**ACM-ICPC**

**算法竞赛模板**

**Author：ZUST Team.20**

**Verb ：1.0.2**

浙江科技学院ACM

目录

[1.常用STL - 1 -](#_Toc88473456)

[1.1 SET 基本操作 - 1 -](#_Toc88473457)

[1.2 VECTOR基本操作 - 2 -](#_Toc88473458)

[1.3 STACK 基本操作 - 3 -](#_Toc88473459)

[1.4 QUEUE 基本操作 - 3 -](#_Toc88473460)

[1.5 MAP 基本操作 - 5 -](#_Toc88473461)

[1.6 bitset - 6 -](#_Toc88473462)

[1.7 迭代器 - 7 -](#_Toc88473463)

[1.8 Pii - 7 -](#_Toc88473464)

[2．大数操作 - 8 -](#_Toc88473465)

[2.1高效模板（效率高，但是无法处理负数） - 8 -](#_Toc88473466)

[2.2效率较低的模板（可以处理负数） - 17 -](#_Toc88473467)

[３.图论 - 23 -](#_Toc88473468)

[3.1 最短路 - 23 -](#_Toc88473469)

[３.2 图的连通 - 25 -](#_Toc88473470)

[３.３　欧拉回路 - 29 -](#_Toc88473471)

[３.４TwoSAT - 31 -](#_Toc88473472)

[３.５网络流 - 33 -](#_Toc88473473)

[4．字符串算法 - 40 -](#_Toc88473474)

[４.１ＫＭＰ - 40 -](#_Toc88473475)

[４.２　前缀自动机 - 42 -](#_Toc88473476)

[４.３　马拉车 - 44 -](#_Toc88473477)

[５．动态规划 - 46 -](#_Toc88473478)

[5.0.1动态规划解题的一般步骤 - 46 -](#_Toc88473479)

[5.0.1资料中的解题方法（摘自邓丝雨老师的《dp进阶之路》） - 47 -](#_Toc88473480)

[5.1线性动态规划 - 48 -](#_Toc88473481)

[5.1.1典型例题 - 48 -](#_Toc88473482)

[5.1.1.5子序列问题 - 56 -](#_Toc88473483)

[5.2.2背包问题 - 64 -](#_Toc88473484)

[5.3.区间型动态规划 - 83 -](#_Toc88473485)

[5.4.状态压缩型动态规划 - 88 -](#_Toc88473486)

[5.4.1 预备知识 - 88 -](#_Toc88473487)

[5.4.2 典型例题 - 89 -](#_Toc88473488)

[5.5 数位DP - 109 -](#_Toc88473489)

[5.5.1 数位dp简介与一般做法 - 109 -](#_Toc88473490)

[5.5.2 典型例题 - 111 -](#_Toc88473491)

[5.6 树与图上的dp - 122 -](#_Toc88473492)

[5.7 其他dp问题 - 126 -](#_Toc88473493)

[5.7.1 插头dp (陈丹琦《基于连通性状态压缩的动态规划问题》中的例题，洛谷P5056，ACWing2934) - 126 -](#_Toc88473494)

[5.7.2 dp求方案数（组合数学+dp）2020icpc上海E - 135 -](#_Toc88473495)

[６．数据结构 - 138 -](#_Toc88473496)

[６.1线段树 - 138 -](#_Toc88473497)

[6.2 分块 - 146 -](#_Toc88473498)

[6.3 cdq分治 - 150 -](#_Toc88473499)

[６.４莫队 - 154 -](#_Toc88473500)

[6.5 可持久数据结构 - 164 -](#_Toc88473501)

[6.6 ST表 - 170 -](#_Toc88473502)

[6.7动态树之LCT - 171 -](#_Toc88473503)

[6.8 树链剖分 - 175 -](#_Toc88473504)

[7．数学模块 - 188 -](#_Toc88473505)

[7.1 扩展欧几里德与贝祖定理 - 188 -](#_Toc88473506)

[7.1.1 概述 - 188 -](#_Toc88473507)

[7.1.2 经典例题 - 189 -](#_Toc88473508)

[7.2欧拉函数 - 195 -](#_Toc88473509)

[7.2.1概述 - 195 -](#_Toc88473510)

[7.2.2 经典例题 - 197 -](#_Toc88473511)

[7.3中国剩余定理与拓展中国剩余定理 - 204 -](#_Toc88473512)

[7.3.1 中国剩余定理 - 204 -](#_Toc88473513)

[7.3.2 扩展中国剩余定理 - 205 -](#_Toc88473514)

[7.3.4经典例题 - 207 -](#_Toc88473515)

[7.4 线性同余方程 - 211 -](#_Toc88473516)

[7.4.1 概述 - 211 -](#_Toc88473517)

[7.4.2经典例题 - 211 -](#_Toc88473518)

[7.5逆元 - 216 -](#_Toc88473519)

[7.5.1 概述 - 216 -](#_Toc88473520)

[7.5.2 扩欧求逆元 - 216 -](#_Toc88473521)

[7.5.3 费马小定理求逆元 - 216 -](#_Toc88473522)

[7.5.4 线性求逆元 - 217 -](#_Toc88473523)

[7.5.5 经典例题 - 218 -](#_Toc88473524)

[7.5 BSGS与扩展BSGS - 220 -](#_Toc88473525)

[7.6.1 概述 - 220 -](#_Toc88473526)

[7.6.2 普通BSGS - 220 -](#_Toc88473527)

[7.6.3 扩展BSGS - 222 -](#_Toc88473528)

[7.6.4 经典例题 - 225 -](#_Toc88473529)

[7.7 高斯消元法 - 228 -](#_Toc88473530)

[7.1.1.概述 - 228 -](#_Toc88473531)

[7.1.2.模板 - 228 -](#_Toc88473532)

[7.7.3.经典例题 - 231 -](#_Toc88473533)

[7.8 线性基 - 241 -](#_Toc88473534)

[7.8.1.概述 - 241 -](#_Toc88473535)

[7.8.2.预备知识 - 241 -](#_Toc88473536)

[7.8.3.定义 - 241 -](#_Toc88473537)

[7.8.4.二进制求线性基 - 242 -](#_Toc88473538)

[7.8.5.判断某数能否由线性基表示 - 243 -](#_Toc88473539)

[7.8.6.求异或和最大 - 243 -](#_Toc88473540)

[7.8.7.求异或和最小 - 244 -](#_Toc88473541)

[7.8.8.求异或和第K大 - 244 -](#_Toc88473542)

[7.8.9.经典例题： - 245 -](#_Toc88473543)

[7.9 排列组合数 - 255 -](#_Toc88473544)

[7.9.1 概述 - 255 -](#_Toc88473545)

[7.9.2各种排列 - 255 -](#_Toc88473546)

[7.9.3三种求组合数的方法 - 257 -](#_Toc88473547)

[7.10斯特林数 - 260 -](#_Toc88473548)

[7.10.1第一类斯特林数 - 260 -](#_Toc88473549)

[7.10.2第二类斯特林数 - 261 -](#_Toc88473550)

[7.11卡特兰数 - 262 -](#_Toc88473551)

[7.11.1概述 - 262 -](#_Toc88473552)

[7.11.2.常见公式 - 262 -](#_Toc88473553)

[7.11.3.经典例题 - 263 -](#_Toc88473554)

[7.12博弈论 - 270 -](#_Toc88473555)

[7.12.1 Bash - 270 -](#_Toc88473556)

[7.13 矩阵 - 274 -](#_Toc88473557)

[8.计算几何 - 277 -](#_Toc88473558)

[8.1凸包模板 - 277 -](#_Toc88473559)

**基本内容与公式**

1. **泰勒公式**

文本, 信件

描述已自动生成

**2）解三角形**

**图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成**

**文本, 信件

描述已自动生成3）枚举子集**

1. **for**(**int** S=0;S<(1<<n);S++){
2. **for**(**int** T=S;T;T=(T-1)&S){
4. }
5. }

**4）\_\_int128型整数输入输出（重载运算符，使用cin、cout输入输出）**

1. istream & operator>>(istream& is, \_\_int128 &t) {
2. string s;
3. **bool** f=0;
4. is>>s;
5. t=0;
6. **for**(**int** i=0;i<s.size();i++){
7. **if**(s[i]=='-'){
8. f=1;
9. **continue**;
10. }
11. t\*=10;
12. t+=s[i]-'0';
13. }
14. **if**(f)t=-t;
15. **return** is;
16. }
18. ostream& operator<<(ostream& os, \_\_int128 t) {
19. **if** (t==0) **return** os << "0";
20. **if** (t<0) {
21. os<<"-";
22. t=-t;
23. }
24. **int** a[50],ai=0;
25. memset(a,0,**sizeof** a);
26. **while** (t!=0){
27. a[ai++]=t%10;
28. t/=10;
29. }
30. **for** (**int** i=1;i<=ai;i++) os<<abs(a[ai-i]);
31. **return** os<<"";
32. }

# 1.常用STL

## SET 基本操作

1. s.begin()       //  返回指向第一个元素的迭代器
2. s.clear()       //  清除所有元素
3. s.count()       //  返回某个值元素的个数
4. s.empty()       //  如果集合为空，返回true(真）
5. s.end()         //  返回指向最后一个元素之后的迭代器，不是最后一个元素
6. s.equal\_range() //  返回集合中与给定值相等的上下限的两个迭代器
7. s.erase()       //  删除集合中的元素
8. s.find()        //  返回一个指向被查找到元素的迭代器
9. s.get\_allocator()   //  返回集合的分配器
10. s.insert()      //  在集合中插入元素
11. s.lower\_bound() //  返回指向大于（或等于）某值的第一个元素的迭代器
12. s.key\_comp()    //  返回一个用于元素间值比较的函数
13. s.max\_size()    //  返回集合能容纳的元素的最大限值
14. s.rbegin()      //  返回指向集合中最后一个元素的反向迭代器
15. s.rend()        //  返回指向集合中第一个元素的反向迭代器
16. s.size()        //  集合中元素的数目
17. s.swap()        //  交换两个集合变量
18. s.upper\_bound() //  返回大于某个值元素的迭代器
19. s.value\_comp()  //  返回一个用于比较元素间的值的函数

多重集合与集合的区别在于集合中不能存在相同元素，而多重集合中可以存在。

1. multiset<**int**> s;
2. multiset<**double**> ss;

multiset和set的基本操作相似，需要注意的是，集合的count()能返回0（无）或者1（有），而多重集合是有多少个返回多少个

## 1.2 VECTOR基本操作

1. vector<**int**> s;
2. //  定义一个空的vector对象，存储的是int类型的元素
3. vector<**int**> s(n);
4. //  定义一个含有n个int元素的vector对象
5. vector<**int**> s(first, last);
6. //  定义一个vector对象，并从由迭代器first和last定义的序列[first, last)中复制初值

**Vector的基本操作**

1. s[i]                //  直接以下标方式访问容器中的元素
2. s.front()           //  返回首元素
3. s.back()            //  返回尾元素
4. s.push\_back(x)      //  向表尾插入元素x
5. s.size()            //  返回表长
6. s.empty()           //  表为空时，返回真，否则返回假
7. s.pop\_back()        //  删除表尾元素
8. s.begin()           //  返回指向首元素的随机存取迭代器
9. s.end()             //  返回指向尾元素的下一个位置的随机存取迭代器
10. s.insert(it, val)   //  向迭代器it指向的元素前插入新元素val
11. s.insert(it, n, val)//  向迭代器it指向的元素前插入n个新元素val
12. s.insert(it, first, last)
13. //将由迭代器first和last所指定的序列[first, last)插入到迭代器it指向的元素前面
14. s.erase(it)         //  删除由迭代器it所指向的元素
15. s.erase(first, last)//  删除由迭代器first和last所指定的序列[first, last)
16. s.reserve(n)        //  预分配缓冲空间，使存储空间至少可容纳n个元素
17. s.resize(n)         //  改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），元素默认值将填满扩展出的空间
18. s.resize(n, val)    //  改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），val将填满扩展出的空间
19. s.clear()           //  删除容器中的所有元素
20. s.swap(v)           //  将s与另一个vector对象进行交换
21. s.assign(first, last)
22. //  将序列替换成由迭代器first和last所指定的序列[first, last)，[first, last)不能是原序列中的一部分 ，要注意的是，resize操作和clear操作都是对表的有效元素进行的操作，但并不一定会改变缓冲空间的大小
23. //  另外，vector还有其他的一些操作，如反转、取反等，不再一一列举
24. //  vector上还定义了序列之间的比较操作运算符（>、<、>=、<=、==、!=），可以按照字典序比较两个序列。

## 1.3 STACK 基本操作

**Stack的基本操作**：

1. s.push(x);  //  入栈
2. s.pop();    //  出栈
3. s.top();    //  访问栈顶
4. s.empty();  //  当栈空时，返回true
5. s.size();   //  访问栈中元素个数

## 1.4 QUEUE 基本操作

**Queue的基本操作：**

1. q.push(x);  //  入队列
2. q.pop();    //  出队列
3. q.front();  //  访问队首元素
4. q.back();   //  访问队尾元素
5. q.empty();  //  判断队列是否为空
6. q.size();   //  访问队列中的元素个数

**优先队列：**

1. priority\_queue<**int**> q;
2. priority\_queue<pair<**int**, **int**> > qq;                 //  注意在两个尖括号之间一定要留空格，防止误判
3. priority\_queue<**int**, vector<**int**>, greater<**int**> > qqq;//  定义小的先出队列
4. 优先队列的基本操作：
5. q.empty()     //  如果队列为空，则返回true，否则返回false
6. q.size()      //  返回队列中元素的个数
7. q.pop()       //  删除队首元素，但不返回其值
8. q.top()       //  返回具有最高优先级的元素值，但不删除该元素
9. q.push(item)  //  在基于优先级的适当位置插入新元素
10. #include <iostream>
11. #include <queue>
12. **using** **namespace** std;
13. **class** T
14. {
15. **public**:
16. **int** x, y, z;
17. T(**int** a, **int** b, **int** c) : x(a), y(b), z(c) {}
18. };
19. **bool** operator < (**const** T &tOne, **const** T &tTwo)
20. {
21. **return** tOne.z < tTwo.z;  //  按照z的顺序来决定tOne和tTwo的顺序
22. }
23. **int** main()
24. {
25. priority\_queue<T> q;
26. q.push(T(4, 4, 3));
27. q.push(T(2, 2, 5));
28. q.push(T(1, 5, 4));
29. q.push(T(3, 3, 6));
31. **while** (!q.empty())
32. {
33. T t = q.top();
34. q.pop();
35. cout << t.x << " " << t.y << " " << t.z << '\n';
36. }
37. **return** 0;
38. }

## 1.5 MAP 基本操作

1. /\*  向map中插入元素  \*/
2. m[key] = value; //  [key]操作是map很有特色的操作,如果在map中存在键值为key的元素对, 则返回该元素对的值域部分,否则将会创建一个键值为key的元素对,值域为默认值。所以可以用该操作向map中插入元素对或修改已经存在的元素对的值域部分。
3. m.insert(make\_pair(key, value));    //  也可以直接调用insert方法插入元素对,insert操作会返回一个pair,当map中没有与key相匹配的键值时,其first是指向插入元素对的迭代器,其second为true;若map中已经存在与key相等的键值时,其first是指向该元素对的迭代器,second为false。
4. /\*  查找元素  \*/
5. **int** i = m[key]; //  要注意的是,当与该键值相匹配的元素对不存在时,会创建键值为key（当另一个元素是整形时，m[key]=0）的元素对。
6. map<string, **int**>::iterator it = m.find(key);    //  如果map中存在与key相匹配的键值时,find操作将返回指向该元素对的迭代器,否则,返回的迭代器等于map的end()(参见vector中提到的begin()和end()操作)。
8. /\*  删除元素  \*/
9. m.erase(key);   //  删除与指定key键值相匹配的元素对,并返回被删除的元素的个数。
10. m.erase(it);    //  删除迭代器it所指定的元素对,并返回指向下一个元素对的迭代器。
12. /\*  其他操作  \*/
13. m.size();       //  返回元素个数
14. m.empty();      //  判断是否为空
15. m.clear();      //  清空所有元素

**二维map应用：**

1. map<string,map<string,**int**> > mp;
2. map<string,map<string,**int**> >::iterator it;
3. map<string,**int**>::iterator it2;
4. **for**(it=mp.begin();it!=mp.end();it++){
5. cout<<it->first<<endl;
6. **for**(it2=it->second.begin();it2!=it->second.end();it2++){
7. cout<<"   |----"<<it2->first<<"("<<it2->second<<")"<<endl;
8. }
9. }
10. }

## 1.6 bitset

在 STLSTL 的头文件中 <bitset>中定义了模版类 bitset，用来方便地管理一系列的 bitbit 位的类。bitset 除了可以访问指定下标的 bitbit 位以外，还可以把它们作为一个整数来进行某些统计。

bitset 模板类需要一个模版参数，用来明确指定含有多少位

定义bitset对象的示例代码：

1. **const** **int** MAXN = 32;
3. bitset<MAXN> bt;            //  bt 包括 MAXN 位，下标 0 ~ MAXN - 1，默认初始化为 0
4. bitset<MAXN> bt1(0xf);      //  0xf 表示十六进制数 f，对应二进制 1111，将 bt1 低 4 位初始化为 1
5. bitset<MAXN> bt2(012);      //  012 表示八进制数 12，对应二进制 1010，即将 bt2 低 4 位初始化为 1010
6. bitset<MAXN> bt3("1010");   //  将 bt3 低 4 位初始化为 1010
7. bitset<MAXN> bt4(s, pos, n);//  将 01 字符串 s 的 pos 位开始的 n 位初始化 bt4
9. bitset的基本操作：
10. bt.any()        //  bt 中是否存在置为 1 的二进制位？
11. bt.none()       //  bt 中不存在置为 1 的二进制位吗？
12. bt.count()      //  bt 中置为 1 的二进制位的个数
13. bt.size()       //  bt 中二进制位的个数
14. bt[pos]         //  访问 bt 中在 pos 处的二进制位
15. bt.test(pos)    //  bt 中在 pos 处的二进制位是否为 1
16. bt.set()        //  把 bt 中所有二进制位都置为 1
17. bt.set(pos)     //  把 bt 中在 pos 处的二进制位置为 1
18. bt.reset()      //  把 bt 中所有二进制位都置为 0
19. bt.reset(pos)   //  把 bt 中在pos处的二进制位置为0
20. bt.flip()       //  把 bt 中所有二进制位逐位取反
21. bt.flip(pos)    //  把 bt 中在 pos 处的二进制位取反
22. bt[pos].flip()  //  同上
23. bt.to\_ulong()   //  用 bt 中同样的二进制位返回一个 unsigned long 值
24. os << bt        //  把 bt 中的位集输出到 os 流

## 1.7 迭代器

1. #include <iostream>
2. #include <vector>
3. **using** **namespace** std;
4. **int** main()
5. {
6. vector<**int**> s;
7. s.push\_back(1);
8. s.push\_back(2);
9. s.push\_back(3);
10. copy(s.begin(), s.end(), ostream\_iterator<**int**> (cout, " "));
11. cout << '\n';
12. **return** 0;
13. }

这段代码中的copy就是STL中定义的一个模版函数，copy(s.begin(), s.end(), ostream\_iterator<int>(cout, ” “));的意思是将由s.begin()至s.end()（不含s.end()）所指定的序列复制到标准输出流out中，用” “作为每个元素的间隔。也就是说，这句话的作用其实就是将表中的所有内容依次输出

## 1.8 Pii

STL的<utility>头文件中描述了一个看上去非常简单的模版类pair，用来表示一个二元组或元素对，并提供了按照字典序对元素对进行大小比较运算符模版函数。   
Example，想要定义一个对象表示一个平面坐标点，则可以：

1. pair<**double**, **double**> p;
2. cin >> p.first >> p.second;

# 2．大数操作

## 2.1高效模板（效率高，但是无法处理负数）

此模板支持使用cin读入大数，通过重载运算符的方式来实现跟int、long long一样的运算，效率较高。缺点：无法处理负数

**代码如下：**

1. #include<bits/stdc++.h>
2. #define mul(a,b,c) (ll(a)\*(b)%(c))
3. #define inv(a,b) pow(a,(b)-2,b)
4. **using** **namespace** std;
5. **typedef** **long** **long** ll;
6. **typedef** **double** lf;
7. **const** lf PI=acos(-1);
8. ll pow(ll a,ll b,ll m)
9. {
10. ll r=1;
11. **for**(a%=m; b; b>>=1,a=mul(a,a,m))
12. **if**(b&1)r=mul(r,a,m);
13. **return** r;
14. }
15. **struct** Rader:vector<**int**>
16. {
17. Rader(**int** n):vector<**int**>(1<<**int**(ceil(log2(n))))
18. {
19. **for**(**int** i=at(0)=0; i<size(); ++i)
20. **if**(at(i)=at(i>>1)>>1,i&1)
21. at(i)+=size()>>1;
22. }
23. };
24. **struct** FFT:Rader
25. {
26. vector<complex<lf> > w;
27. FFT(**int** n):Rader(n),w(size(),polar(1.0,2\*PI/size()))
28. {
29. w[0]=1;
30. **for**(**int** i=2; i<size(); ++i)w[i]\*=w[i-1];
31. }
32. vector<complex<lf> > fft(**const** vector<complex<lf> > &a)**const**
33. {
34. vector<complex<lf> > x(size());
35. **for**(**int** i=0; i<a.size(); ++i)x[at(i)]=a[i];
36. **for**(**int** i=1; i<size(); i<<=1)
37. **for**(**int** j=0; j<i; ++j)
38. **for**(**int** k=j; k<size(); k+=i<<1)
39. {
40. complex<lf> &l=x[k],&r=x[k+i],t=w[size()/(i<<1)\*j]\*r;
41. r=l-t,l+=t;
42. }
43. **return** x;
44. }
45. vector<ll> ask(**const** vector<ll> &a,**const** vector<ll> &b)**const**
46. {
47. vector<complex<lf> > xa(a.begin(),a.end()),xb(b.begin(),b.end());
48. xa=fft(xa),xb=fft(xb);
49. **for**(**int** i=0; i<size(); ++i)xa[i]\*=xb[i];
50. vector<ll> ans(size());
51. xa=fft(xa),ans[0]=xa[0].real()/size()+0.5;
52. **for**(**int** i=1; i<size(); ++i)ans[i]=xa[size()-i].real()/size()+0.5;
53. **return** ans;
54. }
55. };
56. **struct** FNTT:Rader
57. {
58. ll M,G;
59. vector<ll> w;
60. FNTT(**int** N,ll M,ll G):Rader(N),M(M),G(G),w(size(),pow(G,(M-1)/size(),M))//M费马素数，G原根
61. {
62. **for**(**int** i=w[0]=1; i<size(); ++i)w[i]=mul(w[i],w[i-1],M);
63. }
64. vector<ll> fntt(**const** vector<ll> &a)**const**
65. {
66. vector<ll> x(size());
67. **for**(**int** i=0; i<a.size(); ++i)x[at(i)]=a[i];
68. **for**(**int** i=1,j; i<size(); i<<=1)
69. **for**(**int** j=0; j<i; ++j)
70. **for**(**int** k=j; k<size(); k+=i<<1)
71. {
72. ll t=mul(w[size()/(i<<1)\*j],x[k+i],M);
73. **if**(x[k+i]=x[k]-t,x[k+i]<0)x[k+i]+=M;
74. **if**(x[k]+=t,x[k]>=M)x[k]-=M;
75. }
76. **return** x;
77. }
78. vector<ll> ask(vector<ll> a,vector<ll> b)**const**
79. {
80. a=fntt(a),b=fntt(b);
81. **for**(**int** i=0; i<size(); ++i)a[i]=mul(a[i],b[i],M);
82. a=fntt(a),reverse(a.begin()+1,a.end());
83. ll u=inv(size(),M);
84. **for**(**int** i=0; i<size(); ++i)a[i]=mul(a[i],u,M);
85. **return** a;
86. }
87. };
88. **struct** Wint:vector<**int**>//继承vector
89. {
90. **static** **const** **int** width=9,base=1e9;
91. Wint(unsigned **long** **long** n=0)//普通初始化，当整型数和Wint同时运算时会提升至Wint
92. {
93. **for**(; n; n/=base)push\_back(n%base);
94. }
95. **explicit** Wint(**const** string &s)//字符串初始化函数，未判断字符串合法情况
96. {
97. **for**(**int** len=**int**(s.size()-1)/width+1,b,e,i=0; i<len; ++i)
98. **for**(e=s.size()-i\*width,b=max(0,e-width),push\_back(0); b!=e; ++b)
99. back()=back()\*10+s[b]-'0';
100. trim(0);
101. }
102. Wint& trim(**bool** up=1)//去前导0，是否需要进位，很常用的小函数，为方便返回自身
103. {
104. **for**(**int** i=1; up&&i<size(); ++i)
105. {
106. **if**((\***this**)[i-1]<0)--(\***this**)[i],(\***this**)[i-1]+=base;
107. **if**((\***this**)[i-1]>=base)(\***this**)[i]+=(\***this**)[i-1]/base,(\***this**)[i-1]%=base;
108. }
109. **while**(!empty()&&back()<=0)pop\_back();
110. **for**(; up&&!empty()&&back()>=base; (\***this**)[size()-2]%=base)
111. push\_back(back()/base);
112. **return** \***this**;
113. }
114. **friend** istream& operator>>(istream &is,Wint &n)
115. {
116. string s;//懒
117. **return** is>>s,n=Wint(s),is;
118. }
119. **friend** ostream& operator<<(ostream &os,**const** Wint &n)
120. {
121. **if**(n.empty())**return** os.put('0');
122. os<<n.back();
123. **char** ch=os.fill('0');
124. **for**(**int** i=n.size()-2; ~i; --i)
125. os.width(n.width),os<<n[i];
126. **return** os.fill(ch),os;
127. }
128. **friend** **bool** operator<(**const** Wint &a,**const** Wint &b)
129. {
130. **if**(a.size()!=b.size())**return** a.size()<b.size();
131. **for**(**int** i=a.size()-1; ~i; --i)
132. **if**(a[i]!=b[i])**return** a[i]<b[i];
133. **return** 0;
134. }
135. **friend** **bool** operator>(**const** Wint &a,**const** Wint &b)
136. {
137. **return** b<a;
138. }
139. **friend** **bool** operator<=(**const** Wint &a,**const** Wint &b)
140. {
141. **return** !(a>b);
142. }
143. **friend** **bool** operator>=(**const** Wint &a,**const** Wint &b)
144. {
145. **return** !(a<b);
146. }
147. Wint& operator+=(**const** Wint &b)
148. {
149. **if**(size()<b.size())resize(b.size());//保证有足够的位数
150. **for**(**int** i=0; i<b.size(); ++i)(\***this**)[i]+=b[i];
151. **return** trim();//单独进位防自运算
152. }
153. **friend** Wint operator+(Wint a,**const** Wint &b)
154. {
155. **return** a+=b;
156. }
157. Wint& operator++()//前置版本
158. {
159. **return** \***this**+=1;//懒
160. }
161. Wint operator++(**int**)//后置版本
162. {
163. Wint b(\***this**);
164. **return** ++\***this**,b;
165. }
166. Wint& operator-=(**const** Wint &b)//a<b会使a变为0
167. {
168. **if**(size()<b.size())resize(b.size());//保证有足够的位数
169. **for**(**int** i=0; i<b.size(); ++i)(\***this**)[i]-=b[i];
170. **return** trim();//单独进位防自运算
171. }
172. **friend** Wint operator-(Wint a,**const** Wint &b)
173. {
174. **return** a-=b;
175. }
176. Wint& operator--()//前置版本
177. {
178. **return** \***this**-=1;//懒
179. }
180. Wint operator--(**int**)//后置版本
181. {
182. Wint b(\***this**);
183. **return** --\***this**,b;
184. }
185. Wint& operator\*=(**const** Wint &b)//高精度乘法，常规写法
186. {
187. Wint c;
188. c.assign(size()+b.size(),0);//多开位用于进位
189. **for**(**int** j=0,k,l; j<b.size(); ++j)
190. **if**(b[j])//稀疏优化，特殊情况很有效
191. **for**(**int** i=0; i<size(); ++i)
192. {
193. unsigned **long** **long** n=(\***this**)[i];
194. **for**(n\*=b[j],k=i+j; n; n/=base)
195. c[k++]+=n%base;
196. **for**(l=i+j; c[l]>=base||l+1<k; c[l++]%=base)
197. c[l+1]+=c[l]/base;
198. }
199. **return** swap(c),trim(0);
200. }
201. /\*
202. Wint& operator\*=(const Wint &b)//一种效率略高但对位宽有限制的写法
203. {
204. vector<unsigned long long> n(size()+b.size(),0);//防爆int，多开位用于进位
205. //乘法算完后统一进位效率高，防止乘法溢出（unsigned long long范围0~1.8e19）
206. //位宽为9时size()不能超过18（十进制162位），位宽为8时size()不能超过1800（十进制14400位）等等。
207. for(int j=0; j!=b.size(); ++j)
208. if(b[j])//稀疏优化，特殊情况很有效
209. for(int i=0; i!=size(); ++i)
210. n[i+j]+=(unsigned long long)(\*this)[i]\*b[j];
211. for(int i=1; i<n.size(); ++i)//这里用<防止位数0，单独进位防自运算
212. n[i]+=n[i-1]/base,n[i-1]%=base;
213. return assign(n.begin(),n.end()),trim(0);
214. }
215. Wint& operator\*=(const Wint &b)//fft优化乘法，注意double仅15位有效数字，调小Wint::width不超过2，计算自2\*log2(base)+2\*log2(len)<53
216. {
217. vector<ll> ax(begin(),end()),bx(b.begin(),b.end());
218. ax=FFT(size()+b.size()).ask(ax,bx);
219. for(int i=1; i<ax.size(); ++i)
220. ax[i]+=ax[i-1]/base,ax[i-1]%=base;
221. return assign(ax.begin(),ax.end()),trim(0);
222. }
223. Wint& operator\*=(const Wint &b)//ntt优化，Wint::width不超过2
224. {
225. vector<ll> ax(begin(),end()),bx(b.begin(),b.end());
226. ax=FNTT(size()+b.size(),(7<<26)+1,3).ask(ax,bx);
227. for(int i=1; i<ax.size(); ++i)
228. ax[i]+=ax[i-1]/base,ax[i-1]%=base;
229. return assign(ax.begin(),ax.end()),trim(0);
230. }
231. \*/
232. **friend** Wint operator\*(Wint a,**const** Wint &b)
233. {
234. **return** a\*=b;
235. }
236. Wint& operator/=(Wint b)
237. {
238. Wint r,c,d=b.base/(b.back()+1);
239. \***this**\*=d,b\*=d,c.assign(size(),0);
240. **for**(**int** i=size()-1; ~i; --i)
241. {
242. r.insert(r.begin(),(\***this**)[i]);
243. unsigned **long** **long** s=0;
244. **for**(**int** j=b.size(); j+1>=b.size(); --j)//b.size()==0肯定第一行就出问题的
245. s=s\*b.base+(j<r.size()?r[j]:0);
246. **for**(d=c[i]=s/b.back(),d\*=b; r<d; r+=b)--c[i];
247. r-=d;
248. }
249. **return** swap(c),trim(0);//r为加倍后的余数，可通过高精度除低精度得到真正余数，此处略
250. }
251. **friend** Wint operator/(Wint a,**const** Wint &b)
252. {
253. **return** a/=b;
254. }
255. Wint& operator%=(**const** Wint &b)
256. {
257. **return** \***this**-=\***this**/b\*b;
258. }
259. **friend** Wint operator%(Wint a,**const** Wint &b)
260. {
261. **return** a%=b;
262. }
263. //开平方，来自ZJU模板，Wint::width设置为9的时候能140ms极速A题，设置为1的时候输入101输出11？待研究。
264. **bool** cmp(**long** **long** c,**int** d,**const** Wint &b)**const**
265. {
266. **long** **long** l=-(base<<1),t=0;
267. **if**(b.size()<size()+d&&c)**return** 1;
268. **for**(**int** i=b.size()-1; l<=t&&t<=0&&i>d; --i)
269. t=t\*base+c\*(i-d-1<size()?(\***this**)[i-d-1]:0)-b[i];
270. **for**(**int** i=d-1; l<=t&&t<=0&&~i; --i)
271. t=t\*base-b[i];
272. **return** t>0;
273. }
274. Wint& sub(**const** Wint &b,**long** **long** k,**int** d)
275. {
276. **int** l=b.size()+d;
277. **for**(**int** i=d+1; i<=l; ++i)
278. {
279. **long** **long** tmp=(\***this**)[i]-k\*b[i-d-1];
280. **if**(tmp<0)
281. {
282. (\***this**)[i+1]+=(tmp-base+1)/base;
283. (\***this**)[i]=tmp-(tmp-base+1)/base\*base;
284. }
285. **else** (\***this**)[i]=tmp;
286. }
287. **for**(**int** i=l+1; i<size()&&(\***this**)[i]<0; ++i)
288. {
289. (\***this**)[i+1]+=((\***this**)[i]-base+1)/base;
290. (\***this**)[i]-=((\***this**)[i]-base+1)/base\*base;
291. }
292. **return** trim(0);
293. }
294. **friend** Wint sqrt(Wint a)
295. {
296. Wint ret;
297. ret.assign((a.size()+1)>>1,0);
298. **for**(**int** i=ret.size()-1; ~i; --i)
299. {
300. **int** l=0,r=a.base,m=ret[i]=(l+r)>>1;
301. **while**(r-l>1)
302. {
303. **if**(ret.cmp(m,i-1,a))r=m;
304. **else** l=m;
305. m=ret[i]=(l+r)>>1;
306. }
307. a.sub(ret,m,i-1);
308. ret[i]+=m;
309. }
310. ret.trim(0);
311. **for**(**int** i=0; i<ret.size(); ++i)ret[i]>>=1;
312. **return** ret;
313. }
314. /\*
315. friend Wint sqrt(const Wint &a)//常规牛顿迭代实现的开平方算法，慢但是好敲
316. {
317. Wint b=a,c=(b+1)/2;
318. while(b!=c)swap(b,c),c=(b+a/b)/2;
319. return c;
320. }
321. friend Wint sqrt(const Wint &a)
322. {
323. Wint ret,t;
324. ret.assign((a.size()+1)>>1,0);
325. for(int i=ret.size()-1,l,r; ~i; --i)
326. {
327. for(l=0,r=a.base; r-l>1;)
328. {
329. ret[i]=l+(r-l)/2;
330. t=ret\*ret;
331. if(a<t)r=ret[i];
332. else l=ret[i];
333. }
334. if(!l&&i==ret.size()-1)ret.pop\_back();
335. else ret[i]=l;
336. }
337. return ret;
338. }
339. \*/
340. };
341. **int** main()
342. {
343. Wint a,b,c;
344. cin>>a>>b;
345. cout<<a+b<<'\n';
346. **if**(a<b)cout<<'-'<<b-a;
347. **else** cout<<a-b;
348. cout<<'\n'<<a\*b<<'\n';
349. c=a/b,a-=c\*b;
350. cout<<c<<'\n'<<a;
351. }

## 2.2**效率较低的模板（可以处理负数**）

本模板可以处理负数，但是相应的效率有所降低，使用读入字符串的方式，调用构造函数构造对象。

**代码如下：**

1. #include <bits/stdc++.h>
2. **class** BigInt
3. {
4. #define Value(x, nega) ((nega) ? -(x) : (x))
5. #define At(vec, index) ((index) < vec.size() ? vec[(index)] : 0)
6. //C风格的比较函数,其正负等于abs(lhs)-abs(rhs)的正负
7. **static** **int** absComp(**const** BigInt &lhs, **const** BigInt &rhs)
8. {
9. **if** (lhs.size() != rhs.size())
10. **return** lhs.size() < rhs.size() ? -1 : 1;
11. **for** (**int** i = lhs.size() - 1; i >= 0; --i)
12. **if** (lhs[i] != rhs[i])
13. **return** lhs[i] < rhs[i] ? -1 : 1;
14. **return** 0;
15. }
16. **using** Long = **long** **long**;
17. **const** **static** **int** Exp = 9;
18. **const** **static** Long Mod = 1000000000;
19. **mutable** std::vector<Long> val;
20. **mutable** **bool** nega = **false**;
21. //规定:0的nega必须是false,0的size必须是0
22. **void** trim() **const**
23. {
24. **while** (val.size() && val.back() == 0)
25. val.pop\_back();
26. **if** (val.empty())
27. nega = **false**;
28. }
29. **int** size() **const** { **return** val.size(); }
30. Long &operator[](**int** index) **const** { **return** val[index]; }
31. Long &back() **const** { **return** val.back(); }
32. BigInt(**int** size, **bool** nega) : val(size), nega(nega) {}
33. BigInt(**const** std::vector<Long> &val, **bool** nega) : val(val), nega(nega) {}
35. **public**:
36. **friend** std::ostream &operator<<(std::ostream &os, **const** BigInt &n)
37. {
38. **if** (n.size())
39. {
40. **if** (n.nega)
41. putchar('-');
42. **for** (**int** i = n.size() - 1; i >= 0; --i)
43. {
44. **if** (i == n.size() - 1)
45. printf("%lld", n[i]);
46. **else**
47. printf("%0\*lld", n.Exp, n[i]);
48. }
49. }
50. **else**
51. putchar('0');
52. **return** os;
53. }
54. **friend** BigInt operator+(**const** BigInt &lhs, **const** BigInt &rhs)
55. {
56. BigInt ret(lhs);
57. **return** ret += rhs;
58. }
59. **friend** BigInt operator-(**const** BigInt &lhs, **const** BigInt &rhs)
60. {
61. BigInt ret(lhs);
62. **return** ret -= rhs;
63. }
64. BigInt(Long x = 0)
65. {
66. **if** (x < 0)
67. x = -x, nega = **true**;
68. **while** (x >= Mod)
69. val.push\_back(x % Mod), x /= Mod;
70. **if** (x)
71. val.push\_back(x);
72. }
73. BigInt(**const** **char** \*s)
74. {
75. **int** bound = 0, pos;
76. **if** (s[0] == '-')
77. nega = **true**, bound = 1;
78. Long cur = 0, pow = 1;
79. **for** (pos = strlen(s) - 1; pos >= Exp + bound - 1; pos -= Exp, val.push\_back(cur), cur = 0, pow = 1)
80. **for** (**int** i = pos; i > pos - Exp; --i)
81. cur += (s[i] - '0') \* pow, pow \*= 10;
82. **for** (cur = 0, pow = 1; pos >= bound; --pos)
83. cur += (s[pos] - '0') \* pow, pow \*= 10;
84. **if** (cur)
85. val.push\_back(cur);
86. }
87. BigInt &operator+=(**const** BigInt &rhs)
88. {
89. **const** **int** cap = std::max(size(), rhs.size()) + 1;
90. val.resize(cap);
91. **int** carry = 0;
92. **for** (**int** i = 0; i < cap - 1; ++i)
93. {
94. val[i] = Value(val[i], nega) + Value(At(rhs, i), rhs.nega) + carry, carry = 0;
95. **if** (val[i] >= Mod)
96. val[i] -= Mod, carry = 1; //至多只需要减一次，不需要取模
97. **else** **if** (val[i] < 0)
98. val[i] += Mod, carry = -1; //同理
99. }
100. **if** ((val.back() = carry) == -1) //assert(val.back() == 1 or 0 or -1)
101. {
102. nega = **true**, val.pop\_back();
103. **bool** tailZero = **true**;
104. **for** (**int** i = 0; i < cap - 1; ++i)
105. {
106. **if** (tailZero && val[i])
107. val[i] = Mod - val[i], tailZero = **false**;
108. **else**
109. val[i] = Mod - 1 - val[i];
110. }
111. }
112. trim();
113. **return** \***this**;
114. }
115. **friend** BigInt operator-(**const** BigInt &rhs)
116. {
117. BigInt ret(rhs);
118. ret.nega ^= 1;
119. **return** ret;
120. }
121. BigInt &operator-=(**const** BigInt &rhs)
122. {
123. rhs.nega ^= 1;
124. \***this** += rhs;
125. rhs.nega ^= 1;
126. **return** \***this**;
127. }
128. //高精\*高精没办法原地操作，所以实现operator\*
129. //高精\*低精可以原地操作，所以operator\*=
130. **friend** BigInt operator\*(**const** BigInt &lhs, **const** BigInt &rhs)
131. {
132. **const** **int** cap = lhs.size() + rhs.size() + 1;
133. BigInt ret(cap, lhs.nega ^ rhs.nega);
134. //j < l.size(),i - j < rhs.size() => i - rhs.size() + 1 <= j
135. **for** (**int** i = 0; i < cap - 1; ++i) // assert(0 <= ret[cap-1] < Mod)
136. **for** (**int** j = std::max(i - rhs.size() + 1, 0), up = std::min(i + 1, lhs.size()); j < up; ++j)
137. {
138. ret[i] += lhs[j] \* rhs[i - j];
139. ret[i + 1] += ret[i] / Mod, ret[i] %= Mod;
140. }
141. ret.trim();
142. **return** ret;
143. }
144. BigInt &operator\*=(**const** BigInt &rhs) { **return** \***this** = \***this** \* rhs; }
145. **friend** BigInt operator/(**const** BigInt &lhs, **const** BigInt &rhs)
146. {
147. **static** std::vector<BigInt> powTwo{BigInt(1)};
148. **static** std::vector<BigInt> estimate;
149. estimate.clear();
150. **if** (absComp(lhs, rhs) < 0)
151. **return** BigInt();
152. BigInt cur = rhs;
153. **int** cmp;
154. **while** ((cmp = absComp(cur, lhs)) <= 0)
155. {
156. estimate.push\_back(cur), cur += cur;
157. **if** (estimate.size() >= powTwo.size())
158. powTwo.push\_back(powTwo.back() + powTwo.back());
159. }
160. **if** (cmp == 0)
161. **return** BigInt(powTwo.back().val, lhs.nega ^ rhs.nega);
162. BigInt ret = powTwo[estimate.size() - 1];
163. cur = estimate[estimate.size() - 1];
164. **for** (**int** i = estimate.size() - 1; i >= 0 && cmp != 0; --i)
165. **if** ((cmp = absComp(cur + estimate[i], lhs)) <= 0)
166. cur += estimate[i], ret += powTwo[i];
167. ret.nega = lhs.nega ^ rhs.nega;
168. **return** ret;
169. }
170. **bool** operator==(**const** BigInt &rhs) **const**
171. {
172. **return** nega == rhs.nega && val == rhs.val;
173. }
174. **bool** operator!=(**const** BigInt &rhs) **const** { **return** nega != rhs.nega || val != rhs.val; }
175. **bool** operator>=(**const** BigInt &rhs) **const** { **return** !(\***this** < rhs); }
176. **bool** operator>(**const** BigInt &rhs) **const** { **return** !(\***this** <= rhs); }
177. **bool** operator<=(**const** BigInt &rhs) **const**
178. {
179. **if** (nega && !rhs.nega)
180. **return** **true**;
181. **if** (!nega && rhs.nega)
182. **return** **false**;
183. **int** cmp = absComp(\***this**, rhs);
184. **return** nega ? cmp >= 0 : cmp <= 0;
185. }
186. **bool** operator<(**const** BigInt &rhs) **const**
187. {
188. **if** (nega && !rhs.nega)
189. **return** **true**;
190. **if** (!nega && rhs.nega)
191. **return** **false**;
192. **return** (absComp(\***this**, rhs) < 0) ^ nega;
193. }
194. **void** swap(**const** BigInt &rhs) **const**
195. {
196. std::swap(val, rhs.val);
197. std::swap(nega, rhs.nega);
198. }
199. };
200. **const** **int** N = 1e4 + 10;
201. **char** a[N], b[N];
202. **int** main()
203. {
204. scanf("%s%s", a, b);
205. BigInt ba(a), bb(b);
206. std::cout << ba + bb << '\n';
207. std::cout << ba - bb << '\n';
208. std::cout << ba \* bb << '\n';
209. BigInt d;
210. std::cout << (d = ba / bb) << '\n';
211. std::cout << ba - d \* bb << '\n';
212. **return** 0;
213. }

# ３.图论

## 3.1 最短路

ＤＩＪ

1. #include<iostream>
2. #include<bits/stdc++.h>
3. #include<queue>
4. #define ll long long
5. **using** **namespace** std;
6. **const** **int** maxn=1e6+2;
7. priority\_queue<pair<ll,**int**> >que;
8. ll w[maxn];
9. **int** to[maxn];
10. **int** head[maxn];
11. **int** vis[maxn];
12. ll dit[maxn];
13. **int** nxt[maxn];
14. //int que[maxn];
15. **int** fr,ed;
16. **int** tot=0;
17. **int** n,m,s;
18. **int** addedge(**int** f,**int** t,**int** we)
19. {
20. tot++;
21. to[tot]=t;
22. w[tot]=we;
23. nxt[tot]=head[f];
24. head[f]=tot;
25. **return** 0;
26. }
27. **void** bfs(**int** p)
28. {
29. vis[p]=1;
30. **while**(1)
31. {
32. vis[p]=1;
33. //cout<<"220"<<endl;
34. //break;
35. **for**(**int** i=head[p];i!=0;i=nxt[i])
36. {
37. **if**(vis[to[i]]!=1&&dit[to[i]]>dit[p]+w[i])
38. {
40. dit[to[i]]=dit[p]+w[i];
41. que.push(make\_pair(-dit[to[i]],to[i]));
42. }
43. }
44. **if**(que.empty())**break**;
45. **while**(1){
46. p=que.top().second;
47. que.pop();
48. **if**(que.empty()||vis[p]!=1)
49. **break**;
50. }
52. }
54. }
55. **int** main()
56. {
57. cin>>n>>m>>s;
58. **for**(**int** i=1;i<=n;i++)
59. dit[i]=1e10;
60. dit[s]=0;
61. **for**(**int** i=1;i<=m;i++)
62. {
63. **int** in1,in2,in3;
64. cin>>in1>>in2>>in3;
65. addedge(in1,in2,in3);
66. }
67. //que.push\_back(s);
68. bfs(s);
70. **for**(**int** i=1;i<=n;i++)
71. {
72. cout<<dit[i]<<" ";
73. }
74. //system("pause");
75. }

## ３.2 图的连通

**３.２.1 强连通分量**

如果有向图中任意两个点可以互相到达，则称为强连通。强连通分量是指有向图中可互相连通的最大导出子图。这些子图可以分成多个，通常我们将一个强连通分量缩成一个点进行操作，叫做缩点。

**kosaraju**



Kosaraju法是通常用来求解强连通分量的一个简单易于理解的方法，先通过反图跑dfs，然后再根据反图跑dfs的dfs出栈时间序列，从大到小的顺序，再在正图上进行dfs，每个点每次(这里指的是第二遍dfs)dfs能到达的点集合构成一个强连通分量。

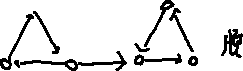


图 5.4.1 -1

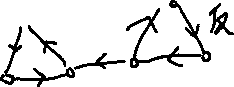


图5.4.1 -2

如图可知，在返图中跑dfs可以保证一个dfs时间序列，保证第二遍优先dfs到达的点是一个强连通分量的。

1. #include <bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **inline** ll rd(){
5. ll x=0;**char** o,f=1;
6. **while**(o=getchar(),o<48)**if**(o==45)f=-f;
7. **do** x=(x<<3)+(x<<1)+(o^48);
8. **while**(o=getchar(),o>47);
9. **return** x\*f;
10. }
11. **const** **int** maxn=1e5+5;
12. **const** ll mod=998244353;
13. **int** head[2][maxn],tot[2];
14. **struct** nn{**int** v,nxt;}g[2][maxn<<1];
15. **void** add\_edge(**int** u,**int** v,**int** p){g[p][++tot[p]]={v,head[p][u]};head[p][u]=tot[p];}
16. **int** \_,n,m;
17. **int** num,q[maxn],vis[maxn],f[maxn],cnt[maxn];
18. **void** dfs1(**int** x){
19. vis[x]=1;
20. **for** (**int** i=head[0][x];i;i=g[0][i].nxt){
21. **int** v=g[0][i].v;
22. **if** (!vis[v])dfs1(v);
23. }
24. q[++num]=x;
25. }
26. **void** dfs2(**int** x,**int** y){
27. vis[x]=0;f[x]=y;
28. **for** (**int** i=head[1][x];i;i=g[1][i].nxt){
29. **int** v=g[1][i].v;
30. **if** (vis[v])dfs2(v,y);
31. }
32. }
33. **int** main() {
34. **for**(\_=rd();\_;\_--){
35. n=rd();m=rd();
36. **for** (**int** i=1;i<=n;i++)cnt[i]=0;
37. **for** (**int** i=1;i<=n;i++)head[0][i]=head[1][i]=0;tot[0]=tot[1]=0;
38. **for** (**int** i=1,u,v;i<=m;i++){
39. u=rd();v=rd();
40. add\_edge(u,v,0);
41. add\_edge(v,u,1);
42. }
43. num=0;
44. **for** (**int** i=1;i<=n;i++)**if** (!vis[i])dfs1(i);
45. **for** (**int** i=n;i;i--)**if** (vis[q[i]])dfs2(q[i],q[i]);
46. **for** (**int** i=1;i<=n;i++)cnt[f[i]]++;
47. // ll ans=0;
48. // for (int i=1;i<=n;i++){  //根据题目要求输出答案
49. //    if (cnt[i]>=2)ans+=1ll\*cnt[i]\*(cnt[i]-1)/2;
50. //}
51. //printf("%lld\n",ans);
52. }
53. **return** 0;
54. }

**tarjan**

Tarjan算法同样用于连通性问题求解，但是tarjan算法的适用范围更为广泛，操作难度较5.4.1.1的算法更低。Tarjan算法依然是基于dfs的算法，从每个点开始dfs，并且把所有的到达的点都并入一个栈空间，当dfs的前向边指向一个先于当前节点被遍历到的点，并且这个点在栈空间中，则进行回溯操作，直到回到点并对之后的点出栈，这些点可以构成一个强连通分量。

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** M = 1e5 + 1;
4. vector<**int**>g[M];
5. **int** n, m;
6. stack<**int**>st;
7. **int** dfn[M], eku[M], pos[M];
8. **int** tot, indx;
9. **int** du[M];
10. **int** sz[M], vis[M];
11. **void** tarjan(**int** u)
12. {
13. dfn[u] = eku[u] = ++tot;
14. st.push(u);
15. vis[u] = 1;
16. **for** (auto i : g[u])
17. {
18. **if** (!dfn[i])
19. {
20. tarjan(i);
21. eku[u] = min(eku[u], eku[i]);
22. }
23. **else** **if** (vis[i])
24. {
25. eku[u] = min(eku[u], dfn[i]);
26. }
27. }
28. **if** (eku[u] == dfn[u])
29. {
30. indx++;
31. **while** (st.top() != u)
32. {
33. sz[indx]++;
34. pos[st.top()] = indx;
35. st.pop();
36. }
37. sz[indx]++;
38. pos[st.top()] = indx;
39. st.pop();
40. }
41. }
42. **int** main()
43. {
44. cin >> n >> m;
45. **for** (**int** i = 1; i <= m; i++)
46. {
47. **int** u, v; scanf("%d%d", &u, &v);
48. g[u].push\_back(v);
49. }
50. **for** (**int** i = 1; i <= n; i++)
51. {
52. **if** (!dfn[i])tarjan(i);
53. }
54. //for (int i = 1; i <= n; i++)
55. //{
56. //for (auto t : g[i])
57. //{
58. //if (pos[t] != pos[i])
59. //{
60. //  du[pos[i]]++;
61. //}
62. //}
63. //}
64. //int ans = 0;
65. //for (int i = 1; i <= indx; i++)
66. //{
67. //if (du[i] == 0&&ans==0)
68. //{
69. //  ans = sz[i];
70. //}
71. //else if (du[i] == 0 && ans > 0)
72. //{
73. //  ans = -1;
74. //}
75. //}
76. //if (ans == -1)cout << 0 << '\n';
77. //else cout << ans << '\n';
78. }

## ３.３　欧拉回路

输入文件由几个块组成。 每个街区描述一个城镇。 块中的每一行包含三个整数x; Ÿ; z，其中x> 0且y> 0是由街道号z连接的交叉点的数量。 块的末尾由包含x = y = 0的行标记。在输入文件的末尾有一个空块，x = y = 0。

产量

每个块的输出一行包含街道编号序列（序列的单个成员由空格分隔），描述Johnny的往返行程。 如果找不到往返，则相应的输出块包含消息“往返不存在”。

1. #include<iostream>
2. #include<cstring>
3. #include<cstdio>
4. #define mst(a) memset(a,0,sizeof(a))
5. **using** **namespace** std;
6. **const** **int** maxn=20000+8;
7. **int** angleNum[maxn]; //表示顶点度数
8. **int** used[maxn];    //判断顶点是否走过
9. **int** n,k;
10. **int** res[maxn];      //路径数组
11. **int** cnt=0;
12. **struct** node{        //结构体定义 起点和终点
13. **int** a;
14. **int** b;
15. }rng[maxn];
16. **bool** is\_OK()   //此方法用于判定每个顶点的度数
17. {
18. **for**(**int** i=1;i<=n;i++)
19. **if**(angleNum[i]%2)   //如果存在一个顶点度数为偶数度，那么就不存在欧拉回路
20. **return** **false**;    //仅仅对于有向图而言
21. **return** **true**;
22. }
23. **void** dfs(**int** x)
24. {
25. **for**(**int** i=1;i<=k;i++)
26. {
27. **if**(!used[i]&&(rng[i].a==x||rng[i].b==x))   //这里处理有点特殊，我们不确定
28. {                                          //走的是每个分支的起点还是终点
29. used[i]=1;
30. dfs(rng[i].a+rng[i].b-x);
31. res[++cnt]=i;
32. }
33. }
34. }
35. **int** main()
36. {
37. ios::sync\_with\_stdio(**false**);   //cin提速
38. cin.tie(0);
39. **int** x,y,z;
40. **while**(cin>>x>>y&&(x+y))
41. {
42. **int** point=min(x,y);         //求出我们的源点
43. n=max(x,y);
44. mst(angleNum);
45. mst(used);
46. cnt=k=0;
47. **do**{
48. cin>>z;
49. rng[z].a=x;
50. rng[z].b=y;
51. ++k;
52. angleNum[x]++; angleNum[y]++;
53. n=max(n,max(x,y));  //求出最大的顶点数
54. }**while**(cin>>x>>y&&(x+y));
55. **if**(!is\_OK())
56. {
57. cout<<"Round trip does not exist."<<endl;
58. **continue**;
59. }
60. dfs(point);    //dfs深搜欧拉回路
61. **for**(**int** i=cnt;i>=1;i--)     //逆序输出路径
62. {
63. **if**(i!=1)
64. cout<<res[i]<<" ";
65. **else**
66. cout<<res[i]<<endl;
67. }
68. }
69. **return** 0;
70. }

## ３.４TwoSAT

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** M = 3e6 + 1;
4. vector<**int**>g[M];
5. **int** n, m;
6. **int** top[M], tot;
7. **int** indx, f[M], vis[M];
8. stack<**int**>st;
9. **int** dfu[M], low[M];
10. **void** tarjan(**int** nw)
11. {
12. dfu[nw] = low[nw] = ++indx;
13. st.push(nw);
14. vis[nw] = 1;
15. **for** (auto v : g[nw])
16. {
17. **if** (!dfu[v])
18. {
19. tarjan(v);
20. low[nw] = min(low[v],low[nw]);
21. }
22. **else** **if** (vis[v] == 1)
23. {
24. low[nw] = min(low[nw], dfu[v]);
25. }
26. }
27. **if** (low[nw] == dfu[nw])
28. {
29. tot++;
30. **while** (st.top() != nw)
31. {
32. f[st.top()] = tot;
33. vis[st.top()]=0;
34. st.pop();
35. }
36. vis[nw]=0;
37. f[nw] = tot;
38. st.pop();
39. }
40. }
41. **int** main()
42. {
43. //    freopen("E:\\Edata\\6.in","r",stdin);
44. //    freopen("E:\\Edata\\6.out","w",stdout);
45. cin >> n >> m;
46. **for** (**int** i = 1; i <= m; i++)  //2-SAT核心代码，建图。
47. {
48. **int** x, bx, y, by;
49. scanf("%d%d%d%d", &x, &y, &bx, &by);
50. g[x + (bx ^ 1) \* n].push\_back(y + by \* n);
51. g[y + (by ^ 1) \* n].push\_back(x + bx \* n);
52. }
53. **for** (**int** i = 1; i <= 2\*n; i++)
54. {
55. **if** (!dfu[i])tarjan(i);
56. }
57. **int** ans = 1;
58. **for** (**int** i = 1; i <= n; i++) {
59. **if** (f[i] == f[i + n])
60. {
61. ans = 0;
62. **break**;
63. }
64. }
65. **if** (ans)
66. {
67. printf("Yes");
68. //      for (int i = 1; i <= n; i++)
69. //      {
70. //          cout << (f[i] > f[i + n]) << " ";
71. //      }
72. //      printf("\n");
73. }
74. **else**
75. {
76. printf("No");
77. }
78. }

## ３.５网络流

**3.5.1 基于贪心的最大流求法（ek）**

最大流问题指的是给定一张有向图,其中的所有边给定一个容量，以及一个源点和一个汇点。规定从源点到汇点流流量，流量经过流经的每一条边和点，其中流过每条边的流量不能超过容量，求源点到汇点的最大流量。

解决这种问题，我们每次找一条可行流，并且不断增广，直到流量无法增广，此时找到的便是最大流，其中给定的最小割等于最大流。注意推流操作（即每有流量流过一条边，反向边的容量就增加正向边流经的流量）。

,.

1. **while**(spfa(s,t))  //Ek算法
2. {
3. minn=INF;
4. **for**(**int** i=pre[t];i!=-1;i=pre[to[i^1]])
5. {
6. **if**(minn>flow[i])
7. {
8. minn=flow[i];
9. }
10. }
11. cost+=minn;
12. **for**(**int** i=pre[t];i!=-1;i=pre[to[i^1]])
13. {
14. flow[i]-=minn;
15. flow[i^1]+=minn;//退流操作
16. }
17. }

**3.5.2 Dinic**

最大流通过5.10.1 中的ek算法求解显然时空效率低下，每次增广最多可以找到一条可行流，通过Dinic算法，一次通常可以找到多条可行流，从而增加效率。

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define ll long long  **()**
4. #define int long long
5. **const** **int** M=1001;
6. **int** head[2\*M],to[2\*M],nxt[2\*M],tot,w[2\*M],d[M];
7. **int** n,m;
8. queue<**int**>qu;
9. ll min(ll x,ll y)
10. {
11. **return** x<y?x:y;
12. }
13. **inline** **void** Add(**int** s,**int** t,**int** f)
14. {
15. to[tot]=t;
16. w[tot]=f;
17. nxt[tot]=head[s];
18. head[s]=tot;
19. tot++;
20. }
21. **int** dinic()
22. {
23. **while**(!qu.empty())qu.pop();
24. memset(d,0,**sizeof** d);
25. d[1]=1;
26. qu.push(1);
27. **while**(!qu.empty())
28. {
29. **int** nw=qu.front();
30. qu.pop();
31. **int** tt;
32. **for**(**int** i=head[nw];i!=-1;i=nxt[i])
33. {
34. tt=to[i];
35. **if**(w[i]>0&&d[tt]==0)
36. {
37. d[tt]=d[nw]+1;
38. qu.push(tt);
39. **if**(tt==n)**return** 1;
40. }
42. }
43. }
44. **return** 0;
45. }
46. ll dfs(**int** nw,**int** v,ll flow)
47. {
48. ll sum=0;**int** tt;
49. **if**(nw==v)**return** flow;
50. **for**(**int** i=head[nw];i!=-1;i=nxt[i])
51. {
52. tt=to[i];
53. **if**(d[tt]==d[nw]+1&&w[i]>0)
54. {
55. **int** tmp=dfs(tt,v,min(flow-sum,w[i]));
56. sum+=tmp;
57. w[i^1]+=tmp;w[i]-=tmp;
58. **if**(sum==flow)**return** sum;
59. }
60. }
61. **return** sum;
62. }
63. **signed** main()
64. {
65. **int** t,u,v,fl;
66. cin>>t;
67. **while**(t--)
68. {
69. tot=0;memset(head,-1,**sizeof** head);
70. scanf("%lld%lld",&n,&m);
72. **for**(**int** i=1;i<=m;i++)
73. {
74. scanf("%lld%lld%lld",&u,&v,&fl);
75. Add(u,v,fl);Add(v,u,0);
76. }
77. ll ans=0;
78. **while**(dinic())
79. {
80. ans+=dfs(1,n,99999999999);
81. }
82. printf("%lld\n",ans);
83. }
84. }

**3.5.3 ISAP**

还有一种处理这种题目的方法，当我们的Dinic被特殊数据卡时，我们还可以用ISAP法处理这类问题，由于网络流的题目难点在于建图，下面只给出代码。

1. **const** **int** N=100;//点数，从1开始编号
2. **const** **int** M=100000;//边数
3. **const** **int** inf=1000000000;//无穷大
4. **struct** E{**int** t,f;E\*nxt,\*pair;}\*g[N],\*d[N],pool[M\*2],\*cur=pool;
5. **int** n,m,i,S,T,h[N],gap[N],maxflow;
6. **void** add(**int** s,**int** t,**int** f){//添加s->t的一条边，容量为f
7. E\*p=cur++;p->t=t;p->f=f;p->nxt=g[s];g[s]=p;
8. p=cur++;p->t=s;p->f=0;p->nxt=g[t];g[t]=p;
9. g[s]->pair=g[t];g[t]->pair=g[s];
10. }
11. **int** sap(**int** v,**int** flow){
12. **if**(v==T)**return** flow;
13. **int** rec=0;
14. **for**(E\*p=d[v];p;p=p->nxt)**if**(h[v]==h[p->t]+1&&p->f){
15. **int** ret=sap(p->t,min(flow-rec,p->f));
16. p->f-=ret;p->pair->f+=ret;d[v]=p;
17. **if**((rec+=ret)==flow)**return** flow;
18. }
19. **if**(!(--gap[h[v]]))h[S]=T;
20. gap[++h[v]]++;d[v]=g[v];
21. **return** rec;
22. }
23. **int** main(){
24. S=n+1;//源点
25. T=S+1;//汇点
26. **for**(cur=pool,i=1;i<=T;i++)g[i]=d[i]=NULL,h[i]=gap[i]=0;//初始化
28. //请在这里连边
30. **for**(gap[maxflow=0]=T,i=1;i<=T;i++)d[i]=g[i];
31. **while**(h[S]<T)maxflow+=sap(S,inf);
32. printf("maxflow=%d\n",maxflow);
33. }

**3.5.4 费用流**

1. 最小费用
2. #define INF 0x7fffffff
3. **const** **int** M=1e5+4;
4. ll w[M];
5. **int** to[M];
6. **int** tot;
7. **int** head[M];
8. **int** flow[M];
9. **int** nxt[M];
10. ll d[M];
11. **bool** vis[M];
12. **int** minn;
13. **int** pre[M];
14. **int** n,m,s,t;
15. queue<**int**>qu;
16. **void** Add(**int** s,**int** t,**int** wl,**int** cst)
17. {
18. to[tot]=t;
19. w[tot]=cst;
20. flow[tot]=wl;
21. nxt[tot]=head[s];
22. head[s]=tot;
23. tot++;
24. }
25. **int** spfa(**int** s,**int** t){
26. **int** nw;
27. **while**(!qu.empty())qu.pop();
28. **for**(**int** i=1;i<=n;i++)
29. {
30. vis[i]=**false**;
31. d[i]=INF;
32. }
33. qu.push(s);
34. d[s]=0;
35. vis[s]=**true**;
36. **while**(!qu.empty())
37. {
38. nw=qu.front();
39. qu.pop();
40. vis[nw]=**false**;
41. pre[s]=-1;
42. **for**(**int** i=head[nw];i!=-1;i=nxt[i])
43. {
44. **int** tt=to[i];
45. **if**(d[tt]>d[nw]+w[i]&&flow[i]>0)
46. {
47. d[tt]=d[nw]+w[i];
48. pre[tt]=i;
49. **if**(vis[tt]==**false**)
50. {
51. vis[tt]=**true**;
52. qu.push(tt);
53. }
54. }
55. }
56. }
57. **if**(d[t]!=INF) **return** 1;
58. **else** **return** 0;
59. }
60. **int** main()
61. {
62. **int** in1,in2,in3,in4;
63. ll ans,cost;
64. cin>>n>>m>>s>>t;
65. memset(head,-1,**sizeof** head);
66. **for**(**int** i=1;i<=m;i++)
67. {
68. scanf("%d%d%d%d",&in1,&in2,&in3,&in4);
69. Add(in1,in2,in3,in4);
70. Add(in2,in1,0,-in4);
71. }
72. ans=0;
73. cost=0;
74. **while**(spfa(s,t))
75. {
76. minn=INF;
77. **for**(**int** i=pre[t];i!=-1;i=pre[to[i^1]])
78. {
79. **if**(minn>flow[i])
80. {
81. minn=flow[i];
82. }
83. }
84. cost+=(minn\*d[t]);
85. ans+=minn;
86. **for**(**int** i=pre[t];i!=-1;i=pre[to[i^1]])
87. {
88. flow[i]-=minn;
89. flow[i^1]+=minn;
90. }
91. }
92. cout<<ans<<" "<<cost<<endl;
93. }

**\*\*最大费用即把最短路算法换成最长路\*\***

# 4．字符串算法

## ４.１ＫＭＰ

1. #include <iostream>
2. #include <cstring>
4. **using** **namespace** std;
6. **const** **int** N = 1000100;
8. **int** n, m;
9. // n 主串长度， m模式串长度
10. **char** a[N], b[N];
11. // a 主串， b 模式串, 下标从零开始
12. **int** nextt[N];
14. **void** get\_nextt()
15. {
16. **int** i = 0, j;
17. j = nextt[0] = -1;
18. **while**(i < m)
19. {
20. **if**(-1 == j || b[i] == b[j] )
21. nextt[++i] = ++j;
22. **else**
23. j = nextt[j];
24. }
25. }
27. **void** get\_nextval() {
28. **int** i = 0, j;
29. j = nextt[0] = -1;
30. **while**(i < m) {
31. **if**(-1 == j || b[i] == b[j]) {
32. ++i, ++j;
33. nextt[i] = (b[i] != b[j]) ? j : nextt[j];
34. }
35. **else** {
36. j = nextt[j];
37. }
38. }
39. }
41. **int** kmp\_index(**int** pos) {
42. // 求模式串在主串 pos位置 字符之后的位置
43. **int** i = pos, j = 0;
44. **while**(i < n && j < m) {
45. **if**(-1 == j || a[i] == b[j]) {
46. ++i, ++j;
47. }
48. **else** {
49. j = nextt[j];
50. }
51. }
52. **if**(j >= m) **return** i-m;
53. **else** **return** -1;
54. }
56. **int** kmp\_count(**int** pos) {
57. // 求模式串在主串中出现了几次
58. **int** i = pos, j = 0;
59. **int** ans = 0;
60. **while**(i < n) {
61. **if**(-1 == j || a[i] == b[j]) {
62. ++i, ++j;
63. **if**(j >= m) {
64. ++ ans;
65. j = nextt[j];
66. }
67. }
68. **else**
69. j = nextt[j];
70. }
71. **return** ans;
72. }
73. **int** main()
74. {
75. cin >> a >> b;
76. get\_nextt();
77. get\_nextval();
78. **int** Index = kmp\_index( 0 );
79. **int** Count = kmp\_count( 0 );
80. **return** 0;
81. }

## ４.２　前缀自动机

1. #include <bits/stdc++.h>
2. #include <cstdlib>
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **const** **int** maxn =  2\*1e6+9;
7. **int** trie[maxn][26]; //字典树
8. **int** cntword[maxn];  //记录该单词出现次数
9. **int** fail[maxn];     //失败时的回溯指针
10. **int** cnt = 0;
12. **void** insertWords(string s){
13. **int** root = 0;
14. **for**(**int** i=0;i<s.size();i++){
15. **int** next = s[i] - 'a';
16. **if**(!trie[root][next])
17. trie[root][next] = ++cnt;
18. root = trie[root][next];
19. }
20. cntword[root]++;      //当前节点单词数+1
21. }
22. **void** getFail(){
23. queue <**int**>q;
24. **for**(**int** i=0;i<26;i++){      //将第二层所有出现了的字母扔进队列
25. **if**(trie[0][i]){
26. fail[trie[0][i]] = 0;
27. q.push(trie[0][i]);
28. }
29. }
31. //fail[now]    ->当前节点now的失败指针指向的地方
32. tire[now][i] -> 下一个字母为i+'a'的节点的下标为tire[now][i]
33. **while**(!q.empty()){
34. **int** now = q.front();
35. q.pop();
37. **for**(**int** i=0;i<26;i++){      //查询26个字母
38. **if**(trie[now][i]){
39. //如果有这个子节点为字母i+'a',则
40. //让这个节点的失败指针指向(((他父亲节点)的失败指针所指向的那个节点)的下一个节点)
41. //有点绕,为了方便理解特意加了括号
43. fail[trie[now][i]] = trie[fail[now]][i];
44. q.push(trie[now][i]);
45. }
46. **else**//否则就让当前节点的这个子节点
47. //指向当前节点fail指针的这个子节点
48. trie[now][i] = trie[fail[now]][i];
49. }
50. }
51. }

54. **int** query(string s){
55. **int** now = 0,ans = 0;
56. **for**(**int** i=0;i<s.size();i++){    //遍历文本串
57. now = trie[now][s[i]-'a'];  //从s[i]点开始寻找
58. **for**(**int** j=now;j && cntword[j]!=-1;j=fail[j]){
59. //一直向下寻找,直到匹配失败(失败指针指向根或者当前节点已找过).
60. ans += cntword[j];
61. cntword[j] = -1;    //将遍历国后的节点标记,防止重复计算
62. }
63. }
64. **return** ans;
65. }
67. **int** main() {
68. **int** n;
69. string s;
70. cin >> n;
71. **for**(**int** i=0;i<n;i++){
72. cin >> s ;
73. insertWords(s);
74. }
75. fail[0] = 0;
76. getFail();
77. cin >> s ;
78. cout << query(s) << endl;
79. **return** 0;
80. }

## ４.３　马拉车

1. #include<iostream>
2. #include<string.h>
3. #include<algorithm>
4. **using** **namespace** std;
5. **char** s[1000];
6. **char** s\_new[2000];
7. **int** p[2000];
8. **int** Init()
9. {
10. **int** len = strlen(s);
11. s\_new[0] = '$';
12. s\_new[1] = '#';
13. **int** j = 2;
14. **for** (**int** i = 0; i < len; i++)
15. {
16. s\_new[j++] = s[i];
17. s\_new[j++] = '#';
18. }
19. s\_new[j] = '\0';  //别忘了哦
20. **return** j;  //返回s\_new的长度
21. }
22. **int** Manacher()
23. {
24. **int** len = Init();  //取得新字符串长度并完成向s\_new的转换
25. **int** maxLen = -1;   //最长回文长度
26. **int** id;
27. **int** mx = 0;
28. **for** (**int** i = 1; i < len; i++)
29. {
30. **if** (i < mx)
31. p[i] = min(p[2 \* id - i], mx - i);  //需搞清楚上面那张图含义, mx和2\*id-i的含义
32. **else**
33. p[i] = 1;
34. **while** (s\_new[i - p[i]] == s\_new[i + p[i]])  //不需边界判断，因为左有'$',右有'\0'
35. p[i]++;
36. **if** (mx < i + p[i])  //我们每走一步i，都要和mx比较，我们希望mx尽可能的远，这样才能更有机会执行if (i < mx)这句代码，从而提高效率
37. {
38. id = i;
39. mx = i + p[i];
40. }
41. maxLen = max(maxLen, p[i] - 1);
42. }
43. **return** maxLen;
44. }
45. **int** main()
46. {
47. **while** (printf("请输入字符串：\n"))
48. {
49. scanf("%s", s);
50. printf("最长回文长度为 %d\n\n", Manacher());
51. }
52. **return** 0;
53. }

# ５．动态规划

动态规划，是一种求一个多阶段决策问题的最优解的过程（有时也用来统计方案数）的思想方法，原理是利用分治的思想，将一个问题分解为多个子问题来解决。

入门例题：数塔问题（洛谷1216数字三角形）这里暂不给出题解。

**动态规划有以下特性：**

1. 无后效性，在一个阶段做出的决策不应对后续的决策有影响。
2. 最优子结构，即在每个子结构中选择最优决策，使得全局决策成为最优解。
3. 重叠子结构，能应用动态规划解决的问题应该能分为若干相同的子结构，而每个最优的子结构应当能被多次使用（否则动态规划不能比暴力搜索有更高的效率）。

## 5.0.1动态规划解题的一般步骤

1. **确定无后效性的子结构，以及确定子结构需要传递的状态和状态变量**。即划分的子结构表示的意义以及在什么状态（可传递的状态）下实现的。一般问题问什么，子结构就要表示什么。如在背包问题中就是表示背包当前放入的物品价值。。一般需要看题目，知道与问题相关 变量是需要传递的状态。如背包问题中就是在放入第i件物品，已经用了j容量背包的状态下放入背包的价值。需要传递的状态是背包的容量和放入的物品数。
2. **确定状态转移方程**，需要根据子结构之间的关系列出符合题目要求的状态转移方程，如果无法列出转移方程或者列出的方程较为复杂难以实现，则需要检查确定的子结构是否有问题，或者确定的状态是否有条件遗漏，传递的状态是否合理。如果都没有问题，则需要检查这道题是否满足动态规划的特性。
3. **初始化和确定边界条件**。初始化要根据问题需要进行，有些问题（一般为求最小值的问题）需要将dp数组初始化为最大值，有些问题则需要初始化为特定值。边界条件需要根据输入量来确定，或者根据数据规模来确定。
4. **将思路翻译成代码并进行调试**。有些题目需要思考优化（剪枝）条件。典型的题目见区间dp部分的icpc2021昆明站题。

## 5.0.1资料中的解题方法（摘自邓丝雨老师的《dp进阶之路》）

**1、模型匹配法：**

最先考虑的就是这个方法了。挖掘问题的本质，如果发现问题是自己熟悉的某个基本的模型，就直接套用，但要小心其中的一些小的变动，现在考题办都是基本模型的变形套用时要小心条件，三思而后行。

**2、三要素法**

仔细分析问题尝试着确定动态规划的三要素，不同问题的却定方向不同：

先确定阶段的问题：数塔问题

先确定状态的问题：大多数都先确定状态的。

先确定决策的问题：背包问题

一般都是先从比较明显的地方入手，至于怎么知道哪个明显就是经验问题了，多做题就会发现。

**3、寻找规律法：**

这个方法很简单，耐心推几组数据后，看他们的规律，总结规律间的共性，有点贪心的意思。

**4、边界条件法**

找到问题的边界条件，然后考虑边界条件与它的领接状态之间的关系。这个方法也很起效。

**5、放宽约束和增加约束**

具体内容就是给问题增加一些条件或删除一些条件使问题变的清晰。

## 5.1线性动态规划

线性动态规划，即子结构有线性关系的动态规划问题，在线性决策链中，筛选出最优解的过程，一般可以知道始末状态或者可能的始末状态，可以满足无后效性的状态转移关系（即不影响后续决策的状态转移关系）和可以选择出最优解的子结构的方法（状态转移）。

### 5.1.1典型例题

**5.1.1.1洛谷p2758编辑距离**

**题目描述**

设A和B是两个字符串。我们要用最少的字符操作次数，将字符串A转换为字符串B。这里所说的字符操作共有三种：

1、删除一个字符；

2、插入一个字符；

3、将一个字符改为另一个字符；

！皆为小写字母！

**输入格式**

第一行为字符串A；第二行为字符串B；字符串A和B的长度均小于2000。

**输出格式**

只有一个正整数，为最少字符操作次数。

**输入**

sfdqxbw

gfdgw

**输出**

4

**分析过程**：

**始末状态**：开始状态s1，结束状态s2

**子结构：**从s1->s2的过程中每部操作后的结果（状态）

**最优子结构：**达到每个状态的最少补数

因此，需要一个数组dp来储存最少步数。因为很容易看出每次操作后两字符串的匹配度一定会上升，所以我们希望经历一定次数的操作后s1的前i个元素变成了s2的前j个元素，dp[i][j]储存操作结束后的最小值。

**转移方法：**

枚举每一种操作，将可行的操作结束状态中选择最小值转移到当前状态。

1. 删除一个元素后，i->j的步数最少，即dp[i][j]=dp[i-1][j]+1(注：把第一个元素删除了，所以是从i-1,j的状态态传递过来，进行了这步操作后步数+1
2. 添加一个元素后,i->j的步数最少，即dp[i][j]=dp[i][j-1]+1(注：前s1的i个元素变成s2的前j-1个元素，在这个状态的字符串结尾添加s2字符的第j个元素，使得字符串变为s2的前j个元素。
3. 将s1的第i个元素变为s2第j个元素，这里有2种情况。如果s1第i个元素等于s2第j个元素，这里就不需要操作，直接将转移dp[i-1[j-1]的状态即可。如果不相等，则需要进行替换操作，所以需要步数+1.

因此，状态转移方程为：

**dp[i][j]=min(dp[i][j-1],dp[i-1][j],s1[i]==s[2]?dp[i-1][j-1]:dp[i-1][j-1]+1)**

**AC代码如下**

1. **int** dp[2000][2000];
2. **int** main() {
3. **char** a[10000];
4. **char** b[10000];
5. **int** s1, s2;
6. cin >> a >> b;
7. s1 = strlen(a);
8. s2 = strlen(b);
9. **for** (**int** i = 0; i <= s1; i++)
10. dp[i][0] = i;
11. **for** (**int** i = 1; i <= s2; i++)
12. dp[0][i] = i;
13. **for** (**int** i = 1; i <= s1; i++) {
14. **for** (**int** j = 1; j <= s2; j++) {
15. dp[i][j] = min(min(dp[i - 1][j], dp[i][j - 1]), dp[i - 1][j - 1]) + 1;
16. **if** (a[i - 1] == b[j - 1])dp[i][j] = min(dp[i][j], dp[i - 1][j - 1]);
17. }
18. }
19. cout << dp[s1][s2] << endl;
20. **return** 0;
21. }

**4.1.1.2 洛谷p4933大师**

**题目背景**

建筑大师最近在跟着数学大师 ljt12138 学数学，今天他学了等差数列，ljt12138 决定给他留一道练习题。

**题目描述**

ljt12138 首先建了 n 个特斯拉电磁塔，这些电塔排成一排，从左到右依次标号为 11 到 n ，第 i个电塔的高度为 h[i]。

建筑大师需要从中选出一些电塔，然后这些电塔就会缩到地下去。这时候，如果留在地上的电塔的高度，从左向右构成了一个等差数列，那么这个选择方案就会被认为是美观的。

建筑大师需要求出，一共有多少种美观的选择方案，答案模 998244353 。

注意，如果地上只留了一个或者两个电塔，那么这种方案也是美观的。地上没有电塔的方案被认为是不美观的。

同时也要注意，等差数列的公差也可以为负数。

**输入格式**

第一行一个正整数 n。

第二行 n个非负整数，第 i个整数是第 i个电塔的高度 h[i]。

**输出格式**

输出一个整数，表示美观的方案数模 998244353998244353 的值。

**思路：**首先读懂题意，知道是求已知序列中等差数列的个数。首先会想到枚举子序列的长度，只需要求得前i个元素的等差数列个数，i+1个元素的等差数列个数会和dp[i]有关系，但是实际上，由于不能知道i+1的元素与前面元素的关系，所以枚举的方向应该是以a[i]元素结尾的等差数列的个数。后来发现要枚举等差数列的公差，因此为了方便，额外开一维数组来存放枚举的公差结果，转移方程就是dp[i][j]=。但是实际上这个转移方程得出的结果是错的，因为枚举了公差，前置元素就已经确定，因此枚举的方向改为枚举第i个元素的前置元素，公差就等于a[i]-a[j]。因为由题意得公差有可能是负数，因此需要开双倍数组来储存负数的结果。转移方程为dp[i][a[i]-a[j]]+=dp[j][a[i]-a[j]]+1。这里说明一下加1的原因，原先应该设想i,k的结果应该是j,k的结果转移过来，假设一组等差数列1 2 3 4 5，以4为结尾的等差数列有3个，而以5结尾的的等差数列有4个，原因是等差数列元素加1，所以多了一种等差可能1 2 3 4 5 。

**代码如下：**

1. **const** **int** N = 998244353;
2. **const** **int** x = 2000;
3. **int** dp[1005][40005];
4. **int** a[1005];
5. **int** main() {
6. **int** n;
7. cin >> n;
8. **long** **long** ans = 0;
9. **for** (**int** i = 1; i <= n; i++) {
10. cin >> a[i];
11. }
12. **for** (**int** i = 2; i <= n; i++) {
13. **for** (**int** j = 1; j < i; j++) {
14. dp[i][a[i] - a[j] + x] = (dp[i][a[i] - a[j] + x] % N + dp[j][a[i] - a[j] + x] % N + 1) % N;
15. ans = (ans % N + dp[j][a[i] - a[j] + x] % N + 1) % N;
16. }
17. }
18. cout << (ans + n % N) % N << endl;
19. **return** 0;
20. }

**4.1.1.3 洛谷P1280 尼克的任务**

**题目描述**

尼克每天上班之前都连接上英特网，接收他的上司发来的邮件，这些邮件包含了尼克主管的部门当天要完成的全部任务，每个任务由一个开始时刻与一个持续时间构成。

尼克的一个工作日为 n*n* 分钟，从第 11 分钟开始到第 n*n* 分钟结束。当尼克到达单位后他就开始干活，公司一共有 k*k* 个任务需要完成。如果在同一时刻有多个任务需要完成，尼克可以任选其中的一个来做，而其余的则由他的同事完成，反之如果只有一个任务，则该任务必需由尼克去完成，假如某些任务开始时刻尼克正在工作，则这些任务也由尼克的同事完成。如果某任务于第 p*p* 分钟开始，持续时间为 t*t* 分钟，则该任务将在第 (p+t-1)(*p*+*t*−1) 分钟结束。

写一个程序计算尼克应该如何选取任务，才能获得最大的空暇时间。

**输入格式**

输入数据第一行含两个用空格隔开的整数 n*n* 和 k*k*。

接下来共有 k*k* 行，每一行有两个用空格隔开的整数 p*p* 和 t*t*，表示该任务从第 p*p* 分钟开始，持续时间为 t*t* 分钟。

**输出格式**

输出文件仅一行，包含一个整数，表示尼克可能获得的最大空暇时间。

**思路：**这道题初见知道肯定是dp，因为搜索显然超时。那么我们先确定最优子结构，由于每个任务可以选择做与不做，但是空闲的时候并且此时有任务的话必须要做任务，那么以当前做的任务作为最优子结构显然比较困难，那么我们先试想以当前时刻的空闲时间作为最优子结构。由于每个任务有持续时间，状态必须从间隔一个任务的时间传递过来，如果我们正向遍历，则需要访问改任务的结束时间，并从其开始时间传递过来。由于输入使用的是开始时间和持续时间，为了简单起见，我们反向遍历，使得只需要访问任务的开始时间并从结束时间传递过来。当此时没有任何开始的任务时，此时的空闲时间的最大值就是这个时刻加1的时刻的空闲时间加1.因此，我们列出转移方程为：

dp[i]=dp[i+1]+1;(当前无任务)

dp[i]=max{dp[i+t[j]]}(当前有任务，t[j]为第j个任务的持续时间)

到了这里，我们很容易想到要对任务进行以开始时间为依据进行排序，还需要记录每个时间段的任务数量。这里不做过多说明，直接给出代码。

**代码如下：**

1. **class** x {
2. **public**:
3. **int** p, t;
4. }a[10005];
6. **bool** cmp(x a, x b) {
7. //if (a.p == b.p)return a.t < b.t;
8. **return** a.p < b.p;
9. }
10. **int** dp[10005];
11. **int** sum[10005];
13. **int** main() {
14. **int** n, k;
15. **int** len;
16. cin >> n >> k;
17. len = k;
18. **for** (**int** i = 1; i <= k; i++) {
19. cin >> a[i].p >> a[i].t;
20. sum[a[i].p]++;
21. }
22. sort(a + 1, a + k + 1, cmp);
23. **for** (**int** i = n; i > 0; i--) {
24. **if** (!sum[i])dp[i] = dp[i + 1] + 1;
25. **else** **for** (**int** j = 1; j <= sum[i]; j++) {
26. dp[i] = max(dp[i + a[len--].t], dp[i]);
27. }
28. }
29. cout << dp[1] << endl;
30. **return** 0;
31. }

**5.1.1.4洛谷p5858 「SWTR-03」Golden Sword**

**题目描述**

制造一把金宝剑需要n种原料，编号为 11 到 n，编号为 i的原料的坚固值为ai。

炼金是很讲究放入原料的顺序的，因此小 E 必须按照 1 到 n 的顺序依次将这些原料放入炼金锅。

但是，炼金锅的容量非常有限，它最多只能容纳 w个原料。

所幸的是，每放入一个原料之前，小 E 可以从中取出一些原料，数量不能超过s个。

* 我们定义第i种原料的耐久度为：放入第i 种原料时锅内的原料总数（包括正在放入的原料） ai，则宝剑的耐久度为所有原料的耐久度之和。

小 E 当然想让他的宝剑的耐久度尽可能得大，这样他就可以带着它进行更多的战斗，请求出耐久度的最大值。

注：这里的“放入第 i种原料时锅内的原料总数包括正在放入锅中的原料,详细信息请见样例。

**输入格式**

第一行，三个整数n,w,s。

第二行，n 个整数 ai。

**输出格式**

一行一个整数，表示耐久度的最大值。

**思路**

这道题要求宝剑的耐久度的最大值，由题意得加入某件原料之后的耐久度只和加入原料前的上一个状态有关，由题意可知最多可以在加入这个物品前拿走s给物品，因此，我们令dp[i][j]为已经加了i给物品，锅里有j给物品时的耐久度，那么加入第物品的上一个状态可以为锅里有j-1个物品到j+s-1个物品（因为拿掉s给物品，锅里剩下j-1给物品，然后加入i物品，这时国里有j给物品）。那么显然得到转移方程为：

**dp[i][j]=max{dp[i-1][k]+j\*a[i]}(k>=j-1&&k<=j+s-1)**

这里有个点需要注意，就是由题目的样例可知结果可能出现负数，因此将0作为初始值并不正确，需要将dp数组初始化为一个比较小的数。可以用滚动数组进行优化空间，但是题目给的内存比较充足就不优化了。下载数据可知极限数据为n约等于5500，而这个dp的时间复杂度为O(n^2)，超过时间限制，考虑进行优化，我们发现，转移方程可以变成dp[i][j]=max{dp[i-1][k]}+a[i]\*j;这样处理之后就是求中间max的问题了，这里考虑对dp[i-1]进行排序，但是排序需要nlogn的复杂度，而且排序后的结果的第一个不一定就是我们需要的结果，因此我们想到单调队列，用单调队列讲新的符合条件的数据入队，并且检查，将不符合条件的结果从左边出队。出队条件为物品超过上一个状态的区间j+s-1，因此，这个单调队列还要存放下标，下面直接给出AC代码。

**AC代码如下：**

1. #include<bits/stdc++.h>
2. #define ll long long
3. **using** **namespace** std;
4. **class** node {
5. **public**:
6. ll index;
7. ll n;
8. }d[5505];
10. ll n, w, s;
11. ll a[5505];
12. ll dp[5505][5505];
13. ll ans = -1008600110086001;

16. **int** main() {
17. cin >> n >> w >> s;
18. **for** (ll i = 1; i <= n; i++) {
19. cin >> a[i];
20. }
21. **for** (ll i = 0; i <= n; i++)  **for** (ll j = 0; j <= w; j++)  dp[i][j] = -1008600110086001;
22. dp[0][0] = 0;
23. **for** (ll i = 1; i <= n; i++) {
24. ll L = 1, R = 1;
25. ll l = 1, r = 1;
26. d[L].n = dp[i - 1][w];
27. d[L].index = w;
28. **for** (ll j = w; j > 0; j--) {
29. **while** (L <= R && dp[i - 1][j - 1] > d[R].n)R--;
30. **while** (L <= R && d[L].index > j + s - 1)L++;
31. d[++R].index = j - 1;
32. d[R].n = dp[i - 1][j - 1];
33. dp[i][j] = d[L].n + a[i] \* j;
34. }
35. }
36. **for** (ll i = 0; i <= w; i++) {
37. ans = max(ans, dp[n][i]);
38. }
39. cout << ans << endl;
40. **return** 0;
41. }

### 5.1.1.5子序列问题

所谓子序列，就是一个序列中按顺序取若干个元素形成一个新的序列，而子序列问题就是形成子序列过程的问题，因为取子序列的过程满足动态规划无后效性的条件，并且取完整的子序列的过程可以分解为取一个个元素的子过程，因此通常使用动态规划来解决。

#### 1、最长上升子序列（LIS）

求最长上升子序列，需要枚举当前元素对前面所有元素的状态比较。我们需要知道加入第i个元素后前i个元素的最长上升子序列，只需要枚举前面每个元素对应的最长上升子序列。即转移方程为f[i]=max{1,f[k]+1}(第i个元素大于第k个元素)。最后只需要访问f数组中的最大之即可，时间复杂度为O(n^2)。

**代码如下**

1. **int** dp[100000];
2. **int** a[100000];
3. **int** main() {
4. **int** n;
5. **int** ans = 0;
6. cin >> n;
7. **for** (**int** i = 1; i <= n; i++) {
8. cin >> a[i];
9. }
10. **for** (**int** i = 1; i <= n; i++) {
11. dp[i] = 1;//子序列最短为1
12. **for** (**int** j = 1; j < i; j++) {
13. **if** (a[i] > a[j] && dp[i] < dp[j] + 1)dp[i] = dp[j] + 1;
14. }
15. ans = max(ans, dp[i]);
16. }
17. cout << ans << endl;
18. **return** 0;
19. }

上述方法的时间复杂度为O(n^2)，其实就是枚举每个元素对前面元素的最长上升子序列，因此效率较低。因此，有一种更高效的算法，使用有点贪心的策略，每次尽量使“刚好”大于已记录的最长子序列的结尾元素的数放入数组中，如果被记录的元素小于结尾元素，则更新结尾元素，但是为了数组依然保持单调，所以即使这个数并不是结尾元素，也要把这个元素则把这个元素放入计数数组中，为了避免出现的数据的特殊情况（即前面已经找到了长度为n的上升序列，然而后面又能找到长度为m的上升序列，后面子序列的最大值校园前子序列的最大值，且n<m，如果只更新结尾元素那么会导致f数组一直不能更新导致f数组长度一直不能变化导致的WA）。

**测试数据：**

20

10 11 12 13 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

**正确输出：**

16

**标准代码如下：**

1. **int** a[100000];
2. **int** f[10000000];
3. **int** main() {
4. **int** n;
5. cin >> n;
6. **for** (**int** i = 1; i <= n; i++) {
7. cin >> a[i];
8. f[i] = 0x7fffffff;
9. }
10. f[1] = a[1];
11. **int** len = 1;
12. **for** (**int** i = 2; i <= n; i++)
13. {
14. **int** l = 0, r = len, mid;
15. **if** (a[i] > f[len])f[++len] = a[i]; //如果刚好大于末尾，暂时向后顺次填充
16. **else**
17. {
18. **while** (l < r)
19. {
20. mid = (l + r) / 2;
21. **if** (f[mid] > a[i])r = mid;
22. **else** l = mid + 1;
23. }
24. f[l] = a[i];//更新最小末尾
25. }
26. }
27. cout << len;
28. **return** 0;
29. }

**注：f数组储存的内容不再是结果，而是记录子序列的状态（不一定记录最长上升子序列）**

**下面的代码提供C++STL中的lower\_bound()来实现的最长上升子序列**

**此函数筛选出大于等于第一个数的元素的指针（迭代器）。**

**upper\_bound()函数筛选大于次函数的第一个元素的指针（迭代器）。**

1. **int** dp[100000];
2. **int** a[100000];
3. **int** main() {
4. **int** n;
5. **int** len = 1;
6. **int** ans = 0;
7. cin >> n;
8. **for** (**int** i = 1; i <= n; i++) {
9. cin >> a[i];
10. }
11. dp[1] = a[1];
12. len = 1;
13. **for** (**int** i = 2; i <= n; i++)
14. {
15. **if** (a[i] > dp[len])
16. dp[++len] = a[i];//加在数列的最后
17. **else**
18. \*lower\_bound(dp+1, dp + len+1, a[i]) = a[i];
19. }
20. cout << len << endl;
21. **return** 0;
22. }

**下面的代码提供了在n^2的复杂度下实现记录最长上升子序列的方法**

1. **int** dp[100000];
2. **int** a[100000];
3. **int** f[100000];
4. **void** out(**int** p) {
5. **if** (p == 0)**return**;
6. out(f[p]);
7. cout << a[p] << ' ';
8. }
9. **int** main() {
10. **int** n;
11. **int** ans = 0;
12. **int** p = 0;
13. cin >> n;
14. **for** (**int** i = 1; i <= n; i++) {
15. cin >> a[i];
16. }
17. **for** (**int** i = 1; i <= n; i++) {
18. dp[i] = 1;//子序列最短为1
19. f[i] = 0;//前驱为0；
20. **for** (**int** j = 1; j < i; j++) {
21. **if** (a[i] > a[j] && dp[i] < dp[j] + 1) {
22. dp[i] = dp[j] + 1;
23. f[i] = j;
24. p = i;//记录最后的元素
25. }
26. }
27. }
28. **for** (**int** i = 1; i <= n; i++) {
29. ans = max(ans, dp[i]);
30. }
31. cout << ans << endl;
32. out(p);//迭代输出
33. cout << endl;
34. **return** 0;
35. }

**变式1：（洛谷1439）**

**题目描述**

给出 1,2,…,n的两个排列 P1和 P2​ ，求它们的最长公共子序列。

**输入格式**

第一行是一个数 n。

接下来两行，每行为 n个数，为自然数 1,2,…,n 的一个排列。

**输出格式**

一个数，即最长公共子序列的长度。

**思路**

这道题是求全排列的性质来实现重新标记，把问题转化为求最长上升子序列，使时间复杂度降到了O(nlogn).

**代码如下：**

1. **int** lib[100005];
2. **int** arr[100005];
3. **int** dp[100005];
4. **int** main() {
5. **int** n;
6. **int** temp;
7. **int** len = 1;
8. cin >> n;
9. **for** (**int** i = 1; i <= n; i++) {
10. cin >> temp;
11. lib[temp] = i;
12. }
13. **for** (**int** i = 1; i <= n; i++) {
14. cin >> temp;
15. arr[i] = lib[temp];
16. dp[i] = 0x7fffffff;
17. }
18. dp[1] = arr[1];
19. **for** (**int** i = 2; i <= n; i++) {
20. **if** (arr[i] > dp[len])dp[++len] = arr[i];
21. **else** \*lower\_bound(dp + 1, dp + len + 1,arr[i]) = arr[i];
22. }
23. cout << len << endl;
24. **return** 0;
25. }

**变式2：洛谷1108低价购入**

**题目描述**

“低价购买”这条建议是在奶牛股票市场取得成功的一半规则。要想被认为是伟大的投资者，你必须遵循以下的问题建议:“低价购买；再低价购买”。每次你购买一支股票,你必须用低于你上次购买它的价格购买它。买的次数越多越好!你的目标是在遵循以上建议的前提下，求你最多能购买股票的次数。你将被给出一段时间内一支股票每天的出售价(2^{16}216范围内的正整数)，你可以选择在哪些天购买这支股票。每次购买都必须遵循“低价购买；再低价购买”的原则。写一个程序计算最大购买次数。

这里是某支股票的价格清单：

日期 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8, 9 ,10 ,11, 121,2,3,4,5,6,7,8,9,10,11,12

价格68 ,69 ,54, 64,68 ,64 ,70 ,67 ,78 ,62, 98, 8768,69,54,64,68,64,70,67,78,62,98,87

最优秀的投资者可以购买最多44次股票，可行方案中的一种是：

日期 2 , 5 , 6 ,102,5,6,10

价格 69, 68 ,64 ,6269,68,64,62

**输入格式**

第1行: N(1 \le N \le 5000)*N*(1≤*N*≤5000)，股票发行天数

第2行: N*N*个数，是每天的股票价格。

**输出格式**

两个数:  
最大购买次数和拥有最大购买次数的方案数(\le 2^{31}≤231)当二种方案“看起来一样”时（就是说它们构成的价格队列一样的时候）,这22种方案被认为是相同的。

**思路**

这道题不是一道简单的最长下降子序列的问题（因为要求最长下降子序列的个数）。方法是对长度数组dp再进行一次dp。令数组f为以i元素结尾的最长下降子序列的长度，如果以i元素结尾的最长下降子序列的长度为dp[i]，那么如果存在j满足dp[j]==dp[i]-1&&a[i]<a[j],那么就可以把以j为结尾的最长下降子序列的值转移到f[i],即f[i]+=f[j]。但是这样会有重复的情况，即题目里说的如果两个序列中的元素完全一样则视为同一个序列，那么我们需要对这种情况进行去重。如果这两个序列的结尾元素相同并且长度相同，那么这两种序列的种类中一定会存在相同的序列，（假设这两个序列不相同，那么我们设想在i和j元素之间插入了元素而少取了一个1到i中的其中一个元素，如果能取到这个元素就必须保证这个元素大于i元素并且大于j元素这里再分两种情况讨论，如果取到的k元素大于j元素的前一个元素且小于被比较元素前面的那个元素，此时产生的序列是不同的序列，如果k元素小于j元素的前一个元素，那么长度一定会比原来的长度要长，与假设矛盾），这时要将f[j]的值清空防止重复计数，这里解神一下为什么要清空，清空之后以j结尾的最长下降子序列的个数情况，但是这个种类从会j的前置元素转移到i元素，以题目的数据为例，假设在数据结尾再加一个62，那么第一个以62为结尾的最长下降子序列中类从2->0,但是第二个62又重新从64和67中转移得到了2，因此从而了重复计数。

**代码如下：**

1. **int** dp[100000];
2. **int** a[100000];
3. **int** f[100000];
4. **int** main() {
5. **int** n;
6. **int** ans = 0;
7. **int** sum = 0;
8. cin >> n;
9. **for** (**int** i = 1; i <= n; i++) {
10. cin >> a[i];
11. }
12. **for** (**int** i = 1; i <= n; i++) {
13. dp[i] = 1;//子序列最短为1
14. **for** (**int** j = 1; j < i; j++) {
15. **if** (a[i] < a[j] && dp[i] < dp[j] + 1)dp[i] = dp[j] + 1;
16. }
17. **if**(dp[i]==1)f[i] = 1;
18. **if** (ans < dp[i]) {
19. ans = dp[i];
21. }
22. **for** (**int** j = 1; j < i; j++) {
23. **if** (dp[i] == dp[j] + 1 && a[j] > a[i])f[i] += f[j];
24. **if** (dp[i] == dp[j] && a[i] == a[j])f[j] = 0;
25. }
26. }
27. **for** (**int** i = 1; i <= n; i++)sum += dp[i] == ans ? f[i] : 0;
28. cout << ans <<' '<< sum << endl;
29. **return** 0;
30. }

**2、最长公共**子**序列（LCS）**

最长公共子序列的思路和求最长上升子序列的思路差不多。当s1的第i个元素不等于s2的第j个元素时，转移i-1,j和i,j-1的最大值，s1[i]=s2[j]时，还要比较dp[i-1][j-1]+1的值

**代码如下：(时间复杂度O(n^2))**

1. **int** dp[1001][1001], a1[2001], a2[2001], n, m;
2. **int** main()
3. {   //dp[i][j]表示两个串从头开始，直到第一个串的第i位
4. //和第二个串的第j位最多有多少个公共子元素
5. cin >> n >> m;
6. **for** (**int** i = 1; i <= n; i++)cin>>a1[i];
7. **for** (**int** i = 1; i <= m; i++)cin>>a2[i];
8. **for** (**int** i = 1; i <= n; i++)
9. **for** (**int** j = 1; j <= m; j++)
10. {
11. dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
12. **if** (a1[i] == a2[j]) dp[i][j] = max(dp[i][j], dp[i - 1][j - 1] + 1);
13. //因为更新，所以++；
14. }
15. cout << dp[n][m] << endl;
16. }

### 5.2.2背包问题

背包问题就是把一定价值的物品放入一定容量的背包的问题。通常来说背包问题分为01背包、完全背包和多重背包问题。通常来说，背包问题有固定的状态转移方程来解决。

**5.2.2.1 01背包（洛谷p1048）**

转移方程：f[v]=max(f[v],f[v-w[i]]+v[i])

原理：两层循环枚举，外层循环枚举准备放入背包的物品，内层循环枚举放入物品后背包的容量（从满背包到刚好放入第i件物品时的容量），反向遍历使得某一物品只对背包最优状态判断1次，从而实现判断放或不放的状态。

**代码：//v代表价值，w代表质量**

1. **int** v[1000000], w[1000000], dp[10000000];
2. **int** main() {
3. **int** t, m;
4. cin >> t >> m;
5. **for** (**int** i = 1; i <= m; i++) {
6. cin >> w[i] >> v[i];
7. }
8. **for** (**int** i = 1; i <= m; i++) {
9. **for** (**int** j = t; j >= w[i]; j--) {
10. dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
11. }
12. }
13. cout << dp[t] << endl;
14. **return** 0;
15. }

**变式：洛谷1077摆花（NOIP普及组2012）**

**题目描述**

小明的花店新开张，为了吸引顾客，他想在花店的门口摆上一排花，共m*m*盆。通过调查顾客的喜好，小明列出了顾客最喜欢的n*n*种花，从11到n*n*标号。为了在门口展出更多种花，规定第i*i*种花不能超过a\_i*ai*​盆，摆花时同一种花放在一起，且不同种类的花需按标号的从小到大的顺序依次摆列。试编程计算，一共有多少种不同的摆花方案。

**输入格式**

第一行包含两个正整数n*n*和m*m*，中间用一个空格隔开。

第二行有n*n*个整数，每两个整数之间用一个空格隔开，依次表示a\_1,a\_2,…,a\_n*a*1​,*a*2​,…,*an*​。

**输出格式**

一个整数，表示有多少种方案。注意：因为方案数可能很多，请输出方案数对10000071000007取模的结果。

**思路**

该问题求方案总数，很容易想到其最优子结果为摆到第i种花，已经摆了j盆花的的结果。由于第i种花摆放个数有限制，很容易想到需要第3个变量来表示第i种花摆的盆数。因此，我们定义k为第i盆花摆的盆数。第i种花摆了k盆，此时摆了j盆花，方案总数等于摆了第i-1种花，背包容量为j-k的方案数，

因此，转移方程为dp[i][j]=

我们发现数组每次只用到2\*m的大小，因此想用滚动数组取优化，观察上面的转移方程，跟背包的转移方程非常类似，因此我们想到去掉第一维，只保留j维，等于dp[j]为取了7盆花时的方案总数，dp[j]=.

从而证明了这道题其实是01背包的变式（枚举取第i种花，当前取了j盆花的方案总数，第i种花的盆数（即第i物品的体积是可变的，枚举其体积的值））。

这道题说明了背包问题不仅仅能求最优解，还能求方案总数。

**代码如下：**

1. **int** a[105];
2. **int** dp[105];
3. **const** **int** x = 1000007;
4. **int** main() {
5. **int** n, m;
6. **int** sum = 0;
7. cin >> n >> m;
8. **for** (**int** i = 1; i <= n; i++) {
9. cin >> a[i];
10. }
11. dp[0] = 1;
12. **for** (**int** i = 1; i <= n; i++) {
13. **for** (**int** j = m; j >=0; j--) {
14. **for** (**int** k = 1; k <= min(j,a[i]); k++) {
15. dp[j] = (dp[j] + dp[j - k] % x) % x;
16. }
17. }
18. }
19. cout << dp[m] << endl;
20. **return** 0;
21. }

**5.2.2.2完全背包（洛谷1616）**

转移方程同01背包

**原理：**两层循环同01背包，正向遍历实现，使得每一物品在遍历时将物品件数正向传递，使得传递到背包容量上限时可能出现多次同样的物品。

**代码：//v代表价值，w代表质量**

1. **int** v[1000000], w[1000000], dp[10000000];
2. **int** main() {
3. **int** t, m;
4. cin >> t >> m;
5. **for** (**int** i = 1; i <= m; i++) {
6. cin >> w[i] >> v[i];
7. }
8. **for** (**int** i = 1; i <= m; i++) {
9. **for** (**int** j = w[i]; j <= t; j++) {
10. dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
11. }
12. }
13. cout << dp[t] << endl;
14. **return** 0;
15. }

**5.2.2.3 多重背包（ZCMU1919）**

**题目描述**  
现在商店有N个药品，物品栏有W的容量。  
第i个药品有重量w\_i,可以恢复法力值v\_i，有数量c\_i个。

计算可以恢复的最大法力值。

**输入**

第一行两个整数N,W(1 <= N <= 300,1 <= W <= 500000 )  
接下来N行，每行三个整数w\_i,v\_i,c\_i(1 <= w\_i <= 10000,1 <= v\_i <= 10000, 1 <= c\_i <= 500)

**输出**

输出一个整数

**输入样例**

3 6

2 2 5

3 3 8

1 4 1

**输出样例**

9

**转移方程同01背包**

**原理：**把多重背包转换为01背包来解决，方法为2进制优化，把一个数转化为1+2+4+……+2^i+k，每个加数都变为一个新的元素，这样使其转化成01背包问题

**代码如下：**

1. **int** dp[1000000];
2. **int** v[1000000], w[1000000];
3. **int** main() {
4. **int** n, t;
5. **int** num = 0;
6. cin >> n >> t;
7. **for** (**int** i = 1; i <= n; i++) {
8. **int** v1, w1, c1;
9. **int** b = 1;
10. cin >> w1 >> v1 >> c1;
11. **while**(c1 - b > 0) {
12. num++;
13. v[num] = b \* v1;
14. w[num] = b \* w1;
15. c1 -= b;
16. b \*= 2;
17. }
18. num++;
19. w[num] = w1 \* (c1);
20. v[num] = v1 \* (c1);
21. }
22. **for** (**int** i = 1; i <= num; i++) {
23. **for** (**int** j = t; j >= w[i]; j--) {
24. dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
25. }
26. }
27. cout << dp[t] << endl;
28. **return** 0;
29. }

**5.2.2.4 混合背包（01、完全、多重）ACWing 7**

**题目描述**

有 NN 种物品和一个容量是 VV 的背包。

物品一共有三类：

* 第一类物品只能用1次（01背包）；
* 第二类物品可以用无限次（完全背包）；
* 第三类物品最多只能用 sisi 次（多重背包）；

每种体积是 vivi，价值是 wiwi。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。  
输出最大价值。

**输入格式**

第一行两个整数，N，VN，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 NN 行，每行三个整数 vi,wi,sivi,wi,si，用空格隔开，分别表示第 ii 种物品的体积、价值和数量。

* si=−1 表示第 ii 种物品只能用1次；
* si=0 表示第 ii 种物品可以用无限次；
* si>0 表示第 ii 种物品可以使用 si 次；

输出格式

输出一个整数，表示最大价值。

数据范围

0<N,V≤1000

0<vi,wi≤1000  
−1≤si≤1000

**输入样例**

4 5

1 2 -1

2 4 1

3 4 0

4 5 2

**输出样例**

8

**思路**

混合背包问题，我们只需要知道每个物品属于的背包类型，然后按照那个背包的解题方法进行转移即可。

**AC代码**

结合01，完全，多重背包的代码，这里不再给出

**5.2.2.5 二维费用背包 ACWing 8**

**题目描述**

有 N 件物品和一个容量是 V 的背包，背包能承受的最大重量是 M。

每件物品只能用一次。体积是 vi，重量是 mi，价值是 wi。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，总重量不超过背包可承受的最大重量，且价值总和最大。

输出最大价值。

**输入格式**

第一行两个整数，N，V,M，用空格隔开，分别表示物品件数、背包容积和背包可承受的最大重量。

接下来有 N 行，每行三个整数 vi,mi,wi，用空格隔开，分别表示第 i 件物品的体积、重量和价值。

**输出格式**

输出一个整数，表示最大价值。

数据范围

0<N≤1000

0<V,M≤100

0<vi,mi≤100

0<wi≤1000

**输入样例**

4 5 6

1 2 3

2 4 4

3 4 5

4 5 6

**输出样例**

8

**思路**

所谓二维费用，就是在背包里放入该物品是会消耗两种不同类型的费用，我们只需要枚举背包已经使用的每一种费用（比如消耗了v个空间，承受了m个质量）所取得的最大价值，然后就是跟01背包一样的转移方程。

**AC代码**

1. **int** n, v, m;
2. **int** wi[N];
3. **int** vi[N];
4. **int** mi[N];
5. **int** dp[N][N];
6. **int** main() {
7. cin >> n >> v >> m;
8. **for** (**int** i = 1; i <= n; i++) {
9. cin >> vi[i] >> mi[i] >> wi[i];
10. }
11. **for** (**int** i = 1; i <= n; i++) {
12. **for** (**int** j = v; j >= vi[i]; j--) {
13. **for** (**int** k = m; k >= mi[i]; k--) {
14. dp[j][k] = max(dp[j][k], dp[j - vi[i]][k - mi[i]] + wi[i]);//放入了体积为j，质量为k的最大价值
15. }
16. }
17. }
18. cout << dp[v][m] << endl;
19. **return** 0;
20. }

**5.2.2.6 分组背包 ACWing 9**

**题目描述**

有 N 组物品和一个容量是 V 的背包。

每组物品有若干个，同一组内的物品最多只能选一个。

每件物品的体积是 vij，价值是 wij，其中 i 是组号，j 是组内编号。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

输入格式

第一行有两个整数 N，V，用空格隔开，分别表示物品组数和背包容量。

接下来有 N 组数据：

每组数据第一行有一个整数 Si，表示第 i 个物品组的物品数量；

每组数据接下来有 Si 行，每行有两个整数 vij,wij，用空格隔开，分别表示第 i 个物品组的第 j 个物品的体积和价值；

**输出格式**

输出一个整数，表示最大价值。

**数据范围**

0<N,V≤100

0<Si≤100

0<vij,wij≤100

**输入样例**

3 5

2

1 2

2 4

1

3 4

1

4 5

**输出样例**

8

**思路**

分组背包问题跟01背包其实很相似，思路近似于把每个组当成一个物品，每个组相当于价值和费用可变的一个物品，然后对每个组进行01背包即可。

**AC代码**

1. **const** **int** N = 1e3 + 5;
2. **int** n, v;
3. **int** s[N];//k
4. **int** dp[N];
5. **int** vi[N][N];
6. **int** wi[N][N];
7. **int** main() {
8. cin >> n >> v;
9. **for** (**int** i = 1; i <= n; i++) {
10. cin >> s[i];
11. **for** (**int** j = 1; j <= s[i]; j++) {
12. cin >> vi[i][j] >> wi[i][j];
13. }
14. }
15. **for** (**int** i = 1; i <= n; i++) {
16. **for** (**int** j = v; j >= 0; j--) {
17. **for** (**int** k = 1; k <= s[i]; k++) {
18. **if**(j >= vi[i][k])dp[j] = max(dp[j], dp[j - vi[i][k]] + wi[i][k]);
19. }
20. }
21. }
22. cout << dp[v] << endl;
23. **return** 0;
24. }

**变式 HDU6968 I love exam**

**题目大意**

你有n门课要考试，有m本教科书，每本教科书要y天复习，考试成绩提高x分，每门考试满分100分，及格60分，你最多挂p门课，你只有t天复习时间，问最高能得几分

**输入**

输入包含组数据

第一行为一个整数T，表示一共T组数据（T<=10）

对于每组数据：

第一行为一个整数n（n<=50）,表示有n门课。

接下来n行包含n个不超过15字符的字符串，表示课程名称。

接下来一行为一个整数m（m<=15000），表示一共m本书，

接下来m行，每行有一个字符串s，两个整数x,y（1<=x,y<=10），表示这本书针对课程名称为s，学这本书提高x分，需要花y天时间学完。

最后一行两个个整数t,p(1<=t<=500,0<=p<=3)，表示复习时间为t天，最多挂p门课。

**输出**

对于每组数据，输出一个整数z表示最多可以获得的分数，如果不存在可行状态就输出-1

**样例输入**

1

3

mathematics physics signals

20

physics 10 1

physics 10 1

physics 10 1

physics 10 1

physics 10 1

physics 10 1

physics 10 1

mathematics 10 1

mathematics 10 1

mathematics 10 1

mathematics 10 1

mathematics 10 1

mathematics 10 1

mathematics 10 1

signals 10 1

signals 10 1

signals 10 1

signals 10 1

signals 10 1

signals 10 2

19 1

**样例输出**

190

**思路**

这道题是很明显的背包，问题就在于把这个问题转化为基础的背包问题。首先我们可以想到对每一门科目进行01背包，求得花费若干天得到的最佳分数。然后我们发现，我们只需要把每门课所得的分数和花费的天数进行背包即可。对于每一门课，会出现多种可能的花费天数和可能的分数，但是我们只能取一种可行的分数和天数，这就符合了分组背包的处理规则，本题还需要知道挂科数量，因此我们需要记录每个可行状态的挂科门数，最后的答案就是dp[n][d][i]中i取0->p的最大值。

**AC代码**

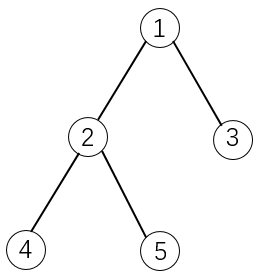
1. #include<bits/stdc++.h>
2. //#define cin in
3. #define ll long long
4. **using** **namespace** std;
5. **const** **int** N = (1 << 19) + 5;
6. **const** ll mod = 998244353;
7. **class** node {
8. **public**:
9. **int** x, y;
10. };
11. ifstream in;
12. **int** t;
13. **int** n;
14. **int** m;
15. **int** d, p;
16. **string** s[55];
17. vector<node>lib[55];
18. **int** f[50][505];
19. **int** dp[50][505][55];
21. node ttt;
23. **int** main() {
24. //in.open("D:\\OneDrive\\桌面\\repos\\ConsoleApplication1\\Debug/1.in");
25. cin >> t;
26. **while** (t--) {
27. cin >> n;
28. memset(f, 0, **sizeof**(f));
29. **for** (**int** i = 1; i <= n; i++) {
30. cin >> s[i];
31. }
32. cin >> m;
33. **for** (**int** i = 1; i <= n; i++) {
34. lib[i].clear();
35. lib[i].push\_back(ttt);
36. }
37. **for** (**int** i = 1; i <= m; i++) {
38. **int** x, y;
39. **string** str;
40. cin >> str >> x >> y;
41. **for** (**int** j = 1; j <= n; j++) {
42. **if** (str == s[j]) {
43. ttt.x = x;
44. ttt.y = y;
45. lib[j].push\_back(ttt);
46. **break**;
47. }
48. }
49. }
50. cin >> d >> p;
51. **for** (**int** L = 1; L <= n; L++) {
52. f[L][0] = 0;
53. **for** (**int** i = 1; i < lib[L].size(); i++) {
54. **for** (**int** j = d; j >= lib[L][i].y; j--) {
55. f[L][j] = min(100, max(f[L][j], f[L][j - lib[L][i].y] + lib[L][i].x));
56. }
57. }
58. }
59. memset(dp, -0x3f, **sizeof**(dp));
60. dp[0][0][0] = -0;
61. dp[0][0][1] = -0;
62. **for** (**int** k = 1; k <= n; k++) {
63. **for** (**int** j = d; j >= 0; j--) {
64. **for** (**int** i = j; i >= 0; i--) {
65. **for** (**int** temp = 0; temp <= p ; temp++) {
66. **if** (f[k][i] < 60) {
67. dp[k][j][temp + 1] = max(dp[k - 1][j - i][temp] + f[k][i], dp[k][j][temp + 1]);
68. }
69. **else** {
70. dp[k][j][temp] = max(dp[k - 1][j - i][temp] + f[k][i], dp[k][j][temp]);
71. }
72. }
73. }
74. }
75. }
76. **int** ans = -1;
77. **for** (**int** i = 0; i <= p; i++) {
78. ans = max(ans, dp[n][d][i]);
79. }
80. cout << ans << endl;
81. }
82. **return** 0;
83. }

**5.2.2.7 有依赖的背包问题 ACWing10**

**问题描述**

有 NN 个物品和一个容量是 VV 的背包。

物品之间具有依赖关系，且依赖关系组成一棵树的形状。如果选择一个物品，则必须选择它的父节点。

如下图所示：

**4.2.2.7图**

如果选择物品5，则必须选择物品1和2。这是因为2是5的父节点，1是2的父节点。

每件物品的编号是 ii，体积是 vi，价值是 wi，依赖的父节点编号是 pi。物品的下标范围是 1…N1…N。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

输出最大价值。

**输入格式**

第一行有两个整数 N，VN，V，用空格隔开，分别表示物品个数和背包容量。

接下来有 N 行数据，每行数据表示一个物品。  
第 ii 行有三个整数 vi,wi,pivi,wi,pi，用空格隔开，分别表示物品的体积、价值和依赖的物品编号。  
如果 pi=−1pi=−1，表示根节点。 **数据保证所有物品构成一棵树。**

**输出格式**

输出一个整数，表示最大价值。

**数据范围**

1≤N,V≤1001≤N,V≤100  
1≤vi,wi≤1001≤vi,wi≤100

父节点编号范围：

* 内部结点：1≤pi≤N1≤pi≤N;
* 根节点 pi=−1pi=−1;

**输入样例**

5 7

2 3 -1

2 2 1

3 5 1

4 7 2

3 6 2

**输出样例：**

11

**思路**

这道题来自背包九讲，源自NOIP2006提高组的金明的预算方案，金明的预算方案规定主件最多只有2个附件，而且附件不能再有附件，因此简单了许多，只需做4次转移：只有主件，有附件1，有附件2，有附件1和2。本题有所不同，主件有多个附件，附件还能再有附件。因此dp数组开1维已经不能使用，需要记录已取的物品。我们令dp[i][j]表示已经取了i物品，使用了背包容量为j时的最大价值。本题保证数据一定能成树，我们只需记录根节点然后进行dfs即可（dfs跑到结尾，计算出儿子节点物品的最优取法，然后再在记录父亲节点的数组中进行取不取儿子节点最优解的01背包即可）。由于本题取儿子节点的物品必须要取父亲节点，因此必须要多开1维dp数组来存储取了每一个节点的最优解。转移的方式是先预留出存放该节点的背包容量，计算出不放该节点的最优解，然后将每个结果移到容量加上该物品容量的结果中。下面先放转移方程：

**dp[indexi][j] = max(dp[indexi][j],dp[indexi][j - k]+ dp[a[indexi].son[i]][k]);**

其中indexi表示当前节点下标，j表示使用的背包容量。

接下来放本题的AC代码，本题的AC代码稍做修改可以也AC掉CTSC1997选课，在此处不再重复给出。这一类有依赖的背包问题也被称为背包类树形dp问题。

**AC代码如下**

1. #include<bits/stdc++.h>
2. #define ll long long
3. **using** **namespace** std;
4. **const** **int** N = 1e3 + 5;
5. **class** node {
6. **public**:
7. vector<**int**> son;
8. **int** v, w;
9. }a[N];
10. **int** n, v;
11. **int** dp[N][N];
12. **int** fw[N];
13. **int** t;
14. **int** ans;
15. **int** father;
16. **void** dfs(**int** indexi) {
17. **for** (**int** i = 0; i < a[indexi].son.size(); i++) {
18. dfs(a[indexi].son[i]);
19. **for** (**int** j = v - a[indexi].v; j >= 0; j--) {
20. **for** (**int** k = 0; k <= j; k++) {
21. dp[indexi][j] = max(dp[indexi][j], dp[indexi][j - k]+ dp[a[indexi].son[i]][k]);
22. }
23. }
24. }
25. **for** (**int** i = v; i >= a[indexi].v; i--)dp[indexi][i] = a[indexi].w + dp[indexi][i - a[indexi].v];
26. **for** (**int** i = 0; i < a[indexi].v; i++)dp[indexi][i] = 0;
27. }
29. **int** main() {
30. cin >> n >> v;
31. **for** (**int** i = 1; i <= n; i++) {
32. cin >> a[i].v >> a[i].w >> t;
33. **if** (t != -1)a[t].son.push\_back(i);
34. **else** father = i;
35. }
36. dfs(father);
37. ans =  dp[father[i]][v];
38. cout << ans << endl;
39. **return** 0;
40. }

**5.2.2.7背包问题求方案 ACWing10**

**题目描述**

有 N 件物品和一个容量是 V 的背包。每件物品只能使用一次。

第 i 件物品的体积是 vi，价值是 wi。

求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。

输出 最优选法的方案数。注意答案可能很大，请输出答案模 1e9+7 的结果。

**输入格式**

第一行两个整数，N，V 用空格隔开，分别表示物品数量和背包容积。

接下来有 N 行，每行两个整数 vi,wi，用空格隔开，分别表示第 i 件物品的体积和价值。

**输出格式**

输出一个整数，表示 方案数 模 1e9+7 的结果。

**数据范围**

0<N,V≤1000

0<vi,wi≤1000

**输入样例**

4 5

1 2

2 4

3 4

4 6

**输出样例：**

2

**思路**

这题就是基础的背包问题求最优方案数，考虑在01背包的基础上，再进行方案数的状态转移，令f[i][j]表示前i个物品，占用背包容量j是的方案数，可得转移方程：

**f[i][j]=f[i-1][j](当前枚举的物品不放入为最优)**

**f[i][j]=f[i-1][j-v[i]](当前枚举的物品放入最优)**

**f[i][j]=f[i-1][j]+f[i-1][j-v[i]](当前枚举物品放入与不放入都存在最优情况)**

这样其实已经可以通过，但是为了提高效率，我们考虑像01背包那样压缩数组以提高效率。很显然我们也可以像01背包一样令f[j]为背包容量为j时的方案总数。我们需要对f数组进行初始化，这里有两种理解：

1、f[j]为恰好使用了j容量的背包时的方案总数，那么我们要初始化f[0]=1，其余为0.因为恰好使用了0个容量的背包时，有1种方案。最后要输出所有满足dp[j]==dp[t]的f[j]的和。

2、f[j]为当背包容量为j（不一定要用完）时的方案总数，那么我们需要初始化f数组的值全部为1，因为不管背包容量有多少，至少有1种方案：即什么都不放。最后要输出f[t]的值（即背包容量为t时的方案数）。

很显然，二者都是正确的，而且第二种理解为数组维数压缩后特有的解法。下面我以第二种理解为例给出AC代码。

**AC代码如下：**

1. **int** dp[M];
2. **int** f[M];
3. **int** g[M][M];
4. **int** v[M], w[M];
5. **int** n;
6. **int** t;
8. **int** main() {
9. cin >> n >> t;
10. **for** (**int** i = 1; i <= n; i++) {
11. cin >> v[i] >> w[i];
12. }
13. **for**(**int** i=0;i<=t;i++){
14. f[i]=1;
15. }
16. **for**(**int** i=1;i<=n;i++){
17. **for**(**int** j=t;j>=v[i];j--){
18. **if**(dp[j]<dp[j-v[i]]+w[i]){
19. dp[j]=dp[j-v[i]]+w[i];
20. f[j]=f[j-v[i]];
21. }**else** **if**(dp[j]==dp[j-v[i]]+w[i]){
22. f[j]+=f[j-v[i]];
23. f[j]%=mod;
24. }
25. }
26. }
27. **int** ans = 0;
29. cout << f[t] << endl;
30. **return** 0;
31. }

## 5.3.区间型动态规划

区间型动态规划，指在一个线性条件下对区间用某种方法进行合并操作，求解这个过程的最优解的过程，其实，对一个环也可以进行区间dp，只需进行断环成链操作即可（石子合并），这种类型的dp一般情况下有非常明显的区间合并或类似操作，因此解题思路非常固定，转移方程也比较类似。

**具体解题过程：**

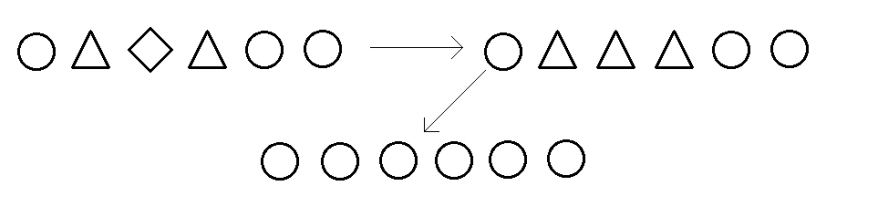
1. 确定要进行合并的区间和区间合并的方法及计分方法
2. 枚举区间长度
3. 枚举左节点，计算出右节点
4. 拓展区间（一般从上个长度减1的区间继承得到）或给此区间的最优接赋初值（一般为0或无穷）
5. 枚举中间节点（有些题目不需要枚举中间节点如关路灯，不存在中间节点合并）
6. 统计最优解

区间dp的难点一般在于找用于操作的区间，一旦找到了这个区间，题目就比较简单了。

**5.3.1 典型例题1 Cities(ICPC2021昆明站C题)**

**题目大意**

给出一个长度为 n的数组，一次操作可以选择一段连续且值都相同的区间，然后使其变成其他值，问最少需要多少次操作，才能使得数组的值全相等。数组中每个数最多出现15次。



**输入描述:**

第一行输入一个整数t (1≤t≤160)t。表示有t组数据

对于每组数据：  
第一行输入一个整数n (1≤n≤5000)n表示城市的数量  
第二行输入n个整数a[i] (1≤ai≤n)，第i个城市为a[i]城主拥有。可以保证，目前没有主将拥有超过 15 个城市，这意味着没有人 ai 将出现超过 15 次在此 $a行。目前没有主将有超过15个城市，这意味着没有人会出现超过15，在这一行。 保证所有测试案例的 n 总和不超过 6000。

**输出描述:**

对于每组数据，输出答案

**思路**

初见这题很容易想到区间dp，但是传统的区间dp的时间复杂度为O(n^3)，不能满足此题的要求，因此必须对转移方程进行优化。刚开始想到区间dp的万能转移方程：dp[i][j]=min(dp[i][k],dp[k+1][j])，这样dp[i][j]需要一个初始值，我们将初始值定为最大结果（6001）。但是这题后续的优化否定了这个初始值。为了节约时间，我们要尽可能减少对k值的枚举，对此可以想到一个贪心的策略，当枚举的k值等于j值时，我们就不需要再进行将k变成j的这步操作，从而实现减少k数量的枚举。那么初始值怎么确定呢？要确定dp[i][j]的值，我们需要用到拓展区间的方法，即将比我们当前枚举的区间小1的区间转移过来，进行将新加入的元素变成当前区间的元素的操作，即dp[i][j]=min(dp[i+1][j],dp[i][j-1])+1.

综上所述，转移方程为：

**dp[i][j]=min{dp[i+1][j]+1,dp[i][j-1]+1,dp[i][k]+dp[k+1][j]}**

如何实现仅枚举与j值相同的k值呢？其实只需要在录入的时候进行一次循环向前遍历，找到第一个等于该数的元素，将其下标储存在数组里，枚举k值的时候只需要访问这个数组即可实现。

**AC代码如下：**

1. **int** dp[7005][7005];
2. **int** a[7005];
3. **int** p[7005];
4. **int** main() {
5. **int** t;
6. **int** n;
7. scanf("%d", &t);
8. **while** (t--) {
9. cin >> n;
10. //memset(dp, 0, sizeof(dp));
11. **for** (**int** i = 1; i <= n; i++) {
12. **int** j = i - 1;
13. //cin >> a[i];
14. scanf("%d", &a[i]);
15. **if** (a[i] == a[i - 1]) {
16. n--;
17. i--;
18. }
19. **else** {
20. **while** (j > 0 && a[j] != a[i]) {
21. j--;
22. }
23. p[i] = j;
24. }
25. }
26. **for** (**int** l = 2; l <= n; l++) {
27. **for** (**int** i = 1; i <= n - l + 1; i++) {
28. **int** j = i + l - 1;
29. dp[i][j] = min(dp[i + 1][j], dp[i][j - 1]) + 1;
30. **for** (**int** k = p[j]; k >= i; k = p[k]) {
31. dp[i][j] = min(dp[i][j], dp[i][k] + dp[k + 1][j]);
32. }
33. //if (a[i] != a[j])dp[i][j]++;
34. }
35. }
36. printf("%d\n", dp[1][n]);
37. **if** (t)**for** (**int** i = 1; i <= n; i++) {
38. **for** (**int** j = 1; j <= n; j++) {
39. dp[i][j] = 0;
40. }
41. }
42. }
43. **return** 0;
44. }

**5.3.2 典型例题2 洛谷P3146 [USACO16OPEN]248 G**

**题目描述**

给定一个1\*n的地图，在里面玩2048，每次可以合并相邻两个（数值范围1-40），问序列中出现的最大数字的值最大是多少。注意合并后的数值并非加倍而是+1，例如2与2合并后的数值为3。

**输入格式**

第一行输入一个整数n，表示n个数字，接下来一行n个数字，表示开始时的数字

**输出格式**

输出最后答案

**输入**

4

1

1

1

2

**输出**

3

**说明/提示**

在此处显示的示例中，贝茜首先将第二个和第三个 1 合并为2，然后她将2 2合并为3。

合并前2个1不是最佳方案。

**思路** 这道题是典型的区间dp，思考过程可以完全套用区间dp的通解，这里需要注意的是，根据题意可知，只有相同的两个数才能进行合并操作，因此最终的结果不一定是dp[1][n]，而应该是每次操作后的结果的最大值（输入也算）。这里直接给出转移方程。

**dp[i][j]=max{dp[i][k]+1}(dp[i][k]==dp[k+1][j])**

每次进行操作时记录此时的最大结果，程序执行结束时的最大结果就是此值。

**下面给出AC代码：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** n;
4. **int** a[1005];
5. **int** dp[1005][1005];
6. **int** ans = 0;
7. **int** main() {
8. scanf("%d", &n);
9. **for** (**int** i = 1; i <= n; i++) {
10. scanf("%d",&a[i]);
11. dp[i][i] = a[i];
12. ans = max(ans, a[i]);
13. }
14. **for** (**int** L = 2; L <= n; L++) {
15. **for** (**int** i = 1; i <= n - L + 1; i++) {
16. **int** j = i + L - 1;
17. **for** (**int** k = i; k < j; k++) {
18. **if** (dp[i][k] == dp[k + 1][j]) {
19. dp[i][j] = max(dp[i][j], dp[i][k] + 1);
20. ans = max(ans, dp[i][j]);
21. }
22. }
23. }
24. }
25. cout << ans << endl;
26. **return** 0;
27. }

## 5.4.状态压缩型动态规划

状态压缩型动态规划，顾名思义，就是用到状态压缩的动态规划，状态压缩，就是将本应该用数组来储存的一串序列的状态用一个数来表示，从而达到节约空间的目的。

### 5.4.1 预备知识

**5.4.1.1位运算**

由于储存状态的数一般用二进制来表示，用取每一位的值（0或1）来判断当前状态，因此我们需要用位运算来实现高效执行。这里不做说明，详细见《信息竞赛一本通提高篇》。

**5.4.1.2状态预处理**

状态预处理可大幅减少空间并且能一定程度上降低时间，方法就是在程序开始的时候遍历所有状态，将可能的状态记录，从而减少dp数组所需的空间，s[i]状态在dp数组中只需要用i来表示。 以洛谷1896和洛谷2704为例。

**洛谷1896**：为了节约时间，我们需要在程序执行前枚举出所有的一行摆放方案，枚举可以用搜索，这里直接用状态压缩枚举所有状态，筛选出符合条件的摆放方案状态储存在数组s中，并且将该状态摆放的国王数储存在数组num中。

代码如下：

1. **for** (**int** i = 0; i < (1 << n); i++) {
2. **if** (i & (i << 1))**continue**;//不符合条件，舍去
3. t = 0;
4. **for** (**int** j = 0; j < n; j++)**if** (i & (1 << j))t++;//记录国王数
5. num[++s0] = t;
6. s[s0] = i;//记录状态
7. }

**洛谷2704：**记录可行态同1896，这里为了节约时间，在记录可行状态的同时将第一行的状态进行枚举。

代码如下：

1. **for** (**register** **int** i = 0; i < (1 << m); i++) {
2. **if** (i & (i << 1) || i & (i << 2) ||(i>>1)&i||(i>>2)&i)**continue**;
3. **int** t = 0;
4. **for** (**register** **int** j = 0; j < m; j++)**if** ((1 << j) & i)t++;
5. s[++s0] = i;
6. num[s0] = t;
7. **if** ((mp[1]&s[s0])==s[s0])dp[1][s0][1] = num[s0];
8. }

### 5.4.2 典型例题

**5.4.2.1典型例题1 互不侵犯（洛谷1896）**

**题目描述**

在N×N的棋盘里面放K个国王，使他们互不攻击，共有多少种摆放方案。国王能攻击到它上下左右，以及左上左下右上右下八个方向上附近的各一个格子，共8个格子。

注：数据有加强（2018/4/25）

**输入格式**

只有一行，包含两个数N，K （ 1 <=N <=9, 0 <= K <= N \* N）

**输出格式**

所得的方案数

**思路**

看到这道题很容易联想到八皇后问题，因此我们想用搜索来实现，但是不同于八皇后，这题剪枝条件少，而且用传统的线性dp难以表示其状态，如果用线性dp则需要开4维，而且难以确定转移方程，因此状态压缩在这里起到了重大的作用。我们设想将每行的状态作为子结构，将每行的状态转移。用一个二进制数来表示每一行的状态，如（1010）b，表示取1、3位置放国王，而2、4位置不放国王，要确定这一行的可行状态，必须知道上一行的可行状态，因此就得出了状态转移过程中需要传递的量：当前的行数、国王摆放的状态和国王摆放的个数。因此我们开dp[i][j][k]来表示第i行，该行国王摆放状态为j，已经放置国王数量为k的方案数，

dp[i][j][k]=

其中s0表示一行中国王的摆放方案总数，t枚举上一行的可行摆放状态num[t]表示在t状态下这一行摆放的国王数。

为了节约时间，我们需要在程序执行前枚举出所有的一行摆放方案，枚举可以用搜索，这里直接用状态压缩枚举所有状态，筛选出符合条件的摆放方案状态储存在数组s中，并且将该状态摆放的国王数储存在数组num中。

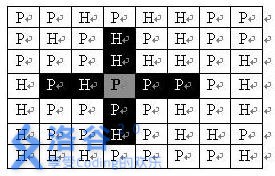
**AC代码如下：**

1. **long** **long** dp[11][200][200];
2. **int** num[200];
3. **int** s0 = 0;
4. **long** **long** s[200];
5. **int** main() {
6. **int** n, k;
7. **int** t;
8. **long** **long** ans = 0;
9. cin >> n >> k;
10. **for** (**int** i = 0; i < (1 << n); i++) {
11. **if** (i & (i << 1))**continue**;
12. t = 0;
13. **for** (**int** j = 0; j < n; j++)**if** (i & (1 << j))t++;
14. num[++s0] = t;
15. s[s0] = i;
16. }
17. **for** (**int** i = 1; i <= s0; i++)dp[1][i][num[i]] = 1;
18. //或者直接dp[0][1][0]=1;（即前0行放了0个国王的方案数）下面的for循环从1开始
19. **for** (**int** i = 2; i <= n; i++) {
20. **for** (**int** j = 1; j <= s0; j++) {
21. **for** (**int** m = 0; m <= k; m++) {
22. **if** (m >= num[j])**for** (**int** x = 1; x <= s0; x++) {
23. **if** ((!(s[j] & s[x])) && (!(s[j] & (s[x] << 1))) && (!((s[j] << 1) & s[x])))dp[i][j][m] += dp[i - 1][x][m - num[j]];
24. }
25. }
26. }
27. }
28. **for** (**int** i = 1; i <= s0; i++)ans += dp[n][i][k];
29. cout << ans << endl;
30. **return** 0;
31. }

**5.4.2.2典型例题2 洛谷2704**

**题目描述**

司令部的将军们打算在 N×M 的网格地图上部署他们的炮兵部队。一个 N×M的地图由 N 行 M 列组成，地图的每一格可能是山地（用H 表示），也可能是平原（用P 表示），如下图。在每一格平原地形上最多可以布置一支炮兵部队（山地上不能够部署炮兵部队）；一支炮兵部队在地图上的攻击范围如图中黑色区域所示：



如果在地图中的灰色所标识的平原上部署一支炮兵部队，则图中的黑色的网格表示它能够攻击到的区域：沿横向左右各两格，沿纵向上下各两格。图上其它白色网格均攻击不到。从图上可见炮兵的攻击范围不受地形的影响。现在，将军们规划如何部署炮兵部队，在防止误伤的前提下（保证任何两支炮兵部队之间不能互相攻击，即任何一支炮兵部队都不在其他支炮兵部队的攻击范围内），在整个地图区域内最多能够摆放多少我军的炮兵部队。

**输入格式**

第一行包含两个由空格分割开的正整数，分别表示 N 和 M。

接下来的 N 行，每一行含有连续的 M 个字符，按顺序表示地图中每一行的数据。

**输出格式**

一行一个整数，表示最多能摆放的炮兵部队的数量。

**思路：**

这题的思路几乎和1896一样，不一样的地方在于这题要确定当前状态必须知道前两行的状态，因此我们在这里需要转移当前行数，当前状态和上一行的状态，所以我们令dp[i][j][k]为第i行，当前状态为j，上一行状态为k时放的最多炮手数，很容易得出状态转移方程为：

dp[i][j][k]=max{dp[i-1][j][t]+num[j]}

和1896一样，我们需要对状态进行预处理，从而达到省时省空间的作用，如果不进行预处理则需要用滚动数组进行优化，否则会MLE。当然，预处理之后我们依然可以用滚动数组继续优化空间，这里就不给出写法了。

这里提一下，为什么不需要在枚举上一个状态时判断上一个状态是否满足地图要求，因为如果我们枚举的状态不满足地图要求，那么在将状态转移到我们枚举的这个状态时程序判断当前状态不符合地图要求，自动跳过了接下来的处理，因此这个状态的值为0，不会影响接下来的判断。

**AC代码如下：**

1. **int** dp[105][105][105];
2. **int** s[105],s0 = 0, n, m, num[105], mp[105];
3. **int** main() {
4. **char** s1;
5. cin >> n >> m;
6. **for** (**register** **int** i = 1; i <= n; i++) {
7. **for**(**register** **int** j = 1; j <= m; j++) {
8. cin >> s1;
9. mp[i] = (mp[i] << 1) + (s1 == 'P');
10. }
11. }
12. **for** (**register** **int** i = 0; i < (1 << m); i++) {
13. **if** (i & (i << 1) || i & (i << 2) ||(i>>1)&i||(i>>2)&i)**continue**;
14. **int** t = 0;
15. **for** (**register** **int** j = 0; j < m; j++)**if** ((1 << j) & i)t++;
16. s[++s0] = i;
17. num[s0] = t;
18. **if** ((mp[1]&s[s0])==s[s0])dp[1][s0][1] = num[s0];
19. }
20. **for** (**register** **int** i = 1; i <= s0; i++)
21. **for** (**register** **int** j = 1; j <= s0; j++)
22. **if** (!(s[i] & s[j]) && (mp[2] & s[i]) == s[i]) dp[2][i][j] = dp[1][j][1] + num[i];
23. **for** (**register** **int** i = 3; i <= n; i++) {
24. **for** (**register** **int** j = 1; j <= s0; j++) {
25. **if** ((mp[i]&s[j]) == s[j])**for** (**register** **int** k = 1; k <= s0; k++) {
26. **if** (!(s[j] & s[k]))**for** (**register** **int** ll = 1; ll <= s0; ll++) {
27. **if** ((!(s[j] & s[k])) && (!(s[j] & s[ll])) && (!(s[k] & s[ll])))dp[i][j][k] = max(dp[i][j][k], dp[i - 1][k][ll]+ num[j]) ;
28. }
29. }
30. }
31. }
32. **int** ans = 0;
33. **for** (**register** **int** j = 1; j <= s0; j++) {
34. **for** (**register** **int** k = 1; k <= s0; k++) {
35. ans = max(ans, dp[n][j][k]);
36. }
37. }
38. cout << ans << endl;
39. **return** 0;
40. }

**5.4.2.3 典型例题3 洛谷3694**

**题目背景**

BanG Dream!里的所有偶像乐队要一起大合唱，不过在排队上出了一些问题。

**题目描述**

N个偶像排成一列，他们来自M个不同的乐队。每个团队至少有一个偶像。

现在要求重新安排队列，使来自同一乐队的偶像连续的站在一起。重新安排的办法是，让若干偶像出列（剩下的偶像不动），然后让出列的偶像一个个归队到原来的空位，归队的位置任意。

请问最少让多少偶像出列？

**输入格式**

第一行2个整数N，M。

接下来N个行，每行一个整数a[i](1≤*ai*​≤*M*)，表示队列中第i个偶像的团队编号。

**输出格式**

一个整数，表示答案

**思路**

这道题输入的人数比较多，用枚举肯定超时，显然也不能用状态压缩枚举每个人。但是这道题的特点是乐团数比较少，考虑对乐团进行状态压缩。

如何将已经拍好的队伍压缩在一起呢，我们发现，只要队伍相邻就可以表示在一起，而且也无需考虑排列方式。这样后面的状态就跟已经排好的队伍有关了。为了方便，我们设想将已经拍好的队伍放在队伍最前面。我们用一个二进制数来表示乐团的排列状态，比如1011就是1、3、4乐团已经排列好了并且放在前面。令dp[i]表示排列状态为i的最少出列人数，假设现在在排第j队伍那么这个状态已经拍了n个队伍，前一个状态就是排了n-1个队伍的结果。dp[i]的结果就是上一个状态+第j队伍的总人数-在目前排列区间内的人数，即转移方程为：

**dp[i] = min(dp[i], dp[i ^ (1 << (j - 1))] + num[j] - sum[j][len] + sum[j][len - num[j]])**

其中len表示当前排了j队伍后已经排列了len个元素，num[j]表示j队伍总人数。

在程序开始时需要初始化dp数组为最大值，dp[0]=0;

**AC代码如下：**

1. **int** sum[21][101000];
2. **int** num[21];
3. **int** dp[1 << 20];
4. **int** n, m;
5. **int** main() {
6. cin >> n >> m;
7. **int** t;
8. **for** (**int** i = 1; i <= n; i++) {
9. cin >> t;
10. **for** (**int** j = 1; j <= m; j++) {
11. sum[j][i] = sum[j][i - 1];
12. }
13. sum[t][i]++;
14. num[t]++;
15. }
16. memset(dp, 0x3f, **sizeof**(dp));
17. dp[0] = 0;
18. **for** (**int** i = 0; i < (1 << m); i++) {
19. **int** len = 0;
20. **for** (**int** j = 0; j < m; j++)**if** ((1 << j) & i)len += num[j + 1];
21. **for** (**int** j = 1; j <= m; j++)**if**((1 << (j - 1)) & i)dp[i] = min(dp[i], dp[i ^ (1 << (j - 1))] + num[j] - sum[j][len] + sum[j][len - num[j]]);
22. }
23. cout << dp[(1 << m) - 1] << endl;
24. **return** 0;
25. }

**5.4.2.4典型例题4 洛谷P1441砝码称重**

**题目描述**

现有n个砝码，重量分别为 ai，在去掉 m个砝码后，问最多能称量出多少不同的重量（不包括 00）。

请注意，砝码只能放在其中一边。

**输入格式**

第 1 行为有两个整数 n 和 m，用空格分隔。

第 2 行有 n个正整数 a1, a2, a3,…,an，表示每个砝码的重量。

**输出格式**

仅包括 1 个整数，为最多能称量出的重量数量。

**输入**

3 1

1 2 2

**输出**

3

**思路**

这题是比较明显的01背包，我们需要枚举当前的状态，和一般的状态压缩一样0表示去掉，1表示选取，如果去掉的数等于m，则对当前进行01背包，这里的dp数组储存当前的质量能否称出来。我们令f[j]表示质量为j的物品能否被称出来，当前取到的砝码为i，质量为lib[i]，那么当且仅当质量为j-lib[i]的物品能被称出来时，质量为j的物品也能被称出来。这里只提供主要代码。

**AC代码如下：**

1. **int** lib[50];
2. **bool** a[50];
3. **bool** f[2001];
4. **int** main() {
5. **int** n, m;
6. **int** sum = 0;
7. **int** ans = 0;
8. cin >> n >> m;
9. **for** (**int** i = 1; i <= n; i++) {
10. cin >> lib[i];
11. sum += lib[i];
12. }
13. **for** (**int** i = 1; i < (1 << n); i++) {
14. **int** t = 0;
15. memset(a, 0, **sizeof**(a));
16. **for** (**int** j = 1; j <= n; j++) {
17. a[j] = (1 << (j - 1)) & i;
18. **if** (a[j])t++;
19. }
20. **if** (t != n - m)**continue**;
21. **int** temp = 0;
22. f[0] = 1;
23. **for** (**int** i = 1; i <= n; i++) {
24. **if** (a[i])**for** (**int** j = sum; j >= lib[i]; j--) {
25. **if** (f[j - lib[i]]&&!f[j]) {
26. f[j] = 1;
27. temp++;
28. }
29. }
30. }
31. memset(f, 0, **sizeof**(f));
32. ans = max(ans, temp);
33. }
34. cout << ans << endl;
35. **return** 0;
36. }

**5.4.2.5典型例题5洛谷P1879[USACO06NOV]Corn Fields G**

**题目描述**

Farmer John has purchased a lush new rectangular pasture composed of M by N (1 ≤ M ≤ 12; 1 ≤ N ≤ 12) square parcels. He wants to grow some yummy corn for the cows on a number of squares. Regrettably, some of the squares are infertile and can't be planted. Canny FJ knows that the cows dislike eating close to each other, so when choosing which squares to plant, he avoids choosing squares that are adjacent; no two chosen squares share an edge. He has not yet made the final choice as to which squares to plant.

Being a very open-minded man, Farmer John wants to consider all possible options for how to choose the squares for planting. He is so open-minded that he considers choosing no squares as a valid option! Please help Farmer John determine the number of ways he can choose the squares to plant.

**题目大意**

农场主John新买了一块长方形的新牧场，这块牧场被划分成M行N列(1 ≤ M ≤ 12; 1 ≤ N ≤ 12)，每一格都是一块正方形的土地。John打算在牧场上的某几格里种上美味的草，供他的奶牛们享用。

遗憾的是，有些土地相当贫瘠，不能用来种草。并且，奶牛们喜欢独占一块草地的感觉，于是John不会选择两块相邻的土地，也就是说，没有哪两块草地有公共边。

John想知道，如果不考虑草地的总块数，那么，一共有多少种种植方案可供他选择？（当然，把新牧场完全荒废也是一种方案）

**输入格式**

第一行：两个整数M和N，用空格隔开。

第2到第M+1行：每行包含N个用空格隔开的整数，描述了每块土地的状态。第i+1行描述了第i行的土地，所有整数均为0或1，是1的话，表示这块土地足够肥沃，0则表示这块土地不适合种草。

**输出格式**

一个整数，即牧场分配总方案数除以100,000,000的余数。

**思路**

这道题思路上跟例1例2一样，首先可以考虑暴力搜索或者dp，搜索缺少优化条件，在数据教大时非常容易超时，而且从题目数据可知搜索情况会非常多，因此考虑dp。

由于数据规模非常小，因此我们考虑状压dp，和例1例2一样，我们需要枚举当前状态，从前往后转移当前记录的种类。同样的，为了节约时间和空间，我们通过预处理，枚举出所有合法状态，为了方便比较，我们开1维数组来储存每一行的地图（同例2）。通过这些条件，我们定义dp[i][j]为第i行的状态为j时种草地的方案数，类比例1，我们很容易得出转移方程为：当(s[j]&mp[i])==s[j]时，

**dp[i][j]=Σdp[i-1][k],s[j]&s[k]==0**

**这里直接给出AC代码：**

1. **int** dp[20][(1 << 14) + 5];
2. **const** **int** mod = 100000000;
3. **int** mp[20];
4. **int** s[1 << 14];
5. **int** s0 = 0;
6. **int** n, m;
7. **int** ans;
8. **int** t;
10. **int** main() {
11. cin >> m >> n;
12. **for** (**int** i = 0; i < (1 << n); i++) {
13. **if** ((!(i & (i >> 1))) && (!(i & (i << 1))))s[++s0] = i;
14. }
16. **for** (**int** i = 1; i <= m; i++) {
17. **for** (**int** j = 1; j <= n; j++) {
18. cin >> t;
19. mp[i] = (mp[i] << 1) + t;
20. }
21. }
22. **for** (**int** i = 1; i <= s0; i++)**if** ((s[i] & mp[1]) == s[i])dp[1][i] = 1;
23. **for** (**int** i = 2; i <= m; i++) {
24. **for** (**int** j = 1; j <= s0; j++) {
25. **if**((s[j] & mp[i])==s[j])**for** (**int** k = 1; k <= s0; k++) {
26. **if** (!(s[j] & s[k]))dp[i][j] = (dp[i][j]%mod + dp[i - 1][k]%mod) %mod;
27. }
28. }
29. }
30. **for** (**int** i = 1; i <= s0; i++)ans = (ans % mod + dp[m][i] % mod) % mod;
31. cout << ans << endl;
32. **return** 0;
33. }

**5.4.2.6 典型例题5 洛谷P3092 [USACO13NOV]No Change G**

**题目描述**

Farmer John is at the market to purchase supplies for his farm. He has in his pocket K coins (1 <= K <= 16), each with value in the range 1..100,000,000. FJ would like to make a sequence of N purchases (1 <= N <= 100,000), where the ith purchase costs c(i) units of money (1 <= c(i) <= 10,000). As he makes this sequence of purchases, he can periodically stop and pay, with a single coin, for all the purchases made since his last payment (of course, the single coin he uses must be large enough to pay for all of these). Unfortunately, the vendors at the market are completely out of change, so whenever FJ uses a coin that is larger than the amount of money he owes, he sadly receives no changes in return!

Please compute the maximum amount of money FJ can end up with after making his N purchases in sequence. Output -1 if it is impossible for FJ to make all of his purchases.

约翰到商场购物，他的钱包里有K(1 <= K <= 16)个硬币，面值的范围是1..100,000,000。

约翰想按顺序买 N个物品(1 <= N <= 100,000)，第i个物品需要花费c(i)块钱，(1 <= c(i) <= 10,000)。

在依次进行的购买N个物品的过程中，约翰可以随时停下来付款，每次付款只用一个硬币，支付购买的内容是从上一次支付后开始到现在的这些所有物品（前提是该硬币足以支付这些物品的费用）。不幸的是，商场的收银机坏了，如果约翰支付的硬币面值大于所需的费用，他不会得到任何找零。

请计算出在购买完N个物品后，约翰最多剩下多少钱。如果无法完成购买，输出-1

**输入格式**

Line 1: Two integers, K and N.

Lines 2..1+K: Each line contains the amount of money of one of FJ's coins.

Lines 2+K..1+N+K: These N lines contain the costs of FJ's intended purchases.

**输出格式**

Line 1: The maximum amount of money FJ can end up with, or -1 if FJ cannot complete all of his purchases.

**思路**

这道题初见看到只有20个硬币，考虑搜索。但是搜索的复杂度过大，由于物品数量过多，使得搜索必然超时。因此考虑状压dp。从数据规模可以很容易想到枚举硬币的选择状态。由题目可知，每次支付只使用一硬币，因此转移的方式就是当前状态少选择1个硬币的状态。这题不像上几题可以预处理状态，因为每个状态都是可行态。我们令dp数组储存当前已经购买物品的数量，为了节约时间，还需要开一个f数组来存储用了多少钱（这样可以防止每次找到可行的状态后都重新处理当前状态计算用了多少钱）。容易得出转移方程为：

**dp[i]=max(dp[x]+num[j])**(x:i状态的上一个状态，j:现在处理的硬币下标，num[j]:用了j硬币后最多再购买num[j]个物品),**f[i]=f[x]+c[j]**

那么现在的问题就是如何找到用来j硬币后最多再购买的物品数。一种简单的方法就是按顺序遍历，直到总价值超过j硬币的面额时返回，但是这样的时间复杂度为O(2^K\*N\*K)超过本题的时间限制，因此必须进行优化。

我们发现，物品必须按顺序购买，那么我们只需要用前缀和处理一下商品数组，然后用一次二分查找，只需要找到最后被购买元素的下标即可。时间复杂度变为O(2^K\*Klog(n)).调整边界条件即可通过本题。下面直接给出AC代码。

**AC代码如下：**

1. **int** ans = 0x7fffffff;
2. **int** dp[1 << 20];
3. **int** f[1 << 20];
4. **int** c[20];
5. **int** w[100005];
6. **int** sum[100005];
7. **int** n;
8. **int** k;
9. **int** t;
10. **int** sc;
12. **inline** **int** read() {
13. **int** x = 0;
14. **int** f = 1;
15. **char** ch = getchar();
16. **while** (ch != '-' && (ch < '0' || ch > '9'))ch = getchar();
17. **if** (ch == '-') {
18. f = -1;
19. ch = getchar();
20. }
21. **while** (ch >= 48 && ch <= '9') {
22. x = (x << 1) + (x << 3) + (ch ^ 48);
23. ch = getchar();
24. }
25. **return** x \* f;
26. }
28. **int** main() {
29. k = read();
30. n = read();
31. **for** (**int** i = 1; i <= k; i++) {
32. c[i] = read();
33. sc += c[i];
34. }
35. **for** (**int** i = 1; i <= n; i++) {
36. w[i] = read();
37. sum[i] = sum[i - 1] + w[i];
38. }
39. **for** (**int** i = 1; i < (1 << k); i++) {
40. **for** (**int** j = 1; j <= k; j++) {
41. **if** ((1 << (j - 1)) & i) {
42. **int** x = i ^ (1 << (j - 1));
43. **int** l = dp[x] + 1;
44. **int** r = n;
45. **int** mid;
46. **while** (l <= r) {
47. mid = (l + r) >> 1;
48. **if** (sum[mid] - sum[dp[x]] == c[j]) {
49. r = mid;
50. **break**;
51. }
52. **else** **if** (sum[mid] - sum[dp[x]] < c[j]) {
53. l = mid + 1;
54. }
55. **else** {
56. r = mid - 1;
57. }
58. }
59. **if** (r > dp[i]) {
60. dp[i] = r;
61. f[i] = f[x] + c[j];
62. **if** (dp[i] == n)ans = min(ans, f[i]);
63. }
64. }
65. }
66. }
67. **if** (sc - ans >= 0)cout << sc - ans << endl;
68. **else** cout << -1;
69. **return** 0;
70. }

**5.4.2.7 典型例题 HDU5418 Victor and World**

**题目大意**

一个无向图有n个点（n <= 16），m条边(m <= 100000,有重边)，从1号店出发，经过全部点后回到1号点，问路径最短是多少。

**输入**

输入含有多组数据。

第一行一个整数他，表示一共有t组数据。

每组数据第一行为两个正整数n，m。

接下来m行，每行3个整数u v w，表示u点到v点有一条边，长度为w。

保证1<=u,v<=n , 1<=w<=100。

**输出**

对于每组数据，输出一个整数，表示最短路径的长度。

**思路**

这道题是明显的最短路，由数据规模很容易就想到状压dp。先用Floyd求任意两点的最短路。然后考虑状态压缩。这道题点数比较少，考虑将当前去过点的状态作为子结构，因此，我们想到令dp[S]作为当前去过点状态为S是的最短路径，但是这道题要进行转移需要知道最后停留的点，需要添加最后停留的点到我们枚举的下一个点的距离来确定下一个状态的值，因此，只用点的经过状态来代表最优子结构是不够的，因为最后停留的点对最后的结果会产生影响。因此，我们令dp[S][i]为当点的经过状态为S，最后停留的点为i的最短路径的长度令下一个点的坐标为k，g[i][k]为i点与k点之间的最短距离，因此状态转移方程为：

**dp[S|(1<<(k–1))][i] = max(dp[S|(1<<(k-1))][i] , dp[S][i] + g[i][k])**

这道题需要注意初始化与边界条件，因为是求最短路，因此需要将初值设置成极大值0x3f3f3f3f，并且将每个点自己到自己的距离设置为0，以及dp[1][1]=0（因为从1）

**AC代码如下**

1. #include <bits/stdc++.h>
2. #define ll long long
3. **using** **namespace** std;
5. **const** **int** N = 20;
6. **const** **int** INF = 0x3f3f3f3f;
8. **int** t;
9. **int** n;
10. **int** m;
11. **int** g[N][N];
12. **int** dp[1 << N][N];
13. **int** u, v, w;
14. **int** ans = 0x7fffffff;

17. **int** main() {
18. ios::sync\_with\_stdio(0);
19. cin >> t;
20. **while** (t--) {
21. cin >> n >> m;
22. memset(g, 0x3f, **sizeof**(g));
23. memset(dp, 0x3f, **sizeof**(dp));
24. dp[1][1] = 0;
25. ans = 0x7fffffff;
26. **for** (**int** i = 1; i <= n; i++) {
27. g[i][i] = 0;
28. }
29. **for** (**int** i = 1; i <= m; i++) {
30. cin >> u >> v >> w;
31. **if** (g[u][v] > w) {
32. g[u][v] = w;
33. g[v][u] = w;
34. }
35. }
36. **for** (**int** k = 1; k <= n; k++) {
37. **for** (**int** i = 1; i <= n; i++) {
38. **for** (**int** j = 1; j <= n; j++) {
39. g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
40. }
41. }
42. }
43. **for** (**int** j = 1; j < (1 << n); j++) {
44. **if**(j&1)**for** (**int** i = 1; i <= n; i++) {
45. **if**((1 << (i - 1)) & j)**for** (**int** k = 1; k <= n; k++) {
46. **if** ((1 << (k - 1)) & j)**continue**;
47. **if**(g[i][k]!=INF)dp[j | (1 << (k - 1))][k] = min(dp[j | (1 << (k - 1))][k], dp[j][i] + g[i][k]);
48. }
49. }
50. }
51. **for** (**int** i = 1; i <= n; i++) {
52. ans = min(ans, dp[(1 << n) - 1][i] + g[i][1]);
53. }
54. cout << ans << endl;
55. }
56. **return** 0;
57. }

**5.4.2.8 典型例题 HDU5691**

**题目描述**

度度熊是他同时代中最伟大的数学家，一切数字都要听命于他。现在，又到了度度熊和他的数字仆人们玩排排坐游戏的时候了。游戏的规则十分简单，参与游戏的N个整数将会做成一排，他们将通过不断交换自己的位置，最终达到所有相邻两数乘积的和最大的目的，参与游戏的数字有整数也有负数。度度熊为了在他的数字仆人面前展现他的权威，他规定某些数字只能在坐固定的位置上，没有被度度熊限制的数字则可以自由地交换位置。

**输入**

第一行一个整数T，表示T组数据。

每组测试数据将以如下格式从标准输入读入：

N

a1 p1

a2 p2

:

aN pN

第一行，整数 N(1≤N≤16)，代表参与游戏的整数的个数。

从第二行到第 (N+1) 行，每行两个整数，ai(−10000≤ai≤10000)、pi(pi=−1 或 0≤pi<N)，以空格分割。ai代表参与游戏的数字的值，pi代表度度熊为该数字指定的位置，如果pi=−1，代表该数字的位置不被限制。度度熊保证不会为两个数字指定相同的位置。

**输出**

第一行输出："Case #i:"。i代表第i组测试数据。

第二行输出数字重新排列后最大的所有相邻两数乘积的和，即max{a1⋅a2+a2⋅a3+......+aN−1⋅aN}。

**样例输入**

2

6

-1 0

2 1

-3 2

4 3

-5 4

6 5

5

40 -1

50 -1

30 -1

20 -1

10 -1

**样例输出**

Case #1:

-70

Case #2:

4600

**思路**

这道题比较接近状压dp的模板题，首先要想到已经排列的数字放在最左端（类似的思路见典型例题3）。唯一的难点是处理一些固定点位的数与不固定点位的数。所以，有两种转移的方式：令dp[S][i]为已经排列的数字的状态为S时，最后一个排的是第i个数字的时候的醉倒值。

第一种是使得当前状态从上一个状态转移过来，即转移方程为

**dp[S][i] = min(dp[S][i] , dp[S ^ (1 << k))][j] +a [i] \* a[k]**

第二种方法就是将当前状态转移到下一个状态，即

**dp[S | (1 << k)][k] = max(dp[S | (1 << k)][k] , dp[S][i] + a[i] \* a[k]**

第一种方法边界条件与合法条件判断复杂且容易些错，因此使用第二种方法，边界条件的我们需要将dp[1][1]初始化为0（或者将dp[1<<i][i]初始化为0），其余元素初始化为-0x7fffffff(极小值)。由转移方程可知，k是下一个状态，所以要求i必须包含于S中，而且k不包含于集合中，k还必须满足指定位置要求（i不需要指定要求是因为已经被上一个状态指定过了，如果不能满足则不能转移）。

**AC代码如下**

1. #include <bits/stdc++.h>
2. #define ll long long
3. **using** **namespace** std;
4. **int** t;
5. **int** n;
6. ll bs;
7. ll a[20];
8. ll b[20];
9. ll dp[20][(1 << 20) + 5];
10. ll ans = 0;
11. ll INF;
12. **int** main() {
13. ios::sync\_with\_stdio(**false**);
14. cin >> t;
15. **int** z = t;
16. **for**(**int** z=1;z<=t;z++) {
17. cin >> n;
18. **for** (**int** i = 0; i < n; i++) {
19. **for** (**int** j = 0; j < 1 << n; j++) {
20. dp[i][j] = -0x7fffffff;
21. }
22. }
23. INF = dp[1][1];
25. **for** (**int** i = 0; i < n; i++) {
26. cin >> a[i] >> b[i];
27. }
28. **for** (**int** i = 0; i < n; i++) {
29. **if**(b[i] == -1 || b[i] == 0)dp[i][1 << i] = 0;
30. }
31. **for** (**int** j = 1; j <= 1 << n; j++) {
32. ll g = 0;
34. **for** (**int** i = 0; i < n; i++) {
35. **if** ((1 << i) & j)g++;
36. }
37. **for** (**int** i = 0; i < n; i++) {
38. **if**(dp[i][j] != INF)**for** (**int** k = 0; k < n; k++) {
40. **if** (j >> k & 1)**continue**;
41. **if** (b[k] != -1) {
42. **if** (b[k] != g)**continue**;
43. }
44. dp[k][j | (1 << k)] = max(dp[k][j | (1 << k)], dp[i][j] + a[i] \* a[k]);
45. }
46. }
47. }
48. ans = INF;
49. **for** (**int** i = 0; i < n; i++) {
50. ans = max(ans, dp[i][(1 << n) - 1]);
51. }
52. cout << "Case #" << z << ':' << endl;
53. cout << ans << endl;
54. }
55. **return** 0;
56. }

**5.4.2.9 集合划分**

**题目描述**

n个人，每个人手上有一个数ai。  
将这些人分成若干组，组没有编号，要求每组人手上的数字之和都是质数。  
求合法的分组方案数。

**输入**

第一行一个正整数T(1≤T≤5)，表示测试数据的组数。  
每组数据第一行一个正整数n(1≤n≤15)。  
每组数据第二行n个正整数a1,a2,…,an(1≤ai≤100)。

**输出**

每组数据输出一行一个整数，即合法的分组方案数。

**样例输入**

1

3

3 2 5

**样例输出**

3

**思路**

这道题时比较明显的状压dp，1表示取了这一号人，0表示不取，然后我们枚举取的队伍数，当前的状态，上一个状态，我们发现即使使用了状态与处理也依然超时，算了一下时间，时间复杂度为O（n\*2^n\*s0），s0表示可行状态数，即最差的时间复杂度为n4^n,因为t为5，所以TLE了，因此考虑优化。先考虑如何不枚举当前队伍数，直接枚举状态，那么需要取当前状态的可行子集，取子集的方法为T=(T-1)&S,为了避免重复计数，我们令取的子集必须包含lowbit(S),因此转移方程就出来了，时间复杂度优化到了O(3^n)，在能接受的范围内。

**AC代码如下**

1. #include <bits/stdc++.h>
3. #define ll long long
4. #define int long long
6. **const** ll mod = 1000000009;
8. **using** **namespace** std;
10. **const** **int** N = 1505000;
12. **int** t;
13. **int** n;
14. **int** visit[N];
15. **int** ok[1 << 20];
16. **int** dp[1 << 20];
17. **int** a[N];
19. **signed** main() {
20. visit[0] = 1;
21. visit[1] = 1;
22. **for** (**int** i = 2; i <= N; i++) {
23. **if**(!visit[i])**for** (**int** j = i \* i; j <= N; j += i) {
24. visit[j] = 1;
25. }
26. }
27. cin >> t;
28. **while** (t--) {
29. cin >> n;
30. **for** (**int** i = 1; i <= n; i++) {
31. cin >> a[i];
32. }
33. memset(ok, 0, **sizeof**(ok));
34. memset(dp, 0, **sizeof**(dp));
35. **for** (**int** j = 1; j < ((ll)1 << n); j++) {
36. **int** sum = 0;
37. **for** (**int** i = 1; i <= n; i++) {
38. **if** (((ll)1 << (i - (ll)1)) & j)sum += a[i];
39. }
40. **if** (!visit[sum])ok[j] = 1;
41. }
42. dp[0] = 1;
43. **for** (**int** S = 1; S < ((ll)1 << n); S++) {
44. **int** x = S & -S;
45. **for** (**int** T = S; T; T = (T - 1) & S) {
46. **if** ((T & x) && ok[T]) {
47. dp[S] += dp[S - T];
48. }
49. }
50. }
51. cout << dp[((ll)1 << n) - 1] << endl;
52. }
53. }

## .5.5 数位DP

### 5.5.1 数位dp简介与一般做法

**简介：**在做题的过程中，时常会碰到这样一种题，要求统计合法数据，数据范围n给的特别大（比如1e18），因此O（n）的算法都会超时，要统计的数很多，而且这种题目也通常看不出i和i+1的联系。通常这种题目可以按10进制位进行统计，这样可以大大减少统计的次数。这种统计数字的方法称为数位dp。

**实现原理：**数位dp对每一位上的数字进行枚举，使得最后能不重不漏地枚举在范围内的每一个数。**一般的枚举办法（以1234为例）：**先枚举最高位，因为最大的数为1234，所以最高位只能是0或者1，当最高位为0时（即最高位小于所给定的最大值时），下面所有位的数字都可以取0-9中的任意一个数字（即可以取**0**000-**0**999）。但当最高位取到1时（即最高位取的数），枚举的下一位就只能最大取2（即下一位的最大值），和最高位同理，当这一位没有取到最大值2时，下一位就可以取任意的数字。枚举过程即当枚举的数字在本位没有取到给定的最大值时，它的下一位开始就可以取0-9之间的任意数字（题目要求排除的数字除外）。

**枚举过程模拟：（加粗的是当前枚举位为最的值的下一位）**

**0**000-**0**999

1**0**00-1**1**99

12**0**0**-**12**2**9

123**0-**123**3**

1234

**一般做法：**数位dp可以使用状态转移方程进行转移，最后统计两端的数的个数即可知道答案。但是dp的转移方程会根据题目的不同而不同，因此，模式化的数位dp做法是使用记忆化dfs实现的。记忆化的条件因题目而异，需要根据题面的描述即可判断怎样的状态是取到这一位当前状态的值可以是相等的。以windy数为例，当前置的数字相同时，该状态的的windy数的个数必然是相等的（如14??和24??的windy数个数必然是一样的），这题还需要判断当前位的上一位是否为前导0，如果是前导0则不能记录。而HDU4507则是到当前为为止的每一位数的和模7相等，并且现在记录的数模7的值相等时，与7无关的数的个数的值必然相等，和与平方和取模之后的值相等。

**一般的记忆化DFS实现代码（以windy数的过程为例）：**

1. #define ll long long
2. ll dfs(**int** pos, ll pre, **bool** ismax,**bool** is0) {
3. **if** (pos == 0)**return** 1;
4. **if** (dp[pos][pre] != -1 && !ismax && !is0)**return** dp[pos][pre];
5. **int** maxi = ismax ? arr[pos] : 9;
6. **int** temp = 0;
7. **for** (**int** i = 0; i <= maxi; i++) {
8. **if** (abs(pre - i) < 2 && !is0 )**continue**;
9. temp += dfs(pos - 1, i, ismax && i == maxi,is0 && i == 0);
10. }
11. **if** (!ismax && !is0)dp[pos][pre] = temp;
12. **return** temp;
13. }

**代码解释：**

pos：当前位的下标，即从左往右数第pos位

pre：上一位枚举的数值（当我们枚举第一位的数时，上一位是0）

ismax：当前位是否枚举的是数值上限的结果

is0：上一位是否为前到0

### 5.5.2 典型例题

**5.5.2.1典型例题1 洛谷P2602数字计数**

**题目描述**

给定两个正整数 a*a* 和 b*b*，求在 [a,b][*a*,*b*] 中的所有整数中，每个数码(digit)各出现了多少次。

**输入格式**

仅包含一行两个整数 a,b*a*,*b*，含义如上所述。

**输出格式**

包含一行十个整数，分别表示 0\sim 90∼9 在 [a,b][*a*,*b*] 中出现了多少次。

**输入**

1 99

**输出**

9 20 20 20 20 20 20 20 20 20

**思路**

这道题从数据规模就可以看出直接打表或者统计是TLE的，因此很容易想到数位dp，因为需要统计0-9每个数出现的次数，最简单的办法就是进行10次数位dp，输出10此的结果即可。本题由样例就可以看出必须处理前导0（因为前导0不算0的个数），我们可以知道当前位数一样时出现的次数相等，然后直接套数位dp的板子。这时我们会发现wa了，仔细分析后，发现取掉基于化后的结果跟原来的结果并不相等，因此发现记忆化的状态设计并不准确，这是可以发现即使后面的未枚举的数位出现我们统计的数ti的此数一样，但是当前已经枚举古的位数ti出现的此数也会影响最后的结果（因为会在枚举完成时记录当前枚举的数出现ti的次数），所以，记录的状态改为记录枚举第pos-1位时，ti已经出现了sum次时，后面的ti的个数。

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define ll long long
4. **const** **int** N = 90;
5. ll a[N];
6. ll dp[N][N][N];
7. ll num(ll x) {
8. **int** pos = 0;
9. **while** (x) {
10. a[++pos] = x % 10;
11. x /= 10;
12. }
13. **return** pos;
14. }
15. ll dfs(**int** pos, **int** q0, **bool** ismax, **int** sum,**int** ti) {
16. **if** (!ismax && dp[ti][pos][sum] != -1)**return** dp[ti][pos][sum];
17. **if** (pos == 0)**return** sum;
18. **int** maxi = ismax ? a[pos] : 9;
19. ll temp = 0;
20. **for** (**int** i = 0; i <= maxi; i++) {
21. temp += dfs(pos - 1, q0 || i, ismax && i == maxi, sum + ((i || q0) && i == ti), ti);
22. }
23. **if** (!ismax && q0)dp[ti][pos][sum] = temp;
24. **return** temp;
25. }
26. **int** main() {
27. ll n, m;
28. memset(dp, -1, **sizeof**(dp));
29. cin >> n >> m;
30. memset(a, 0, **sizeof**(a));
31. **int** len1 = num(n - 1);
32. ll ans1[15];
33. ll ans2[15];
34. memset(ans1, 0, **sizeof**(ans1));
35. memset(ans2, 0, **sizeof**(ans2));
36. **for** (**int** i = 0; i <= 9; i++) {
37. ans1[i] = dfs(len1, 0, 1, 0,i);
38. memset(dp, -1, **sizeof**(dp));
39. }
40. memset(a, 0, **sizeof**(a));
41. **int** len2 = num(m);
42. **for** (**int** i = 0; i <= 9; i++) {
43. ans2[i] = dfs(len2, 0, 1, 0, i);
44. memset(dp, -1, **sizeof**(dp));
45. }
46. **for** (**int** i = 0; i <= 9; i++) {
47. cout << ans2[i] - ans1[i] << ' ';
48. }
49. cout << endl;
50. **return** 0;
51. }

**5.5.2.2典型例题2 HDU4507吉哥系列故事——恨7不成妻**

**问题描述**

单身!依然单身！吉哥依然单身！DS级码农吉哥依然单身！所以，他生平最恨情人节，不管是214还是77，他都讨厌！

吉哥观察了214和77这两个数，发现：  
　　2+1+4=7  
　　7+7=7\*2  
　　77=7\*11  
最终，他发现原来这一切归根到底都是因为和7有关！所以，他现在甚至讨厌一切和7有关的数！什么样的数和7有关呢？如果一个整数符合下面3个条件之一，那么我们就说这个整数和7有关——  
　　1、整数中某一位是7；  
　　2、整数的每一位加起来的和是7的整数倍；  
　　3、这个整数是7的整数倍；  
现在问题来了：吉哥想知道在一定区间内和7无关的数字的平方和。

**输入**

输入数据的第一行是case数T(1 <= T <= 50)，然后接下来的T行表示T个case;每个case在一行内包含两个正整数L, R(1 <= L <= R <= 10^18)。

**输出**

请计算[L,R]中和7无关的数字的平方和，并将结果对10^9 + 7 求模后输出。

**样例输入**

3

1 9

10 11

17 17

**样例输出**

236

221

0

**思路**

这道题数据非常大，首先考虑数位dp，一般的数位dp肯定考虑要先求符合条件的数字的个数，统计与7无关的数的个数还是比较简单的，就是直接套数位dp的板子。这道题的难点就是如何求平方和。这道题需要用到平方和的性质，即。以231为例，，然后31又可以继续往下拆，因此，我们得出结论：当我们枚举到第pos位时，第pos位为i时，这个数pos位再往下的平方和为:

Sum(i-1)的意思就是第i-1为往下的数字的总值，因此我们还要记录当前位往下的数字值。这个记录就比较容易了，就是当前这位数字\*pow（10，pos）加上往后已记录的位数即可。接下来考虑记忆化，怎样的数字结果会一样。仔细看一下题目，整个题目都是表述与7有关的信息，可以猜想当前记录的数%7值一样时，取到的结果可能是一样的，再编写程序进行调试，发现确实是这样的，实际上，题目里还说要考虑每一位的数字和%7也不能为0，所以我们还要考虑每一位的数字和，因此dp数组记录的状态就确定了：dp[pos][sum][num]表示当前取到第pos位，每一位数字和为sum，已经记录的数字前len-pos位的结果为num时，记录取到的个数、下面的数字和、平方和的取值（用结构体储存）。实际上，调试时还可以发现当前pos位每位数字和%7的值相等时，记录的状态是一样的。所以dp数组可以改为：dp[pos][sum%7][num%7]。下面直接给出AC代码。

**AC代码如下：**

1. #include <bits/stdc++.h>
2. #define ll long long
3. #define int long long
4. **const** ll mod = 1e9 + 7;
5. **using** **namespace** std;
6. **class** node {
7. **public**:
8. ll cnt, sum, qsum;
9. };
10. **int** t;
11. ll p[100];
12. ll n, m;
13. ll a[100];
14. node f1[100][1000][10];
15. node bg;
16. node dfs(**int** pos, ll sum, ll num, **bool** ismax) {
17. node x = bg;
18. **if** (pos == 0) {
19. x.cnt = ((sum % 7) != 0) && (num % 7 != 0);
20. x.sum = 0;
21. x.qsum = 0;
22. **return** x;
23. }
24. **if** (!ismax && f1[pos][sum % 7][num % 7].cnt != -1)**return** f1[pos][sum % 7][num % 7];
25. **int** maxi = ismax ? a[pos] : 9;
26. node temp;
27. temp.cnt = 0;
28. temp.sum = 0;
29. temp.qsum = 0;
30. **for** (**int** i = 0; i <= maxi; i++) {
31. **if** (i == 7)**continue**;
32. node z = dfs(pos - 1, (sum + i) % 7, (num \* 10 + i) % 7, ismax && i == maxi);
33. temp.cnt += z.cnt % mod;
34. temp.cnt %= mod;
35. temp.sum += (z.cnt \* i) % mod \* p[pos - 1] % mod + z.sum % mod;
36. temp.sum %= mod;
37. temp.qsum += z.qsum + (((2 \* i \* p[pos - 1]) % mod) \* (z.sum % mod)) % mod + (z.cnt \* i \* i) % mod \* (p[pos - 1] \* p[pos - 1] % mod) % mod;
38. temp.qsum %= mod;
39. }
40. **if** (!ismax)f1[pos][sum % 7][num % 7] = temp;
41. **return** temp;
42. }
43. ll slove(ll x) {
44. **int** pos = 0;
45. **while** (x) {
46. a[++pos] = x % 10;
47. x /= 10;
48. }
49. **return** dfs(pos, 0,0, 1).qsum;
50. }
51. **signed** main()
52. {
53. ll x = 1;
54. memset(f1, -1, **sizeof**(f1));
55. **for** (**int** i = 0; i <= 20; i++) {
56. p[i] = x % mod;
57. x \*= 10;
58. }
59. scanf("%lld", &t);
60. **while**(t--) {
61. scanf("%lld%lld", &n, &m);
62. memset(f1, -1, **sizeof**(f1));
63. **int** t1 = slove(m);
64. memset(f1, -1, **sizeof**(f1));
65. **int** t2 = slove(n - 1);
66. cout << (t1 - t2 + mod) % mod << endl;
67. **int** ans = 0;
68. }
69. }

**5.5.2.3.典型例题3 洛谷P2518计数**

**题目描述**

你有一组非零数字（不一定唯一），你可以在其中插入任意个0，这样就可以产生无限个数。比如说给定{1,2},那么可以生成数字12,21,102,120,201,210,1002,1020,等等。

现在给定一个数，问在这个数之前有多少个数。（注意这个数不会有前导0）.

**输入格式**

只有1行，为1个整数n.

**输出格式**

只有整数，表示N之前出现的数的个数。

**输入**

1020

**输出**

7

**说明/提示**

n的长度不超过50，答案不超过2^63-1.

**思路**

这道题n可能有50位，因此必须用字符串输入数字，处理的时候转换成\_\_int64。相比数位dp的板子题，这道题的特点就是并不是所有数字都能取到的，而且取的次数也有限制，因为dfs有回溯功能，因此记忆化dfs实现起来比递推容易了不少。但是，怎样的状态才是要记录的？很明显，当每个数字取的次数已知，这个数字可能的个数也就确定了。但是如果这样记录数字的话，dp数组就需要开11维，内存远远超出限制。因此我们需要换一种策略。我们发现，当前面取的个数总和确定之后，后面的数字的总数就是有重复全排列的总数，因此可以套用有重复全排列的计算公式。但是这题数字特别大，计算n！的时候可能会爆long long，题解有个思想就特别巧妙：开一个数组a[i]，该数组的下标表示i所记录的质因数为i，比如a[2]的值表示最后的答案有几个2相乘。对每个因数或者除数进行分解质因数，需要乘这个数时该数分解出来的质因数个数全部+1，否则-1，最后将记录的所有质因数乘回来就是最后的答案。最后附上AC代码。

**AC代码如下：**

1. #include <bits/stdc++.h>
2. #define ll long long
3. #define int long long
4. **const** ll mod = 1e9 + 7;
5. **using** **namespace** std;
6. **const** **int** N = 60;
7. ll n;
8. **char** s[N];
9. ll a[N];
10. ll counti[N];
11. ll tempi[N];
12. **const** ll p[20] = { 0,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47 };
13. ll len;
14. ll tp[N];
15. ll sumcount;
16. **void** che(ll x) {
17. **int** i = 1;
18. **while** (i <= 15 && x != 1) {
19. **while** (x % p[i] == 0 && x != 1) {
20. x /= p[i];
21. tp[i]++;
22. }
23. i++;
24. }
25. }
26. **void** chu(ll x) {
27. **int** i = 1;
28. **while** (i <= 15 && x != 1) {
29. **while** (x % p[i] == 0 && x != 1) {
30. x /= p[i];
31. tp[i]--;
32. }
33. i++;
34. }
35. }
37. ll remember(**int** pos, ll num) {
38. ll temp = 0;
39. **if** (pos < sumcount - num)**return** 0;
40. memset(tp, 0, **sizeof**(tp));
41. **for** (**int** i = 1; i <= pos; i++) {
42. che(i);
43. }
44. **for** (**int** i = 1; i <= 9; i++) {
45. **if** (counti[i])**for** (**int** j = 1; j <= counti[i] - tempi[i]; j++) {
46. chu(j);
47. }
48. **if** (counti[i])temp += counti[i] - tempi[i];
49. }
50. **for** (**int** i = 1; i <= pos - temp; i++) {
51. chu(i);
52. }
53. ll ans = 1;
54. **for** (**int** i = 1; i <= 15; i++) {
55. **for** (**int** j = 1; j <= tp[i]; j++)ans \*= p[i];
56. }
57. **return** ans;
58. }
59. ll dfs(**int** pos,ll num, ll sum, **bool** ismax) {
60. **if** (pos == 0) {
61. **for** (**int** i = 1; i <= 9; i++) {
62. **if** (tempi[i] > counti[i])**return** 0;
63. }
64. **return** num == sumcount;
65. }
66. **if** (!ismax)**return** remember(pos, num);
67. ll temp = 0;
68. **int** maxi = ismax ? a[pos] : 9;
69. **for** (**int** i = 0; i <= maxi; i++) {
70. **if** (tempi[i] >= counti[i] && i)**continue**;
71. tempi[i]++;
72. temp += dfs(pos - 1, num + (i > 0), sum + i, ismax && i == maxi);
73. tempi[i]--;
74. }
75. **return** temp;
76. }
77. ll solve() {
78. **int** pos = 0;
79. len = strlen(s);
80. **while** (pos < len) {
81. pos++;
82. a[len - pos + 1] = (ll)((ll)s[pos - 1] - (ll)'0');
83. **if** (a[len - pos + 1]) {
84. counti[a[len - pos + 1]]++;
85. }
86. }
87. counti[0] = 0x7fffffff;
88. **for** (**int** i = 1; i <= 9; i++) {
89. sumcount += counti[i];
90. }
91. **return** dfs(pos, 0, 0, 1);
92. }
94. **signed** main() {
95. cin >> s;
96. cout << solve() - (ll)1 << endl;
97. }

**5.5.2.4 典型例题4 2020icpc济南站L**

**题目描述**

令f(x)为x二进制为上1的个数。

Mianking有一个数组a[0]……a[m-1],他想知道有多少个x属于[0,L]，使得任意i属于[0,m-1]，f(x+i)mod 2 = a[i].

**输入描述**

本题共有t组测试数据。

第一行为一个整数t，表示一共t组数据。

对于每组数据，第一行两个整数m，L，第二行有m个整数，表示a[0]到a[m-1].

t<=1000

1 <= m <= 100

L <= 1e18

a[i]为0或者1

**输出描述**

对于每组数据输出一共整数，表示一共有几个x符合条件。

**输入样例**

3

3 10

0 1 0

1 1000

0

9 1000000

1 0 1 1 0 1 0 0 1

**输出样例**

2

501

41667

**思路**

一般看到求什么的数量，并且范围非常大，一般考虑数位dp。

这题跟普通的数位dp的区别主要有2个，一个是这题讨论的是一个数的二进制是否符合要求，而且显然如果按十进制位进行讨论是比较困难的。第二就是成立状态的判定，即判定一个数是否能对整个数组成立，1到100一共有7位2进制数，如果只判断后7位代表一共数的话会有进位的问题。

和一般的数位dp一样，我们需要先设计dp状态，很显然，二进制位1的个数的奇偶性肯定影响结果。然后我们考虑进位问题，进位影响一段连续的1，使得一段连续的1变成0，把第一个0变成1.因此，我们要记录从第8位开始连续1的个数的奇偶性。X+i的值影响最大的部分为前7位，我们可以将后7位的值作为状态记录，也可以暴力枚举前7位的值。下面的AC代码仅展示将后7位的值作为状态记录的写法。

然后，这道题就可以套用数位dp的通用模板来写了。

**下面给出AC代码**

1. #include<bits/stdc++.h>
2. #define ll long long
3. #define int long long
4. #define endl '\n'
5. **using** **namespace** std;
6. **const** **int** N = 5e6 + 5;
7. **const** **int** M = 2e3 + 5;
8. **const** ll mod = 998244353;
10. **void** io() {
11. ios::sync\_with\_stdio(0);
12. cin.tie(0);
13. cout.tie(0);
14. }
16. **int** t;
17. **int** m,L;
18. **int** a[105];
19. **int** num[100];
20. **int** dp[100][2][128][2];
21. **int** len;
23. **void** setnum(**int** L) {
24. **while** (L) {
25. num[++len] = L & 1;
26. L >>= 1;
27. }
28. }
30. **bool** numof1(**int** x) {
31. **int** ans = 0;
32. **while** (x) {
33. ans += (x & 1);
34. x >>= 1;
35. }
36. **return** ans & 1;
37. }
39. **int** isok(**int** x,**int** isji,**int** jin) {
40. **bool** f = 1;
41. **for** (**int** i = 0; i < m && f; i++) {
42. **if** (x + i < 128)f &= ((numof1(x + i) ^ isji) == a[i]);
43. **else** f &= (numof1(x + i) ^ isji ^ jin) == a[i];
44. }
45. **return** f;
46. }


50. **int** dfs(**int** pos, **int** qji, **int** state, **int** jin,**bool** ismax) {
51. **if** (pos == 0)**return** isok(state,qji,jin);
52. **if** (!ismax && dp[pos][qji][state][jin] != -1)**return** dp[pos][qji][state][jin];
53. **int** temp = 0;
54. **int** maxn = ismax ? num[pos] : 1;
55. **for** (**int** i = 0; i <= maxn; i++) {
56. **if**(pos > 7)temp += dfs(pos - 1, qji ^ i, (((state << 1) + i)& 127), i&(!jin), ismax && i == maxn);
57. **else** temp += dfs(pos - 1, qji, (((state << 1) + i) & 127), jin, ismax && i == maxn);
58. }
59. **if** (!ismax)dp[pos][qji][state][jin] = temp;
60. **return** temp;
61. }

64. **signed** main() {
65. io();
66. cin >> t;
67. **while** (t--) {
68. cin >> m >> L;
69. **for** (**int** i = 0; i < m; i++) {
70. cin >> a[i];
71. }
72. setnum(L);
73. memset(dp, -1, **sizeof**(dp));
74. cout << dfs(len, 0, 0, 0, 1) << endl;
75. len = 0;
76. }
77. **return** 0;
78. }

## 5.6 树与图上的dp

树与图上dp，指在树和图上进行dp，这种题目常常会结合图论法或者搜索一同考察。

**4.6.1 树上dp经典例题 洛谷P1352 没有上司的舞会**

**题目描述**

某大学有 n 个职员，编号为 1…*n*。

他们之间有从属关系，也就是说他们的关系就像一棵以校长为根的树，父结点就是子结点的直接上司。现在有个周年庆宴会，宴会每邀请来一个职员都会增加一定的快乐指数 r\_i*ri*​，但是呢，如果某个职员的直接上司来参加舞会了，那么这个职员就无论如何也不肯来参加舞会了。所以，请你编程计算，邀请哪些职员可以使快乐指数最大，求最大的快乐指数。

**输入格式**

输入的第一行是一个整数 n。

第 2 到第 (n + 1) 行，每行一个整数，第 (i+1)行的整数表示 i号职员的快乐指数 ri。

第 (n + 2) 到第 2n 行，每行输入一对整数 l, k，代表 k是 l的直接上司。

**输出格式**

输出一行一个整数代表最大的快乐指数。

**思路**

这道题是树上dp的模板题，首先肯定要用dfs跑一遍树，记录当第i号职员和他的下属的最大快乐值，由于他既可以去也可以不去，因此要记录两个状态，因此dp数组要开2维，我们令dp[i][0]为当第i号职员不去时，他和他的下属的快乐值总和，dp[i][1]为他去的时候，他和他的职员的快乐值总和，所以，转移方程为：

**下面给出主要代码**

1. **int** n , a[N] , f[N];
2. vector<**int**>e[N];
3. **int** dp[N][2];
4. **void** dfs(**int** x) {
5. dp[x][1] = a[x];
6. **for** (**int** i = 0; i < e[x].size(); i++) {
7. dfs(e[x][i]);
8. dp[x][0] += max(dp[e[x][i]][0], dp[e[x][i]][1]);
9. dp[x][1] += dp[e[x][i]][0];
10. }
11. }

**5.6.2 树上背包（有依赖的背包）CTSC1997选课**

**题目描述**

在大学里每个学生，为了达到一定的学分，必须从很多课程里选择一些课程来学习，在课程里有些课程必须在某些课程之前学习，如高等数学总是在其它课程之前学习。现在有 **N***N* 门功课，每门课有个学分，每门课有一门或没有直接先修课（若课程 a 是课程 b 的先修课即只有学完了课程 a，才能学习课程 b）。一个学生要从这些课程里选择 **M***M* 门课程学习，问他能获得的最大学分是多少？

**输入格式**

第一行有两个整数 N , M 用空格隔开。(1≤*N*≤300, 1≤*M*≤300 )

接下来的N行,第I+1行包含两个整数 ki​和 si, ki表示第I门课的直接先修课si​,表示第I门课的学分。若 ki=0 表示没有直接先修课（1≤*ki*​≤*N* , 1≤*si*​≤20）。

**输出格式**

只有一行，选 M*M* 门课程的最大得分。

**思路**

这道题跟背包问题中有依赖的背包一模一样，这里不再说明如何思考。唯一不一样的地方在于这道题是一个森林而有依赖的背包就只是一棵树，解决的办法也很简单，把所有的父亲节点合并到第0个节点上，再对0节点做一次有依赖的背包，dp[0][M]就是最后的答案。本题代码参考背包问题的有依赖的背包部分。

**5.6.3 图上dp例1 洛谷P1613 跑路**

**题目描述**

小A买了一个十分牛B的空间跑路器，每秒钟可以跑2^k千米（k是任意自然数）。小A的家到公司的路可以看做一个有向图，小A家为点1，公司为点n，每条边长度均为一千米。帮他算算，他最少需要几秒才能到公司。数据保证1到n至少有一条路径。

**输入格式**

第一行两个整数n，m，表示点的个数和边的个数。

接下来m行每行两个数字u，v，表示一条u到v的边。

**输出格式**

一行一个数字，表示到公司的最少秒数。

**输入**

4 4

1 1

1 2

2 3

3 4

**输出**

1

**思路**

这道题初见很容易以为是直接求最短路然后输出，这样肯定是错的，以为可能会存在一条更长的路但是这条路的长度刚好为2^k千米，结果实际上还是长一点的路用时短。那么我们考虑对最短路进行转化，我们可以令一个数值g[i][j][k]表示i点到j点存在一条长度为2^k的路，要确保i到j有这样一条路，必须满足下面两个条件中的一个：

1、i到j有长度刚好为2^k的边。

2、存在一个中间点k，满足i🡪k、k🡪j都存在至少一条长度刚好为2^(k-1)的路。

以上述条件进行类似Floyd的dp进行递推。如果存在k，使得g[i][j][k] = 1成立，我们就令i到j花费的最少时间为1.即再开一个数组G[i][j]，对所有符合上述条件的初始化为1，否则初始化为无穷，然后再对G数组跑一次最短路（这里可以使用Dijkstra或者SPFA，但是时间给的充足直接跑Floyd，毕竟Floyd还是方便）最后输出1🡪n花费的时间。

**AC代码如下**

1. #include<bits/stdc++.h>
2. #define ll long long
3. **using** **namespace** std;
4. **const** **int** N = 55;
5. **int** n, m;
6. **bool** G[N][N][100];
7. ll g[N][N];
8. **int** main() {
9. cin >> n >> m;
10. memset(g, 0x3f, **sizeof**(g));
11. **for** (**int** i = 1; i <= m; i++) {
12. **int** u, v;
13. cin >> u >> v;
14. G[u][v][0] = 1;
15. g[u][v] = 1;
16. }
17. **for** (**int** b = 0; b <= 64; b++) {
18. **for** (**int** k = 1; k <= n; k++) {
19. **for** (**int** i = 1; i <= n; i++) {
20. **for** (**int** j = 1; j <= n; j++) {
21. **if** (G[i][k][b] == 1 && G[k][j][b] == 1) {
22. G[i][j][b + 1] = 1;
23. g[i][j] = 1;
24. }
25. }
26. }
27. }
28. }
29. **for** (**int** k = 1; k <= n; k++) {
30. **for** (**int** i = 1; i <= n; i++) {
31. **for** (**int** j = 1; j <= n; j++) {
32. g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
33. }
34. }
35. }
36. cout << g[1][n] << endl;
37. **return** 0;
38. }

## 5.7 其他dp问题

### 5.7.1 插头dp (陈丹琦《基于连通性状态压缩的动态规划问题》中的例题，洛谷P5056，ACWing2934)

**题目描述**

给出 n×m 的方格，有些格子不能铺线，其它格子必须铺，形成一个闭合回路。问有多少种铺法？

**输入格式**

第一行，两个整数，分别代表 n,m。（2 <= n,m <= 12）

从第二行到第 (n+1) 行，每行有一个长度为m的只含 \* 和 . 的字符串，\* 表不能铺线，. 表必须铺。

**输出格式**

输出一行一个整数，表示总方案数。

**输入输出样例**

**输入 #1**

4 4

\*\*..

....

....

....

**输出 #1**

2

**输入 #2**

4 4

....

....

....

....

**输出 #2**

6

**思路**

插头dp是一种状压dp，在本题中，决策当前棋盘的状态，用一般的状态压缩是很难做到的，因为每个格子有6种状态，直接进行转移的话状态总数会非常多，单一行就会出现2e9种状态，会产生非常多的冗余状态，即使使用状态预处理也必然超时（在处理状态合法性的过程就已经超时了）。因此引入插头的概念。在格子里的线必然有2两条，我们可以认为一条是进入格子的线，一条是出去的线，这样可以把格子连接处看成插头。对于整张图，我们已经考虑了很多个格子，还有很多个格子没有考虑，考虑的格子和没考虑的格子之间会形成一条分隔线，我们称之为轮廓线。在轮廓线上可能存在插头，也可能不存在插头。我们可以依据轮廓线上的插头状态进行转移。如下图所示：

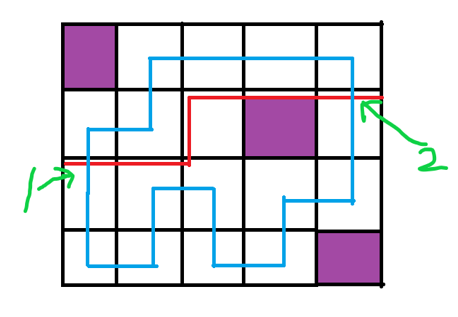


图5.7.1-1

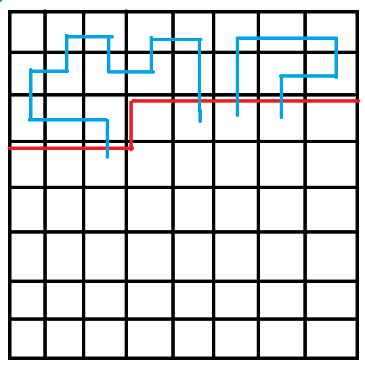
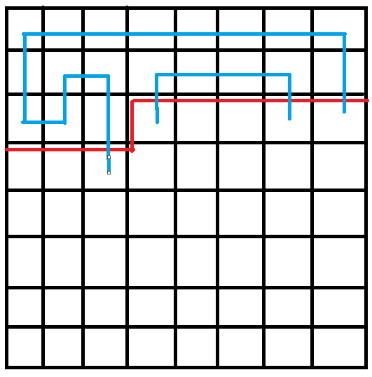
该图的轮廓线上一共有2个插头，这两个插头是相连的，我们令相连的两个插头是配对的。一般有两种方法表示轮廓线上插头的状态：1、最小表示法。2、括号配对法。最小表示法就是把第一组配对的括号标1，第二组配对的括号标2……以此类推。而括号表示法是最小表示法的一种特殊情况，即当不存在交叉的两组插头的情况下，可以用类似于括号匹配的方式来配对插头（即不存在{[}]的情况，所有情况必然是{[]}）,方法是用0表示没有括号，1表示左括号，2表示右括号。显然，括号表示法实现比最小表示法更省时，算法的时间复杂度会更小，因为只有3种状态，所以可以用3进制来表示轮廓线的状态，又因为位运算非常快，所以我们直接使用4进制来表示。不难看出，这题要求所有所有格子都铺上线，画图就可以看出不存在两对插头，使得铺的线相交（因为这样铺线一个格子就会有铺2跟线，图4.7.1-2，1-3，1-4表示了可行状态与不可行状态），因此可以用括号匹配法。

图5.7.1-3 该图轮廓线的插头状态为001010022，为可行状态



图5.7.1-2 该图轮廓线上的插头状态为001002120，为可行状态



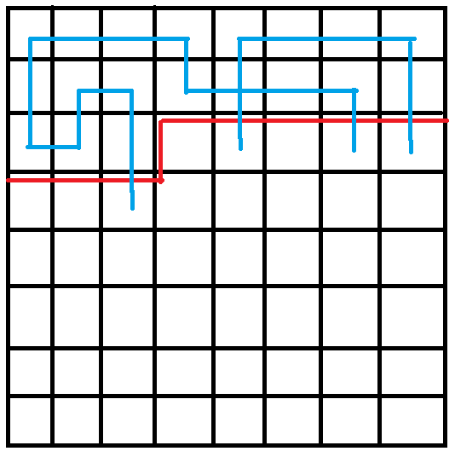
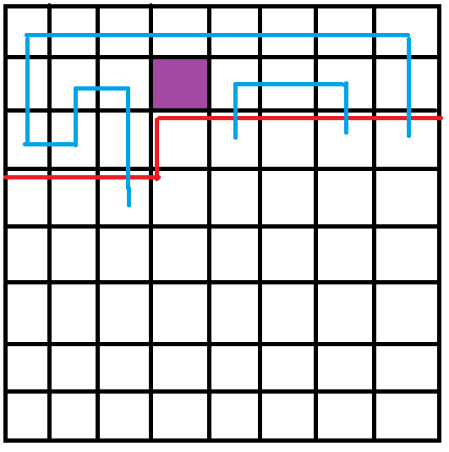




图5.7.1-5

图5.7.1-4 该图的轮廓线状态为001001022，但由于铺的线有交叉，所以并不是不是可行状态，如果出现该状态，将表示为图5.7.1-5的情况

确定思路后，我们开始考虑实现。分情况考虑状态转移。

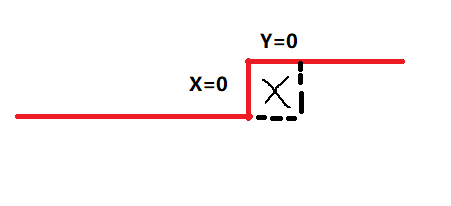
* 1. 当我们枚举的格子是障碍物时，必须满足当前格子边上的两条轮廓线不存在插头，保留当前的状态向下转移。（图4.7.1-6）

图5.7.1-6

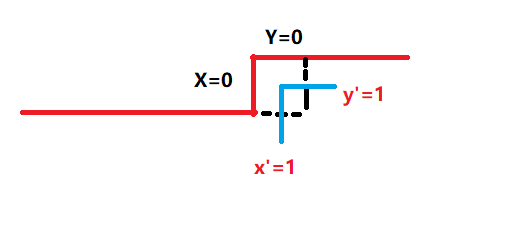
* 1. 当我们枚举的格子不是障碍物，并且x=0,y=0时，下一个状态的原x位置增加一个插头，原y位置也增加一个插头，且新增的两个插头相连，即x为y的左端点。（图4.7.1-7）

图5.7.1-7

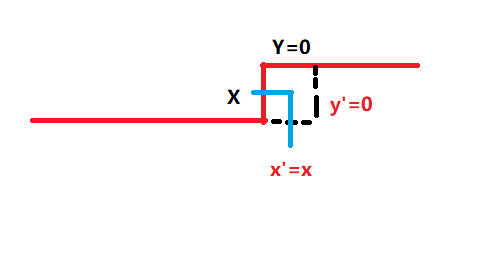
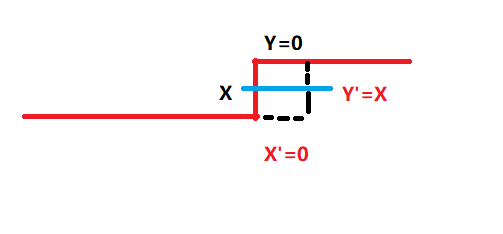
* 1. 当x！= 0，且y = 0时，有两种情况，第一种情况是x连着的线连到这个格子的下面（图4.7.1-8），这种情况状态不会改变，还有一种情况是x连着的线通向了格子右边（图4.7.1-9），这种情况x与y互换状态。这里需要注意的是，如果我们希望把线连到下一个状态的轮廓线，必须满足该线连着的下一个连着的格子不是障碍物，如4.7.1-8的情况要能成立，必须满足我们枚举的格子的正下方的格子不能是障碍物。

图5.7.1-9

图5.7.1-8

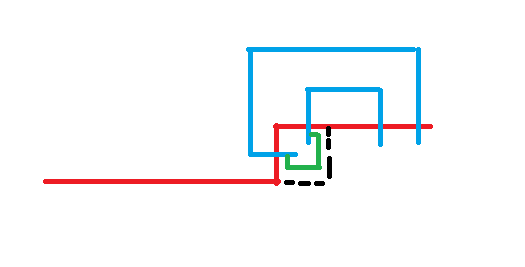
* 1. 当x=0，y！=0时，该情况与上一种情况类似，也分两种情况讨论。这里不再给出图解。
  2. 当x=1，y=1时，我们必须把x插头与y插头连在一起，组成通路。这种情况下，x与y均变成0，并且我们要将y插头连接的插头的值变为1（即变为左插头），如图4.7.1-10.

图5.7.1-10



* 1. 当x=2，y=2时，这种情况也需要连接x与y，和上一种情况类似，不再给出图解。
  2. 当x=2，y=1时，我们也需要连接x与y，连接后发现，这个状态仅仅删除了x和y的状态，其他插头的状态保持不变。如图4.7.1-11.

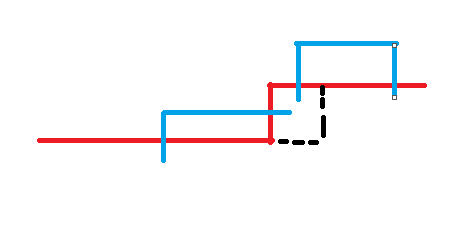


图5.7.1-10

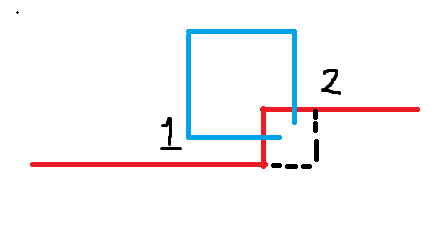
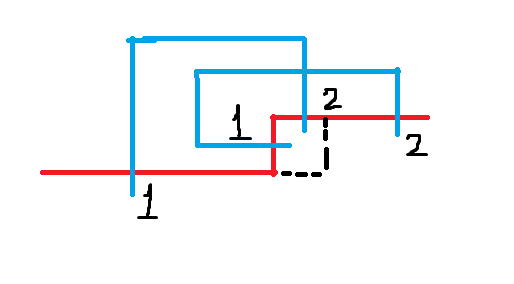
* 1. 当x=1，y=2时，由题意可知两条通路不能相交，因此不存在x连向y右边、y连向x左边的情况。唯一的可能就是将一条通路连成一条完整的回路，但是如果连成回路，意味着整个流程结束了，这时得到了枚举的结果，因此这种情况只会出现在最后一个格子中。当出现这种情况时，我们需要把之前传递的结果加到最后统计结果数的变量中即可。(图4.7.1-11为这种情况的可行状态，4.7.1-12为不可行状态)

图5.7.1-12

图5.7.1-11

列出所有情况后，我们开始代码实现，由于4^13非常大，无法用数组存下，因此使用哈希表来存。而且为了节约空间，我们使用滚动数组来存哈希表。哈希表存的是所有可行的状态，我们需要开一个数组v来存每个状态的出现的次数。还需要开一个数组来记录每个可行的方案所对应的哈希表的下标。

为了方便，我们定义find函数来查找某个状态在哈希表中的下标。如果哈希表中没有这个元素就找到一个空的位置返回。定义insert函数来插入状态，如果哈希表中已经存在该状态，就把该状态的出现此此数加上插入的个数，如果这个状态没有出现过，就在哈希表中插入这个状态，并把出现次数设置成w。为了方便构造新状态，我们定义get函数来获取state状态第k位的状态，set函数来构造一个第k位为v的四进制数。

在程序开始时，清空哈希表，并且在哈希表中插入无插头的状态（因为在第一个格子时轮廓线的状态必然为全0），这种情况的在第一行出现1次。

接下来我们考虑把一整行的状态转移到下一行。如图4.7.1-13和图4.7.1-14可以看出，上一行的结尾必然是0，而转移状态只会的状态的开头必然是0，并且中间n个状态完全一样。这里需要注意一下，为了储存方便，我们是把一格的状态放在状态数的结尾，即表示轮廓线状态的数字的最后一位对应着轮廓线上第一格的状态。因此在行之间转移状态时，我们希望将整个状态右移一位，并在状态第一个位置补0，我们就需要将哈希表中存的每一个可行状态数左移一位，在空出来的最右边一位补0。开始遍历下一行的格子前，清空当前行的哈希表，并且把当前行的状态数设置为0 。



图5.7.1-13

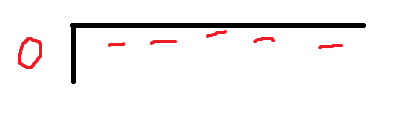


图5.7.1-14

我们先按二维坐标来遍历每一个格子，再遍历上一个的每个可行状态，对于每个可行的状态，获取当前格子的轮廓线状态，再根据上面给出的8种情况分别进行判断，并且将下一个可行状态插入哈希表中。当我们执行到最后一个格子，如果当前状态合法，就将方案数统计到一个变量中。程序执行结束时，输出这个变量。下来给出AC代码。

**AC代码如下**

1. #include<iostream>
2. #include<string.h>
3. #define ll long long
4. //#define int long long
5. **using** **namespace** std;
6. **const** **int** N = 5e4;
7. **const** **int** M = 5e5;
9. **int** n, m;
11. **int** g[20][20];
12. **int** h[2][M];
13. ll v[2][M];
14. **int** q[2][M];
15. **int** cnt[2];
16. **int** end\_x, end\_y;
18. **int** find(**int** cur ,**int** x) {
19. **int** t = x % N;
20. **while** (h[cur][t] != -1 && h[cur][t] != x) {
21. **if** (++t == N) {
22. t = 0;
23. }
24. }
25. **return** t;
26. }
28. **int** get(**int** state, **int** k) {
29. **return** (state >> (k \* 2)) & 3;
30. }
32. **int** set(**int** k, **int** v) {//构造第k位数
33. **return** v \* (1 << (k \* 2));
34. }
36. **void** insert(**int** cur, **int** state, ll w) {
37. **int** t = find(cur, state);
38. **if** (h[cur][t] == -1) {
39. h[cur][t] = state;
40. v[cur][t] = w;
41. q[cur][++cnt[cur]] = t;
42. }
43. **else** {
44. v[cur][t] += w;
45. }
46. }
48. **int** main() {
49. ios::sync\_with\_stdio(0);
50. cin >> n >> m;
51. **for** (**int** i = 1; i <= n; i++) {
52. **char** s[50];
53. cin >> s + 1;
54. **for** (**int** j = 1; j <= strlen(s + 1); j++) {
55. **if** (s[j] == '.') {
56. g[i][j] = 1;
57. end\_x = i;
58. end\_y = j;
59. }
60. }
61. }
62. memset(h, -1, **sizeof**(h));
63. ll res = 0;
64. //ll w = 0;
65. **int** cur = 0;
66. insert(cur, 0, 1);
67. **for** (**int** i = 1; i <= n; i++) {
68. **for** (**int** j = 1; j <= cnt[cur]; j++) {
69. h[cur][q[cur][j]] <<= 2;
70. }
71. **for** (**int** j = 1; j <= m; j++) {
72. **int** last = cur;
73. cur ^= 1;
74. cnt[cur] = 0;
75. memset(h[cur], -1, **sizeof**(h[cur]));
76. **for** (**int** k = 1; k <= cnt[last]; k++) {
77. **int** state = h[last][q[last][k]];
78. ll w = v[last][q[last][k]];
79. **int** x = get(state, j - 1);
80. **int** y = get(state, j);
81. //cout << x << ' ' << y << endl;
82. **if** (g[i][j] == 0) {
83. **if** (x == 0 && y == 0)insert(cur, state, w);
84. }
85. **else** **if** (x == 0 && y == 0) {
86. **if** (g[i + 1][j] && g[i][j + 1])insert(cur, state + set(j - 1, 1)+set(j ,2), w);
87. }
88. **else** **if** (x == 0 && y != 0) {
89. **if** (g[i][j + 1])insert(cur, state, w);
90. **if** (g[i + 1][j])insert(cur, state + set(j - 1, y) - set(j, y), w);
91. }
92. **else** **if** (x != 0 && y == 0) {
93. **if** (g[i][j + 1])insert(cur, state + set(j, x) - set(j - 1, x), w);
94. **if** (g[i + 1][j])insert(cur, state, w);
95. }
96. **else** **if** (x == 1 && y == 1) {
97. **int** s = 1;
98. **for** (**int** u = j + 1;; u++) {
99. **int** z = get(state, u);
100. **if** (z == 1)s++;
101. **else** **if**(z == 2){
102. s--;
103. **if** (s == 0) {
104. insert(cur, state - set(j - 1, x) - set(j, y) - set(u, 1), w);
105. **break**;
106. }
107. }
108. }
109. }
110. **else** **if** (x == 2 && y == 2) {
111. **int** s = 1;
112. **for** (**int** u = j - 2;; u--) {
113. **int** z = get(state, u);
114. **if** (z == 2)s++;
115. **else** **if**(z == 1){
116. s--;
117. **if** (s == 0) {
118. insert(cur, state - set(j - 1, x) - set(j, y) + set(u, 1), w);
119. **break**;
120. }
121. }
122. }
123. }
124. **else** **if** (x == 2 && y == 1) {
125. insert(cur, state - set(j - 1, x) - set(j, y), w);
126. }
127. **else** **if** (x == 1 && y == 2) {
128. **if** (i == end\_x && j == end\_y)
129. res += w;
130. }
131. }
132. }
133. }
134. cout << res << endl;
135. **return** 0;
136. }

**总结**

插头dp的题目还可以解决非联通性问题，只要找对了表示插头状态的方法，并且列出状态转移方程（就是上题的判断过程），虽然代码非常复杂，但是只要仔细写，遇到问题后通读代码，看看与思路是否有出入，如果没有问题就思考判断的逻辑是否有错，只要有耐心，问题就可以得到解决。

### 5.7.2 dp求方案数（组合数学+dp）2020icpc上海E

**题目大意**

求满足任意i>k，使得a[i]>min(a[i-k]……a[i-1])的所有全排列的个数，由于答案可能很大，只需要输出对998244353取模的结果。

**输入描述**

输入包含2个数n、k，表示一共n个数。

1 <= n,k <=1e7

**输出描述**

一个数，表示最后答案对998244353取模的结果

**示例1**

**输入**

1 1

**输出**

1

**示例2**

**输入**

2 3

**输出**

2

**示例3**

**输入**

3 2

**输出**

4

**示例4**

**输入**

4 2

**输出**

10

**思路**

求方案数的题目，一般考虑先放i个数，然后再放第i+1个数，求这个时候的方案数变化是设计转移方程的关键，但是这题有些不同，这题如果用这种思路来求解状态转移会非常复杂，因此需要改变思路。这道题有一个特点，就是最小的数必然出现在1->k中，我们假设第i个方案已经得出，再放入i+1个数的时候，我们可以考虑把已经排好的数+1，然后再枚举1的位置就可以了，当我们放好了1之后，我们发现，1前面的数是可以随便放的，而1后面的数需要满足题目要求，用数学的式子来表达就是 ， 很显然，这个式子的复杂度为O,不能满足题目要求，但是我们只需要把排列数拆开，可以得到的式子是,这里我们就可以看出可以对该式子进行前缀和优化，可以让转移的时间复杂度降到O.

**下面给出AC代码**

1. #include<bits/stdc++.h>
2. #define int long long
4. **using** **namespace** std;
6. **const** **int** N=1e7+5;
7. **const** **int** mod=998244353;

10. **int** fan[N];
11. **int** inv[N];
12. **int** dp[N];
13. **int** n,k;
15. **int** qpow(**int** a,**int** b){
16. **if**(b==0)**return** 1;
17. **if**(b==1)**return** a;
18. **if**(b&1)**return** a\*(qpow(a\*a%mod,b>>1)%mod)%mod;
19. **return** qpow(a\*a%mod,b>>1)%mod;
20. }
22. **void** init(){
23. fan[0]=1;
24. **for**(**int** i=1;i<=1e7;i++){
25. fan[i]=fan[i-1]\*i;
26. fan[i]%=mod;
27. }
28. inv[(**int**)1e7]=qpow(fan[(**int**)1e7],mod-2);
29. **for**(**int** i=1e7-1;i;i--){
30. inv[i]=inv[i+1]\*(i+1)%mod;
31. inv[i]%=mod;
32. }
33. }
34. **int** sum[N];
35. **signed** main()
36. {
37. ios::sync\_with\_stdio(0);
38. cin.tie(0);
39. cout.tie(0);
40. init();
41. cin>>n>>k;
42. dp[0]=1;
43. **for**(**int** i=1;i<=k;i++){
44. dp[i]=dp[i-1]\*i;
45. dp[i]%=mod;
46. sum[i]=sum[i-1]+dp[i]\*inv[i]%mod;
47. sum[i]%=mod;
48. }
49. **for**(**int** i=k+1;i<=n;i++){
50. dp[i]+=(sum[i-1]-sum[i-k-1]+mod)\*fan[i-1]%mod;
51. sum[i]=sum[i-1]+dp[i]\*inv[i]%mod;
52. sum[i]%=mod;
53. }
54. cout<<dp[n]<<endl;
55. **return** 0;
56. }

# ６．数据结构

## ６.１线段树

**６.1.1 普通线段树**

线段树将一个线性数据分区间维护，从而提高查找和修改的效率。其一般分为建树，修改和更新三个部分。

建树：

1. **void** build(**int** l,**int** r,**int** nw)
2. {
3. **if**(l==r)
4. {
5. scanf("%lld",&tree[nw].sum);
6. **return**;
7. }
8. **int** mid=(l+r)>>1;
9. build(l,mid,nw<<1);
10. build(mid+1,r,nw<<1|1);
11. tree[nw].sum=tree[nw<<1].sum+tree[nw<<1|1].sum;
12. }

**更新(区间和，平方和类型)：**

1. **void** update(**int** l,**int** r,**int** nw,**int** L,**int** R,ll k\_add,ll k\_mul)
2. {
3. **if**(l>=L&&r<=R)
4. {
5. tree[nw].Ad+=k\_add;
6. tree[nw].Mu\*=k\_mul;
7. tree[nw].Ad\*=k\_mul;
8. tree[nw].sum+=(k\_add\*(r-l+1));
9. tree[nw].sum\*=k\_mul;
10. tree[nw].sq\*=(k\_mul\*k\_mul);
11. tree[nw].sq=tree[nw].sq+2\*k\_add\*(tree[nw].sum-k\_add\*(r-l+1))+k\_add\*k\_add\*(r-l+1);
12. **return**;
13. }
14. **if**(l>r)**return**;
15. **if**(l>R||r<L)
16. {
17. **return**;
18. }
19. **if**(tree[nw].Ad||tree[nw].Mu>1)
20. {
21. push\_down(nw,l,r);
22. }
23. **int** mid=(l+r)>>1;
24. update(l,mid,nw<<1,L,R,k\_add,k\_mul);
25. update(mid+1,r,nw<<1|1,L,R,k\_add,k\_mul);
26. tree[nw].sum=tree[nw<<1].sum+tree[nw<<1|1].sum;
27. tree[nw].sq=tree[nw<<1].sq+tree[nw<<1|1].sq;
28. **return** ;
29. }

**更新（区间最大值类型）：**

1. **void** update(**int** l,**int** r,**int** nw,**int** L,**int** R,**int** k)
2. {
3. **if**(l>=L&&r<=R)
4. {
5. tree[nw].lazy+=k;
6. tree[nw].maxnum+=k;
7. **return**;
8. }
9. **if**(l>r)**return**;
10. **if**(l>R||r<L)
11. {
12. **return**;
13. }
14. **if**()
15. {
16. push\_down(nw,l,r);
17. }
18. **int** mid=(l+r)>>1;
19. update(l,mid,nw<<1,L,R,k\_add,k\_mul);
20. update(mid+1,r,nw<<1|1,L,R,k\_add,k\_mul);
21. tree[nw].maxnum=max(tree[nw<<1].maxnum,tree[nw<<1|1].maxnum);
22. }

**访问：**

1. ll query(**int** l,**int** r,**int** nw,**int** L,**int** R …………)
2. {
3. **if**(l>R||r<L)
4. {
5. **return** 0;
6. }
7. **if**(l>=L&&r<=R)
8. {
9. **return** tree[nw]. …………
10. }
11. **int** mid=(l+r)>>1;
12. **if**(tree[nw].Ad||tree[nw].Mu>1)
13. {
14. push\_down(nw,l,r);
15. }
16. **return** query(l,mid,nw<<1,L,R,…)+query(mid+1,r,nw<<1|1,L,R,…);   (return max::query(l,mid,nw<<1,L,R,…),query(mid+1,r,nw<<1|1,L,R,…))
17. }

PS：push\_down函数根据具体题目而异 分为 当前节点的子节点的值修改，lazy标记下传，当前节点lazy标记清空三个部分。

**6.1.2 区间平方和线段树**

由于要维护维护每个点的平方和， 当区间修改，比如增加x时 所以只要知道新加的值，所有点的和，原来的平方和，就可以计算出新的平方和。即维护sum（）和 sum（）即可。

**6.1.3 线段树应用--扫描线，以面积并为例**

线段树在扫描线算法中，线段树的作用是快速查询y轴的覆盖范围（假定我们扫描的基准是x轴。）

图表, 箱线图

描述已自动生成图3.1.3-1

对于第1块矩形，它的x轴范围是在这一段范围内，对于y轴覆盖增加一次，此时第一块矩形的面积就是x的范围和y的覆盖范围的乘积。来到第二块矩形，y轴的覆盖范围增加了的覆盖，此时，我们将修改后的y轴覆盖和第二块的x范围作积，便是面积。对于第三块矩形，这个范围的覆盖被减少了一次，所以，此时的y覆盖范围只剩下了，同理作积，求出第三块矩形的面积。

不难发现，每块小矩形的x轴可以通过排序得到，我们只要用线段树维护y轴覆盖即可，这里就要用到覆盖型线段树以及离散化了。

在数据读入时，记录下每一个y轴的坐标，对这些坐标离散化后，我们要求解的就是各个y坐标之间的覆盖情况，要注意的是，这时候如果以y轴的离散化坐标为线段树的分段依据，叶子节点的左端点比右端点要小1。由于矩形的特殊性，cover标记并不用下传。

1. #include<iostream>
2. #define y1 yx
3. **using** **namespace** std;
4. **struct** node
5. {
6. **double** y\_up,y\_down;
7. **int** io;**double** x;  //io记录是否是出边，入边
8. };
9. **struct** node\_t
10. {
11. **int** cover;**double** len,l,r;  //len表示当前段的贡献长度（覆盖长度）
12. };
13. node line[300];
14. **double** y[401];
15. **double** rk[401];
16. node\_t tree[10010];
17. **double** x1,yx,x2,y2;
18. **int** n;**int** cnt;
19. **bool** cmp(node x,node y)
20. {
21. **return** x.x<y.x;
22. }
23. **void** pushup(**int** l,**int** r,**int** nw)  //上传
24. {
25. **if**(tree[nw].cover)
26. {
27. tree[nw].len=rk[r]-rk[l];
28. }
29. **else** **if**(l+1==r)
30. {
31. tree[nw].len=0;
32. }
33. **else**
34. {
35. tree[nw].len=tree[nw<<1].len+tree[nw<<1|1].len;
36. }
37. }
38. **void** build(**int** l,**int** r,**int** nw)
39. {
40. **if**(l+1==r)
41. {
42. tree[nw].cover=0;
43. tree[nw].l=rk[l];
44. tree[nw].r=rk[r];
45. tree[nw].len=0.0;
46. pushup(l,r,nw);
47. **return**;
48. }
49. **int** mid=(l+r)>>1;
50. build(l,mid,nw<<1);
51. build(mid,r,nw<<1|1);
52. pushup(l,r,nw);
53. }
54. **void** up(**int** l,**int** r,**int** nw,**int** yl,**int** yr,**int** k)
55. {
56. **if**(yl>r||yr<l)
57. {
58. **return**;
59. }
60. **if**(yl<=l&&yr>=r)
61. {
62. tree[nw].cover+=k;  //由于覆盖操作是相互独立的，所以不用下传
63. pushup(l,r,nw);
64. }
65. **if**(l+1==r)**return**;
66. **int** mid=(l+r)>>1;
67. up(l,mid,nw<<1,yl,yr,k);
68. up(mid,r,nw<<1|1,yl,yr,k);
69. pushup(l,r,nw);
70. }
71. **int** main()
72. {
73. **int** \_t=0;
74. **while**(scanf("%d",&n),n)
75. {
76. \_t++;
77. cnt=0;
78. **for**(**int** i=1;i<=n;i++)
79. {
80. scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
81. line[++cnt].x=x1;
82. line[cnt].y\_down=y1;
83. line[cnt].y\_up=y2;
84. line[cnt].io=1;
85. y[cnt]=y1;
86. line[++cnt].x=x2;
87. line[cnt].y\_down=y1;
88. line[cnt].y\_up=y2;
89. line[cnt].io=-1;
90. y[cnt]=y2;
91. }
92. sort(line+1,line+1+cnt,cmp);
93. sort(y+1,y+1+cnt);
94. y[0]=-1;
95. **int** cnty=0;
96. **for**(**int** i=1;i<=cnt;i++)  //离散化y坐标
97. {
98. **if**(y[i-1]!=y[i])
99. {
100. rk[++cnty]=y[i];
101. }
102. }
103. build(1,cnty,1);**double** ans=0.0;
104. **for**(**int** i=1;i<=cnt;i++)
105. {
106. ans=ans+(line[i].x-line[i-1].x)\*tree[1].len; //当前y的覆盖
107. **int** yl=lower\_bound(rk+1,rk+1+cnty,line[i].y\_down)-rk;
108. **int** yr=lower\_bound(rk+1,rk+1+cnty,line[i].y\_up)-rk;
109. up(1,cnty,1,yl,yr,line[i].io);
110. }
111. printf("Test case #%d\nTotal explored area: %.2lf\n\n",\_t,ans);
112. }
113. }

**6.1.4 线段树区间染色问题**

区间染色问题也称区间覆盖问题，染色线段树在3.1.3中的扫描线算法上经常使用。其本质就是每次操作对一个区间L，R进行覆盖（每次覆盖的信息可以是相同的，亦可是不同的）。然后对具体覆盖信息进行询问操作。如果用数组实现这些操作，难免会造成时间的浪费，所以一般会使用线段树或者set（平衡树）解决这一类问题。这类问题的线段树在访问（查询）操作时，与其他线段树有所不同。

染色线段树的建树：如果当前线段树节点有不同的颜色覆盖，记为-1，没有颜色覆盖，记0，否则记录具体是那种颜色覆盖当前节点。

染色线段树特殊的访问操作：对询问区间q\_l,q\_r中所有的覆盖颜色进行遍历，对遍历到的颜色进行实时记录，最后看看遍历到了几种不同的颜色，就是访问区间q\_l,q\_r的颜色种类。询问过程中要对颜色进行下传操作。

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define pii pair<int,int>
4. **struct** node\_t
5. {
6. **int** cover, w;
7. };
8. **const** **int** M = 3e5 + 10;
9. **int** ans;
10. **int** cover[5 \* M];
11. **int** color[M];
12. **int** vis[M];
14. **void** up(**int** l, **int** r, **int** nw, **int** L, **int** R,**int** k)
15. {
16. **if** (l > R || r < L)**return**;
17. **if** (l >= L && r <= R&&cover[nw]!=-1)
18. {
19. color[cover[nw]] = 1;
20. cover[nw] = k;
21. **return**;
22. }
23. **if** (cover[nw] > 0)
24. {
25. color[cover[nw]] = 1;
26. color[cover[nw<<1]] = 1;
27. cover[nw << 1] = cover[nw];
28. color[cover[nw<<1|1]] = 1;
29. cover[nw << 1|1] = cover[nw];
30. }
32. **int** mid = (l + r) >> 1;
33. up(l, mid, nw << 1, L, R,k);
34. up(mid + 1, r, nw << 1 | 1, L, R,k);
35. **if** (cover[nw << 1] == cover[nw << 1 | 1])cover[nw] = cover[nw << 1];
36. **else** cover[nw] = -1;
37. }
38. **int** query(**int** l, **int** r, **int** nw, **int** L, **int** R, **int**& tmp)
39. {
40. **if** (l > R || r < L)
41. {
42. **return** 0;
43. }
44. **if** (l > r)**return** 0;
45. **if** (cover[nw] > 0)
46. {
47. **if** (vis[cover[nw]]!=tmp && color[cover[nw]]==0)
48. {
49. vis[cover[nw]] = tmp;
50. ans++;
51. **return** 1;
52. }
53. **else** **return** 0;
54. }
55. **if** (cover[nw] == 0)**return** 0;
56. **int** mid = (l + r) >> 1;
57. **return** query(l, mid, nw << 1, L, R, tmp) + query(mid + 1, r, nw << 1 | 1, L, R, tmp);
58. }
59. **int** main()
60. {
61. /\*freopen("E:\\Edata\\2.in","r",stdin);
62. freopen("E:\\Edata\\outfile", "w", stdout);\*/
63. **int** n; cin >> n;
64. **for** (**int** i = 1; i <= n; i++)
65. {
66. **int** q, l, r;
67. scanf("%d", &q);
68. **if** (q == 1)
69. {
70. scanf("%d%d", &l, &r);
72. **int** tmp = 0;
73. ans = 0;
74. query(1, M, 1, l, r, i);
75. printf("%d\n", ans);
76. up(1, M, 1, l, r, i);
78. }
79. **else**
80. {
81. **int** tmp = 0; ans = 0;
82. query(1, M, 1, 1, 250001, i); printf("%d\n", ans);
83. }
84. }
85. }

## 6.2 分块

**6.2.1 分块基本思想**

分块算法是将原有的线性数据分成诺干块，以减少数据数量，来达到方便处理的目的。一般情况下我们将数据分成块，n是数据个数。一般情况下，我们从0号元素开始，将每个元素分至第块，其中每个元素的定位pos=块，为了计算方便，我们做

式3.8.1-1

Pos= 式3.8.1-2

**6.2.2 分块的实现**

分块算法我们首先要将原数集与分块后进行相互映射，对于第i个数，它所在块是第几块的求法以及在3.8.1中给出。对于第j个块，它所包含的数位于St到Ed的范围内。其中第j块的与可以由下列方法以迭代法求出

式3.8.2-1

式3.8.2-2

分块完成后，可能需要数据进行修改或者查询，对于区间[L,R],我们先找到L和R分别所在的块（利用式子3.8.1-2）求出，然后分别对L所在的块与R所在的块进行零散操作，对L+1与R-1块进行整体操作（访问或修改）即可。

6.2.3 区间最值问题与区间求和问题

分块算法可以实现区间最值与区间求和问题的快速求解，其时间复杂度与线段树55开。

a)区间最值问题（Range max&minnum query）

对于区间最值问题，我们只要记录每一块的最值，单点修改时

**6.2.4 代码实现-区间众数**

题目描述：

给定一个序列a1,a2,…,an，m个询问。每个询问指定一个区间[l,r]，你需要输出al,al+1,…,ar这些数字里出现次数最多的数的出现次数。

思路分析

区间众数是分块算法的典序应用，由于题目只能在线，所以莫队算法无法实现。首先对题目所给序列进行分块。分块后，预处理出 表示i块到j块的众数数量，以及G()表示前j块中，数i出现的次数。对于每一次询问区间[l,r]，令l所在第L块，r所在第R块。求出[l,]，[]中的数在[l,r]中的出现次数（通过G()）高校求出，与进行比较，取大的输出。

1. #include<bits/stdc++.h>
2. **using**  **namespace** std;
3. #define ll long long
4. **const** **int** M=60007;
5. **int** Ed[M];
6. **int** St[M];
7. **struct** node
8. {
9. **int** num,pos;
10. };
11. **bool** cmp(node x,node y)
12. {
13. **return** x.num<y.num;
14. }
15. **const** **int** K=9;
16. **int** rk[M];
17. node a[M];
18. **int** f[300][300];
19. **int** G[M][300];
20. **int** n,q;
21. **int** cnt[M];
22. **int** solve(**int** l,**int** r)
23. {
24. **int** ans=0;
25. **int** L=(l-1)>>K;
26. **int** R=(r-1)>>K;
27. **if**(L+1>=R)
28. {
30. **for**(**int** i=l;i<=r;i++)
31. {
32. cnt[rk[i]]++;
33. ans=max(ans,cnt[rk[i]]);
34. }
35. **for**(**int** i=l;i<=r;i++)cnt[rk[i]]=0;
36. }
37. **else**
38. {
39. **for**(**int** i=l;i<=Ed[L];i++)
40. {
41. **if**(cnt[rk[i]]==0)
42. {
43. cnt[rk[i]]=G[rk[i]][R-1]-G[rk[i]][L];
44. }
45. cnt[rk[i]]++;
46. ans=max(ans,cnt[rk[i]]);
47. }
48. **for**(**int** i=St[R];i<=r;i++)
49. {
50. **if**(cnt[rk[i]]==0)
51. {
52. cnt[rk[i]]=G[rk[i]][R-1]-G[rk[i]][L];
53. }
54. cnt[rk[i]]++;
55. ans=max(ans,cnt[rk[i]]);
56. }
57. ans=max(ans,f[L+1][R-1]);
58. **for**(**int** i=l;i<=Ed[L];i++)
59. {
60. cnt[rk[i]]=0;
61. }
62. **for**(**int** i=St[R];i<=r;i++)
63. {
64. cnt[rk[i]]=0;
65. }
66. }
67. **return** ans;
68. }
69. **int** main()
70. {
71. **int** t;
72. scanf("%d",&t);
73. **while**(t--)
74. {
75. scanf("%d%d",&n,&q);
76. **int** m=(n-1)>>K;
77. **for**(**int** i=1;i<=n;i++)
78. {
79. scanf("%d",&a[i].num);
80. a[i].pos=i;
81. }
82. sort(a+1,a+1+n,cmp);
83. memset(G,0,**sizeof** G);
84. memset(f,0,**sizeof** f);
85. **int** number=0;
86. a[0].num=-1;
87. **for**(**int** i=1;i<=n;i++)
88. {
89. **if**(a[i].num!=a[i-1].num)number++;
90. rk[a[i].pos]=number;
91. }
92. **for**(**int** i=1;i<=n;i++)
93. {
94. Ed[(i-1)>>K]=i;
95. }
96. **for**(**int** i=n;i;i--)
97. {
98. St[(i-1)>>K]=i;
99. }
100. **int** maxn=0;
101. **for**(**int** i=0;i<=m;i++)
102. {
103. maxn=0;
104. **for**(**int** j=i;j<=m;j++)
105. {
106. **for**(**int** k=St[j];k<=Ed[j];k++)
107. {
108. cnt[rk[k]]++;
109. maxn=max(cnt[rk[k]],maxn);
110. }
111. f[i][j]=maxn;
112. }
113. **for**(**int** k=St[i];k<=n;k++)cnt[rk[k]]=0;
114. }
115. **for**(**int** i=0;i<=m;i++)
116. {
117. **if**(i!=0)
118. {
119. **for**(**int** j=1;j<=n;j++)G[j][i]=G[j][i-1];
120. }
121. **for**(**int** k=St[i];k<=Ed[i];k++)
122. {
123. G[rk[k]][i]++;
124. }
125. }
126. **int** previous\_ans=0;
127. **int** ans;
128. **for**(**int** \_=1;\_<=q;\_++)
129. {
130. **int** l,r;
131. scanf("%d%d",&l,&r);
132. l^=previous\_ans;
133. r^=previous\_ans;
134. **if**(l>r)
135. {
136. printf("0\n");
137. **continue**;
138. }
139. ans=solve(l,r);
140. printf("%d\n",ans);
141. previous\_ans=ans;
142. }
143. }
144. }

## 6.3 cdq分治

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** N=1e5+10;
4. **struct** node
5. {
6. **int** num,pos;
7. }a[N];
8. **int** n;
9. node tmp[N];
10. **int** f[N];
11. **void** divide(**int** l,**int** r)
12. {
13. **int** mid=(l+r)>>1;
14. **if**(l==r)**return**;
15. divide(l,mid);
16. sort(a+mid+1,a+r+1,[](node x,node y){**return** x.num<y.num;});
17. **int** p1=l,p2=mid+1,ans=0;
18. **for**(**int** i=l;i<=r;++i)
19. {
20. **if**(a[p1].num<a[p2].num&&p1<=mid)
21. {
22. ans=max(ans,f[a[p1].pos]);p1++;
23. }
24. **else** **if**(p2<=r)
25. {
26. f[a[p2].pos]=max(f[a[p2].pos],ans+1);p2++;
27. }
28. **else** **if**(p1>mid)
29. {
30. f[a[p2].pos]=max(f[a[p2].pos],ans+1);p2++;
31. }
32. }
33. sort(a+mid+1,a+r+1,[](node x,node y){**return** x.pos<y.pos;});
34. divide(mid+1,r);
35. p1=l,p2=mid+1,ans=0;
36. **for**(**int** i=l;i<=r;++i)
37. {
38. **if**(a[p1].num<a[p2].num&&p1<=mid)
39. {
40. tmp[i]=a[p1];p1++;
41. }
42. **else** **if**(p2<=r)
43. {
44. tmp[i]=a[p2];p2++;
45. }
46. **else** **if**(p1>mid)
47. {
48. tmp[i]=a[p2];p2++;
49. }
50. **else**
51. {
52. tmp[i]=a[p1];p1++;
53. }
54. }
55. **for**(**int** i=l;i<=r;++i)
56. {
57. a[i]=tmp[i];
58. }
59. }
60. **void** divide1(**int** l,**int** r)
61. {
62. **int** mid=(l+r)>>1;
63. **if**(l==r)**return**;
64. divide1(l,mid);
65. sort(a+mid+1,a+r+1,[](node x,node y){**return** x.num<y.num;});
66. **int** p1=l,p2=mid+1,ans=0;
67. **for**(**int** i=l;i<=r;++i)
68. {
69. **if**(a[p1].num<a[p2].num&&p1<=mid)
70. {
71. ans=max(ans,f[a[p1].pos]);p1++;
72. }
73. **else** **if**(p2<=r)
74. {
75. f[a[p2].pos]=max(f[a[p2].pos],ans+1);p2++;
76. }
77. **else** **if**(p1>mid)
78. {
79. f[a[p2].pos]=max(f[a[p2].pos],ans+1);p2++;
80. }
81. }
82. sort(a+mid+1,a+r+1,[](node x,node y){**return** x.pos>y.pos;});
83. divide1(mid+1,r);
84. p1=l,p2=mid+1,ans=0;
85. **for**(**int** i=l;i<=r;++i)
86. {
87. **if**(a[p1].num<a[p2].num&&p1<=mid)
88. {
89. tmp[i]=a[p1];p1++;
90. }
91. **else** **if**(p2<=r)
92. {
93. tmp[i]=a[p2];p2++;
94. }
95. **else** **if**(p1>mid)
96. {
97. tmp[i]=a[p2];p2++;
98. }
99. **else**
100. {
101. tmp[i]=a[p1];p1++;
102. }
103. }
104. **for**(**int** i=l;i<=r;++i)
105. {
106. a[i]=tmp[i];
107. }
108. }
109. **int** main()
110. {
111. n=0;
112. **for**(**int** i=1;i<=n;++i)f[i]=1;
113. **while**(cin>>a[++n].num)
114. {
115. a[n].pos=n;
116. }
117. // cout<<n<<"s";
118. n--;
119. **for**(**int** i=1;i<=n;++i)
120. {
121. f[i]=1;
122. }
123. divide(1,n);
124. **int** maxn=0;
125. **for**(**int** i=1;i<=n;++i)
126. {
127. maxn=max(maxn,f[i]);
128. }
129. sort(a+1,a+1+n,[](node x,node y){**return** x.pos>y.pos;});
130. **for**(**int** i=1;i<=n;++i)
131. {
132. f[i]=1;
133. }
134. **int** maxnT=0;
136. divide1(1,n);
137. **for**(**int** i=1;i<=n;++i)
138. {
139. maxnT=max(maxnT,f[i]);
140. }
141. cout<<maxnT<<'\n';
142. cout<<maxn<<'\n';
143. }

## ６.４莫队

**６.４.１ 普通离线莫队**

莫队是一种将在线访问的问题转化为离线处理的一种方法，其核心就是利用了访问区间的重叠部分不用重新计算的特点。对于每次访问的区间(l,r)，先将所有区间离线，先按照l分块排序，l在同一块的按照r排序。然后根据排完序后的序列进行操作，利用双指针不断调整的方式，解出每个询问所对于的答案。简言之就是两个sort四个while。

例题：yzcode 1027 区间众数统计（简单版）

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **struct** node
4. {
5. **int** l,r,pos,ans;
6. };
7. **const** **int** N=2e5+10;
8. **struct** lsh
9. {
10. **int** pos,num;
11. };
12. **int** ans;
13. **int** a[N];
14. lsh rk[N];
15. node q[N];
16. **int** pos[N];
17. **int** cnt[N]；
18. **int** n,m;
19. **int** block[N];
20. **bool** cmp(node x,node y)
21. {
22. **if**(block[x.l]==block[y.l])**return** x.r<y.r;
23. **return** block[x.l]<block[y.l];
24. }
25. **int** maxn;
26. **void** add(**int** x)
27. {
28. cnt[a[x]]++;
29. pos[cnt[a[x]]-1]--;
30. pos[cnt[a[x]]]++;
31. maxn=max(cnt[a[x]],maxn);
32. ans=maxn;
33. }
34. **void** del(**int** x)
35. {
37. pos[cnt[a[x]]-1]++;
38. pos[cnt[a[x]]]--;
39. cnt[a[x]]--;
40. **if**(pos[maxn]==0)
41. {
42. **for**(**int** i=maxn;i;i--)
43. {
44. **if**(pos[i]>0)
45. {
46. maxn=i;
47. **break**;
48. }
49. }
50. }
51. ans=maxn;
52. }
53. **signed** main()
54. {
55. // freopen("E:\\Edata\\1027data\\1.in","r",stdin);
56. // freopen("E:\\Edata\\1027data\\1.out","w",stdout);
57. cin>>n>>m;
58. **int** sqr=sqrt(n);
59. **for**(**int** i=1;i<=n;++i)
60. {
61. block[i]=(i-1)/sqr;
62. scanf("%d",&rk[i].num);
63. rk[i].pos=i;
64. }
65. **int** cntl=0;
66. sort(rk+1,rk+1+n,[](lsh x,lsh y){**return** x.num<y.num;});
67. **for**(**int** i=1;i<=n;++i)
68. {
69. **if**(rk[i].num!=rk[i-1].num)cntl++;
70. a[rk[i].pos]=cntl;
71. }
72. **for**(**int** i=1;i<=m;++i)
73. {
74. scanf("%d%d",&q[i].l,&q[i].r);
75. q[i].pos=i;
76. }
77. **int** l=1,r=0;ans=0;
78. sort(q+1,q+1+m,cmp);pos[0]=n;
79. **for**(**int** i=1;i<=m;++i)
80. {
81. **while**(r<q[i].r)add(++r);
82. **while**(r>q[i].r)del(r--);
83. **while**(l>q[i].l)add(--l);
84. **while**(l<q[i].l)del(l++);
85. q[i].ans=ans;
86. }
87. sort(q+1,q+1+m,[](node x,node y){**return** x.pos<y.pos;});
88. **for**(**int** i=1;i<=m;++i)
89. {
90. **if**(i==m)printf("%d",q[i].ans);
91. **else** printf("%d\n",q[i].ans);
92. }
93. }

**6.4.2 树上离线莫队**

树上莫队与普通的莫队没有什么太大的区别，我们需要将树上的数据转化成线性的数据进行操作，然后将询问离线，之后的操作与普通莫队无异，一个分块排序，四个循环。

例题：[A-树上莫队模板题\_牛客竞赛数据结构专题班分块、莫队 (nowcoder.com)](https://ac.nowcoder.com/acm/contest/20376/A)

图片包含 示意图

描述已自动生成

这道题利用了欧拉序，将树上问题转化成了线性的问题。（欧拉序：按照dfs入栈出栈顺序对每个节点进行编号，即入栈标号一次，出栈标号一次）

对于任意两个点（l，r）之间的路径：如果有一个点x，它的欧拉序位于（l,r）之间，并且之出现了一次，那么这个点在（l,r）路径上。并且，这两个点的LCA无论如何都会在（l,r）的路径上。（路径指简单路径）

这题在莫队算法的过程中，不断记录每个点在当前扩展区间中的出现次数，就能算出相应的出现了几种颜色。

**本题用到了倍增求 LCA法。**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** N=2e5+10;
4. **const** **int** M=10\*N;
5. **int** n,m;
6. std::vector<**int**>vt[N];
7. **int** rk[N],id[N],out[N],cnt,col[M],vis\_p[N],vis\_c[M],block[N],ans,f[M][34],dep[M];
8. **struct** node
9. {
10. **int** l,r,pos,ans;
11. };
12. node q[N];
13. **void** work(**int** x,**int** k)
14. {
15. vis\_p[id[x]]^=1;
16. **if**(vis\_p[id[x]]==1)
17. {
18. vis\_c[col[id[x]]]++;
19. **if**(vis\_c[col[id[x]]]==1)ans++;
20. }
21. **else**
22. {
23. vis\_c[col[id[x]]]--;
24. **if**(vis\_c[col[id[x]]]==0)ans--;
25. }
26. }
27. **void** dfs(**int** u,**int** fa)
28. {
29. rk[u]=++cnt;
30. id[cnt]=u;
31. dep[u]=dep[fa]+1;
32. f[u][0]=fa;
33. **for**(**int** i=1;i<=30;++i)
34. {
35. f[u][i]=f[f[u][i-1]][i-1];
36. }
37. **for**(auto v:vt[u])
38. {
39. **if**(v==fa)**continue**;
40. dfs(v,u);
41. }
42. out[u]=++cnt;
43. id[cnt]=u;
44. }
45. **bool** cmp(node x,node y)
46. {
47. **return** x.r<y.r;
48. }
49. **int** LCA(**int** x,**int** y)
50. {
51. **if**(dep[x]<dep[y])swap(x,y);
52. **for**(**int** i=30;i>=0;i--)
53. {
54. **if**(dep[f[x][i]]>=dep[y])
55. {
56. x=f[x][i];
57. }
58. }
59. **if**(x==y)**return** x;
60. **for**(**int** i=30;i>=0;i--)
61. {
62. **if**(f[x][i]!=f[y][i])
63. {
64. x=f[x][i];y=f[y][i];
65. }
66. }
67. **return** f[x][0];
68. }
69. **int** main()
70. {
71. cin>>n;
72. **int** sqr=sqrt(n);
73. **for**(**int** i=1;i<=n;++i)block[i]=(i-1)/sqr;
74. **for**(**int** i=1;i<=n;++i)scanf("%d",&col[i]);
75. **for**(**int** i=1;i<n;++i)
76. {
77. **int** u,v;
78. scanf("%d%d",&u,&v);
79. vt[u].push\_back(v);vt[v].push\_back(u);
80. }
81. dfs(1,0);
82. cin>>m;
83. **for**(**int** i=1;i<=m;++i)
84. {
85. scanf("%d%d",&q[i].l,&q[i].r);q[i].pos=i;
86. **if**(rk[q[i].l]>rk[q[i].r])swap(q[i].l,q[i].r);
87. }
88. **int** l,r;l=r=1;ans=1;vis\_c[col[id[1]]]=1;vis\_p[id[1]]=1;
89. **for**(**int** i=1;i<=m;++i)
90. {
91. **int** L=rk[q[i].l],R=rk[q[i].r];
92. **if**(out[q[i].r]>out[q[i].l])L=rk[q[i].l]+1;
93. **while**(r<R)work(++r,1);
94. **while**(r>R)work(r--,-1);
95. **while**(l<L)work(l++,-1);
96. **while**(l>L)work(--l,1);
97. **if**(out[q[i].r]<=out[q[i].l])q[i].ans=ans;
98. **else**
99. {
100. **int** lca=LCA(q[i].l,q[i].r);
101. **if**(vis\_c[col[lca]]==0)q[i].ans=ans+1;
102. **else** q[i].ans=ans;
103. }
104. }
105. sort(q+1,q+1+m,[](node x,node y){**return** x.pos<y.pos;});
106. **for**(**int** i=1;i<=m;++i)
107. {
108. printf("%d\n", q[i].ans);
109. }
110. }

**6.4.3 查询拆分**

通常，对于一些复杂的询问，可能有多个区间。这样，我们就不能利用莫队法来求解问题，此时，对于一些询问，我们可以进行拆分，拆分成一些简单的询问，以便于使用莫队查询（！！注意：此时空间需要开多倍！！）。

例题：SNOI 2017

文本

描述已自动生成

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** N=1e5+10;
4. **int** hs[2][N],a[N],ans,cntq,n,m,block[N];
5. **struct** node
6. {
7. **int** l,r,ans,pos;
8. }q[2\*N];
9. **bool** cmp1(node x,node y){
11. **if**(block[x.l]==block[y.l])**return** x.r<y.r;
12. **return** block[x.l]<block[y.l];
13. }
14. **void** add(**int** x,**int** p)
15. {
16. hs[p][a[x]]++;
17. ans+=hs[!p][a[x]];
18. }
19. **void** mus(**int** x,**int** p)
20. {
21. hs[p][a[x]]--;
22. ans-=hs[!p][a[x]];
23. }
24. **int** main()
25. {
26. cin>>n;**int** sqr=sqrt(n);
27. **for**(**int** i=1;i<=n;++i)
28. {
29. block[i]=(i-1)/sqr;
30. scanf("%d",&a[i]);
31. }
32. cin>>m;
33. **for**(**int** i=1;i<=m;i++)
34. {
35. **int** l1,r1,l2,r2;
36. scanf("%d%d%d%d",&l1,&r1,&l2,&r2);
37. q[++cntq].l=r1;q[cntq].r=r2;q[cntq].pos=cntq;
38. q[++cntq].l=r1;q[cntq].r=l2-1;q[cntq].pos=cntq;
39. q[++cntq].l=l1-1;q[cntq].r=r2;q[cntq].pos=cntq;
40. q[++cntq].l=l1-1;q[cntq].r=l2-1;q[cntq].pos=cntq;
41. }
42. sort(q+1,q+1+cntq,cmp1);
43. **int** l=0,r=0;
44. **for**(**int** i=1;i<=cntq;++i)
45. {
46. **while**(r<q[i].r)add(++r,1);
47. **while**(r>q[i].r)mus(r--,1);
48. **while**(l<q[i].l)add(++l,0);
49. **while**(l>q[i].l)mus(l--,0);
50. q[i].ans=ans;
51. }
52. sort(q+1,q+cntq+1,[](node x,node y){**return** x.pos<y.pos;});
53. **for**(**int** i=1;i<=m;++i)
54. {
55. **int** bs=(i-1)\*4;
56. printf("%d\n",q[bs+1].ans-q[bs+2].ans-q[bs+3].ans+q[bs+4].ans);
57. }
58. }

**6.4.4 回滚莫队**

有些题目在区间转移时，可能会出现增加或者删除无法实现的问题。在只有增加不可实现或者只有删除不可实现的时候，就可以使用回滚莫队在 O 的时间内解决问题。回滚莫队的核心思想就是既然我只能实现一个操作，那么我就只使用一个操作，剩下的交给回滚解决。回滚莫队分为只使用增加操作的回滚莫队和只使用删除操作的回滚莫队。以下仅介绍只使用增加操作的回滚莫队，只使用删除操作的回滚莫队和只使用增加操作的回滚莫队只在算法实现上有一点区别，故不再赘述。

具体算法

对原序列进行分块，对询问按以左端点所属块编号升序为第一关键字，右端点升序为第二关键字的方式排序

按顺序处理询问

如果询问左端点所属块 B 和上一个询问左端点所属块的不同，那么将莫队区间的左端点初始化为 B 的右端点加 1, 将莫队区间的右端点初始化为 B 的右端点

如果询问的左右端点所属的块相同，那么直接扫描区间回答询问

如果询问的左右端点所属的块不同

如果询问的右端点大于莫队区间的右端点，那么不断扩展右端点直至莫队区间的右端点等于询问的右端点

不断扩展莫队区间的左端点直至莫队区间的左端点等于询问的左端点

回答询问

撤销莫队区间左端点的改动，使莫队区间的左端点回滚到 B 的右端点加 1

例题：JOI2017コンテストーーー歴史の研究

文本, 信件

描述已自动生成

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **const** **int** N=1e5+10;
5. **struct** nd
6. {
7. **int** l,r,pos,ans;
8. }q[N];
9. **int** a[N],block[N],t[N],n,m;
10. **int** lsh[N];
11. **int** rk[N];
12. **int** B\_l[N],B\_r[N];
13. **int** ans;
14. **bool** cmp(nd x,nd y)
15. {
16. **if**(block[x.l]==block[y.l])**return** x.r<y.r;
17. **return** block[x.l]<block[y.l];
18. }
19. **void** solve1(**int** l,**int** r)
20. {
21. **for**(**int** i=l;i<=r;++i)
22. {
23. t[lsh[i]]+=a[i];
24. ans=max(t[lsh[i]],ans);
25. }
26. **for**(**int** i=l;i<=r;++i)
27. {
28. t[lsh[i]]-=a[i];
29. }
30. }
31. **void** add(**int** x)
32. {
33. t[lsh[x]]+=a[x];
34. ans=max(ans,t[lsh[x]]);
35. }
36. **void** mus(**int** x)
37. {
38. t[lsh[x]]-=a[x];
39. }
40. **signed** main()
41. {
42. cin>>n>>m;
43. **int** sqr=sqrt(n);
44. **for**(**int** i=1;i<=n;++i)
45. {
46. block[i]=(i-1)/sqr;
47. scanf("%lld",&a[i]);
48. rk[i]=a[i];
49. }
50. sort(rk+1,rk+1+n);
51. **int** cnt=unique(rk+1,rk+1+n)-rk-1;
52. **for**(**int** i=1;i<=n;++i)
53. {
54. lsh[i]=lower\_bound(rk+1,rk+1+cnt,a[i])-rk;
55. }
56. **for**(**int** i=1;i<=n;++i)B\_r[block[i]]=i;
57. **for**(**int** i=n;i;i--)B\_l[block[i]]=i;
58. **for**(**int** i=1;i<=m;++i)
59. {
60. **int** l,r;scanf("%lld%lld",&l,&r);
61. q[i].l=l;q[i].r=r;q[i].pos=i;
62. }
63. sort(q+1,q+1+m,cmp);
64. **int** l=B\_r[0]+1,r=B\_r[0];
65. **for**(**int** i=1;i<=m;++i)
66. {
67. **int** tmp=0;
68. **if**(block[q[i].l]!=block[q[i-1].l]){l=B\_r[block[q[i].l]]+1;r=B\_r[block[q[i].l]];ans=0;}
69. **if**(block[q[i].l]==block[q[i].r])solve1(q[i].l,q[i].r);
70. **else**
71. {
72. **while**(r<q[i].r)add(++r);
73. tmp=ans;
74. **while**(l>q[i].l)add(--l);
75. **while**(l<B\_r[block[q[i].l]]+1)mus(l++);
76. **if**(block[q[i].l]!=block[q[i+1].l])**while**(r>B\_r[block[q[i].l]])mus(r--);
77. }
78. q[i].ans=ans;
79. ans=tmp;
80. tmp=0;
81. }
82. sort(q+1,q+1+m,[](nd x,nd y){**return** x.pos<y.pos;});
83. **for**(**int** i=1;i<=m;++i)
84. {
85. printf("%lld\n",q[i].ans);
86. }
87. }

## 6.5 可持久数据结构

**6.5 可持久数据结构**

**6.5 可持久线段树**

线段树是维护数据的常用数据结构，但是有时要维护数据并保留修改之前的状态（版本T），就需要用到可持久化线段树。一般可持久化权值SGT用的比较多。比较典型的就是求解区间k小值。

线段树基于链表的形式存储而非线性结构，这里使用数组模拟链表。

a)建立持久化线段树空树。

1. **int** build(**int** l,**int** r)
2. {
3. **int** y=++tot; //建立新的节点
4. **if**(l==r)
5. {
6. tree[y]=0;
7. **return** y;
8. }
9. **int** mid=(l+r)>>1;
10. lson[y]=build(l,mid);  //建立左子树
11. rson[y]=build(mid+1,r);  //建立右子树
12. **return** y;  //返回新节点的编号，连接链表
13. }  //可以直接写T[0]=0来代替

空树一般也认为是第0版本的线段树（）

b)可持久化线段树的更新（迭代）

1. **int** up(**int** l,**int** r,**int** pre,**int** pos,**int** val)
2. {
3. **int** y=++tot;
4. **if**(l==r)
5. {
6. tree[y]=tree[pre]+val;
7. **return** y;
8. }
9. **int** mid=(l+r)>>1;
10. **if**(pos<=mid)
11. {
12. rson[y]=rson[pre];
13. lson[y]=up(l,mid,lson[pre],pos,val);
14. }
15. **else**
16. {
17. lson[y]=lson[pre];
18. rson[y]=up(mid+1,r,rson[pre],pos,val);
19. }  //接下一页Nxt pge。。。。
20. tree[y]=tree[lson[y]]+tree[rson[y]];
21. **return** y;
22. }

更新时，从的跟节点开始修改(设当前的更新是基于版本)，只修改需要修改的那条路径，创建新的节点，其他没有修改的部分保持不变，最后记录这个新版本的根节点。

c)访问，查询

可持久化线段树的访问较为灵活，区间访问传入两个版本的()，也有就访问一个版本的

大致的代码框架如下:

1. **int** query(**int** l,**int** r,**int** x,**int** y,**... ... ...**)
2. {
3. **if**(l==r)
4. {
5. **return** ...;
6. }
7. **int** mid=(l+r)>>1;
8. **if**(...)
9. {
10. **return** query(l,mid,ls[x],ls[y],...);
11. }
12. **else** **if**(...)
13. {
14. **return** query(mid+1,r,rs[x],rs[y],...);
15. }
16. **return** ...;
17. }

其中x，y是要查询比较的两个版本。

d）合并

**6.5.2 可持久化前缀树**

可持久化线段树经常用于区间异或问题的求解。其特征基本与可持久化线段树类似，但是由于其的全集不确定性，不能build（），我们只能令

与普通的前缀树不同的是，可持久化前缀树我们通常使用递归进行插入或者查询。

文本, 信件

描述已自动生成

例题算法为分治求解+可持久化前缀树

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** N=1e5+5;
4. **const** **int** M=34000005;
5. **int** ch[M][2],T[N],tot,n,a[N],num[M];
6. **int** ans;
7. **int** mina[N],maxa[N],suma[N],minl,maxl,suml;
8. **int** inst(**int** pre,**int** k,**int** bit)
9. {
10. **int** y=++tot;
11. **if**(bit==0)  //注意到0
12. {
13. num[y]=num[pre]+1;
14. **return** y;
15. }
16. **if**((k>>(bit-1))&1)
17. {
18. ch[y][0]=ch[pre][0];
19. ch[y][1]=inst(ch[pre][1],k,bit-1);
20. }
21. **else**
22. {
23. ch[y][1]=ch[pre][1];
24. ch[y][0]=inst(ch[pre][0],k,bit-1);
25. }
26. num[y]=num[ch[y][1]]+num[ch[y][0]];
27. **return** y;
28. }
29. **int** query(**int** L,**int** R,**int** bit,**int** k,**int** sum=0)
30. {
31. **int** nw=(k>>(bit-1))&1;
32. **if**(bit==0)**return** sum;
33. **if**(ch[R][nw^1]-ch[L][nw^1]>0)
34. {
35. **return** query(ch[L][nw^1],ch[R][nw^1],bit-1,k,sum\*2+1);
36. }
37. **else**
38. {
39. **return** query(ch[L][nw],ch[R][nw],bit-1,k,sum\*2);
40. }
41. }
42. **void** divide(**int** l,**int** r)
43. {
44. **if**(l==r)
45. {
46. ans=max(ans,a[l]);
47. **return**;
48. }
49. **int** mid=(l+r)>>1;
50. minl=0x7fffffff;maxl=-1;suml=0;
51. **int** p1,p2;
52. **for**(**int** i=mid;i>=l;i--)
53. {
54. minl=min(minl,a[i]);
55. maxl=max(maxl,a[i]);
56. suml^=a[i];
57. mina[i]=minl;
58. maxa[i]=maxl;
59. suma[i]=suml;
60. }
61. minl=0x7fffffff;maxl=-1;suml=0;
62. **for**(**int** i=mid+1;i<=r;++i)
63. {
64. minl=min(minl,a[i]);
65. maxl=max(maxl,a[i]);
66. suml^=a[i];
67. mina[i]=minl;
68. maxa[i]=maxl;
69. suma[i]=suml;
70. }
71. tot=0;
72. T[mid]=0;//min max left
73. **for**(**int** i=mid+1;i<=r;++i)
74. {
75. T[i]=inst(T[i-1],suma[i],30);
76. }
77. p1=mid+1;
78. **for**(**int** i=mid;i>=l;i--)
79. {
80. **while**(mina[p1]>=mina[i]&&maxa[p1]<=maxa[i]&&p1<=r)p1++;
81. **int** tmp=mina[i]^maxa[i]^suma[i];
82. ans=max(ans,tmp);
83. **if**(p1>mid+1)
84. {
85. ans=max(ans,query(T[mid],T[p1-1],30,tmp));
86. }
87. }tot=0;
88. T[mid+1]=0;//min max rit
89. **for**(**int** i=mid;i>=l;i--)
90. {
91. T[i]=inst(T[i+1],suma[i],30);
92. }
93. p1=mid;
94. **for**(**int** i=mid+1;i<=r;++i)
95. {
96. **while**(mina[p1]>=mina[i]&&maxa[p1]<=maxa[i]&&p1>=l)p1--;
97. **int** tmp=maxa[i]^mina[i]^suma[i];
98. ans=max(ans,tmp);
99. **if**(p1<mid)
100. {
101. ans=max(ans,query(T[mid+1],T[p1+1],30,tmp));
102. }
103. }tot=0;
104. T[mid]=0;//min left max rit
105. **for**(**int** i=mid+1;i<=r;++i)
106. {
107. T[i]=inst(T[i-1],suma[i]^maxa[i],30);
108. }
109. p1=p2=mid+1;
110. **for**(**int** i=mid;i>=l;--i)
111. {
112. **while**(maxa[p1]<maxa[i]&&p1<=r)p1++;
113. **while**(mina[p2]>=mina[i]&&p2<=r)p2++;
114. **int** tmp=mina[i]^suma[i];
115. **if**(p2>p1)ans=max(ans,query(T[p1-1],T[p2-1],30,tmp));
116. }tot=0;
117. T[mid]=0;//min rit max left
118. **for**(**int** i=mid+1;i<=r;++i)
119. {
120. T[i]=inst(T[i-1],suma[i]^mina[i],30);
121. }
122. p1=p2=mid+1;
123. **for**(**int** i=mid;i>=l;--i)
124. {
125. **while**(mina[p1]>mina[i]&&p1<=r)p1++;
126. **while**(maxa[p2]<=maxa[i]&&p2<=r)p2++;
127. **int** tmp=maxa[i]^suma[i];
128. **if**(p2>p1)ans=max(ans,query(T[p1-1],T[p2-1],30,tmp));
129. }
130. divide(l,mid);divide(mid+1,r);
131. }
132. **signed** main()
133. {
134. cin>>n;
135. **for**(**int** i=1;i<=n;++i)
136. {
137. scanf("%d",&a[i]);
138. }
139. ans=0;
140. divide(1,n);
141. printf("%d\n",ans);
142. }

## 6.6 ST表

1. /\*
2. \*  求最大值,数组下标从1开始。
3. \*  求最小值,或者最大最小值下标,或者数组从0开始对应修改即可。
4. \*/
5. **const** **int** MAXN = 50010;
6. **int** dp[MAXN][20];
7. **int** mm[MAXN];
9. //  初始化RMQ,b数组下标从1开始,b数组是区间元素序列
10. **void** initRMQ(**int** n, **int** b[])
11. {
12. mm[0] = -1;
13. **for** (**int** i = 1; i <= n; i++)
14. {
15. mm[i] = ((i & (i - 1)) == 0) ? mm[i - 1] + 1 : mm[i - 1];
16. dp[i][0] = b[i];
17. }
18. **for** (**int** j = 1; j <= mm[n]; j++)
19. {
20. **for** (**int** i = 1; i + (1 << j) - 1 <= n; i++)
21. {
22. dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
23. }
24. }
25. }
27. //  查询最大值
28. **int** rmq(**int** x, **int** y)
29. {
30. **int** k = mm[y - x + 1];
31. **return** max(dp[x][k], dp[y - (1 << k) + 1][k]);
32. }

## 6.7动态树之LCT

数据结构动态树（Link Cut tree）常常用于解决树上问题，这些问题通常要求修改树的结构和查询树链信息。

LCT是一种基于伸展树(Splay Tree)的一种高级数据结构，支持多种修改以及询问。

**什么是辅助树？**

**文本

描述已自动生成**

**原树和辅助树的关系？**

**文本

描述已自动生成**

**LCT用到的数组：**

|  |  |  |  |
| --- | --- | --- | --- |
| 数组名 | 作用 | *val[N]* | 点权 |
| *f[N]* | 父亲节点 | *laz[N]* | 权值标记 |
| *tag[N]* | 翻转标记 | *ch[N][2]* | LCT结构 |
| *sum[N]* | 权值和 | *siz[N]* | 辅助树大小 |

**函数：**

**图形用户界面, 文本

描述已自动生成**

**图片包含 图形用户界面

描述已自动生成**

**具体实现：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **const** **int** maxn = 1e5+10;
5. **const** **int** mod = 999999999;
6. **int** n, q, u, v, c;
7. **char** op;
8. **struct** Link\_Cut\_Tree {
9. **int** ch[maxn][2], fa[maxn], siz[maxn], val[maxn], sum[maxn], rev[maxn],add[maxn], mul[maxn];
10. **void** clear(**int** x) {
11. ch[x][0] = ch[x][1] = fa[x] = siz[x] = val[x] = sum[x] = rev[x] = add[x] =0;
12. mul[x] = 1;
13. }
14. **int** getch(**int** x) { **return** (ch[fa[x]][1] == x); }
15. **int** isroot(**int** x) {
16. clear(0);
17. **return** ch[fa[x]][0] != x && ch[fa[x]][1] != x;
18. }
19. **void** maintain(**int** x) {
20. clear(0);
21. siz[x] = (siz[ch[x][0]] + 1 + siz[ch[x][1]]) % mod;
22. sum[x] = (sum[ch[x][0]] + val[x] + sum[ch[x][1]]) % mod;
23. }
24. **void** pushdown(**int** x) {
25. clear(0);
26. **if** (mul[x] != 1) {
27. **if** (ch[x][0])
28. mul[ch[x][0]] = (mul[x] \* mul[ch[x][0]]) % mod,
29. val[ch[x][0]] = (val[ch[x][0]] \* mul[x]) % mod,
30. sum[ch[x][0]] = (sum[ch[x][0]] \* mul[x]) % mod,
31. add[ch[x][0]] = (add[ch[x][0]] \* mul[x]) % mod;
32. **if** (ch[x][1])
33. mul[ch[x][1]] = (mul[x] \* mul[ch[x][1]]) % mod,
34. val[ch[x][1]] = (val[ch[x][1]] \* mul[x]) % mod,
35. sum[ch[x][1]] = (sum[ch[x][1]] \* mul[x]) % mod,
36. add[ch[x][1]] = (add[ch[x][1]] \* mul[x]) % mod;
37. mul[x] = 1;
38. }
39. **if** (add[x]) {
40. **if** (ch[x][0])
41. add[ch[x][0]] = (add[ch[x][0]] + add[x]) % mod,
42. val[ch[x][0]] = (val[ch[x][0]] + add[x]) % mod,
43. sum[ch[x][0]] = (sum[ch[x][0]] + add[x] \* siz[ch[x][0]] % mod) % mod;
44. **if** (ch[x][1])
45. add[ch[x][1]] = (add[ch[x][1]] + add[x]) % mod,
46. val[ch[x][1]] = (val[ch[x][1]] + add[x]) % mod,
47. sum[ch[x][1]] = (sum[ch[x][1]] + add[x] \* siz[ch[x][1]] % mod) % mod;
48. add[x] = 0;
49. }
50. **if** (rev[x]) {
51. **if** (ch[x][0]) rev[ch[x][0]] ^= 1, swap(ch[ch[x][0]][0], ch[ch[x][0]][1]);
52. **if** (ch[x][1]) rev[ch[x][1]] ^= 1, swap(ch[ch[x][1]][0], ch[ch[x][1]][1]);
53. rev[x] = 0;
54. }
55. }
56. **void** update(**int** x) {
57. **if** (!isroot(x)) update(fa[x]);
58. pushdown(x);
59. }
60. **void** print(**int** x) {
61. **if** (!x) **return**;
62. pushdown(x);
63. print(ch[x][0]);
64. printf("%lld ", x);
65. print(ch[x][1]);
66. }
67. **void** rotate(**int** x) {
68. **int** y = fa[x], z = fa[y], chx = getch(x), chy = getch(y);
69. fa[x] = z;
70. **if** (!isroot(y)) ch[z][chy] = x;
71. ch[y][chx] = ch[x][chx ^ 1];
72. fa[ch[x][chx ^ 1]] = y;
73. ch[x][chx ^ 1] = y;
74. fa[y] = x;
75. maintain(y);
76. maintain(x);
77. maintain(z);
78. }
79. **void** splay(**int** x) {
80. update(x);
81. **for** (**int** f = fa[x]; f = fa[x], !isroot(x); rotate(x))
82. **if** (!isroot(f)) rotate(getch(x) == getch(f) ? f : x);
83. }
84. **int** access(**int** x) {
85. **int** f;
86. **for** (f = 0; x; f = x, x = fa[x]) splay(x), ch[x][1] = f, maintain(x);
87. **return** f;
88. }
89. **void** makeroot(**int** x) {
90. access(x);
91. splay(x);
92. swap(ch[x][0], ch[x][1]);
93. rev[x] ^= 1;
94. }
95. **int** find(**int** x) {
96. access(x);
97. splay(x);
98. **while** (ch[x][0]) x = ch[x][0];
99. splay(x);
100. **return** x;
101. }
102. } st;
103. **signed** main()
104. {
105. cin>>n;
106. **for**(**int** i=1;i<n;++i)
107. {
108. scanf("%lld%lld",&u,&v);
109. st.makeroot(u);st.fa[v]=u;
110. }
111. cin>>q;
112. **for**(**int** i=1;i<=q;++i)
113. {
114. scanf("%lld%lld%lld",&c,&u,&v);
115. st.makeroot(c);
116. st.access(u);
117. printf("%lld\n",st.access(v));
118. }
119. }

## 6.8 树链剖分

**5.5.1 dfs序，欧拉序，扩展dfn**

dfs序是一种基于树的思想，如果我们对一棵树进行dfs，记录每个点的出栈入栈时间序列，一个点的子树上的所有结点入栈时间序列一定在这个点的入栈时间序列之后，出栈时间序列一定在这个点的出栈时间序列之前。也就是说，我们对一颗树进行dfs，按照入栈顺序时间序列进行标号。那么一个节点的子树上的所有节点的编号一定是连续的， 如图5.5.1-1所示。

图 5.5.1-1



图中x的子树上的所有节点按照dfs入栈时间序列标号后，严格在[1,6]的范围内。我们就可以用这些特性来将树上问题进行线性存储。

欧拉序和扩展dfn均为双倍长度的dfs序，欧拉序为每个点在出栈与入栈时间轴分别给予一个dfs序标号，多用于树链问题（比如树上离线莫队算法），扩展dfn则为在遍历每一个点之前，都给予他的父节点一个标号，这样有利于区间求LCA，可以将区间求LCA问题转化为RMQ（区间深度最小）问题求解。

**5.5.2 树链剖分**

通常有些题目要对树链上的数据进行区间操作。由于不管是分块算法，树状数组还是线段树，都只能对线性存储的数据进行维护。所以我们就要对树链进行剖分，将树上的数据转化成线性的数据进行存储。

我们需要知道 ：

id[M],fa[M],top[M],sz[M],rk[M],out[M],dep[M],son[M];

这些信息，分别代表

|  |  |  |  |
| --- | --- | --- | --- |
| Id数组 | dfs序编号为下标值为原来的编号 | Dep数组 | 原编号为下标存该点的深度 |
| Fa数组 | 原编号为下标值为该点的父亲节点 | Son数组 | 原编号为下标存该点的重儿子 |
| Top数组 | 原编号为下标，值为该点的top节点 | Sz数组 | 原编号为下标存该点与子树的大小 |
| Rk数组 | 原编号为下标，存dsf序 | \*in out数组 | Dfs序，部分题目使用 |

树链剖分的主要部分由两遍dfs组成，一般情况下，第一遍dfs确定每个点的重儿子，第二遍dfs，每个点与其重儿子成链，并记录每个节点所在链的链首（每一条链上深度最小的节点）。

第一遍dfs

1. **int** dfs1(**int** f, **int** nw)
2. {
3. **int** tt;
4. **int** maxn = 0, nex;
5. a[nw].dep = a[f].dep + 1;
6. a[nw].fa = f;
7. a[nw].sz = **1**; //（**注意**！！！！！！size（）=1）
8. **for** (**int** i = head[nw]; i; i = nxt[i])
9. {
10. tt = to[i];
11. **if** (tt == f)**continue**;
12. nex = dfs1(nw, tt);
13. **if** (nex > maxn)
14. {
15. maxn = nex;
16. a[nw].son = tt;
17. }
18. a[nw].sz += nex;
19. }
20. **return** a[nw].sz;
21. }

第二遍dfs

1. **void** dfs2(**int** tp, **int** nw)
2. {
3. **int** tt;
4. cnt++;
5. a[nw].top = tp;
6. a[nw].rk = cnt;
7. id[cnt] = nw;
8. **if** (a[nw].son)
9. {
10. dfs2(tp, a[nw].son);
11. }
12. **for** (**int** i = head[nw]; i; i = nxt[i])
13. {
14. tt = to[i];
15. **if** (tt == a[nw].fa || tt == a[nw].son)**continue**;
16. dfs2(tt, tt);
17. }
18. a[nw].out = cnt;
19. }

这样，我们就将所有树链进行了剖分，如果要修改某条链上的数据，只要从端点开始往上跑到LCA，由于每条链上的编号都是连续的，所以我们对其中同一条链上的数据进行区间修改，可以大大降低操作时间。

1. **void** LCA(**int** x, **int** y)
2. {
3. **while** (a[x].top != a[y].top)
4. {
5. **if** (a[a[x].top].dep < a[a[y].top].dep)
6. {
7. swap(x, y);
8. }
9. Update/query(x与x的链首的区间，注意x是原始编号);
10. x = a[a[x].top].fa;
11. }
12. **if** (a[x].rk > a[y].rk)
13. {
14. swap(x, y);
15. }
16. Update/query(x与y的区间，注意x,y是原始编号);
17. }

**5.5.3 树链剖分典型题目**

5.5.3.1 Micercn and country树链剖分：

***Problem Description***

*Miceren finds a huge country named HY. HY has N cities numbered from 1 to N connected by N − 1 bidirectional roads. There exists a path between any two cities.  
  
It can be imagined as a tree with n vertices rooted at vertex 1.  
  
Miceren wants to occupy some cities here. Each city has a value vi. (Notice that the value of a city may be negative. Nevertheless, Miceren wants to occupied this city.)  
  
As some usual stories, someone named Cloud wants to "steal" some cities from Miceren.  
  
At the beginning, Miceren and Cloud don't occupy any city.  
  
In the following Q days, one of three events may happen  
  
1. Miceren will walk from the a-th city to the b-th city and all cities visited in this trip will belong to Miceren. (1 ≤ a,b ≤ N)  
  
2. Cloud will steal the x-th city. If Miceren occupied the x-th city before, Miceren will lost the control of this city. (1 ≤ x ≤ N)  
  
3. Miceren will occupy the subtree rooted at x.(1 ≤ x ≤ N)  
  
As Miceren's friend, you must tell Miceren the total value of all cities which belong to Miceren after each day.*

1. **struct** node
2. {
3. **int** v,fa,top,son,sz,rk,out,dep;
4. };
5. **int** id[M],cnt,tot,head[M],to[2\*M],nxt[2\*M],tree[4\*M],lazy[4\*M];
6. node a[M];
7. **int** n,m,in1,in2,in3,q;
8. **int** presum[M];
9. **inline** **void** Add(**int** s,**int** t)
10. {
11. tot++;
12. to[tot]=t;
13. nxt[tot]=head[s];
14. head[s]=tot;
15. }
16. **signed** main()
17. {
18. **int** dfs1(**int** f,**int** nw);
19. **void** dfs2(**int** tp,**int** nw);
20. **void** build(**int** l,**int** r,**int** nw);
21. **void** LCA(**int** a,**int** b);
22. **void** update(**int** l,**int** r,**int** L,**int** R,**int** nw,**int** k);
23. **int** t;
24. cin>>t;
25. **while**(t--)
26. {
27. memset(lazy,-1,**sizeof** lazy);
28. memset(a,0,**sizeof** a);
29. memset(head,0,**sizeof** head);
31. cnt=tot=0; presum[0]=0;
32. cin>>n;
33. **for**(**int** i=1;i<=n;i++)
34. {
35. cin>>a[i].v;
36. }
37. **for**(**int** i=1;i<n;i++)
38. {
39. cin>>in1>>in2;
40. Add(in1,in2);
41. Add(in2,in1);
42. }
43. dfs1(0,1);
44. dfs2(1,1);
45. build(1,n,1);
46. **for**(**int** i=1;i<=n;i++)
47. {
48. presum[i]=presum[i-1]+a[id[i]].v;
49. }
50. cin>>q;
51. **for**(**int** i=1;i<=q;i++)
52. {
53. cin>>in1;
54. **if**(in1==1)
55. {
56. cin>>in2>>in3;
57. LCA(in2,in3);
58. }
59. **else** **if**(in1==2)
60. {
61. cin>>in2;
62. update(1,n,a[in2].rk,a[in2].rk,1,0);
63. }
64. **else**
65. {
66. cin>>in2;
67. update(1,n,a[in2].rk,a[in2].out,1,1);
68. }
69. cout<<tree[1]<<endl;
70. }
71. }
72. **return** 0;
73. }
75. **int** dfs1(**int** f,**int** nw)
76. {
77. **int** tt;
78. **int** maxn=0,nex;
79. a[nw].dep=a[f].dep+1;
80. a[nw].fa=f;
81. a[nw].sz=1;
82. **for**(**int** i=head[nw];i;i=nxt[i])
83. {
84. tt=to[i];
85. **if**(tt==f)**continue**;
86. nex=dfs1(nw,tt);
87. **if**(nex>maxn)
88. {
89. maxn=nex;
90. a[nw].son=tt;
91. }
92. a[nw].sz+=nex;
93. }
94. **return** a[nw].sz;
95. }
96. **void** dfs2(**int** tp,**int** nw)
97. {
98. **int** tt;
99. cnt++;
100. a[nw].top=tp;
101. a[nw].rk=cnt;
102. id[cnt]=nw;
103. **if**(a[nw].son)
104. {
105. dfs2(tp,a[nw].son);
106. }
107. **for**(**int** i=head[nw];i;i=nxt[i])
108. {
109. tt=to[i];
110. **if**(tt==a[nw].fa||tt==a[nw].son)**continue**;
111. dfs2(tt,tt);
112. }
113. a[nw].out=cnt;
114. }
115. **void** build(**int** l,**int** r,**int** nw)
116. {
117. **if**(l==r)
118. {
119. tree[nw]=0;
120. **return**;
121. }
122. **int** mid=(l+r)/2;
123. build(l,mid,nw\*2);
124. build(mid+1,r,nw\*2+1);
125. tree[nw]=tree[nw\*2]+tree[nw\*2+1];
126. **return**;
127. }
128. **void** pushdown(**int** l,**int** mid,**int** r,**int** nw)
129. {
130. lazy[nw\*2]=lazy[nw];
131. lazy[nw\*2+1]=lazy[nw];
132. tree[nw\*2]=(presum[mid]-presum[l-1])\*lazy[nw];
133. tree[nw\*2+1]=(presum[r]-presum[mid])\*lazy[nw];
134. lazy[nw]=-1;
135. }
136. **void** update(**int** l,**int** r,**int** L,**int** R,**int** nw,**int** k)
137. {
138. **if**(l>=L&&r<=R)
139. {
140. lazy[nw]=k;
141. tree[nw]=k\*(presum[r]-presum[l-1]);
142. **return**;
143. }
144. **int** mid=(l+r)/2;
145. **if**(l>R||r<L)
146. {
147. **return**;
148. }
149. **if**(l==r)**return**;
150. **if**(lazy[nw]!=-1)
151. {
152. pushdown(l,mid,r,nw);
153. }
154. update(l,mid,L,R,nw\*2,k);
155. update(mid+1,r,L,R,nw\*2+1,k);
156. tree[nw]=tree[nw\*2]+tree[nw\*2+1];
157. **return**;
158. }
159. **void** LCA(**int** x,**int** y)
160. {
161. **while**(a[x].top!=a[y].top)
162. {
163. **if**(a[a[x].top].dep<a[a[y].top].dep)
164. {
165. swap(x,y);
166. }
167. update(1,n,a[a[x].top].rk,a[x].rk,1,1);
168. x=a[a[x].top].fa;
169. }
170. **if**(a[x].rk>a[y].rk)
171. {
172. swap(x,y);
173. }
174. update(1,n,a[x].rk,a[y].rk,1,1);
175. }

5.5.3.2 I love tree（树链剖分+平方和线段树）

*Given a tree with n nodes and q operations, there are two kinds of operations.  
1 a b & 2 x*

*1 a b: For the chain between <a,b>point x is the p.th point . operation: +=p\*p;*

*2 x: query Valx.*

*Solve: with each operations 1.Each point increase num\*num. We can take it to this type : (x-h)\*(x-h)=.so we should only known the sum of h and the square sum of h in every time.*

其中 x-h 是点x在每次操作的链的位置。对于位于之间的点 位于之间的点

所以 由 其中sump 是链上的总节点数。

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **const** **int** M = 2 \* 1e5 + 1;
5. **struct** node
6. {
7. **int** v, fa, top, son, sz, rk, out, dep;
8. **int** sqh, sumh, lazy;
9. **int** val;
10. };
11. **int** id[M], cnt, tot, head[M], to[2 \* M], nxt[2 \* M];
12. node tree[4 \* M + 1];
13. node a[M];
14. **int** n, m, in1, in2, in3, q;
15. //int presum[M];
16. **inline** **void** Add(**int** s, **int** t)
17. {
18. tot++;
19. to[tot] = t;
20. nxt[tot] = head[s];
21. head[s] = tot;
22. }
23. **int** Qlca(**int** x, **int** y)
24. {
25. **while** (a[x].top != a[y].top)
26. {
27. **if** (a[a[x].top].dep < a[a[y].top].dep)
28. {
29. swap(x, y);
30. }
31. x = a[a[x].top].fa;
32. }
33. **if** (a[x].rk > a[y].rk)
34. {
35. swap(x, y);
36. }
37. **return** x;
38. }
39. **void** push\_down(**int** nw)
40. {
41. tree[nw \* 2].lazy += tree[nw].lazy;
42. tree[nw \* 2].sqh += tree[nw].sqh;
43. tree[nw \* 2].sumh += tree[nw].sumh;
44. tree[nw \* 2 + 1].lazy += tree[nw].lazy;
45. tree[nw \* 2 + 1].sqh += tree[nw].sqh;
46. tree[nw \* 2 + 1].sumh += tree[nw].sumh;
47. tree[nw].lazy = tree[nw].sumh = tree[nw].sqh = 0;
48. }
49. **void** update(**int** l, **int** r, **int** L, **int** R, **int** nw, **int** k)
50. {
51. **if** (l > r)**return**;
52. //    cout<<L<<" "<<R<<" "<<l<<" "<<r<<endl;
53. **if** (l >= L && r <= R)
54. {
55. //        cout<<" 修改的区间为："<<l<<" "<<r<<endl;
56. tree[nw].sumh += k;
57. tree[nw].sqh += (k \* k);
58. tree[nw].lazy++;
59. //        cout<<tree[nw].lazy<<"懒惰标记"<<endl;
60. **return**;
61. }
62. **if** (l == r)**return**;
63. **if** (l > R || r < L)
64. {
65. **return**;
66. }
67. **if** (tree[nw].lazy)
68. {
69. push\_down(nw);
70. }
71. **int** mid = (l + r) / 2;
72. update(l, mid, L, R, nw \* 2, k);
73. update(mid + 1, r, L, R, nw \* 2 + 1, k);
74. }
75. **void** LCA1(**int** x, **int** y, **int** h)
76. {
77. **while** (a[x].top != a[y].top)
78. {
79. **if** (a[a[x].top].dep < a[a[y].top].dep)
80. {
81. swap(x, y);
82. }
83. update(1, n, a[a[x].top].rk, a[x].rk, 1, h);
84. x = a[a[x].top].fa;
85. }
86. **if** (a[x].rk > a[y].rk)
87. {
88. swap(x, y);
89. }
90. update(1, n, a[x].rk + 1, a[y].rk, 1, h);
91. }
92. **int** query(**int** l, **int** r, **int** goal, **int** nw)
93. {
94. **if** (l == r)
95. {
96. **return** tree[nw].lazy \* a[id[l]].dep \* a[id[l]].dep - 2 \* a[id[l]].dep \* tree[nw].sumh + tree[nw].sqh;
97. }
98. **int** mid = (l + r) / 2;
99. **if** (tree[nw].lazy)
100. {
101. //        cout<<"下传"<<nw<<endl;
102. push\_down(nw);
103. }
104. **if** (goal <= mid)**return** query(l, mid, goal, nw \* 2);
105. **else** **return** query(mid + 1, r, goal, nw \* 2 + 1);
106. }
107. **signed** main()
108. {
109. ios::sync\_with\_stdio(0);
110. **int** dfs1(**int** f, **int** nw);
111. **void** dfs2(**int** tp, **int** nw);
112. **void** build(**int** l, **int** r, **int** nw);
113. **void** LCA(**int** a, **int** b);
114. **void** update(**int** l, **int** r, **int** L, **int** R, **int** nw, **int** k);
116. //        memset(lazy,-1,sizeof lazy);
117. memset(a, 0, **sizeof** a);
118. memset(head, 0, **sizeof** head);
120. cnt = tot = 0;
121. //    presum[0]=0;
122. cin >> n;
123. **for** (**int** i = 1; i < n; i++)
124. {
125. cin >> in1 >> in2;
126. Add(in1, in2);
127. Add(in2, in1);
128. }
129. dfs1(0, 1);
130. dfs2(1, 1);
131. //    build(1,n,1);
132. //for(int i=1;i<=n;i++)cout<<i<<"成链下标为："<<a[i].rk<<endl;
133. cin >> q;
134. **for** (**int** i = 1; i <= q; i++)
135. {
136. cin >> in1;
137. **if** (in1 == 1)
138. {
139. cin >> in2 >> in3;
140. **int** lca = Qlca(in2, in3);
141. //cout<<in2<<" "<<in3<<"的最早公共祖先为："<<lca<<endl;
142. LCA(in2, lca);
143. **int** depsum = (a[lca].dep - a[in2].dep + a[lca].dep - a[in3].dep - 1);
144. depsum = -depsum;
145. LCA1(in3, lca,  a[in3].dep-depsum);
146. }
147. **else** **if** (in1 == 2)
148. {
149. cin >> in2;
150. //            update(1,n,a[in2].rk,a[in2].rk,1,0);
151. **int** ans = query(1, n, a[in2].rk, 1);
152. cout << ans << endl;
153. }
154. }
155. **return** 0;
156. }
158. **int** dfs1(**int** f, **int** nw)
159. {
160. **int** tt;
161. **int** maxn = 0, nex;
162. a[nw].dep = a[f].dep + 1;
163. a[nw].fa = f;
164. a[nw].sz = 1;
165. **for** (**int** i = head[nw]; i; i = nxt[i])
166. {
167. tt = to[i];
168. **if** (tt == f)**continue**;
169. nex = dfs1(nw, tt);
170. **if** (nex > maxn)
171. {
172. maxn = nex;
173. a[nw].son = tt;
174. }
175. a[nw].sz += nex;
176. }
177. **return** a[nw].sz;
178. }
179. **void** dfs2(**int** tp, **int** nw)
180. {
181. **int** tt;
182. cnt++;
183. a[nw].top = tp;
184. a[nw].rk = cnt;
185. id[cnt] = nw;
186. **if** (a[nw].son)
187. {
188. dfs2(tp, a[nw].son);
189. }
190. **for** (**int** i = head[nw]; i; i = nxt[i])
191. {
192. tt = to[i];
193. **if** (tt == a[nw].fa || tt == a[nw].son)**continue**;
194. dfs2(tt, tt);
195. }
196. a[nw].out = cnt;
197. }
198. **void** LCA(**int** x, **int** y)
199. {
200. **int** low = a[x].dep;
201. **while** (a[x].top != a[y].top)
202. {
203. **if** (a[a[x].top].dep < a[a[y].top].dep)
204. {
205. swap(x, y);
206. }
208. update(1, n, a[a[x].top].rk, a[x].rk, 1, low + 1);
209. x = a[a[x].top].fa;
210. }
211. **if** (a[x].rk > a[y].rk)
212. {
213. swap(x, y);
214. }
215. update(1, n, a[x].rk, a[y].rk, 1, low + 1);
216. }

# 7．数学模块

## 7.1 扩展欧几里德与贝祖定理

### 7.1.1 概述

为了引进扩展欧几里德，首先讲讲贝祖定理：如果a、b是整数，那么一定存在整数x、y使得。换句话说，如果有解，那么m一定是gcd(a,b)的若干倍。（可以来判断一个这样的式子有没有解），扩展欧几里德则是用来求解方程

**7.1.1.2 扩欧模板**

1. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
2. {
3. **if**(b==0)
4. {
5. x=1;
6. y=0;
7. **return** a;
8. }
9. **int** ans=exgcd(b,a%b,x,y);
10. **int** tmp=x;
11. x=y;
12. y=tmp-(a/b)\*y;
13. **return** ans;
14. }

**7.1.1.3 判断方程ax+by=m或是否有解**

**思路：**当gcd(a,b)|m时 方程有解，反之则无解

**7.1.1.4 求解方程ax+by=m( ax≡m(mod b) )的最小正整数解**

**思路：**

1.首先判断方程是否有解，无解则直接退出

2.扩欧求x，将( ( (x\*m/gcd(a,b) )%(b/gcd(a,b) )+b/gcd(a,b))%(b/gcd(a,b)即可

### 7.1.2 经典例题

**7.1.2.1 洛谷 P1082 同余方程**

**题目描述**

求关于x的同余方程ax≡1(mod b)的最小正整数解。

**输入格式**

一行，包含两个正整数a,b,用一个空格隔开。

**输出格式**

一个正整数x0，即最小正整数解。输入数据保证一定有解。

**输入样例**

3 10

**输出样例**

7

**思路：**模板题，由于题目规定一定有解，因此a⊥b（a与b互质） ，此时既可以用扩欧求解，也可用欧拉定理求解

在这科普一下欧拉定理：

(其中，a与n均为正整数，且两者互质。) 费马小定理为欧拉定理的特殊形式

**题解（见下页）**

**1、扩欧：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
5. {
6. **if**(b==0)
7. {
8. x=1;
9. y=0;
10. **return** a;
11. }
12. **int** ans=exgcd(b,a%b,x,y);
13. **int** tmp=x;
14. x=y;
15. y=tmp-(a/b)\*y;
16. **return** ans;
17. }
18. **signed** main()
19. {
20. **int** a,b;
21. **int** x,y;
22. cin>>a>>b;
23. exgcd(a,b,x,y);
24. x=(x%b+b)%b;
25. cout<<x<<endl;
26. **return** 0;
27. }

**2、欧拉定理：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **int** qpow(**int** a,**int** n,**int** b)
5. {
6. **int** ans=1;
7. **while**(n)
8. {
9. **if**(n&1) ans=ans%b\*a%b;
10. a=a%b\*a%b;
11. n>>=1;
12. }
13. **return** ans;
14. }
15. **int** euler(**int** n)
16. {
17. **int** ans=n;
18. **for**(**int** i=2;i\*i<=n;i++)
19. {
20. **if**(n%i==0) ans-=ans/i;
21. **while**(n%i==0) n/=i;
22. }
23. **if**(n>1) ans-=ans/n;
24. **return** ans;
25. }
26. **signed** main()
27. {
28. **int** a,b;
29. cin>>a>>b;
30. cout<<qpow(a,euler(b)-1,b)<<endl;
31. **return** 0;
32. }

**7.1.5.2 洛谷P5656 二元一次方程**

**题目描述**

给定不定方程ax+by=c

若该方程无整数解，输出-1。

若该方程有整数解，且有正整数解，则输出其正整数解的数量，所有正整数解中x的最小值，所有正整数解中y的最小值，所有正整数解中x的最大值，以及所有正整数解中y 的最大值。

若方程有整数解，但没有正整数解，你需要输出所有整数解中x的最小正整数值，y的最小正整数值。

正整数解即为 x, y 均为正整数的解， 0 不是正整数。

整数解即为 x,y 均为整数的解。

x的最小正整数值即所有x为正整数的整数解中x的最小值，y同理。

**输入格式**

第一行一个正整数 T，代表数据组数。

接下来 T 行，每行三个由空格隔开的正整数a,b,c。

**输出格式**

T 行。

若该行对应的询问无整数解，一个数字 -1。

若该行对应的询问有整数解但无正整数解，包含 2个由空格隔开的数字，依次代表整数解中，x的最小正整数值，y 的最小正整数值。

否则包含5个由空格隔开的数字，依次代表正整数解的数量，正整数解中，x的最小值，y的最小值，x的最大值，y的最大值。

**输入样例**

7

2 11 100

3 18 6

192 608 17

19 2 60817

11 45 14

19 19 810

98 76 5432

**输出样例**

4 6 2 39 8

2 1

-1

1600 1 18 3199 30399

34 3

-1

2 12 7 50 56

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
5. {
6. **if**(b==0)
7. {
8. x=1;
9. y=0;
10. **return** a;
11. }
12. **int** ans=exgcd(b,a%b,x,y);
13. **int** tmp=x;
14. x=y;
15. y=tmp-(a/b)\*y;
16. **return** ans;
17. }
18. **signed** main()
19. {
20. **int** t,a,b,c;
21. **int** x,y;
22. **int** minx,miny;
23. **int** maxx,maxy;
24. **int** cnt=1;
25. scanf("%lld",&t);
26. **bool** flag=**false**;
27. **while**(t--)
28. {
29. flag=**false**;
30. cnt=0;
31. scanf("%lld %lld %lld",&a,&b,&c);
32. **int** gcd=exgcd(a,b,x,y);
33. **if**(c%gcd)
34. {
35. cout<<"-1"<<endl;
36. **continue**;
37. }
38. **int** bg=b/gcd;
39. **int** cg=c/gcd;
40. **int** ag=a/gcd;
41. **int** tmp;
42. x\*=cg;
43. y\*=cg;
44. //cout<<"x= "<<x<<"y="<<y<<endl;
45. tmp=x;
46. x=(x%bg+bg-1)%bg+1;//最小正整数解  若是最小整数解则不用-1
47. **int** ansy=(y%ag+ag-1)%ag+1;
48. **int** t=(x-tmp)/bg;
49. y-=t\*ag;
50. **if**(x>0&&y>0)
51. {
52. minx=x;
53. maxy=y;
54. flag=**true**;
55. cnt=1;
56. **int** k=y/ag;
57. **if**(y%ag==0)
58. {
59. cnt+=k-1;
60. miny=ag;
61. maxx=x+(k-1)\*bg;
62. }
63. **else**
64. {
65. cnt+=k;
66. miny=y-k\*ag;
67. maxx=x+k\*bg;
68. }
69. }
70. **if**(flag) printf("%lld %lld %lld %lld %lld\n",cnt,minx,miny,maxx,maxy);
71. **else** printf("%lld %lld\n",x,ansy);
72. }
73. **return** 0;
74. }

## 7.2欧拉函数

**##规定 phi为φ##**

### 7.2.1概述

欧拉函数就是对于一个正整数n，小于n且和n互质的正整数（包括1）的个数，记作φ(n) 。

欧拉函数的通式：φ(n)=n\*(1-1/p1)(1-1/p2)(1-1/p3)\*(1-1/p4)……(1-1/pn)

其中p1, p2……pn为n的所有质因数，n是不为0的整数。φ(1)=1（唯一和1互质的数就是1本身）。

**7.2.1.2 性质**

① **当m,n互质时，有phi（m\*n）= phi（m）\*phi（n）；**

② **若i%p==0，有phi（i\*p） = p \* phi(i)；**

③ 对于互质x与p，有x^phi§≡1（mod p),因此x的逆元为x^(phi§-1)，**即欧拉定理**。

（特别地，当p为质数时，phi（p）=p-1,此时逆元为x^(p-2)，即费马小定理）

④ 当n为奇数时，phi(2n)=phi(n)

⑤ 若x与p互质，则p-x也与p互质，因此小于p且与p互质的数之和为phi(x)\*x/2;

⑥N>1，不大于N且和N互素的所有正整数的和是 1/2 \*N \*eular(N)。

**7.2.1.3 欧拉筛模板**

1. **void** euler(**int** n)
2. {
3. vis[1]=**true**;
4. **int** cnt=0;
5. **for**(**int** i=2;i<=n;i++)
6. {
7. **if**(!vis[i]) prime[++cnt]=i;
8. **for**(**int** j=1;prime[j]\*i<=n;j++)
9. {
10. vis[i\*prime[j]]=**true**;
11. **if**(i%prime[j]==0) **break**;
12. }
13. }
14. }

**7.2.1.4 欧拉函数模板**

1. **int** euler(**int** x)
2. {
3. **int** ans=x;
4. **for**(**int** i=2;i\*i<=x;i++)//模拟（1-1/p1）\*（1-1/p2）……过程，pi为x的质因数
5. {
6. **if**(x%i==0)
7. {
8. ans=ans-ans/i;
9. }
10. **while**(x%i==0)
11. {
12. x/=i;
13. }
14. }
15. **if**(x>1) ans=ans-ans/x;//若x不为1，说明仍有一个质因数没有乘
16. **return** ans;
17. }

**7.2.5 线性求欧拉函数模板**

1. **void** euler(**int** n)
2. {
3. vis[1]=**true**;
4. phi[1]=1;
5. **int** cnt=0;
6. **for**(**int** i=2;i<=n;i++)
7. {
8. **if**(!vis[i]) prime[++cnt]=i,phi[i]=i-1;
9. **for**(**int** j=1;prime[j]\*i<=n;j++)
10. {
11. vis[prime[j]\*i]=**true**;
12. **if**(i%prime[j]==0)
13. {
14. phi[i\*prime[j]]=phi[i]\*prime[j];
15. **break**;
16. }
17. phi[i\*prime[j]]=phi[i]\*(prime[j]-1);
18. }
19. }
20. }

### 7.2.2 经典例题

**7.2.6.1 HDU2824 The Euler function**

**题目描述**

The Euler function phi is an important kind of function in number theory, (n) represents the amount of the numbers which are smaller than n and coprime to n, and this function has a lot of beautiful characteristics. Here comes a very easy question: suppose you are given a, b, try to calculate (a)+ (a+1)+....+ (b)

**输入格式**

There are several test cases. Each line has two integers a, b (2<a<b<3000000).

**输出格式**

Output the result of (a)+ (a+1)+....+ (b)

**输入样例**

3 100

**输出样例**

3042

**思路：**线性求欧拉函数模板题

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **int** N=3e6+10;
5. **int** phi[N];
6. **bool** vis[N];
7. **int** prime[N];
8. **int** sum[N];
9. **void** eular\_()//欧拉筛模板
10. {
11. memset(vis,**true**,**sizeof**(vis));
12. memset(phi,0,**sizeof**(phi));
13. **int** cnt=1;
14. vis[1]=**false**;
15. phi[1]=1;//特判1,gcd(1,1)=1
16. **for**(**int** i=2;i<=N-5;i++)
17. {
18. **if**(vis[i])
19. {
20. prime[cnt++]=i;
21. phi[i]=i-1;
22. }
23. **for**(**int** j=1;j<cnt&&prime[j]\*i<=N-5;j++)
24. {
25. vis[prime[j]\*i]=**false**;
26. **if**(i%prime[j]==0)
27. {
28. phi[prime[j]\*i]=phi[i]\*prime[j];
29. **break**;
30. }
31. phi[prime[j]\*i]=phi[i]\*phi[prime[j]];
32. //若i为p[j]的倍数，则i=k\*p[j]，在和p[j+1]相乘后得出的x=i\*p[j+1]=p[j]\*k\*p[j+1]
33. //则在i=k\*p[j+1]时，由于j从小到大，必定经过之前的p[j]，会重复计算
34. }
35. }
36. }
37. **int** main()
38. {
39. **int** a,b;
40. eular\_();
41. **while**(cin>>a>>b)
42. {
43. ll ans=0;
44. **for**(**int** i=a;i<=b;i++) ans+=phi[i];
45. cout<<ans<<endl;
46. }
47. **return** 0;
48. }

**7.2.6.2 HDU 2588 GCD**

**题目描述**

The greatest common divisor GCD(a,b) of two positive integers a and b,sometimes written (a,b),is the largest divisor common to a and b,For example,(1,2)=1,(12,18)=6.

(a,b) can be easily found by the Euclidean algorithm. Now Carp is considering a little more difficult problem:

Given integers N and M, how many integer X satisfies 1<=X<=N and (X,N)>=M.

**输入格式**

The first line of input is an integer T(T<=100) representing the number of test cases. The following T lines each contains two numbers N and M (2<=N<=1000000000, 1<=M<=N), representing a test case.

**输出格式**

For each test case,output the answer on a single line.

**输入样例**

3

1 1

10 2

10000 72

**输出样例**

1

6

260

**思路：**当M为1 时，易知答案为N，当M>1时，可以确定满足条件的数一定时N大于M的约数，如果x不能整除N，则gcd（x,N）=1<M，我们知道N的约数（设为x）和N的最大公约数为x，那是否本题只有N的约数满足条件呢？显然不是，数x\*k（k为满足条件的某些数）与N的约数同样为x，倒退一下，当gcd（N，x\*k）=x时，有gcd（N/x，k）=1，即k满足与N/x互质时满足N与x\*k的最大公约数为x，于是本题就变成了就1-N/x内，与N/x互质的数的个数，用欧拉函数求即可

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **int** N=1e5;
5. ll eular(ll x)
6. {
7. ll ans=x;
8. **for**(**int** i=2;i\*i<=x;i++)
9. {
10. **if**(x%i==0)
11. {
12. ans=ans-ans/i;
13. }
14. **while**(x%i==0)
15. {
16. x/=i;
17. }
18. }
19. **if**(x>1) ans=ans-ans/x;
20. **return** ans;
21. }
22. **int** main()
23. {
24. **int** t;
25. cin>>t;
26. **while**(t--)
27. {
28. ll ans=0;
29. ll n,m;
30. cin>>n>>m;
31. **int** i;
32. **for**(i=1;i\*i<n;i++)
33. {
34. **if**(n%i==0)
35. {
36. **if**(i>=m)
37. {
38. ans+=eular(n/i);
39. }
40. **if**(n/i>=m)
41. {
42. ans+=eular(i);
43. }
44. }
45. }
46. **if**(i\*i==n&&i>=m) ans+=eular(i);
47. cout<<ans<<endl;
48. }
49. **return** 0;
50. }

**7.2.6.3 HDU phi**

**题目描述**

给出若干个正整数n，请你求出最小的m，使得φ(m)≥n。

**输入格式**

本题有多组输入。

第一行一个正整数T表示数据组数

接下来T行每行一个正整数n

数据保证1≤T≤104,1≤n≤106。

**输出格式**

共T行，每行一个数代表对应的答案

**输入样例**

5

1

2

3

4

5

**输出样例**

1

3

5

5

7

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** ll N=5e6+10;
5. **int** phi[N-5];
6. **bool** vis[N-5];
7. **int** prime[N-5];
8. **struct** node
9. {
10. **int** pos;
11. **int** val;
12. **friend** **bool** operator < (**const** node x,**const** node y)//对输入进行处理，优先处理n小的输入并记录
13. {
14. **return** x.val<y.val;
15. }
16. }node[10005];
17. **int** ans[10005];
18. **int** t;
19. **void** euler()
20. {
21. **int** now=1;
22. memset(vis,**false**,**sizeof**(vis));
23. memset(phi,0,**sizeof**(phi));
24. vis[1]=**true**;
25. phi[1]=1;
26. **int** cnt=0;
27. **for**(**int** i=1;i<=N-10;i++)
28. {
29. **if**(!vis[i])
30. {
31. prime[++cnt]=i;
32. phi[i]=i-1;
33. }
34. **for**(**int** j=1;j<=cnt&&prime[j]\*i<=N-10;j++)
35. {
36. vis[prime[j]\*i]=**true**;
37. **if**(i%prime[j]==0)
38. {
39. phi[i\*prime[j]]=phi[i]\*prime[j];
40. **break**;
41. }
42. phi[i\*prime[j]]=phi[i]\*phi[prime[j]-1];
43. }
44. **while**(phi[i]>=node[now].val&now<=t)
45. {
46. ans[node[now].pos]=i;
47. now++;
48. }
49. **if**(now>t) **return** ;
50. }
51. }
52. **int** main()
53. {
54. euler();
55. cin>>t;
56. **for**(**int** i=1;i<=t;i++)
57. {
58. cin>>node[i].val;
59. node[i].pos=i;
60. }
61. sort(node+1,node+t+1);
62. euler();
63. **for**(**int** i=1;i<=t;i++)
64. cout<<ans[i]<<endl;
65. **return** 0;
66. }

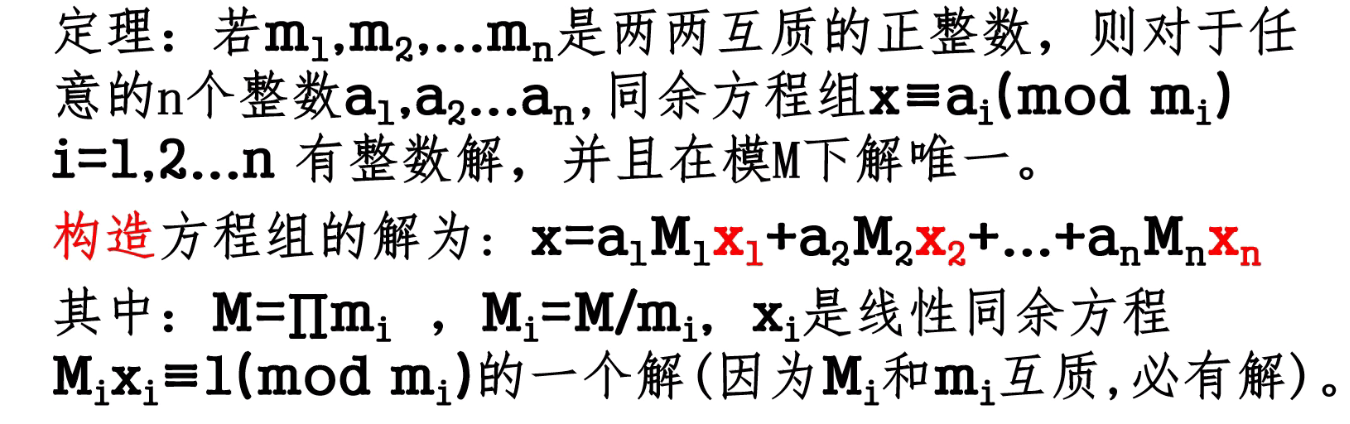
## 7.3中国剩余定理与拓展中国剩余定理

### 7.3.1 中国剩余定理

**7.3.1.1概述**

用于求解模数互质的线性同余方程组

**7.3.1.2 解题方法**



**7.3.1.3 代码模板**

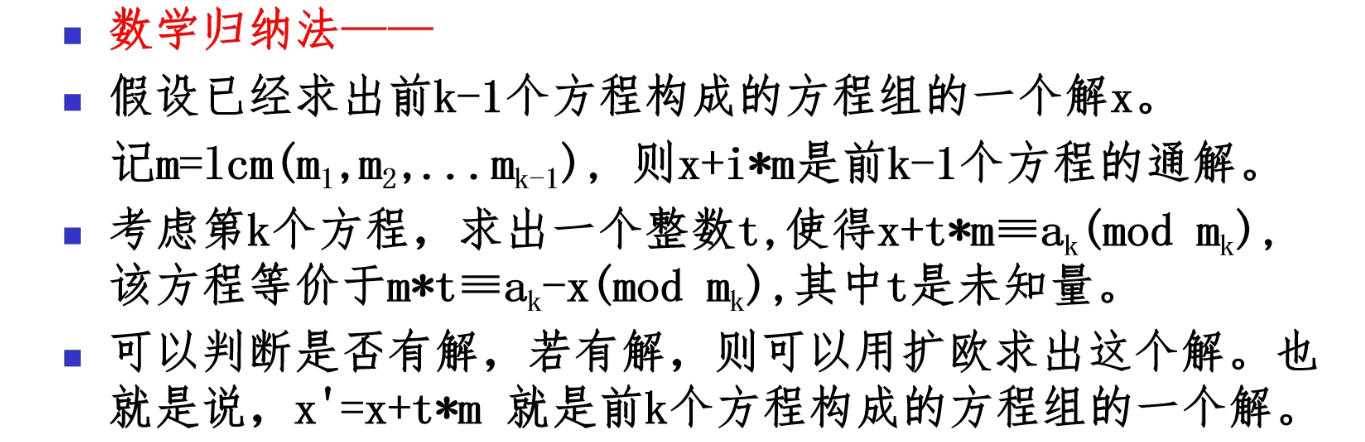
1. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
2. {
3. **if**(b==0)
4. {
5. x=1;
6. y=0;
7. **return** a;
8. }
9. **int** ans=exgcd(b,a%b,x,y);
10. **int** tmp=x;
11. x=y;
12. y=tmp-(a/b)\*y;
13. **return** ans;
14. }
15. **int** crt(**int** n)
16. {
17. **int** ans=0;
18. **for**(**int** i=1;i<=n;i++)
19. {
20. **int** x,y;
21. exgcd(MOD/mod[i],mod[i],x,y);
22. ans=(ans+MOD/mod[i]\*x\*yu[i])%MOD;
23. }
24. **return** (ans%MOD+MOD)%MOD;
25. }

### 7.3.2 扩展中国剩余定理

**7.3.2.1 概述**

**用于求解模数不互质的线性同余方程组**

**7.3.2.2 解题方法**



**7.3.2.3 代码模板**

1. #include<bits/stdc++.h>
2. #define int long long
3. **using** **namespace** std;
4. **const** **int** N=1e5+5;
5. **int** x,y;
6. **int** n;
7. **int** mod[N],yu[N];
8. **int** qmul(**int** a,**int** n,**int** mod)
9. {
10. **int** res=0;
11. **while**(n)
12. {
13. **if**(n&1) res=(res+a)%mod;
14. a=(a+a)%mod;
15. n>>=1;
16. }
17. **return** res;
18. }
19. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
20. {
21. **if**(b==0)
22. {
23. x=1;
24. y=0;
25. **return** a;
26. }
27. **int** ans=exgcd(b,a%b,x,y);
28. **int** tmp=x;
29. x=y;
30. y=tmp-(a/b)\*y;
31. **return** ans;
32. }
33. **int** ex\_crt(**int**\* mod,**int** \*yu)
34. {
35. **int** ans=yu[1];
36. **int** m=mod[1];
37. **for**(**int** i=2;i<=n;i++)
38. {
39. **int** tmp=(yu[i]%mod[i]-ans%mod[i]+mod[i])%mod[i];//yu[i]-x=t\*m(mod mod[i]) 求t
40. **int** gcd=exgcd(m,mod[i],x,y);
41. **if**(tmp%gcd) **return** -1;
42. **int** times=tmp/gcd;
43. **int** bg=mod[i]/gcd;
44. x=qmul(x,times,bg);//这里乘法可能会溢出，所以写个快速加法
45. ans=ans+m\*x;
46. m=m/gcd\*mod[i];
47. ans=(ans%m+m)%m;
48. }
49. **return** (ans%m+m)%m;
50. }
51. **signed** main()
52. {
53. cin>>n;
54. **for**(**int** i=1;i<=n;i++) cin>>mod[i]>>yu[i];
55. cout<<ex\_crt(mod,yu)<<endl;
56. **return** 0;
57. }

### 7.3.4经典例题

**7.3.4.1洛谷P1495 中国剩余定理**

**题目描述**

自从曹冲搞定了大象以后，曹操就开始捉摸让儿子干些事业，于是派他到中原养猪场养猪，可是曹冲满不高兴，于是在工作中马马虎虎，有一次曹操想知道母猪的数量，于是曹冲想狠狠耍曹操一把。举个例子，假如有 1616 头母猪，如果建了 33 个猪圈，剩下 11 头猪就没有地方安家了。如果建造了 55 个猪圈，但是仍然有 11 头猪没有地方去，然后如果建造了 77 个猪圈，还有 22 头没有地方去。你作为曹总的私人秘书理所当然要将准确的猪数报给曹总，你该怎么办？

**输入格式**

第一行包含一个整数 nn——建立猪圈的次数，接下来 n 行，每行两个整数ai，bi表示建立了ai个猪圈，有bi头猪没有去处，规定ai，aj­互质

**输出格式**

输出包含一个正整数，即为曹冲至少养母猪的数目。

**输入样例**

3

3 1

5 1

7 2

**输出样例**

16

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. ll mod[15];
5. ll yu[15];
6. ll MOD=1;
7. ll exgcd(ll a,ll b,ll &x,ll &y)
8. {
9. **if**(b==0)
10. {
11. x=1;
12. y=0;
13. **return** a;
14. }
15. ll ans=exgcd(b,a%b,x,y);
16. ll tmp=x;
17. x=y;
18. y=tmp-(a/b)\*y;
19. **return** ans;
20. }
21. ll crt(**int** n)
22. {
23. ll ans=0;
24. **for**(**int** i=1;i<=n;i++)
25. {
26. ll x,y;
27. exgcd(MOD/mod[i],mod[i],x,y);
28. ans=(ans+MOD/mod[i]\*x\*yu[i])%MOD;
29. }
30. **return** ans;
31. }
32. **int** main()
33. {
34. **int** n;
35. cin>>n;
36. **for**(**int** i=1;i<=n;i++)
37. {
38. cin>>mod[i]>>yu[i];
39. }
40. **for**(**int** i=1;i<=n;i++)  MOD\*=mod[i];
41. cout<<(crt(n)+MOD)%MOD;
42. **return** 0;
43. }

**7.3.4.2 HDU3579 Hello Kiki**

**题目背景**

有一天，我在超市购物。有一个收银员在认真地数硬币，这时一个小孩子跑过来唱着 "门前大桥下游过一群鸭，快来快来数一数，二四六七八"。然后收银员把数好的硬币放回原处，再数一遍......

你好，琪琪是个可爱的女孩，她喜欢用不同的方式进行计数。例如，当她在数X个硬币时，她会数N次。每次她都把硬币分成几个同样大小的组，并在纸条上写下组的大小Mi和剩余硬币的数量Ai。

有一天，琪琪的父亲发现了她的纸条，他想知道琪琪在数多少硬币。

**输入格式**

The first line is T indicating the number of test cases.

Each case contains N on the first line, Mi(1 <= i <= N) on the second line, and corresponding Ai(1 <= i <= N) on the third line.

All numbers in the input and output are integers.

1 <= T <= 100, 1 <= N <= 6, 1 <= Mi <= 50, 0 <= Ai < Mi

**输出格式**

For each case output the least positive integer X which Kiki was counting in the sample output format. If there is no solution then output -1.

**输入样例**

2

2

14 57

5 56

5

19 54 40 24 80

11 2 36 20 76

**输出样例**

Case 1: 341

Case 2: 5996

**题解：**

1. #include<bits/stdc++.h>
2. **typedef** **long** **long** ll;
3. **using** **namespace** std;
4. ll M[10];
5. ll A[10];
6. ll ex\_gcd(ll a,ll b,ll &x,ll &y)
7. {
8. **if**(b==0)
9. {
10. x=1;
11. y=0;
12. **return** a;
13. }
14. ll ans=ex\_gcd(b,a%b,x,y);
15. ll tmp=x;
16. x=y;
17. y=tmp-(a/b)\*y;
18. **return** ans;
19. }
20. ll ex\_crt(ll\* yu,ll\* mod,ll n)
21. {
22. ll ans=yu[1];
23. ll m=mod[1];
24. ll x,y;
25. ll now;
26. **for**(ll i=2;i<=n;i++)
27. {
28. now=(yu[i]%mod[i]-ans%mod[i]+mod[i])%mod[i];//t\*m=ak-x(mod m)
29. //now=ak-x  满足 now<m
30. ll gcd=ex\_gcd(m,mod[i],x,y);//扩欧求方程的解
31. **if**(now%gcd) **return** -1;//判断t是否存在
32. ll k=now/gcd;
33. x\*=k;//求的t'为t'\*m=gcd (mod m)的解，需要乘以倍数
34. **int** s=mod[i]/gcd;
35. x=(x%s+mod[i]/gcd)%(mod[i]/gcd);//x=x+t\*m更新x
36. ans=ans+x\*m;//
37. m=m\*mod[i]/gcd;
38. }
39. **return** (ans+m)%m?(ans+m)%m:m;
40. }
41. **int** main()
42. {
43. ll t;
44. cin>>t;
45. **for**(ll i=1;i<=t;i++)
46. {
47. memset(A,0,**sizeof**(A));
48. memset(M,0,**sizeof**(M));
49. ll n;
50. cin>>n;
51. **for**(ll j=1;j<=n;j++) cin>>M[j];
52. **for**(ll j=1;j<=n;j++) cin>>A[j];
53. cout<<"Case "<<i<<": "<<ex\_crt(A,M,n)<<endl;
54. }
55. **return** 0;
56. }

## 7.4 线性同余方程

### 7.4.1 概述

a,b是整数，形如ax≡b(mod m)，且x是未知整数的同余式称为一元线性同余方程。

容易发现，可以化成ax+my=b的形式，于是可以用扩欧或欧拉定理求解，当(a,m)|b时有解。

### 7.4.2经典例题

* + - 1. **P1082 [NOIP2012 提高组] 同余方程**

**题目描述**

求关于x的同余方程 ax≡1(mod b) 的最小正整数解。

**输入格式**

一行，包含两个正整数 a,b用一个空格隔开。

**输出格式**

一个正整数 x0，即最小正整数解。输入数据保证一定有解。

**输入输出样例**

**输入**

3 10

**输出**

7

**题解1.欧拉定理**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **int** qpow(**int** a,**int** n,**int** b)
5. {
6. **int** ans=1;
7. **while**(n)
8. {
9. **if**(n&1) ans=ans%b\*a%b;
10. a=a%b\*a%b;
11. n>>=1;
12. }
13. **return** ans;
14. }
15. **int** euler(**int** n)
16. {
17. **int** ans=n;
18. **for**(**int** i=2;i\*i<=n;i++)
19. {
20. **if**(n%i==0) ans-=ans/i;
21. **while**(n%i==0) n/=i;
22. }
23. **if**(n>1) ans-=ans/n;
24. **return** ans;
25. }
26. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
27. {
28. **if**(b==0)
29. {
30. x=1;
31. y=0;
32. **return** a;
33. }
34. **int** ans=exgcd(b,a%b,x,y);
35. **int** tmp=x;
36. x=y;
37. y=tmp-(a/b)\*y;
38. **return** ans;
39. }
40. **signed** main()
41. {
42. **int** a,b;
43. cin>>a>>b;
44. cout<<qpow(a,euler(b)-1,b)<<endl;
45. **return** 0;
46. }

**题解2.扩欧**

#include<bits/stdc++.h>

**using** **namespace** std;

#define int long long

**int** exgcd(**int** a,**int** b,**int** &x,**int** &y)

{

**if**(b==0)

    {

        x=1;

        y=0;

**return** a;

    }

**int** ans=exgcd(b,a%b,x,y);

**int** tmp=x;

    x=y;

    y=tmp-(a/b)\*y;

**return** ans;

}

**signed** main()

{

**int** a,b;

**int** x,y;

    cin>>a>>b;

    exgcd(a,b,x,y);

    x=(x%b+b)%b;

    cout<<x<<endl;

**return** 0;

}

**7.4.2.2洛谷P1516 青蛙的约会**

**题目描述**

两只青蛙在网上相识了，它们聊得很开心，于是觉得很有必要见一面。它们很高兴地发现它们住在同一条纬度线上，于是它们约定各自朝西跳，直到碰面为止。可是它们出发之前忘记了一件很重要的事情，既没有问清楚对方的特征，也没有约定见面的具体位置。不过青蛙们都是很乐观的，它们觉得只要一直朝着某个方向跳下去，总能碰到对方的。但是除非这两只青蛙在同一时间跳到同一点上，不然是永远都不可能碰面的。为了帮助这两只乐观的青蛙，你被要求写一个程序来判断这两只青蛙是否能够碰面，会在什么时候碰面。

我们把这两只青蛙分别叫做青蛙A和青蛙B，并且规定纬度线上东经0度处为原点，由东往西为正方向，单位长度1米，这样我们就得到了一条首尾相接的数轴。设青蛙A的出发点坐标是x，青蛙B的出发点坐标是y。青蛙A一次能跳m米，青蛙B一次能跳n米，两只青蛙跳一次所花费的时间相同。纬度线总长L米。现在要你求出它们跳了几次以后才会碰面。

**输入格式**

输入只包括一行5个整数x，y，m，n，L

其中保证

0<x≠y < =2000000000，

0 < m、n < =2000000000，

0 < L < =2100000000。

**输出格式**

输出碰面所需要的天数，如果永远不可能碰面则输出一行"Impossible"。

**输入输出样例**

**输入#1**

**1 2 3 4 5**

**输出#1**

**4**

**题解**

假设在时间t后相遇，则A在x+t\*m-1(mod L)+1,B在y+t\*n-1(mod L)+1,于是就有方程

(m-n)\*t≡y-x(mod L),扩欧即可

**代码如下：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. ll exgcd(ll a,ll b,ll &x,ll &y)
5. {
6. **if**(b==0)
7. {
8. x=1;
9. y=0;
10. **return** a;
11. }
12. ll ans=exgcd(b,a%b,x,y);
13. ll tmp=x;
14. x=y;
15. y=tmp-(a/b)\*y;
16. **return** ans;
17. }
18. **int** main()
19. {
20. ll x,y,m,n,l;
21. cin>>x>>y>>m>>n>>l;
22. ll a,b;
23. ll gcd=exgcd(m-n,l,a,b);
24. **if**((y-x)%gcd)
25. {
26. cout<<"Impossible"<<endl;
27. **return** 0;
28. }
29. **else**
30. {
31. ll times=(y-x)/gcd;
32. ll d=abs(l/gcd);
33. cout<<((a\*times)%d+d)%d<<endl;
34. }
35. **return** 0;
36. }

## 7.5逆元

### 7.5.1 概述

如果一个线性同余方程ax≡1（mod b）,则称x是a mod b的逆元，记作a-1。

### 7.5.2 扩欧求逆元

1. **void** exgcd(**int** a, **int** b, **int**& x, **int**& y) {
2. **if** (b == 0) {
3. x = 1, y = 0;
4. **return**;
5. }
6. exgcd(b, a % b, y, x);
7. y -= a / b \* x;
8. }

### 7.5.3 费马小定理求逆元

**费马小定理：**若存在整数a,p且gcd(a,p)=1,即二者互为质数，则有a^(p-1)≡1(mod p)。

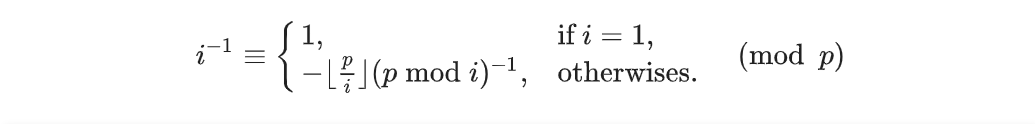
此时 a^(p-2)是a mod p的逆元

由于p一般很大，用快速幂来求

1. ll qpow(ll a,ll n)
2. {
3. ll ans=1;
4. **while**(n)
5. {
6. **if**(n%2) ans=ans\*a%mod;
7. a=a\*a%mod;
8. n/=2;
9. }
10. **return** ans;
11. }

### 7.5.4 线性求逆元

**7.5.4.1 求1-n的逆元**



1. inv[1] = 1;
2. **for** (**int** i = 2; i <= n; ++i) {
3. inv[i] = (**long** **long**)(p - p / i) \* inv[p % i] % p;
4. }

**7.5.4.2 求给定n个数的逆元**

首先计算n个数的前缀积，记为prei，然后使用快速幂或扩展欧几里得法计算pren 的逆元，记为Invn 。

因为Invn是n个数的积的逆元，所以当我们把它乘上an时，就会和an的逆元抵消，于是就得到了a1 到an-1的积逆元，记为Invn-1。

同理我们可以依次计算出所有的Invi，于是ai-1就可以用prei-1\*Invi求得。

所以我们就在O(n+logp)的时间内计算出了n个数的逆元。

1. **void** init()
2. {
3. pre[0] = 1;
4. **for** (**int** i = 1; i <= n; ++i) pre[i] = pre[i - 1] \* a[i] % p;
5. inv[n] = qpow(pre[n], p - 2);
6. // 当然这里也可以用 exgcd 来求逆元,视个人喜好而定.
7. **for** (**int** i = n; i >= 1; --i) inv[i - 1] = inv[i] \* a[i] % p;
8. **for** (**int** i = 1; i <= n; ++i) inv[i] = inv[i] \* pre[i - 1] % p;
9. }

### 7.5.5 经典例题

**7.5.5.1 LiberOJ #161 乘法逆元2**

**题目描述**

这可能是一道模板题。

给定 个正整数 ，求每个数在模 意义下的乘法逆元。

提示：请使用高效的读入方式。

**输入格式**

第一行一个整数n。

第二行n个整数ai。

**输出格式**

一行一个数，表示Σi=1nai\*998244353n-i（mod p）

**输入样例**

5

4 7 8 12 123456

**输出样例**

650798912

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** ll mod=1e9+7;
5. **const** ll N=998244353;
6. ll a[5000005];
7. ll pre[5000005];
8. ll qpow(ll a,ll n)
9. {
10. ll ans=1;
11. **while**(n)
12. {
13. **if**(n&1) ans=(ans%mod)\*(a%mod)%mod;
14. a=a\*a%mod;
15. n/=2;
16. }
17. **return** ans;
18. }
20. **int** read() {
21. **int** x=0,f=1;
22. **char** c=getchar();
23. **while**(c<'0'||c>'9'){**if**(c=='-') f=-1;c=getchar();}
24. **while**(c>='0'&&c<='9') x=x\*10+c-'0',c=getchar();
25. **return** x\*f;
26. }
27. ll inv[5000005];
28. **int** main()
29. {
30. **int** n;
31. scanf("%d",&n);
32. pre[0]=1;
33. ll ans=0;
34. **for**(**int** i=1;i<=n;i++)
35. {
36. a[i]=read();
37. pre[i]=pre[i-1]\*a[i]%mod;
38. }
39. inv[n]=qpow(pre[n],mod-2);
40. **for**(**int** i=n;i>=1;i--) inv[i-1]=inv[i]%mod\*a[i]%mod;
41. **for**(**int** i=1;i<=n;i++)
42. {
43. ans=ans\*N%mod+inv[i]\*pre[i-1]%mod;
44. }
45. printf("%lld\n",ans%mod);
46. **return** 0;
47. }

## BSGS与扩展BSGS

### 7.6.1 概述

主要用于解决an=b(mod p) 问题，求解最小非负整数n，使其满足左式

普通BSGS求解a与p互质的情况，扩展BSGS可求解不互质情况

### 7.6.2 普通BSGS

**7.6.2.1解题步骤**

1.取m=ceil(sqrt(p))（向上取整），若解存在，可令n=i\*m-j，即数对(i,j)存在

2.an = b ( mod p )⬄ai\*m-j = b( mod p )⬄ai\*m = b\*aj ( mod p )

3.再0-m范围内枚举b\*aj的值，并用hash表存储j

4.在1-m范围内枚举ai\*m的值，在hash表中查找，若找到，则输出i\*m-j，反之无答案

**7.6.2.2代码模板**

1. #include<bits/stdc++.h>
2. #define int long long
3. **using** **namespace** std;
4. **int** a,p,b;
5. unordered\_map<**int**,**int**> mp;
6. **int** qpow(**int** a,**int** n,**int** mod)
7. {
8. **int** ans=1;
9. **while**(n)
10. {
11. **if**(n&1) ans=ans\*a%mod;
12. a=a\*a%mod;
13. n>>=1;
14. }
15. **return** ans;
16. }
17. **int** bsgs(**int** a,**int** b,**int** p)
18. {
19. a%=p;
20. b%=p;
21. **if**(1==b%p) **return** 0;  //特判结果为0的情况
22. **int** m=ceil(sqrt(p));
23. **int** baj=1;
24. **for**(**int** i=0;i<=m;i++)
25. {
26. **if**(i==0)
27. {
28. baj=b%p;
29. mp[baj]=i;
30. **continue**;
31. }
32. baj\*=a;
33. baj%=p;
34. mp[baj]=i;
35. }
36. **int** am=qpow(a,m,p);
37. **int** tmp=1;
38. **for**(**int** i=1;i<=m;i++)//由于假设n为i\*m-j，若i从0开始枚举则有可能输出负数
39. {
40. tmp\*=am;
41. tmp%=p;
42. **if**(mp.count(tmp)) **return** i\*m-mp[tmp];
43. }
44. **return** -1;
45. }
46. **signed** main()
47. {
48. cin.tie(0);
49. cout.tie(0);
50. ios::sync\_with\_stdio(0);
51. **while**(cin>>a>>p>>b)
52. {
53. mp.clear();
54. **if**(a==0&&p==0&&b==0) **break**;
55. **int** ans=bsgs(a,b,p);
56. **if**(ans==-1) cout<<"No Solution"<<endl;
57. **else** cout<<ans<<endl;
58. }
59. **return** 0;
60. }

### 7.6.3 扩展BSGS

**7.6.3.1 解题步骤**

1.当a与p互质时，套用普通BSGS即可

2.当a与p不互质时，设gcd为a与p的最大公约数，对于方程an=b ( mod p ) ，一定有 a/gcd \* an-1 = b/gcd ( mod p/gcd )存在，因此若b不为gcd的倍数，则方程无解

3.对于新方程 令 b/ gcd /( a / gcd) = b’ , p/gcd = p’ , 用逆元处理b’ , 递归求解即可

**7.6.3.2代码模板**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **const** **int** inf=1e8;
5. **int** a,p,b,x,y;
6. **int** qpow(**int** a,**int** n,**int** mod)
7. {
8. **int** ans=1;
9. **while**(n)
10. {
11. **if**(n&1) ans=ans\*a%mod;
12. a=a\*a%mod;
13. n>>=1;
14. }
15. **return** ans;
16. }
17. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
18. {
19. **if**(!b)
20. {
21. x=1;
22. y=0;
23. **return** a;
24. }
25. **int** ans=exgcd(b,a%b,x,y);
26. **int** tmp=x;
27. x=y;
28. y=tmp-(a/b)\*y;
29. **return** ans;
30. }
31. unordered\_map<**int**,**int**> mp;
32. **int** bsgs(**int** a,**int** b,**int** p)
33. {
34. mp.clear();
35. a=a%p;
36. //b=(b%p+p)%p;
37. **if**(1%p==b%p) **return** 0;//特判结果为0的情况
38. **int** m=ceil(sqrt(p));
39. **int** baj=1;
40. **for**(**int** i=0;i<=m;i++)
41. {
42. **if**(i==0)
43. {
44. baj=b%p;
45. mp[baj]=i;
46. **continue**;
47. }
48. baj\*=a;
49. baj%=p;
50. mp[baj]=i;
51. }
52. **int** am=qpow(a,m,p);
53. **int** tmp=1;
54. **for**(**int** i=1;i<=m;i++)
55. {
56. tmp\*=am;
57. tmp%=p;
58. **if**(mp.count(tmp)) **return** i\*m-mp[tmp];
59. }
60. **return** -1;
61. }
62. **int** exbsgs(**int** a,**int** b,**int** p)
63. {
64. b=(b%p+p)%p;  //令b为正
65. **if**(1%p==b%p) **return** 0;
66. **int** gcd=exgcd(a,p,x,y);
67. **if**(gcd>1)
68. {
69. **if**(b%gcd) **return** -inf;
70. exgcd(a/gcd,p/gcd,x,y);
71. **int** a2=a;
72. **int** b2=b/gcd\*x%(p/gcd);
73. **int** p2=p/gcd;
74. **return** exbsgs(a2,b2,p2)+1;
75. }
76. **return** bsgs(a,b,p);
77. }
78. **signed** main()
79. {
80. ios::sync\_with\_stdio(0);
81. **while**(cin>>a>>p>>b)
82. {
83. **if**(!a&&!b&&!p) **break**;
84. **int** ans=exbsgs(a,b,p);
85. **if**(ans<0) cout<<"No Solution"<<endl;
86. **else** cout<<ans<<endl;
87. }
88. **return** 0;
89. }

### 7.6.4 经典例题

**7.6.4.1 洛谷P4861 按钮**

**题目描述**

Ada被关在了一个房间里。房间的铁门上有一个按钮，还有一个显示屏显示着“1”。

旁边还有一行小字：“这是一个高精度M进制计算器，每按一次按钮，屏幕上的数便会乘以K。当个位数再次变为1时，门就开了。”

由于Ada急于出去，所以你要在1s之内求出她的最小按键次数**。**

**输入格式**

一行，两个整数m和k

**输出格式**

一行一个数字，表示最小按键次数。

如果无论Ada按多少次都无法让门打开，输出"Let's go Blue Jays!"（不含引号）。

**输入样例1**

**11 2**

**输出样例1**

**10**

**输入样例2**

**6 26**

**输出样例2**

**Let's go Blue Jays!**

**思路**

**若最后答案存在，则有两种表达方式，kn和m\*x+1，于是可以列出方程**

**kn = 1 ( mod m ) ,套用由于答案不能为0，于是特判一下原模板中输出0的情况即可**

**题解**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **int** m,k;
5. unordered\_map<**int**,**int**> mp;
6. **int** exgcd(**int** a,**int** b,**int** &x,**int** &y)
7. {
8. **if**(b==0)
9. {
10. x=1;
11. y=0;
12. **return** a;
13. }
14. **int** ans=exgcd(b,a%b,x,y);
15. **int** tmp=x;
16. x=y;
17. y=tmp-(a/b)\*y;
18. **return** ans;
19. }
20. **int** qpow(**int** a,**int** n,**int** mod)
21. {
22. **int** ans=1;
23. **while**(n)
24. {
25. **if**(n&1) ans=ans%mod\*a%mod;
26. a=a%mod\*a%mod;
27. n>>=1;
28. }
29. **return** ans;
30. }
31. **int** bsgs(**int** a,**int** b,**int** p)
32. {
33. a%=p;
34. b%=p;
35. **int** m=ceil(sqrt(p));
36. **int** tmp;
37. **for**(**int** i=0;i<=m;i++)
38. {
39. **if**(i==0)
40. {
41. tmp=b%p;
42. mp[tmp]=i;
43. **continue**;
44. }
45. tmp\*=a;
46. tmp%=p;
47. mp[tmp]=i;
48. }
49. **int** am=qpow(a,m,p);
50. **int** now=1;
51. **for**(**int** i=1;i<=m;i++)
52. {
53. now\*=am;
54. now%=p;
55. //cout<<"now="<<now<<endl;
56. **if**(mp.count(now)&&i\*m-mp[now]==0) **continue**;  //特判结果为0的情况
57. **else** **if**(mp.count(now)) **return** i\*m-mp[now];
58. }
59. **return** -1;
60. }
61. **int** x,y;
62. **signed** main()
63. {
64. cin>>m>>k;
65. **int** gcd=exgcd(m,k,x,y);
66. **if**(gcd!=1)
67. {
68. cout<<"Let's go Blue Jays!"<<endl;
69. **return** 0;
70. }
71. **int** ans=bsgs(k,1,m);
72. **if**(ans==-1) cout<<"Let's go Blue Jays!"<<endl;
73. **else** cout<<ans<<endl;
74. **return** 0;
75. }

## 7.7 高斯消元法

### 7.1.1.概述

高斯消元法(Gaussian elimination)是求解线性方阵组的一种算法，它也可用来求矩阵的秩，以及求可逆方阵的逆矩阵。它通过逐步消除未知数来将原始线性系统转化为另一个更简单的等价的系统。它的实质是通过初等行变化(Elementary row operations)，将线性方程组的增广矩阵转化为行阶梯矩阵(row echelon form).

### 7.1.2.模板

**以洛谷P3389 高斯消元法为例**

**题目描述**

给定一个线性方程组，对其求解

**输入格式**

第一行一个正整数n

第二至n+1行，每行n+1个整数，为a1,a2……和b，代表一组方程

**输出格式**

共n行，每行一个数，第i行为xi（保留2位小数）

如果不存在唯一解，在第一行输出“No Solution”.

**输入样例**

3

1 3 4 5

1 4 7 3

9 3 2 2

**输出样例**

-0.97

5.18

-2.39

**题解**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **const** **int** N=105;
5. **double** mp[N][N];
6. **int** n;
7. **signed** main()
8. {
9. cin>>n;
10. **for**(**int** i=1;i<=n;i++)
11. {
12. **for**(**int** j=1;j<=n+1;j++)
13. {
14. cin>>mp[i][j];
15. }
16. }
17. **for**(**int** i=1;i<=n;i++)//列主元
18. {
19. **double** maxx=mp[i][i];//寻找一列中最大的元素，将此行放在最前面，防止主对角线为mp[i][i]为0
20. **int** pos=i;
21. **for**(**int** j=i+1;j<=n;j++)
22. {
23. **if**(mp[j][i]>maxx)
24. {
25. pos=j;
26. maxx=mp[j][i];
27. }
28. }
29. **if**(pos!=i)
30. {
31. **for**(**int** j=i;j<=n+1;j++)
32. {
33. **double** k=mp[pos][j];
34. mp[pos][j]=mp[i][j];
35. mp[i][j]=k;
36. }
37. }
38. **for**(**int** j=i+1;j<=n;j++)//将第i列的元素置0
39. {
40. **double** tmp=1.0\*mp[j][i]/mp[i][i];
41. **for**(**int** k=i;k<=n+1;k++)
42. {
43. mp[j][k]-=tmp\*mp[i][k];
44. }
45. }
46. }
47. **bool** flag=**true**;
48. **for**(**int** i=n;i>=1;i--)
49. {
50. **for**(**int** j=i+1;j<=n;j++)
51. {
52. mp[i][n+1]-=mp[i][j]\*mp[j][n+1];//mp[j][n+1]存储的是xj的值
53. }
54. **if**(mp[i][i]==0)//当主对角线有元素为0时无解
55. {
56. cout<<"No Solution"<<endl;
57. flag=**false**;
58. **break**;
59. }
60. mp[i][n+1]/=mp[i][i];//主对角线不一定为1，将结果进行处理
61. }
62. **if**(flag)
63. **for**(**int** i=1;i<=n;i++)
64. {
65. printf("%.2lf\n",mp[i][n+1]);
66. }
67. **return** 0;
68. }

### 7.7.3.经典例题

**7.7.3.1 洛谷P2455 线性方程组**

**题目描述**

已知n元线性方程组，求解方程组

**输入格式**

第一行输入未知数个数n

接下来n行，每行n+1个整数，表示每一个方程的系数及方程右边的值

**输入输出样例**

**输出格式**

如果有唯一解，则输出解，保留两位小数

无解输出-1

无穷多解输出0

**输入样例**

3

2 -1 1 1

4 1 -1 5

1 1 1 0

**输出样例**

x1=1.00

x2=0.00

x3=-1.00

**思路**

**本题比模板题多了判断无解与无穷解的情况**

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **const** **double** eps=1e-8;
5. **int** n;
6. **const** **int** N=55;
7. **double** mp[N][N];
8. **bool** flag=**true**;
9. **void** Gauss(**double** mp[][N])
10. {
11. **for**(**int** i=1;i<=n;i++)
12. {
13. **double** maxx=mp[i][i];
14. **int** pos=i;
15. **for**(**int** j=i+1;j<=n;j++)
16. {
17. **if**(mp[j][i]-maxx>eps)
18. {
19. maxx=mp[j][i];
20. pos=j;
21. }
22. }
23. **if**(fabs(maxx)<=eps) **continue**;//若最大值为0，说明mp[i][i]整列为0，进入下一列
24. **if**(pos!=i) **for**(**int** j=i;j<=n+1;j++) swap(mp[pos][j],mp[i][j]);
25. **for**(**int** j=i+1;j<=n;j++)
26. {
27. **if**(fabs(mp[j][i])<=eps) **continue**;
28. **double** tmp=mp[j][i]/mp[i][i];
29. **for**(**int** k=i;k<=n+1;k++)
30. {
31. mp[j][k]-=tmp\*mp[i][k];
32. }
33. }
34. }
35. **for**(**int** i=1;i<=n;i++)//判断0,0,0,0……x 的情况
36. {
37. **if**(fabs(mp[i][n+1])>eps)
38. {
39. **bool** check0=**false**;
40. **for**(**int** j=1;j<=n;j++) **if**(fabs(mp[i][j])>eps)
41. {
42. check0=**true**;
43. **break**;
44. }
45. **if**(!check0)
46. {
47. flag=**false**;
48. cout<<"-1"<<endl;
49. **return** ;
50. }
51. }
52. }
53. flag=**true**;
54. **for**(**int** i=n;i>=1;i--)
55. {
56. **for**(**int** j=i+1;j<=n;j++)
57. {
58. mp[i][n+1]-=mp[i][j]\*mp[j][n+1];
59. }
60. **if**(fabs(mp[i][i])<=eps&&fabs(mp[i][n+1])>eps){cout<<"-1"<<endl; flag=**false**; **return** ;};//首先判断无解，再判断无穷多解
61. **if**(fabs(mp[i][i]<=eps&&fabs(mp[i][n+1])<=eps)) {cout<<"0"<<endl; flag=**false**; **return** ;}
62. mp[i][n+1]/=mp[i][i];
63. }
64. }
65. **signed** main()
66. {
67. cin>>n;
68. **for**(**int** i=1;i<=n;i++)
69. {
70. **for**(**int** j=1;j<=n+1;j++)
71. {
72. cin>>mp[i][j];
73. }
74. }
75. Gauss(mp);
76. **if**(flag)
77. **for**(**int** i=1;i<=n;i++) printf("x%d=%.2lf\n",i,mp[i][n+1]+eps);
78. **return** 0;
79. }

**7.7.3.2 acwing 883.** **高斯消元解线性方程组**

**代码**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** N=105;
4. **const** **double** eps=1e-8;
5. **double** a[N][N];
6. **void** out(**int** n)
7. {
8. **for**(**int** i=1;i<=n;i++)
9. {
10. **for**(**int** j=1;j<=n+1;j++)
11. {
12. cout<<a[i][j]<<" ";
13. }
14. cout<<endl;
15. }
16. cout<<endl;
17. }
18. **void** guass(**int** n)
19. {
20. **int** r,c;
21. **for**(r=1,c=1;c<=n;c++)
22. {
23. **int** pos=r;
24. **for**(**int** i=r+1;i<=n;i++)
25. {
26. **if**(fabs(a[pos][c])<fabs(a[i][c])) pos=i;
27. }
28. **if**(fabs(a[pos][c])<eps) **continue**;
29. **for**(**int** i=c;i<=n+1;i++) swap(a[pos][i],a[r][i]);
30. **for**(**int** i=n+1;i>=c;i--) a[r][i]/=a[r][c];
31. **for**(**int** i=r+1;i<=n;i++)
32. {
33. **if**(fabs(a[i][c])>eps)
34. {
35. **for**(**int** j=n+1;j>=c;j--)
36. {
37. a[i][j]-=a[i][c]\*a[r][j];
38. }
39. }
40. }
41. r++;
42. }
43. **if**(r<=n)
44. {
45. **for**(**int** i=r;i<=n;i++)
46. {
47. **if**(fabs(a[i][n+1])>eps)
48. {
49. cout<<"No solution"<<endl;
50. **return** ;
51. }
52. }
53. cout<<"Infinite group solutions"<<endl;
54. **return** ;
55. }
56. //out(n);
57. **for**(**int** i=n;i>=1;i--)
58. {
59. **for**(**int** j=i+1;j<=n;j++)
60. {
61. a[i][n+1]-=a[i][j]\*a[j][n+1];
62. }
63. }
64. **for**(**int** i=1;i<=n;i++) printf("%.2lf\n",a[i][n+1]);
65. }
66. **int** main()
67. {
68. **int** n;
69. cin>>n;
70. **for**(**int** i=1;i<=n;i++)
71. {
72. **for**(**int** j=1;j<=n+1;j++)
73. {
74. cin>>a[i][j];
75. }
76. }
77. guass(n);
78. **return** 0;
79. }

**7.7.3.3 acwing 884.** **高斯消元解异或线性方程组**

****

**代码**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** n;
4. **const** **int** N=105;
5. **int** a[N][N];
6. **void** out(**int** n)
7. {
8. **for**(**int** i=1;i<=n;i++)
9. {
10. **for**(**int** j=1;j<=n+1;j++)
11. {
12. cout<<a[i][j]<<" ";
13. }
14. cout<<endl;
15. }
16. cout<<endl;
17. }
18. **void** guass(**int** n)
19. {
20. **int** c,r;
21. **for**(c=1,r=1;c<=n;c++)
22. {
23. **int** pos=r;
24. **for**(**int** i=r;i<=n;i++)
25. {
26. **if**(a[i][c])
27. {
28. pos=i;
29. **break**;
30. }
31. }
32. **if**(!a[pos][c]) **continue**;
33. **for**(**int** i=c;i<=n+1;i++) swap(a[pos][i],a[r][i]);
34. **for**(**int** i=r+1;i<=n;i++)
35. {
36. **if**(a[i][c])
37. {
38. **for**(**int** j=n+1;j>=c;j--)
39. {
40. a[i][j]^=a[r][j];
41. }
42. }
43. }
44. r++;
45. }
46. //out(n);
47. **if**(r<=n)
48. {
49. **for**(**int** i=r;i<=n;i++)
50. {
51. **if**(a[i][n+1])
52. {
53. cout<<"No solution"<<endl;
54. **return** ;
55. }
56. }
57. cout<<"Multiple sets of solutions"<<endl;
58. **return** ;
59. }
60. **for**(**int** i=n;i>=1;i--)
61. {
62. **for**(**int** j=i+1;j<=n;j++)
63. {
64. a[i][n+1]^=a[i][j]\*a[j][n+1];
65. }
66. }
67. **for**(**int** i=1;i<=n;i++) cout<<a[i][n+1]<<endl;
68. }
69. **int** main()
70. {
71. cin>>n;
72. **for**(**int** i=1;i<=n;i++)
73. **for**(**int** j=1;j<=n+1;j++)
74. cin>>a[i][j];
75. guass(n);
76. **return** 0;
77. }

## 7.8 线性基

### 7.8.1.概述

线性基，是线性代数中的概念，在信息学竞赛中，前缀线性基是线性基的扩展，他们主要用于处理有关异或和的极值问题。

一组线性无关的向量即可作为一组基底，张起一个线性的向量空间，这个基底即称为线性基，利用线性基的基底进行线性运算，可表示向量空间内的所有向量，换句话说，所有向量都可以拆成基底的线性组合。

根据异或的原理，将一个数字拆成他的二进制形式，将二进制形式用向量来表示，由于一组线性无关的向量可以张起一个向量空间，因此可以考虑构造这样一组数字的二进制形式组成的线性基，在这个线性基中，通过基底的线性组合、异或运算，从而可以表达所有的异或结果。

简单来说，若一个数集 T 的值域范围为 ，那么 T 的线性基是 T 的一个子集 A={a1,a2,a3,...,an}，A 中元素相互异或而成的集合，等价于原数集 T 的元素相互异或形成异或集合.

### 7.8.2.预备知识

交换变量位置：

a = a ^ b; b = a ^ b; a = a ^ b;

如果a^b==c 则a^c==b b^c==a a^b^c==0

### 7.8.3.定义

线性基是一个数的集合，并且每个序列都拥有至少一个线性基，取线性基中若干个数异或起来可以得到原序列中的任何一个数。

### 7.8.4.二进制求线性基

1. **void** init()//求非最简线性基
2. {
3. **for**(**int** i=1;i<=n;i++)//1-n遍历数组元素
4. {
5. ll x=num[i];
6. **for**(**int** j=63;j>=0;j--)//考察元素的每一位
7. {
8. **if**(x&(1LL<<j))//如果元素的第j位为1
9. {
10. **if**(p[j]) x^=p[j];//在线性基中已经出现第j位为1时，消去当前元素//的1，保证为下三角形式,化为下三角是为了更容易异//或得出某数
11. **else**
12. {
13. p[j]=x;//否则将第j位元素置1
14. **break**;
15. }
16. }
17. }
18. }
19. **return** ;
20. }

### 7.8.5.判断某数能否由线性基表示

**思路：将该数通过线性基，看结果是否为0即可**

**代码：**

1. **bool** check(**long** **long** x)
2. {
3. **for**(**int** j=60;j>=0;j--)
4. {
5. **if**(x&(1LL<<j))
6. {
7. **if**(d[j]==0)
8. {
9. **return** **false**;
10. }
11. **else** x^=d[j];
12. }
13. }
14. **return** x==0;
15. }

### 7.8.6.求异或和最大

**思路**

**求出线性基，**由于高位为1贡献更大，每次优先异或高位，遍历线性基即可

**代码**

1. **long** **long** query\_max()
2. {
3. **long** **long** ret=0;
4. **for** (**int** i=60;i>=0;i--)
5. **if** ((ret^d[i])>ret)//贪心
6. ret^=d[i];
7. **return** ret;
8. }

### 7.8.7.求异或和最小

**思路：**求最小的线性基元素即可

**代码**

1. **long** **long** query\_min()
2. {
3. **for** (**int** i=0;i<=60;i++)
4. **if** (d[i])  **return** d[i];
5. **return** 0;
6. }

### 7.8.8.求异或和第K大

**思路：**将线性基化为最简，K用二进制表示，选出1的位，将以其为下标的线性基元素相乘即可

**代码**

1. **void** rebuild()//高斯消元化为最简线性基
2. {
3. **for** (**int** i=60;i>=0;i--)
4. **for** (**int** j=i-1;j>=0;j--)
5. **if** (d[i]&(1LL<<j))
6. d[i]^=d[j];
7. **for** (**int** i=0;i<=60;i++)
8. **if** (d[i])
9. p[cnt++]=d[i];
10. }
11. **long** **long** kthquery(**long** **long** k)
12. {
13. **int** ret=0;
14. **if** (k>=(1LL<<cnt))
15. **return** -1;
16. **for** (**int** i=60;i>=0;i--)
17. **if** (k&(1LL<<i))
18. ret^=p[i];
19. **return** ret;
20. }

### 7.8.9.经典例题：

**7.8.9.1．HDU 3949 XOR**

**Problem Description**

XOR is a kind of bit operator, we define that as follow: for two binary base number A and B, let C=A XOR B, then for each bit of C, we can get its value by check the digit of corresponding position in A and B. And for each digit, 1 XOR 1 = 0, 1 XOR 0 = 1, 0 XOR 1 = 1, 0 XOR 0 = 0. And we simply write this operator as ^, like 3 ^ 1 = 2,4 ^ 3 = 7. XOR is an amazing operator and this is a question about XOR. We can choose several numbers and do XOR operatorion to them one by one, then we get another number. For example, if we choose 2,3 and 4, we can get 2^3^4=5. Now, you are given N numbers, and you can choose some of them(even a single number) to do XOR on them, and you can get many different numbers. Now I want you tell me which number is the K-th smallest number among them.

**Input**

First line of the input is a single integer T(T<=30), indicates there are T test cases.  
For each test case, the first line is an integer N(1<=N<=10000), the number of numbers below. The second line contains N integers (each number is between 1 and 10^18). The third line is a number Q(1<=Q<=10000), the number of queries. The fourth line contains Q numbers(each number is between 1 and 10^18) K1,K2,......KQ.

**Output**

For each test case,first output Case #C: in a single line,C means the number of the test case which is from 1 to T. Then for each query, you should output a single line contains the Ki-th smallest number in them, if there are less than Ki different numbers, output -1.

**Sample input**

2

2

1 2

4

1 2 3 4

3

1 2 3

5

1 2 3 4 5

**Sanple output**

Case #1:

1

2

3

-1

Case #2:

0

1

2

3

-1

**题意**

**给出n个数，求他们任意组合的最小值**

**思路**

**求出这些数的线性基，将k二进制拆分，第i位为1，答案就异或第i个线性基，若线性基个数小于原个数，即最小值可为0，则用类似方法求k-1小即可**

**代码：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. ll p[65];//存储线性基
5. ll np[65];//存储最简线性基
6. ll num[10005];//存储原数组
7. **int** n;
8. **int** k;
9. **void** init()//求非最简线性基
10. {
11. **for**(**int** i=1;i<=n;i++)//1-n遍历数组元素
12. {
13. ll x=num[i];
14. **for**(**int** j=63;j>=0;j--)//考察元素的每一位
15. {
16. **if**(x&(1LL<<j))//如果元素的第j位为1
17. {
18. **if**(p[j]) x^=p[j];//在线性基中已经出现第j位为1时，消去当前元素的1，保证为下三角形式
19. //化为下三角是为了更容易异或得出某数
20. **else**
21. {
22. p[j]=x;//否则将第j位元素置1
23. **break**;
24. }
25. }
26. }
27. }
28. **return** ;
29. }
30. **int** ninit()//求最简线性基   化为对角矩阵
31. {
32. **int** cnt=0;
33. **for**(**int** i=0;i<=63;i++)//从右下角开始
34. {
35. **for**(**int** j=i-1;j>=0;j--)//从左往右遍历
36. {
37. **if**(p[i]&(1LL<<j)) p[i]^=p[j];//如果p[i]在除第i位为1外有第j位为1，高斯消元消去第j位的1
38. //由于p[j]的第j位必为1，因此与p[J]异或
39. }
40. }
41. **for**(**int** i=0;i<=63;i++)
42. **if**(p[i])
43. {
44. np[cnt++]=p[i];//将最简的线性基存入新数组中
45. }
46. **return** cnt;
47. }
48. ll kth(ll k,**int** z)//查询异或和第k小的函数
49. {
50. ll ans=0;
51. **for**(**int** i=0;i<=62;i++)//将k用二进制表示，逐一搜索它的每一位，若第j位为1，则将答案异p[j]
52. {
53. **if**(k&(1LL<<i)) ans^=np[i];
54. }
55. **return** ans;
56. }
57. ll q[10005];
58. **int** main()
59. {
60. **int** t;
61. cin>>t;
62. **int** command;
63. **int** now=0;
64. **while**(t--)
65. {
66. now++;
67. memset(p,0,**sizeof**(p));
68. memset(num,0,**sizeof**(num));
69. memset(np,0,**sizeof**(np));
70. cin>>n;
71. **for**(**int** i=1;i<=n;i++) cin>>num[i];
72. cin>>command;
73. init();
74. k=ninit();
75. **for**(**int** i=1;i<=command;i++) cin>>q[i];
76. cout<<"Case #"<<now<<":"<<endl;
77. **for**(**int** i=1;i<=command;i++)
78. {
79. **if**(k<n&&q[i]==1)
80. {
81. cout<<"0"<<endl;
82. **continue**;
83. }
84. **if**(k<n) q[i]--;//有重复的，异或和最小为0，由于线性基无法组合出0，因此先减
85. **if**(q[i]>=(1LL<<k))//如果查询的位次比所有的组合都要多，则输出-1
86. {
87. cout<<"-1"<<endl;
88. **continue**;
89. }
90. cout<<kth(q[i],k)<<endl;
91. }
92. }
93. **return** 0;
94. }

**7.8.9.2．洛谷P3812 线性基**

**题目描述**

给定 n个整数（数字可能重复），求在这些数中选取任意个，使得他们的异或和最大。

**输入格式**

第一行一个数n，表示元素个数

接下来一行n个数

**输出格式**

仅一行，表示答案。

**输入样例**

2

1 1

**输出样例**

1

**思路**

**遍历线性基，每次异或高位为1的线性基元素**

**代码（见下一页）**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. ll mp[65];
5. ll p[65];
6. ll np[65];
7. **int** n;
8. **void** init()//求非最简线性基（下三角）
9. {
10. memset(p,0,**sizeof**(p));
11. memset(np,0,**sizeof**(np));
12. **for**(**int** i=1;i<=n;i++)//遍历数组
13. {
14. ll x=mp[i];
15. **for**(**int** j=62;j>=0;j--)//遍历元素每一位
16. {
17. **if**(x&(1LL<<j))//若元素第j位为1
18. {
19. **if**(!p[j])//并且线性基中无对应的值
20. {
21. p[j]=x;//加入元素
22. **break**;
23. }
24. **else** x^=p[j];//否则消去第j位的1
25. }
26. }
27. }
28. }
29. **int** main()
30. {
31. cin>>n;
32. **for**(**int** i=1;i<=n;i++) cin>>mp[i];
33. init();
34. ll ans=0;
35. **for**(**int** i=62;i>=0;i--)//求最大值贪心
36. {
37. **if**((ans^p[i])>ans) ans^=p[i];
38. }
39. cout<<ans<<endl;
40. **return** 0;
41. }

**7.8.9.3．洛谷P4570[BJWC2011]元素**

**题目描述**

相传，在远古时期，位于西方大陆的 Magic Land 上，人们已经掌握了用魔法矿石炼制法杖的技术。那时人们就认识到，一个法杖的法力取决于使用的矿石。

一般地，矿石越多则法力越强，但物极必反：有时，人们为了获取更强的法力而使用了很多矿石，却在炼制过程中发现魔法矿石全部消失了，从而无法炼制出法杖，这个现象被称为“魔法抵消”。特别地，如果在炼制过程中使用超过一块同一种矿石，那么一定会发生“魔法抵消”。后来，随着人们认知水平的提高，这个现象得到了很好的解释。经过了大量的实验后，著名法师 Dmitri 发现：如果给现在发现的每一种矿石进行合理的编号（编号为正整数，称为该矿石的元素序号），那么，一个矿石组合会产生“魔法抵消”当且仅当存在一个非空子集，那些矿石的元素序号按位异或起来为零（如果你不清楚什么是异或，请参见下一页的名词解释 ）。

例如，使用两个同样的矿石必将发生“魔法抵消”，因为这两种矿石的元素序号相同，异或起来为零。并且人们有了测定魔力的有效途径，已经知道了：合成出来的法杖的魔力等于每一种矿石的法力之和。人们已经测定了现今发现的所有矿石的法力值，并且通过实验推算出每一种矿石的元素序号。

现在，给定你以上的矿石信息，请你来计算一下当时可以炼制出的法杖最多有多大的魔力。

**输入格式**

第一行包含一个正整数 N*N*，表示矿石的种类数。

接下来 N行，每行两个正整数Numberi​ 和 Magici，表示这种矿石的元素序号和魔力值。

**输出格式**

仅包含一行，一个整数代表最大的魔力值。

**输入样例**

3

1 10

2 20

3 30

**输出样例**

50

**思路**

由于线性基不唯一，本题要求最大魔力值，可优先选魔力值大的值生成线性基，贪心即可

**题解：**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **struct** node
5. {
6. ll val,magic;
7. **friend** **bool** operator < (**const** node x,**const** node y)
8. {
9. **return** x.magic>y.magic;
10. }
11. }mp[1005];
12. ll p[65];
13. **int** init(**int** n)
14. {
15. ll ans=0;
16. **for**(**int** i=1;i<=n;i++)
17. {
18. ll x=mp[i].val;
19. **for**(**int** j=62;j>=0;j--)
20. {
21. **if**(x&(1LL<<j))
22. {
23. **if**(!p[j])
24. {
25. p[j]=x;
26. ans+=mp[i].magic;
27. **break**;
28. }
29. **else** x^=p[j];
30. }
31. }
32. }
33. **return** ans;
34. }
35. **int** main()
36. {
37. **int** n;
38. cin>>n;
39. **for**(**int** i=1;i<=n;i++)
40. {
41. cin>>mp[i].val>>mp[i].magic;
42. }
43. sort(mp+1,mp+n+1);
44. ll ans=init(n);
45. cout<<ans<<endl;
46. **return** 0;
47. }

**7.8.9.4 洛谷P3265 装备购买**

**题目描述**

有n件装备，每件装备有m个属性，用向量（a1，a2，a3……am）表示，需要花费c购买，现在规定如果一件装备能由其他装备组合出，那么就不需要购买（即可由其他向量线性表示），求在买下最多装备的情况下花费最少的钱

**输入格式**

第一行两个数n，m。接下来n行，每行m个数，其中第i行描述装备i的各项属性值。接下来一行n个数，其中ci表示购买第i件装备的花费。

**输出格式**

一行两个数，第一个数表示能够购买的最多装备数量，第二个数表示在购买最多数量的装备的情况下的最小花费

**输入样例**

3 3

1 2 3

3 4 5

2 3 4

1 1 2

**输出样例**

2 2

**思路**

实数线性基+贪心

**题解**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **double** eps=1e-6;
5. **int** n,m;
6. **int** p[505];
7. **struct** node\_
8. {
9. **double** mp[505];
10. **int** c;
11. **friend** **bool** operator < (**const** node\_ x,**const** node\_ y)
12. {
13. **return** x.c<y.c;
14. }
15. }node[505];
16. **int** cnt=0;
17. **int** ans=0;
18. **void** init()
19. {
20. **for**(**int** i=1;i<=n;i++)
21. {
22. **for**(**int** j=1;j<=m;j++)
23. {
24. **if**(fabs(node[i].mp[j])>eps)//卡精度
25. {
26. **if**(!p[j])
27. {
28. cnt++;
29. ans+=node[i].c;
30. p[j]=i;
31. **break**;
32. }
33. **else**
34. {
35. **long** **double** times=node[i].mp[j]\*1.0/node[p[j]].mp[j];
36. **for**(**int** k=j;k<=m;k++)//初等行变换，化为下三角
37. {
38. node[i].mp[k]-=times\*node[p[j]].mp[k];
39. }
40. }
41. }
42. }
43. }
44. }
45. **int** main()
46. {
47. cin>>n>>m;
48. **for**(**int** i=1;i<=n;i++)
49. **for**(**int** j=1;j<=m;j++)
50. cin>>node[i].mp[j];
51. **for**(**int** i=1;i<=n;i++)
52. cin>>node[i].c;
53. sort(node+1,node+n+1);
54. memset(p,0,**sizeof**(p));
55. init();
56. cout<<cnt<<" "<<ans<<endl;
57. **return** 0;
58. }

## 7.9 排列组合数

### 7.9.1 概述

**排列数:**

从n个不同元素，任取个元素按照一定的顺序排成一列，叫做从n个不同元素中取出m个元素的一个排列；从n个不同元素中取出个元素的所有排列的个数，叫做从n个不同元素中取出m个元素的排列数，用符号(或者是)表示。

**组合数:**

从n个不同元素中，任取 个元素组成一个集合，叫做从n个不同元素中取出m个元素的一个组合；从n个不同元素中取出个元素的所有组合的个数，叫做从n个不同元素中取出m个元素的组合数。用符号来表示,组合数也常用表示

### 7.9.2各种排列

**7.9.2.1不相邻的排列**

1-n这n个自然数中选k个，这k个中任何两个数都不相邻的组合有

**7.9.2.2错位排列**

我们把错位排列问题具体化，考虑这样一个问题：

n封不同的信，编号分别是1,2,3,4,5，现在要把这五封信放在编号1,2,3,4,5的信封中，要求信封的编号与信的编号不一样。问有多少种不同的放置方法？

假设我们考虑到第n个信封，初始时我们暂时把第n封信放在第n个信封中，然后考虑两种情况的递推：

前面n-1个信封全部装错；

前面n-1个信封有一个没有装错其余全部装错。

对于第一种情况，前面n-1个信封全部装错：因为前面n-1个已经全部装错了，所以第n封只需要与前面任一一个位置交换即可，总共有种情况。

对于第二种情况，前面n-1个信封有一个没有装错其余全部装错：考虑这种情况的目的在于，若n-1个信封中如果有一个没装错，那么我们把那个没装错的与n交换，即可得到一个全错位排列情况。

其他情况，我们不可能通过一次操作来把它变成一个长度为 的错排。

于是可得错位排列的递推式为。

错位排列数列的前几项为0,1,2,9,44,265。

**7.9.2.3圆排列**

n个人全部来围成一圈，所有的排列数记为。考虑其中已经排好的一圈，从不同位置断开，又变成不同的队列。 所以有

部分圆排列公式

### 7.9.3三种求组合数的方法

**1、**

**通过预处理**

**代码**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **int** N=2005,mod=1e9+7;
4. **int** c[N][N];
5. **void** init(**int** n,**int** m)
6. {
7. **for**(**int** i=0;i<=n;i++)
8. {
9. **for**(**int** j=0;j<=i;j++)
10. {
11. **if**(j==0) c[i][j]=1;
12. **else** c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
13. }
14. }
15. }
16. **int** main()
17. {
18. init(2000,2000);
19. **int** n,a,b;
20. cin>>n;
21. **for**(**int** i=1;i<=n;i++)
22. {
23. cin>>a>>b;
24. cout<<c[a][b]<<endl;
25. }
26. **return** 0;
27. }

**2、**

**随取随用，可预处理逆元和阶乘**

**代码（见下页）**

1. #include<bits/stdc++.h>
2. #define int long long
3. **using** **namespace** std;
4. **const** **int** N=1e5+5,mod=1e9+7;
5. **int** fac[N],infac[N];
6. **int** qpow(**int** a,**int** n,**int** mod)
7. {
8. **int** ans=1;
9. **while**(n)
10. {
11. **if**(n&1) ans=ans\*a%mod;
12. a=a\*a%mod;
13. n>>=1;
14. }
15. **return** ans;
16. }
17. **void** init()
18. {
19. fac[0]=1;
20. infac[0]=1;
21. **for**(**int** i=1;i<N;i++)
22. {
23. fac[i]=fac[i-1]\*i%mod;
24. infac[i]=infac[i-1]\*qpow(i,mod-2,mod)%mod;
25. }
26. }
27. **int** C(**int** a,**int** b)
28. {
29. **return** fac[a]\*infac[b]%mod\*infac[a-b]%mod;
30. }
31. **signed** main()
32. {
33. **int** n;
34. cin>>n;
35. init();
36. //for(int i=1;i<=100;i++) cout<<fac[i]<<" "<<infac[i]<<endl;
37. **for**(**int** i=1;i<=n;i++)
38. {
39. **int** a,b;
40. cin>>a>>b;
41. cout<<C(a,b)<<endl;
42. }
43. **return** 0;
44. }

**3、**

**利用Lucas定理，当p为质数时，有**

1. #include<bits/stdc++.h>
2. #define int long long
3. **using** **namespace** std;
4. **int** qpow(**int** a,**int** n,**int** mod)
5. {
6. **int** ans=1;
7. **while**(n)
8. {
9. **if**(n&1) ans=ans\*a%mod;
10. a=a\*a%mod;
11. n>>=1;
12. }
13. **return** ans;
14. }
15. **int** C(**int** a,**int** b,**int** p)
16. {
17. **int** res=1;
18. **for**(**int** i=1,j=a;i<=b;i++,j--)
19. {
20. res=res\*j%p\*qpow(i,p-2,p)%p;
21. }
22. **return** res;
23. }
24. **int** lucas(**int** a,**int** b,**int** p)
25. {
26. **if**(a<b) **return** 0;
27. **if**(a<p&&b<p) **return** C(a,b,p);
28. **else** **return** lucas(a%p,b%p,p)\*lucas(a/p,b/p,p)%p;
29. }
30. **signed** main()
31. {
32. **int** n;
33. cin>>n;
34. **for**(**int** i=1;i<=n;i++)
35. {
36. **int** a,b,p;
37. cin>>a>>b>>p;
38. cout<<lucas(a,b,p)<<endl;
39. }
40. **return** 0;
41. }

## 7.10斯特林数

### 7.10.1第一类斯特林数

**求将n个互不相同的球分成k个圆排列的方案数**

**思路**

s[i][j]表示将i个球分成j个圆排列的方案数，当放入第i个球时，可以将其额外新增一个原排列，此时方案数位s[i-1][j-1]，也可以将其插入到前i-1个数的空隙中，由于是圆排列，x个球就有x个空，于是新增了（i-1)\*s[i-1][j]种方案。

**代码**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **int** k,n;
5. **const** **int** N=1005;
6. **int** s[N][N];
7. **const** **int** mod=1e9+7;
8. **signed** main()
9. {
10. cin>>n>>k;
11. s[0][0]=1;
12. **for**(**int** i=1;i<=n;i++)
13. {
14. **for**(**int** j=1;j<=k;j++)
15. {
16. s[i][j]=s[i-1][j-1]+(i-1)\*s[i-1][j];
17. s[i][j]%=mod;
18. }
19. }
20. cout<<s[n][k]<<endl;
21. }

### 7.10.2第二类斯特林数

**求将n个不同的球分成k组的方案数**

**思路：**S[i][j][表示将i个球分成j组的方案数，当放入第i个球时，同第一类斯特林数，可以选择额外生成一个原排列，则方案数为S[i-1][j-1]，也可以将其插入到前j组中，方案数为j\*S[i-1][j]。

**代码**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. #define int long long
4. **const** **int** N=1005;
5. **int** S[N][N];
6. **int** n,k;
7. **const** **int** mod=1e9+7;
8. **signed** main()
9. {
10. cin>>n>>k;
11. S[0][0]=1;
12. **for**(**int** i=1;i<=n;i++)
13. {
14. **for**(**int** j=1;j<=k;j++)
15. {
16. S[i][j]=S[i-1][j-1]+j\*S[i-1][j];
17. S[i][j]%=mod;
18. }
19. }
20. cout<<S[n][k]<<endl;
21. **return** 0;
22. }

## 7.11卡特兰数

### 7.11.1概述

**卡特兰数是一种经典的组合数，经常出现在各种计算中，其前几项为 :**

**1 , 2 , 5 , 14 , 42 , 132 , 429, 1430, 4862, 16796,**

**58786, 208012, 742900, 2674440, 9694845…**

### 7.11.2.常见公式

### 7.11.3.经典例题

**例1 P1641 [SCOI2010]生成字符串**

**题目描述**

lxhgww 最近接到了一个生成字符串的任务，任务需要他把 n 个 1 和 m 个 0 组成字符串，但是任务还要求在组成的字符串中，在任意的前 k 个字符中，1 的个数不能少于 0 的个数。现在 lxhgww 想要知道满足要求的字符串共有多少个，聪明的程序员们，你们能帮助他吗？

**输入格式**

输入数据只有一行，包括 2 个数字 n 和 m。

**输出格式**

输出数据是一行，包括 1 个数字，表示满足要求的字符串数目，这个数可能会很大，只需输出这个数除以 20100403 的余数

**输入输出样例**

**输入#1**

**2 2**

**输入#1**

**2**

**思路**

**将1和0的操作转换到坐标系中，假设有1则（x+1，y+1），有0则（x+1,y-1），我们发现终点为（n+m,n-m），若存在前k字母中0个数大于1，则在坐标系中曲线经过y=-1，将终点关于y=-1对称，则终点对称点为（n+m，m-n-2），于是就有n-m+1步的向上变成向下，即一共有n+1步向下，所有情况有种可能，不合法的有种可能，答案相减即可，另一种对称方法是将起点关于y=-1对称，则起点变为（0，-2），到达（n+m，n-m）,需要将一步向下变为向上，则不合法有种可能**

**代码**

1. #include<bits/stdc++.h>
2. #define int long long
3. **using** **namespace** std;
4. **const** **int** mod=20100403;
5. **int** qpow(**int** a,**int** n,**int** mod)
6. {
7. **int** res=1;
8. **while**(n)
9. {
10. **if**(n&1) res=res\*a%mod;
11. a=a\*a%mod;
12. n>>=1;
13. }
14. **return** res;
15. }
16. **int** C(**int** a,**int** b,**int** mod)
17. {
18. **int** res=1;
19. **for**(**int** i=1,j=a;i<=b;i++,j--)
20. {
21. res\*=j;
22. res%=mod;
23. res\*=qpow(i,mod-2,mod);
24. res%=mod;
25. }
26. **return** res;
27. }
28. **signed** main()
29. {
30. **int** n,m;
31. cin>>n>>m;
32. cout<<((C(n+m,m,mod)-C(n+m,m-1,mod))%mod+mod)%mod<<endl;
33. **return** 0;
34. }

**例2 P1044 [NOIP2003 普及组] 栈**

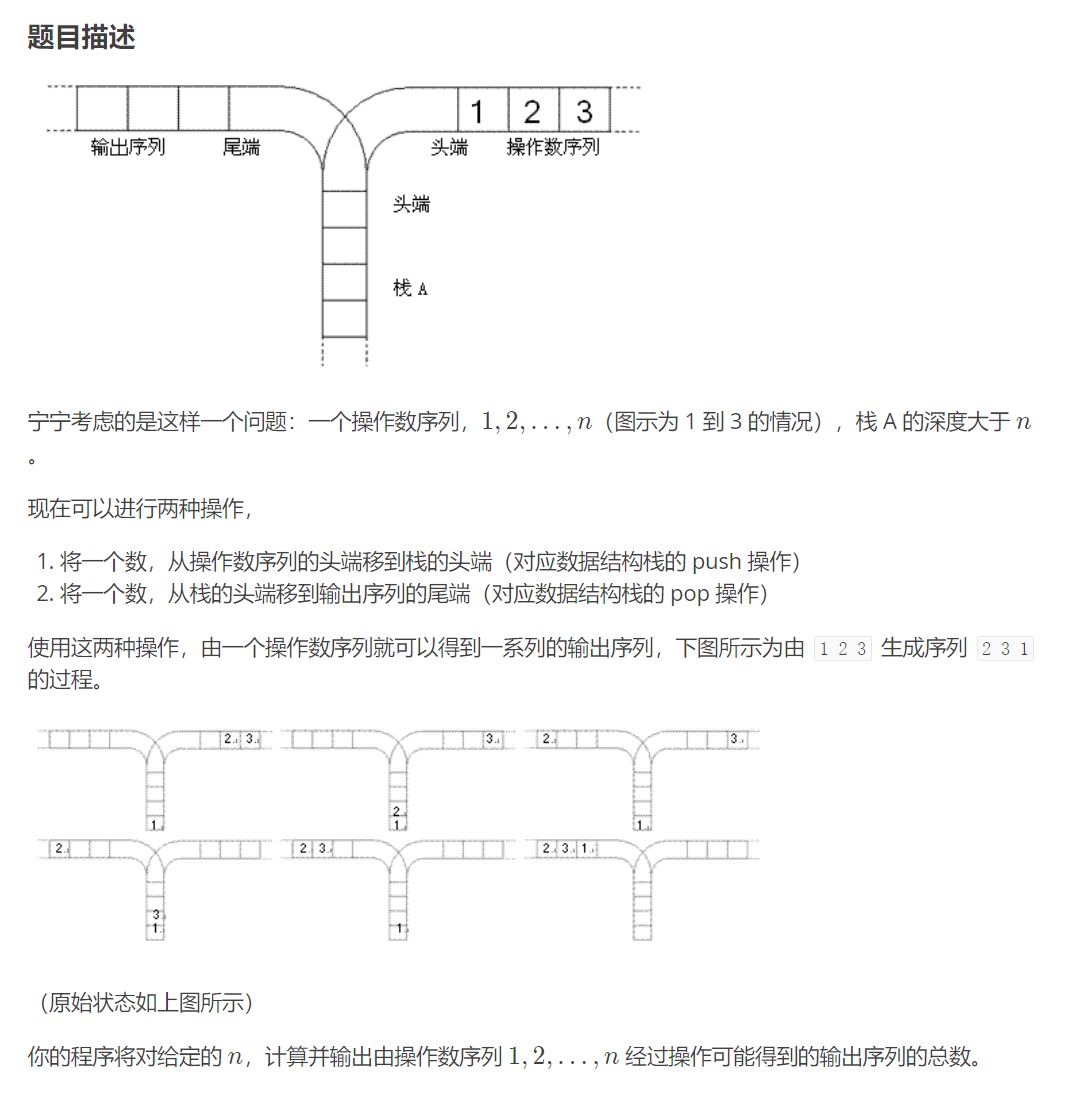
**题目背景**

栈是计算机中经典的数据结构，简单的说，栈就是限制在一端进行插入删除操作的线性表。

栈有两种最重要的操作，即 pop（从栈顶弹出一个元素）和 push（将一个元素进栈）。

栈的重要性不言自明，任何一门数据结构的课程都会介绍栈。宁宁同学在复习栈的基本概念时，想到了一个书上没有讲过的问题，而他自己无法给出答案，所以需要你的帮忙。

**题目描述**

**输入格式**

输入文件只含一个整数

**输出格式**

输出文件只有一行，即可能输出序列的总数目。

**输入输出样例#1**

**输入#1**

3

**输出#1**

5

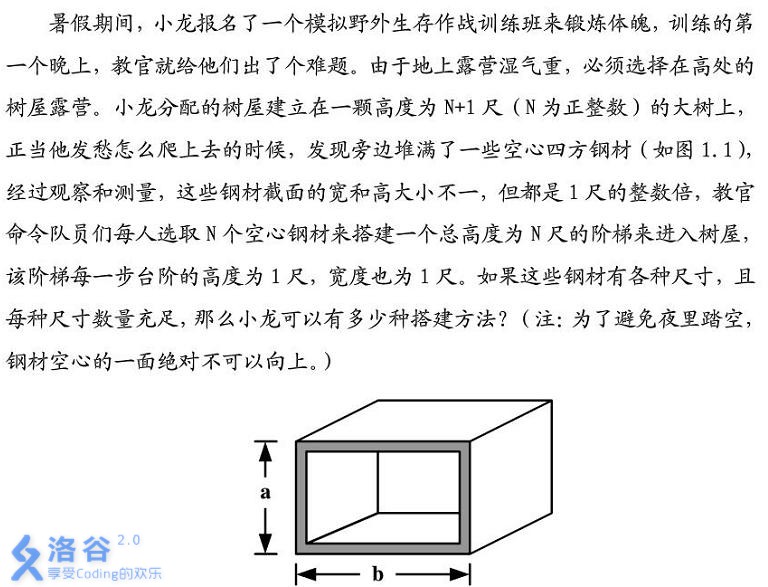
**思路**

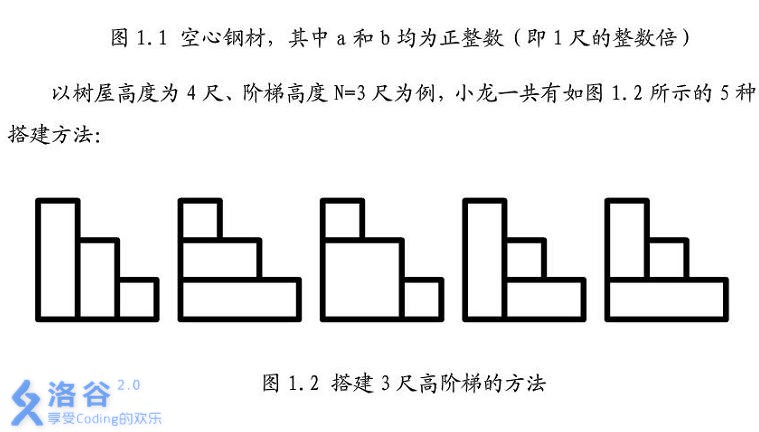
**将出入栈投射到坐标轴中，入栈为(x+1,y+1)，出栈为(x+1,y-1)，则入栈数始终大于出栈数，保证坐标曲线不经过y=-1，同例1**

**代码**

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** qpow(**int** a,**int** n)
4. {
5. **int** res=1;
6. **while**(n)
7. {
8. **if**(n&1) res=res\*a;
9. a=a\*a;
10. n>>=1;
11. }
12. **return** res;
13. }
14. **const** **int** N=50;
15. **int** c[N][N];
16. **void** init(**int** n)
17. {
18. **for**(**int** i=0;i<=n;i++)
19. {
20. **for**(**int** j=0;j<=i;j++)
21. {
22. **if**(!j) c[i][j]=1;
23. **else** c[i][j]=c[i-1][j-1]+c[i-1][j];
24. }
25. }
26. }
27. **int** main()
28. {
29. **int** n;
30. cin>>n;
31. init(n\*2+1);
32. cout<<c[2\*n][n]-c[2\*n][n-1]<<endl;
33. **return** 0;
34. }

**例1 P2532 [AHOI2012]树屋阶梯**

****

****

输入格式

一个正整数N（1<=N<=500），表示阶梯的高度。

**输出格式**

一个正整数，表示搭建方法的个数。（注：搭建方法的个数可能很大）

**输入输出样例**

输入

**3**

输出

**5**

**思路**

**我们固定左下角的方块，假设变成为i，则方块上方需组成高度i-1的台阶，方块右边需组成高度为n-i-1的，于是就可以用公式1解决，注意本题需要用到高精度**

**代码**

1. #include<iostream>
2. #include<cstring>
3. **const** **int** N=505,L=500;
4. std::string f[N];
5. **int** na[L],nb[L],nc[L];
6. std::string mul(std::string a,std::string b)//高精度乘法a,b,均为非负整数
7. {
8. std::string s;
9. **int** La=a.size(),Lb=b.size();//na存储被乘数，nb存储乘数，nc存储积
10. memset(na,0,**sizeof**(na));//将na,nb,nc都置为0
11. memset(nb,0,**sizeof**(nb));
12. memset(nc,0,**sizeof**(nc));
13. **for**(**register** **int** i=La-1;i>=0;i--) na[La-i]=a[i]-'0';//将字符串表示的大整形数转成i整形数组表示的大整形数
14. **for**(**register** **int** i=Lb-1;i>=0;i--) nb[Lb-i]=b[i]-'0';
15. **for**(**register** **int** i=1;i<=La;i++)
16. **for**(**register** **int** j=1;j<=Lb;j++)
17. nc[i+j-1]+=na[i]\*nb[j];//a的第i位乘以b的第j位为积的第i+j-1位（先不考虑进位）
18. **for**(**register** **int** i=1;i<=La+Lb;i++)
19. nc[i+1]+=nc[i]/10,nc[i]%=10;//统一处理进位
20. **if**(nc[La+Lb]) s+=nc[La+Lb]+'0';//判断第i+j位上的数字是不是0
21. **for**(**register** **int** i=La+Lb-1;i>=1;i--)
22. s+=nc[i]+'0';//将整形数组转成字符串
23. **return** s;
24. }
25. std::string add(std::string a,std::string b)//只限两个非负整数相加
26. {
27. std::string ans;
28. memset(na,0,**sizeof**(na));
29. memset(nb,0,**sizeof**(nb));
30. **int** la=a.size(),lb=b.size();
31. **for**(**register** **int** i=0;i<la;i++) na[la-1-i]=a[i]-'0';
32. **for**(**register** **int** i=0;i<lb;i++) nb[lb-1-i]=b[i]-'0';
33. **int** lmax=la>lb?la:lb;
34. **for**(**register** **int** i=0;i<lmax;i++) na[i]+=nb[i],na[i+1]+=na[i]/10,na[i]%=10;
35. **if**(na[lmax]) lmax++;
36. **for**(**register** **int** i=lmax-1;i>=0;i--) ans+=na[i]+'0';
37. **return** ans;
38. }
39. **void** init(**int** n)
40. {
41. f[0]="1";
42. f[1]="1";
43. f[2]="2";
44. **for**(**int** i=3;i<=n;i++)
45. {
46. **for**(**int** j=0;j<i;j++)
47. {
48. f[i]=add(mul(f[j],f[i-j-1]),f[i]);
49. //cout<<i<<" "<<f[i]<<endl;
50. }
51. }
52. }
53. **int** main()
54. {
55. **int** n;
56. scanf("%d",&n);
57. init(n);
58. std::cout<<f[n];
59. **return** 0;
60. }

## 7.12博弈论

### 7.12.1 Bash

1. #define \_MAX 10000
2. **int** a[\_MAX];
3. **int** b[\_MAX];
5. **int** bash(**int** N, **int** K)
6. {
7. **if** (N % (K + 1) == 0)
8. {
9. **return** 2;
10. }
11. **return** 1;
12. }
14. **int** main()
15. {
16. **int** T;
17. scanf("%d", &T);
18. **for** (**int** i = 0; i < T; i++)
19. {
20. scanf("%d%d", a + i, b + i);
21. }
22. **for** (**int** i = 0; i < T; i++)
23. {
24. **if** (bash(a[i], b[i]) == 1)
25. {
26. printf("A\n");
27. }
28. **else**
29. {
30. printf("B\n");
31. }
32. }
33. **return** 0;
34. }

##### 7.12.2 Nim

1. **int** main(**int** argc, **const** **char** \* argv[])
2. {
3. **int** N, stone, tag = 0;
4. scanf("%d", &N);
5. **while** (N--)
6. {
7. scanf("%d", &stone);
8. tag ^= stone;
9. }
10. //tag为0则为后手赢，否则为先手赢
11. printf("%c\n", tag == 0 ? 'B' : 'A');
12. **return** 0;
13. }

##### 7.12.3 SG打表

1. **const** **int** MAX\_DIG = 64;
3. //  SG打表
4. //  f[]:可以取走的石子个数
5. //  sg[]:0~n的SG函数值
6. //  hash[]:mex{}
7. **int** f[MAX\_DIG];
8. **int** sg[MAX\_DIG];
9. **int** hash[MAX\_DIG];
11. **void** getSG(**int** n)
12. {
13. memset(sg, 0, **sizeof**(sg));
14. **for** (**int** i = 1; i <= n; i++)
15. {
16. memset(hash, 0, **sizeof**(hash));
17. **for** (**int** j = 1; f[j] <= i; j++)
18. {
19. hash[sg[i - f[j]]] = 1;
20. }
21. **for** (**int** j = 0; j <= n; j++)    //  求mes{}中未出现的最小的非负整数
22. {
23. **if** (hash[j] == 0)
24. {
25. sg[i] = j;
26. **break**;
27. }
28. }
29. }
30. }

##### 7.12.4 SG DFS

1. **const** **int** MAX\_DIG = 64;
3. //  DFS
4. //  注意 S数组要按从小到大排序 SG函数要初始化为-1 对于每个集合只需初始化1遍
5. //  n是集合s的大小 S[i]是定义的特殊取法规则的数组
6. **int** s[MAX\_DIG];
7. **int** sg[MAX\_DIG \* 100];
8. **int** n;
10. **int** SG\_dfs(**int** x)
11. {
12. **if** (sg[x] != -1)
13. {
14. **return** sg[x];
15. }
16. **bool** vis[MAX\_DIG];
17. memset(vis, 0, **sizeof**(vis));
18. **for** (**int** i = 0; i < n; i++)
19. {
20. **if** (x >= s[i])
21. {
22. SG\_dfs(x - s[i]);
23. vis[sg[x - s[i]]] = 1;
24. }
25. }
26. **int** e;
27. **for** (**int** i = 0; ; i++)
28. {
29. **if** (!vis[i])
30. {
31. e = i;
32. **break**;
33. }
34. }
35. **return** sg[x] = e;
36. }

##### 7.12.5 Wythoff

1. **int** main()
2. {
3. **int** t, a, b, m, k;
4. scanf("%d", &t);
5. **while** (t--)
6. {
7. scanf("%d%d", &a, &b);
8. **if** (a > b)
9. {
10. a ^= b;
11. b ^= a;
12. a ^= b;
13. }
14. m = b - a;
15. k = (**int**)(m \* (1 + sqrt(5)) / 2.0);
16. //m = ? \* a
17. //k = m / ?
18. //?:黄金分割数
19. //如果a == k，则为后手赢，否则先手赢（奇异局）
20. printf("%s\n", a == k ? "B" : "A");
21. }
22. **return** 0;
23. }

## 7.13 矩阵

**7.13 关系矩阵初步**

在dp或者递推中，我们经常会用到矩阵加速，下面介绍矩阵的一些基本定义以及操作方法。

a)矩阵定义。

矩阵可以快速解决一系列值之间的线性关系。下面定义一个矩阵。

其中 MAX\_MAT是矩阵的大小（关系矩阵必定是方阵）

1. **struct** Mat
2. {
3. **long** **long** a[MAX\_MAT][MAX\_MAT];
4. Mat()
5. {
6. **for** (**int** i = 0; i < MAX\_MAT; ++i)
7. {
8. **for** (**int** j = 0; j < MAX\_MAT; ++j)
9. {
10. a[i][j] = 0;
11. }
12. }
13. **for** (**int** i = 0; i < MAX\_MAT; ++i)
14. {
15. a[i][i] = 1;
16. }
17. }  //构造矩阵时自动初始化为0.
18. Mat(**long** **long** a1, **long** **long** a2, **long** **long** a3, **long** **long** a4)
19. {
20. a[0][0] = a1;
21. a[0][1] = a2;
22. a[1][0] = a3;
23. a[1][1] = a4;
24. }  //给矩阵赋值，依据题目不同可以调整。
25. };

b)矩阵乘法

矩阵乘法是矩阵线性变换经常用到的一个操作（包括矩阵ksm）

这里的矩阵乘法返回值是一个Mat，输入的也是Mat，需要先进行a）中的定义。

1. Mat operator \* (Mat x, Mat y)
2. {
3. Mat c;
4. **for** (**int** i = 0; i < MAX\_MAT; ++i) {
5. **for** (**int** j = 0; j < MAX\_MAT; ++j) {
6. c.a[i][j] = 0;
7. }
8. }
9. **for** (**int** i = 0; i < MAX\_MAT; ++i) {
10. **for** (**int** j = 0; j < MAX\_MAT; ++j) {
11. **for** (**int** k = 0; k < MAX\_MAT; ++k) {
12. c.a[i][j] = (c.a[i][j] + x.a[i][k] \* y.a[k][j] % mod) % mod;
13. }
14. }
15. }
16. **return** c;
17. }

c)矩阵求逆

1. **long** **long** quickpow(**long** **long** x, **long** **long** y, **long** **long** MOD = 9223372036854775807LL)
2. {
3. **long** **long** ans = 1;
4. **while** (y)
5. {
6. **if** (y & 1)
7. {
8. ans = (x \* ans) % MOD;
9. }
10. x = (x \* x) % MOD;
11. y >>= 1;
12. }
13. **return** ans;
14. }
15. **long** **long** A[MAX\_MAT][MAX\_MAT << 1];
16. **long** **long** get\_inv(**long** **long** x)
17. {
18. **return** quickpow(x, mod - 2, mod);
19. }
20. **void** row\_minus(**int** a, **int** b, **long** **long** k)
21. {
22. **for** (**int** i = 0; i < 2 \* MAX\_MAT; ++i)
23. {
24. A[a][i] = (A[a][i] - A[b][i] \* k % mod) % mod;
25. **if** (A[a][i] < 0)A[a][i] += mod;
26. }
27. **return**;
28. }
29. **void** row\_multiplies(**int** a, **long** **long** k)
30. {
31. **for** (**int** i = 0; i < 2 \* MAX\_MAT; ++i)
32. {
33. A[a][i] = (A[a][i] \* k) % mod;
34. }
35. **return**;
36. }
37. **void** row\_swap(**int** a, **int** b)
38. {
39. **for** (**int** i = 0; i < 2 \* MAX\_MAT; ++i)
40. {
41. swap(A[a][i], A[b][i]);
42. }
43. }
44. Mat getinv(Mat x)  //矩阵求逆
45. {
46. memset(A, 0, **sizeof**(A));
47. **for** (**int** i = 0; i < MAX\_MAT; ++i)
48. {
49. **for** (**int** j = 0; j < MAX\_MAT; ++j)
50. {
51. A[i][j] = x.a[i][j];
52. A[i][MAX\_MAT + j] = i == j;
53. }
54. }
55. **for** (**int** i = 0; i < MAX\_MAT; ++i)
56. {
57. **if** (!A[i][i])
58. {
59. **for** (**int** j = i + 1; j < MAX\_MAT; ++j)
60. {
61. **if** (A[j][i])
62. {
63. row\_swap(i, j);
64. **break**;
65. }
66. }
67. }
68. row\_multiplies(i, get\_inv(A[i][i]));
69. **for** (**int** j = i + 1; j < MAX\_MAT; ++j)
70. {
71. row\_minus(j, i, A[j][i]);
72. }
73. }
74. **for** (**int** i = MAX\_MAT - 1; i >= 0; --i)
75. {
76. **for** (**int** j = i - 1; j >= 0; --j)
77. {
78. row\_minus(j, i, A[j][i]);
79. }
80. }
81. Mat ret;
82. **for** (**int** i = 0; i < MAX\_MAT; ++i)
83. {
84. **for** (**int** j = 0; j < MAX\_MAT; ++j)
85. {
86. ret.a[i][j] = A[i][MAX\_MAT + j];
87. }
88. }
89. **return** ret;
90. }

# 8.计算几何

## 8.1凸包模板

1. #include <iostream>
2. #include <algorithm>
3. #include <iomanip>
4. #include<stdio.h>
5. **using** **namespace** std;
6. **const** **int** maxn = 50010;
7. **struct** Point {
8. **int** x , y;
9. **bool** operator < (Point **const** &rhs) **const** {
10. **return** (x < rhs.x) || (x == rhs.x && y < rhs.y);
11. }
12. };
14. **int** Cross(Point **const** &O , Point **const** &A , Point **const** &B)
15. {
16. **int** xoa = A.x - O.x;
17. **int** xob = B.x - O.x;
18. **int** yoa = A.y - O.y;
19. **int** yob = B.y - O.y;
20. **return** xoa \* yob - xob \* yoa;
21. }
22. **int** Andrew(Point \*p , **int** n , Point \*ch)
23. {
24. sort(p , p + n);
25. **int** m = 0;
26. **for**(**int** i = 0; i < n; i++)
27. { //下凸包
28. **while**(m > 1 && Cross(ch[m-2] , ch[m-1] , p[i]) < 0) m--;
29. ch[m++] = p[i];
30. }
31. **int** k = m;
32. **for**(**int** i = n - 2; i >= 0; i--) {  //上凸包
33. **while**(m > k && Cross(ch[m-2] , ch[m-1] , p[i]) < 0) m--;
34. ch[m++] = p[i];
35. }
36. **if**(n > 1) m--;
37. **return** m;
38. }
39. Point p[maxn] , ch[maxn];
40. **int** main()
41. {
42. **int** n;
43. **while**(cin >> n)
44. {
45. **for**(**int** i = 0; i < n; i++) cin >> p[i].x >> p[i].y;
46. **int** m = Andrew(p , n , ch); ///求凸包
48. ///旋转卡壳法
49. **int** ans = 0;
50. **for**(**int** i = 0; i < m; i++)
51. {
52. **int** q = 1;
53. **for**(**int** j = i + 1; j < m; j++)
54. {
55. **while**(Cross(ch[i],ch[j],ch[q+1]) > Cross(ch[i],ch[j],ch[q]))
56. q = (q + 1) % m;
57. ans = max(ans , Cross(ch[i],ch[j],ch[q]));
58. }
59. }
60. //cout << fixed << setprecision(2) << ans / 2.0 << endl;
61. printf("%.2lf\n",ans/2.0);
62. }
63. **return** 0;
64. }