KetchupZ

1249553079@qq.com

摘要

博客地址：https://blog.csdn.net/Dch19990825

ACM模板

****

目录

[1 目录 1](#_Toc24362216)

[1字符串处理 1](#_Toc24362217)

[1.1 KMP处理 1](#_Toc24362218)

[1.2 KMP最小循环节 2](#_Toc24362219)

[1.3 扩展KMP 3](#_Toc24362220)

[1.4 Manacher最长回文子串 5](#_Toc24362221)

[1.5 字典树 6](#_Toc24362222)

[1.6 AC自动机 (带注释) 7](#_Toc24362223)

[1.7 回文树/回文自动机 10](#_Toc24362224)

[1.8 后缀数组 15](#_Toc24362225)

[1.8.1 DA算法 21](#_Toc24362226)

[1.8.2 DC3算法 25](#_Toc24362227)

[1.9 字符串HASH 26](#_Toc24362228)

[2 数论 27](#_Toc24362229)

[2.1 打表（素数，欧拉函数，莫比乌斯） 27](#_Toc24362230)

[2.1.1 素数打表 27](#_Toc24362231)

[2.1.2 素数，欧拉函数打表，莫比乌斯函数打表 28](#_Toc24362232)

[2.1.3 欧拉函数打表 29](#_Toc24362233)

[2.1.4 求出N的约数的莫比乌斯函数的值 31](#_Toc24362234)

[2.1.5 线性筛出每个数的最小素因子（快速求每个数的素因子分解形式） 32](#_Toc24362235)

[2.2 扩展欧几里得算法（求 ax+by=gcd 的解以及逆元素） 33](#_Toc24362236)

[2.3 求逆元 34](#_Toc24362237)

[2.4 一次同余方程 求a\*x=c（mod m） 34](#_Toc24362238)

[2.5 模线性方程组（中国剩余定理） 35](#_Toc24362239)

[2.6 高斯消元 38](#_Toc24362240)

[2.7 欧拉降幂 39](#_Toc24362241)

[2.8 莫比乌斯反演，求n的约数的莫比乌斯函数值 41](#_Toc24362242)

[2.9 狄利克雷卷积(未添加) 44](#_Toc24362243)

[2.10 部分数列和公式 44](#_Toc24362244)

[2.11 部分定理 48](#_Toc24362245)

[2.12 一些常识 49](#_Toc24362246)

[2.13 关于向上取整和向下取整 49](#_Toc24362247)

[2.14 线性求阶乘，自然数的逆元 51](#_Toc24362248)

[3 组合数学 51](#_Toc24362249)

[3.1 组合数打表 51](#_Toc24362250)

[3.2 容斥定理 52](#_Toc24362251)

[3.3 部分枚举子集 54](#_Toc24362252)

[3.4 排列组合问题总结 55](#_Toc24362253)

[4 图论 57](#_Toc24362254)

[4.1 最小生成树 57](#_Toc24362255)

[4.1.1 克鲁斯卡尔（Kruskal)算法 57](#_Toc24362256)

[4.1.2 prime算法+优先队列优化 58](#_Toc24362257)

[4.2 最短路 59](#_Toc24362258)

[4.2.1 Dijstra+优先队列 只适用于正边权 60](#_Toc24362259)

[4.2.2 SPFA ，边权正负均可，无负环 61](#_Toc24362260)

[4.2.3 Spfa判断有无负环， 62](#_Toc24362261)

[4.2.4 Bellman-ford判断是否有负环,有返回true ， 63](#_Toc24362262)

[4.2.5 Floyd+输出路径 63](#_Toc24362263)

[4.2.6 差分约束 64](#_Toc24362264)

[4.3 最小瓶颈路 65](#_Toc24362265)

[4.4 欧拉路径 见Bing神 65](#_Toc24362266)

[4.5 二分图匹配 66](#_Toc24362267)

[4.5.1 性质 66](#_Toc24362268)

[4.5.2 二分图匹配匈牙利算法 66](#_Toc24362269)

[4.5.3 二部图的二分图匹配 68](#_Toc24362270)

[4.6 网络流 69](#_Toc24362271)

[4.6.1 EK算法 70](#_Toc24362272)

[4.6.2 我bing神的模板贼快 ISAP+BSF+栈优化 72](#_Toc24362273)

[4.7 最小割 74](#_Toc24362274)

[4.8 最大权闭合子图 75](#_Toc24362275)

[4.9 最小费用流 79](#_Toc24362276)

[4.9.1 SPFA寻找最短增广路，对于有负环无法操作（可以找到负环，然后将负环消去）。 80](#_Toc24362277)

[4.9.2 采用dijstra的最短增广路算法：（有时候性能不如SPFA的，可能是因为容器的问题） 82](#_Toc24362278)

[4.10 Color coding K-th alogoritm 85](#_Toc24362279)

[4.11 无向图三元环计数 87](#_Toc24362280)

[4.12 普吕弗序列（Prufer） 90](#_Toc24362281)

[4.13 图的割点、桥和双联通分支的概念 95](#_Toc24362282)

[4.14 割点 96](#_Toc24362283)

[4.15 桥 98](#_Toc24362284)

[4.16 边双连通分支 100](#_Toc24362285)

[4.17 点双联通分支 102](#_Toc24362286)

[4.18 强连通分量 105](#_Toc24362287)

[4.19 LCA最近公共祖先 107](#_Toc24362288)

[4.19.1 lCA在线倍增算法 107](#_Toc24362289)

[4.19.2 Lca求树上两点路径距离 108](#_Toc24362290)

[5 数据结构 109](#_Toc24362291)

[5.1 树状数组 109](#_Toc24362292)

[5.1.1 一维树状数组 109](#_Toc24362293)

[5.1.2 二维树状数组 110](#_Toc24362294)

[5.2 单调栈和单调队列 114](#_Toc24362295)

[5.3 线段树 116](#_Toc24362296)

[5.4 可持久化线段树 120](#_Toc24362297)

[5.5 珂朵莉树/old\_driver\_tree 123](#_Toc24362298)

[6 几何数学 129](#_Toc24362299)

[6.1 基本函数（已封印） 129](#_Toc24362300)

[6.2 三角形求外心 129](#_Toc24362301)

[6.3 判断是否线段与多边形规范相交 129](#_Toc24362302)

[6.4 最小圆覆盖/最小球覆盖 129](#_Toc24362303)

[6.5 平面方程表示 134](#_Toc24362304)

[6.6 自适应Simpson积分法 135](#_Toc24362305)

[7 动态规划经典例题 136](#_Toc24362306)

[7.1 区间dp 136](#_Toc24362307)

[7.2 数位DP 138](#_Toc24362308)

[7.3 状压DP 140](#_Toc24362309)

[7.4 树形DP 142](#_Toc24362310)

[7.5 期望DP 147](#_Toc24362311)

[7.6 插头DP(未加入) 151](#_Toc24362312)

[7.7 轮廓线DP(未加入) 151](#_Toc24362313)

[7.8 背包问题之退背包 151](#_Toc24362314)

[8 杂项 152](#_Toc24362315)

[8.1 归并排序求逆序对 152](#_Toc24362316)

[8.2 cdq分治解三维偏序 153](#_Toc24362317)

[8.3 表达式求值（留小坑） 160](#_Toc24362318)

[8.4 约瑟夫问题(留小坑) 163](#_Toc24362319)

[9 奇淫技巧 166](#_Toc24362320)

[9.1 超级快读 166](#_Toc24362321)

[9.2 其他库函数 166](#_Toc24362322)

[9.3 全排列函数 167](#_Toc24362323)

[9.4 随机库函数 167](#_Toc24362324)

[9.5 C++输出格式控制 168](#_Toc24362325)

[9.6 二次函数求解 169](#_Toc24362326)

[9.7 日期计算 169](#_Toc24362327)

[9.8 线性递推BM 170](#_Toc24362328)

[9.9 程序对拍 173](#_Toc24362329)

[10 Python 174](#_Toc24362330)

[11 JAVA大数和排序（未做） 180](#_Toc24362331)

# 1字符串处理

## KMP处理

1.得到net数组

1. //net[]的含义:字符串s前缀长度为k与前缀的最大匹配长度为net[k]   eg:abcabc的最长匹配为abc 那么net[6]=3
2. //输入：字符串s[],s的长度ls,net的储存位置
3. //结果：得到s的net[]
4. **void** getNext(**const** **char** s[],**int** ls,**int** net[])
5. {
6. net[0]=-1;
7. **int** j=0,k=-1;
8. **while**(j<ls)
9. {
10. **if**(k==-1||s[j]==s[k])
11. net[++j]=++k;
12. **else**
13. k=net[k];
14. }
15. }

2. 模式串在主串的所有出现位置，和出现次数

1. //输入:模式串tt[]，主串ss[]
2. //结果：保存模式串在主串匹配的动态数组match，返回tt[]在ss[]的出现次数
3. **const** **int** maxn=1e6+100;
4. **int** net[maxn];
5. vector<**int**> matchP;
6. **int** kmpCount(**const** **char** tt[],**const** **char** ss[])
7. {
8. **int** ls,lt,i=0,j=0;
9. ls=strlen(ss);
10. lt=strlen(tt);
11. getNext(tt,lt,net);//求net数组
12. matchP.clear();//\*tot=0
13. **while**(i<ls&&(ls-i)>=(lt-j))
14. {
15. **while**(j!=-1&&ss[i]!=tt[j])  j=net[j];
16. i++;j++;
17. **if**(j>=lt){
18. matchP.push\_back(i-j+1);//若只记录出现次数,改为计数 ++tot。
19. j=net[j];
20. }
21. }
22. **return** matchP.size();
23. }

## KMP最小循环节

**原理：**

设T[i]为前i个字符串最小周期为T[i]  eg：aabaab   T[1]=1 ,T[2]=1,T[3]=3

设next[i]为前i个字符最多匹配前next[i]个字符（不包括自身  这里的next函数与KMP中函数意思有些更改  这里的next[i]等效于书上的nextval[i+1]-1）  eg:aabaab next[1]=0  next[2]=1  next[3]=0  next[4]=1 next[5]=2 next[6]=3

用语言来表达就是

         如果i是个周期字符串必定有 I-next[i]=T[next[i]]   其周期T[i]=T[next[i]]

         并且如果i-next[i]=T[next[i]] 那么  i必定是个周期字符周期为T[i]=T[next[i]]

 HDU - 1358 （找出字符串的最小周期）

1. **const** **int** maxn=1001000;
2. **int** net[maxn];//net[j] 下标为j的失配后 匹配到下标net[j]  即前j个字符串最大匹配为net[j]个
3. **int** T[maxn];//T[i]代表最小周期为T[i]
4. **void** Get\_nextval(**int** \*net,**char** \*str)//这是求i以及i之前的最大匹配数目
5. {
6. **int** len=strlen(str);
7. net[0]=-1;
8. **int** j=0,k=-1;
9. **while**(j<len)//j为字符串的下标
10. {
11. **if**(k==-1||str[j]==str[k])
12. net[++j]=++k;
13. **else**
14. k=net[k];
15. }
16. }
17. **int** main()
18. {
19. **char** str[maxn];
20. **int** cas=0;
21. **int** len;
22. **while**(scanf("%d",&len)&&len)
23. {
24. scanf("%s",str);
25. Get\_nextval(net,str);
26. printf("Test case #%d\n",++cas);
27. T[0]=0;
28. T[1]=1;
29. **int** k;
30. **for**(**int** i=2;i<=len;i++)
31. {
32. k=net[i];
33. **if**(i-k==T[k])
34. {
35. T[i]=T[k];
36. cout<<i<<" "<<i/T[i]<<endl;//输出最小循环节的周期
37. }
38. **else**
39. T[i]=i;
40. }
41. cout<<endl;
42. }
43. }

## 扩展KMP

普通kmp是以**i为后缀的相同的最长前缀,** 而扩展kmp是**i为前缀的相同的最长前缀**)

EXkmp[]:s[i...k+i-1]与s[0...k-1]相等的最大的k

Kmp利用了上一个的最长前缀，而EXkmp是利用已经计算过的exkmp，与manacher算法有些相似，但与kmp算法觉得还是有很大区别。

1. **const** **int** maxn=1e6+10;
2. **char** a[maxn],b[maxn];
3. **int** exb[maxn],exkmp[maxn];
4. /\*
5. EXkmp:s[i...k+i-1]与s[0...k-1]相等的最大的k  (普通kmp是以i为后缀的相同的最长前缀,而扩展kmp是i为前缀的最长前缀)
6. exb[i]:b[i...k+i-1]与b[0...k-1]相等的最大的k
7. exkmp[i]:a[i..k+i-1]与b[0..k-1]相同的最大的k
8. \*/
9. **void** pre\_kmp(**char** s[],**int** ls,**int** next[])//预处理字符串b的kmp
10. {
11. **int** id,maxr;//最长扩展的i和延申的位置maxr
12. next[0]=ls;
13. id=-1,maxr=-1;
14. **for**(**int** i=1;i<ls;++i)
15. {
16. **int** qs;//next[i]的值
17. **if**(maxr-i+1<=0)//不在可以扩展的范围,那就自己先为0
18. qs=0;
19. **else**
20. qs=min(maxr-i+1,next[i-id]);//可扩展,取min以防出边界
21. **while**(i+qs<ls&&s[i+qs]==s[qs]) ++qs;//暴力扩展未知的
22. next[i]=qs;
23. **if**(qs+i-1>maxr){
24. maxr=qs+i-1;
25. id=i;
26. }
27. }
28. }
29. **void** EXkmp(**char** a[],**int** la,**char** b[],**int** lb,**int** next[],**int** exkmp[])
30. {
31. pre\_kmp(b,lb,next);
32. **int** id=-1,maxr=-1;
33. **for**(**int** i=0;i<la;++i){
34. **int** qs;
35. **if**(maxr-i+1<=0) qs=0;
36. **else** qs=min(maxr-i+1,next[i-id]);
37. **while**(i+qs<la && qs<lb && a[i+qs]==b[qs]) ++qs;
38. exkmp[i]=qs;
39. **if**(qs+i-1>maxr){
40. maxr=qs+i-1;
41. id=i;
42. }
43. }
44. }

## Manacher最长回文子串

求最长回文串长度

1. //求最长回文子串
2. //输入: 字符串s和长度len
3. //结果: 得到mp数组  mp[i]
4. //最长回文字串长度=max(mp[i]-1)
5. **char** Ma[MAXN\*2];
6. **int** Mp[MAXN\*2];
7. **void** Manacher(**char** s[],**int** len)
8. {
9. **int** l=0;
10. Ma[l++]='$';
11. Ma[l++]='#';
12. **for**(**int** i=0; i<len; i++)//len\*2+2 个字符   Ma[i]=s[(i-2)/2]
13. {
14. Ma[l++]=s[i];
15. Ma[l++]='#';
17. }
18. Ma[l]=0;
19. **int** mx=0,id=0;
20. **for**(**int** i=0; i<l; i++)
21. {
22. Mp[i]=mx>i?min(Mp[2\*id-i],mx-i):1;
23. **while**(Ma[i+Mp[i]]==Ma[i-Mp[i]])Mp[i]++;
24. **if**(i+Mp[i]>mx)
25. {
26. mx=i+Mp[i];
27. id=i;
28. }
29. }
30. }

## 字典树

一个字符串是否在多个字符串出现过，最坏情况下为树的深度

数组大小开n\*len个即可。

1. //字典树：在多个字符串中查找一个字符串，也可用于单个异或最大值求解。
2. //操作：初始化树，插入串，查找串
3. **const** **int** maxn=500000+20;
4. **const** **int** Branch=26;
5. **struct** Node
6. {
7. **int** cnt;//标记节点是否为一个单词
8. **int** net[Branch];
9. **void** clear()
10. {
11. cnt=0;
12. memset(net,-1,**sizeof**(net));
13. }
14. } node[maxn];//每个节点都代表一个路径
15. **int** top;//字典树的节点个数
16. **void** initTree()
17. {
18. node[0].clear();
19. top=0;
20. }
21. **int** hash\_letter(**char** c)//字符的节点表示
22. {
23. **return** c-'a';
24. }
25. **void** insertS(**char** s[],**int** ls)//插入字符串s
26. {
27. **int** now=0;
28. **for**(**int** i=0; i<ls; ++i)
29. {
30. **int** k=hash\_letter(s[i]);
31. **if**(node[now].net[k]==-1)
32. {
33. node[top+1].clear();
34. node[now].net[k]=++top;
35. }
36. now=node[now].net[k];
37. }
38. node[now].cnt=1;
39. }
40. **int** searchS(**char** s[],**int** ls)//查看是否有这个单词
41. {
42. **int** now=0;
43. **for**(**int** i=0; i<ls; ++i)
44. {
45. now=node[now].net[hash\_letter(s[i])];
46. **if**(now==-1)
47. **return** 0;
48. }
49. **if**(node[now].cnt)
50. **return** 1;
51. **return** 0;
52. }

## AC自动机 (带注释)

功能：查找多个模式串在文本串出现了多少次

例题：HDU2222 Keywords Search 查找文本串tx中出现了多少个单词

1. /\*
2. AC自动机 函数：
3. 1.初始化树
4. 2.输入模式串 构建字典树
5. 3.构建fail指针
6. 4.匹配文本串
7. \*/
8. //功能：查找多个模式串在文本串出现了多少次
9. **const** **int** maxn=5e5+7;
10. **class** AcTire
11. {
12. **public**:
13. **struct** Node
14. {
15. **int** next[26];
16. **int** fail,cnt;
17. **void** clear()
18. {
19. memset(next,-1,**sizeof**(next));
20. fail=cnt=0;//初始化
21. }
22. };
23. Node node[maxn];
24. **int** top;//使用了多少个节点
25. **void** clear()//初始化
26. {
27. node[0].clear();
28. top=0;
29. }
30. **int** hash\_letter(**char** c)//字符的映射位置
31. {
32. **return** c-'a';
33. }
34. **void** insert(**char** \*p)//插入字符串p
35. {
36. **int** now=0;
37. **while**(\*p)
38. {
39. **if**(node[now].next[hash\_letter(\*p)]==-1)
40. {
41. node[now].next[hash\_letter(\*p)]=++top;
42. node[top].clear();//初始化该节点的信息
43. }
45. now=node[now].next[hash\_letter(\*p)];
46. ++p;
47. }
48. node[now].cnt++;
49. }
50. **void** bulid\_fail()
51. {
52. **int** now=0;
53. **int** to;
54. /\*
55. 保证队列里的fail指针全部OK
56. 从队列中取一个 把其下面的fail指针OK 并且加入队列
57. \*/
58. queue<**int**> mmp;
59. **for**(**int** i=0; i<26; ++i) //将根节点的所有儿子加入队列
60. {
61. **if**(node[0].next[i]!=-1)
62. mmp.push(node[0].next[i]);
63. }
64. **while**(!mmp.empty())//
65. {
66. now=mmp.front();
67. mmp.pop();
68. **for**(**int** i=0; i<26; ++i) //将此节点的儿子的 fail指针计算出来并加入队列
69. {
70. **if**(node[now].next[i]!=-1)
71. {
72. mmp.push(node[now].next[i]);
73. /\*   计算now的 第i个儿子的fail指针   \*/
74. to=node[now].fail;
75. **while**(to>0&&node[to].next[i]==-1)
76. to=node[to].fail;
77. /\* 直至根节点 或者fail的父节点\*/
78. **if**(node[to].next[i]!=-1)//否则没有最大匹配  即为空
79. node[node[now].next[i]].fail=node[to].next[i];
80. }
81. }
82. }
84. }
85. **int** Find\_words(**char** \*tx)
86. {
87. **int** ans=0;
88. **int** now=0;//当前最大匹配下标
89. **int** to,i;
90. **while**(\*tx)
91. {
92. i=hash\_letter(\*tx);
93. //没有下面的匹配，则找跟此字符的最大匹配
94. **while**(now>0&&node[now].next[i]==-1)
95. now=node[now].fail;
96. **if**(node[now].next[i]!=-1)
97. now=node[now].next[i];
98. /\* 开始计算以now为后缀的单词出现数\*/
99. to=now;
100. **while**(to&&node[to].cnt!=-1)/\*精髓之处，匹配过后设为-1代表该后缀对应所有的字符串都匹配过了，避免后来重复匹配已经匹配过的单词\*/
101. {
102. ans+=node[to].cnt;
103. node[to].cnt=-1;//标记该单词已经加过了  就算这个单词没出现 标记为-1代表这个字符串的后缀节点全扫描过了
104. to=node[to].fail;
105. }
106. ++tx;
107. }
108. **return** ans;
109. }
111. };
112. **char** tx[1000007],words[55];
113. AcTire dch;
114. //hdu 2222  ，查找文本串tx中，出现了多少个模式串
115. **int** main()
116. {
117. **int** t,n;
118. scanf("%d",&t);
119. **while**(t--)
120. {
121. dch.clear();
122. scanf("%d",&n);
123. **for**(**int** i=0; i<n; ++i)
124. {
125. scanf("%s",words);
126. dch.insert(words);
127. }
128. dch.bulid\_fail();
129. scanf("%s",tx);
130. printf("%d\n",dch.Find\_words(tx));
131. }
132. **return** 0;
133. }

## 回文树/回文自动机

回文自动机是一个两颗树的森林，两棵树的根分别代表偶长度回文字符串节点树的根，和奇长度回文字符串节点树的根，**其中每个不同节点都代表一个不同的回文字符串**，经过证明得出长度为n，字符种类为k的字符串中回文串的种类是O(nlogk)级的，即长度为n的字符串的回文树的节点是O(n)级的。

每个回文字符串节点的信息有len，cnt，next[]，fail。其中len代表该回文串的长度，cnt为该回文串出现次数，next[i]，表示该回文串两端扩展字符i后的回文字符串的节点索引，0为无效值。fail代表该回文串的非本身最长回文后缀的节点索引。**回文串a两端加上字符i所代表的回文串在回文树中代表节点a的next[i]所指向的节点。**

初始时让节点0代表长度为0的回文串，fail指向1。让节点1代表长度为-1的回文串，fail的值为1，这里的-1只是方便以后的计算，没有实际意义。

遍历第i个字符求以i结尾的最长回文字符串时。我们设last为第个i-1字符结尾的最长回文字符串代表的节点，now第i个字符求以​i结尾的最长回文字符串代表的节点。那么求now我们将利用last和last的fail，求now的fail将用last的fail。

**构建后的回文树包含原字符串中所有回文串的信息，和该回文串在原字符串中的出现次数。也构建出了回文串之间的关系。**

**模板：**

1. #define mset(a,b) memset(a,b,sizeof(a))
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **int** N=3e5+10;
5. **struct** PA\_tree
6. {
7. **static** **const** **int** branch=26;
8. **static** **const** **int** MAXN=3e5+10;
9. **struct** Node //每个节点代表一个回文串
10. {
11. **int** len;//回文串的长度
12. ll cnt;//回文串出现次数
13. **int** next[branch],fail;
14. //next[c]:该节点左右增加字符c的回文串节点位置,默认为0
15. //fail :该节点非本身的最长回文后缀节点
16. } node[MAXN];
17. **int** ls,top=0;//长度,此时使用的节点个数
18. **char** \*s;//字符首指针,下标从1开始
19. **int** initnode(**int** id)//需要手动初始化fail和len
20. {
21. node[id].cnt=0;
22. mset(node[id].next,0);
23. **return** id;
24. }
25. **int** getfail(**int** last,**int** i)//从last以及其fail开始找字符s[i]的最长回文后缀的位置
26. {  //若s从0开始，i-node[last].len-1<0|| s[i-node[last].len-1]!=s[i]
27. **while**(s[i-node[last].len-1]!=s[i]) last=node[last].fail;
28. **return** last;
29. }
30. **void** init(**char** \*s,**int** ls)
31. {
32. **this**->s=s;
33. **this**->ls=ls;
34. top=0;
35. initnode(top++);
36. initnode(top++);
37. node[0].fail=node[1].fail=1;
38. node[0].len=0,node[1].len=-1;
39. s[0]=-1;  //若s从0开始，这地方不要
40. }
41. **int** gv(**char** c)//计算每个字符的映射值
42. {
43. **return** c-'a';
44. }
45. **void** bulid\_tree()
46. {
47. //目标,构建fail指针并生成回文树
48. **int** last=0;
49. **for**(**int** i=1; i<=ls; ++i)  //若s从0开始，0-ls-1
50. {
52. **int** c=gv(s[i]);
53. **int** cur=getfail(last,i);//得到该位置的最长回文后缀字符串节点的父亲
54. **int** now=node[cur].next[c];
55. **if**(!now)//如果还没有该节点,就新建一个
56. {
57. now=initnode(top++);//新建一个新的节点作为该节点索引
58. node[now].len=node[cur].len+2;
59. node[now].fail=node[getfail(node[cur].fail,i)].next[c];
60. node[cur].next[c]=now;
61. }
62. node[now].cnt++;
63. last=now;
64. }
65. }
66. **void** calc\_count()//计算每回文串在字符串中的出现次数
67. {
68. //基于fail的节点标号一定比自身小,所以我们倒着累加
69. **for**(**int** i=top-1; i; --i)
70. node[node[i].fail].cnt+=node[i].cnt;
71. }
73. };
74. **char** s[N];
75. PA\_tree solve;
76. **int** main()
77. {
78. scanf("%s",s+1);
79. **int** ls=strlen(s+1);
80. solve.init(s,ls);
81. solve.bulid\_tree();
82. solve.calc\_count();
83. //根据题目要求:
84. //剩下的可以对节点进行操作,或者在构建回文串节点时增加信息
85. **return** 0;
86. }

**例题：**

1. /\*
2. 题意:一个字符串的价值为其字符串中出现字符的种类个数，
3. 现在给你一个字符串S，求S中所有回文串的价值。
4. 思路:我们可以构建一颗回文树，并且在构建过程记录每个回文串节点中字符种类个数和出现情况。最后遍历所有回文串节点统计和即可。
5. 代码：
6. \*/
7. #include <bits/stdc++.h>
8. #define mset(a,b) memset(a,b,sizeof(a))
9. **using** **namespace** std;
10. **typedef** **long** **long** ll;
11. **typedef** pair<**int**,**int**> P;
12. **const** **int** N=3e5+10;
13. **struct** PA\_tree
14. {
15. **static** **const** **int** branch=26;
16. **static** **const** **int** MAXN=3e5+10;
17. **struct** Node //每个节点代表一个回文串
18. {
19. ll len,cnt;//回文串的长度,回文串出现次数
20. **int** next[branch],fail;
21. **int** book[branch],dif;
22. //next[c]:该节点左右增加字符c的回文串节点位置,默认为0
23. //fail :该节点非本身的最长回文后缀节点
24. } node[MAXN];
25. **int** ls,top=0;//长度,此时使用的节点个数
26. **char** \*s;//字符首指针,下标从1开始
27. **int** initnode(**int** id)//需要手动初始化fail和len
28. {
29. node[id].cnt=0;
30. mset(node[id].next,0);
31. mset(node[id].book,0);
32. **return** id;
33. }
34. **int** getfail(**int** last,**int** i)
35. {
36. **while**(s[i-node[last].len-1]!=s[i]) last=node[last].fail;
37. **return** last;
38. }
39. **void** init(**char** \*s,**int** ls)
40. {
41. **this**->s=s;
42. **this**->ls=ls;
43. top=0;
44. initnode(top++);
45. initnode(top++);
46. node[0].fail=1;
47. node[1].fail=0x3f3f3f3f;
48. node[0].len=0,node[1].len=-1;
49. node[0].dif=node[1].dif=0;
50. s[0]=-1;
51. }
52. **int** gv(**char** c)
53. {
54. **return** c-'a';
55. }
56. **void** bulid\_tree()
57. {
58. //目标,构建fail指针并生成回文树
59. **int** last=0;
60. **for**(**int** i=1; i<=ls; ++i)
61. {
63. **int** c=gv(s[i]);
64. **int** cur=getfail(last,i);
65. **int** now=node[cur].next[c];
66. **if**(!now)
67. {
68. now=initnode(top++);//新建一个新的节点作为儿子
69. node[now].len=node[cur].len+2;
70. node[now].fail=node[getfail(node[cur].fail,i)].next[c];
71. node[cur].next[c]=now;
72. **if**(node[cur].book[c]==0) node[now].dif=node[cur].dif+1;
73. **else** node[now].dif=node[cur].dif;
74. **for**(**int** j=0;j<branch;++j)
75. node[now].book[j]=node[cur].book[j];
76. node[now].book[c]=1;
78. }
79. node[now].cnt++;
80. last=now;
81. }
82. }
83. **void** calc\_count()
84. {
85. //基于fail的节点标号一定比自身小,所以我们倒着累加
86. **for**(**int** i=top-1; i; --i)
87. node[node[i].fail].cnt+=node[i].cnt;
88. }
89. ll getans()
90. {
91. ll ans=0;
92. **for**(**int** i=2;i<top;++i)
93. ans+=node[i].cnt\*node[i].dif;
94. **return** ans;
95. }
97. };
98. **char** s[N];
99. PA\_tree solve;
100. **int** main()
101. {
102. scanf("%s",s+1);
103. **int** ls=strlen(s+1);
104. solve.init(s,ls);
105. solve.bulid\_tree();
106. solve.calc\_count();
107. printf("%lld\n",solve.getans());
108. **return** 0;
109. }

## 后缀数组

后缀排序后的序列(信息：sa[],height[],rank[])有诸多性质，要灵活应用，并且一定要记得，**一个字符串的子串就是某个后缀的前缀，那么子串之间的关系在后缀数组上非常明显**！

后缀数组经典问题

1. **求两后缀的最长公共前缀——LCP(a,b)。**

LCP(a,b)定义为后缀a与后缀b的最长相同前缀的长度，设， 即后缀 在后缀数组中的排名。不访假设，那么

1. **比较一个字符串的两个子串的大小关系**

假设需要比较的是和的大小关系。

若 ,

否则

1. **不同子串的数目**

子串就是后缀的前缀，所以可以枚举每个后缀，计算前缀总数，再减掉重复，“前缀总数”其实就是子串个数，为 。

如果按后缀排序的顺序枚举后缀，每次新增的子串就是除了与上一个后缀的 LCP 剩下的前缀。这些前缀一定是新增的，否则会破坏的性质。只有这些前缀是新增的，因为 LCP 部分在枚举上一个前缀时计算过了。

所以答案为：

1. **求在字符串中至少出现k次的最长子串的长度**

即height[]中相邻k-1个的最小值的最大值。

答案为：。

可以使用RMQ 做到O(nlogn)预处理，O(1)查询。

1. **查询一个字符串在文本串的出现次数。**

假设文本串为S，查询串为T，那么我们可以在串S的后缀数组上二分出字典序大于等于串T的最小排名（注意此时的二分只考虑前|T|个长度），然后二分字典序大于串T的最小排名，出现次数即这两个位置相减的值。

1. **求两个字符串的最长相同子串 Poj2774**

很容易想出答案是这两个字符串后缀中的最长相同前缀的长度。

我们假设这两个字符串分别为S1，S2，令S=S1+’#’+S2。

接下来求出S的后缀数组，因为对于S2（S1）的某个后缀t2，另一个串S1(S2)与它相同的最长子串一定是S的后缀数组中离后缀t2的最近的两侧的某个后缀串t1的lcp的值。(t1,t2分别是S1,S2的后缀串)

所以我们考虑这样做，从左到右依此遍历后缀数组，维护下S1后缀串的最近的出现过的排名位置p1，S2后缀串最近的出现过的排名位置p2，每遇到一个S1（S2）的后缀串，就求上个另一个后缀串与自己的最长前缀长度即 更新答案即可。

注：区间min可以使用RMQ预处理，总体时间复杂度O((n+m)log(n+m))

1. **某字符串最长重复子串问题**

一个字符串t是S的重复子串当且仅当t在S中至少出现了两次(出现位置不同)

1. **求字符串可重叠最长重复子串的长度**

即height[]数组的最大值

注：DA算法时间复杂度O(nlogn),DC3时间复杂度O(n)

1. **不可重叠最长重复子串的长度**

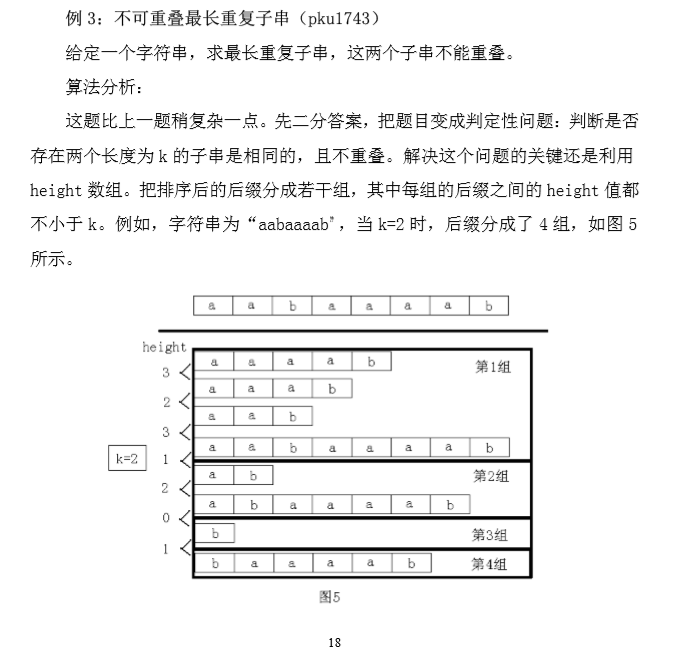
这里不可重叠是指要求字符串t在字符串S中至少两次出现的区间无重叠。

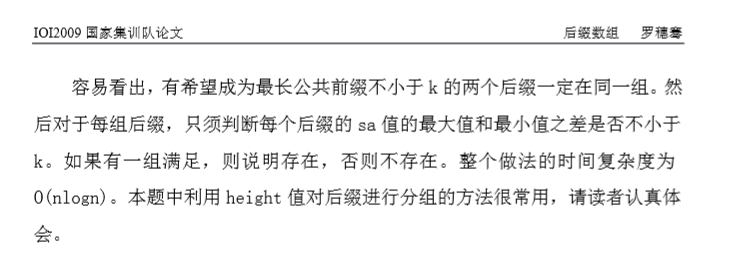
对于这个问题可改为判定性问题，我们可以二分t的长度，对于每个长度lt，我们可以将height数组**分组**，即在height[i]<lt的位置分开，这样每个区间的任意两个后缀串的相同前缀长度都>=lt,**那么只要某个区间的sa[]的最大值和sa[]最小值相差只要>=lt即这个长度可以。（这里可以扩展一下，要求出现的重复子串相隔间距至少一定位置也可以）**

注：二分O(logn)，judge O(n)，总体时间复杂度为O(nlogn)

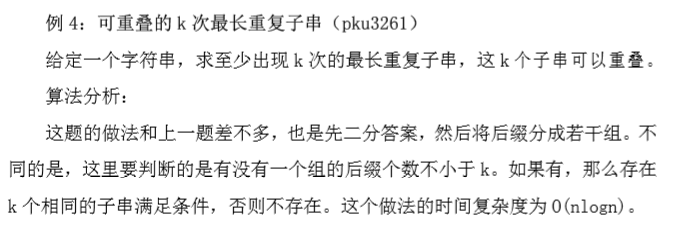
**这个二分+分组的技巧会经常常见。**

*下面有关图片取自—2009年论文集之后缀数组——罗穗骞*





1. **可重叠的k次最长重复子串（pku3261）**



1. **多字符串之间的关系问题**

对于多个字符串，我们常常把这些字符串用特殊字符（比如char(1),char(2)等等，反正只要不出现在字符集里且连接所使用的字符不同即可）连接起来，然后在后缀数组上进行一些操作即可。这么做的作用是对于每个后缀串分隔符处的典序比较肯定出排名结果，这样分隔符处以及之后字符对于这个后缀串就不起作用了

注：通常也使用二分长度+分组的技巧，具体请看下面多字符串的关系

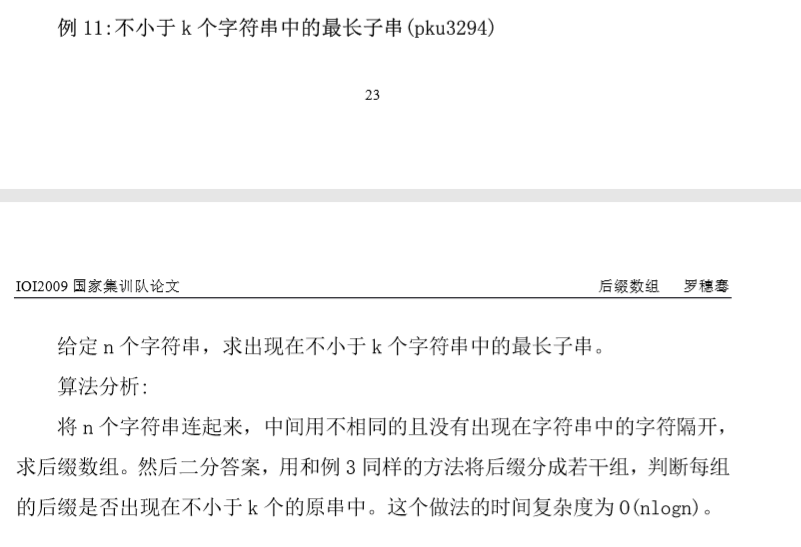
* 1. **给定 n 个字符串，求出现在不小于 k 个字符串中的最长子串。**

如果k=1的话就直接输出最长即可。

否则我们将这些字符串用特殊且不相等的字符连接起来，求他们的后缀数组，接下来二分满足要求的子串的长度（改成判定性问题），对于每个长度w，我们将height[]<w处分隔开从而将height[]数组分组，我们只需判断这些组内是否有k个不同字符串的后缀即可（因为每组后缀的最长相同前缀长度>=w，容易看出答案一定也在某个组中，所以该算法正确）。

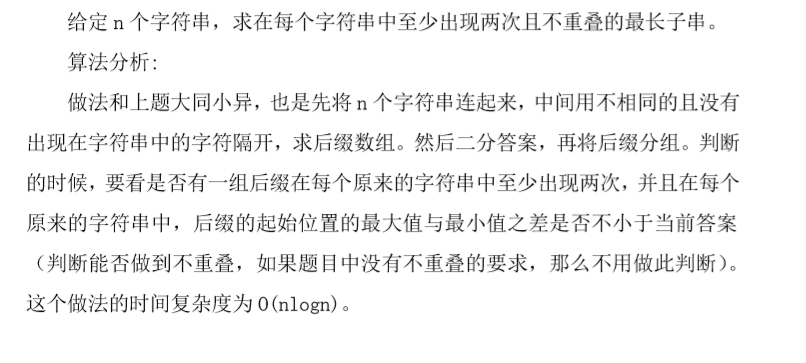
注：时间复杂度O(nlogn)

*下面有关图片取自—2009年论文集之后缀数组——罗穗骞*



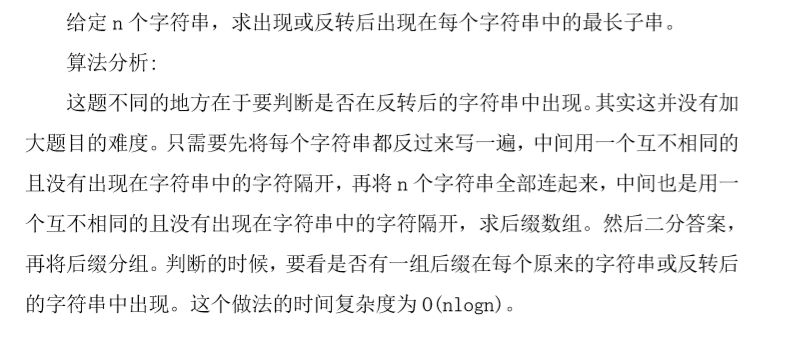
* 1. **每个字符串至少出现两次且不重叠的最长子串(spoj220)**

*下面有关图片取自—2009年论文集之后缀数组——罗穗骞*



* 1. **出现或反转后出现在每个字符串中的最长子串(PKU3294**

*下面有关图片取自—2009年论文集后缀数组——罗穗骞*



### DA算法

时间复杂度O(nlogn)，空间复杂度O(n)

后缀数组DA算法：’’

1. //子串与子串的关系可以转化为两后缀的前缀的关系,所以后缀数组在字符串方面有很大用处
2. /\*
3. 注: 排名从1开始,后缀下标从1开始.
5. 名词解释:下面后缀i代指原串下标i开始的后缀,排名指字典序从小到大的排名.
6. s[] :原串数组,下标从1开始,要在下标n+1处有一最小字符,比如'\0'.
7. n   :字符串长度.
8. sa[]:sa[i]代表排名为i的后缀下标为sa[i]
9. rank[]: 后缀i的排名为rank[i]
10. height[]: height[i]代表排名i与排名i-1后缀的最长相同前缀的长度.
11. 空间复杂度O(n),时间复杂度O(nlogn)
12. \*/
13. **const** **int** N=1e5+10;
14. **int** t1[N],t2[N],c1[N];//辅助数组
15. **void** SA(**char** \*s,**int** n,**int** sa[],**int** rank[],**int** height[])//基数排序的版本中
16. {
17. **int** m=128;//桶的大小,会在下面循环中变化,第一关键词r的最大值,初始时是字符的最大值,之后都<=n
18. **int** \*sb=t1,\*r=t2,\*c=c1;//辅助数组,分别为:基数排序所用的第二2,rak',cnt数组
19. //用基数排序求出长度为1的sa[]和rank[],如果字符最大值较大,第一轮可以采用sort
20. **for**(**int** i=0; i<=m; ++i) c[i]=0;
21. **for**(**int** i=1; i<=n; ++i) c[r[i]=s[i]]++;
22. **for**(**int** i=1; i<=m; ++i) c[i]+=c[i-1];
23. **for**(**int** i=n; i>=1; --i) sa[c[s[i]]--]=i;
25. **for**(**int** k=1,p ; k < n; k<<=1 ) //p是一个计数器，现在还没用。
26. {
27. //sb[i]:第二关键词排名为i的位置为sb[i]
28. p=0;
29. **for**(**int** i=n-k+1; i<=n; ++i) sb[++p]=i;
30. **for**(**int** i=1; i<=n; ++i) **if**(sa[i]>k) sb[++p]=sa[i]-k;
31. //基数排序求出2k长度的sa数组
32. **for**(**int** i=0; i<=m; ++i) c[i]=0;
33. **for**(**int** i=1; i<=n; ++i) c[r[i]]++;
34. **for**(**int** i=1; i<=m; ++i) c[i]+=c[i-1];
35. **for**(**int** i=n; i>=1; --i) sa[ c[r[sb[i]]]-- ]=sb[i];
36. std::swap(r,sb);
37. //现在要利用上轮的r和这轮的sa求这轮的r
38. r[sa[1]]=p=1;
39. **for**(**int** i=2,a,b; i<=n; ++i)
40. {
41. a=sa[i],b=sa[i-1];
42. **if**(sb[a]==sb[b]&&(a+k<=n && b+k<=n&&sb[a+k]==sb[b+k]) ) r[a]=p;
43. **else** r[a]=++p;
44. }
45. **if**(p>=n) **break**;//可以提前退出
46. m=p;
47. }
48. /\*计算高度数组\*/
49. **int** k=0;
50. **for**(**int** i=1; i<=n; ++i) rank[sa[i]]=i;
51. height[1]=0;
52. **for**(**int** i=1; i<=n; ++i)
53. {
54. **if**(k) k--;
55. **int** j=sa[rank[i]-1];
56. **if**(j==0) **continue**;
57. **while**(s[j+k]==s[i+k]) ++k;
58. height[rank[i]]=k;
59. }
60. }
61. **int** rank[N],sa[N],height[N]; //height[1]的值为0
62. **char** s[N];
63. //预处理O(nlogn)- - 查询O(1)求height[]的区间最小值的RMQ代码
64. **int** RMQ[N],mm[N],best[N][20];//rmq[]=height[]
65. **void** initRMQ(**int** n, **int** height[])
66. {
67. **for**(**int** i=1; i<=n; ++i) RMQ[i]=height[i];
68. mm[0]=-1;
69. **for**(**int** i=1; i<=n; ++i) mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
70. **for**(**int** i=1; i<=n; ++i) best[i][0]=i;
71. **for**(**int** j=1; j<=mm[n]; ++j)
72. **for**(**int** i=1; i+(1<<j)-1<=n; ++i)
73. {
74. **int** a=best[i][j-1];
75. **int** b=best[i+(1<<(j-1))][j-1];
76. **if**(RMQ[a] < RMQ[b]) best[i][j]=a;
77. **else** best[i][j]=b;
78. }
79. }
80. **int** askRMQ(**int** a,**int** b)//求height[]中区间[a,b]最小值的下标
81. {
82. **int** t;
83. t=mm[b-a+1];
84. b-=(1<<t)-1;
85. a=best[a][t],b=best[b][t];
86. **if**(RMQ[a] < RMQ[b]) **return** a;
87. **else** **return** b;
88. }
89. **int** lcp(**int** a,**int** b)
90. {
91. a=rank[a],b=rank[b];
92. **if**(a > b) std::swap(a,b);
93. **return** height[askRMQ(a+1,b)];
94. }

如果传入的数组字符最大值太大，第一波可以使用快排（当然离散化也是可以的）

1. //第一波基数排序改为快排
2. **const** **int** N=1e6+10;
3. **int** t1[N],t2[N],c1[N];//辅助数组
4. **void** SA(**int** \*s,**int** n,**int** sa[],**int** rank[],**int** height[])//基数排序的版本中
5. {
6. **int** m=n;//桶的大小,会在之后循环变化,<=n
7. **int** \*sb=t1,\*r=t2,\*c=c1;
8. /\*串最大值较大,第一轮采用sort\*/
9. **for**(**int** i=1; i<=n; ++i) sa[i]=i;
10. auto cmp=[&](**int** x,**int** y)
11. {
12. **return** s[x]<s[y]||(s[x]==s[y]&&x < y);
13. };
14. std::sort(sa+1,sa+n+1,cmp);
15. **int** p;
16. r[sa[1]]=p=1;
17. **for**(**int** i=2; i<=n; ++i)
18. {
19. **if**(s[sa[i-1]]==s[sa[i]]) r[sa[i]]=p;
20. **else** r[sa[i]]=++p;
21. }
23. **for**(**int** k=1,p ; k < n; k<<=1 ) //p是一个计数器，现在还没用。
24. {
26. p=0;
27. **for**(**int** i=n-k+1; i<=n; ++i) sb[++p]=i;
28. **for**(**int** i=1; i<=n; ++i) **if**(sa[i]>k) sb[++p]=sa[i]-k;
30. **for**(**int** i=0; i<=m; ++i) c[i]=0;
31. **for**(**int** i=1; i<=n; ++i) c[r[i]]++;
32. **for**(**int** i=1; i<=m; ++i) c[i]+=c[i-1];
33. **for**(**int** i=n; i>=1; --i) sa[ c[r[sb[i]]]-- ]=sb[i];
34. std::swap(r,sb);
36. r[sa[1]]=p=1;
37. **for**(**int** i=2,a,b; i<=n; ++i)
38. {
39. a=sa[i],b=sa[i-1];
40. **if**(sb[a]==sb[b]&&(a+k<=n && b+k<=n&&sb[a+k]==sb[b+k]) ) r[a]=p;
41. **else** r[a]=++p;
42. }
43. **if**(p>=n) **break**;//可以提前退出
44. m=p;
45. }
46. /\*计算高度数组\*/
47. **int** k=0;
48. **for**(**int** i=1; i<=n; ++i) rank[sa[i]]=i;
49. **for**(**int** i=1; i<=n; ++i)
50. {
51. **if**(k) k--;
52. **int** j=sa[rank[i]-1];
53. **if**(j==0) **continue**;
54. **while**(s[j+k]==s[i+k]) ++k;
55. height[rank[i]]=k;
56. }
57. }

### DC3算法

1. **const** **int** Max = 200001;
2. **int** num[Max];
3. **int** r[Max \* 3], sa[Max \* 3];
4. **int** rank[Max], height[Max];
5. **int** wa[Max], wb[Max], wv[Max], wd[Max];
6. #define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
7. #define G(x) ((x) < tb ? (x) \* 3 + 1 : ((x) - tb) \* 3 + 2)
8. **int** c0(**int** \*r, **int** a, **int** b)
9. { **return** r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2]; }
10. **int** c12(**int** k, **int** \*r, **int** a, **int** b)
11. { **if** (k == 2) **return** r[a] < r[b] || r[a] == r[b] && c12(1, r, a + 1, b + 1);
12. **else** **return** r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];}
13. **void** sort(**int** \*r,**int** \*a,**int** \*b,**int** n,**int** m)
14. {
15. **int** i;
16. **for**(i=0;i<n;i++) wv[i]=r[a[i]];
17. **for**(i=0;i<m;i++) wd[i]=0;
18. **for**(i=0;i<n;i++) wd[wv[i]]++;
19. **for**(i=1;i<m;i++) wd[i]+=wd[i-1];
20. **for**(i=n-1;i>=0;i--) b[--wd[wv[i]]]=a[i];
21. **return**;
22. }
23. **void** dc3(**int** \*r,**int** \*sa,**int** n,**int** m)
24. {
25. **int** i,j,\*rn=r+n,\*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
26. r[n]=r[n+1]=0;
27. **for**(i=0;i<n;i++) **if**(i%3!=0) wa[tbc++]=i;
28. sort(r+2,wa,wb,tbc,m);
29. sort(r+1,wb,wa,tbc,m);
30. sort(r,wa,wb,tbc,m);
31. **for**(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
32. rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
33. **if**(p<tbc) dc3(rn,san,tbc,p);
34. **else** **for**(i=0;i<tbc;i++) san[rn[i]]=i;
35. **for**(i=0;i<tbc;i++) **if**(san[i]<tb) wb[ta++]=san[i]\*3;
36. **if**(n%3==1) wb[ta++]=n-1;
37. sort(r,wb,wa,ta,m);
38. **for**(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
39. **for**(i=0,j=0,p=0;i<ta && j<tbc;p++)
40. sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
41. **for**(;i<ta;p++) sa[p]=wa[i++];
42. **for**(;j<tbc;p++) sa[p]=wb[j++];
43. **return**;
44. }

## 字符串HASH

**Hash函数**

给定一个字符串，对字母，**我们规定。** （当然也可以直接用si的ASCII值）

，在模mod下

，在模mod下

其中p和mod都是我们指定的参数，一般情况下将p和mod尽量取大即可，这样冲突的概率是很低的。

注：如果对于数字表示的字符串，id(0)设为0，那么我们可以O(1)比较两子串的值是否相同（即忽略前缀0），当然这样的也就会把001与1看作一样的了。

**都有哪些Hash方法？**

**自然溢出方法**

我们可以将hash[] 的类型定为unsigned long long ，也就是。并且我们选取为模数，这样不用每次都进行取模操作( 溢出自动取模) 。这在Acm解决问题是常见的。

**BKDR Hash函数怎么O(1)获得子串的hash值？**

假设有一|S|=5的字符串，设Si为第i个字符，其中1≤i≤5。

根据定义分别求出hash[i]

hash[1]=s1

hash[2]=s1∗p+s2

hash[3]=s1∗p2+s2∗p+s3

hash[4]=s1∗p3+s2∗p2+s3∗p+s4

hash[5]=s1∗p4+s2∗p3+s3∗p2+s4∗p+s5

假设我们现在想获得字串的hash值，简单计算我们可以知道的hash值为, 我们再看下上面列出来的公式，咦~，这不是hash[4]的后缀吗！该后缀前面的部分是 ! ，为此我们发现所有的子串都可以按照这样的方式得出该字串的hash值。

**为此我们得出子串** **的hash值为:** .

当然因为减法可能会出现负数，为了一致我们将统一处理：

# 数论

## 打表（素数，欧拉函数，莫比乌斯）

### 素数打表

1. #define mset(a,b)   memset(a,b,sizeof(a))
2. **const** **int** maxn=100000;
3. **bool** check[maxn+10];
4. **int** prime[maxn+10],tot;
5. **void** init()//线性筛素数
6. {
7. mset(check,0);
8. tot=0;
9. **for**(**int** i=2;i<=maxn;++i)
10. {
11. **if**(!check[i])   prime[tot++]=i;
12. **for**(**int** j=0;j<tot;j++)
13. {
14. **if**(i\*prime[j]>maxn) **break**;
15. check[i\*prime[j]]=**true**;
16. **if**(i%prime[j]==0)
17. **break**;
18. }
19. }
20. }

### 素数，欧拉函数打表，莫比乌斯函数打表

1. #define mset(a,b)   memset(a,b,sizeof(a))
2. **const** **int** maxn=100000;
3. **const** **int** inf=0x3f3f3f3f;
4. **bool** check[maxn+10];
5. **int** prime[maxn+10],mu[maxn+10];
6. **int** phi[maxn+10];
7. **void** init()//欧拉 素数 莫比乌斯反演函数
8. {
9. mset(check,0);
10. mu[1]=1;
11. phi[1]=1;
12. **int** tot=0;
13. **for**(**int** i=2;i<=maxn;++i)
14. {
15. **if**(!check[i])
16. {
17. prime[tot++]=i;
18. phi[i]=i-1;
19. mu[i]=-1;
20. }
21. **for**(**int** j=0;j<tot;j++)
22. {
23. **if**(i\*prime[j]>maxn) **break**;
24. check[i\*prime[j]]=**true**;
25. **if**(i%prime[j]==0)  //优化 此时break
26. {
27. mu[i\*prime[j]]=0;
28. phi[i\*prime[j]]=phi[i]\*prime[j];
29. **break**;
30. }
31. **else**
32. {
33. mu[i\*prime[j]]=-mu[i];
34. phi[i\*prime[j]]=phi[i]\*phi[prime[j]];//积性函数的性质
35. }
36. }
37. }
38. }

### 欧拉函数打表

线性筛欧拉函数把上面代码的莫比乌斯去了即可

单个求欧拉函数

1. **long** **long** eular(**long** **long** n)
2. {
3. **long** **long** ans=n;
4. **for**(**int** i=2;i\*i<=n;++i)
5. {
6. **if**(n%i==0)
7. {
8. ans-=ans/i;
9. **while**(n%i==0)
10. n/=i;
11. }
12. }
13. **if**(n>1)
14. ans-=ans/n;
15. **return** ans;
16. }

筛法求欧拉函数

1. **const** **int** maxn=1e5;
2. **int** euler[maxn+10];
3. **void** getEuler()
4. {
5. mset(euler,0);
6. euler[1]=1;
7. **for**(**int** i=2; i<=maxn; ++i)
8. **if**(!euler[i])
9. **for**(**int** j=i; j<=maxn; j+=i)
10. {
11. **if**(!euler[j])
12. euler[j]=j;
13. euler[j]=euler[j]/i\*(i-1);
14. }
15. }

用素因子求欧拉函数

1. **const** **int** maxn=1e6;
2. **int** book[maxn+10],prime[(**int**)1e5+10];
3. **int** top;
4. **void** init()//求出100w以内的所有素因子
5. {
6. top=0;
7. book[0]=book[1]=1;
8. **for**(**int** i=2;i\*i<=maxn;++i)
9. **if**(!book[i])
10. {
11. **for**(**int** j=i\*i;j<=maxn;j+=i)
12. book[j]=1;
13. }
14. **for**(**int** i=2;i<=maxn;++i)
15. **if**(!book[i])
16. prime[top++]=i;
17. }
18. **int** phi(**int** val)//求1~n中与n互质的个数
19. {
20. **int** ans=val;
21. **for**(**int** i=0;prime[i]\*prime[i]<=val;++i)
22. {
23. **if**(!(val%prime[i]))
24. {
25. ans=ans/prime[i]\*(prime[i]-1);
26. **while**(!(val%prime[i]))
27. {
28. val/=prime[i];
29. }
30. }
31. }
32. **if**(val>1)
33. ans=ans/val\*(val-1);
34. **return** ans;
35. }

### 求出N的约数的莫比乌斯函数的值

1. //把n的约数的莫比乌斯反演反演函数的值使用map的形式返回.O(sqrt(n))
2. map<**int**,**int**> moebius(**int** n)
3. {
4. map<**int**,**int**> res;
5. vector<**int**> primes;
6. **for**(**int** i=2; i\*i<=n; ++i)
7. {
8. **if**(n%i==0)
9. {
10. primes.push\_back(i);
11. **while**(n%i==0) n/=i;
12. }
13. }
14. **if**(n!=1) primes.push\_back(n);
15. **int** m=primes.size();
16. **for**(**int** i=0; i< (1<<m) ; ++i)
17. {
18. **int** mu=1,d=1;
19. **for**(**int** j=0; j<m; ++j)
20. {
21. **if**(i >>j & 1){
22. mu\*=-1;
23. d\*=primes[j];
24. }
25. }
26. res[d]=mu;
27. }
28. **return** res;
29. }
30. ll qpow(ll a,ll b)
31. {
32. ll ans=1;
33. **while**(b)
34. {
35. **if**(b&1)
36. ans=ans\*a%mod;
37. a=a\*a%mod;
38. b>>=1;
39. }
40. **return** ans;
41. }
42. **int** F(ll n)
43. {
44. **return** (**int**)qpow(2,n-1);
45. }
46. ll f(**int** n)
47. {
48. map<**int**,**int**> mu;
49. mu=moebius(n);
50. **int** ans=0;
51. **for**(auto it=mu.begin();it!=mu.end();++it)
52. {
53. ans += it->second \* (F( n / it->first));
54. ans=(ans%mod+mod)%mod;
55. }
56. **return** ans;
57. }
58. **int** main()
59. {
60. **int** x,y;
61. cin>>x>>y;
62. **if**(y%x!=0)
63. cout<<0<<endl;
64. **else**
65. cout<<f(y / x)<<endl;
66. }

### 线性筛出每个数的最小素因子（快速求每个数的素因子分解形式）

筛出每个数的最小素因子。

1. **int** mf[maxn+10];//最小素因子
2. vector<**int**> prime;
3. **void** init()
4. {
5. mset(mf,0);
6. **for**(**long** **long** i=2;i<=maxn;++i)
7. {
8. **if**(!mf[i]){
9. prime.push\_back(i);
10. mf[i]=i;
11. }
12. **for**(**int** j=0;j<**int**(prime.size());++j){
13. **if**(prime[j]\*i>maxn) **break**;
14. mf[prime[j]\*i]=prime[j];
15. **if**(i%prime[j]==0) **break**;
16. }
17. }
18. }

对于一个数求其素因子标准分解形式，每次除以其最小素因子即可求出。

1. cin>>x;
2. **while**(x > 1)
3. {
4. **int** cnt=0;
5. **int** m=mf[x];
6. **while**(x % m==0){
7. cnt++;
8. x/=m;
9. }//素因子m在x中出现了cnt次
10. }

## 扩展欧几里得算法（求 ax+by=gcd 的解以及逆元素）

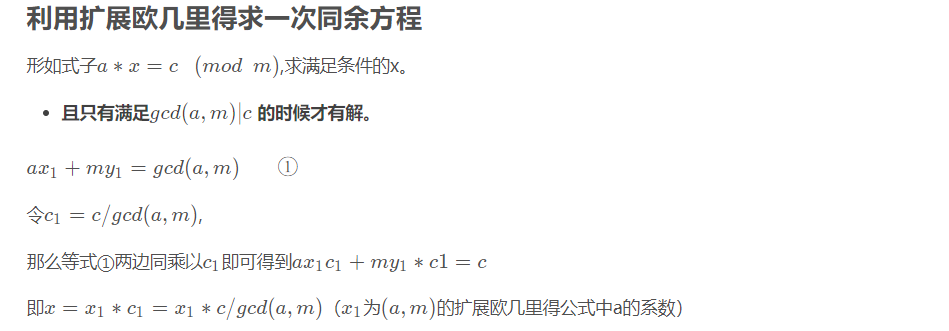
1. //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
2. //返回d=gcd(a,b);和对应于等式ax+by=d中的x,y
3. **long** **long** extend\_gcd(**long** **long** a,**long** **long** b,**long** **long** &x,**long** **long** &y)
4. {//函数结束后满足 x\*a+y\*b=gcd
5. **if**(a==0&&b==0) **return** -1;
6. //无最大公约数
7. **if**(b==0)
8. {
9. x=1;
10. y=0;
11. **return** a;
12. }
13. **long** **long** d=extend\_gcd(b,a%b,y,x);
14. y-=a/b\*x;
15. **return** d;
16. }
17. //\*\*\*\*\*\*\*\*\*求逆元素\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
18. //ax = 1(mod n)
19. **long** **long** mod\_reverse(**long** **long** a,**long** **long** n)
20. {
21. **long** **long** x,y;
22. **long** **long** d=extend\_gcd(a,n,x,y);
23. **if**(d==1) **return** (x%n+n)%n;
24. **else** **return** -1;
25. }

## 求逆元

1. 4.1 扩展欧几里德法（见上面）
2. 4.2 简洁写法 注意：这个只能求a < m的情况，而且必须保证a和m互质
3. //求ax = 1( mod m) 的x值，就是逆元(0<a<m)
4. **long** **long** inv(**long** **long** a,**long** **long** m)
5. {
6. **if**(a == 1)**return** 1;
7. **return** inv(m%a,m)\*(m-m/a)%m;
8. }
9. 4.3 利用欧拉函数 mod为素数,而且a和m互质
10. **long** **long** inv(**long** **long** a,**long** **long** mod)//mod为素数
11. {
12. **return** pow\_m(a,mod-2,mod);
13. }

## 一次同余方程 求a\*x=c（mod m）

前提：gcd(a,m) **|** c ，即gcd(a,m) 是c的因子



1. #include<cstdio>
2. #include<algorithm>
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. ll ExGcd(ll a,ll b,ll &x,ll &y)//求 a b 最大公约数，且得到gcd(a,b)=x\*a+y\*b;
6. {
7. **if**(!b)
8. {
9. x=1;
10. y=0;
11. **return** a;
12. }
13. ll gcd=ExGcd(b,a%b,x,y);
14. ll temp,k=a/b;
15. temp=x;
16. x=y;
17. y=temp-k\*y;
18. **return** gcd;
19. }
20. **bool** IsOk;
21. ll calc(ll a,ll c,ll m)//Isok=false 表示无解
22. {
23. ll x,y;
24. ll gcd=ExGcd(a,m,x,y);
25. **if**(c%gcd!=0)
26. {
27. IsOk=**false**;
28. **return** 0ll;
29. }
30. **return** c/gcd\*x%m;
31. }

## 模线性方程组（中国剩余定理）

一次同余方程组结果的模数是所有一次同余方程模数的最小公倍数，即。

扩展中国剩余定理，每次合并两个一次同余方程组, 迭代到最后的一次同余方程就是解，可**处理模数不互质的情况**。

1. **long** **long** extend\_gcd(**long** **long** a,**long** **long** b,**long** **long** &x,**long** **long**  &y)
3. {
4. **if**(a == 0 && b == 0)**return** -1;
5. **if**(b ==0 )
6. {
7. x = 1;
8. y = 0;
9. **return** a;
10. }
11. **long** **long** d = extend\_gcd(b,a%b,y,x);
12. y -= a/b\*x;
13. **return** d;
14. }
15. **int** m[10],a[10];//模数为m,余数为a, X % m = a
16. **bool** solve(**int** &m0,**int** &a0,**int** m,**int** a)
17. {
18. **long** **long** y,x;
19. **int** g = extend\_gcd(m0,m,x,y);//得到的x是m0/g关于m1/g的逆元
20. **if**( abs(a - a0)%g )**return** **false**;
21. x \*= (a - a0)/g;
22. x %= m/g;
23. a0 = (x\*m0 + a0);
24. m0 \*= m/g;  //合并方程后的模数是增大的
25. a0 %= m0;
26. **if**( a0 < 0 )a0 += m0;  //a0也是慢慢增大的
27. **return** **true**;
28. }
29. /\*
30. \* 无解返回false,有解返回true;
31. \* 解的形式最后为 a0 + m0 \* t  (0<=a0<m0)
32. \*/
33. **bool** MLES(**int** &m0,**int** &a0,**int** n) //解为  X = a0 + m0 \* k
34. {
35. **bool** flag = **true**;
36. m0 = 1;
37. a0 = 0;
38. **for**(**int** i = 0; i < n; i++)
39. **if**( !solve(m0,a0,m[i],a[i]) )
40. {
41. flag = **false**;
42. **break**;
43. }
44. **return** flag;
45. }

有时候可能long long相乘爆long long，那么这时候就需要**龟速乘取模（可能这也是很多CRT思路对了但答案错了的地方吧）**

1. ll mul(ll a,ll b,ll m)
2. {
3. ll ans=0,det=1;
4. **if**(a < 0){ det\*=-1; a=-a; }
5. **if**(b < 0){ det\*=-1; b=-b; }
6. **while**(a)
7. {
8. **if**(a&1) ans=(ans+b)%m;
9. b=b\*2%m;
10. a/=2;
11. }
12. **return** (ans\*det%m+m)%m;
13. }

模数互质的中国剩余定理

思路：

对于n个一次同余方程组，

我们令，，，那么其通解的表达形式

证明：对于对于第i 个方程组的贡献是，其他方程组的贡献都是0，故改解满足一次同余方程组。

模数不互质的一次同余方程组分解质因子转化为模数互质的一次同余方程组

思路：

对于n个一次同余方程组，

假设第i 个模数的标准分解式为 ，其中为质数，且。

引理：设p为质数，且，对于一次同余方程组，那么在方程有解的情况下方程等价为。(两个一次同余方程组有解则其解的表达形式中模数为一次同余方程模数的最小公倍数)

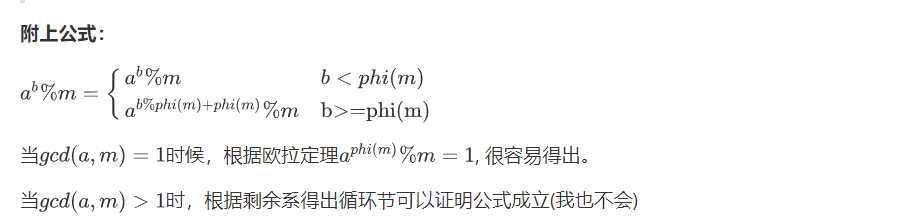
那么方程分两步

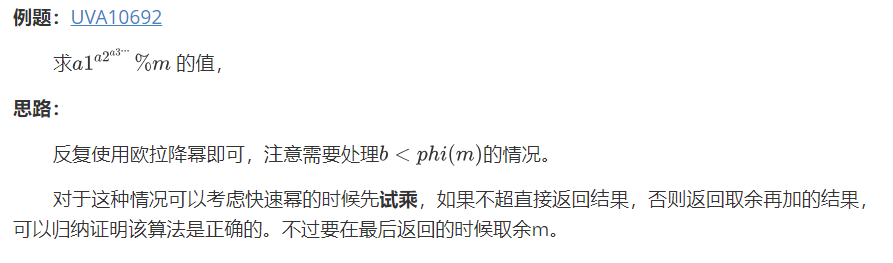
1. 求出所有出现过的素因子在所有模数的标准分解式的最大次幂，形式化的：假设现有，那么对于满足
2. 列出所有k个一次同余方程，且第k个方程的表达形式为，我们首先检查对于之前的n个方程，其是否相同，不相同则无解，否则则用中国剩余定理求出其解。

## 高斯消元

1. /\*
2. 输入：N个未知量，n个方程，表示为一个n\*(n+1)的矩阵
3. 输出：
4. \*/
5. #include<bits/stdc++.h>
6. **using** **namespace** std;
7. **const** **double** eps=1e-7;
8. **double** g[105][105],ans[105];
9. **bool** Gauss(**int** n)//有解返回true  并将答案存进false
10. {
11. **for**(**int** i=1; i<=n; ++i)
12. {
13. **int** r=i;
14. **for**(**int** j=i+1; j<=n; ++j) **if**(fabs(g[r][i]) < fabs(g[r][j])) r=i; //把当第i行换成第i~n行的第i个元素最大的那一行(如果有解等效于这个不等于0)
15. **if**(fabs(g[r][i]) < eps) **return** **false**;
16. swap(g[i],g[r]);
17. **double** div=g[r][i];
18. **for**(**int** j=i; j<=n+1; ++j) g[i][j]/=div; //把第i行的这个元素换成单位为1
19. **for**(**int** j=i+1; j<=n; ++j)
20. {
21. div=g[j][i];//第j行减去 div倍的第i行
22. **for**(**int** p=i; p<=n+1; ++p) g[j][p]-=div\*g[i][p];
23. }
24. }
25. //回溯答案
26. ans[n]=g[n][n+1];
27. **for**(**int** i=n-1; i>=1; --i)
28. {
29. ans[i]=g[i][n+1];
30. **for**(**int** j=i+1; j<=n; ++j) ans[i]-=g[i][j]\*ans[j];
31. }
32. **return** **true**;
33. }
34. **int** main()
35. {
36. **int** n;
37. scanf("%d",&n);
38. **for**(**int** i=1; i<=n; ++i) **for**(**int** j=1; j<=n+1; ++j) scanf("%lf",&g[i][j]);
39. **if**(Gauss(n))
40. {
41. **for**(**int** i=1; i<=n; ++i) printf("%.2f\n",ans[i]);
42. }
43. **else**
44. puts("No Solution");
45. **return** 0;
46. }

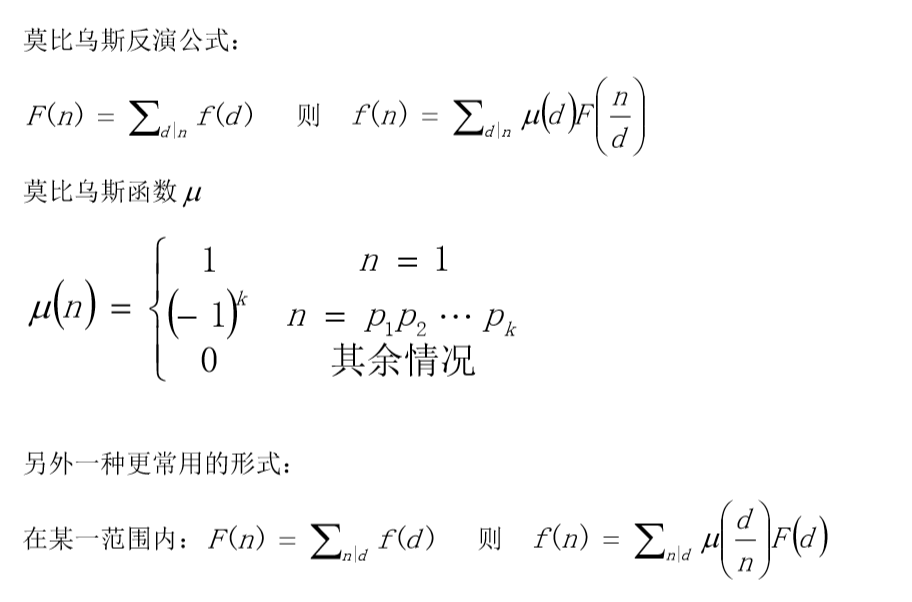
## 欧拉降幂





1. **typedef** **long** **long** ll;
2. **const** **int** maxn=1e6+10;
3. ll a[maxn],phi[maxn];
4. ll exmod(ll a,ll m)
5. {
6. **return** a>=m?a%m+m:a;
7. }
8. ll exqpow(ll a,ll b,ll mod)//a^b>=m 返回a^b%m+m,否则返回a^b
9. {
10. ll ans=1;
11. **while**(b)
12. {
13. **if**(b&1)
14. ans=exmod(ans\*a,mod);
15. b>>=1;
16. a=exmod(a\*a,mod);
17. }
18. **return** ans;
19. }
20. ll getphi(ll x)
21. {
22. **if**(phi[x]!=-1) **return** phi[x];
23. ll xx=x,ans=x;
24. **for**(ll i=2; i\*i<=x; i++)
25. {
26. **if**(x%i==0)
27. {
28. ans=ans\*(i-1)/i;
29. **while**(x%i==0)
30. x/=i;
31. }
32. }
33. **if**(x>1)
34. ans=ans\*(x-1)/x;
35. phi[xx]=ans;
36. **return** ans;
37. }
38. **int** n;//a[]的个数
39. //val=a[k]^(a[k+1]^(a[k+2]^(...^a[n])))
40. //现在准备计算val%m,且val>=m返回val%m+m,否则返回val
41. ll solve(**int** k,ll m)
42. {
43. //并且注意m==1时,a>0应该返回m.这里在exmod会自动计算
44. **if**(k==n||m==1) **return** exmod(a[k],m);//大优化，从满数据迭代到1不会太多次
45. **return** exqpow(a[k],solve(k+1,getphi(m)),m);
46. }
47. **char** s[110];
48. **int** main()
49. {
50. memset(phi,-1,**sizeof**(phi));
51. **int** cas=0;
52. **while**(scanf("%s",s)&&s[0]!='#')
53. {
54. //m为模数,n个数,求a1^(a2^(a3^(...)))%m
55. ll m;
56. sscanf(s,"%lld",&m);
57. cin>>n;
58. **for**(**int** i=1;i<=n;++i)
59. cin>>a[i];
60. cout<<"Case #"<<++cas<<": "<<solve(1,m)%m<<endl;
61. }
62. **return** 0;
63. }

## 莫比乌斯反演，求n的约数的莫比乌斯函数值



900D. Unusual Sequences（莫比乌斯反演）题目链接：[传送门](https://codeforces.com/contest/900/problem/D)

**题意**：

给出 和 ,求序列形如 ()满足 且 的序列的个数。( 没有约束)

数据范围：

**思路：**

令，我们很容易知道问题可以转化为。

我们用 表示序列和为 , 的序列的个数。表示序列和为 的序列的个数。

对于，我们用隔板法可知,，且序列的所有可能的 都是 的约数，所以

我们用莫比乌斯反演可将之转化为

所以我们只需筛出n的所有约数的莫比乌斯函数，然后遍历求出即可

**代码**：

1. #include<bits/stdc++.h>
2. #define mse(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **const** **int** N=1e5+10;
6. **const** ll mod=1e9+7;
7. //把n的约束的莫比乌斯反演反演函数的值使用map的形式返回.O(sqrt(n))
8. map<**int**,**int**> moebius(**int** n)
9. {
10. map<**int**,**int**> res;
11. vector<**int**> primes;
12. **for**(**int** i=2; i\*i<=n; ++i)
13. {
14. **if**(n%i==0)
15. {
16. primes.push\_back(i);
17. **while**(n%i==0) n/=i;
18. }
19. }
20. **if**(n!=1) primes.push\_back(n);
21. **int** m=primes.size();
22. **for**(**int** i=0; i< (1<<m) ; ++i)
23. {
24. **int** mu=1,d=1;
25. **for**(**int** j=0; j<m; ++j)
26. {
27. **if**(i >>j & 1){
28. mu\*=-1;
29. d\*=primes[j];
30. }
31. }
32. res[d]=mu;
33. }
34. **return** res;
35. }
36. ll qpow(ll a,ll b)
37. {
38. ll ans=1;
39. **while**(b)
40. {
41. **if**(b&1)
42. ans=ans\*a%mod;
43. a=a\*a%mod;
44. b>>=1;
45. }
46. **return** ans;
47. }
48. **int** g(ll n)
49. {
50. **return** (**int**)qpow(2,n-1);
51. }
52. ll f(**int** n)
53. {
54. map<**int**,**int**> mu;
55. mu=moebius(n);
56. **int** ans=0;
57. **for**(auto it=mu.begin();it!=mu.end();++it)
58. {
59. ans += it->second \* (g( n / it->first));
60. ans=(ans%mod+mod)%mod;
61. }
62. **return** ans;
63. }
64. **int** main()
65. {
66. **int** x,y;
67. cin>>x>>y;
68. **if**(y%x!=0)
69. cout<<0<<endl;
70. **else**
71. cout<<f(y / x)<<endl;
72. }

## 狄利克雷卷积(未添加)

## 部分数列和公式

1. **第二类斯特林数**

Stirling数是把基数为n的集划分为正好k个非空集的方案数

递推公式：。且

其递推意义是第n个数单独一个集合，或第n个数和某个数在一个集合)

1. **贝尔数**

Bell number数是基数为n的集合的划分方法的数目。集合*S*的一个划分是定义为*S*的两两不相交的非空子集的族，它们的并是*S*。例如*B*3 = 5因为3个元素的集合{*a*, *b*, *c*}有5种不同的划分方法：

{{*a*}, {*b*}, {*c*}}

{{*a*}, {*b*, *c*}}

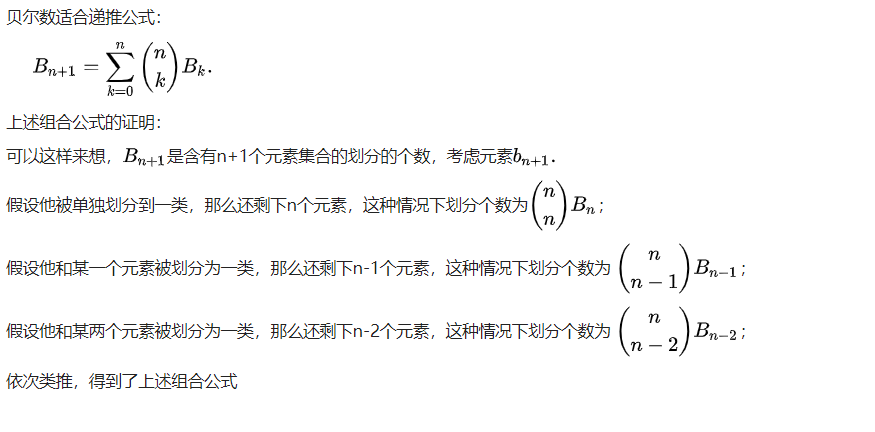
{{*b*}, {*a*, *c*}}

{{*c*}, {*a*, *b*}}

{{*a*, *b*, *c*}};

B(0)=1,因为空集的每个成员都是非空集合。

**第一种形式：**



**Bell数适合“Touchard同余”：若*p*是任意质数，那么**

每个Bell数又是 第二类Stirling数的和，即枚举使用的集合数目

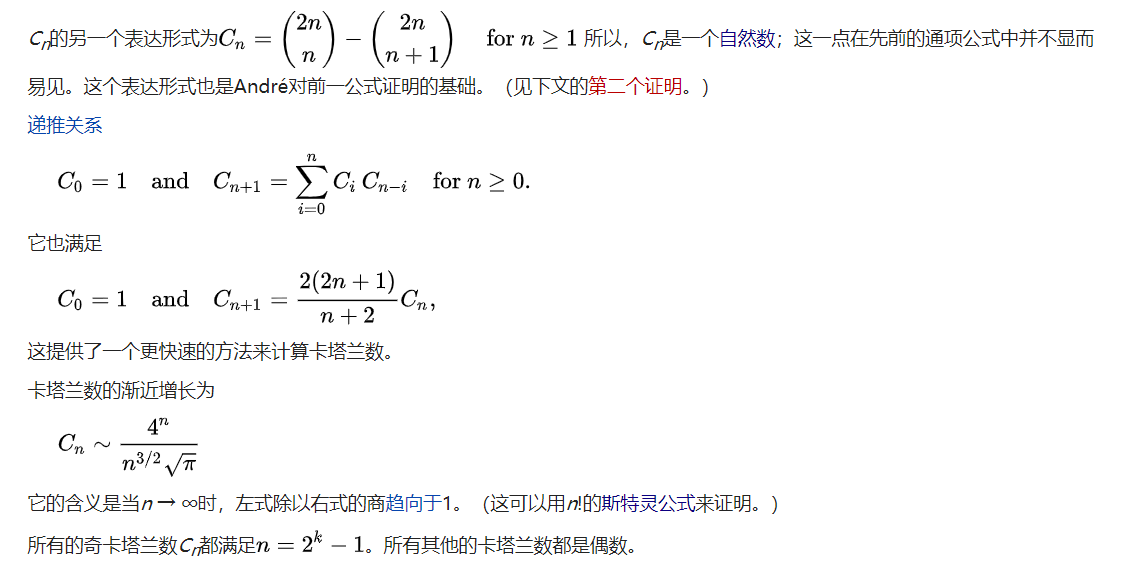
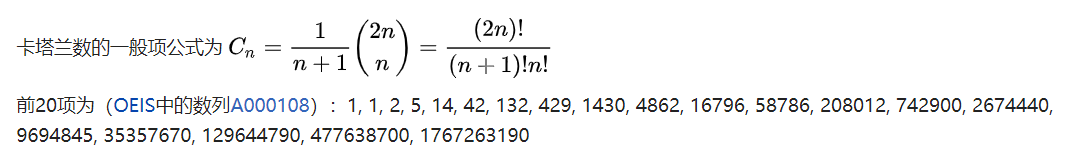
**第二种形式：**

其中是n个元素划分为k个非空集合的方案数

1. **卡塔兰数**

记第n个卡塔兰数。

注：满足长度为2\*n的序列中有n个左括号和n个右括号的合法括号序列的个数为



1. **斯特灵公式**

斯特灵公式是一条用来取n[阶乘](https://zh.wikipedia.org/wiki/%E9%9A%8E%E4%B9%98)[近似值](https://zh.wikipedia.org/wiki/%E8%BF%91%E4%BC%BC%E5%80%BC)的[数学](https://zh.wikipedia.org/wiki/%E6%95%B8%E5%AD%B8)[公式](https://zh.wikipedia.org/wiki/%E5%85%AC%E5%BC%8F)

对于足够大的整数n，这两个数互为近似值。

1. **错排公式**

表示为n个人有n个位置，满足对所有人第i个人不坐在第i个位置的方案数

1. **欧拉函数：**

设phi(x)为关于x的欧拉函数，那么phi(x)的值定义为1~x中与x互质的数的个数。

，其中p是m的素因子。

当n>1时，1…n中与n互质的整数和为

1. 生日悖论

假设一年有n天，那么k个人中至少有两个人在同一天生日的概率为

1. 二项式定理

**（1）二次项定理**

根据此定理，可以将 的{\displaystyle x+y}任意次幂展开成和的形式

也可以表示为

**（2）二次项定理的一个变型**

二项式定理的一个变形是用 1 来代换 得到的，所以它只涉及一个[变量](https://zh.wikipedia.org/wiki/%E5%8F%98%E9%87%8F)。在这种形式中，公式写作

也可以表示为

**（3）二次项展开的奇数项或偶数项的和**

设的二次项展开的奇数项和为

即 ，其中 且

设的二次项展开的偶数项和为

即 ，其中 且

所以 ①​

我们令

那么 ②

我们联立方程① ②就可以解出的值

## 部分定理

1. **威尔逊定理：**

如果p是质数，那么 (

1. **欧拉定理：**

假设Phi(m)是m的欧拉函数值。

**如果a与m互质**，那么，特殊的，如果m为质数，且a<m，则

1. **广义欧拉定理:**

如果：

注意该公式只在时成立，且不论a,m是否互质，该公式都成立。

1. **拉格朗日定理：**
2. **Lucas定理：**

**用于模数为素数且较小，但组合数太大的的定理。**

若p是质数，那么。形式上可以理解为a和b的p进制对应取模相乘。

1. ll lucas(ll n,ll m,ll p)//    C(a,b)%c的值
2. {
4. **if**(n < m) **return** 0;
5. **if**(!n) **return** 1;
6. **return** lucas(n/p,m/p,p)\*C(n%p,m%p,p)%p;
7. }

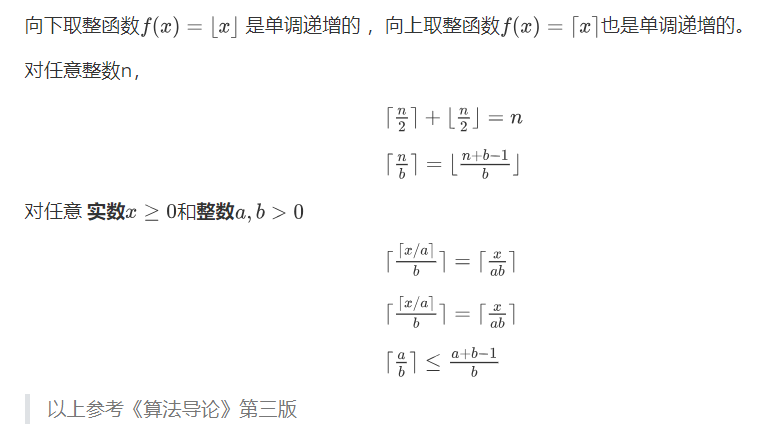
其中 是求直接求组合数的值

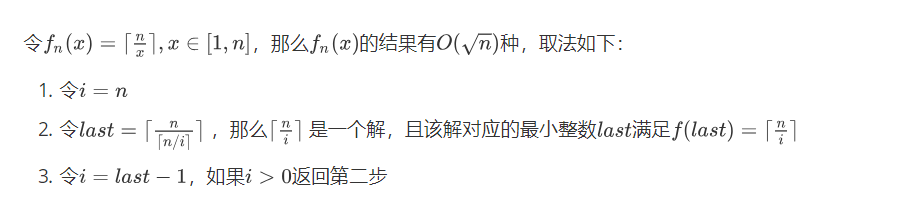
1. ll C(ll n,ll m,ll p)//要求:预处理mod p的阶乘f[]
2. {
3. **if**(n<m) **return** 0;
4. **return** f[n]\*qpow(f[m],p-2,p)%p\*qpow(f[n-m],p-2,p)%p;
5. }

## 一些常识

1. N的因子的个数为

## 关于向上取整和向下取整

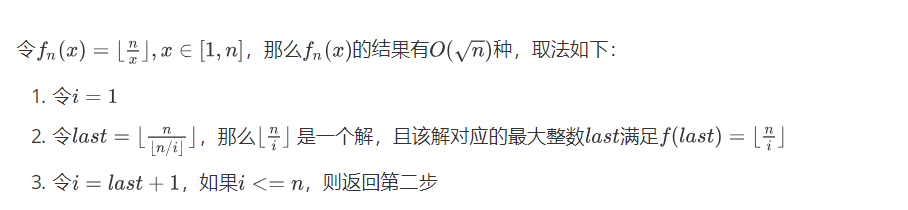




让n等于某个值，**然后列个x到f(x)的映射关系就很容易理解了。向下取整也一样**。

代码表示：

1. **int** cdiv(**int** a,**int** b)// a/b的向上取整
2. {
3. **return** (a+b-1)/b;
4. }
5. **void** work(**int** n)
6. {
7. **int** tol=0;
8. **for**(**int** i=n,last;i>0;i=last-1,tol++)
9. {
10. cout<<"f(x):"<<cdiv(n,i)<<endl;//print the solutions
11. last=cdiv(n,cdiv(n,i));
12. }
13. cout<<"The number of solutions :"<<tol<<endl;
14. }
15. **int** main()
16. {
17. **int** n;
18. cin>>n;
19. work(n);
20. }



代码表示 ：

1. **int** fdiv(**int** a,**int** b)
2. {
3. **return** a/b;
4. }
5. **void** work(**int** n)
6. {
7. **int** tol=0;
8. **for**(**int** i=1,last;i<=n;i=last+1,tol++)
9. {
10. cout<<"f(x):"<<fdiv(n,i)<<endl;//print the solutions
11. last=fdiv(n,fdiv(n,i));
12. }
13. cout<<"The number of solutions :"<<tol<<endl;
14. }
15. **int** main()
16. {
17. **int** n;
18. cin>>n;
19. work(n);
20. }

## 线性求阶乘，自然数的逆元

我们用 表示 取模 的值，表示 关于 的逆元， 代表 关于的逆元。

对于我们可以线性递推

对于我们先朴素 求出的值，又因为，所以我们也可以倒着线性递推出的值，

因为= ，那么，即求出

# 组合数学

## 组合数打表

线性求C（n，i） C（n，i+1）,线性求C（i，k）  C（i+1，k）

C（i，k）=C（i-1，k）\*i/（i-k）

// C（k，i）=C（k-1，i）\*k/（k-i）//ok的

普通打表

其实也可以预处理阶乘使用阶乘的形式:

## 容斥定理

For循环写

1. **const** **int** maxn=1e5;
2. //假设有tot个数，存储在num数组中，求他们的所有组合并放到rc[]
3. //奇数个组合值是负的
4. **int** num[20],tot;//存放数字种类,数字种类数
5. **int** rc[maxn],cnt;//存放容斥的组合,容斥组合的个数
6. **void** rc\_for()//求出num[]中 tot个数的所有组合
7. {
8. cnt=0;
9. rc[cnt++]=1;//初始状态
10. **for**(**int** i=0; i<tot; ++i)
11. {
12. **int** k=cnt;//下标为i的数与前k个数的所有组合的组合
13. **for**(**int** j=0; j<k; ++j)
14. {
15. rc[cnt++]=num[i]\*rc[j]\*-1;//奇数个为负
16. }
17. }
18. }

二进制写

1. **const** **int** maxn=1e5;
2. //假设有tot个数，存储在num数组中，求他们的所有组合并放到rc[]
3. //奇数个组合值是负的
4. **int** num[20],tot;//存放数字种类,数字种类数
5. **int** rc[maxn],cnt;//存放容斥的组合,容斥组合的个数
6. **void** rc\_bit()//用二进制来写
7. {
8. **int** top=1<<tot;//共tot个数字
9. cnt=0;
10. **for**(**int** i=0; i<top; ++i)
11. {
12. **int** val=1;
13. **for**(**int** k=0; k<tot; ++k) //枚举每个位是否为1 为1的组合整上去
14. {
15. **if**((1<<k)&i)
16. {
17. val\*=-1\*num[k];
18. }
19. }
20. rc[cnt++]=val;
21. }
22. }

递归写：

1. **const** **int** maxn=1e5;
2. //假设有tot个数，存储在num数组中，求他们的所有组合并放到rc[]
3. //奇数个组合值是负的
4. **int** num[20],tot;//存放数字种类,数字种类数
5. **int** rc[maxn],cnt;//存放容斥的组合,容斥组合的个数
6. **void** dfs(**int** k,**int** val)//正在枚举下标为k的状态, 之前组合乘积为val
7. {
8. **if**(k==tot)
9. {
10. rc[cnt++]=val;
11. **return** ;
12. }
13. dfs(k+1,val\*-1\*num[k]);//下标为k的状态在组合中
14. dfs(k+1,val);
15. }
16. **void** rc\_dfs()
17. {
18. cnt=0;
19. dfs(0,1);//
20. }
21. **int** main()
22. {
23. scanf("%d",&tot);
24. **for**(**int** i=0; i<tot; ++i)
25. scanf("%d",num+i);
26. cnt=0;
27. rc\_dfs();
28. }

## 部分枚举子集

**枚举一个二进制状态的子集（其实我更喜欢for循环枚举）**

1. **void** EnumSubSet(**int** sup)/\*作用：对于二进制状态sup,枚举该状态的子集\*/
2. {
3. **int** sub=sup;
4. **do**{
5. /\*对子集的处理\*/
6. sub=(sub-1)&sup
7. }
8. **while**(sub!=sup);/\*当sub=0之后 sub=0-1=-1,退出\*/
9. }
10. /\*
11. 原理：针对sup中的二进制为1的位开始进行减法，假设有k个二进制位,那么像枚举(2^k-1)~0一样枚举其子集
12. 输出：
13. 状态为降序输出
14. \*/

**枚举n个状态中，k个状态成立的所有状态**

1. **void** EnumK(**int** k,**int** n)/\*求出总共n个状态中，有k个状态为1的所有情况\*/
2. {
3. **int** comb=(1<<k)-1;
4. **while**(comb<1<<n)//comb>=2^n退出
5. {
6. //针对组合的处理
7. **int** x=comb&-comb,y=comb+x;
8. comb=((comb&~y)/x>>1)|y;
9. }
10. }
11. /\*原理：
12. 根据当前的符合要求的状态求出第一个大于该状态的符合要求的状态
13. 输出：
14. 升序输出
15. 算法描述：
16. 按照字典序的话，最小的子集是(1<<k)-1,所以用它作为初始值。现在哦我们求出comb其后的二进制码
17. 1.求出最低位的1开始连续的1的区间  （x&(-x)的值就是将最低位的1独立出来的值）
18. 2.将这一区间全部变为0，并将区间最左侧的0变为1
19. 3.将第1步取出的区间右移，知道剩下的1的个数少了1个
20. 4.将第2步和第3步的结果按位取或
21. \*/

## 排列组合问题总结

根本思想还是组合数学的加法原则，将一个状态分成几个不相交的状态，然后用加法原则加起来即可

求有n个球，m个盒子，在不同条件下放的方法数

1. **球同，盒不同，无空箱**

如果

否则

**分析** :

使用插板法：n个球中间有n-1个间隙，现在要分成m个盒子，而且不能有空箱子，所以只要在n-1个间隙选出m-1个间隙即可。

1. **球同，盒不同，允许空箱**

我们在第1类情况下继续讨论，我们可以先假设m个盒子里都放好了1个球，所以说白了就是，现在有m+n个相同的球，要放入m个不同的箱子，没有空箱。也就是第1种情况

1. **球不同，盒相同，无空箱**

**第二类斯特林数dp[n][m]**

dp[n][m]表示球不同，盒不同，无空箱条件下n个球 m个盒子的方法数

分两种状态

1.第n个球单独在一个盒子里 有dp[n-1][m-1]

2.第n个球不单独在一个盒子里 有m\*dp[n-1][m]

**边界条件：**

**其他情况：**

1. **球不同，盒相同，允许空箱**

枚举使用箱子的个数即可 此时的dp[n][m]是情况三的第二类的斯特林数

1. **球不同，盒不同，无空箱**

dp[n][m]是情况三的第二类的斯特林数

因为球是不同的，所以dp[n][m]得到的盒子相同的情况，只要再给盒子定义顺序，就等于现在的答案了

1. **球不同，盒不同，允许空箱**

每个球都有m种选择，所以就等于

1. **球同，盒同，允许空箱**

n个球m个盒子，分两种情况

dp[n][m]代表n个球m个盒子允许空箱的个数

两种状态

1.每个盒子至少有一个球 有dp[n-m][m]种方法

2.至少有一个盒子没有球 有dp[n][m-1]种方法

**所以：**

**当**

**否则：**

1. **球同，盒同，无空箱**

dp同第7种情况

因为要求无空箱，我们先在每个箱子里面放1个球，然后还剩下n-m个球了，再根据情况7答案就出来了

# 图论

## 最小生成树

### 克鲁斯卡尔（Kruskal)算法

1. //点标号为1~n, 边在数组edge[]中，下标从0开始，共有m个边
2. //输入:顶点个数n，m个边edge[]
3. //结果:返回 -1或者最小生成树的权值
5. **const** **int** MAXM=200000+100;
6. **const** **int** MAXN=5000+100;
7. **struct** Edge{
8. **int** u,v,w;
9. }edge[MAXM];
10. **int** father[MAXN];
11. **bool** cmp(Edge q,Edge qq)
12. {
13. **return** q.w<qq.w;
14. }
15. **int** Find(**int** x){
16. **if**(x==father[x])
17. **return** x;
18. **return** father[x]=Find(father[x]);
19. }
20. **int** Kruskal(**int** n,**int** m){
21. sort(edge,edge+m,cmp);
22. **int** sum=0,tot=0;
23. **for**(**int** i=0;i<=n;++i) father[i]=i;//初始化
24. **for**(**int** i=0;i<m;++i){
25. **int** a=edge[i].u,b=edge[i].v;
26. **int** ra=Find(a),rb=Find(b);
27. **if**(ra!=rb){//不在一个集合，合并并加上权值
28. sum+=edge[i].w;
29. tot++;
30. father[ra]=rb;
31. }
32. }
33. **if**(tot!=n-1)
34. **return** -1;
35. **return** sum;
36. }
37. **int** main(){
38. **int** n,m;
39. scanf("%d %d",&n,&m);
40. **for**(**int** i=0;i<m;++i) scanf("%d%d%d",&edge[i].u,&edge[i].v,&edge[i].w);
41. **int** ans=Kruskal(n,m);
42. **if**(ans==-1)
43. puts("orz");
44. **else**
45. printf("%d\n",ans);
46. **return** 0;
47. }

### prime算法+优先队列优化

1. //输入：图的邻接表形式adja[] ，顶点个数
2. //结果：返回-1或者最小生成树的权值
4. **const** **int** MAXM=200000+100;
5. **const** **int** MAXN=5000+100;
6. **const** **int** inf=0x3f3f3f3f;
7. **typedef** pair<**int**,**int**> P;
8. **struct** Edge{
9. **int** v,w;
10. Edge(){}
11. Edge(**int** v,**int** w):v(v),w(w){}
12. };
13. vector<Edge> adja[MAXN]; //邻接表
14. priority\_queue<P,vector<P>, greater<P> >mmp;
15. **int** dis[MAXN],book[MAXN];
16. **int** prime(**int** n)//顶带从1~n
17. {
18. **int** s=1;//如果下标从0开始，s=0
19. memset(dis,inf,**sizeof**(dis));memset(book,0,**sizeof**(book));
20. mmp.push({0,s});dis[s]=0;
21. **while**(!mmp.empty()){
22. P p=mmp.top();mmp.pop();
23. **if**(p.first>dis[p.second]) **continue**;
24. **int** u=p.second;
25. book[u]=1;//选取未加入的代价最小的顶点
26. **for**(Edge &e:adja[u])
27. {
28. **if**(book[e.v]==0&&dis[e.v]>e.w){
29. dis[e.v]=e.w;
30. mmp.push({e.w,e.v});
31. }
32. }
33. }
34. **int** sum=0;
35. **for**(**int** i=1;i<=n;++i){//统计
36. **if**(!book[i]) **return** -1;
37. sum+=dis[i];
38. }
39. **return** sum;
40. }
41. **int** main(){
42. **int** n,m;
43. scanf("%d %d",&n,&m);
44. **for**(**int** i=0;i<m;++i)
45. {
46. **int** u,v,w;
47. scanf("%d%d%d",&u,&v,&w);
48. adja[u].push\_back({v,w});
49. adja[v].push\_back({u,w});
50. }
51. **int** ans=prime(n);
52. **if**(ans==-1)
53. puts("orz");
54. **else**
55. printf("%d\n",ans);
56. **return** 0;
57. }

## 最短路

**注意：**

1. 最长路可以用spfa做，如果有正环就无解。

**技巧：**

1. DAG图求最短路可以O(m+n)，每次取出一个度数为0的点，然后更新相邻的顶点即可。
2. 如果边权都为1，则可以使用广搜bfs求最短路。
3. 遇见规模较大的的但边权只有0和1的图，可以考虑将边权为0的两个顶点缩为一个点，然后bfs求最短路即可。若边权可能还有2，则可以将2拆为两个边权为1的边，并增加一个节点。

### Dijstra+优先队列 只适用于正边权

1. /\*
2. 输入：邻接表adja[],源点s.
3. 结果：返回s到其他所有点的最短路径和
4. \*/
5. **const** **int** MAXN=110000;
6. **const** **int** inf=0x3f3f3f3f;
7. **struct** Edge
8. {
9. **int** v,w;
10. Edge() {}
11. Edge(**int** v,**int** w):v(v),w(w) {}
12. };
13. vector<Edge> adja[MAXN];//邻接表
14. **int** dis[MAXN];
15. **int** dijkstra(**int** s)//单源最短路
16. {
17. priority\_queue<P,vector<P>,greater<P> > mp;
18. **int** sum=0;
19. memset(dis,inf,**sizeof**(dis));
20. dis[s]=0;mp.push(make\_pair(0,s));
21. **while**(!mp.empty())
22. {
23. P p=mp.top();
24. mp.pop();
25. **if**(dis[p.second]<p.first)   **continue**;//每次找出距离最小的(dis,v)这个v顶点已经松弛过了，就不再需要
26. **int** u=p.second;// 只是求u的最短路，此处可以直接跳出
27. sum+=dis[u];
28. **for**(Edge &e:adja[u])
29. {
30. **if**(dis[e.v]>dis[u]+e.w)
31. {
32. dis[e.v]=dis[u]+e.w;
33. mp.push(make\_pair(dis[e.v],e.v));
34. }
35. }
36. }
37. **return** sum;
38. }

### SPFA ，边权正负均可，无负环

1. /\*
2. 输入：邻接表adja[],源点s.
3. 结果：得到s 到其他点的最短距离dis[]
4. \*/
5. **const** **int** MAXN=110000;
6. **const** **int** inf=0x3f3f3f3f;
7. **struct** Edge
8. {
9. **int** v,w;
10. Edge() {}
11. Edge(**int** v,**int** w):v(v),w(w) {}
12. };
13. vector<Edge> g[MAXN];//邻接表
14. **int** dis[MAXN],book[MAXN];
15. **void** SPFA(**int** s)
16. {
17. memset(dis,inf,**sizeof**(dis));memset(book,0,**sizeof**(book));
18. dis[s]=0;book[s]=1;
19. queue<**int**> mmp;
20. mmp.push(s);
21. **while**(!mmp.empty()){
22. **int** u=mmp.front();mmp.pop();
23. book[u]=0;
24. **for**(Edge &e:g[u]){
25. **if**(dis[e.v]>dis[u]+e.w){
26. dis[e.v]=dis[u]+e.w;
27. **if**(!book[e.v]){
28. mmp.push(e.v);
29. book[e.v]=1;
30. }
31. }
32. }
33. }
34. }

### Spfa判断有无负环，

如果一个顶点的入队数>顶点数，则存在负环。未证明

SPFA优化找负环是根据图的性质来优化的，比如对于1到n的点，每个点都有一条i到i-1费用为0的边，那么如果某个过程dis[u]<0了，那么必定有负环

求最长路则dis[other]=-inf

1. **const**  **int** inf=0x3f3f3f3f;
2. **struct** Edge{
3. **int** to,w;
4. Edge(){}
5. Edge(**int** to,**int** w):to(to),w(w){}
6. };
7. **int** dis[1005],in[1005];//距离
8. vector<Edge> g[1005];//邻接表
9. **bool** book[1005];//标记是否够在队列
10. **bool** spfa(**int** s,**int** n)//源点为s,共有n个点,求最短路
11. {//有负环返回true，无负环返回false
12. queue<**int**> qe;
13. mset(dis,inf);
14. mset(in,0);mset(book,0);
15. dis[s]=0;
16. qe.push(s);book[s]=**true**;
17. **while**(!qe.empty())
18. {
19. **int** u=qe.front();qe.pop();
20. in[u]++;book[u]=**false**;
21. **if**(in[u]>n) **return** **true**;
22. **for**(Edge &e:g[u])
23. {
24. **int** v=e.to,w=e.w;
25. **if**(dis[v]>dis[u]+w){
26. dis[v]=dis[u]+w;
27. **if**(!book[v]){
28. book[v]=**true**;
29. qe.push(v);
30. }
31. }
32. }
33. }
34. **return** **false**;
35. }

### Bellman-ford判断是否有负环,有返回true ，

1. //判断有无s可以到达的负环
2. //也可判断图中有无负环，将\*注释删去即可
3. //输入：边的的表示
4. //输出：有s可以到达的负环返回true,否则返回false
5. **const** **int** MAXN=2200;
6. **const** **int** MAXM=6100;
7. **const** **int** inf=0x3f3f3f3f;
8. **struct** Edge
9. {
10. **int** u,v,w;
11. Edge() {}
12. Edge(**int** v,**int** w):v(v),w(w) {}
13. }edge[MAXM];
14. **int** dis[MAXN],cnt;
15. **bool** Bellman(**int** s,**int** n)
16. {
17. memset(dis,inf,**sizeof**(dis));
18. dis[s]=0;
19. **int** update=1,tot=0;
20. **while**(update&&tot<n)
21. {
22. update=0;
23. tot++;
24. **for**(**int** i=0;i<cnt;++i){
25. Edge e=edge[i];
26. **if**(dis[e.u]==inf) **continue**;//\*若要判 图中有无负环 则将这个删去。
27. **if**(dis[e.v]>dis[e.u]+e.w){
28. dis[e.v]=dis[e.u]+e.w;
29. update=1;
31. }
32. }
33. }
34. **if**(update)//update为true代表在第n次循环中有需要更新的，即有负环
35. **return** **true**;
36. **return** **false**;
37. }

### Floyd+输出路径

代码暂时没有，思路用pre[i][j]表示i到j的最短路的最后一条边的前驱顶点

Eg： 1->2->3 pre[1][3]=2,pre[1][2]=1,pre[1][1]=1

### 差分约束





## 最小瓶颈路

解法1：最小瓶颈路： (二分瓶颈也行)

给定一个加权无向图，并给定无向图中两个结点u和v，求u到v的一条路径，使得路径上边的最大权值最小。

解决方案：dijstra板子套上，有点不同的是，前者每次找出未被标记的距离最短的点，并延申。后者每次找出最小瓶颈并标记，然后延申。

解法2：

二分最大权值，只能走边权小于该值的边，检查是否有S到T的路径

## 欧拉路径 见Bing神

必要条件:

1.存在一个环

2.所有点的入度等于出度

其实在这些条件也是充分的

Bfs（u）：

For e：adja[u]

If(!book[e]) dfs(e.v)

Seq[tot++]=u

## 二分图匹配

### 性质

**二分图中：**

* 二分图最大匹配 = 最小顶点覆盖。
* 最大独立集 + 最小顶点覆盖（最大匹配） = 顶点个数。
* 对于不存在孤立点的图，最大匹配+最小边覆盖=顶点个数

其他名词：

**二分图**:   
二分图又称二部图，是图论中的一种特殊模型。设G=(V,E)是一个无向图，如果顶点V可以分割为两个互不相交的子集(A,B),并且图中的每条边(i,j)所关联的两个顶点i和j分别属于这两个不同的顶点集(i in A, j in B), 则称图G是二分图。   
**匹配：**  
给定一个二分图，在G的一个子图G’中，如果G’的边集中的任意两条边都不依附于同一个顶点，则称G’的边集为G的一个匹配

**最大匹配：**   
在所有的匹配中，边数最多的那个匹配就是二分图的最大匹配了

**顶点覆盖：**   
在顶点集合中，选取一部分顶点，这些顶点能够把所有的边都覆盖了。这些点就是顶点覆盖集

**最小顶点覆盖：**   
在所有的顶点覆盖集中，顶点数最小的那个叫最小顶点集合。

**独立集：**  
在所有的顶点中选取一些顶点，这些顶点两两之间没有连线，这些点就叫独立集

**最大独立集：**  
在左右的独立集中，顶点数最多的那个集合

**边覆盖**：   
在图中找一些边，这些路径覆盖图中所有的顶点，每个顶点都只与一条边相关联。

**最小边覆盖：**在所有的路径覆盖中，边个数最少的就是最小边覆盖了。   
**二分图的存在判定：**

一个图是二分图的充分必要条件是图中不存在奇环，可以用交叉染色法判定。

### 二分图匹配匈牙利算法

任意无向图二分图匹配。

1. //
2. //输入顶点个数和边即可,不用转化为左右顶点集合的有向边
3. //时间复杂度O(N\*E)
4. **int** V;//顶点个数
5. vector<**int**> G[510];
6. **int** match[510];
7. **bool** used[510];
8. **void** add\_edge(**int** u,**int** v)
9. {
10. G[u].push\_back(v);
11. G[v].push\_back(u);
12. }
13. **bool** dfs(**int** v)
14. {
15. used[v]=**true**;
16. **for**(**int** i=0; i<G[v].size(); ++i)
17. {
18. **int** u=G[v][i],w=match[u];
19. **if**(w<0 || !used[w]&&dfs(w) )
20. {
21. match[v]=u;
22. match[u]=v;
23. **return** **true**;
24. }
25. }
26. **return** **false**;
27. }
28. **int** bipartice\_matching() //返回匹配数
29. {
30. **int** res=0;
31. memset(match,-1,**sizeof**(match));
32. **for**(**int** v=1; v<=V; ++v)//从下标1开始
33. {
34. **if**(match[v]<0)
35. {
36. memset(used,0,**sizeof**(used));
37. **if**(dfs(v))
38. {
39. res++;
40. }
41. }
42. }
43. **return** res;
44. }
45. **int** main()
46. {
48. **int** m;
49. **while**(~scanf("%d%d",&V,&m))
50. {
51. **for**(**int** i=1; i<=V; ++i) G[i].clear();
52. **for**(**int** i=1; i<=m; ++i)
53. {
54. **int** u,v;
55. scanf("%d %d",&u,&v);
56. add\_edge(u,v);
57. }
58. **int** ans=bipartice\_matching();
59. cout<<ans<<endl;
60. }
61. **return** 0;
62. }
63. **void** out\_match()//输出匹配
64. {
65. memset(used,0,**sizeof**(used));
66. **for**(**int** i=1; i<=V; ++i)
67. {
68. **if**(used[i]==0&&match[i]>=0)//这个点被匹配过且没输出过
69. {
70. printf("%d %d\n",i,match[i]);
71. used[i]=used[match[i]]=1;
72. }
73. }
74. }

### 二部图的二分图匹配

时间复杂度：O(n^2)

1. /\*
2. matchx[]表示左边顶点集合的匹配状态,matchy[]表示右边顶点集合的匹配状态
3. love[i][j]=true代表左边顶点的第i和右边顶点的第j个匹配
4. \*/
5. **int** matchx[N],matchy[N];
6. **bool** love[N][N];
7. **bool** used[N];//只会标记左边x,是否dfs过
8. **bool** dfs(**int** u,**int** m)
9. {
10. used[u]=**true**;
11. **for**(**int** v=1;v<=m;++v){
12. **if**(!love[u][v]) **continue**;
13. **int** w=matchy[v];
14. **if**(w<0||(!used[w]&&dfs(w,m))){
15. matchx[u]=v;
16. matchy[v]=u;
17. **return** **true**;
18. }
19. }
20. **return** **false**;
21. }
22. **int** bipartite\_match(**int** n,**int** m)
23. {
24. **int** res=0;
25. mset(matchx,-1);mset(matchy,-1);
26. **for**(**int** i=1;i<=n;++i){
27. **if**(matchx[i]<0){
28. mset(used,0);
29. **if**(dfs(i,m)) res++;
30. }
31. }
32. **return** res;
33. }

## 网络流

方法：每次从残留网络找出一条增广路，并且沿着这条路增广即可，直至没有增广路

思想：网络流就像水流一样，求最大的水流速率，很多问题可以转化为网络流模型来解决

1.求最大消耗量：将产生值的连接到源点，消耗值的连接到汇点.

2.顶点上有最大容量限制： 插点即可.

注：网络流图是有向图，但按照网络流意义最后的流不会同时走正反两边。(合并成流大的那个方向)

**特殊模型转换方法：**

**1. 二分图匹配转换网络流方法**： s到左顶点集合U连接容量为1的边，右顶点集合V到t连接容量为1的边。对于所有的边变成U到V的单向边（容量为1）

2 .**每个顶点都有一个失效费用，求失效部分点的最小费用使得S到T不连通。**

解决方案：对于**非源点和汇点的顶点进行插点**，**插点的边v到v’的容量为其失效费用（相当于给顶点v限制最大流量）**，**对于初始边u到v，让u‘到v连接一个inf的边，然后求最小割即可**。因为最后肯定不会割开初始边(初始边的容量为inf)，割开的边一定是插点的边，割开的这个边就相当于删去这个点。

### EK算法

1. //1.初始化 顶点个数n，下标从0开始
2. //2.添加边
3. //3.求 s到t的最大流
4. #define mset(a,b) memset(a,b,sizeof(a))
5. **using** **namespace** std;
6. **const** **int** maxn=120; //顶点数量
7. **const** **int** inf=0x3f3f3f3f;
8. **struct** edge{
9. **int** cap,rev,to;//储存 容量、邻接点、反向边
10. edge(){};
11. edge(**int** to,**int** cap,**int** rev){**this**->to=to,**this**->cap=cap,**this**->rev=rev;}
12. };
13. **class** EK{
14. **public**:
15. vector<edge> adja[maxn];
16. **int** prevv[maxn],top;
17. **int** preve[maxn];
18. **void** init(**int** n)
19. {
20. **for**(**int** i=0;i<n;++i)    adja[i].clear();
21. top=n;
22. }
23. **void** bfs(**int** s,**int** t)
24. {
25. memset(prevv,-1,**sizeof**(prevv));
26. queue<**int**> mmp;
27. mmp.push(s);
28. prevv[s]=s;
29. **while**(!mmp.empty()){
30. **int** u=mmp.front();
31. mmp.pop();
32. **for**(**int** i=0;i<adja[u].size();++i){
33. edge& e=adja[u][i];
34. **if**(prevv[e.to]==-1&&e.cap>0)
35. {
36. prevv[e.to]=u;
37. preve[e.to]=i;
38. mmp.push(e.to);
39. }
40. }
41. }
42. }
43. **void** addEdge(**int** u,**int** v,**int** f)
44. {
45. adja[u].push\_back(edge(v,f,adja[v].size()));
46. adja[v].push\_back(edge(u,0,adja[u].size()-1));
47. }
48. **int** maxFlow(**int** s,**int** t)
49. {
50. **int** flow=0;
51. **for**(;;)
52. {
53. bfs(s,t);
54. **if**(prevv[t]==-1)
55. **return** flow;
56. **else**
57. {
58. **int** minn=inf;
59. **for**(**int** last=t;prevv[last]!=last;last=prevv[last]){
60. minn=min(minn,adja[prevv[last]][preve[last]].cap);
61. }
62. flow+=minn;
63. **for**(**int** last=t;prevv[last]!=last;last=prevv[last]){
64. edge& e=adja[prevv[last]][preve[last]];
65. e.cap-=minn;
66. adja[last][e.rev].cap+=minn;
67. }
68. }
69. }
70. }
71. };
72. EK kit;
74. **int** main()
75. {
77. }

### 我bing神的模板贼快 ISAP+BSF+栈优化

1. **const** **int** MAXN=100010;//点数的最大值
2. **const** **int** MAXM=400010;//边数的最大值
3. **const** **int** INF=0x3f3f3f3f;
4. **const** **int** inf=0x3f3f3f3f;
5. **struct** Edge
6. {
7. **int** to,next,cap,flow;
8. } edge[MAXM];
9. **class** ISAP
10. {
11. //输入参数:起点、终点、点的总数
12. //点的编号没有影响，只输入点的总数
13. **public**:
14. **int** tol;
15. **int** head[MAXN];
16. **int** gap[MAXN],dep[MAXN],cur[MAXN];
17. **void** init()
18. {
19. tol=0;
20. mset(head,-1);
21. }
22. **void** addEdge(**int** u,**int** v,**int** w,**int** rw=0)
23. {
24. edge[tol].to=v,edge[tol].cap=w,edge[tol].next=head[u];
25. edge[tol].flow=0,head[u]=tol++;
26. edge[tol].to=u,edge[tol].cap=rw,edge[tol].next=head[v];
27. edge[tol].flow=0,head[v]=tol++;
28. }
29. **int** Q[MAXN];
30. **void** BFS(**int** start,**int** end)
31. {
32. mset(dep,-1),mset(gap,0);
33. gap[0]=1;
34. **int** front=0,rear=0;
35. dep[end]=0;
36. Q[rear++]=end;
37. **while**(front!=rear)
38. {
39. **int** u=Q[front++];
40. **for**(**int** i=head[u]; i!=-1; i=edge[i].next)
41. {
42. **int** v=edge[i].to;
43. **if**(dep[v]!=-1)  **continue**;
44. Q[rear++]=v;
45. dep[v]=dep[u]+1;
46. gap[dep[v]]++;
47. }
48. }
49. }
50. **int** S[MAXN];
51. **int** sap(**int** start,**int** end,**int** N)//只输入源点和汇点的编号   和顶点个数
52. {
53. BFS(start,end);
54. memcpy(cur,head,**sizeof**(head));
55. **int** top=0,u=start,ans=0;
56. **while**(dep[start]<N)
57. {
58. **if**(u==end)
59. {
60. **int** Min=INF;
61. **int** inser;
62. **for**(**int** i=0;i<top;i++)
63. {
64. **if**(Min>edge[S[i]].cap-edge[S[i]].flow){
65. Min=edge[S[i]].cap-edge[S[i]].flow;
66. inser=i;
67. }
68. }
69. **for**(**int** i=0;i<top;++i){
70. edge[S[i]].flow+=Min;
71. edge[S[i]^1].flow-=Min;
72. }
73. ans+=Min;
74. top=inser;
75. u=edge[S[top]^1].to;
76. **continue**;
77. }
78. **bool** flag=**false**;
79. **int** v;
80. **for**(**int** i=cur[u]; i!=-1; i=edge[i].next)
81. {
82. v= edge[i].to;
83. **if**(edge[i].cap-edge[i].flow&&dep[v]+1==dep[u])
84. {
85. flag=**true**;
86. cur[u]=i;
87. **break**;
88. }
89. }
90. **if**(flag)
91. {
92. S[top++]=cur[u];
93. u=v;
94. **continue**;
95. }
96. **int** Min= N;
97. **for**(**int** i=head[u]; i!=-1; i=edge[i].next)
98. {
99. **if**(edge[i].cap-edge[i].flow&&dep[edge[i].to]<Min)
100. {
101. Min=dep[edge[i].to];
102. cur[u]=i;
103. }
104. }
105. gap[dep[u]]--;
106. **if**(!gap[dep[u]])    **return** ans;
107. dep[u]=Min+1;
108. gap[dep[u]]++;
109. **if**(u!=start) u=edge[S[--top]^1].to;
110. }
111. **return** ans;
112. }
113. };

## 最小割

**什么是最小割？**

​ 割掉网络图的一些边，使得之后的顶点分为两部分，即S可以到达的顶点集合和可以到达T的顶点集合，每个边割开都有会产生一个费用；而最小割就是我们使顶点分为两部分的费用和最小的割法。

结论：**且我们已经证明最小割等于该网络流图中的最大流。**

**最小割多用于什么时候？**

所有将顶点划分为两个集合且使得费用最小的问题都可以转化为最小割。

**那么如何借此最小割呢?**

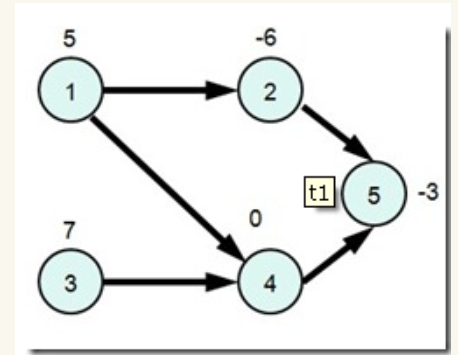
​ 我们可以先求出最大流,以S为起点在残留网络上DFS,搜索到的点就是包含S的一部分且我们将之染色,剩余的则是另一部分,而最小割即是连接两部分的所有边.如果一个边的两个顶点的颜色不一样则该边是割边。

## 最大权闭合子图

**什么是最大权闭合图？**

考虑一个有向图G，其中顶点集合为V，且每个顶点i都有一个权值（正负均有可能），现在要求从中选出某一顶点子集，且满足若顶点u在该集合内，所有关于顶点u所有的出边所指的顶点v都在集合 中（称为原图的闭合子图），要求选出的顶点集合 权值和最大。

例如该图：



上图中闭合图有

     {5}、{2,5}、{4,5}

     {2,4,5}、{3,4,5}

     {1,2,3,4,5}、{1,2,4,5}

最大权闭合图为{3,4,5}。

**怎么求最大权闭合图？**

我们现在构建一个网络图，假设顶点i的权值为，我们将源点S连向的顶点，容量为；将 的顶点i连向汇点T，容量为；对于原图G中的有向边，我们将顶点u连向顶点v，容量为inf。然后我们将顶点权值为正的权值和减去S到T的最小割就是答案。

注：，即所有顶点权值为正的权值和。

**为什么这样连接呢？**

我们可以这样理解，**原图中的顶点最后会分为两个顶点集合A，B，其中A是被选的点，B是没有选的点，**那么这自然而然的想到最小割模型了，既然最小，那么我们最小的是什么？。

我们每选一个 的点i，我们就有的损失，所以我们可以让损失最小。那么对于 的点呢？总不能有负损失吧？

我们可以假设对于的点i，我们刚开始时已经选了，那么不选就有了的损失。

对于原图中的边，这些边是没有损失的，只是一个限制，所以我们可以让u到v的容量设为inf，这样最小割就不会包含这条边，也保留了u选则v必须选的限制。

**所以我们让源点S连向 的点i，容量为；让的点i 连向汇点T，容量为；对于原图的边，让u连向v容量为inf。最终 (S到T的最小割)就是答案。**

可以很容易知道这是一个简单割（即割边只会是与S，或T相连的边），**最后所有S可以到达的顶点都是属于集合A，其他顶点属于集合B**。

例题：Wifi-Towers（2009 Google Jam World Final D）

题意：

给定一个无线电塔的网络。对于每座无线电塔，都有一个半径参数，这座无线电塔可以给这个半径范围内的其他无线电塔发送信号，刚开始时无线电塔之间都使用的是古老的协议A进行通信，现在要将某些电塔升级到协议B，升级有个要求：如果电塔a升级到协议B，那么电塔a范围内所有电塔都必须升级到协议B（如果仅仅是某个协议B的电塔a被电塔b覆盖，那么电塔b不用必须升级到协议B）

​ 考虑到升级后的花费和升级后的收益，现我们给每座无线电塔打一个分数（正分表示升级之后收益更多，负分表示升级的花费更多），现要选择一些合适的无线电塔进行升级，使得升级的无线电塔的总分最大。

假设无线电塔的个数为，无线电塔的位置为，覆盖半径为，分数是.

数据范围：

代码：

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **const** **int** maxn=550;
6. **const** **int** inf=0x3f3f3f3f;
7. **struct** edge{
8. **int** cap,rev,to;
9. edge(){};
10. edge(**int** to,**int** cap,**int** rev){**this**->to=to,**this**->cap=cap,**this**->rev=rev;}
11. };
12. **class** EK{
13. **public**:
14. vector<edge> adja[maxn];
15. **int** prevv[maxn],top;
16. **int** preve[maxn];
17. **void** init(**int** n)
18. {
19. **for**(**int** i=0;i<n;++i)    adja[i].clear();
20. top=n;
21. }
22. **void** bfs(**int** s,**int** t)
23. {
24. memset(prevv,-1,**sizeof**(prevv));
25. queue<**int**> mmp;
26. mmp.push(s);
27. prevv[s]=s;
28. **while**(!mmp.empty()){
29. **int** u=mmp.front();
30. mmp.pop();
31. **for**(**int** i=0;i<adja[u].size();++i){
32. edge& e=adja[u][i];
33. **if**(prevv[e.to]==-1&&e.cap>0)
34. {
35. prevv[e.to]=u;
36. preve[e.to]=i;
37. mmp.push(e.to);
38. }
39. }
40. }
41. }
42. **void** addEdge(**int** u,**int** v,**int** f)
43. {
44. adja[u].push\_back(edge(v,f,adja[v].size()));
45. adja[v].push\_back(edge(u,0,adja[u].size()-1));
46. }
47. **int** maxFlow(**int** s,**int** t)
48. {
49. **int** flow=0;
50. **for**(;;)
51. {
52. bfs(s,t);
53. **if**(prevv[t]==-1)
54. **return** flow;
55. **else**
56. {
57. **int** minn=inf;
58. **for**(**int** last=t;prevv[last]!=last;last=prevv[last]){
59. minn=min(minn,adja[prevv[last]][preve[last]].cap);
60. }
61. flow+=minn;
62. **for**(**int** last=t;prevv[last]!=last;last=prevv[last]){
63. edge& e=adja[prevv[last]][preve[last]];
64. e.cap-=minn;
65. adja[last][e.rev].cap+=minn;
66. }
67. }
68. }
69. }
70. }ooo;
71. **const** **int** N=505;
72. **int** x[N],y[N],r[N],s[N];
73. **int** solve(**int** n)
74. {
75. **int** S=0,T=n+1,sum=0;
76. ooo.init(n+2);
77. **for**(**int** i=1;i<=n;++i)
78. {
79. **if**(s[i]>0){
80. sum+=s[i];
81. ooo.addEdge(S,i,s[i]);
82. }
83. **else**{
84. ooo.addEdge(i,T,-s[i]);
85. }
86. }
87. **for**(**int** i=1;i<=n;++i)
88. {
89. **for**(**int** j=1;j<=n;++j){
90. **if**(i==j) **continue**;
91. **int** m=(x[j]-x[i])\*(x[j]-x[i])+(y[j]-y[i])\*(y[j]-y[i]);
92. **if**(m<=r[i]\*r[i]){
93. ooo.addEdge(i,j,inf);
94. }
95. }
96. }
97. **int** flow=ooo.maxFlow(S,T);
98. **return** sum-flow;
99. }
100. **int** main()
101. {
102. //    freopen("1.in","r",stdin);
103. //    freopen("1.out","w",stdout);
104. **int** t,cas=0;
105. scanf("%d",&t);
106. **while**(t--)
107. {
108. **int** n;
109. scanf("%d",&n);
110. **for**(**int** i=1;i<=n;++i) scanf("%d%d%d%d",x+i,y+i,r+i,s+i);
111. printf("Case #%d: %d\n",++cas,solve(n));
112. }
113. **return** 0;
114. }

## 最小费用流

**特殊模型转换方法：**

**1.二分图最大权匹配转化方法**：

**解决方案：** s到左顶点集合U连接容量为1的边，费用为0；右顶点集合V到t连接容量为1的边费用为0；。对于所有的边变成U到V的单向边（容量为1），费用为-1\*权；

**2.s到t的的k条最短路，每个边最多只能使用一次，要求总距离最小**：

**解决方案：**对于每个边u,v,w，让其连接一条容量为1，费用为w的边；然后求s到t的满足流量为k的最小费用即可。若不能满足则无解。

**3.指派问题**：求n个区间，从中选取一些区间，使得每个点最多被覆盖k次，使得权值和最大。等效问题：选出一些区间，使得区间分成k 个区间集合。每个集合里面区间不相交，要求总权值和最大。

**解决方案**：网络流将所有端点排序，相邻节点连接一条费用为0，容量为inf的边，对于每个区间i ，a 到b 端点连接一条费用为−w[i] 的边。容量为1 ，让S连接1，容量为k，费用为0，n连接T，容量为k，费用为0。那么求S到T的最小费用流，每一个流都代表一个不重叠的区间选取。其权值和等于费用的相反数。

4.**判断某个流是否等流情况下的是最小费用流:**

解决方案： 流f是最小费用流 **等效** 流f的在G中的残留网络没有负环。假设流f’是同流的最小费用流，因为f和f’除端点外每个顶点的流入量等于流出量，那么f’-f的流每个顶点的流入量等于流出量，即他是由若干个圈组成的且路径和为负的。即流f的残留网络中存在至少一个负环。

**5.**注：如果费用是浮点数，那么在SPFA更新最短路的时候考虑下eps，这将会是个大优化。

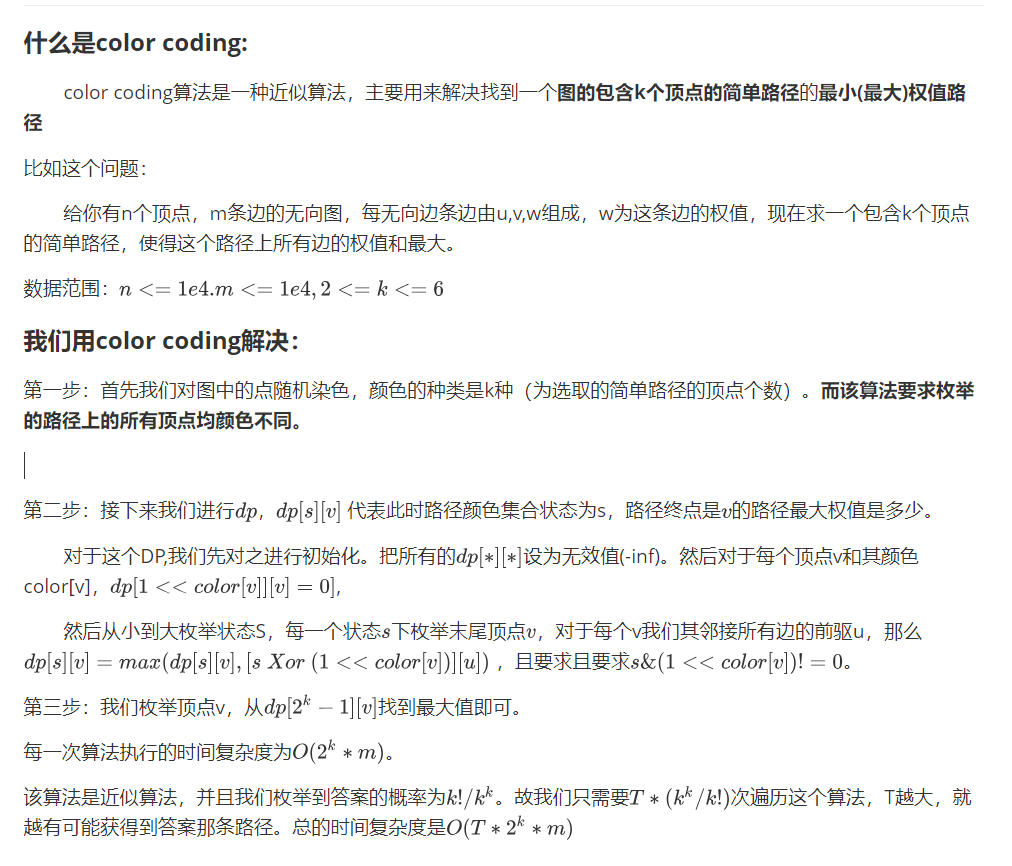
### SPFA寻找最短增广路，对于有负环无法操作（可以找到负环，然后将负环消去）。

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **const** **int** maxn=220;//图中最大顶点数量
5. **const** **int** inf=0x3f3f3f3f;
6. **struct** edge{
7. **int** to,cap,cost,rev;
8. edge(){}
9. edge(**int** to,**int** cap,**int** cost,**int** rev){**this**->to=to,**this**->cap=cap,**this**->cost=cost,**this**->rev=rev;}
10. };
11. **class** MCMF{
12. **public**:
13. vector<edge> adja[maxn];
14. **int** dis[maxn],prevv[maxn],preve[maxn],top;
15. **bool** inque[maxn];
16. **void** init(**int** n)
17. {
18. **for**(**int** i=0;i<n;++i)    adja[i].clear();
19. top=n;
20. }
21. **void** addEdge(**int** u,**int** v,**int** f,**int** cost){
22. adja[u].push\_back(edge(v,f,cost,adja[v].size()));
23. adja[v].push\_back(edge(u,0,-1\*cost,adja[u].size()-1));
24. }
25. **bool** spfa(**int** s,**int** t){
26. queue<**int**> mp;
27. mset(dis,inf);
28. mset(prevv,-1);
29. mset(inque,0);
30. mp.push(s),prevv[s]=s,dis[s]=0,inque[s]=**true**;
31. **while**(!mp.empty()){
32. **int** u=mp.front();
33. mp.pop();
34. inque[u]=**false**;
35. **for**(**int** i=0;i<adja[u].size();++i){
36. edge& e=adja[u][i];
37. **if**(e.cap>0&&dis[e.to]>dis[u]+e.cost){
38. dis[e.to]=dis[u]+e.cost;
39. prevv[e.to]=u;
40. preve[e.to]=i;
41. **if**(!inque[e.to]){
42. inque[e.to]=**true**;
43. mp.push(e.to);
44. }
45. }
46. }
47. }
48. **if**(~prevv[t]) **return** **true**;
49. **return** **false**;
50. }
51. **int** minCostMaxFlow(**int** s,**int** t,**int** &cost){//返回s到t的最大流， 并把花费存到cost里面
52. cost=0;
53. **int** flow=0;
54. **while**(spfa(s,t))
55. {
56. **int** minn=inf;
57. **for**(**int** v=t;v!=prevv[v];v=prevv[v]){
58. minn=min(adja[prevv[v]][preve[v]].cap,minn);
59. }
60. flow+=minn;
61. /\*找出最小minn，进行增广 即flow+= cost+=  残余网络的更新\*/
62. **for**(**int** v=t;v!=prevv[v];v=prevv[v]){
63. edge& e=adja[prevv[v]][preve[v]];
64. cost+=minn\*e.cost;
65. e.cap-=minn;
66. adja[v][e.rev].cap+=minn;
67. }
68. }
69. **return** flow;
70. }
71. };
72. //他的第三个函数也可以这样写，即满足流f的最小花费
73. **int** minCostMaxFlow(**int** s,**int** t,**int** f) //不能满足流f则返回-1
74. {
75. **int** cost=0;
76. **while**(f>0)
77. {
78. spfa(s,t);
79. **if**(dis[t]==inf)
80. **return** -1;
81. **int** d=f;
82. **for**(**int** v=t; v!=prevv[v]; v=prevv[v]) //找到d
83. d=min(d,adja[prevv[v]][preve[v]].cap);
84. cost+=d\*dis[t];
85. f-=d;
86. **for**(**int** v=t; v!=prevv[v]; v=prevv[v])
87. {
88. edge &e=adja[prevv[v]][preve[v]];
89. e.cap-=d;
90. adja[v][e.rev].cap+=d;
91. }
92. }
93. **return** cost;
94. }

### 采用dijstra的最短增广路算法：（有时候性能不如SPFA的，可能是因为容器的问题）

1. **typedef** pair<**int**, **int**>P;//first保存最短距离，second保存顶点的编号
2. **const** **int** maxn=250; //顶点最大个数
3. **const** **int** INF=0x3f3f3f3f;
4. **struct** Edge
5. {
6. **int** to, cap, cost, rev;//终点，容量（指残量网络中的），费用，反向边编号
7. Edge(**int** t, **int** c, **int** cc, **int** r) :to(t), cap(c), cost(cc), rev(r) {}
8. };
9. **class** MCMF
10. {
11. **public**:
12. **int** V;//顶点数
13. vector<Edge>G[maxn];//图的邻接表
14. **int** h[maxn];//上一次顶点的最短路s到v的最短路径
15. **int** dist[maxn];//最短距离
16. **int** prevv[maxn];//最短路中的父结点
17. **int** preve[maxn];//最短路中的父边
18. **void** init(**int** n)
19. {
20. **for**(**int** i=0; i<n; ++i)    G[i].clear();
21. V = n;
22. }
23. **void** addEdge(**int** from, **int** to, **int** cap, **int** cost)
24. {
25. G[from].push\_back(Edge( to, cap, cost, G[to].size()));
26. G[to].push\_back(Edge( from, 0, -cost, G[from].size() - 1 ));
27. }
28. **int** min\_cost\_flow(**int** s, **int** t, **int** f)//返满足流f的最小费用  不能满足返回-1
29. {
30. **int** res = 0;
31. fill(h, h + V, 0);
32. **while** (f>0)//f>0时还需要继续增广
33. {
34. priority\_queue<P, vector<P>, greater<P> >q;
35. fill(dist, dist + V, INF);//距离初始化为INF
36. dist[s] = 0;
37. q.push(P(0, s));
38. **while** (!q.empty())
39. {
40. P p = q.top();
41. q.pop();
42. **int** v = p.second;
43. **if** (dist[v]<p.first)    **continue**;//p.first是v入队列时候的值，dist[v]是目前的值，如果目前的更优，扔掉旧值
44. **for** (**int** i = 0; i<G[v].size(); i++)
45. {
46. Edge&e = G[v][i];
47. **if** (e.cap>0 && dist[e.to]>dist[v] + e.cost + h[v] - h[e.to])//松弛操作
48. {
49. dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
50. prevv[e.to] = v;//更新父结点
51. preve[e.to] = i;//更新父边编号
52. q.push(P(dist[e.to], e.to));
53. }
54. }
55. }
56. **if** (dist[t] == INF)//如果dist[t]还是初始时候的INF，那么说明s-t不连通，不能再增广了
57. **return** -1;
58. **for** (**int** j = 0; j<V; j++)//更新h
59. h[j] = dist[j];
60. **int** d = f;
61. **int** sum=0;
62. **for** (**int** x = t; x != s; x = prevv[x]){
63. d = min(d, G[prevv[x]][preve[x]].cap);//从t出发沿着最短路返回s找可改进量
64. sum+=G[prevv[x]][preve[x]].cost;
65. }
66. f -= d;
67. res += d\*sum;//h[t]表示最短距离的同时，也代表了这条最短路上的费用之和，乘以流量d即可得到本次增广所需的费用
68. **for** (**int** x = t; x != s; x = prevv[x])
69. {
70. Edge&e = G[prevv[x]][preve[x]];
71. e.cap -= d;//修改残量值
72. G[x][e.rev].cap += d;
73. }
74. }
75. **return** res;
76. }
77. };

## Color coding K-th alogoritm

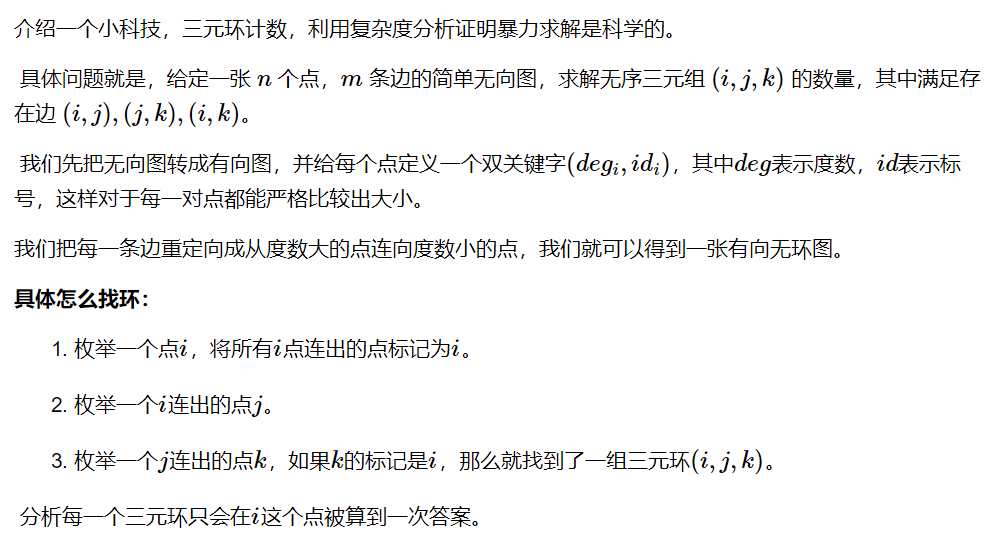


例题：[hdu 6664](http://acm.hdu.edu.cn/showproblem.php?pid=6664)（题意思与上图一致

代码：

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **typedef** unsigned **long** **long** u64;
6. **typedef** pair<**int**,**int**> P;
7. **const** **int** inf=0x3f3f3f3f;
8. **const** **int** N=1e4+10;
9. **int** n,m,k;
10. vector<P> g[N];
11. **int** color[N],dp[80][N];//s,v
12. **void** init(**int** n)
13. {
14. **for**(**int** i=1; i<=n; ++i) color[i]=rand()%k;
15. }
16. **int** solve(**int** n,**int** m,**int** k)
17. {
18. init(n);
20. **int** top=1<<k;
21. **for**(**int** i=1; i<=n; ++i)
22. {
23. **for**(**int** s=0; s<top; ++s) dp[s][i]=-inf; // 无效值
24. dp[1<<color[i]][i]=0;
25. }
26. **for**(**int** s=0; s<top; ++s)
27. {
28. **for**(**int** u=1; u<=n; ++u) //枚举顶点
29. {
30. **if**(!(s&(1<<color[u]))) **continue**;//不符合
31. //                cout<<"asdasd"<<endl;
32. **for**(P &p:g[u])
33. {
34. **int** v=p.first,w=p.second;
35. dp[s][u]=max(dp[s][u],dp[s^(1<<color[u])][v]+w);
36. }
37. }
38. }
39. **int** ans=-inf;
40. **for**(**int** i=1; i<=n; ++i) ans=max(ans,dp[top-1][i]);
41. **return** ans;
42. }
43. **int** main()
44. {
45. ios::sync\_with\_stdio(**false**);
46. cin.tie(0);
47. **int** t;
48. srand(time(NULL));
49. cin>>t;
50. **while**(t--)
51. {
52. cin>>n>>m>>k;
53. **for**(**int** i=1; i<=n; ++i) g[i].clear();
54. **for**(**int** i=0; i<m; ++i)
55. {
56. **int** u,v,w;
57. cin>>u>>v>>w;
58. g[u].push\_back({v,w});
59. g[v].push\_back({u,w});
60. }
61. **int** p=300;//随机300次
62. **int** ans=-inf;
63. **while**(p--)
64. ans=max(ans,solve(n,m,k));
65. //        cout<<"ans:"<<ans<<endl;
66. **if**(ans<0)
67. cout<<"impossible"<<endl;
68. **else**
69. cout<<ans<<endl;
70. }
71. **return** 0;
72. }

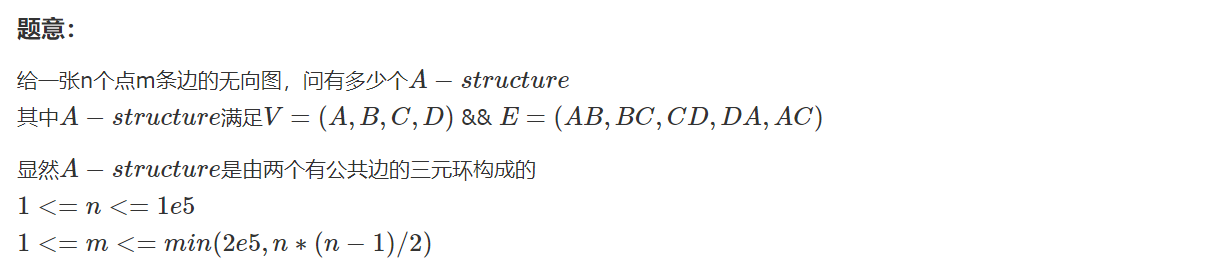
## 无向图三元环计数

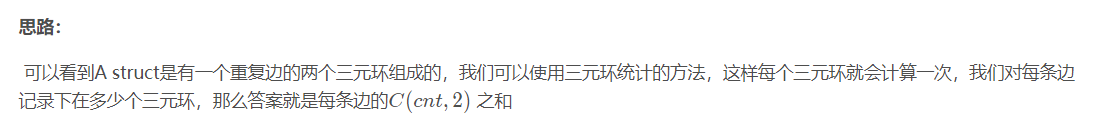


**三元环计数代码：**

1. #include <cstdio>
2. #include <vector>
3. #include <algorithm>
5. **using** **namespace** std;
7. **const** **int** N = 250005;
9. **int** n, m, ans;
10. **int** du[N], vi[N], eu[N], ev[N];
11. vector<**int**> g[N];
13. **inline** **bool** cmp(**int** x, **int** y) {
14. **return** du[x] != du[y]? du[x] > du[y] : x < y;
15. }
17. **int** main() {
18. scanf("%d%d", &n, &m);
19. **for** (**int** i = 1; i <= m; ++i)
20. scanf("%d%d", &eu[i], &ev[i]), ++du[eu[i]], ++du[ev[i]];
21. **for** (**int** i = 1; i <= m; ++i)
22. **if** (cmp(eu[i], ev[i])) g[eu[i]].push\_back(ev[i]);
23. **else** g[ev[i]].push\_back(eu[i]);
24. **for** (**int** i = 1; i <= n; ++i) {
25. **for** (**int** j : g[i]) vi[j] = i;
26. **for** (**int** j : g[i])
27. **for** (**int** k : g[j])
28. **if** (vi[k] == i) ++ans;
29. }
30. printf("%d\n", ans);
31. **return** 0;
32. }

例题：hdu 6184





1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **typedef** pair<**int**,**int**> P;
6. **typedef** P Edge;
7. Edge edge[200005];
8. **int** degree[100005];
9. **bool** cmp(**int** x,**int** y)
10. {
11. **return** degree[x]!=degree[y]?degree[x]<degree[y]:x<y;
12. }
13. vector<**int**> adja[100005];
15. **int** color[100005];
16. map<P,**int**> cunt;//记录边的出现次数
17. **int** main()
18. {
19. ios::sync\_with\_stdio(**false**);
20. cin.tie(0);
21. **int** n,m;
22. **while**(cin>>n>>m)
23. {
24. **for**(**int** i=1; i<=n; ++i)
25. {
26. adja[i].clear();
27. degree[i]=0;
28. color[i]=0;
29. }
30. cunt.clear();
31. **for**(**int** i=1; i<=m; ++i)
32. {
33. **int** u,v;
34. cin>>u>>v;
35. edge[i].first=u;
36. edge[i].second=v;
37. ++degree[u];
38. ++degree[v];
39. }
40. **for**(**int** i=1; i<=m; ++i)//把无向边变为有向边
41. {
42. **int** u=edge[i].first,v=edge[i].second;
43. **if**(cmp(u,v))
44. adja[u].push\_back(v);
45. **else**
46. adja[v].push\_back(u);
47. }
48. **for**(**int** u=1; u<=n; ++u)
49. {
50. **for**(**int** v:adja[u])
51. color[v]=u;
52. **for**(**int** v:adja[u])//枚举第1条边
53. {
54. **for**(**int** vv:adja[v]){//枚举第2条边
55. **if**(color[vv]==u){//是个三元环
56. cunt[{u,v}]++;
57. cunt[{v,vv}]++;
58. cunt[{u,vv}]++;
59. }
60. }
61. }
62. }
63. ll ans=0;
64. **for**(auto p:cunt)
65. {
66. ll cnt=p.second;
67. **if**(cnt>=2){
68. ans+=(cnt\*(cnt-1))/2;
69. }
70. }
71. cout<<ans<<endl;
72. }
73. **return** 0;
74. }

## 普吕弗序列（Prufer）

**什么是Prufer序列？**

Prufer序列是将一个带有节点编号的无根树转化为一个序列的过程，且每一个无根树唯一的确定一个prufer序列。反过来也成立。

**Prufer序列的性质**

* 原树中顶点 的度数 序列中顶点v的出现次数
* n个顶点的无根树的Prufer序列的长度为n-2
* 一个Prufer序列与一个无根树一一对应

**将无根树转化为Prufer序列**

**方法：**

一棵树要得到普吕弗序列，方法是逐次去掉树的顶点，直到剩下两个顶点。考虑树*T*，其顶点为{1, 2, ..., *n*}。在第*i*步，去掉标号最小的叶，并把普吕弗序列的第*i*项设为这叶的邻顶点的标号。

一棵树的序列明显是唯一的，而且长为*n* − 2。

**算法：**

**使用的数据结构**：维护一个度数为1的set集合即可.

1. 初始度数为1的集合（所有叶子）.
2. 每次从度数为1的叶子顶点集合中找到编号最小的顶点u，并将其从度数为1的集合中删去。将u连接的顶点v加入prufer序列，删除u连接的所有边。
3. 更新u和v的度数，v的度数为1则加入度数为1的集合。
4. 当加入序列的数的个数为n-2时候停止。否则返回到第二步。

**具体实现；**

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **const** **int** N=1e5+10;
5. vector<**int**> prufer;
6. vector<**int**> g[N];
7. **int** du[N];
8. /\*
9. 输入:顶点个数n和树g[]
10. 输出:prufer序列
11. \*/
12. **void** TreeToPrufer(**int** n,vector<**int**> g[])//g[]中存储的是无向边,共有n个顶点
13. {
14. set<**int**> leaf;
15. mset(du,0);
16. prufer.clear();
17. **for**(**int** u=1;u<=n;++u)
18. **for**(**int** v:g[u]) du[v]++;
19. **for**(**int** i=1;i<=n;++i) **if**(du[i]==1) leaf.insert(i);
20. **while**(prufer.size()!=n-2)
21. {
22. **int** u=\*leaf.begin();
23. du[u]=0;
24. leaf.erase(leaf.begin());
25. **for**(**int** v:g[u])
26. {
27. **if**(!du[v]) **continue**;
28. prufer.push\_back(v);
29. du[v]--;
30. **if**(du[v]==1) leaf.insert(v);
31. }
32. }
33. printf("following is prufer sequence of tree:\n");
34. **for**(**int** v:prufer) cout<<v<<" ";
35. cout<<endl;
36. }
37. **int** main()
38. {
39. **int** n;
40. cin>>n;//输入顶点个数
41. **for**(**int** i=0;i<n-1;++i){//输入n-1条边
42. **int** u,v;
43. cin>>u>>v;
44. g[u].push\_back(v);
45. g[v].push\_back(u);
46. }
47. TreeToPrufer(n,g);
48. **return** 0;
49. }

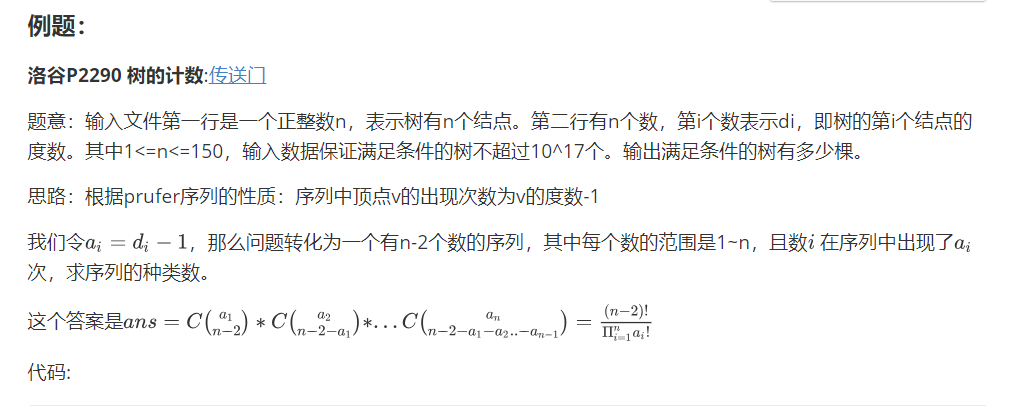
**将一个Prufer序列转化为无根树**

**方法：**设这普吕弗序列长*n* − 2。首先写出数1至*n*。第一步，找出1至*n*中没有在序列中出现的最小数。把标号为这数的顶点和标号为序列首项的顶点连起来，并把这数从1至*n*中删去，序列的首项也删去。接着每一步以1至*n*中剩下的数和余下序列重复以上步骤。最后当序列用完，把1至*n*中最后剩下的两数的顶点连起来。

**算法：**

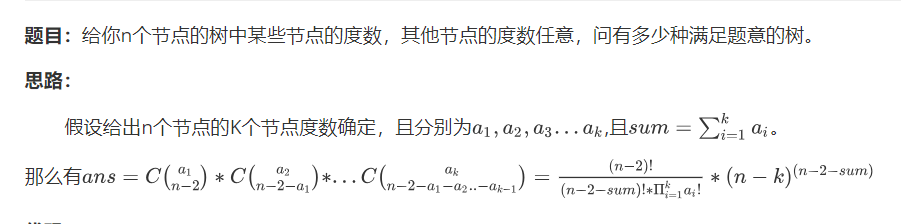
**使用的数据结构**：假设数列A初始化为1~n的n个数。使用两个次数数组，分别记录prufer序列每个数的出现次数和数列A中每个数的出现次数。维护一个set集合：为A中有但prufe序列中没有顶点集合。

1. 初始化次数数组和set集合
2. 每次取出prufer序列首部u和set集合中一个最小编号顶点v。那么u,v为原图的一个无向边。然后将v从集合中删去，并判断u是否可以加入集合。
3. 直到prufer序列为空，则取set集合内的两个顶点连成一条边。否则返回第二步。
4. #include<bits/stdc++.h>
5. #define mset(a,b) memset(a,b,sizeof(a))
6. **using** **namespace** std;
7. **typedef** pair<**int**,**int**> P;
8. **const** **int** N=1e5+10;
9. vector<P> tree;
10. **int** pvtimes[N],avtimes[N];
11. vector<**int**> prufer;
12. /\*
13. 输入:prufer序列
14. 输出:prufer序列对应的tree的n-1条边
15. \*/
16. **void** PruferToTree(vector<**int**> &prufer)
17. {
18. **int** n=prufer.size()+2;
19. set<**int**> rem;
20. tree.clear();
21. mset(pvtimes,0);
22. **for**(**int** v:prufer) pvtimes[v]++;
24. **for**(**int** i=1;i<=n;++i) {
25. avtimes[i]=1;
26. **if**(!pvtimes[i]) rem.insert(i);
27. }
28. **int** p=0;//prufer序列首指针
29. **while**(p<prufer.size())
30. {
31. **int** u=prufer[p],v=\*rem.begin();
32. tree.push\_back({u,v});
33. avtimes[v]=0;
34. pvtimes[u]--;
35. /\*维护rem集合\*/
36. rem.erase(rem.begin());
37. **if**(pvtimes[u]==0&&avtimes[u]>0)
38. rem.insert(u);
39. ++p;
40. }
41. auto it=rem.begin();
42. **int** u=\*it++;
43. **int** v=(\*it);
44. tree.push\_back({u,v});
45. printf("following is tree edge  of prufer:\n");
46. **for**(P& p:tree)
47. cout<<p.first<<" "<<p.second<<endl;
48. }
49. **int** main()
50. {
51. **int** n;
52. cin>>n;
53. prufer.resize(n);
54. **for**(**int** i=0;i<n;++i)//要求这n个顶点的范围都属于1~n+2
55. {
56. cin>>prufer[i];
57. }
58. PruferToTree(prufer);
59. }



1. n=int(input().strip())
2. du=list(map(int,input().strip().split()))
3. **if** n==1:#n=1可以存在度数为0的点
4. **if** du[0]==0:
5. **print**(1)
6. **else**:
7. **print**(0)
8. exit()
9. sum=0
10. **for** i **in** du:
11. **if** i==0:#特判度数为0,防止RE
12. **print**(0)
13. exit(0)
14. sum+=i-1
15. **if** sum!=n-2:#提前判断是否不符合
16. **print**(0)
17. exit()
18. fac=[0 **for** i **in** range(n+1)]
19. fac[0]=1
20. **for** i **in** range(1,n+1):
21. fac[i]=fac[i-1]\*i
22. ans=fac[n-2]
23. **for** i **in** du:
24. ans//=fac[i-1]
25. **print**(int(ans))

**例题2：**



## 图的割点、桥和双联通分支的概念

**[点连通度与边连通度]**

在一个无向连通图中，如果有一个顶点集合，删除这个顶点集合，以及这个集合中所有顶点相关联的边以 后，原图变成多个连通块，就称这个点集为割点集合。一个图的点连通度的定义为，最小割点集合中的顶 点数。 类似的，如果有一个边集合，删除这个边集合以后，原图变成多个连通块，就称这个点集为割边集合。一 个图的边连通度的定义为，最小割边集合中的边数。

**[双连通图、割点与桥]**

如果一个无向连通图的点连通度大于 1，则称该图是点双连通的(point biconnected)，简称双连通或重连通。 一个图有割点，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为割点(cut point)，又叫关节 点(articulation point)。 如果一个无向连通图的边连通度大于 1，则称该图是边双连通的(edge biconnected)，简称双连通或重连通。 一个图有桥，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为桥(bridge)，又叫关节边 (articulation edge)。 可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通， 均既可指点双连通，又可指边双连通。

**[双连通分支]**

在图 G 的所有子图 G'中，如果 G'是双连通的，则称 G'为双连通子图。如果一个双连通子图 G'它不是任何一 个双连通子图的真子集，则 G'为极大双连通子图。双连通分支(biconnected component)，或重连通分支， 就是图的极大双连通子图。特殊的，点双连通分支又叫做块。

**[求割点与桥]**

该算法是 R.Tarjan 发明的。对图深度优先搜索，定义 DFS(u)为 u 在搜索树（以下简称为树）中被遍历到的次 序号。定义 Low(u)为 u 或 u 的子树中能通过非父子边追溯到的最早的节点，即 DFS 序号最小的节点。根据 定义，则有： Low(u)=Min { DFS(u) DFS(v) (u,v)为后向边(返祖边) 等价于 DFS(v)<DFS(u)且 v 不为 u 的父亲节点 Low(v) (u,v) 为树枝边(父子边) } 一个顶点 u 是割点，当且仅当满足(1)或(2) (1) u 为树根，且 u 有多于一个子树。 (2) u 不为树根，且满足存 在(u,v)为树枝边(或称父子边，即 u 为 v 在搜索树中的父亲)，使得 DFS(u)<=Low(v)。 一条无向边(u,v)是桥，当且仅当(u,v)为树枝边，且满足 DFS(u)<Low(v)。

**[求双连通分支]**

下面要分开讨论点双连通分支与边双连通分支的求法。 对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前 双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中。如果遇到某时满 足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)，取出的这些边与 其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边只属于且属于一 个点双连通分支。

对于边双连通分支，求法更为简单。只需在求出所有的桥以后，把桥边删除，原图变成了多个连通块，则 每个连通块就是一个边双连通分支。桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属 于一个边双连通分支。这个也可以用Tarjan有向图求强连通分量的思想实现：Tarjan算法在图中走无向边，形式与求强连通分量相似，但不走回路边，这样就能把一个无向图转化为有向图,环内的就是一个边双边连通分支。将所有边双连通分支缩点后，对于每条边，如果两顶点颜色不同，则该边为割边。

**[构造双连通图]**

一个有桥的连通图，如何把它通过加边变成边双连通图？方法为首先求出所有的桥，然后删除这些桥边， 剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这 个图一定是一棵树，边连通度为 1。 统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 leaf。则至少在树上添加(leaf+1)/2 条边，就能 使树达到边二连通，所以至少添加的边数就是(leaf+1)/2。具体方法为，首先把两个最近公共祖先最远的两 个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一 定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2 次， 把所有点收缩到了一起。

## 割点

无向图求割点。

重边对tarjan判割点无影响。

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **int** dfn[200005],low[200005];
6. **bool** isCutPoint[200005];
7. vector<**int**> adja[200005];
8. **int** tol=0;
9. //根节点的dfs儿子数目>=2，则为割点.其他点满足low[v]>=low[u]则是割点
10. **void** tarjan(**int** u,**int** fa)
11. {
12. **int** son=0;
13. dfn[u]=low[u]=++tol;
14. **for**(**int** v:adja[u]){
15. **if**(v==fa) **continue**;
16. **if**(!dfn[v]){
17. tarjan(v,u);
18. low[u]=min(low[u],low[v]);
19. **if**(u==fa) ++son;
20. **if**(u!=fa&&low[v]>=dfn[u]) isCutPoint[u]=**true**;
21. //if(low[v]>dfn[u]) u-v是个割边
22. }
23. **else**//被遍历过，因为是无向图,所以肯定v肯定是遍历过程的u祖先节点
24. low[u]=min(low[u],dfn[v]);
25. }
26. **if**(u==fa&&son>=2) isCutPoint[u]=**true**;
27. }
28. **void** solve(**int** n)//求出所有割点
29. {
30. //mset(dfn,0);mset(low,0);mset(isCutPoint,0); //因初始化都是0,这里就不初始化了
31. //tol=0;
32. **for**(**int** i=1;i<=n;++i){
33. **if**(!dfn[i]){
34. tarjan(i,i);//求割点传入的必须是i,i
35. }
36. }
37. }
38. **int** main()
39. {
40. ios::sync\_with\_stdio(**false**);cin.tie(0);
41. **int** n,m;
42. cin>>n>>m;
43. **for**(**int** i=0;i<m;++i){
44. **int** u,v;
45. cin>>u>>v;
46. adja[u].push\_back(v);
47. adja[v].push\_back(u);
48. }
49. solve(n);
50. **int** tot=0;
51. **for**(**int** i=1;i<=n;++i) **if**(isCutPoint[i]) tot++;
52. cout<<tot<<endl;
53. **for**(**int** i=1;i<=n;++i) **if**(isCutPoint[i]) cout<<i<<" ";
54. cout<<endl;
55. **return** 0;
56. }

## 桥

无向图求桥。

UVA796, 给出无向图,三无图，求出桥，输出割边并按字典序输出,

若有重边，则重边不会为桥，且在dfs过程中,可以重边的树枝边可以更新该语句low[u]=min(low[u],dfn[v])

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **const** **int** N=1e5+100;
5. **typedef** pair<**int**,**int**> P;
6. vector<**int**> g[N];
7. vector<P> cutEdge;
8. **int** low[N],dfn[N],tol;
9. **void** tarjan(**int** u,**int** fa)
10. {
11. low[u]=dfn[u]=++tol;
12. **bool** into=**false**;
13. **for**(**int** v:g[u]){
14. **if**(v==fa&&into==**false**){//into用来处理重边的情况
15. into=**true**;
16. **continue**;
17. }
18. **if**(!dfn[v]){
19. tarjan(v,u);
20. low[u]=min(low[u],low[v]);
21. **if**(low[v]>dfn[u]) cutEdge.push\_back({min(u,v),max(u,v)}); //u-v边为桥
22. }
23. **else**
24. low[u]=min(low[u],dfn[v]);
25. }
26. }
27. **void** solve(**int** n)
28. {
29. mset(dfn,0);tol=0;
30. cutEdge.clear();
31. **for**(**int** i=0;i<=n;++i){
32. **if**(!dfn[i])
33. tarjan(i,i);
34. }
35. cout<<cutEdge.size()<<" critical links"<<endl;
36. sort(cutEdge.begin(),cutEdge.end());
37. **for**(P &p:cutEdge){
38. cout<<p.first<<" - "<<p.second<<endl;
39. }
40. cout<<endl;
41. }
42. **int** main()
43. {
44. **int** n,maxn=0;
45. **while**(~scanf("%d",&n))
46. {
47. **for**(**int** i=0;i<=maxn;++i) g[i].clear();
48. **for**(**int** o=0;o<n;++o){
49. **int** u,cnt,v;
50. scanf("%d (%d)",&u,&cnt);
51. **for**(**int** i=0;i<cnt;++i){
52. scanf("%d",&v);
53. g[u].push\_back(v);
54. maxn=max(u,v);
55. }
56. }
57. solve(maxn);
58. }
59. **return** 0;
60. }

## 边双连通分支

无向图求边双连通分支。

去掉桥，其余的连通分支就是边双连通分支了。一个有桥的连通图要变成边双连通图的话，把双连通子图 收缩为一个点，形成一颗树。需要加的边为(leaf+1)/2

POJ 3177 给定一个连通的无向图G，求至少添加多少边，才能使其变为双联通图

1. #define mset(a,b) memset(a,b,sizeof(a))
2. **const** **int** N=1e5+100;
3. vector<**int**> g[N];
4. **int** low[N],dfn[N],tol,ins[N],top,S[N],cgt,color[N];
5. **void** tarjan(**int** u,**int** fa)
6. {
7. low[u]=dfn[u]=++tol;
8. ins[u]=1;
9. S[top++]=u;
10. **bool** in\_rep=**false**;
11. **for**(**int** i=0; i<g[u].size(); ++i)
12. {
13. **int** v=g[u][i];
14. **if**(v==fa&&in\_rep==**false**){//如果有重边,可通过重边回去
15. in\_rep=**true**;
16. **continue**;
17. }
18. **if**(!dfn[v])
19. {
20. tarjan(v,u);
21. low[u]=min(low[u],low[v]);
22. }
23. **else** **if**(ins[v]) low[u]=min(low[u],dfn[v]);
24. }
25. **if**(low[u]==dfn[u])
26. {
27. ++cgt;
28. **int** v;
29. **do**
30. {
31. v=S[--top];
32. ins[v]=0;
33. color[v]=cgt;
34. }
35. **while**(v!=u);
36. }
37. }
38. **int** du[N];
39. **void** solve(**int** n)
40. {
41. **for**(**int** i=1;i<=n;++i)
42. dfn[i]=ins[i]=color[i]=du[i]=0;
43. tol=cgt=top=0;
44. **for**(**int** i=1; i<=n; ++i) **if**(!dfn[i]) tarjan(i,i);
45. **for**(**int** u=1; u<=n; ++u)
46. {
47. **for**(**int** i=0; i<g[u].size(); ++i)
48. {
49. **int** v=g[u][i];
50. **if**(color[u]!=color[v]) // 因为是双向边,所有每个边只增加第一个
51. du[color[u]]++;
52. }
53. }
54. **int** leaf=0;
55. **for**(**int** i=1; i<=cgt; ++i)
56. {
57. **if**(du[i]==1) leaf++;
58. }
59. cout<<(leaf+1)/2<<endl;;
60. }
61. **int** main()
62. {
63. **int** n,m;
64. **while**(~scanf("%d%d",&n,&m))
65. {
66. **for**(**int** i=1; i<=n; ++i) g[i].clear();
67. **for**(**int** i=1; i<=m; ++i)
68. {
69. **int** u,v;
70. scanf("%d%d",&u,&v);
71. g[u].push\_back(v);
72. g[v].push\_back(u);
73. }
74. solve(n);
75. }
76. **return** 0;
77. }

## 点双联通分支

无向图中若G的某一子图G’连通，且删去任意一顶点，剩下的顶点仍连通，则子图G’为图G的一个点双连通分支

对于点双连通分支，实际上在求割点的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储 当前双连通分支，在搜索图时，每找到一条树枝边或后向边(非横叉边)，就把这条边加入栈中。如果遇到 某时满足 DFS(u)<=Low(v)，说明 u 是一个割点，同时把边从栈顶一个个取出，直到遇到了边(u,v)， 取出的这些边与其关联的点，组成一个点双连通分支。割点可以属于多个点双连通分支，其余点和每条边 只属于且属于一个点双连通分支 ---来自kuangbin

例题

/\* POJ 2942 Knights of the Round Table

1. 亚瑟王要在圆桌上召开骑士会议，为了不引发骑士之间的冲突，
2. 并且能够让会议的议题有令人满意的结果，每次开会前都必须对出席会议的骑士有如下要求：
3. 1、  相互憎恨的两个骑士不能坐在直接相邻的2个位置；
4. 2、  出席会议的骑士数必须是奇数，这是为了让投票表决议题时都能有结果。
6. 注意：
7. 1、所给出的憎恨关系一定是双向的，不存在单向憎恨关系。
8. 2、由于是圆桌会议，则每个出席的骑士身边必定刚好有2个骑士。 即每个骑士的座位两边都必定各有一个骑士。
9. 3、一个骑士无法开会，就是说至少有3个骑士才可能开会。
11. 首先根据给出的互相憎恨的图中得到补图。 然后就相当于找出不能形成奇圈的点。
12. 利用下面两个定理：
13. （1）如果一个点双连通分量内的某些顶点在一个奇圈中（即点双连通分量含有奇圈）， 那么这个点双连通分量的其他顶点也在某个奇圈中；
14. （2）如果一个点双连通分量含有奇圈，则他必定不是一个二分图。反过来也成立，这是一个充要条件。
16. 所以本题的做法，就是对补图求点双连通分量。
17. 然后对于求得的点双连通分量，使用染色法判断是不是二分图，不是二分图，这个双连通分量的点是可以 存在的 \*/
18. #define mset(a,b) memset(a,b,sizeof(a))
19. **using** **namespace** std;
20. **const** **int** N=1e3+10;
21. **bool** hate[N][N];
22. vector<**int**> g[N];
23. **int** dfn[N],tol,low[N],S[N],top,belong[N],block;
24. **bool** ins[N];
25. **bool** have[N],canjoin[N];//have[i]表示i点是否在当前点双连通分量中
26. **int** temp[N];//temp用来储存当前点双联通分量的点,用于记录可以参加会议的骑士
27. **int** color[N];
28. **bool** dfs(**int** u,**int** sg)//二分图交叉染色法
29. {
30. **if**(color[u]==sg) **return** **true**;
31. **if**(color[u]!=-1&&color[u]!=sg) **return** **false**;
32. color[u]=sg;
33. **for**(**int** i=0;i<g[u].size();++i)
34. {
35. **int** v=g[u][i];
36. **if**(have[v]){
37. **if**(!dfs(v,sg^1)) **return** **false**;
38. }
39. }
40. **return** **true**;
41. }
42. **bool** judge(**int** tot)//有奇圈返回true，没有奇圈(是二分图)返回false
43. {
44. mset(color,-1);
45. **bool** ok=dfs(temp[0],0);
46. **return** !ok;
47. }
48. **void** tarjan(**int** u,**int** fa)
49. {
50. dfn[u]=low[u]=++tol;
51. S[top++]=u;
52. ins[u]=**true**;
53. **for**(**int** i=0;i<g[u].size();++i)
54. {
55. **int** v=g[u][i];
56. **if**(v==fa) **continue**;
57. **if**(!dfn[v]){
58. tarjan(v,u);
59. low[u]=min(low[u],low[v]);
60. **if**(low[v]>=dfn[u])
61. {
62. **int** vn,tot=0;
63. mset(have,0);
64. block++;
65. **do**{
66. vn=S[--top];
67. ins[vn]=**false**;
68. belong[vn]=block;
69. /\*这里存储所有点双连通分量的点,用于检测奇环\*/
70. have[vn]=**true**;
71. temp[tot++]=vn;
72. }
73. **while**(vn!=v);
74. //注意u也是该点双连通分量的点,但在这里并没有把u加入, 因为一个割点可以在多个点双连通分量。
75. /\*下面是处理判断该点双连通分量能否参加会议\*/
76. have[u]=**true**;
77. temp[tot++]=u;
78. **if**(judge(tot))
79. {
80. **while**(tot--) canjoin[temp[tot]]=**true**;
81. }
82. }
83. }
84. **else** **if**(ins[v])
85. low[u]=min(low[u],dfn[v]);
86. }
87. }
88. **void** solve(**int** n)
89. {
90. top=tol=block=0;
91. mset(dfn,0);mset(belong,0);mset(ins,0);
92. mset(canjoin,0);
93. **for**(**int** i=1;i<=n;++i) **if**(!dfn[i]) tarjan(i,i);
94. **int** ans=0;
95. **for**(**int** i=1;i<=n;++i)
96. **if**(!canjoin[i]) ans++;
97. printf("%d\n",ans);
98. }
99. **int** main()
100. {
101. **int** n,m;
102. **while**(scanf("%d%d",&n,&m),n)
103. {
104. mset(hate,0);
105. **for**(**int** i=0;i<m;++i){
106. **int** u,v;
107. scanf("%d%d",&u,&v);
108. hate[u][v]=**true**;
109. hate[v][u]=**true**;
110. }
111. **for**(**int** i=1;i<=n;++i) g[i].clear();
112. **for**(**int** i=1;i<=n;++i)
113. {
114. **for**(**int** j=1;j<=n;++j)
115. **if**(!hate[i][j]&&i!=j)
116. g[i].push\_back(j);
117. }
118. solve(n);
119. }
120. **return** 0;

## 强连通分量

有向图求强连通分量tarjan算法。

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **typedef** pair<**int**,**int**> P;
5. **const** **int** N=2e5+10;
6. **int** S[N],dfn[N],low[N],ins[N],top=0,tol=0;//top
7. **int** color[N],cgt=0;
8. vector<**int**> adja[N];//单向图的图
9. //dfn[i]: 第i个点遍历的时间戳，未遍历过为0
10. //low[i]:在bfs树中，第i个点可到达的最早时间戳，且low[i]可以到达i。
11. //ins[i]:代表第i个顶点是否在栈中
12. //tol:当前遍历节点数
13. //cgt:当前强联通块数
14. //color[i]=k,代表第i个节点属于第k个强连通块
15. **void** tarjan(**int** u)//求出low[u]，如果是这个强连通分量的代表并处理强连通分量
16. {
17. dfn[u]=low[u]=++tol;
18. S[top++]=u;ins[u]=1;
19. **for**(**int** i=0;i<adja[u].size();++i){
20. **int** v=adja[u][i];
21. **if**(!dfn[v]){//v未被遍历过
22. tarjan(v);
23. low[u]=min(low[u],low[v]);
24. }
25. **else** **if**(ins[v]){//v被遍历过，且是自己的祖先
26. low[u]=min(low[u],dfn[v]);
27. }
28. }
29. **if**(low[u]==dfn[u])//这是一个环
30. {
31. **int** v;
32. ++cgt;
33. **do**{//让该环上的点都出队,然后都属于同一个联通块
34. v=S[top-1];
35. top--;ins[v]=0;
36. color[v]=cgt;
38. }
39. **while**(v!=u);
40. }
41. }
42. **void** getAll(**int** n)
43. {  //清空dfn[],color[],ins[],tol=cgt=top=0
44. **for**(**int** i=1;i<=n;++i){//求出所有强连通分量,
45. **if**(!dfn[i]) tarjan(i);
46. }
47. }
48. **int** main()
49. {
51. }

## LCA最近公共祖先

### lCA在线倍增算法

无向图，有向图都适用，不过需要指出树的根。

1. **const** **int** N=5e5+10;
2. **const** **int** DEG=20;
4. vector<**int**> adja[N];
5. **int** deg[N];//deg[i],代表节点i的深度
6. **int** fa[N][DEG];//fa[i][j]表示节点i的第2^j个祖先.如果超过的话，他们都是root
8. //BFS预处理倍增数组fa和深度数组deg O(nlogn)
9. **void** bfs(**int** root)
10. {
11. queue<**int**> qe;
12. fa[root][0]=root;deg[root]=0;
13. qe.push(root);
14. **while**(!qe.empty())
15. {
16. **int** u=qe.front();qe.pop();
17. **for**(**int** i=1;i<DEG;++i) fa[u][i]=fa[fa[u][i-1]][i-1];//倍增状态转移
18. **for**(**int** v:adja[u]){
19. **if**(v==fa[u][0]) **continue**;
20. fa[v][0]=u;
21. deg[v]=deg[u]+1;
22. qe.push(v);
23. }
24. }
25. }
27. //在线查询u,v的lca.时间复杂度O(logn)
28. **int** LCA(**int** u,**int** v)
29. {
30. **if**(deg[u]>deg[v]) swap(u,v);
31. **int** tu=u,tv=v;
32. **for**(**int** det=deg[v]-deg[u],i=0;det;det>>=1,++i)//先让最深的移动到与浅的深度相同的位置
33. **if**(det&1) tv=fa[tv][i];
34. **if**(tv==tu) **return** tv;//u是lca
35. **for**(**int** i=DEG-1;i>=0;--i)//找tu和tv的祖先o,最后使得tu和tv的父亲就是o
36. {
37. **if**(fa[tu][i]!=fa[tv][i]) {
38. tu=fa[tu][i];
39. tv=fa[tv][i];
40. }
41. }
42. **return** fa[tu][0];
43. }

### Lca求树上两点路径距离

在线倍增算法：

无向图和有向图都适用，需要指出树的根（若只求两点最短路径距离，随便指出哪个为根都可以）

1. **const** **int** DEG=20;
2. vector<P> g[N];//存储边 v,w=(first,second)
3. **int** fa[N][DEG+1],deg[N];  //lca祖先倍增数组和深度数组
4. **int** dis[N][DEG+1];//祖先距离倍增数组，dis[u][i]为u到2^i个祖先的路径和
5. **void** bfs(**int** root)
6. {
7. queue<**int**> Q;
8. Q.push(root);
9. fa[root][0]=root;
10. dis[root][0]=0;
11. deg[root]=0;
12. **while**(!Q.empty())
13. {
14. **int** u=Q.front();
15. Q.pop();
16. **for**(**int** i=1; i<=DEG; ++i) {
17. fa[u][i]=fa[fa[u][i-1]][i-1];
18. dis[u][i]=dis[u][i-1]+dis[fa[u][i-1]][i-1];
19. }
20. **for**(P& e:g[u])
21. {
22. **int** v=e.first,w=e.second;
23. **if**(v==fa[u][0]) **continue**;
24. deg[v]=deg[u]+1;
25. fa[v][0]=u;
26. dis[v][0]=w;
27. Q.push(v);
28. }
29. }
30. }
31. **int** LCAds(**int** u,**int** v)  //返回u,v两点的最短路径长度
32. {
33. **int** ans=0;
34. **if**(deg[u]>deg[v]) swap(u,v);
35. **int** tu=u,tv=v;
36. **for**(**int** det=deg[tv]-deg[tu],i=0; det; ++i)
37. {
38. **if**(det&(1<<i))
39. {
40. ans+=dis[tv][i];
41. det-=(1<<i);
42. tv=fa[tv][i];
43. }
44. }
45. **if**(tu==tv) **return** ans;
46. **for**(**int** i=DEG; ~i; --i)
47. {
48. **if**(fa[tu][i]!=fa[tv][i])
49. {
50. ans+=dis[tv][i];
51. ans+=dis[tu][i];
52. tu=fa[tu][i];
53. tv=fa[tv][i];
54. }
55. }
56. ans+=dis[tu][0];
57. ans+=dis[tv][0];
58. **return** ans;
59. }

# 数据结构

## 树状数组

### 一维树状数组

1. **int** lowbit(**int** val){ //获得val的1的最低位的二进制权值
2. **return**  val&(-val);
3. }
4. **int** getsum(**int** k)//遍历1~k区间的管理节点。
5. {
6. **int** ans=0;
7. **while**(k>0)
8. {
9. ans+=bt[k];
10. k-=lowbit(k);
11. }
12. **return** ans;
13. }
14. **void** modify(**int** k,**int** val)// 维护树状数组信息，向上更新管理叶子节点k的节点信息。
15. {
16. **while**(k<=MAX)
17. {
18. bt[k]+=val;
19. k+=lowbit(k);
20. }
21. }

### 二维树状数组

**更点查块：**

1. **const** ll MAXN=1050;
2. ll MAX=1025;
3. ll bt[MAXN][MAXN];
4. ll lowbit(ll k)
5. {
6. **return** k&-k;
7. }
8. ll getsum(ll x,ll y)//求矩阵（1，1）到（x，y）的元素和
9. {
10. ll ans=0;
11. **for**(ll i=x;i>0;i-=lowbit(i))
12. **for**(ll j=y;j>0;j-=lowbit(j))
13. ans+=bt[i][j];
14. **return** ans;
15. }
16. **void** modify(ll x,ll y,ll val) //维护树状数组信息，向上更新所有与他有关的节点
17. {
18. **for**(ll i=x;i<=MAX;i+=lowbit(i))
19. **for**(ll j=y;j<=MAX;j+=lowbit(j))
20. bt[i][j]+=val;;
21. }
22. ll calc(ll x1,ll y1,ll x2,ll y2) //计算子矩阵的和
23. {
24. ll ans=getsum(x2,y2)-getsum(x1-1,y2)-getsum(x2,y1-1)+getsum(x1-1,y1-1);
25. **return** ans;
26. }
27. **int** main(){
28. ll n,cm;
29. scanf("%lld %lld",&cm,&n);
30. MAX=n+1;
31. **while**(scanf("%lld",&cm),cm!=3)
32. {
33. **if**(cm==1)
34. {
35. ll x,y,val;
36. scanf("%lld%lld%lld",&x,&y,&val);
37. ++x,++y;
38. modify(x,y,val);
39. }
40. **else**
41. {
42. ll x1,x2,y1,y2;
43. scanf("%lld%lld%lld%lld",&x1,&y1,&x2,&y2);
44. ++x1,++y1,++x2,++y2;
45. printf("%lld\n",calc(x1,y1,x2,y2));
46. }//
47. }
48. **return** 0;
49. }

**更块查点：**

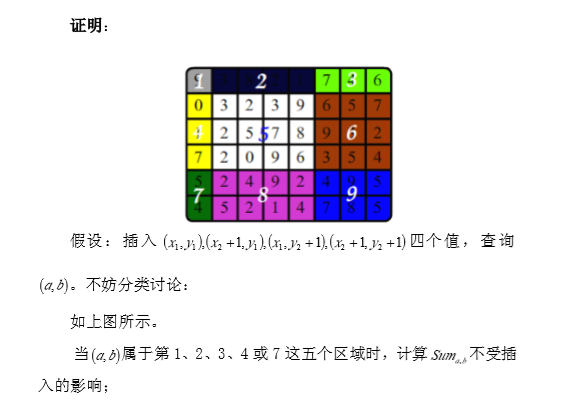
**原理：**

二维树状数组成块更新，查定点的值。

我们以一维的推广，我们可不可以成块更新转化为修改一些点的值，然后求点的值转化统计一块的和？

我们可以构造一个M矩阵，让M矩阵初始化为0，我们可以让坐标（x，y）的值等于以（1，1）和（x，y）组成的M子矩阵的元素之和（你看看，这个条件在刚开始的时候是成立的），我们更新以（x1，y1）和（x2，y2）组成的矩阵的值的时候，我们只需在M矩阵中插入（x1，y1）（x1，y2+1）（x2+1，y1）（x2+1，y2+1）的值。

比如我们将以（x1，y1）和（x2，y2）组成的矩阵的元素都加上val。那么我们将（x1，y1）+val，（x1，y2+1）-val，（x2+1，y1）-val，（x2+1，y2+1）+val即可。下面的图片证明该操作成立



1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **int** bt[1100][1100];
4. **int** MAXN,MAXM;//矩阵的行和列
5. **int** lowbit(**int** k)
6. {
7. **return** k&-k;
8. }
9. **void** modify(**int** x,**int** y,**int** val){//维护树状数组信息，更新所有与他有关的节点
10. **for**(**int** i=x;i<=MAXN;i+=lowbit(i)){
11. **for**(**int** j=y;j<=MAXM;j+=lowbit(j))
12. bt[i][j]+=val;
13. }
14. }
15. **int** getsum(**int** x,**int** y)//统计信息，返回其左上角矩阵的和
16. {
17. **int** ans=0;
18. **for**(**int** i=x;i>0;i-=lowbit(i)){
19. **for**(**int** j=y;j>0;j-=lowbit(j))
20. ans+=bt[i][j];
21. }
22. **return** ans;
23. }
24. **void** update(**int** x1,**int** y1,**int** x2,**int** y2,**int** val)//更新差值矩阵
25. {
26. modify(x1,y1,val);
27. modify(x1,y2+1,-val);
28. modify(x2+1,y1,-val);
29. modify(x2+1,y2+1,val);
30. }
31. **int** main()
32. {
33. **int** q,val,x,y,x1,y1,x2,y2;
34. cin>>MAXN>>MAXM;
35. cin>>q;
36. **while**(q--){//修改操作
37. cin>>x1>>y1>>x2>>y2>>val;
38. update(x1,y1,x2,y2,val);
39. }
40. cin>>q;
41. **while**(q--)
42. {
43. cin>>x>>y;
44. **int** ans=getsum(x,y);
45. cout<<ans<<endl;
46. }
47. }

题意：对一个n∗m的矩阵进行一些操作和查询，操作：成块修改。查询：求某个元素的值，

## RMQ

一维RMQ，求区间最大最小值

1. /\*
2. n:b[]元素个数
3. b[]:数据数组,下标1..n
4. \*/
5. **int** mm[maxn];
6. **void** initRMQ(**int** n,**int** b[])
7. {
8. mm[0]=-1;
9. **for**(**int** i=1;i<=n;++i){
10. mm[i]=( (i&(i-1))==0 )?mm[i-1]+1:mm[i-1];
11. dp[i][0]=b[i];
12. }
13. **for**(**int** j=1;j<=mm[n];++j)
14. **for**(**int** i=1;i+(1<<j)-1 <=n ;++i){
15. dp[i][j]=max(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
16. }
17. }
18. **int** rmq(**int** x,**int** y)//查询下标x到y的最大值
19. {
20. **int** k=mm[y-x+1];
21. **return** max(dp[x][k],dp[y-(1<<k)+1][k]);
22. }

## 单调栈和单调队列





**洛谷1886滑动窗口（单调队列）:** 

**代码：**

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **typedef** pair<**int**,**int**> P;
6. **const** **int** inf=0x3f3f3f3f;
7. deque<P> swmax;//最大值单调队列(位置,大小)
8. deque<P> swmin;//最小值单调队列(位置,大小)
9. vector<**int**> ansmax,ansmin;//存放答案
10. vector<**int**> A(1000000+10);
11. **int** main()
12. {
13. ios::sync\_with\_stdio(**false**);cin.tie(0);
14. **int** n,k;
15. cin>>n>>k;
16. **for**(**int** i=1;i<=n;++i) cin>>A[i];
17. **for**(**int** i=1;i<=k;++i){
18. //维护max windows
19. **while**(!swmax.empty()&&A[i]>=swmax.back().second) swmax.pop\_back();
20. swmax.push\_back({i,A[i]});
21. //维护min windows
22. **while**(!swmin.empty()&&A[i]<=swmin.back().second) swmin.pop\_back();
23. swmin.push\_back({i,A[i]});
24. }
25. ansmax.push\_back(swmax.front().second);
26. ansmin.push\_back(swmin.front().second);
27. **for**(**int** i=k+1;i<=n;++i){
28. // 放入最大值单调队列
29. **while**(!swmax.empty()&&A[i]>=swmax.back().second) swmax.pop\_back();
30. swmax.push\_back({i,A[i]});
31. // 放入最小值单调队列
32. **while**(!swmin.empty()&&A[i]<=swmin.back().second) swmin.pop\_back();
33. swmin.push\_back({i,A[i]});
35. **int** l=i-k+1;
36. //从中取出指定范围的最大值
37. **while**(swmax.front().first<l) swmax.pop\_front();
38. ansmax.push\_back(swmax.front().second);
39. //从中取出指定范围的最小值
40. **while**(swmin.front().first<l) swmin.pop\_front();
41. //存放答案
42. ansmin.push\_back(swmin.front().second);
43. }
44. **for**(**int** i:ansmin){
45. cout<<i<<" ";
46. }
47. cout<<endl;
48. **for**(**int** i:ansmax){
49. cout<<i<<" ";
50. }
51. cout<<endl;
52. }

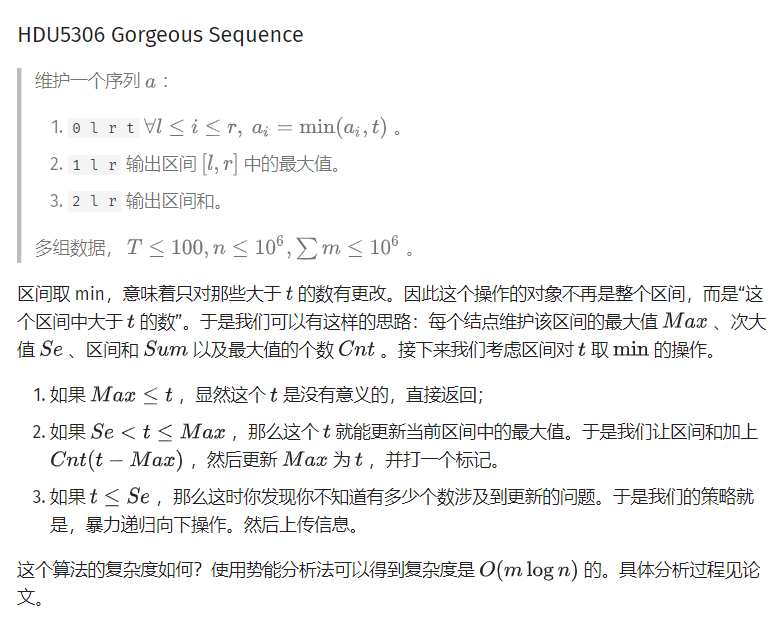
## 线段树

**写下这个是为了保留线段树的简洁写法，并且快速回忆线段树的写法，但这的例题可能不太适合(不简单)：**

注：何时push\_down,push\_up？

当懒标记会影响接下来（对儿子查询或修改）的操作时，需要push\_down让儿子更新到最新。

当当前节点信息依靠对儿子的信息结合后时，需要对当前节点push\_up，使得当前节点信息变为最新。



**代码：**

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. #define lson u<<1,l,m
4. #define rson u<<1|1,m+1,r
5. **using** **namespace** std;
6. **typedef** **long** **long** ll;
7. **const** **int** N=1e6+20;
8. **const** **int** inf=0x3f3f3f3f;
9. **int** w[N];
10. **int** n;
11. **struct** SegmentTree{
12. **int** mx[N<<2],se[N<<2],cn[N<<2],tag[N<<2];
13. ll su[N<<2];
14. **void** push\_up(**int** u)//根据儿子节点更新自己(区间)
15. {
16. su[u]=su[u<<1]+su[u<<1|1];
17. **if**(mx[u<<1]==mx[u<<1|1]){
18. mx[u]=mx[u<<1];
19. cn[u]=cn[u<<1]+cn[u<<1|1];
20. se[u]=max(se[u<<1],se[u<<1|1]);
21. }
22. **else** **if**(mx[u<<1] > mx[u<<1|1])
23. {
24. mx[u]=mx[u<<1];
25. cn[u]=cn[u<<1];
26. se[u]=max(se[u<<1],mx[u<<1|1]);
27. }
28. **else**{
29. mx[u]=mx[u<<1|1];
30. cn[u]=cn[u<<1|1];
31. se[u]=max(se[u<<1|1],mx[u<<1]);
32. }
33. }
34. **void** putTag(**int** u,**int** t)//laze,不下传
35. {// 因为t<se[u]是暴力修改的,所以每个节点的laze 都满足se<=t
36. **if**(mx[u]<=t) **return**;
37. su[u]-=1ll\*cn[u]\*(1ll\*mx[u]-t);
38. //     mx[u]=tag[u]=t;
39. mx[u]=t;
40. **if**(tag[u]==-1){
41. tag[u]=t;
42. }
43. **else** tag[u]=min(tag[u],t);
44. }
45. **void** push\_down(**int** u,**int** l,**int** r)
46. {
47. //传tag
48. **if**(tag[u]==-1) **return** ;
49. putTag(u<<1,tag[u]);
50. putTag(u<<1|1,tag[u]);
51. tag[u]=-1;
52. }
53. **void** bulid(**int** u,**int** l,**int** r)
54. {
55. tag[u]=-1;
56. **if**(l==r){
57. mx[u]=su[u]=w[l];
58. cn[u]=1,se[u]=-1;
59. **return** ;
60. }
61. **int** m=l+r>>1;
62. bulid(lson);
63. bulid(rson);
64. push\_up(u);
65. }
66. **void** modifyMin(**int** ql,**int** qr,**int** t,**int** u=1,**int** l=1,**int** r=n)
67. {
68. **if**(mx[u]<=t) **return** ;
69. **if**(l>=ql&&r<=qr&&se[u]<t){
70. //修改最大值,修改和,儿子更新
71. putTag(u,t);
72. **return** ;
73. }
74. push\_down(u,l,r);
75. **int** m=l+r>>1;
76. **if**(m>=ql) modifyMin(ql,qr,t,lson);
77. **if**(m<qr)  modifyMin(ql,qr,t,rson);
78. push\_up(u);
79. }
80. **int** queryMax(**int** ql,**int** qr,**int** u=1,**int** l=1,**int** r=n)
81. {
82. **if**(ql<=l&&r<=qr){
83. **return** mx[u];
84. }
85. push\_down(u,l,r);
86. **int** maxx=-1,m=l+r>>1;
87. **if**(m>=ql){
88. maxx=queryMax(ql,qr,lson);
89. }
90. **if**(m<qr){
91. maxx=max(maxx,queryMax(ql,qr,rson));
92. }
93. **return** maxx;
94. }
95. ll querySum(**int** ql,**int** qr,**int** u=1,**int** l=1,**int** r=n)
96. {
97. **if**(ql<=l&&r<=qr){
98. **return** su[u];
99. }
100. push\_down(u,l,r);
101. ll sum=0;
102. **int** m=l+r>>1;
103. **if**(m>=ql) sum+=querySum(ql,qr,lson);
104. **if**(m<qr) sum+=querySum(ql,qr,rson);
105. **return** sum;
106. }
107. }ooo;
108. **int** main()
109. {
110. ios::sync\_with\_stdio(**false**);cin.tie(0);cout.tie(0);
111. **int** T;
112. cin>>T;
113. **while**(T--)
114. {
115. **int** m;
116. cin>>n>>m;
117. **for**(**int** i=1;i<=n;++i) cin>>w[i];  //输入初始序列
118. ooo.bulid(1,1,n);
119. **for**(**int** i=1;i<=m;++i)  //m个操作
120. {
121. **int** cmd,l,r,t;
122. cin>>cmd>>l>>r;
123. **if**(cmd==0){
124. cin>>t;
125. ooo.modifyMin(l,r,t);
126. }
127. **if**(cmd==1){
128. cout<<ooo.queryMax(l,r)<<endl;
129. }
130. **if**(cmd==2){
131. cout<<ooo.querySum(l,r)<<endl;
132. }
133. }
134. }
135. **return** 0;
136. }

## 可持久化线段树

数组大小：可持久化线段树每次单点修改形成的线段树只添加了O(log(n))个节点，一般开20-40倍

1. /\*
2. 题目描述：
3. 如题，给定N个整数构成的序列，将对于指定的闭区间查询其区间内的第K小值。
5. 输入格式：
6. 第一行包含两个正整数N、M，分别表示序列的长度和查询的个数。
7. 第二行包含N个整数，表示这个序列各项的数字。
8. 接下来M行每行包含三个整数l, r, k 表示查询区间[l,r]内的第k小值。
10. 输出格式：
11. 输出包含k行，每行1个整数，依次表示每一次查询的结果
13. \*/
14. #include <bits/stdc++.h>
15. **using** **namespace** std;
16. **typedef** **long** **long** ll;
17. **const** **int** N = 2e5+5;
18. **int** T[N];
19. **struct** HJTtree
20. {
21. **int** sum[N\*20],L[N\*20],R[N\*20];
22. **int** top;
23. **void** init()
24. {
25. top=0;
26. }
27. **void** bulid(**int** &rt,**int** l,**int** r)//建立第一颗空的线段树
28. {
29. rt= ++top;
30. sum[rt]=0;
31. **if**(l==r) **return** ;
32. **int** m=l+r>>1;
33. bulid(L[rt],l,m);
34. bulid(R[rt],m+1,r);
35. }
36. **void** update(**int** &nt,**int** l,**int** r,**int** pt,**int** k)//会给当前节点定义新的编号
37. {
38. nt=++top;
39. sum[nt]=sum[pt]+1;
40. L[nt]=L[pt];
41. R[nt]=R[pt];
42. **if**(l==r) **return** ;
43. **int** m=l+r>>1;
44. **if**(k<=m) update(L[nt],l,m,L[pt],k);
45. **else** update(R[nt],m+1,r,R[pt],k);
46. }
47. **int** query(**int** nt,**int** l,**int** r,**int** pt,**int** k)
48. {
49. **if**(l==r) **return** l;
50. **int** m=l+r>>1;
51. **if**(k<=sum[L[nt]]-sum[L[pt]]) **return** query(L[nt],l,m,L[pt],k);
52. **else** **return** query(R[nt],m+1,r,R[pt],k-(sum[L[nt]]-sum[L[pt]]));
53. }
54. } ooo;
55. **int** w[N];
56. vector<**int**> sv;
57. **int** getid(**int** x)
58. {
59. **return** lower\_bound(sv.begin(),sv.end(),x)-sv.begin()+1;
60. }
61. **int** main()
62. {
63. **int** n,m;
64. scanf("%d%d",&n,&m);
65. sv.clear();
66. **for**(**int** i=1; i<=n; ++i)
67. {
68. scanf("%d",w+i);
69. sv.push\_back(w[i]);
70. }
71. //离散化
72. sort(sv.begin(),sv.end());
73. sv.erase(unique(sv.begin(),sv.end()),sv.end());
74. **int** sz=**int**(sv.size());
75. **for**(**int** i=1; i<=n; ++i) w[i]=getid(w[i]);
76. //离散化完成
77. ooo.init();
78. ooo.bulid(T[0],1,sz);
79. **for**(**int** i=1; i<=n; ++i) ooo.update(T[i],1,sz,T[i-1],w[i]);
80. **for**(**int** i=1; i<=m; ++i)
81. {
82. **int** l,r,k;
83. scanf("%d%d%d",&l,&r,&k);
84. **int** th=ooo.query(T[r],1,sz,T[l-1],k);
85. printf("%d\n",sv[th-1]);
86. }
87. **return** 0;
88. }

## 珂朵莉树/old\_driver\_tree

注：此算法只有在区间推平操作和区间修改查询操作都有且数据随机情况下，复杂度正确约为O(nlogn)。但数据不随机出题人可以卡成O(n^2logn)

貌似在数据随机下时间复每次操作大概需要O(n/q)次在set中查找。

**核心思想：**

把值相同的区间合并成一个结点保存在 set 里面。

首先，结点的保存方式：

1. **struct** Node\_t {
2. **int** l, r;
3. **mutable** **int** v;
4. Node\_t(**const** **int** &il, **const** **int** &ir, **const** **int** &iv) : l(il), r(ir), v(iv) {}
5. **inline** **bool** operator<(**const** Node\_t &o) **const** { **return** l < o.l; }
6. };

其中， int v 是你自己指定的附加数据。然后，我们定义一个 set<Node\_t> odt; 来维护这些结点。 为简化代码，可以 typedef set<Node\_t>::iterator iter ，当然在题目支持 C++11 时也可以使用 auto 。

**split**[:](https://oi-wiki.org/ds/odt/#split)

split 是最核心的操作之一，它用于将原本包含点 x的区间（设为 [ l, r] ）分裂为两个区间 [ l, x )和 [ x,r ] 并返回指向后者的迭代器。 参考代码如下：

1. auto split(**int** x) {
2. **if** (x > n) **return** odt.end();
3. auto it = --odt.upper\_bound((Node\_t){x, 0, 0});
4. **if** (it->l == x) **return** it;
5. **int** l = it->l, r = it->r, v = it->v;
6. odt.erase(it);
7. odt.insert(Node\_t(l, x - 1, v));
8. **return** odt.insert(Node\_t(x, r, v)).first;
9. }

这个玩意有什么用呢？ 任何对于[ l, r ]的区间操作，都可以转换成 set 上  [split(l),split(r+1)]的操作。

**assign**[:](https://oi-wiki.org/ds/odt/#assign)

另外一个重要的操作 assign 用于对一段区间进行赋值。 对于 ODT 来说，区间操作只有这个比较特殊，**也是保证复杂度的关键**。 如果 ODT 里全是长度为 1 的区间，就成了暴力，但是有了 assign ，可以使 ODT 的大小下降。 参考代码如下：

1. **void** assign(**int** l, **int** r, **int** v) {
2. auto itr = split(r + 1), itl = split(l);
3. odt.erase(itl, itr);
4. odt.insert(Node\_t(l, r, v));
5. }

**其他操作:**

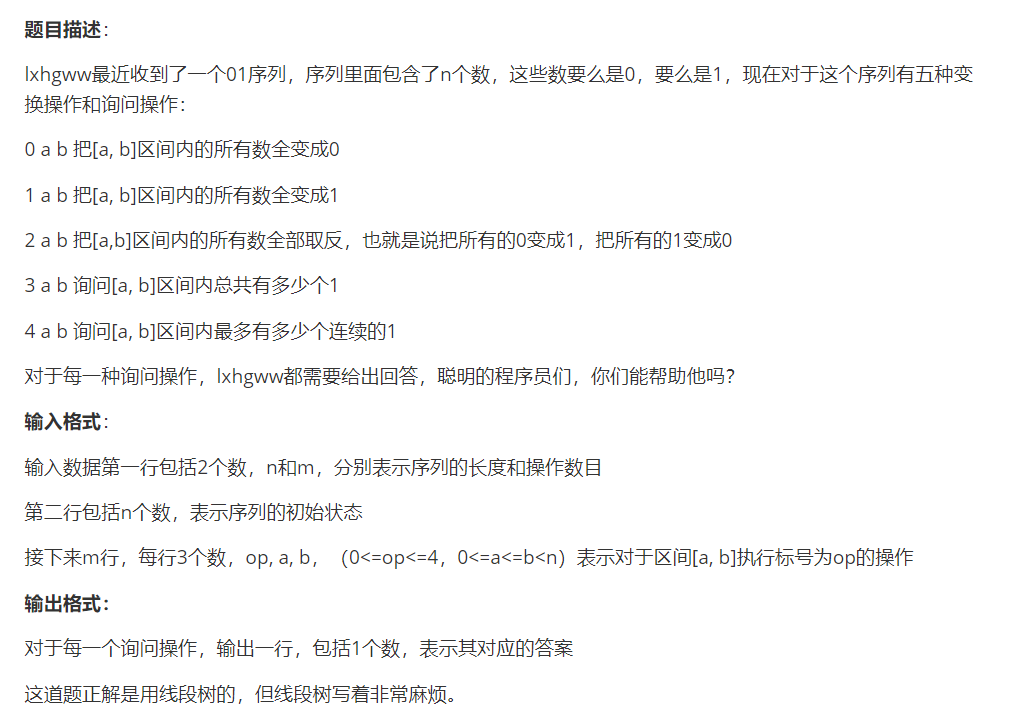
套模板就好了，参考代码如下：

1. **void** performance(**int** l, **int** r) {
2. auto itr = split(r + 1), itl = split(l);
3. **for** (; itl != itr; ++itl) {
4. // Perform Operations here
5. }
6. }

**注：珂朵莉树在进行求取区间左右端点操作时，必须先 split 右端点，再 split 左端点。否则在处理边界情况时会导致 RE。**

例题：

**P2572 [SCOI2010]序列操作**



**代码**：

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **typedef** pair<**int**,**int**> P;
6. **const** **int** N=1e5+20;
7. **struct** ODT\_tree
8. {
9. **struct** node//闭区间[l,r]
10. {
11. **int** l,r,v;
12. node(**int** l,**int** r,**int** v):l(l),r(r),v(v) {}
13. node() {}
14. **bool** operator < (**const** node & o) **const**
15. {
16. **return** l<o.l;
17. }
18. };
19. **int** n;
20. set<node> odtst;
21. **typedef** set<node>::iterator iter;
22. **inline** **void** iadd(**int** l,**int** r,**int** v)//init\_add
23. {
24. odtst.insert(node(l,r,v));
25. }
26. iter split(**int** x)
27. {
28. **if** (x > n) **return** odtst.end();
29. auto it = --odtst.upper\_bound(node(x,0,0));
30. **if** (it->l == x) **return** it;
31. **int** l = it->l, r = it->r, v = it->v;
32. odtst.erase(it);
33. odtst.insert(node(l, x - 1, v));
34. **return** odtst.insert(node(x, r, v)).first;
35. }
36. **void** assign(**int** l, **int** r, **int** v)
37. {
38. auto itr = split(r + 1), itl = split(l);
39. odtst.erase(itl, itr);//可能有很多个,也可能很少
40. odtst.insert(node(l, r, v));
41. pt();
42. }
43. **void** up1(**int** l,**int** r)//[l,r]区间取反
44. {
45. auto itr = split(r + 1), itl = split(l);
46. **for** (; itl != itr;)
47. {
48. //            itl->v^=1;
49. **int** a=itl->l,b=itl->r,v=itl->v;
50. odtst.erase(itl++);
51. odtst.insert(node(a,b,v^1));
52. }
53. pt();
54. }
55. **int** qy1(**int** l,**int** r)//询问区间[l,r] 1的个数
56. {
57. auto itr = split(r + 1), itl = split(l);
58. **int** ans=0;
59. **for** (; itl != itr; ++itl)
60. {
61. **if**(itl->v==1) ans+=itl->r - (itl->l)+1;
62. // Perform Operations here
63. }
64. **return** ans;
65. }
66. **int** qy2(**int** l,**int** r)//询问区间[l,r]最多有多少个连续的1
67. {
68. auto itr = split(r + 1), itl = split(l);
69. **int** lastv=-1,ls=0;
70. **int** ans=0;
71. **for** (; itl != itr; ++itl)
72. {
73. **if**(itl->v==lastv){
74. ls+=itl->r - itl->l + 1;
75. **if**(lastv==1 )ans=max(ans,ls);
76. }
77. **else**{
78. lastv=itl->v;
79. ls=itl->r - itl->l + 1;
80. **if**(lastv==1 )ans=max(ans,ls);
81. }
82. // Perform Operations here
83. }
84. **return** ans;
85. }
86. }odt;
87. **int** data[N];
88. **int** main()
89. {
90. **int** n,q;
91. scanf("%d%d",&n,&q);
92. odt.n=n-1;
93. **for**(**int** i=0; i<n; ++i)
94. {
95. scanf("%d",data+i);
96. }
97. **int** last=-1,ls=0;
98. **for**(**int** i=0; i<n; ++i)
99. {
100. **if**(last==-1)
101. {
102. last=i;
103. ls=1;
104. **continue**;
105. }
106. **if**(data[i]==data[i-1]){
107. ls++;
108. }
109. **else**{
110. odt.iadd(last,last+ls-1,data[i-1]);
111. last=i;
112. ls=1;
113. }
114. }
115. odt.iadd(last,last+ls-1,data[n-1]);
116. **for**(**int** i=1;i<=q;++i)
117. {
118. **int** cmd,a,b;
119. scanf("%d%d%d",&cmd,&a,&b);
120. **if**(cmd==0){
121. odt.assign(a,b,0);
122. }
123. **else** **if**(cmd==1){
124. odt.assign(a,b,1);
125. }
126. **else** **if**(cmd==2){
127. odt.up1(a,b);
128. }
129. **else** **if**(cmd==3){
130. printf("%d\n",odt.qy1(a,b));
131. }
132. **else** {
133. printf("%d\n",odt.qy2(a,b));
134. }
136. }
137. **return** 0;
138. }

# 几何数学

## 基本函数（已封印）

## 三角形求外心

1. Point waixin(Point a,Point b,Point c)//求三角形外心,
2. {
3. **double** a1=b.x-a.x,b1=b.y-a.y,c1=(a1\*a1+b1\*b1)/2.0;
4. **double** a2=c.x-a.x,b2=c.y-a.y,c2=(a2\*a2+b2\*b2)/2.0;
5. **double** d=a1\*b2-a2\*b1;
6. **return** Point(a.x+(c1\*b2-c2\*b1)/d,a.y+(a1\*c2-a2\*c1)/d);
7. }

## 判断是否线段与多边形规范相交

线段与多边形不规范相交的充分必要条件是线段上的每个点不在多边形内，但可以在多边形边上。

满足下面三种条件即可：

* 线段与多边形的每条边都不规范相交(可以使用判断线段是否相交的规范相交部分)
* 线段的两个端点都不在多边形内（可以使用点是否在多边形内的判断函数）
* 取线段上的数百个点，每个点都不在多边形内。

## 最小圆覆盖/最小球覆盖

**最小圆覆盖**: 求覆盖n个点的最小的圆。点顺序随机情况下，平均时间复杂度O(n)。

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **const** **double** eps=1e-8;
4. **struct** Point
5. {
6. **double** x,y;
7. };
9. Point p[100005];
11. **double** dist(Point a,Point b)
12. {
13. **return** sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));
14. }
16. /\*\*\*返回三角形的外心 \*/
17. Point circumcenter(Point a,Point b,Point C)
18. {
19. Point ret;
20. **double** a1=b.x-a.x,b1=b.y-a.y,c1=(a1\*a1+b1\*b1)/2;
21. **double** a2=C.x-a.x,b2=C.y-a.y,c2=(a2\*a2+b2\*b2)/2;
22. **double** d=a1\*b2-a2\*b1;
23. ret.x=a.x+(c1\*b2-c2\*b1)/d;
24. ret.y=a.y+(a1\*c2-a2\*c1)/d;
25. **return** ret;
26. }
28. /\*\*\*c为圆心，r为半径 平均时间复杂度O(n)\*/
29. **void** min\_cover\_circle(Point \*p,**int** n,Point &c,**double** &r)
30. {
31. random\_shuffle(p,p+n);//随机打乱
32. c=p[0]; r=0;
33. **for**(**int** i=1;i<n;i++)
34. {
35. **if**(dist(p[i],c)>r+eps)   //第一个点
36. {
37. c=p[i]; r=0;
38. **for**(**int** j=0;j<i;j++)
39. **if**(dist(p[j],c)>r+eps)  //第二个点
40. {
41. c.x=(p[i].x+p[j].x)/2;
42. c.y=(p[i].y+p[j].y)/2;
43. r=dist(p[j],c);
44. **for**(**int** k=0;k<j;k++)
45. **if**(dist(p[k],c)>r+eps)  //第三个点
46. {   //求外接圆圆心，三点必不共线
47. c=circumcenter(p[i],p[j],p[k]);
48. r=dist(p[i],c);
49. }
50. }
51. }
52. }
53. }
54. /\*
55. 输入n和n个点，输出最小覆盖圆的半径和圆心坐标.
56. \*/
57. **int** main()
58. {
59. **int** n;
60. Point c;
61. **double** r;
62. **while**(~scanf("%d",&n)&&n)
63. {
64. **for**(**int** i=0;i<n;i++)
65. scanf("%lf%lf",&p[i].x,&p[i].y);
66. min\_cover\_circle(p,n,c,r);
67. printf("%.10f\n",r);
68. printf("%.10f %.10f\n",c.x,c.y);
69. }
70. **return** 0;
71. }

**最小球覆盖**：

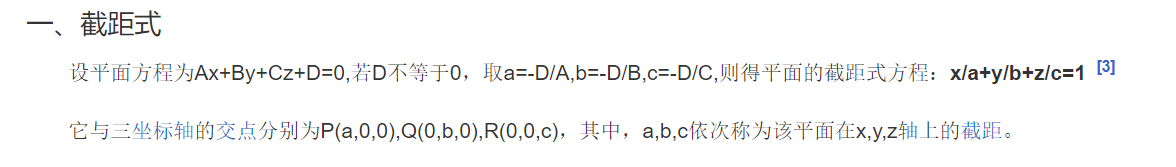
三分套三分套三分，时间复杂度

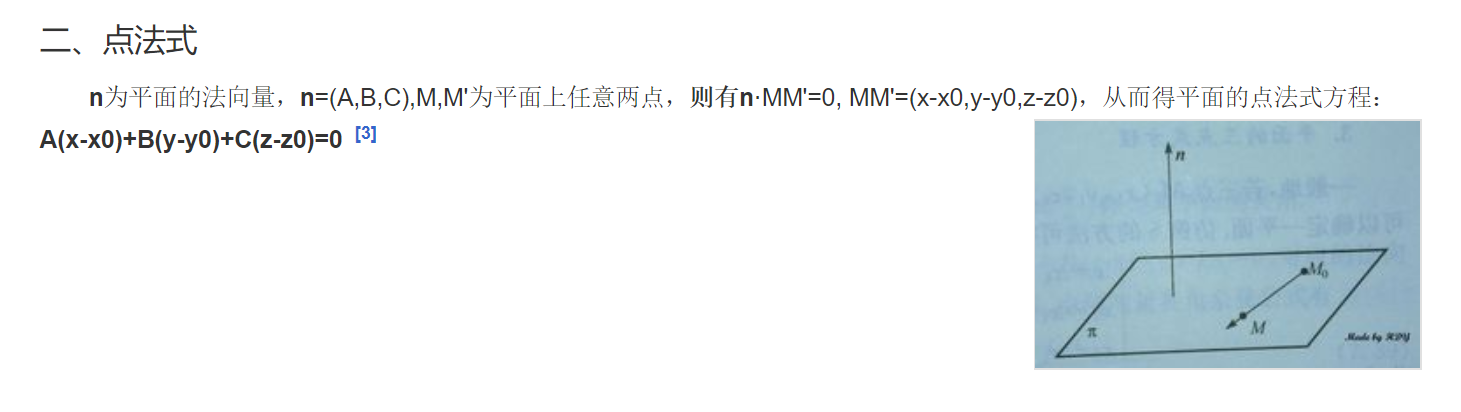
1. /\*
2. 三分套三分套三分求球的最小覆盖
3. \*/
4. #include<iostream>
5. #include<cstdio>
6. #include<cstring>
7. #include<algorithm>
8. #include<cmath>
9. **using** **namespace** std;
10. **typedef** **long** **long** ll;
11. **const** **double** eps=1e-3;
12. **int** n;
13. **double** x[105],y[105],z[105];
14. **double** dis3(**double** a,**double** b,**double** c){
15. **double** ans=0;
16. **for**(**int** i=1;i<=n;i++){
17. ans=max(ans,(x[i]-a)\*(x[i]-a)+(y[i]-b)\*(y[i]-b)+(z[i]-c)\*(z[i]-c));
18. }
19. **return** ans;
20. }
21. **double** dis2(**double** a,**double** b){
22. **double** l=-100000;
23. **double** r=100000;
24. **double** ans=0;
25. **while**(r-l>=eps){
26. **double** rmid=(r+l)/2;
27. **double** lmid=(l+rmid)/2;
28. **if**(dis3(a,b,lmid)<dis3(a,b,rmid)){
29. r=rmid;
30. }
31. **else** l=lmid;
32. }
33. **return** dis3(a,b,l);
34. }
35. **double** dis(**double** a){
36. **double** l=-100000;
37. **double** r=100000;
38. **while**(r-l>=eps){
39. **double** rmid=(r+l)/2;
40. **double** lmid=(l+rmid)/2;
41. **if**(dis2(a,lmid)<dis2(a,rmid)){
42. r=rmid;
43. }
44. **else** l=lmid;
45. }
46. **return** dis2(a,l);
47. }
48. /\*
49. 输入n个n个点的坐标,求可以覆盖这n个点的最小球的半径
50. \*/
51. **int** main(){
52. // int n;
53. scanf("%d",&n);
54. **for**(**int** i=1;i<=n;i++)scanf("%lf%lf%lf",&x[i],&y[i],&z[i]);
55. **double** l=-100000;
56. **double** r=100000;
57. **while**(r-l>=eps){
58. **double** rmid=(r+l)/2;
59. **double** lmid=(l+rmid)/2;
60. **if**(dis(lmid)<dis(rmid)){
61. r=rmid;
62. }
63. **else** l=lmid;
64. }
65. printf("%.6lf\n",sqrt(dis(l)));
66. **return** 0;
67. }

**爬山算法**，时间复杂度

1. /\*
2. 爬山算法:
3. 先随便找一个圆心,
4. 然后找离该圆心最远的点,然后让圆心往最远点的方向移动.
5. 循环这个过程,且移动步长随着时间距离慢慢减少,当步长减
6. 小到足够小则停止,最后的圆心位置就是答案。
8. 依据这个算法思想可以应用到很多地方,比如说重物到一个点的拉力合力为0
9. 的点,也可以在移动过程中改进,比如:每次步长为最远点与该点向量的一个比例
10. 此时随着时间减少的就是比例
11. \*/
12. #include <bits/stdc++.h>
13. #define mset(a,b) memset(a,b,sizeof(a))
14. **using** **namespace** std;
15. **typedef** **long** **long** ll;
16. **const** **int** N=105;
17. **struct** pnode{
18. **double** x,y,z;
19. pnode(){}
20. pnode(**double** x,**double** y,**double** z):x(x),y(y),z(z){}
21. }p[N];
22. **double** getdis(**const** pnode&a,**const** pnode &b)
23. {
24. **double** dx=a.x-b.x,dy=a.y-b.y,dz=a.z-b.z;
25. **return** dx\*dx+dy\*dy+dz\*dz;
26. }
27. /\*
28. \*/
29. **int** main()
30. {
31. **int** n;
32. scanf("%d",&n);
33. **double** ans=1e10;
34. **for**(**int** i=1;i<=n;++i) scanf("%lf%lf%lf",&p[i].x,&p[i].y,&p[i].z);
35. **double** x=0,y=0,z=0;//先随便找个位置当圆心
36. **double** L=2e5,desc=0.99;
37. **while**(L>=1e-7)
38. {
39. //找到最远的点,然后移动部分距离
40. pnode far;
41. **double** d=0;
42. **for**(**int** i=1;i<=n;++i)//找最远的点
43. {
44. **double** m=getdis(pnode(x,y,z),p[i]);
45. **if**(m>d)
46. {
47. d=m;
48. far=p[i];
49. }
50. }
51. **double** dx=far.x-x,dy=far.y-y,dz=far.z-z;
52. **double** m=sqrt(dx\*dx+dy\*dy+dz\*dz);
53. ans=min(ans,m);
54. x+=L\*dx/m;//x,y,z 分别往那个方向移动步长距离
55. y+=L\*dy/m;
56. z+=L\*dz/m;
57. L\*=desc;//步长随着时间推移减少
58. }
59. printf("%.10f\n",ans);
60. **return** 0;
61. }

## 平面方程表示





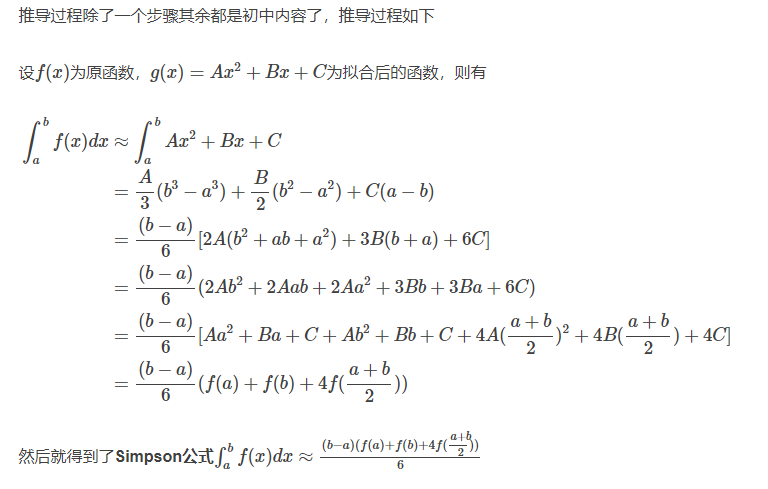
## 自适应Simpson积分法

**Simpson积分法**

公式表示：

原理：用二次函数去逼近函数f(x)，**它要求f(x)可以表示为二次函数的形式（即 的形式），这样只需要三个点的值就可以知道函数在区间的定积分。**

公式证明：



代码表示：

1. **double** f(**double** x)
2. {
3. **return** x\*x+x;//this is a example
4. }
5. **double** simpson(**double** a,**double** b)//要求a<b
6. {
7. **double** c=(a+b)/2.0;
8. **return** (b-a)/6.0\*(f(a)+4\*f(c)+f(b));
9. }

**自适应Simpson积分法:**

对于原函数不能表达为的我们该怎么计算呢？，

**如果函数 连续**，且可以分割为数个**区间的函数曲线为抛物线**的形式，那么我们可以使用自适应Simpson积分法进行计算。

区间分少了精度可能不够，区间分多了又会太慢，自动控制区间分割的大小，就是自适应辛普森法的好处。

实现很简单，就是二分递归的过程，如果满足了精度需要，则可以终止递归，而这就是自适应辛普森法能够自动控制区间分割大小的手段。

1. **double** asr(**double** l,**double** r,**double** eps,**double** ans)
2. {
3. **double** m=(l+r)/2;
4. **double** ls=simpson(l,m),rs=simpson(m,r);
5. **if**(fabs(ls+rs-ans)<=15\*eps) **return** ls+rs+(ls+rs-ans)/15;
6. **return**  asr(l,m,eps/2,ls)+asr(m,r,eps/2,rs);
7. }
8. **double** asr(**double** l,**double** r,**double** eps)
9. {
10. **return** asr(l,r,eps,simpson(l,r));
11. }

# 动态规划经典例题

## 区间dp

>注：这道题并不是入门的，但也差不多

[HihoCoder - 1636](https://vjudge.net/problem/1212860/origin)：

**题意**：

给出n个石子和石子的重量，每次可以选择L到R个石子合并成一个，花费为合并的石子的重量，求合并成一堆的最小花费，不能合成一堆输出0。

**思路**：

这题感觉很有意思，我们用表示状态，但是这个状态的含义有些丰富（

当时，表示 到 的石子合并成一堆所需的最小花费。

当时，表示到 的石子划分成 堆的最小花费，注意这里的划分是还没有合并，只是划分。

**转移方程**：

。注意这里的表示i 到 的总重量,这个状态转移表示的是一次合并操作，

当时，，这个状态转移表示的是划分操作。

为了使DP无后效性，我们区间长度从小到大开始DP。

**代码**：

1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **const** **int** N=105;
6. **const** **int** inf=0x3f3f3f3f;
7. **int** a[N],sum[N];
8. **int** dp[N][N][N];
9. **int** main()
10. {
11. **int** n,L,R;
12. **while**(~scanf("%d%d%d",&n,&L,&R))
13. {
14. **for**(**int** i=1;i<=n;++i) scanf("%d",a+i);
15. **for**(**int** i=1; i<=n; ++i)
16. **for**(**int** j=1; j<=n; ++j)
17. **for**(**int** k=1; k<=R; ++k) dp[i][j][k]=inf;
18. **for**(**int** i=1; i<=n; ++i) sum[i]=sum[i-1]+a[i];
19. **for**(**int** i=1; i<=n; ++i) dp[i][i][1]=0;
20. **for**(**int** ls=2; ls<=n; ++ls)
21. {
22. **for**(**int** l=1; l+ls-1 <= n; ++l)
23. {
24. **int** r=l+ls-1;
25. //先处理k\in[2,R]的情况
26. **for**(**int** k=2; k<=R; ++k) //dp    r-c>=k-1  c<=r-k+1
27. {
28. **for**(**int** c=l; c<=r-k+1; ++c) //dp[l][c][1] +dp[c+1][r][k-1]
29. dp[l][r][k]=min(dp[l][r][k],dp[l][c][1]+dp[c+1][r][k-1]);
30. }
31. //处理k=1的情况
32. **for**(**int** k=L; k<=R; ++k)
33. dp[l][r][1]=min(dp[l][r][1],dp[l][r][k]+sum[r]-sum[l-1]);
34. }
35. }
36. **if**(dp[1][n][1]==inf)
37. printf("0\n");
38. **else**
39. printf("%d\n",dp[1][n][1]);
40. }
41. **return** 0;
42. }

## 数位DP

入门题：hdu2089 不要62

**题意**：不吉利的数字为所有含有4或62的号码，对于每个整数对，输出之间不含有不吉利数字的统计个数，

代码：

1. #include<cstdio>
2. #include<algorithm>
3. #define mset(a,b)   memset(a,b,sizeof(a))
4. **using** **namespace** std;
5. **typedef** unsigned **long** **long** ull;
6. **typedef** **long** **long** ll;
7. **const** **int** maxn=1e5+10;
8. **const** **int** branch=26;
9. **const** **int** inf=0x3f;
10. **const** **int** MOD=1e5+7;
11. **int** n,m;
12. ll dp[20][2];//dp[k][0/1]k位，前位有无限制6的符合条件的种类数
13. **int** num[20];
14. ll dfs(**int** pos,**bool** limit,**bool** istop)//前面是否限制位6，是否到顶端
15. {
16. /\*如果没有到顶端即前一位没有到最大num[pos+1],那么这个过程就相当在求dp[pos][limit]
17. 否则就会枚举0~num[pos]，dp求所有值，
18. 注释：limit=1则表示前一位为6，istop=1表示该位已经到顶端
19. \*/
20. **if**(!pos)
21. **return** 1;
22. **if**(!istop&&dp[pos][limit]!=-1)
23. **return** dp[pos][limit];
24. ll res=0ll;
25. **int** rmax=istop?num[pos]:9;
26. **for**(**int** i=0;i<=rmax;++i)
27. {
28. **if**(i==4||(limit&&i==2))
29. **continue**;
30. res+=dfs(pos-1,i==6,istop&&i==rmax);
31. }
32. **if**(!istop)
33. dp[pos][limit]=res;
34. **return** res;
35. }
36. ll solve(ll val)
37. {
38. **if**(val==0)
39. **return** 1;
40. **int** pos=0;
41. **while**(val)//分解数位，位置从1~pos
42. {
43. num[++pos]=val%10;
44. val/=10;
45. }
46. **return** dfs(pos,0,1);
47. }
48. **int** main()
49. {
50. mset(dp,-1);
51. ll a,b;
52. **while**(~scanf("%lld %lld",&a,&b))
53. {
54. **if**(a==0&&b==0)
55. **break**;
56. printf("%lld\n",solve(b)-solve(a-1));
57. }
58. **return** 0;
59. }

## 状压DP

Zoj 3375 Imperishable Night(类似曼哈顿路径，没要求回到起始点)

**题意**：

现在有n个顶点，有m个边，每个边有相应的权值，要求每个顶点最多经过两次，要求经过n个顶点的权值和最小。如果不能经过n个点，则输出-1

**思路**：

每个顶点可以最多经过两次，个顶点，用3进制数字代表对应的顶点有没有经过，代表没有经过，代表经过一次，代表此状态中该顶点经过次，那么所有情况有种，我们让该三进制数的最低位到最高位分别代表顶点 ，那么我们要求的就是进制数中个位中状态都非的形成这种状态的最小值。

因为每一个通路可以用一个状态和一个端点表示

所以用代表状态state，且该路径的端点为。这种情况所花费的最小值。

那么求 只需枚举连接端点的另一个端点即可。

初始化： ，其他都是，即只有端点v的状态的最小花费为

又因为每种状态的值至于前面的状态(状态值比自己小)有关，所以循环 三进制的值从小到大遍历所有状态即可。最后找出满足条件的最小值即可

**代码**：

1. #define mset(a,b)   memset(a,b,sizeof(a))
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **int** maxn=11;
5. **const** **int** inf=0x3f3f3f3f;
6. **int** mm[maxn][maxn];//存放地图
7. **int** dp[60000][maxn];
8. **int** a[maxn];//存放状态
9. **int** n,m;
10. **int** qpow(**int** a,**int** b){**int** ans=1;
11. **while**(b)
12. {
13. **if**(b&1) ans=ans\*a;
14. a=a\*a; b>>=1;
15. }
16. **return** ans;
17. }
18. **int** Add(**int** state,**int** v)//从state中增加v顶点的一次
19. {
20. **return** state+qpow(3,v-1);
21. }
22. **int** Subtract(**int** state,**int** v)//从state中减去v顶点的一次数
23. {
24. **return** state-qpow(3,v-1);
25. }
26. **int** getcnt(**int** state,**int** v)//得到state中 端点v的出现次数
27. {
28. **return** (state/qpow(3,v-1))%3;
29. }
30. **int** judge(**int** state)//判断state是否符合条件
31. {
32. **for**(**int** i=0;i<n;++i)
33. {
34. **if**(state%3==0)
35. **return** 0;
36. state/=3;
37. }
38. **return** 1;
39. }
40. **int** main()
41. {
42. **while**(~scanf("%d %d",&n,&m))
43. {
44. mset(mm,inf);
45. **int** u,v,w;
46. **for**(**int** i=0;i<m;++i)
47. {
48. scanf("%d %d %d",&u,&v,&w);
49. **if**(mm[u][v]>w)
50. mm[u][v]=mm[v][u]=w;
51. }
52. /\*开始状态转移 dp[state][r]，代表这个状态时statte，且改状态末尾的顶点是r,所花费的最小代价\*/
53. mset(dp,inf);
54. **for**(**int** i=1;i<=n;++i)
55. {
56. dp[Add(0,i)][i]=0;
57. }//初始化 刚开始的状态
58. **int** tot=qpow(3,n);
59. **for**(**int** i=0;i<tot;++i)//枚举要求的状态
60. {
61. **for**(**int** j=1;j<=n;++j)//枚举now state
62. {
63. **if**(!getcnt(i,j))// out can't rule
64. **continue**;
65. **int** sss=Subtract(i,j);// out the state
66. **for**(**int** s=1;s<=n;++s)//enum last v
67. {
68. **if**(!getcnt(sss,s)||mm[s][j]==inf||dp[sss][s]==inf)
69. **continue**;
70. dp[i][j]=min(dp[i][j],dp[sss][s]+mm[s][j]);
71. }
72. }
73. }
74. **int** minn=inf;
75. **for**(**int** i=(qpow(3,n)-1)/2;i<tot;++i)//(qpow(3,n)-1)/2是n位全1的状态值
76. **if**(judge(i))
77. **for**(**int** j=1;j<=n;++j)
78. minn=min(minn,dp[i][j]);
79. **if**(minn==inf)
80. printf("-1\n");
81. **else**
82. printf("%d\n",minn);
83. }
84. **return** 0;
85. }

## 树形DP

例题，HDU2196 (树形DP+换根)

题意：给出一个无根树，每条边的权值为两点距离，求距每个点最远点的距离

思路：先定根，记录每个节点到子树上的最远和次远距离和位置，然后换根即可，对于u-fa，从fa换根到u需fa的最远和次远距离和位置（同时要更新以u为根的最远和次远信息）。

1. /\*
2. 换根法dp:
3. 先把无根树化为有根树
4. 第一遍dfs,对于顶点u,求u的子树到u的最大和次大距离
5. 第二遍dfs,将树化为顶点u为根,求u的子树到u的最大距离和次大距离
6. 对于u的最大距离就是u的答案res[u],而u的次大距离是用来儿子换根的时候进行状态转移
7. \*/
8. #include<bits/stdc++.h>
9. #define mset(a,b) memset(a,b,sizeof(a))
10. **using** **namespace** std;
11. **typedef** **long** **long** ll;
12. **typedef** pair<**int**,**int**> P;
13. **const** **int** inf=0x3f3f3f3f;
14. **const** ll N=1e4+10;
15. **const** ll mod=1e9+7;
16. vector<P> g[N];
17. **int** dp[N][2];//u的最大值与次大值
18. **int** pos[N][2];//u的最大值与次大值的位置
19. **int** res[N];
20. **void** dfs1(**int** u,**int** fa)
21. {
22. **for**(P &p:g[u])
23. {
24. **int** v=p.first,w=p.second;
25. **if**(v==fa) **continue**;
26. dfs1(v,u);
27. **if**(dp[v][0]+w>dp[u][0]){
28. dp[u][1]=dp[u][0];
29. pos[u][1]=pos[u][0];
30. dp[u][0]=dp[v][0]+w;
31. pos[u][0]=v;
32. }
33. **else** **if**(dp[v][0]+w>dp[u][1]){
34. dp[u][1]=dp[v][0]+w;
35. pos[u][1]=v;
36. }
37. }
38. }
39. **void** dfs2(**int** u,**int** fa,**int** w)
40. {
41. **int** other=0;
42. **if**(fa!=u)//再次换根为u，更新最大和次大
43. {
44. **if**(pos[fa][0]!=u)//other代表从父亲到该根的最大与次大
45. other=dp[fa][0]+w;
46. **else**
47. other=dp[fa][1]+w;
48. **if**(other>dp[u][0]){
49. pos[u][1]=pos[u][0];
50. dp[u][1]=dp[u][0];
51. pos[u][0]=fa;
52. dp[u][0]=other;
53. }
54. **else** **if**(other>dp[u][1])
55. {
56. pos[u][1]=fa;
57. dp[u][1]=other;
58. }
59. }
60. res[u]=dp[u][0];
61. **for**(P &p:g[u])
62. {
63. **int** v=p.first;
64. **if**(v==fa) **continue**;
65. dfs2(v,u,p.second);
66. }
67. }
68. **int** main()
69. {
70. ios::sync\_with\_stdio(**false**);cin.tie(0);
71. **int** n;
72. **while**(cin>>n)
73. {
74. **for**(**int** i=1;i<=n;++i) g[i].clear();
75. **for**(**int** i=2;i<=n;++i){
76. **int** w,v;
77. cin>>v>>w;
78. g[i].push\_back({v,w});
79. g[v].push\_back({i,w});
80. }
81. mset(dp,0);
82. mset(pos,0);mset(res,0);
83. dfs1(1,1);
84. dfs2(1,1,0);
85. **for**(**int** i=1;i<=n;++i){
86. cout<<res[i]<<endl;
87. }
88. }
89. }

树形依赖背包：HDU1561

题意：

在一个地图上，有N座城堡，每座城堡都有一定的宝物，在每次游戏中ACboy允许攻克M个城堡并获得里面的宝物。但由于地理位置原因，有些城堡不能直接攻克，要攻克这些城堡必须先攻克其他某一个特定的城堡。你能帮ACboy算出要获得尽量多的宝物应该攻克哪M个城堡吗？

**Input**

每个测试实例首先包括2个整数，N,M.(1 <= M <= N <= 200);在接下来的N行里，每行包括2个整数，a,b. 在第 i 行，a 代表要攻克第 i 个城堡必须先攻克第 a 个城堡，如果 a = 0 则代表可以直接攻克第 i 个城堡。b 代表第 i 个城堡的宝物数量, b >= 0。当N = 0, M = 0输入结束。

**Output**

对于每个测试实例，输出一个整数，代表ACboy攻克M个城堡所获得的最多宝物的数量。

思路：

1. /\*
2. 对于每个节点u,要么有唯一的父亲fa,要么没有父亲,自形一颗树
3. 所以按照题目给的要求该图是一个森林.
4. 我们考虑将森林中的每一颗树的根连接到超级根root.那么形成的图是一个以超级根
5. 为根的树,题目也转化为从该树上取m+1个节点,使得价值最大,取u节点必须先取父亲节点.
6. 这就是树形依赖背包的问题了
7. \*/
8. #include<bits/stdc++.h>
9. #define mset(a,b) memset(a,b,sizeof(a))
10. **using** **namespace** std;
11. **typedef** **long** **long** ll;
12. **typedef** pair<**int**,**int**> P;
13. **const** **int** inf=0x3f3f3f3f;
14. **const** **int** N=1e4+10;
15. **const** **int** mod=1e9+7;
16. **int** w[305];
17. vector<**int**> g[305];
18. **int** dp[305][305];
19. **int** n,m;
20. **void** dfs(**int** u)
21. {
22. **for**(**int** v:g[u])
23. {
24. dfs(v);
25. /\*因为要求该儿子中要么只取一个，要么不取, 所以我们必须先枚举背包容量从大到小,其次枚举儿子的容量大小\*/
26. **for**(**int** s=m;s>=1;--s)//枚举背包容量从大到小
27. {
28. **for**(**int** a=1;a<=s;++a){//其次枚举儿子容量
29. dp[u][s]=max(dp[u][s],dp[u][s-a]+dp[v][a]);
30. }
31. }
32. }
33. dp[u][0]=0;
34. **for**(**int** i=m;i>=1;--i)
35. dp[u][i]=dp[u][i-1]+w[u];
37. }
38. **int** solve()
39. {
40. dfs(0);
41. **return** dp[0][m];
42. }
43. **int** main()
44. {
45. ios::sync\_with\_stdio(**false**);cin.tie(0);
46. **while**(cin>>n>>m,n)
47. {
48. **for**(**int** i=0;i<=n;++i) g[i].clear();
49. **for**(**int** i=1;i<=n;++i){
50. **int** fa,ww;
51. cin>>fa>>ww;
52. w[i]=ww;
53. g[fa].push\_back(i);
54. }
55. ++m;//0这个节点额外的且是必须取的,所以m要+1
56. mset(dp,0);
57. cout<<solve()<<endl;
58. }
59. **return** 0;
60. }

## 期望DP

**第一个简单问题：**

有n个门，每个门有一个值，这个值的绝对值代表你开门的时间，正数则代表打开这个门你就能出去，负数代表进入这个门你将回到现在的位置，每次你等概率的选择其中一个们打开，求出去所用时间的期望。并以分数形式输出。

解：

举个例子 有两个门一个门分别是2，-3，假设选择出去的时间的期望是E，那么

得出

**第二个例题：**

题目：LOJ 1265-Snakes and Ladders（期望DP+高斯消元）

**题意**：

有个方格，其中从左到右编号依此为 到 ，现在我们有一个6个面的骰子，我们每次掷骰子，得到多少点数当前就会跳几个格子，比如我们现在在号格子，点数为 ，那么我们就会直接跳到号点。

除了掷骰子外，格子上有  个传送关系，表示为，且 ，传送关系的作用是，如果你到了  号点，那么你必须会被传送到  号点。

现在我们的初始位置是 ，且给出个传送关系，问到达位置的掷骰子的期望次数是多少？需要注意的是，如果掷骰子后将要跳跃的位置超过时，需要重新再掷骰子，直到将要跳跃的位置不超过时停止。

**思路**：

众所周知，期望DP倒着求（容易求）。

我们用表示 号格子到终点掷骰子的期望次数。

* 如果 号点有传送且传送到：
* 否则：令，

然后化简后可以得出的表达式。

因为 可能小于 ，所以我们不能单纯的从后往前。我们考虑将设为第 个未知数，则可以建立100个有100个未知数的一次方程，然后高斯消元进行求解即可。

注：高斯消元解期望DP的思想经常用。时间复杂度O(n^3)

代码：

1. #include<bits/stdc++.h>
2. **using** **namespace** std;
3. **typedef** **long** **long** ll;
4. **const** **int** N=1e5+10;
5. **const** **double** eps=1e-7;
6. **double** g[105][105],ans[105];
7. **bool** Gauss(**int** n)
8. {
9. //A\*x=B
10. **for**(**int** i=1;i<=n;++i)
11. {
12. **int** r=i;
13. **for**(**int** j=i+1;j<=n;++j) **if**(fabs(g[r][i])<fabs(g[r][j])) r=i;
14. **if**(fabs(g[r][i]) < eps) **return** **false**;
15. swap(g[i],g[r]);
16. **double** div=g[r][i];
17. **for**(**int** j=i;j<=n+1;++j) g[i][j]/=div;
18. **for**(**int** j=i+1;j<=n;++j)
19. {
20. div=g[j][i];
21. **for**(**int** p=i;p<=n+1;++p) g[j][p]-=div\*g[i][p];
22. }
23. }
24. ans[n]=g[n][n+1];
25. **for**(**int** i=n-1;i>=1;--i)
26. {
27. ans[i]=g[i][n+1];
28. **for**(**int** j=i+1;j<=n;++j) ans[i]-=g[i][j]\*ans[j];
29. }
30. **return** **true**;
31. }
32. **int** tp[105];//传送关系
33. **int** main()
34. {
35. **int** t,cas=0;
36. scanf("%d",&t);
37. **while**(t--)
38. {
39. **int** m,n=100;
40. scanf("%d",&m);
41. fill(tp+1,tp+n+1,0);
42. **for**(**int** i=1;i<=m;++i)
43. {
44. **int** a,b;
45. scanf("%d%d",&a,&b);
46. tp[a]=b;//a可以传送到to[a]
47. }
48. **for**(**int** i=1;i<=n;++i)
49. **for**(**int** j=1;j<=n+1;++j) g[i][j]=0.0;//初始化高斯消元矩阵
50. g[n][n]=1.0;//终点的方程
51. g[n][n+1]=0.0;
52. **for**(**int** i=1;i < n;++i)
53. {
54. **if**(tp[i])
55. {
56. g[i][i]=1;
57. g[i][tp[i]]=-1;
58. g[i][n+1]=0.0;
59. }
60. **else**
61. {
62. **int** k=min(6,100-i);
63. g[i][i]=1;
64. **for**(**int** j=1;j<=k;++j)
65. g[i][i+j]=-1.0/k;
66. g[i][n+1]=6.0/k;
67. }
68. }
69. Gauss(n);
70. printf("Case %d: %.9f\n",++cas,ans[1]);
71. }
72. **return** 0;
73. }

**第三个例题：**

题目链接：[LightOJ - 1287](https://vjudge.net/problem/26997/origin)

题意：

​ 先给一个n个点m条无向边的图，每条边都有一个权值，顶点编号从0开始，刚开始自己站在0号点，现在要躲避警察的追踪。走的路线有如下限制：

1. 离开一个顶点之后就不能回到该顶点
2. 如果站在某个顶点u，**剩下的未走的邻接顶点不存在 E J 顶点就停止并被警察抓到**，E J 顶点指如果到该顶点后，满足条件1的情况下可以走完所有未走完的顶点。

如果存在E J顶点，则会进行如下**随机选择**，如果有k个E J顶点，那么有1/(k+1)的概率选择先停留原地5分钟（隐藏）或走剩下的E J 顶点。

如果走u到v这条边，则花费的时间为u到v这条边的权值。

求从 号顶点开始出发**躲避警察的时间的期望值**。

数据范围：

思路：

​ 我们用代表经过的顶点状态是 ,现在处于节点 的剩下的期望躲避时间，所以我们要求的就是

期间我一直在想怎么判断剩下的邻接顶点是E J顶点，期间想了各种方法，但都太麻烦且时间复杂度大。

实际上对于状态，我们用代表这个状态下从 开始能否遍历完其他未遍历完的顶点，判断状态中是否为为E J顶点只需判断可选的顶点中是否存在E J 顶点即可

形式化的，我们现在来求，假设有边，且状态中未经过顶点，那么如果，则

对于状态转移方程

假设状态可选的顶点有  个

* 如果 :

对于状态下u的所有邻接E J顶点，且边权，那么

* 否则：

因为方程中  是递增的，所以我们可以按照  从大到小遍历，其次遍历$ u$ 求和即可，特殊的满状态下

## 插头DP(未加入)

## 轮廓线DP(未加入)

## 背包问题之退背包

退背包就是从可选物品中删除其中一个物品，问满足所取总价值为 的方案数。

像普通背包一样，退背包先普通dp以下，然后退去所选物品。

对于**01背包**，假设为未退背包前满足所取总价值为 的方案数。 为退去第个物品后满足所取总价值为的方案数，那么 dp方程为

* 当时，
* 当时，

代码表示：

**for**(**int** i=w[x];i<=m;++i) dp[i]-=dp[i-w[x]];

对于**多重背包**，假设为未退背包前满足所取总价值为 的方案数。 为退去第个物品后满足所取总价值为的方案数，那么 dp方程为

* 当时，
* 当时，

代码表示：

**for**(**int** i=m;i>=w[x];--i) dp[i]-=dp[i-w[x]];

不难看出，其实这就是普通背包DP过程的逆过程。

相关题目：

**The Preliminary Contest for ICPC Asia Shanghai 2019 J. Stone game** :[传送门](https://nanti.jisuanke.com/t/41420)

# 杂项

## 归并排序求逆序对

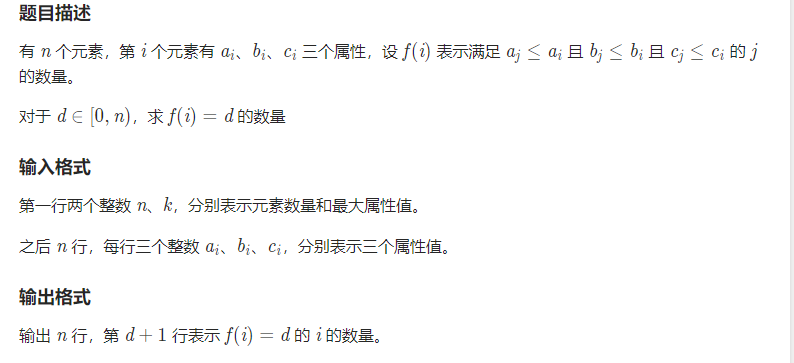
1. **typedef** **long** **long** ll;
2. **const** **int** maxn=5e5+100;
3. //marr中间储存的数组
4. //sum:记录逆序对的个数
5. //输入：arr[]
6. //结果：得到arr[]逆序对的个数
7. **int** arr[maxn],marr[maxn];
8. ll sum;
9. **void** mergeSort(**int**\* arr,**int** l,**int** r) //左闭右闭
10. {
11. **if**(l==r)
12. **return** ;
13. **int** mid=(l+r)/2;
14. mergeSort(arr,l,mid);
15. mergeSort(arr,mid+1,r);
16. **int** tot=0;
17. **int** ll=l,rr=mid+1;//开始需要比较的位置
18. **while**(ll<=mid&&rr<=r) //next get date postion
19. {
20. **if**(arr[rr]<arr[ll])//逆序了，加上左边大于它的个数
21. {
22. sum+=mid-ll+1;
23. marr[tot++]=arr[rr++];
24. }
25. **else**
26. {
27. marr[tot++]=arr[ll++];
28. }
29. }
30. **for**(**int** i=ll; i<=mid; ++i)
31. marr[tot++]=arr[i];
32. **for**(**int** i=rr; i<=r; ++i)
33. marr[tot++]=arr[i];
34. **for**(**int** i=0; i<r-l+1; ++i)
35. arr[l+i]=marr[i];
36. **return** ;
37. }
38. **int** main()
39. {
40. **int** n;
41. scanf("%d",&n);
42. **for**(**int** i=0; i<n; ++i)    scanf("%d",arr+i);;
43. sum=0;
44. mergeSort(arr,0,n-1);
45. cout<<sum<<endl;
46. **return** 0;
47. }

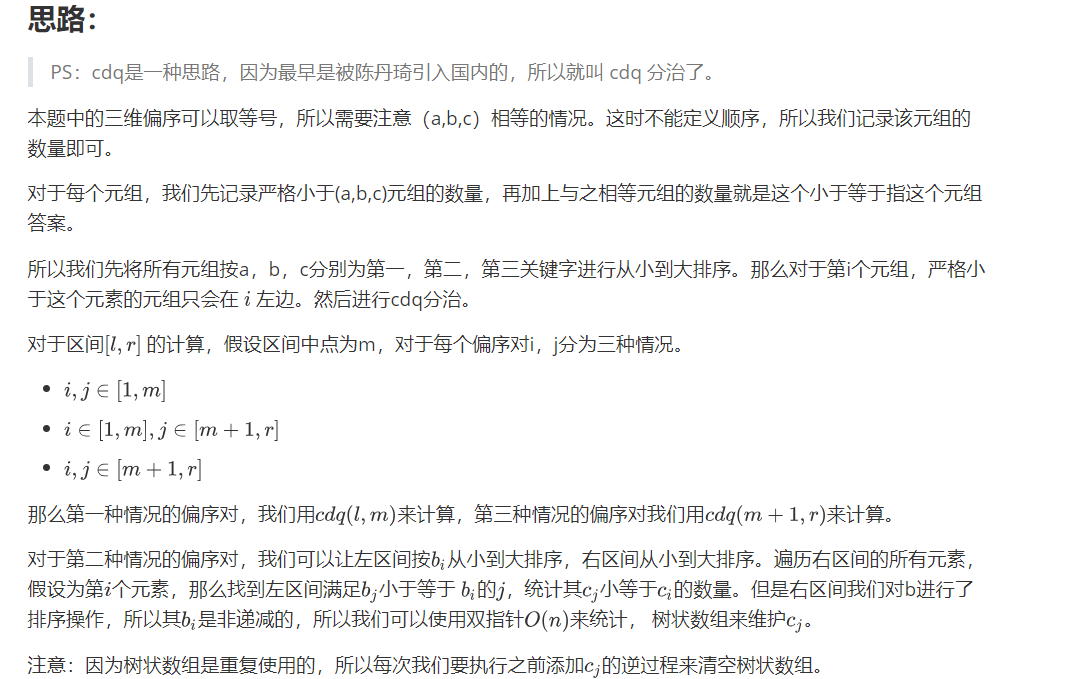
## cdq分治解三维偏序

1. cdq分治解决和点对有关的问题
2. cdq分治优化1D/2D动态规划的转移
3. 通过cdq分治，将一些动态问题转化为静态问题

分治到现在自己还并没有多大的使用，但是这种做法和思想却非常重要

**经典例题：三维偏序**





1. #include<bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **typedef** pair<**int**,**int**> P;
6. **const** **int** N=1e5+20;
7. **int** MXN;
8. **int** bt[200005],f[N];
9. **int** lowbit(**int** k)
10. {
11. **return** k&-k;
12. }
13. **void** modify(**int** k,**int** cnt)
14. {
15. **while**(k<=MXN)
16. {
17. bt[k]+=cnt;
18. k+=lowbit(k);
19. }
20. }
21. **int** getpre(**int** k)
22. {
23. **int** sum=0;
24. **while**(k)
25. {
26. sum+=bt[k];
27. k-=lowbit(k);
28. }
29. **return** sum;
30. }
31. **struct** node{
32. **int** a,b,c,cnt,sum;
33. }t[N];//用来保证 a不同
34. **bool** cmp(**const** node &a,**const** node &b)
35. {
36. **if**(a.a!=b.a) **return** a.a<b.a;
37. **if**(a.b!=b.b) **return** a.b<b.b;
38. **return** a.c<b.c;
39. }
40. **bool** cmpb(**const** node &a,**const** node &b)
41. {
42. **return** a.b<b.b;
43. }
44. **void** cdq(**int** l,**int** r)
45. {
46. **if**(l==r) **return** ;
47. **int** m=(l+r)>>1;
48. cdq(l,m);
49. cdq(m+1,r);
50. sort(t+l,t+m+1,cmpb);
51. sort(t+m+1,t+r+1,cmpb);
52. **int** p=l-1;
53. **for**(**int** i=m+1;i<=r;++i){
54. //把所有<=t[i].b的元素全加
55. **while**(p+1<=m&&t[p+1].b<=t[i].b){
56. ++p;
57. modify(t[p].c,t[p].cnt);
58. }
59. t[i].sum+=getpre(t[i].c);
60. }
61. **for**(**int** i=l;i<=p;++i) modify(t[i].c,-t[i].cnt);
62. }
63. **int** main()
64. {
65. ios::sync\_with\_stdio(**false**);cin.tie(0);
66. **int** n;
67. cin>>n>>MXN;
68. **for**(**int** i=1;i<=n;++i)
69. {
70. **int** a,b,c;
71. cin>>t[i].a>>t[i].b>>t[i].c;
72. t[i].cnt=1;
73. t[i].sum=0;
74. }
75. **int** last=0;
76. sort(t+1,t+n+1,cmp);
77. **for**(**int** i=1;i<=n;++i)
78. {
79. **if**(!last) {
80. t[++last]=t[i];
81. }
82. **else** **if**( t[last].a==t[i].a && t[last].b==t[i].b &&t[last].c==t[i].c )
83. t[last].cnt++;
84. **else**
85. t[++last]=t[i];
86. }
88. cdq(1,last);
90. **for**(**int** i=1;i<=last;++i) t[i].sum+=t[i].cnt;
91. **for**(**int** i=1;i<=last;++i) f[t[i].sum]+=t[i].cnt;
92. **for**(**int** i=1;i<=n;++i) cout<<f[i]<<endl;
93. **return** 0;
94. }

有些二维平面上关于点的在线问题可以转化为三维偏序的离线问题，用cdq写很简单。

比如：矩阵加矩阵求和。

**题目描述**：

维护一个W\*W的矩阵，初始值均为S.每次操作可以增加某格子的权值,或询问某子矩阵的总权值.修改操作数M<=160000,询问数Q<=10000,W<=2000000.

第一行两个整数,S,W;其中S为矩阵初始值;W为矩阵大小

接下来每行为一下三种输入之一(不包含引号):

"1 x y a"

"2 x1 y1 x2 y2"

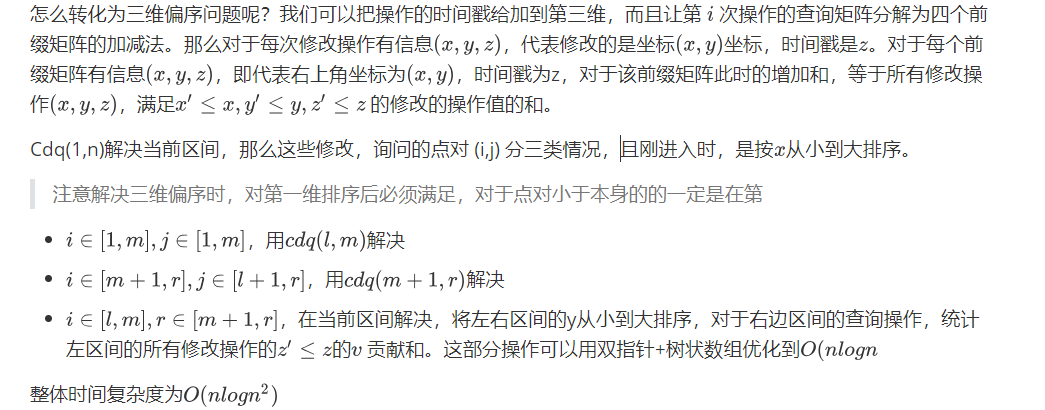
"3"

输入1:你需要把(x,y)(第x行第y列)的格子权值增加a

输入2:你需要求出以左下角为(x1,y1),右上角为(x2,y2)的矩阵内所有格子的权值和,并输出

输入3:表示输入结束

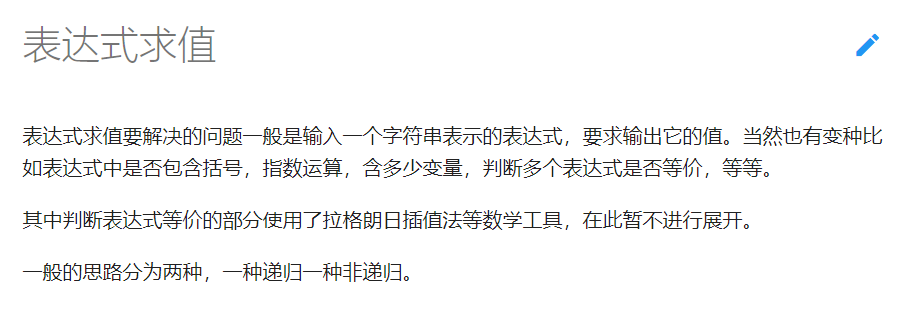
对于每个输入2,输出一行,即输入2的答案。

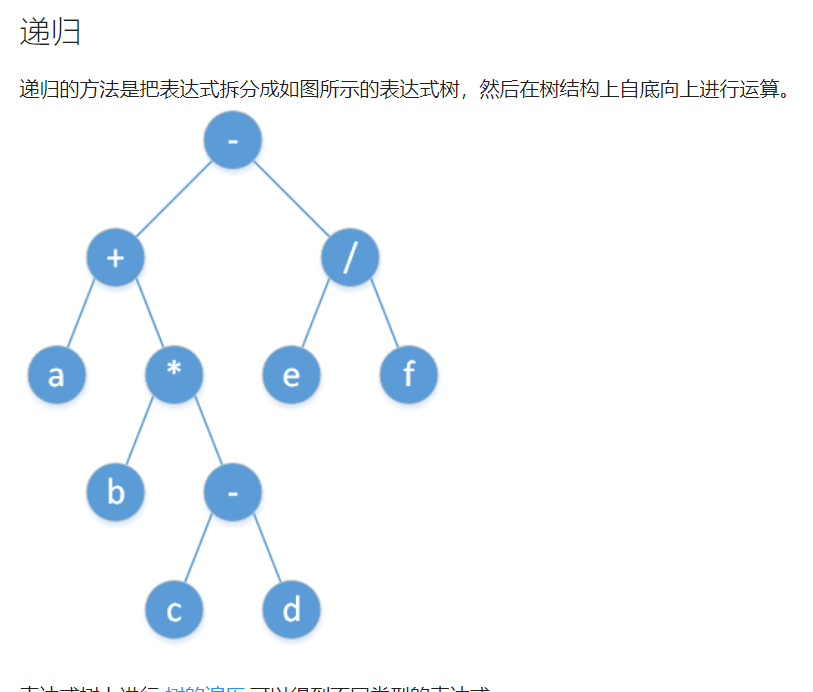


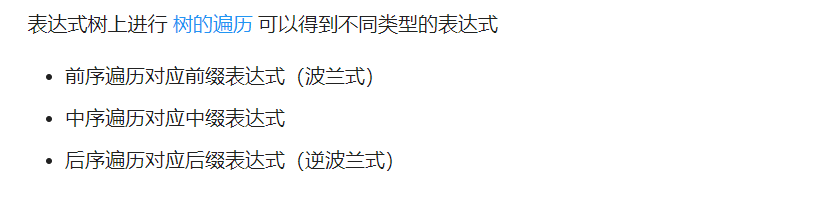
**代码**：

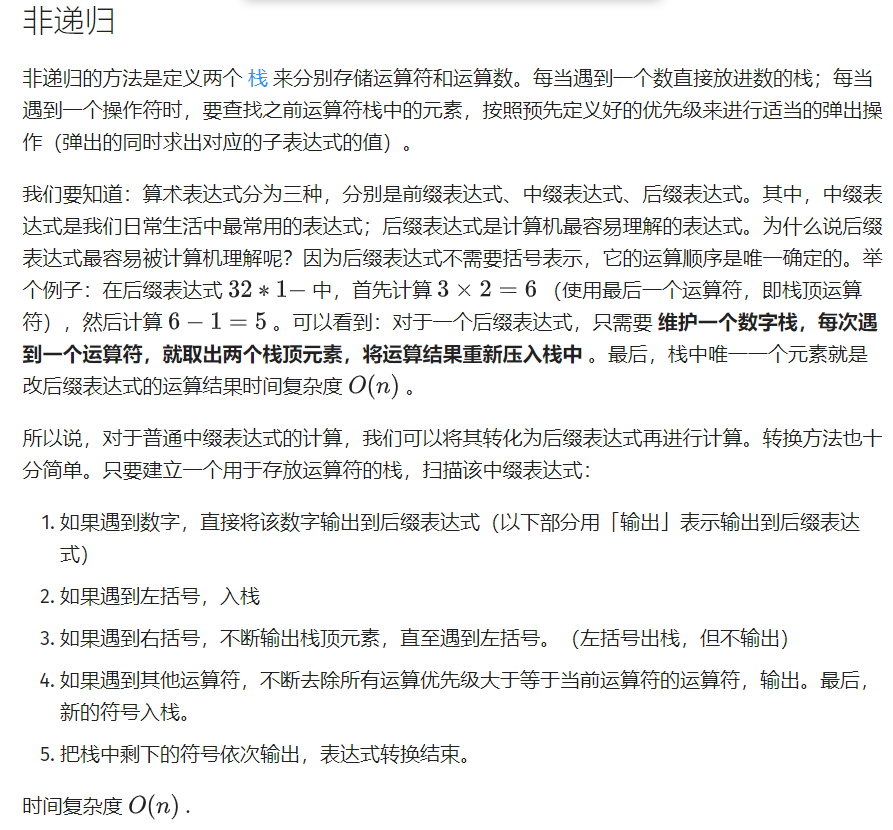
1. #include <bits/stdc++.h>
2. #define mset(a,b) memset(a,b,sizeof(a))
3. **using** **namespace** std;
4. **typedef** **long** **long** ll;
5. **typedef** pair<**int**,**int**>  P;
6. **const** **int** N=2e5+20;
7. /\*
8. 三维偏序，第一维x，第二维y，第三维时间戳d
9. 点分两种:一种修改,一种求和
10. \*/
11. **struct** BITtree
12. {
13. **int** bt[N];
14. **int** MXN;
15. **int** lowbit(**int** k)
16. {
17. **return** k&-k;
18. }
19. **void** add(**int** k,**int** v)
20. {
21. **while**(k<=MXN)
22. {
23. bt[k]+=v;
24. k+=lowbit(k);
25. }
26. }
27. **int** getpre(**int** k)
28. {
29. **int** ans=0;
30. **while**(k)
31. {
32. ans+=bt[k];
33. k-=lowbit(k);
34. }
35. **return** ans;
36. }
37. **void** en(**int** k)//销毁 k 路径上的值
38. {
39. **while**(k<=MXN)
40. {
41. bt[k]=0;
42. k+=lowbit(k);
43. }
44. }
45. }ooo;
46. **struct** pnode
47. {
48. **int** x,y,z,v;//s为时间戳,f为flag,v为值
49. **int** k;//询问则表示这是第k个询问,-1代表这是个修改
50. pnode() {}
51. pnode(**int** x,**int** y,**int** z,**int** v,**int** k):x(x),y(y),z(z),v(v),k(k) {}
52. } t[N];
53. **bool** cmpx(**const** pnode & a,**const** pnode &b)
54. {
55. **if**(a.x==b.x) **return** a.z<b.z;
56. **return** a.x<b.x;
57. }
58. **bool** cmpy(**const** pnode & a,**const** pnode &b)
59. {
60. **return** a.y<b.y;
61. }
62. **int** ans[10005];
63. **void** cdq(**int** l,**int** r)
64. {
65. **if**(l==r) **return** ;
66. **int** m=l+r>>1;
67. cdq(l,m);
68. cdq(m+1,r);
69. sort(t+l,t+m+1,cmpy);
70. sort(t+m+1,t+r+1,cmpy);
71. **int** p=l-1;
72. **for**(**int** i=m+1;i<=r;++i)
73. {
74. **while**(p<m&&t[p+1].y<=t[i].y) {
75. ++p;
76. **if**(t[p].k==-1)
77. ooo.add(t[p].z,t[p].v);
78. }
79. **if**(t[i].k!=-1)
80. {
82. **int** sum=ooo.getpre(t[i].z);
83. ans[t[i].k]+=sum\*t[i].v;
84. }
85. }
86. **for**(**int** i=l;i<=p;++i)
87. **if**(t[i].k==-1) ooo.en(t[i].z);
88. }
89. **int** main()
90. {
91. **int** n,s,tol=0;
92. scanf("%d%d",&s,&n);
93. **int** qid=0,dfn=0;
94. **while**(**true**)
95. {
96. **int** cmd,x,y,xx,yy,a;
97. ++dfn;
98. scanf("%d",&cmd);
99. **if**(cmd==1)//修改
100. {
101. scanf("%d%d%d",&x,&y,&a);
102. t[++tol]=pnode(x,y,dfn,a,-1);
103. }
104. **else** **if**(cmd==2)
105. {
106. ++qid;
107. scanf("%d%d%d%d",&x,&y,&xx,&yy);
108. t[++tol]=pnode(xx,yy,dfn,1,qid);
109. t[++tol]=pnode(xx,y-1,dfn,-1,qid);
110. t[++tol]=pnode(x-1,yy,dfn,-1,qid);
111. t[++tol]=pnode(x-1,y-1,dfn,1,qid);
112. ans[qid]=(xx-x+1)\*(yy-y+1)\*s;
113. }
114. **else**
115. **break**;
116. }
117. ooo.MXN=dfn;
118. sort(t+1,t+tol+1,cmpx);
119. cdq(1,tol);
120. **for**(**int** i=1;i<=qid;++i) printf("%d\n",ans[i]);
121. **return** 0;
122. }

## 表达式求值（留小坑）









**代码：**

1. // 下面代码摘自笔者 NOIP2005 等价表达式
2. std::string convert(**const** std::string &s) {  // 把中缀表达式转换为后缀表达式
3. std::stack<**char**> oper;
4. std::stringstream ss;
5. ss << s;
6. std::string t, tmp;
7. **while** (ss >> tmp) {
8. **if** (isdigit(tmp[0]))
9. t += tmp + " ";  // 1. 如果遇到一个数，输出该数
10. **else** **if** (tmp[0] == '(')
11. oper.push(tmp[0]);       // 2. 如果遇到左括号，把左括号入栈
12. **else** **if** (tmp[0] == ')') {  // 3. 如果遇到右括号，
13. **while** (!oper.empty() && oper.top() != '(')
14. t += std::string(1, oper.top()) + " ",
15. oper.pop();  // 不断取出栈顶并输出，直到栈顶为左括号，
16. oper.pop();        // 然后把左括号出栈
17. } **else** {             // 4. 如果遇到运算符
18. **while** (!oper.empty() && level[oper.top()] >= level[tmp[0]])
19. t += std::string(1, oper.top()) + " ",
20. oper.pop();  // 只要栈顶符号的优先级不低于新符号，就不断取出栈顶并输出
21. oper.push(tmp[0]);  // 最后把新符号进栈
22. }
23. }
24. **while** (!oper.empty()) t += std::string(1, oper.top()) + " ", oper.pop();
25. **return** t;
26. }
28. **int** calc(**const** std::string &s) {  // 计算转换好的后缀表达式
29. std::stack<**int**> num;
30. std::stringstream ss;
31. ss << s;
32. std::string t, tmp;
33. **while** (ss >> tmp) {
34. **if** (isdigit(tmp[0]))
35. num.push(stoi(tmp));
36. **else** {
37. **int** b, a;  // 取出栈顶元素，注意顺序
38. **if** (!num.empty()) b = num.top();
39. num.pop();
40. **if** (!num.empty()) a = num.top();
41. num.pop();
42. **if** (tmp[0] == '+') num.push(a + b);
43. **if** (tmp[0] == '-') num.push(a - b);
44. **if** (tmp[0] == '\*') num.push(a \* b);
45. **if** (tmp[0] == '^') num.push(qpow(a, b));
46. }
47. }
48. **return** num.top();
49. }

## 约瑟夫问题(留小坑)

约瑟夫问题由来已久，而这个问题的解法也在不断改进，只是目前仍没有一个极其高效的算法（log 以内）解决这个问题。

问题描述

n 个人标号 。逆时针站一圈，从 号开始，每一次从当前的人逆时针数 个，然后让这个人出局。问最后剩下的人是谁。

这个经典的问题由约瑟夫于公元 1 世纪提出，尽管他当时只考虑了 的情况。现在我们可以用许多高效的算法解决这个问题。

线性算法

设 表示规模分别为 的约瑟夫问题的答案。我们有如下递归式

这个也很好推。你从 开始数 个，让第 个人出局后剩下 个人，你计算出在 个人中选的答案后，再加一个相对位移 得到真正的答案。这个算法的复杂度显然是 的。

1. **int** josephus(**int** n, **int** k) {
2. **int** res = 0;
3. **for** (**int** i = 1; i <= n; ++i) res = (res + k) % i;
4. **return** res;
5. }

对数算法

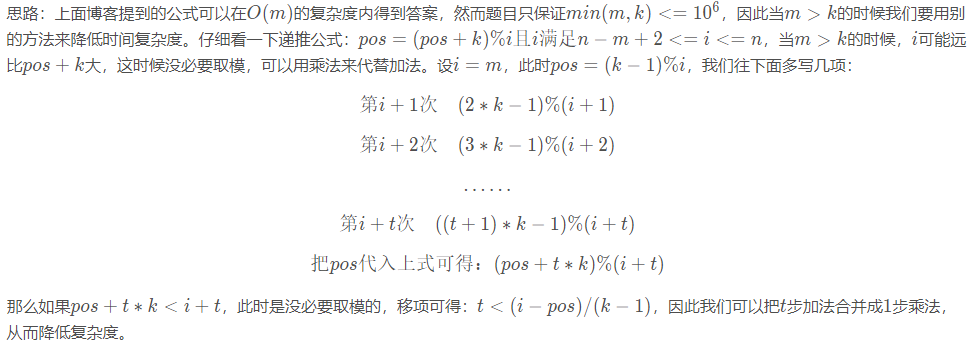
对于 较小 较大的情况，本题还有一种复杂度为 的算法。

考虑到我们每次走 个删一个，那么在一圈以内我们可以删掉 个，然后剩下了 个人。这时我们在第 个人的位置上。而你发现这个东西它等于 。于是我们继续递归处理，算完后还原它的相对位置。得到如下的算法：

1. **int** josephus(**int** n, **int** k) {
2. **if** (n == 1) **return** 0;
3. **if** (k == 1) **return** n - 1;
4. **if** (k > n) **return** (josephus(n - 1, k) + k) % n;  // 线性算法
5. **int** res = josephus(n - n / k, k);
6. res -= n % k;
7. **if** (res < 0)
8. res += n;  // mod n
9. **else**
10. res += res / (k - 1);  // 还原位置
11. **return** res;
12. }

可以证明这个算法的复杂度是 的。

**N个人从0开始，每k个出局，求第m个出局的编号**（非递归写法）



1. ll calc(ll n,ll m,ll k)  //n个人每k次出局一个人，求第m个出局的人的编号
2. {
3. **if**(k==1) **return** (m-1);
4. ll pos=-1,i=n-m;
5. **while**(i < n)
6. {
7. ll t=(i-pos-1)/(k-1);
8. **if**(t==0) //增加一次
9. {
10. pos=(pos+k)%(++i);
11. **if**(i >=n ) **return** pos;
12. }
13. **else**
14. {
15. **if**(i+t>n)
16. **return** pos+(n-i)\*k;
17. **else**
18. pos=(pos+t\*k),i+=t;
19. }
20. }
21. **return** pos;
22. }

# 奇淫技巧

## 超级快读

1. **namespace** IO       //使用的时候先IO::begin()
2. {
3. //然后输入时IO::read(x)
4. **const** **int** MX = 4e7;
5. **char** buf[MX];
6. **int** c, sz;
7. **void** begin()
8. {
9. c = 0;
10. sz = fread(buf, 1, MX, stdin);
11. }
12. **inline** **bool** read(**int** &t)
13. {
14. **while**(c < sz && buf[c] != '-' && (buf[c] < '0' || buf[c] > '9')) c++;
15. **if**(c >= sz) **return** **false**;
16. **bool** flag = 0;
17. **if**(buf[c] == '-') flag = 1, c++;
18. **for**(t = 0; c < sz && '0' <= buf[c] && buf[c] <= '9'; c++) t = t \* 10 + buf[c] - '0';
19. **if**(flag) t = -t;
20. **return** **true**;
21. }
22. }

## 其他库函数

1. **void** random\_shuffle(\_RandomAccessIterator \_\_first, \_RandomAccessIterator \_\_last)//随机打乱数组
2. **void** nth\_element(\_RandomAccessIterator \_\_first, \_RandomAccessIterator \_\_nth,\_RandomAccessIterator \_\_last);//O（n） 求第n 小
3. **void** reverse(\_BidirectionalIterator \_\_first, \_BidirectionalIterator \_\_last)//反转数组
4. **void** fill(\_ForwardIterator \_\_first, \_ForwardIterator \_\_last, **const** \_Tp& \_\_value)//数组填充value
5. //用于计算一个 32 位无符号整数有多少个位为1
6. \_\_builtin\_popcount (unsigned u);
7. //关闭同步流
8. ios::sync\_with\_stdio(**false**);
9. cin.tie(0);
10. cout.tie(0);

## 全排列函数

1. #include<cstdio>
2. #include<algorithm>
3. **using** **namespace** std;
4. **int** main()
5. {
6. **int** arr[10]={};
7. **int** n=3;
8. **for**(**int** i=1;i<=n;++i)   arr[i]=i;//初始化全排列
9. **do**{
10. **for**(**int** i=1;i<=n;++i)
11. printf("%d%c",arr[i],i==n?'\n':' ');
13. }**while**(next\_permutation(arr+1,arr+n+1));//得到该排列的下一个全排列,没有则返回0，且不改变数组
14. }

## 随机库函数

1. //c 随机数
2. srand(time(NULL))//初始化随机驱动器
3. **int** rand\_section(**int** l,**int** r){ **return**  rand()%(r-l+1)+l; }//随机生成l-r范围的数
4. **void** random\_shuffle(\_RandomAccessIterator \_\_first, \_RandomAccessIterator \_\_last)//随机打乱数组

//部分示范

1. default\_random\_engine send(2);//驱动器
2. uniform\_int\_distribution<**int**> distribution(1,100);//生成器
3. bernoulli\_distribution bool\_dis(0.4);//伯努利分布 p为true，1-p为false
4. normal\_distribution<**double**> normal\_dis(5.0,2.0);//正态分布
5. exponential\_distribution<**double**> exp\_dis(3.5);//指数函数分布
6. **while**(**true**)
7. {
8. auto  v=distribution(send);
9. cout<<v<<endl;//随机数
10. getchar();
11. }
12. //生成50个0~1范围的小数
13. default\_random\_engine send(time(NULL));
14. uniform\_real\_distribution<**float**> rgen(0.0,1.0);
15. **for**(**int** i=1; i<=50; ++i)
16. cout<<rgen(send)<<endl;//生成50个0~1范围的小数

## C++输出格式控制

**普通控制**

1. //<iomanip>           ---I/O流的控制头文件
2. setw(n)  //设域宽为n个字符
3. setfill(c)  //未用完的域宽填充字符c
4. cout<<setw(4)<<255<<endl;
5. // \_255 cout<<setfill('\*')<<setw(4)<<255<<endl;
6. // \*255; 注意：setw(n)只控制后一位的输出。setw(n)+setfill 组合用。
7. setbase(**int** n) //将数字转换为 n 进制
8. cout<<setbase(8)<<255<<' '<<16<<endl;
9. // 377 20       cout<<setbase(10)<<0x12A<<' '<<016<<endl;
10. // 298 14       cout<<setbase(3)<<255<<' '<<16<<endl;      //255 16 注意：⑴setbase(n)控制多位的输出。⑵参数只能为 8,10,16。
12. setprecision(n)  //控制输出显示浮点数的数字个数
13. setiosflags(ios::fixed) //固定的浮点显示，合用可控制小数位数。
14. cout<<setprecision(2)<<fixed<<2.316<<endl;  // 2.32（四舍五入）
15. setiosflags(ios::left)  //左对齐（默认右对齐

**C++多行输出**

1. **int** main()
2. {
3. cout<<R"(hello
4. world
5. )"<<endl;
6. }

## 二次函数求解

1. //ax^2+bx+c=0
2. **const** **double** EPS=1e-10;
3. **int** ff(**double** a,**double** b,**double** c,**double** &x1,**double** &x2)
4. {
5. **double** m=b\*b-4\*a\*c;
6. **if**(m<0.0)//无解
7. **return** 0;
8. m=sqrt(m);
9. **double** xa=(-b+m)/(a\*2.0),xb=(-b-m)/(a\*2.0);
10. x1=min(xa,xb);
11. x2=max(xa,xb);
12. **if**(m\*m<EPS)//一个解
13. **return** 1;
14. **else**
15. **return** 2;
16. }

## 日期计算

输出两个日期相差多少天

1. /\*
2. y-m-d
3. 365\*y+闰年的个数
4. 闰年=4的倍数不是100的倍数但是400的倍数
5. 从第1年开始计算,1-1-1是第1天
6. \*/
7. **int** getDay(**int** y,**int** m,**int** d)//第y年m月d天
8. {
9. **int** sum=0;
10. **int** yy=y-1;
11. sum=yy\*365+(yy/4-yy/100+yy/400);
12. **for**(**int** i=1;i<m;++i){
13. **if**(i==2) sum+=28+(((y%4==0&&y%100!=0)||y%400==0)?1:0);
14. **else** **if**(i==1||i==3||i==5||i==7 ||i==8 ||i==10 ||i==12) sum+=31;
15. **else** sum+=30;
16. }
17. sum+=d;
18. **return** sum;
19. }
20. **int** main()
21. {
22. **int** y1,y2,m1,m2,d1,d2;
23. **while**(cin>>y1>>m1>>d1)
24. {
25. cin>>y2>>m2>>d2;
26. cout<<getDay(y2,m2,d2)-getDay(y1,m1,d1)<<endl;//输出两个日期相差多少天
27. }
28. **return** 0;
29. }

## 线性递推BM

1. #include <bits/stdc++.h>
2. #define rep(i,a,n) for (long long i=a;i<n;i++)
3. #define pb push\_back
4. #define SZ(x) ((long long)(x).size())
5. **using** **namespace** std;
6. **typedef** vector<**long** **long**> VI;
7. **typedef** **long** **long** ll;
8. **const** ll mod=1e9+7;
9. ll qpow(ll a,ll b)
10. {
11. ll res=1;
12. a%=mod;
13. assert(b>=0);
14. **for**(; b; b>>=1)
15. {
16. **if**(b&1)res=res\*a%mod;
17. a=a\*a%mod;
18. }
19. **return** res;
20. }
21. **namespace** linear\_seq
22. {
23. **const** **long** **long** N=10010;
24. ll res[N],base[N],w[N],wd[N];
25. vector<ll> md;
26. **void** mul(ll \*a,ll \*b,ll k)
27. {
28. rep(i,0,k+k) w[i]=0;
29. rep(i,0,k) **if** (a[i])
30. rep(j,0,k)  w[i+j]=(w[i+j]+a[i]\*b[j])%mod;
32. **for** (ll i=k+k-1; i>=k; i--) **if** (w[i])
33. rep(j,0,SZ(md))
34. w[i-k+md[j]]=(w[i-k+md[j]]-w[i]\*wd[md[j]])%mod;
35. rep(i,0,k) a[i]=w[i];
36. }
37. ll solve(ll n,VI a,VI b)
38. {
39. // a 系数 b 初值 b[n+1]=a[0]\*b[n]+...
40. //        printf("%d\n",SZ(b));
41. ll ans=0,pnt=0;
42. ll k=SZ(a);
43. assert(SZ(a)==SZ(b));
44. rep(i,0,k) wd[k-1-i]=-a[i];
45. wd[k]=1;
46. md.clear();
47. rep(i,0,k) **if** (wd[i]!=0) md.push\_back(i);
48. rep(i,0,k) res[i]=base[i]=0;
49. res[0]=1;
50. **while** ((1ll<<pnt)<=n) pnt++;
51. **for** (ll p=pnt; p>=0; p--)
52. {
53. mul(res,res,k);
54. **if** ((n>>p)&1)
55. {
56. **for** (ll i=k-1; i>=0; i--) res[i+1]=res[i];
57. res[0]=0;
58. rep(j,0,SZ(md))
59. res[md[j]]=(res[md[j]]-res[k]\*wd[md[j]])%mod;
60. }
61. }
62. rep(i,0,k) ans=(ans+res[i]\*b[i])%mod;
63. **if** (ans<0) ans+=mod;
64. **return** ans;
65. }
66. VI BM(VI s)
67. {
68. VI C(1,1),B(1,1);
69. ll l=0,m=1,b=1;
70. rep(n,0,SZ(s))
71. {
72. ll d=0;
73. rep(i,0,l+1) d=(d+(ll)C[i]\*s[n-i])%mod;
74. **if** (d==0) ++m;
75. **else** **if** (2\*l<=n)
76. {
77. VI T=C;
78. ll c=mod-d\*qpow(b,mod-2)%mod;
79. **while**(SZ(C)<SZ(B)+m) C.pb(0);
80. rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;
81. l=n+1-l;
82. B=T;
83. b=d;
84. m=1;
85. }
86. **else**
87. {
88. ll c=mod-d\*qpow(b,mod-2)%mod;
89. **while**(SZ(C)<SZ(B)+m) C.pb(0);
90. rep(i,0,SZ(B)) C[i+m]=(C[i+m]+c\*B[i])%mod;
91. ++m;
92. }
93. }
94. **return** C;
95. }
96. ll gao(VI a,ll n)
97. {
98. VI c=BM(a);
99. c.erase(c.begin());
100. rep(i,0,SZ(c)) c[i]=(mod-c[i])%mod;
101. **return** solve(n,c,VI(a.begin(),a.begin()+SZ(c)));
102. }
103. };
105. **int** main()
106. {
107. ll n;
108. **while**(~scanf("%I64d", &n))
109. {
110. printf("%I64d\n",linear\_seq::gao(VI{1,5,11,36,95,281,781,2245,6336,18061, 51205},n-1));
111. }
112. }

## 程序对拍

产生数据代码：

data.cpp

1. freopen("data.txt","w",stdout);
2. /\*
3. 生成数据代码
4. \*/

我的代码：

my.cpp

1. freopen("data.txt","r",stdin);
2. freopen("my.txt","w",stdout);
3. //程序运行

正确代码：

ac.cpp

对拍代码：

Work.cpp

1. freopen("data.txt","r",stdin);
2. freopen("ac.txt","w",stdout);
3. //程序运行
4. #include <bits/stdc++.h>
5. **using** **namespace** std;
7. **int** main()
8. {
9. //注意该程序名字不能是fc,不然会与cmd的 fc冲突,
10. //比如说work.cpp
11. **for**(**int** \_t = 1; \_t <= 20; ++\_t)//测试 20 组
12. {
13. //产生数据程序, 产生数据到data.txt文件
14. system("C:\\Users\\12495\\Desktop\\coding\\data.exe");
15. **double** st = clock();
17. //找bug程序
18. //根据data.txt文件将结果输出到 my.txt
19. system("C:\\Users\\12495\\Desktop\\coding\\my.exe");
20. **double** en = clock();
22. //正确程序
23. //正确代码, 根据data.txt文件将结果输出到 ac.txt
24. system("C:\\Users\\12495\\Desktop\\coding\\ac.exe");
25. **double** fuck = clock();
27. //
28. //利用 cmd的 fc程序对比文件有无差异
29. **int** state=system("fc C:\\Users\\12495\\Desktop\\coding\\my.txt C:\\Users\\12495\\Desktop\\coding\\ac.txt");
30. **if**(state)
31. {
32. printf("\_\*\*\*\_\n");
33. puts("WA\n");
34. **return** 0;
35. }
36. printf("case %d:  my\_time = %fms   ac\_time = %f\n",\_t,en-st,fuck-en);
37. }
38. **return** 0;
39. }

# Python

**Python部分参考代码：**

1. #输入两个整数m n
2. n,m= [int(i) **for** i **in** input().strip().split()]
4. **if** \_\_name\_\_ == '\_\_main\_\_':#文件开头限制
5. #输入一行整数，形成一个整数列表
6. li=[int(i) **for** i **in** input().strip().split()]
8. #创建一个100\*100的二维数组
9. dp= [[0 **for** j **in** range(100)] **for** i **in** range(100)]
11. #全局变量
12. **global** n
14. #多组输入
15. **try**:
16. **while** True:
17. **pass**
18. **except**:
19. **pass**
21. #t组输入
22. t=int(input().strip())
23. **while** t>0:
24. t-=1
25. **pass**
27. #格式化输出 保留小数点
28. s="{}".format(100)#{}作为占位符，使用format成员函数可以对占位符进行填充
29. **print**("%.2f %.3f"% (100,5.155))#“里面的为”格式化控制字符串，右边的数据，用空格隔开
31. #表达式求值eval函数
32. s="1^1+5+max(1,-1)+pow(10,2)"
33. **print**(eval(s))#输出106
34. #地板除符号//  a//b

**数学工具库**

1. #数学工具库
2. **import** math
3. **print**(math.pow(1,2))
4. **print**(math.log2(8))
5. **print**(math.log(100,10))
6. #sin asin等与c++相同

**字符串成员函数**

1. #字符串成员函数
2. str.strip()#左右两边去空格
3. str.lstrip())#左边去空格
4. str.rstrip())#右边去空格
5. str.upper())# 将str全部转化为大写
6. str.lower())#将str全部转化为小写
7. str.count(s))#返回字符串有多少个字符串s
8. str.isdight())#是否为一个数字
9. str.isalnum())#是否为一个数字

**列表成员函数**

1. #列表成员函数
2. list.sort()#引用排序
3. list.reverse()#引用反转
4. list.insert(int sub,val)
5. list.remove(val)#删除val 但是只删除一个 从左到右数的第一个
6. list.append(val)#追加
7. list.pop()#弹出一个 可以指定一个下标
8. list+list #列表的合并
10. list\*num  #List为常量最好,否则可能会出现指针混乱
12. list[index]#用索引进行读取
13. **print**(list)#输出列表

**字典成员函数**

1. #初始化方法
2. dictName={key1:val1,key2,val2}
3. dictName={} 代表空字典
5. #添加方法
6. ​dictName.setdefault(key,default)//如果有的话不操作 没有的话设置默认值 并且返回内容值
7. ​dictName[key]=val
9. #删除方法
10. ​**del** dictName[key] //没有返回值 没有则报错
12. #查找方法
13. ​dicttName.get(key,default)//有的话返回 没有的话返回默认值
14. ​key **in** dictName : 返回为真则存在 否则Flase
16. #清除容器
17. ​dictName.clear()//
19. #返回所有key值
20. ​dictName.keys()//列表？
22. #返回所有内容值
23. ​dicttName.values()#列表

**集合成员函数**

1. #空集合
2. setname=set()
3. #初始化
4. setname{name,name,name}
6. #添加:
7. setname.add()
9. #删除
10. setname.discard()//删除 没有也不会报错
11. setname.remove()//待测试
13. #查找
14. key **in** setnam #返回为真则存在 否则Flase
16. #遍历集合
17. **for** val **in** setname:
19. #集合的 并 交 异或（吧）
20. numbers1={1,2,3,4,5}
21. numbers2={1,3,4,6}
22. # 并  交  异或
23. **print**(numbers1 | numbers2)
24. **print**(numbers1 & numbers2)
25. **print**(numbers1 - numbers2)

**py自定义排序**

1. #自定义排序
2. **class** Dch:
3. **def** \_\_init\_\_(self,a,b):
4. self.first=a
5. self.second=b
6. **def** \_\_str\_\_(self):
7. **return** "first={},second={}".format(self.first,self.second)
8. **def** \_\_lt\_\_(self, other):
9. **if** (self.first == other.first):
10. **return** int(self.second < other.second)
11. **return** int(self.first < other.first)
12. # isinstance(100,int)
13. aa=Dch(1,2)
14. bb=Dch(100,1)
15. cc=Dch(50,50)
16. dd=Dch(30,-1)
17. ee=Dch(1,20)
18. Mylist=[aa,bb,cc,dd,ee]
19. Mylist=sorted(Mylist)
20. **for** i **in** range(len(Mylist)):
21. **print**(Mylist[i])



**第二种用sort的key参数和lambda表达式**

可以参考python的官网文档：<https://docs.python.org/zh-cn/3/library/stdtypes.html#list.sort>

1. **class** pair:
2. **def**  \_\_init\_\_(self,x=0,y=0):
3. self.x=x
4. self.y=y
5. **def** \_\_lt\_\_(self, other):
6. **if** self.x==other.x:
7. **return** self.y<other.y
8. **return** self.x-self.y
9. **def** \_\_str\_\_(self):
10. **return** "%d,%d"% (self.x,self.y)
11. me=[]
12. me.append(pair(3,4))
13. me.append(pair(1,3))
14. me.append(pair(-1,-1))
15. me.sort(key=**lambda** x:(x.y))
16. **for** i **in** me:
17. **print**(i)
18. #输出：
19. #-1,-1
20. #1,3
21. #3,4

**深入使用sort的key参数**

1. **class** pair:
2. **def** \_\_init\_\_(self,x=0,y=0):
3. self.x=x
4. self.y=y
5. **def** \_\_str\_\_(self):
6. **return** "{}  {}".format(self.x,self.y)
7. **if** \_\_name\_\_ == '\_\_main\_\_':
8. mmp=[]
9. mmp.append(pair(1,2))
10. mmp.append(pair(0, 2))
11. mmp.append(pair(4, 5))
12. mmp.append(pair(7, 8))
13. mmp.append(pair(4, 3))
14. mmp.sort(key=**lambda** me:(me.x,me.y))#这样就会按照元组的字典序进行排序
15. **for** i **in** mmp:
16. **print**(i)
17. '''''
18. 输出
19. 0  2
20. 1  2
21. 4  3
22. 4  5
23. 7  8
24. '''
25. #

**Random的部分方法**

1. # random各种使用方法
2. **import** random
4. # 随机生成[0.1)的浮点数
5. **print**("random():", random.random())
7. # 随机生成1000-9999之间的整数
8. **print**("randint(1000, 9999):", random.randint(1000, 9999))
10. # 随机生成0-20之间的偶数
11. **print**("randrange(0, 21, 2):", random.randrange(0, 21, 2))
13. # 随机生成0-20之间的浮点数
14. **print**("uniform(0, 20):", random.uniform(0, 20))
16. # 从序列中随机选择一个元素
17. list\_string = ['a', 'b', 'c', 'd', 'e']
18. **print**("choice(list):", random.choice(list\_string))
19. **print**("choice(string):", random.choice('abcd'))
21. # 对列表元素随机排序
22. list\_number = [1, 2, 3, 4, 5]
23. random.shuffle(list\_number)
24. **print**("shuffle(list):", list\_number)
25. # 从指定序列中随机获取指定长度的片断
26. **print**("sample(sequence):", random.sample('abcdefg', 2))

# JAVA大数和排序（未做）