易错的点：循环的m有没有写错成n；变量范围；开long

思想：贪心，dfs/bfs，字符串

大致类型：树形（图论），字符串，搜索，动态规划，数论

动态规划dp[i]=min(dp[i],dp[...]+1)；关键字：方案数/可能性，最小/最大，

压缩状态，状态小的情况下。糖果匹配型，棋盘型

区间dp，二维动态规划，划分，缩小

树状dp ，需要vector<int> node[n]来保存子节点

背包，01背包

排序：

归并排序，可以用临时数组存储，最后复制到原数组；也可以先将原数组复制到临时数组里，然后直接修改原数组。

快速排序 双指针

桶排序

堆排序

二叉树

字符串相关

数组、矩阵：

求逆序对

前缀和：s5-s3=a4+a5；5(s3)%3=2，8(s5)%3=2,则8-5%3=0，即如果s3和s5对k的余数相同，则其差值的余数为0。（前缀和取余数法）

并查集int find(x){if(x!=fa[x])fa=find(fa[x]);return fa[x];}

图论/树形，考虑权值、是否只有一颗树，是否需要包含所有节点

最小生成树（包含所有节点）：kruskals算法搭配并查集，有按权值排序；

LCA最近公共祖先（模板没有权值，需要有根节点），用并查集来直接排除不是同一集合的，节省时间；无向图的时候直接从1开始深搜，有向图需要遍历一边parent[]找根节点；a和b的公共祖先有两种情况，一种是a为b的祖先，另一种是a和b在同一深度，还要往上找。

字典树，在输入的时候，也要用一个ptr往下走，

Dijkstra （单源最短路径）从某个点到某个点的最短路径，所以叫单源，有按权值排序，结果不需要所有节点，用一个dis[]存储该点到其他点的距离

//和树状数组的区别，树状数组求整颗树权值的和最小或最大

一层一层遍历树，用队列实现

矩阵存储方式：

链式前向星，或者叫链接表，用于存储邻接表：e[total].next=head[e[total].u]，head[e[total].u]=total++; 无向表的时候记得存两遍

括号等匹配问题想一下栈模拟

动态规划（找状态方程）

确定状态

研究最优策略的最后一步（return dp[s.size] ）

化为子问题

转移方程

根据子问题定义得到

初始条件和边界情况

计算顺序

最值型的动态规划

**基础**

Int（32位）大小约为2\*10(9)

Long long（64位） 约为9\*10（18），long在32位的平台是32，在64位平台为64

//找字母在序列中最后出现的位置

for(int i=0;i<len;i++){

last[s[i]-'a']=i;

}

//字符串遍历比较，匹配字母

For(i){

For(j){

compare(i-j)

}

Bfs使用队列进行记录，用循环来

第一种由小到大，一步一步，小偷隔间问题

第二种背包问题

动态规划是一种解决问题的算法范例，通常用于优化问题，可以分为以下几种主要类型：

0/1 背包问题：这是动态规划中的经典问题之一。在0/1背包问题中，你有一组物品，每个物品都有自己的重量和价值，以及一个固定容量的背包。目标是选择一组物品，使其总重量不超过背包容量，并且总价值最大化。

最长公共子序列（Longest Common Subsequence，LCS）：LCS问题涉及两个序列，通常是字符串，目标是找到这两个序列中的最长公共子序列。这在字符串比对和编辑距离计算中非常有用。

最长递增子序列（Longest Increasing Subsequence，LIS）：LIS问题是寻找一个序列中的最长递增子序列。这在排序和搜索问题中很常见。

class Solution {

public:

int lengthOfLIS(vector<int>& nums) {

int len=nums.size();

if(len==0){

return 0;

}

vector<int> dp(len,1);

for(int i=0;i<len;i++){

for(int j = 0;j<i;j++){

if(nums[i]>nums[j]){

dp[i]=max(dp[i],dp[j]+1);

}

}

}

return \*max\_element(dp.begin(),dp.end());

}

};

矩阵链乘法：在矩阵链乘法问题中，你有一组矩阵，目标是以最有效的方式将它们相乘，以最小化乘法运算的次数。这在计算机图形学和线性代数中很有用。

最短路径问题：这包括单源最短路径问题（如Dijkstra算法和Bellman-Ford算法）和所有对最短路径问题（如Floyd-Warshall算法）。这些算法用于在图形和网络中寻找最短路径。

划分型动态规划：在这种类型的动态规划中，问题被划分成多个子问题，解决这些子问题可以解决原始问题。典型的例子包括切割钢材和最优矩阵链乘法。

线性动态规划：这是一类特殊的动态规划问题，其中问题的状态以线性序列的形式表示。典型的例子包括最长公共子串问题和编辑距离问题。

背包问题的变种：除了0/1背包问题，还有其他背包问题的变种，如多重背包问题、分数背包问题和二维背包问题。

连续区间动态规划：这种类型的动态规划通常用于解决与数组和连续区间相关的问题，如最大子数组和问题。

区域划分动态规划：在这种类型的动态规划中，问题通常与给定区域的划分有关，如划分矩阵链乘法问题。

概率动态规划：这种类型的动态规划涉及不确定性和概率，通常用于决策问题和概率模型中。

矩阵

对角线遍历

For(i=1;i<n;i++)

For(j=i;j<m;j++

unordered\_map<> map

it =map.begin()；map.insert()；map.erase(key)

queue<int> q；q.front()；

vector1=vector2

vector1.assign(vector2.begin(),vector2.end())

//求位数while(num>0) vec.push\_back(num%i);num=num/10

// 辗转相除法求最大公约数

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;}

return a;

// 最小公倍数 = 两数之积 / 最大公约数

int lcm(int a, int b) {

return (a \* b) / gcd(a, b);

//判断素数 朴素思路for(int i=0)if(num%i==0)return true;

//找(1,n)范围内素数

vector<> primes.push\_back(2);

for(int p=3;p\*p<=n;p+=2){

if(isPrme[p/2])

for((int i=p\*p;i<=n;i+=2\*p)

isPrime[i/2]=false;

for(int p=3;p<=n;p+=2) if(isPrime[p/2]){primes.push\_back(p);

//求因子

for (int i = 1; i \* i <= n; ++i) {

if (n % i == 0) {

factors.push\_back(i); // 将 i 加入因子列表

int m=n/i;

if (i != m) {

factors.push\_back(m); // 将 n/i 加入因子列表（除非 i == n/i）

//判断完全平方数for(int i=0;i\*i<=n;i++) if(i\*i==num) factor.push\_back(num);

string s="2022";

//快速转换进制

String s=””

int sum=0;

for(int i=0;i<4;i++){

sum+=s[i]-'0';

sum\*=9;

}

cout<<sum/9;