dp[] //存储结果

假设1为根

//求解关于树的结果，求解dp[1]

dp[1]依赖于这个树本身，依赖于1的孩子节点

dp[i]表示以节点i为根的子树的结果

状态转移：考虑对节点i，如何用dp[j]（j是i的所有孩子）来求解dp[i]，最终求得dp[1]

dp[1]通过递归自动完成底部的求解

x[i] //存储以i为根的子树的直径，目标是x[1]

y[i] //存储i到叶子的最长路径

考虑x[i]的转移过程

任意一棵树的直径：

1.直径是从根到叶子的一条路径 //x1[i]＝1+max(y[j])(j是i的孩子)

2.直径是经过根的一条路径 //x2[i]＝1+max(y[j])+second(y[j])

3.直径是不经过根的一条路径 //x3[i]＝max(x[j])(j是i的孩子)

x[i]=max{x1[i],x2[i],x3[i]}

求直径的个数？

a[i]表示节点i的气氛值

dst：在树T中，选取点集V的一个子集V’，使得Σa[k]（k∈V’）最大，

且满足V’中任意两个点不相邻

dp[i]记录以i为根的子树的最大气氛值

转移时，我们已经知道了i的所有孩子的dp值

dp[i][0]表示i不取的时候，以i为根的子树的最大气氛值

dp[i][1]表示i取的时候，以i为根的子树的最大气氛值

dp[i][0]＝Σmax(dp[j][0], dp[j][1])（j是i的孩子）

dp[i][1]＝a[i]＋Σdp[j][0]（j是i的孩子）

答案：max(dp[1][0], dp[1][1])

void f1(int i, int\* d)

{

求距离i最近的j，距离记为d[i]（j属于这M个点）

}

对于前20分：

if (K==0 || K==1) cout<<0;

if (K==2) { for(i∈V) f1; cout<<min(d);}

40分：

从16个点的点集中选出一个子集，使这个子集中任意两点的距离之和最小

2^16＝1000×64

遍历所有的子集√

100分：

f[i] 表示以i为根的子树的答案

f[i][k] 其中k表示在子树中选取了k个节点

边对于答案的贡献

a b——c d

b的左边选取了x个点，d的右边选取了y个点

bc这条边对于答案的贡献＝x·y·edge[b][c]

i

p q

以i为根的子树的答案，等于

1.i的子节点的子树答案

2.经过i的路径

例：边ip，出现的次数＝子树p中选取的个数×（1＋子树q中选取的个数）

f[i][j]由f[i-1][j]和f[i-1][j-w[i]]来更新

0 0 **0** 0 **0** 0 0 0

0 0 0 0 **0** 0 0 0

所以在采取滚动数组时，从右向左进行更新

在更新蓝色的时候，当前数组中蓝色左边的部分是矩阵中上一行的内容

0 0 0 0 0 0 0 0

0 **0** 0 0 **0** 0 0 0

f[i][j]依赖于f[i][j-k]

更新蓝色时，希望一维数组左边的部分记录的是当前行左边的部分，而不是上一行

所以采取的是从左向右更新

前缀和：

实现：把从第一项到第i项所有值加起来

优点（查找）：在O(1)的时间内求解一段区间内所有元素的和（sum＝s[right]-s[left-1]）

（否则需要用O(长度)的时间来计算这个值）

缺点（更新）：更新a[t]时，s[t+1]、s[t+2]、。。。、s[n]都需要更新

如果一个问题，只在开头输入一些值，后续不更新，而是频繁的查找区间和（前缀和）

如果一个问题，频繁的更新，前缀和×

线段树（查找和更新的折中）

单调队列：把O(m)的窗口内查找时间优化为O(1)，整个问题由O(mn)→O(n)

f

u

p q

//l：p子树中选取的点

j-l：代表的是在u中除p之外的选取的点

k-l：代表的是在整个树中除p之外的选取的点

up边的贡献：p中的每个点与除p之外的所有点产生关联，包括u的父节点f

树型：

用存的方式存储输入数据，构造有根树（记录的父子关系）

设计状态（希望通过dp[1]直接得到整个树的答案，dp[i]表示子树i的答案）

如果一个状态不够，如何考虑设计新的状态

↑

如何状态转移？

在更新dp[i]时需要用到哪些内容，通过更新过程的需求引入新的状态

对于新的状态也要给出转移方程

动态规划优化：

空间：

滚动数组

优先考虑能不能用一维解决（正序/逆序）

→考虑用二维解决（两行交替更新）

时间：

①前缀和（优化查找，牺牲更新），线段树（查找和更新的折中）

②单调栈，单调队列（O(1)时间的窗口内最值维护）

③容器（map，set，STL模板各自的性能）

如果只需要查找，不需要有序：unorderd\_set（查找插入O(1)）

不卡时间T1，卡时间T2

输入一行数a[1]到a[n]，输入一个目标值x，要求输出a中的两个和为x的数

以目标为策略

每输入一个数，把它记入容器，在输入新数的时候，看看容器中有没有(x-新数)存在

不想，选择set，O(nlogn)

选择哈希容器，选择unorderd\_set，每次查找O(1)，整个问题线性时间