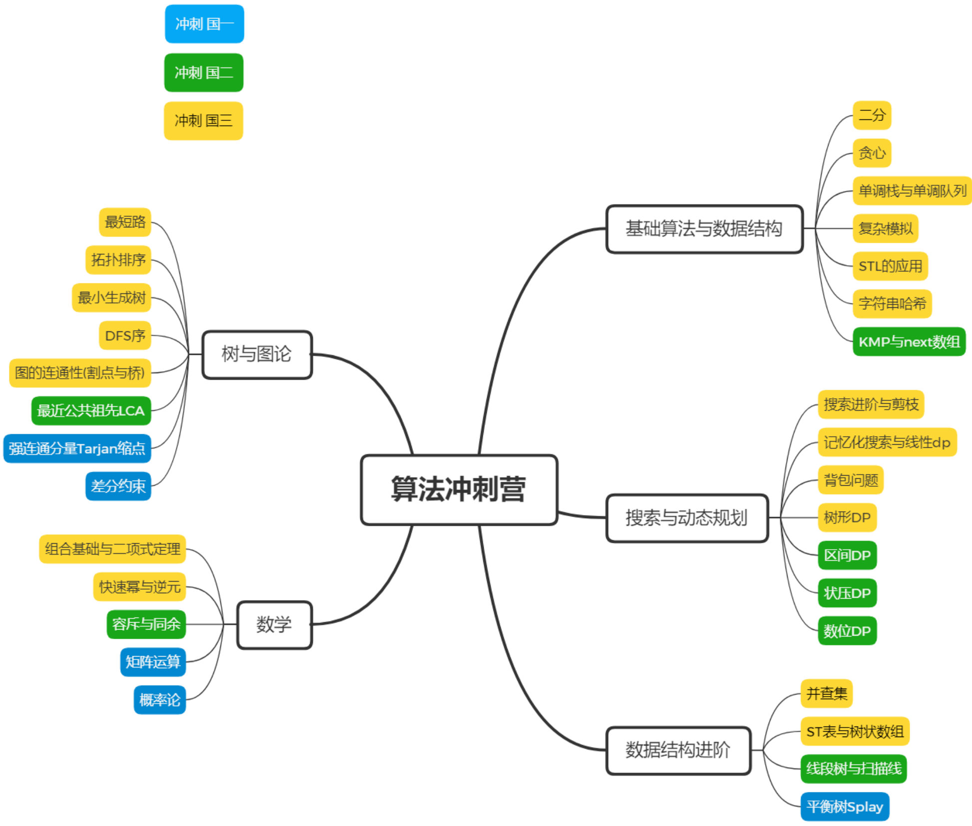
蓝桥杯30天算法冲刺集训



Day-9 树形DP

树形动态规划

在树上设计动态规划算法时，一般就以节点从深到浅（子树从小到大）的顺序作为 DP 的“阶段”DP 的状态表示中，第一维通常是节点编号（代表以该节点为根的子树）。大多数时候，我们采用递归的方式实现树形动态规划。对于每个节点 x，先递归在它的每个子节点上进行 DP，在回溯时，从子节点向节点 x 进行状态转移去推父结点的状态信息，边界条件则是直接由叶子节点确定。整体上采用**后序遍历**的方法求解状态。

例题1：没有上司的舞会

因为题目要求选出的点中不能有两个点具有父子关系，也就是选了父亲，那么所有儿子都不能选；没选父亲，那么儿子才可以选。

如果我们确定父亲不选，那各个儿子和它的子树就构成了一个子问题。如果父亲确定选，那就构成了确定儿子不选的一个子问题。

所以设计状态的时候，需要将当前的结点选还是不选给设计进去。我们设 $f[i][0/1]$ 表示以 i 号点为根的子树在 i 号点不选/选的情况下，能选出的最大权值和。如果 i 号点我们没选，那对于它的每个儿子我们都有选或不选两种选择，对应的 dp 值也就是 $f[son(i)][0]$ 和 $f[son(i)][1]$ 取两者中较大的那一个。如果 i 号点我们选了，那么对于它的每个儿子都只有不选一个选择，对应的dp值就是 $f[son(i)][0]$。自然，转移的时候就可以写出

$$\begin{cases} f[i][0] = \sum_{j=son(i)} \max(f[j][0], f[j][1]) \\ f[i][1] = \sum_{j=son(i)} f[j][0] \end{cases}$$

我们发现，在所有可能的 4 种转移中唯独缺少了 $f[j][1] \to f[i][1]$，这其实也就对应题目要求的父子结点不能同时被选的情况。

由于每个结点只会被遍历一次，因此时间复杂度为 $O(n)$。

```python
import sys
from collections import defaultdict

sys.setrecursionlimit(10**6)

N = 6005

def dfs(u):
    f[u][0] = 0
    f[u][1] = r[u]
    for v in g[u]:
        f[u][0] += dfs(v)
        f[u][1] += f[v][0]
    return max(f[u][0], f[u][1])

n = int(input())
r = [0] * N
g = defaultdict(list)
in_degree = [0] * N

for i in range(1, n + 1):
    r[i] = int(input())
while True:
    u, v = map(int, input().split())
    if u == 0 and v == 0:
        break
    g[v].append(u)
    in_degree[u] += 1

for i in range(1, n + 1):
    if in_degree[i] == 0:
        f = [[0, 0] for _ in range(N)]
```

```
        print(dfs(i))
        break
```

例题2： 二叉苹果树

因为要求剩下 q 条边，所以剩下多少条边是需要写进状态里面的。根据树形 DP 的一般特点，可以设出：$f[i][j]$ 表示以 i 为根节点的子树上可以选 j 条边时，边上苹果之和的最大值，易得 $f[i][0]=0$。转移时，每次枚举左子树选择了 k 条边，右边剩 $j-k-?$ 条边(注意，选择左/右子树时，必选父子之间的树边，因此这里用 ? 来表示不确定)。

记左孩子为 l_i，左边树枝上苹果数为 w_l，右孩子为 r_i，右边树枝上苹果数为 w_r，可以得到转移：

$$f[i][j]=\max_{1\le k\le j-2}\{f[l_i][k]+f[r_i][j-k-2]+w_l+w_r, f[l_i][j-1]+w_l, f[r_i][j-1]+w_r$$

上面的转移分别对应：把边分配到左右子树、不选右子树、不选左子树。

由于在枚举每个结点时，对每个状态都花了 $O(j)$ 的时间枚举怎么分配边，所以时间复杂度为 $O(nq^2)$。

```python
N = 105

def dfs(u, fa):
    f[u][0] = 0
    l = -1
    r = -1
    for i in range(len(g[u])):
        v = g[u][i][0]
        if v == fa:
            continue
        if l == -1:
            l = i
        else:
            r = i
        dfs(v, u)
    if l == -1:
        return
    for j in range(1, q + 1):
        for k in range(q + 1):
            if q >= 2 and k + 2 <= j:
                f[u][j] = max(f[u][j], f[g[u][l][0]][k] + f[g[u][r][0]][j - k - 2] + g[u][l][1] + g[u][r][1])
        f[u][j] = max(f[u][j], f[g[u][l][0]][j - 1] + g[u][l][1])
        f[u][j] = max(f[u][j], f[g[u][r][0]][j - 1] + g[u][r][1])

n, q = map(int, input().split())
f = [[0] * (N) for _ in range(N)]
g = [[] for _ in range(N)]
for _ in range(n - 1):
```

```
    u, v, w = map(int, input().split())
    g[u].append((v, w))
    g[v].append((u, w))

dfs(1, 0)
print(f[1][q])
```

蓝桥杯真题

非对称二叉树（蓝桥杯2023JavaB组国赛）

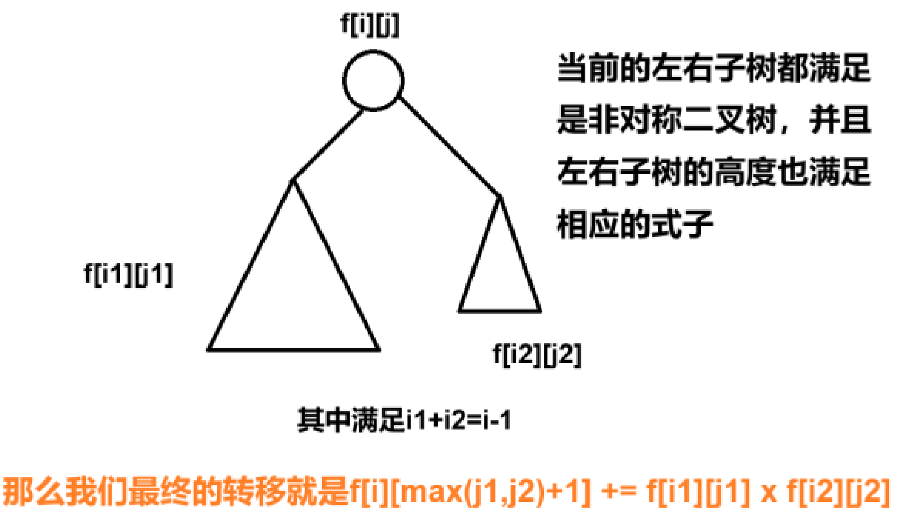
本题是一道理解树形DP的好题。

首先我们看题目里面的定义，也就是说对于一颗任意的二叉树，如果要满足题目给定条件是不是对于所有的子树都得满足: $max(h_{l_i}, h_{r_i}) \geq k \times min(h_{l_i}, h_{r_i})$这个式子成立。那么我们应用树形DP回溯转移的思想来计算即可。

首先我们设$f[i][j]$代表**当前有i个节点高度为j的非对称二叉树有多少个**

那么我们最终要求的答案是不是就是$\sum_{i=1}^{k} f[n][i]$，那么我们再考虑如何转移。

我们可以显然的知道，现在是一个二叉树，最多只有两个孩子，并且要保证两个孩子所在的子树也都得满足: $max(h_{l_i}, h_{r_i}) \geq k \times min(h_{l_i}, h_{r_i})$这个式子成立，同时孩子的孩子也得满足等等，那么我们是不是直接枚举两个孩子各自的节点的数量和高度，然后用给定的式子去进行判断，如果判断符合的话，直接把相应的DP值乘起来就可以了。计算流程可以看下图：



之后我们还需要设置初始值，显然如果是$f[0][0]$的话，也就说当前没有节点，那么答案设置为贡献1，属于一种情况，可以跟兄弟一块计算。如果是$f[1][1]$的话，说明只有一个节点也是初试的情况，需要设置为1.

最后我们进行转移即可，时间复杂度为$O(N^4)$

```python
n, k = map(int, input().split())
f = [[0] * 50 for _ in range(50)]
ans = 0

f[1][1] = 1
f[0][0] = 1

for i in range(2, n + 1):
    for x in range(i):
        y = i - 1 - x
        for c in range(x + 1):
            for d in range(y + 1):
                if max(c, d) >= min(c, d) * k:
                    f[i][max(c, d) + 1] += f[x][c] * f[y][d]

for i in range(1, n + 1):
    ans += f[n][i]

print(ans)
```

习题

小蓝与一笔画

定义状态：

- $f[u][0]$ 为以 u 为根遍历完子树的最小权值且走回节点 u。
- $f[u][1]$ 为以 u 为根遍历完子树的最小权值且不走回节点 u。

$f[u][0]$：当需要走回根节点时，所有子节点也需要走回根节点，且连接两点的边权 w 需要走两遍，去和回。所以，子结点 v 对 $f[u][0]$ 的贡献即为 $f[v][0]+w*2$。

$f[u][0]=\sum_{u\xrightarrow{e}v}(f[e.v][0]+2\times e.w)$

$f[u][1]$：当不需要走回来时，必然有一个子节点不需要返回，假设该结点在子结点 v 所在子树内，考虑做减法 $f[v][0]-f[v][1]$，分析可发现，其值 $+w$ 即为 v 子树内距离 u 最远的点的距离。

$f[u][1]=f[u][0]-\max_{u\xrightarrow{e}v}\{e.w+f[e.v][0]-f[e.v][1]\}$

最后答案为 $f[1][1]$。

时间复杂度：$O(N)$

空间复杂度：$O(N)$

```python
import sys
sys.setrecursionlimit(10**6)
```

```python
N = 200005

class Edge:
    def __init__(self, v, w):
        self.v = v
        self.w = w

g = [[] for _ in range(N)]
f = [[0, 0] for _ in range(N)]

def dfs(u, fa):
    s = 0
    for e in g[u]:
        if e.v == fa:
            continue
        dfs(e.v, u)
        f[u][0] += f[e.v][0] + e.w * 2
        s = max(s, e.w + f[e.v][0] - f[e.v][1])
    f[u][1] = f[u][0] - s

def main():
    n = int(input())
    for _ in range(n - 1):
        u, v, w = map(int, input().split())
        g[u].append(Edge(v, w))
        g[v].append(Edge(u, w))
    dfs(1, 0)
    print(f[1][1])

if __name__ == "__main__":
    main()
```

树

这题数据对Python不太友好，可以跳过

明显是树形DP，状态设 $f[u][x][0/1]$表示以u为根的子树，形成x个连通块，根节点u$ 选不选的方案数。

转移的时候考虑如下情况：

- 如果 u 选，儿子 v 也选，边 (u, v) 就会被选，那么 u 的连通块和的 v 连通块就会合并成一个连通块，所以总的连通块数量减 1。
- 如果 u 或者 v 不选，那么就不会发生连通块合并。

于是枚举每个儿子状态 $f[v][y][0/1]$，有

- $f[u][x+y][0] = \sum_{x=0}^{size[u]} \sum_{y=1}^{size[v]} f[u][x][0] * (f[v][y][0] + f[v][y][1])$
- $f[u][x+y][1] = \sum_{x=0}^{size[u]} \sum_{y=1}^{size[v]} f[u][x][1] * f[v][y][0]$

- $f[u][x+y-1][1]=\sum_{x=0}^{size[u]}\sum_{y=1}^{size[v]}f[u][x][1]*f[v][y][1]$

不难看出这就是一个背包方案数计数问题，由于第二维 $x<size[u]$，所以可以用树形背包的 $O(n^2)$ 优化，通过限制枚举的上下界来优化（见上面的方程的枚举范围），进而通过此题。

```
#include <bits/stdc++.h>
using namespace std;

const int N = 5e3+5, mod = 998244353;
long long n, u, v, f[N][N][2], siz[N];
vector<int> g[N];
// f[u][x][0] 表示以 u 为根的子树，根不要，总共保留x个连通块的方案数
// f[u][x][1] 表示以 u 为根的子树，保留根，总共保留x个连通块的方案数

void dfs(int u, int fa) {
    siz[u] = 1;
    f[u][0][0] = 1;  // 当前结点不选，其他也不选
    f[u][1][1] = 1;  // 只选当前结点
    for(auto v : g[u]) {
        if(v != fa) {
            dfs(v, u);
            // 先计算不选 u 的情况
            for(int i = siz[u]; i >= 0; i--)
                for(int j = siz[v]; j >= 1; j--) {
                    //注意这里j不取0，0的时候有一种方案，和原来的dp[u][0][i]乘完放回去，相当于dp[u][0][i]的值不动
                    //u不要，v要不要都行
                    f[u][i+j][0] = (f[u][i+j][0] + f[u][i][0]*(f[v][j][0]+f[v][j][1])%mod) % mod;
                }
            // 计算选 u 的情况
            for(int i = siz[u]; i >= 1; i--)
                for(int j = siz[v]; j >= 1; j--) {
                    //注意这两句话顺序不能反，j等于1的时候，i + j - 1等于i，会改掉之前的值
                    f[u][i+j][1] = (f[u][i+j][1] + f[u][i][1]*f[v][j][0]%mod) % mod;
                    f[u][i+j-1][1] = (f[u][i+j-1][1] + f[u][i][1]*f[v][j][1]%mod) % mod;
                }
            siz[u] += siz[v];
        }
    }
}

int main(){
    cin >> n;
```

```
    for(int i = 1; i < n; i++) {
        cin >> u >> v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(1, 0);
    for(int i = 1; i <= n; i++) {
        cout << (f[1][i][0]+f[1][i][1]) % mod << '\n';
    }
    return 0;
}
```