

复杂DP

一、AcWing 1050. 鸣人的影分身

【题目描述】

在火影忍者的世界里，令敌人捉摸不透是非常关键的。

我们的主角漩涡鸣人所拥有的一个招数：多重影分身之术，就是一个很好的例子。

影分身是由鸣人身体的查克拉能量制造的，使用的查克拉越多，制造出的影分身越强。

针对不同的作战情况，鸣人可以选择制造出各种强度的影分身，有的用来佯攻，有的用来发起致命一击。

那么问题来了，假设鸣人的查克拉能量为 M ，他影分身的个数为 N ，那么制造影分身时有多少种不同的分配方法？

注意：

- 影分身可以分配0点能量。
- 分配方案不考虑顺序，例如： $M = 7, N = 3$ ，那么 $(2, 2, 3)$ 和 $(2, 3, 2)$ 被视为同一种方案。

【输入格式】

第一行是测试数据的数目 t 。

以下每行均包含二个整数 M, N ，以空格分开。

【输出格式】

对输入的每组数据 M, N ，用一行输出分配的方法数。

【数据范围】

$$0 \leq t \leq 20$$

$$1 \leq M, N \leq 10$$

【输入样例】

1	1
2	7 3

【输出样例】

【分析】

状态表示： $f[i][j]$ 表示总和为 i 且恰好表示成 j 个数的和的方案数。

状态计算：

- 当前的 j 个数中的最小值为0，那么我们将这个0删去，则总和不变，元素数量减一，即 $f[i][j] = f[i][j - 1]$ ；
- 当前的 j 个数中的最小值大于0，那么我们将每个数减1，则总和减 j ，元素数量不变，即 $f[i][j] = f[i - j][j]$ 。

由于元素数量固定，因此最后的答案即为 $f[m][n]$ 。

【代码】

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  const int N = 20;
7  int f[N][N]; // f[i][j]表示总和为i且恰好表示成j个数的和的方案
8  int n, m;
9
10 int main()
11 {
12     int T;
13     cin >> T;
14     while (T--)
15     {
16         cin >> m >> n;
17         f[0][0] = 1;
18         for (int i = 0; i <= m; i++)
19             for (int j = 1; j <= n; j++)
20             {
21                 f[i][j] = f[i][j - 1]; // 集合中的最小值为0
22                 if (i >= j) f[i][j] += f[i - j][j]; // 集合中的最小值大于0
23             }
24         cout << f[m][n] << endl;
25     }
```

```
26     return 0;
27 }
```

二、AcWing 1047. 糖果

【题目描述】

由于在维护世界和平的事务中做出巨大贡献，Dzx被赠予糖果公司2010年5月23日当天无限量糖果免费优惠券。

在这一天，Dzx可以从糖果公司的 N 件产品中任意选择若干件带回家享用。

糖果公司的 N 件产品每件都包含数量不同的糖果。

Dzx希望他选择的产品包含的糖果总数是 K 的整数倍，这样他才能平均地将糖果分给帮助他维护世界和平的伙伴们。

当然，在满足这一条件的基础上，糖果总数越多越好。

Dzx最多能带走多少糖果呢？

注意：Dzx只能将糖果公司的产品整件带走。

【输入格式】

第一行包含两个整数 N 和 K 。

以下 N 行每行1个整数，表示糖果公司该件产品中包含的糖果数目，不超过1000000。

【输出格式】

符合要求的最多能达到的糖果总数，如果不能达到 K 的倍数这一要求，输出0。

【数据范围】

$$1 \leq N \leq 100$$

$$1 \leq K \leq 100$$

【输入样例】

```
1 5 7
2 1
3 2
4 3
5 4
6 5
```

【输出样例】

```
1 14
```

【样例解释】

Dzx的选择是 $2 + 3 + 4 + 5 = 14$ ，这样糖果总数是7的倍数，并且是总数最多的选择。

【分析】

状态表示： $f[i][j]$ 表示所有只从前 i 个物品中选，且糖果总和除以 k 的余数为 j 的所有方案的最大糖果数。

状态计算：

- 不选第 i 个物品： $f[i][j] = f[i-1][j]$;
- 选第 i 个物品：那么前 $i-1$ 个物品的余数应该为 $(j-w_i)\%k$ ，即 $f[i][j] = f[i-1][(j-w_i)\%k + k]\%k$ 。

注意除了 $f[0][0]$ 合法之外其他所有方案都是不合法的，因此除了 $f[0][0]$ 初始化为0以外其余的 f 都初始化为负无穷。最后的答案即为 $f[n][0]$ 。

【代码】

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  const int N = 110;
7  int f[N][N]; // f[i][j]表示所有只考虑前i个物品,且价值总和模k的余数为j的方案的最大值
8  int n, k;
9
10 int main()
11 {
12     cin >> n >> k;
13     memset(f, 0x8f, sizeof f);
14     f[0][0] = 0;
15     for (int i = 1; i <= n; i++)
16     {
17         int w;
18
19         cin >> w;
```

```

19         for (int j = 0; j < k; j++)
20             f[i][j] = max(f[i - 1][j], f[i - 1][((j - w) % k + k) % k] +
21                 w);
22     }
23     cout << f[n][0] << endl;
24     return 0;
25 }

```

三、AcWing 1222. 密码脱落

【题目描述】

X星球的考古学家发现了一批古代留下来的密码。

这些密码是由A,B,C,D四种植物的种子串成的序列。

仔细分析发现，这些密码串当初应该是前后对称的（也就是我们说的镜像串）。

由于年代久远，其中许多种子脱落了，因而可能会失去镜像的特征。

你的任务是：给定一个现在看到的密码串，计算一下从当初的状态，它要至少脱落多少个种子，才可能会变成现在的样子。

【输入格式】

共一行，包含一个由大写字母A,B,C,D构成的字符串，表示现在看到的密码串。

【输出格式】

输出一个整数，表示至少脱落了多少个种子。

【数据范围】

输入字符串长度不超过1000。

【输入样例1】

```
1 ABCBA
```

【输出样例1】

```
1 0
```

【输入样例2】

```
1 ABDCCBABC
```

【输出样例2】

1 | 3

【分析】

首先我们在原字符串中找出最长的回文子序列，如下图所示，其中红色标出的即为最长回文子序列：



对于所有不在最长回文子序列中的字符（蓝色标记的字符），我们都可以其在对应的位置插入同样的字符使其匹配（绿色标记的字符），那么从当前字符串变成初始状态需要添加叶子的数量就等价于当前字符串变成最长回文子序列需要减去的叶子数量。即至少脱落多少个种子等价于原序列的长度减去最长回文子序列的长度。

状态表示： $f[i][j]$ 表示所有在区间 $[i, j]$ 中的回文子序列的最大长度。

状态计算：

- 若区间 $[i, j]$ 中的回文子序列包含 $s[i]$ 和 $s[j]$ ，也就是满足 $s[i] = s[j]$ 时，则区间 $[i, j]$ 中的回文子序列最大长度就为区间 $[i + 1, j - 1]$ 中的回文子序列最大长度再加上这两个字符，即 $f[i][j] = f[i + 1][j - 1] + 2$ ；
- 若不包含 $s[i]$ ，则区间 $[i, j]$ 中的回文子序列最大长度就为区间 $[i + 1, j]$ 中的回文子序列最大长度，即 $f[i][j] = f[i + 1][j]$ ；
- 若不包含 $s[j]$ ，则区间 $[i, j]$ 中的回文子序列最大长度就为区间 $[i, j - 1]$ 中的回文子序列最大长度，即 $f[i][j] = f[i][j - 1]$ 。

初始化：当区间长度为1时，每个字符自身为回文子序列，即 $f[i][j] = 1$ 。

【代码】

```
1 #include <iostream>
2 #include <cstring>
3 #include <algorithm>
4 using namespace std;
5
```

```

6  const int N = 1010;
7  char s[N];
8  int f[N][N];
9
10 int main()
11 {
12     cin >> s;
13     int n = strlen(s);
14     for (int len = 1; len <= n; len++)
15     {
16         for (int i = 0; i < n; i++)
17         {
18             int j = i + len - 1;
19             if (len == 1) { f[i][j] = 1; continue; }
20             if (s[i] == s[j]) f[i][j] = f[i + 1][j - 1] + 2;
21             f[i][j] = max(f[i][j], max(f[i + 1][j], f[i][j - 1]));
22         }
23     }
24     cout << n - f[0][n - 1] << endl;
25     return 0;
26 }

```

四、AcWing 1220. 生命之树

【题目描述】

在X森林里，上帝创建了生命之树。

他给每棵树的每个节点（叶子也称为一个节点）上，都标了一个整数，代表这个点的和谐值。

上帝要在这棵树内选出一个非空节点集 S ，使得对于 S 中的任意两个点 a, b ，都存在一个点列 $\{a, v_1, v_2, \dots, v_k, b\}$ 使得这个点列中的每个点都是 S 里面的元素，且序列中相邻两个点间有一条边相连。

在这个前提下，上帝要使得 S 中的点所对应的整数的和尽量大。

这个最大的和就是上帝给生命之树的评分。

经过atm的努力，他已经知道了上帝给每棵树上每个节点上的整数。

但是由于atm不擅长计算，他不知道怎样有效地求评分。

他需要你为他写一个程序来计算一棵树的分数。

【输入格式】

第一行一个整数 n 表示这棵树有 n 个节点。

第二行 n 个整数，依次表示每个节点的评分。

接下来 $n - 1$ 行，每行2个整数 u, v ，表示存在一条 u 到 v 的边。

由于这是一棵树，所以是不存在环的。

树的节点编号从 $1 \sim n$ 。

【输出格式】

输出一行一个数，表示上帝给这棵树的分数。

【数据范围】

$$1 \leq n \leq 10^5$$

每个节点的评分的绝对值均不超过 10^6 。

【输入样例】

```
1 5
2 1 -2 -3 4 5
3 4 2
4 3 1
5 1 2
6 2 5
```

【输出样例】

```
1 8
```

【分析】

状态表示： $f[u]$ 表示以 u 为根的子树中且包含 u 的所有连通块的权值的最大值。

状态计算：假设 s_1, s_2, \dots, s_k 是 u 的子节点，则 $f[u] = w[u] + \max(0, f[s_1]) + \dots + \max(0, f[s_k])$ 。

最后我们遍历每个节点 i ，求出最大的 $f[i]$ 即为答案。

【代码】

```
1 #include <iostream>
2 #include <cstring>
```



```

3  #include <algorithm>
4  using namespace std;
5
6  typedef long long LL;
7  const int N = 100010, M = N << 1;
8  int e[M], ne[M], h[N], idx;
9  LL w[N], f[N];
10 int n;
11
12 void add(int u, int v)
13 {
14     e[idx] = v, ne[idx] = h[u], h[u] = idx++;
15 }
16
17 void dfs(int u, int fa)
18 {
19     f[u] = w[u];
20     for (int i = h[u]; ~i; i = ne[i])
21         if (e[i] != fa)
22             {
23                 dfs(e[i], u);
24                 f[u] += max(0ll, f[e[i]]);
25             }
26 }
27
28 int main()
29 {
30     cin >> n;
31     memset(h, -1, sizeof h);
32     for (int i = 1; i <= n; i++) cin >> w[i];
33     for (int i = 0; i < n - 1; i++)
34     {
35         int a, b;
36         cin >> a >> b;
37         add(a, b), add(b, a);
38     }
39     dfs(1, -1);
40     LL res = f[1];
41     for (int i = 2; i <= n; i++) res = max(res, f[i]);
42     cout << res << endl;
43     return 0;
44 }

```

五、AcWing 1303. 斐波那契前n项和

【题目描述】

大家都知道Fibonacci数列吧， $f_1 = 1, f_2 = 1, f_3 = 2, f_4 = 3, \dots, f_n = f_{n-1} + f_{n-2}$ 。

现在问题很简单，输入 n 和 m ，求 f_n 的前 n 项和 $S_n \bmod m$ 。

【输入格式】

共一行，包含两个整数 n 和 m 。

【输出格式】

输出前 n 项和 $S_n \bmod m$ 的值。

【数据范围】

$$1 \leq n \leq 2 \times 10^9$$

$$1 \leq m \leq 10^9 + 10$$

【输入样例】

```
1 | 5 1000
```

【输出样例】

```
1 | 12
```

【分析】

分析:

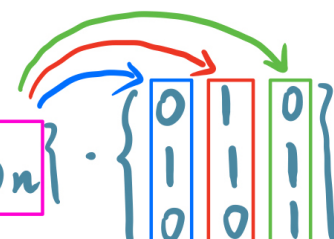
设行矩阵: $F_n = \{f_n, f_{n+1}, S_n\}$

则: $F_{n+1} = \{f_{n+1}, f_{n+2}, S_{n+1}\}$

我们要构造一个矩阵 A , 使得 $F_n \cdot A = F_{n+1}$

那么 $A = \begin{Bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix}$, 则 $F_n = F_1 \cdot A^{n-1}$, 可用快速幂求解

证明:


$$F_n \cdot A = \{f_n, f_{n+1}, S_n\} \cdot \begin{Bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix}$$
$$\left. \begin{aligned} f_n \cdot 0 + f_n \cdot 1 + S_n \cdot 0 &= f_{n+1} \\ f_n \cdot 1 + f_{n+1} \cdot 1 + S_n \cdot 0 &= f_{n+2} \\ f_n \cdot 0 + f_{n+1} \cdot 1 + S_n \cdot 1 &= S_{n+1} \end{aligned} \right\} \{f_{n+1}, f_{n+2}, S_{n+1}\}$$

CSDN @聆歌

【代码】

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  typedef long long LL;
7  const int N = 3;
8  int F[N] = { 1, 1, 1 };
9  int A[N][N] = { { 0, 1, 0 }, { 1, 1, 1 }, { 0, 0, 1 } };
10 int n, m;
11
12 void mul(int c[], int a[], int b[][N])
13 {
14     int temp[N] = { 0 };
15     for (int i = 0; i < N; i++)
16         for (int j = 0; j < N; j++)
```

```

17         temp[i] = (temp[i] + (LL)a[j] * b[j][i]) % m;
18     memcpy(c, temp, sizeof temp);
19 }
20
21 void mul(int c[][N], int a[][N], int b[][N])
22 {
23     int temp[N][N] = { 0 };
24     for (int i = 0; i < N; i++)
25         for (int j = 0; j < N; j++)
26             for (int k = 0; k < N; k++)
27                 temp[i][j] = (temp[i][j] + (LL)a[i][k] * a[k][j]) % m;
28     memcpy(c, temp, sizeof temp);
29 }
30
31 int main()
32 {
33     cin >> n >> m;
34     n--;
35     while (n)
36     {
37         if (n & 1) mul(F, F, A); // F = F * A
38         mul(A, A, A); // A = A * A
39         n >>= 1;
40     }
41     cout << F[2] << endl;
42     return 0;
43 }

```

六、AcWing 1226. 包子凑数

【题目描述】

小明几乎每天早晨都会在一家包子铺吃早餐。

他发现这家包子铺有 N 种蒸笼，其中第 i 种蒸笼恰好能放 A_i 个包子。

每种蒸笼都有非常多笼，可以认为是无限笼。

每当有顾客想买 X 个包子，卖包子的大叔就会迅速选出若干笼包子来，使得这若干笼中恰好一共有 X 个包子。

比如一共有 3 种蒸笼，分别能放 3, 4, 5 个包子。

当顾客想买 11 个包子时，大叔就会选 2 笼 3 个的再加 1 笼 5 个的（也可能选出 1 笼 3 个的再加 2 笼 4 个的）。

当然有时包子大叔无论如何也凑不出顾客想买的数量。

比如一共有**3**种蒸笼，分别能放**4,5,6**个包子。

而顾客想买**7**个包子时，大叔就凑不出来了。

小明想知道一共有多少种数目是包子大叔凑不出来的。

【输入格式】

第一行包含一个整数 **N** 。

接下来 **N** 行，每行包含一个整数 **A_i** 。

【输出格式】

输出一个整数代表答案。

如果凑不出的数目有无限多个，输出 **INF**。

【数据范围】

$$1 \leq N \leq 100$$

$$1 \leq A_i \leq 100$$

【输入样例1】

```
1 2
2 4
3 5
```

【输出样例1】

```
1 6
```

【输入样例2】

```
1 2
2 4
3 6
```

【输出样例2】

```
1 INF
```

【样例解释】

对于样例1，凑不出的数目包括：1, 2, 3, 6, 7, 11。

对于样例2，所有奇数都凑不出来，所以有无限多个。

【分析】

题目一看，是个组合问题，是完全背包问题的变形：有若干种物品，每种物品无限个，选的时候每种物品可以选任意个，求是否能够凑出某个重量。

如果有无限个数凑不出来，说明这些数的最大公约数不是1。这里用到裴蜀定理：任意两个数的组合必定是他们的最大公约数的倍数。同样可以推广到更多数：假设这些数的 gcd 是 g ，那么他们的组合就一定是 g 的倍数，如果 $g \neq 1$ ，那么必然有无限个数无法被组合出来。

那么 $gcd = 1$ 呢？最大不能表示出来的数必定有个上界，因为两个数 a, b （当 $gcd(a, b) = 1$ 时），最大的不能表示出来的数是 $(a - 1)(b - 1) - 1$ 。当数字更多的时候，这个上界必然更小（因为可选的数字变多了），而99和98是100内最大的互质的数， $(99 - 1)(98 - 1) - 1 < 10000$ ，因此这个上界可以选择10000。

那么下面的事情就是看1 ~ 10000中有多少个数不能被组合出来，回到了刚开始分析的完全背包问题：

状态定义： $f[i][j]$ 表示只选前 i 个物品是否能凑出重量 j 。

状态计算：

- $f[i][j] = f[i - 1][j] \mid f[i - 1][j - v_i] \mid f[i - 1][j - 2v_i] \mid \dots \mid f[i - 1][j - kv_i]$
- $f[i][j - v_i] = f[i - 1][j - v_i] \mid f[i - 1][j - 2v_i] \mid \dots \mid f[i - 1][j - kv_i]$

因此状态转移方程为： $f[i][j] = f[i - 1][j] \mid f[i][j - v_i]$ 。

初始化 $f[0][0] = true$ ，最后我们遍历 $f[n][i]$, ($0 \leq i \leq 10000$)，统计出 $f[n][i] = false$ 的数量即为答案。

【代码】

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  const int N = 10010;
7  bool f[N];
8  int n, g;
9
10 int gcd(int a, int b)
```

```

11 {
12     return b ? gcd(b, a % b) : a;
13 }
14
15 int main()
16 {
17     cin >> n;
18     f[0] = true;
19     for (int i = 0; i < n; i++)
20     {
21         int v;
22         cin >> v;
23         g = gcd(g, v);
24         for (int j = v; j <= 10000; j++) f[j] |= f[j - v];
25     }
26     if (g > 1) { puts("INF"); return 0; }
27     int res = 0;
28     for (int i = 0; i <= 10000; i++)
29         if (!f[i]) res++;
30     cout << res << endl;
31     return 0;
32 }

```

七、AcWing 1070. 括号配对

【题目描述】

Hecy又接了个新任务：BE处理。

BE中有一类被称为GBE。

以下是GBE的定义：

- 空表达式是GBE；
- 如果表达式A是GBE，则[A]与(A)都是GBE；
- 如果A与B都是GBE，那么AB是GBE。

下面给出一个BE，求至少添加多少字符能使这个BE成为GBE。

注意：BE是一个仅由() []四种字符中的若干种构成的字符串。

【输入格式】

输入仅一行，为字符串BE。

【输出格式】

输出仅一个整数，表示增加的最少字符数。

【数据范围】

对于所有输入字符串，其长度小于100。

【输入样例】

```
1 | [)]
```

【输出样例】

```
1 | 1
```

【分析】

本题和第三题《密码脱落》基本一样，从当前BE变成GBE需要添加最少字符的数量等价于当前BE变成最长GBE子序列需要去掉字符的数量。即添加的最少字符等价于原字符串长度减去最长GBE子序列的长度。

本题的最长GBE子序列与第三题的最长回文子序列有点区别，本题的序列可以由两个回文序列拼接而成，即不仅 `([])` 属于GBE子序列，`[]()` 也属于GBE子序列，因此状态计算有一点区别。

状态表示： $f[i][j]$ 表示所有在区间 $[i, j]$ 中的GBE子序列的最大长度。

状态计算：

- 当 $s[i]$ 和 $s[j]$ 匹配时， $f[i][j] = f[i+1][j-1] + 2$;
- 当 $s[i]$ 和 $s[j]$ 不匹配时，我们枚举两个GBE子序列的拼接点 $k(i \leq k < j)$ ，则 $f[i][j] = f[i][k] + f[k+1][j]$ 。在枚举 k 的时候我们也会将状态 $f[i+1][j]$ 和 $f[i][j-1]$ 进行计算。

初始化：当区间长度为1时，括号不可能匹配，因此 $f[i][i] = 0$ ，直接从区间长度为2开始计算即可。最后的答案为 $n - f[0][n-1]$ ，其中 n 为原字符串的长度。

【代码】

```
1 #include <iostream>
2 #include <cstring>
3 #include <algorithm>
4 using namespace std;
5
6 const int N = 110;
```



```

7 char s[N];
8 int f[N][N];
9
10 bool check(int l, int r)
11 {
12     if (s[l] == '(' && s[r] == ')') return true;
13     if (s[l] == '[' && s[r] == ']') return true;
14     return false;
15 }
16
17 int main()
18 {
19     cin >> s;
20     int n = strlen(s);
21     for (int len = 2; len <= n; len++)//len为1时一定不匹配
22         for (int i = 0; i + len - 1 < n; i++)
23             {
24                 int j = i + len - 1;
25                 if (check(i, j)) f[i][j] = f[i + 1][j - 1] + 2;
26                 for (int k = i; k < j; k++)
27                     f[i][j] = max(f[i][j], f[i][k] + f[k + 1][j]);
28             }
29     cout << n - f[0][n - 1] << endl;
30     return 0;
31 }

```

八、AcWing 1078. 旅游规划

【题目描述】

W市的交通规划出现了重大问题，市政府下定决心在全市各大交通路口安排疏导员来疏导密集的车流。

但由于人员不足，W市市长决定只在最需要安排人员的路口安排人员。

具体来说，W市的交通网络十分简单，由 n 个交叉路口和 $n - 1$ 条街道构成，交叉路口路口编号依次为 $0, 1, \dots, n - 1$ 。

任意一条街道连接两个交叉路口，且任意两个交叉路口间都存在一条路径互相连接。

经过长期调查的结果显示，如果一个交叉路口位于W市交通网最长路径上，那么这个路口必定拥挤不堪。

所谓最长路径，定义为某条路径 $p = (v_1, v_2, \dots, v_k)$ ，路径经过的路口各不相同，且城市中不存在长度大于 k 的路径（因此最长路径可能不唯一）。

因此W市市长想知道哪些路口位于城市交通网的最长路径上。

【输入格式】

第一行包含一个整数 n 。

之后 $n - 1$ 行每行两个整数 u, v ，表示编号为 u 和 v 的路口间存在着一条街道。

【输出格式】

输出包括若干行，每行包括一个整数表示某个位于最长路径上的路口编号。

为了确保解唯一，请将所有最长路径上的路口编号按编号顺序由小到大依次输出。

【数据范围】

$$1 \leq n \leq 2 \times 10^5$$

【输入样例】

```
1 10
2 0 1
3 0 2
4 0 4
5 0 6
6 0 7
7 1 3
8 2 5
9 4 8
10 6 9
```

【输出样例】

```
1 0
2 1
3 2
4 3
5 4
6 5
7 6
8 8
9 9
```

【分析】

变量定义与解释：我们使用 $dis1[i]$ 表示点 i 往下走所能走的最远距离， $dis2[i]$ 表示点 i 往下走所能走的次远距离（非严格）， $next1[i]$ 表示点 i 往下走的最远路径上的下一个点的编号， $up[i]$ 表示点 i 往上走所能走的最远距离。

树的直径：即树中的最长路径。求解树的直径的其中一种方式是任选一点 x ，求出距离 x 最远的点 y ，然后再以 y 为起点求出距离 y 最远的点的距离，则该距离就为树的直径，但是这种解法不适用于本题，本题需要求解出可能在树的直径（可能有多条直径）上的所有点。那么我们换个思路，树的直径 $maxd$ 就是所有节点往下走的最大距离与次大距离之和中的最大值，即 $maxd = \max(dis1[i] + dis2[i])$ 。

变量求解：求解 dis 时需要用子节点 j 更新父节点 u ，即 u 往 j 这跳路径上走的最远距离就为 $dis1[j] + 1$ ，如果该值大于 $dis1[u]$ ，那么 $dis2[u] = dis1[u]$, $dis1[u] = dis1[j] + 1$ ，即次远距离先更新为最远距离然后再更新最远距离，然后记录 $next1[u] = j$ ；否则如果大于 $dis2[u]$ 则单独更新 $dis2$ 即可。求解 up 时需要用父节点 u 更新子节点 j ， j 往上走有两种情况，一种是走到 u 然后再往上走，即 $up[j] = up[u] + 1$ ，一种是走到 u 然后往下走，往下走的话肯定是优先走最远的那条路，但是如果最远的那条路就是 $u \rightarrow j$ 这条则不能走，因此如果 $next1[u] \neq j$ ，则 $up[j] = dis1[u] + 1$ ，否则 $up[j] = dis2[u] + 1$ 。

如何判断某个点 i 是否在直径上呢？这个点往下走有多条路径，往上走只有一条路径，也就是走到父节点，那么经过这个点的所有路径中最长的那条的长度一定是 $dis1[i]$, $dis2[i]$, $up[i]$ 中较大的两者之和（因为不确定 $dis2[i]$ 和 $up[i]$ 谁更大因此两者都需要考虑）。如果较大的两个数之和等于树的直径，那么这个点就在某条直径上。

【代码】

```
1  #include <iostream>
2  #include <cstring>
3  #include <algorithm>
4  using namespace std;
5
6  const int N = 200010, M = N << 1;
7  int e[M], ne[M], h[N], idx;
8  int dis1[N], dis2[N], next1[N], up[N];
9  int n, maxd; //maxd为直径
10
11 void add(int u, int v)
12 {
13     e[idx] = v, ne[idx] = h[u], h[u] = idx++;
14 }
15
16 void dfs_dis(int u, int fa)
17 {
```

```

18     for (int i = h[u]; ~i; i = ne[i])
19     {
20         int j = e[i];
21         if (j != fa)
22         {
23             dfs_dis(j, u);
24             int dis = dis1[j] + 1;
25             if (dis > dis1[u]) dis2[u] = dis1[u], dis1[u] = dis, next1[u]
= j;
26             else if (dis > dis2[u]) dis2[u] = dis;
27         }
28     }
29     maxd = max(maxd, dis1[u] + dis2[u]);
30 }
31
32 void dfs_up(int u, int fa)
33 {
34     for (int i = h[u]; ~i; i = ne[i])
35     {
36         int j = e[i];
37         if (j != fa)
38         {
39             up[j] = up[u] + 1; //从u往上走
40             if (next1[u] != j) up[j] = max(up[j], dis1[u] + 1); //从u往下
走最长路
41             else up[j] = max(up[j], dis2[u] + 1); //从u往下走次长路
42             dfs_up(j, u);
43         }
44     }
45 }
46
47 int main()
48 {
49     cin >> n;
50     memset(h, -1, sizeof h);
51     for (int i = 0; i < n - 1; i++)
52     {
53         int a, b;
54         cin >> a >> b;
55         add(a, b), add(b, a);
56     }
57     dfs_dis(0, -1);
58     dfs_up(0, -1);

```

```

59     for (int i = 0; i < n; i++)
60     {
61         int dis[3] = { dis1[i], dis2[i], up[i] };
62         sort(dis, dis + 3); //排序方便找出三者中最大的两个
63         if (dis[1] + dis[2] == maxd) cout << i << endl;
64     }
65     return 0;
66 }

```

九、AcWing 1217. 垒骰子

【题目描述】

赌圣atm晚年迷恋上了垒骰子，就是把骰子一个垒在另一个上边，不能歪歪扭扭，要垒成方柱体。

经过长期观察，atm发现了稳定骰子的奥秘：有些数字的面贴着会互相排斥！

我们先来规范一下骰子：1的对面是4，2的对面是5，3的对面是6。

假设有 m 组互斥现象，每组中的那两个数字的面紧贴在一起，骰子就不能稳定的垒起来。

atm想计算一下有多少种不同的可能的垒骰子方式。

两种垒骰子方式相同，当且仅当这两种方式中对应高度的骰子的对应数字的朝向都相同。

由于方案数可能过多，请输出模 $10^9 + 7$ 的结果。

【输入格式】

第一行包含两个整数 n, m ，分别表示骰子的数目和排斥的组数。

接下来 m 行，每行两个整数 a, b ，表示数字 a 和 b 不能紧贴在一起。

【输出格式】

共一个数，表示答案模 $10^9 + 7$ 的结果。

【数据范围】

$$1 \leq n \leq 10^9$$

$$1 \leq m \leq 36$$

$$1 \leq a, b \leq 6$$

【输入样例】

1	2 1
2	1 2

【输出样例】

1	544
---	-----

【分析】

状态表示: $f(i, j)$ 表示所有由 i 个骰子坐在一起,
最上面的数字是 j 的所有合法方案的集合.



状态计算:

如果不考虑互斥问题,每一面朝上都有4种情况,则:

$$f(i, j) = f(i-1, 1) \times 4 + f(i-1, 2) \times 4 + \dots + f(i-1, 6) \times 4$$

假设存在1和2互斥的条件,1的对面是4,2的对面是5.

那么如果第 $i-1$ 个骰子最上面的数字为1,则第 i 个骰子最下面的数字就不能为2,即最上面的数字不能为5.也就是 $f(i, 5)$ 不能由 $f(i-1, 1)$ 转移过来.

因此考虑互斥条件的状态转移方程为:

$$f(i, j) = \alpha_1 f(i-1, 1) + \alpha_2 f(i-1, 2) + \dots + \alpha_6 f(i-1, 6)$$

其中, α_i 为0或4.

我们设矩阵 $F_i = \{f(i,1), f(i,2), \dots, f(i,6)\}$

设存在1和2互斥的条件, 则我们可以构造矩阵A如下:

$$A = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{matrix} & \left(\begin{array}{cccccc} 4 & 4 & 4 & 4 & 0 & 4 \\ 4 & 4 & 4 & 0 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 \end{array} \right) \end{matrix} \quad \text{则 } F_{i-1} \cdot A = F_i$$

我们单独看存在互斥的数字:

$$f(i,4) = 4f(i-1,1) + 0f(i-1,2) + 4f(i-1,3) + \dots$$

$$f(i,5) = 0f(i-1,1) + 4f(i-1,2) + 4f(i-1,3) + \dots$$

这样便能将存在冲突的情况不算入总方案.

即我们设 $op[i]$ 表示 i 对面的数, 如果 x, y 存在冲突, 那么 $A[x][op[y]] = 0, A[y][op[x]] = 0$.

CSDN @聆歌

【代码】

```
1 #include <iostream>
2 #include <cstring>
3 #include <algorithm>
4 using namespace std;
5
6 typedef long long LL;
7 const int N = 6, MOD = 1e9 + 7;
8 int A[N][N], F[N];
9 int op[6] = { 3, 4, 5, 0, 1, 2 }; // op[i]表示i对面的数字
10 int n, m;
11
12 void mul(int c[], int a[], int b[][N])
13 {
14     int temp[N] = { 0 };
15 }
```

```

15     for (int i = 0; i < N; i++)
16         for (int j = 0; j < N; j++)
17             temp[i] = (temp[i] + (LL)a[j] * b[j][i]) % MOD;
18     memcpy(c, temp, sizeof temp);
19 }
20
21 void mul(int c[][N], int a[][N], int b[][N])
22 {
23     int temp[N][N] = { 0 };
24     for (int i = 0; i < N; i++)
25         for (int j = 0; j < N; j++)
26             for (int k = 0; k < N; k++)
27                 temp[i][j] = (temp[i][j] + (LL)a[i][k] * b[k][j]) % MOD;
28     memcpy(c, temp, sizeof temp);
29 }
30
31 int main()
32 {
33     cin >> n >> m;
34     n--;
35     fill(A[0], A[0] + N * N, 4);
36     fill(F, F + N, 4);
37     while (m--)
38     {
39         int a, b;
40         cin >> a >> b;
41         A[a - 1][op[b - 1]] = 0, A[b - 1][op[a - 1]] = 0;
42     }
43     while (n)
44     {
45         if (n & 1) mul(F, F, A);
46         mul(A, A, A);
47         n >>= 1;
48     }
49     LL res = 0;
50     for (int i = 0; i < N; i++) res = (res + F[i]) % MOD;
51     cout << res << endl;
52     return 0;
53 }

```