facebook 在半年内面试中考过）
- 跳跃游戏 II（亚马逊、华为、字节跳动在半年内面试中考过）

三、二分查找

二分查找有哪些高级变形？

前提条件：

二分查找的前提

1. 目标函数单调性（单调递增或者递减）
2. 存在上下界（bounded）
3. 能够通过索引访问（index accessible）

代码模版

```
/* JavaScript */
let left = 0, right = len(array) - 1;
while (left <= right) {
    let mid = (left + right) >> 1;
    if (array[mid] === target) {
        /*find the target*/
        return;
    } else if (array[mid] < target) {
```

```
    left = mid + 1;
} else {
    right = mid - 1;
}
}
```

参考链接

- [二分查找代码模板](#)
- [Fast InvSqrt\(\) 扩展阅读](#)

实战题目

- [x 的平方根](#)（字节跳动、微软、亚马逊在半年内面试中考过）
- [有效的完全平方数](#)（亚马逊在半年内面试中考过）

课后作业

- [搜索旋转排序数组](#)（Facebook、字节跳动、亚马逊在半年内面试常考）
- [搜索二维矩阵](#)（亚马逊、微软、Facebook 在半年内面试中考过）
- [寻找旋转排序数组中的最小值](#)（亚马逊、微软、字节跳动在半年内面试中考过）
- 使用二分查找，寻找一个半有序数组 [4, 5, 6, 7, 0, 1, 2] 中间无序的地方说明