$DP\ pro$

20数媒技欧芃芃

内容

（一）背包DP复建

（二）树上DP折磨

（三）总结

（四）参考资料

（一）背包DP复建

- **0/1背包与完全背包：**
    1. 题胚区别
    2. 注意事项
    3. 关键代码复习
- **DP回顾：**
    1. 入手思路参考
    2. 初始化

复建

- 0/1背包
- 完全背包

（二）**树上**DP

> 树形动态规划，即在树上进行的动态规划。

一个**仅供入门**的思路：

- 树上DP一般**递归**求解的。
- 建好树之后，从根节点出发，一直搜到叶节点，然后递归上去得到答案。

前置知识回顾：

- DFS
- 链式前向星
    1. 初始化
    2. 建树
    3. 遍历

(1) 最大子树和

> 给定一棵有 n 个点的树，树上每个点有各自的点权。(**存在负值**)
> 在树中选取一棵子树，使得其权值之和最大。

思路：

- 就像大部分~~我写过的~~树形DP，或者树的题目一样，我们首先考虑维护根节点的信息。~~然后你发现这题就结束了~~
- 设 $dp[i]$ 表示以 i 为根节点的最大子树和。
- 那么对于节点 u 以及它的子节点（们） to 而言，有 $dp[u] = val[u] + \sum max(dp[to],\ 0)$
- **初始化**

代码：

```
void dfs(int u, int father) {
    dp[u] = val[u]; //初始化

    for(int i = head[u]; i != -1; i = edge[i].next) { //遍历u的所有边
        int to = edge[i].to; //取出节点

        if(to == father) { //不要回头搜
            continue;
        }

        dfs(to, u); //递归过程

        if(dp[to] > 0) { //不使答案劣化
            dp[u] += dp[to]; //才参与贡献
        }
    }
}
```

$Question\ Time:$

1. 如何**调用**函数
2. 我的**答案**在哪里

$Answer:$

~~随机挑选一个幸运小孩~~

另一节代码：

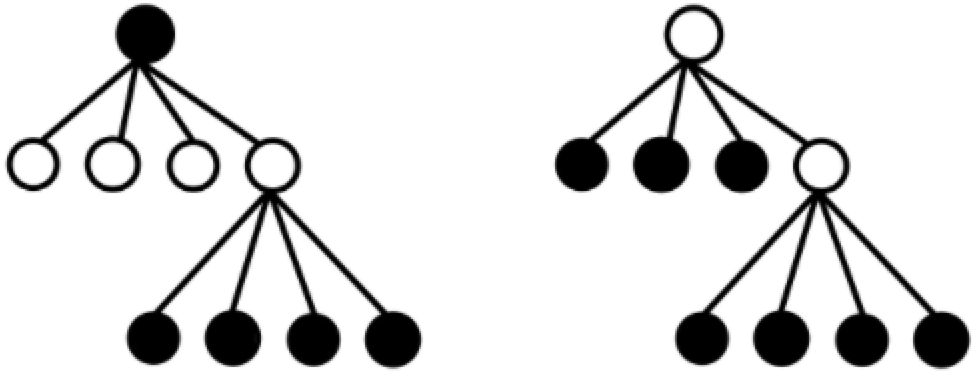
```
dfs(1, -1); //根节点和它不存在的父节点

int ans = ninf; //负无穷
for(int i = 1; i <= n; i++) {
    ans = max(ans, dp[i]);
}
printf("%d", ans); //输出
```

快乐写题

（2）最大（权值）独立集

> 最大独立集定义：
> 对于一棵有 n 个结点的树，选出尽量多的结点，使得任两个结点均不相邻。



题胚

> 给定一棵有 n 个结点的树，及每个节点的权值 val
> 选择合适的节点，使得节点**权值之和最大**，并且**任两节点不相邻。**

思路：

- 和之前一样，我们需要**维护当前节点信息。**
  同时考虑到每个节点的选取，会影响与之相邻的节点的选取状态，所以我们还需要**维护每个节点的选取状态。**
- 设 $dp[i][0]$ 表示**不选择**节点 i 时，以节点 i 为根的子树的最大独立集的权值。
  设 $dp[i][1]$ 表示**选择**节点 i 时，以节点 i 为根的子树的最大独立集的权值。
- 那么**转移方程**有：
  i. $dp[u][0] = \sum max(dp[to][0],\ dp[to][1])$
  ii. $dp[u][1] = val[u] + \sum dp[to][0]$
- **初始化**与**目标值**

(非唯一）代码：

```
void dfs(int u, int father) {
    //初始化
    dp[u][0] = 0;
    dp[u][1] = val[u];

    for(int i = head[u]; i != -1; i = edge[i].next) {
        int to = edge[i].to;

        if(to == father) {
            continue;
        }

        dfs(to, u);

        dp[u][0] += max(dp[to][0], dp[to][1]);
        dp[u][1] += dp[to][0];
    }
}
```

实例

> 有 n 名职员和其对应的快乐指数。现将举办一场聚会，不过，职员**不愿意和自己的直接上司一起参会。**
> 在这个条件下，决定邀请一部分职员参会，使得所有参会职员的快乐指数总和最大，求这个最大值。

（3）树上背包

> 满足如果选取节点 v，则其**所有祖先节点 u 都要选择**的限制

边权题胚

> 给定一棵树。每条**边**有对应的**非负**权值 val 。
> 现今需要剪枝。给定需要保留的**边的数量** m，求所剩边的最大权值之和。

点权题胚

> 给定一棵树。每个**节点**有对应的**非负**权值 val。
> 现今需要剪枝。给定需要保留的**节点数量** m，求所剩节点的最大权值之和。

以边权题胚为例：

- 我们照例维护**节点信息**。同时我们还需要知道**已经保留的边数**。
- 设 $dp[u][j]$ 表示以 u 为根节点的子树，在保留 j 条边时的最大权值之和。
- 那么有：
  $dp[u][j] = max(dp[u][k] + dp[to][j-k-1] + val[u][to])，\ 0 \le k < j$
- **初始化**
- **目标值**：$dp[root][m]$

参考意义不大的代码：

```
void dfs(int u, int father) {
    for(int i = head[u]; i != -1; i = edge[i].next) {
        int to = edge[i].to;

        if(to == father) {
            continue;
        }

        dfs(to, u);

        for(int j = m; j >= 1; j--) { //注意01背包逆序更新
            for(int k = j - 1; k >= 0; k--) {
                //注意这里k是可以取到0的，这就意味着把之前所有的子树都剪掉，而选择新的一个子树
                dp[u][j] = max(dp[u][j], dp[u][k] + dp[to][j - k - 1] + num[u][to]);
            }
        }
    }
}
```

注意事项：

- 之所以说是参考意义不大，是因为背包问题或者说DP问题都比较灵活多变。不同的题目或许是一样的考点，但转移方程或别的地方存在细小差别。
- 点权的题思路是类似的，~~稍微改改就能AC~~
- ***切勿死记硬背！！***
- ***切勿死记硬背！！***
- ***切勿死记硬背！！***

快乐写题

- 边权树上背包
- 点权树上背包

另附一些值得了解的知识点

覆盖问题：（如何用最少的点覆盖整棵树）

- 最小点覆盖(覆盖所有边)
- 最小支配集(覆盖所有点)

学有余力或者感兴趣的同学可以下课找我唠嗑唠嗑

（三）伪总结

- 背包DP**深入了解**可参考后附材料**背包九讲**
- 树上DP初次接触的难点部分在于对**树形结构**和**递归过程**的不熟悉。这种时候建议多敲几遍代码，光看懂了不一定能顺利写出来。
  ~~（这本质是数据结构的锅，DP不背啊喂）~~
- DP初讲中，个人觉得**线性DP**也很重要，建议复习
- 目前能介绍的DP题目代码量都不大，但是**思维量要求较高**，建议大家攻克DP路上多写多练，每一题大多不会完全相似的。

参考资料

- 背包九讲

THANKS