**ACM算法模版**





**Dawxy**

<http://dawxy.com>

Email：shy@dawxy.com

Last bulid at 2016/03/19

**Catalog**

[1. 字符串 - 1 -](#_Toc446170376)

[1.1. KMP算法 - 1 -](#_Toc446170377)

[1.2. 字符串最小(大)表示法 - 2 -](#_Toc446170378)

[1.3. Manacher算法-最长回文子串 - 2 -](#_Toc446170379)

[1.4. AC自动机 - 3 -](#_Toc446170380)

[1.5. 后缀数组(DA倍增算法) - 5 -](#_Toc446170381)

[2. 数学 - 6 -](#_Toc446170382)

[2.1. 乘法逆元 - 6 -](#_Toc446170383)

[2.2. 快速乘法和快速幂 - 7 -](#_Toc446170384)

[2.3. 矩阵快速幂 - 7 -](#_Toc446170385)

[2.3.1. 优化递推式 - 7 -](#_Toc446170386)

[2.3.2. 优化动态规划 - 9 -](#_Toc446170387)

[2.4. 快速查找素数 - 11 -](#_Toc446170388)

[2.4.1. 直接bool数组判断是否是prime - 11 -](#_Toc446170389)

[2.4.2. 数组顺序储存每一个素数并在prime[0]保存2~n范围内有几个素数 - 11 -](#_Toc446170390)

[2.4.3. Miller Rabin素数测试 - 11 -](#_Toc446170391)

[2.5. 统计每个数的约数个数 - 13 -](#_Toc446170392)

[2.6. 分解质因数求最小公倍数(LCM) - 13 -](#_Toc446170393)

[2.6.1. 单个数字的分解质因数 - 13 -](#_Toc446170394)

[2.6.2. 筛法求多个数的分解质因数 - 14 -](#_Toc446170395)

[2.7. 互质统计 - 14 -](#_Toc446170396)

[2.7.1. 求1~n与n互质个数(欧拉函数) - 14 -](#_Toc446170397)

[2.7.2. 求x~y中与n互质的个数(容斥原理) - 16 -](#_Toc446170398)

[2.8. 扩展GCD - 16 -](#_Toc446170399)

[2.9. 求解模线性方程 - 17 -](#_Toc446170400)

[2.10. 费马小定理+指数循环节公式 - 19 -](#_Toc446170401)

[2.11. 高斯消元(求解方程组) - 20 -](#_Toc446170402)

[2.11.1. 浮点数(整数) - 20 -](#_Toc446170403)

[2.11.2. 开关问题(异或方程组) - 21 -](#_Toc446170404)

[2.12. lucas定理(c(n,m) mod p) - 22 -](#_Toc446170405)

[2.13. 中国剩余定理(CRT) - 23 -](#_Toc446170406)

[2.13.1. 求x%(m[1]\*m[2]\*…m[n])的值 - 23 -](#_Toc446170407)

[2.13.2. 求出x的值 - 24 -](#_Toc446170408)

[2.14. 博弈 - 24 -](#_Toc446170409)

[2.14.1. 巴什博奕（Bash Game） - 24 -](#_Toc446170410)

[2.14.2. 威佐夫博奕（Wythoff Game） - 25 -](#_Toc446170411)

[2.14.3. 尼姆博奕（Nimm Game） - 25 -](#_Toc446170412)

[2.14.4. 斐波那契博弈(Fibonacci Game) - 26 -](#_Toc446170413)

[3. 数据结构 - 26 -](#_Toc446170414)

[3.1. Hash表 - 26 -](#_Toc446170415)

[3.2. 树状数组(fenwick树) - 27 -](#_Toc446170416)

[3.3. 静态RMQ问题ST-在线算法 - 28 -](#_Toc446170417)

[3.4. 线段树 - 28 -](#_Toc446170418)

[3.4.1. 优化空间写法 - 29 -](#_Toc446170419)

[3.4.2. ZKW线段树(从下往上更新) - 29 -](#_Toc446170420)

[3.4.3. 单调性修改线段树 - 30 -](#_Toc446170421)

[3.4.4. 二维线段树 - 32 -](#_Toc446170422)

[3.5. 划分树-(静态区间第k大,k小和) - 35 -](#_Toc446170423)

[3.6. 最近公共祖先(LCA) - 36 -](#_Toc446170424)

[3.6.1. 离线 Tarjan 算法 - 36 -](#_Toc446170425)

[3.6.2. 在线转RMQ算法 - 37 -](#_Toc446170426)

[3.7. 树上DFS序和LCA的深度序区别 - 38 -](#_Toc446170427)

[3.8. 主席树(函数式/可持久化线段树) - 38 -](#_Toc446170428)

[3.8.1. 静态区间第k大-HDU2665 - 38 -](#_Toc446170429)

[3.8.2. 单点修改的区间第K大-ZOJ2112 - 40 -](#_Toc446170430)

[3.8.3. 树上第K大-bzoj1146 - 42 -](#_Toc446170431)

[3.9. 树链剖分 - 45 -](#_Toc446170432)

[3.9.1. 点权(线段树) - 45 -](#_Toc446170433)

[3.9.2. 边权(线段树) - 47 -](#_Toc446170434)

[3.9.3. 有序合并 - 50 -](#_Toc446170435)

[3.10. 扫描线(线段树辅助) - 50 -](#_Toc446170436)

[3.10.1. 矩形面积并 - 50 -](#_Toc446170437)

[3.10.2. 多个矩形总周长 - 53 -](#_Toc446170438)

[3.10.3. 矩形能包含多少点 - 53 -](#_Toc446170439)

[3.11. Trie树 - 54 -](#_Toc446170440)

[3.12. 平衡树-Treap - 56 -](#_Toc446170441)

[3.13. 伸展树Splay-Tree - 58 -](#_Toc446170442)

[3.14. 动态树-LCT(link-cut-tree) - 62 -](#_Toc446170443)

[3.15. 莫队算法 - 66 -](#_Toc446170444)

[4. 图论 - 68 -](#_Toc446170445)

[4.1. 图中是否有环(圈)的判定 - 68 -](#_Toc446170446)

[4.2. 欧拉路径和欧拉回路判定 - 69 -](#_Toc446170447)

[4.3. 平面图-完全子图性质 - 70 -](#_Toc446170448)

[4.4. 树的直径(树的最长路) - 70 -](#_Toc446170449)

[4.5. 无向图的割顶和桥 - 71 -](#_Toc446170450)

[4.5.1. 无向图的割顶(Tarjan) - 71 -](#_Toc446170451)

[4.5.2. 无向图的桥(Tarjan) - 72 -](#_Toc446170452)

[4.6. 有向图强连通分量(SCC) - 72 -](#_Toc446170453)

[4.7. 拓扑排序模版 - 74 -](#_Toc446170454)

[4.8. 2-sat问题 - 75 -](#_Toc446170455)

[4.9. 最小生成树两种模板(MST) - 76 -](#_Toc446170456)

[4.9.1. kruskal算法 - 76 -](#_Toc446170457)

[4.9.2. prim算法 - 77 -](#_Toc446170458)

[4.10. 最小树形图 - 79 -](#_Toc446170459)

[4.11. 次小生成树 - 80 -](#_Toc446170460)

[4.12. 最短路算法模板 - 82 -](#_Toc446170461)

[4.12.1. dijkstra - 82 -](#_Toc446170462)

[4.12.2. SPFA - 83 -](#_Toc446170463)

[4.12.3. floyd - 84 -](#_Toc446170464)

[4.13. 最大流 - 85 -](#_Toc446170465)

[4.13.1. 最大流(最小割)-dinic算法（非递归） - 85 -](#_Toc446170466)

[4.13.2. 最大流(最小割)-EdmondsKarp - 87 -](#_Toc446170467)

[4.13.3. 最大流(最小割)-isap - 88 -](#_Toc446170468)

[4.14. 最小费用最大流 - 92 -](#_Toc446170469)

[4.14.1. 普通SPFA版 - 92 -](#_Toc446170470)

[4.14.2. ZKW费用流 - 93 -](#_Toc446170471)

[4.15. 二分图匹配模板 - 95 -](#_Toc446170472)

[4.15.1. 无权图最大匹配 - 95 -](#_Toc446170473)

[4.15.2. 有权图最佳完美匹配 - 98 -](#_Toc446170474)

[4.16. 差分约束 - 100 -](#_Toc446170475)

[5. 动态规划 - 102 -](#_Toc446170476)

[5.1. 可以找零的多重背包+完全背包问题 - 102 -](#_Toc446170477)

[5.2. LCS最长公共子序列 - 103 -](#_Toc446170478)

[5.3. LIS(最长上升子序列两种算法模板) - 105 -](#_Toc446170479)

[5.4. LCS转LIS - 106 -](#_Toc446170480)

[5.5. LCIS最长公共上升子序列模板 - 106 -](#_Toc446170481)

[5.6. 数位DP - 108 -](#_Toc446170482)

[5.6.1. 统计个数 - 108 -](#_Toc446170483)

[5.6.2. 统计和(平方和) - 109 -](#_Toc446170484)

[5.7. 树形DP - 110 -](#_Toc446170485)

[5.7.1. 每个人最多只有一个上级，但可以有多个下,求最大价值 - 110 -](#_Toc446170486)

[5.7.2. 结点占领花费取价值最大 - 112 -](#_Toc446170487)

[5.8. 最大连续子串(子矩阵)和 - 113 -](#_Toc446170488)

[5.9. 最大M子段和 - 114 -](#_Toc446170489)

[5.10. 状压DP+插头DP - 116 -](#_Toc446170490)

[5.11. 第k大01背包 - 118 -](#_Toc446170491)

[5.12. 斜率DP - 119 -](#_Toc446170492)

[6. 搜索 - 120 -](#_Toc446170493)

[6.1. Dancing Links(DLX舞蹈链) - 120 -](#_Toc446170494)

[6.1.1. 精确覆盖 - 120 -](#_Toc446170495)

[6.1.2. 重复覆盖(不冲突子集) - 122 -](#_Toc446170496)

[7. 其他 - 124 -](#_Toc446170497)

[7.1. 一些注意事项 - 124 -](#_Toc446170498)

[7.2. 解决递归爆栈问题 - 124 -](#_Toc446170499)

[7.3. IO输入输出挂 - 125 -](#_Toc446170500)

[7.3.1. fread正负数 - 125 -](#_Toc446170501)

[7.3.2. putchar输出(非负整数) - 126 -](#_Toc446170502)

[7.3.3. fwrite输出(字符，字符串，正负整数) - 126 -](#_Toc446170503)

[7.3.4. JAVA输入 大数A+B - 126 -](#_Toc446170504)

[7.4. 高精度 - 127 -](#_Toc446170505)

[7.4.1. C++手写BigInteger类 - 127 -](#_Toc446170506)

[7.4.2. JAVA -BigInteger - 131 -](#_Toc446170507)

[7.4.3. JAVA -BigDecimal - 132 -](#_Toc446170508)

[7.5. 公式计算，给定年月日输出星期几 - 132 -](#_Toc446170509)

[7.6. n!的位数 - 133 -](#_Toc446170510)

[7.7. 时针和分针角度计算 - 133 -](#_Toc446170511)

[7.8. 整点正多边形判断(只有正方形) - 133 -](#_Toc446170512)

[7.9. STL - 133 -](#_Toc446170513)

[7.9.1. 优先队列 priority\_queue - 133 -](#_Toc446170514)

[7.9.2. set,multiset,map… - 134 -](#_Toc446170515)

[7.9.3. lower\_bound，upper\_bound, binary\_search - 135 -](#_Toc446170516)

[7.9.4. nth\_element(第k小元素) - 136 -](#_Toc446170517)

[7.9.5. bitset-超大位数位运算容器 - 136 -](#_Toc446170518)

[7.9.6. 非标准库rope(可持久化平衡树) - 136 -](#_Toc446170519)

[7.10. VS配置 - 137 -](#_Toc446170520)

# 字符串

## KMP算法

Kmp O(n+m) KMP是单模版匹配文本串

最小循环节: i%(i-next[i])==0,此字符串的最大循环串的循环次数就为i/(i-next[i]);//用get\_next1()得到的

【next下标0~n-1，i对应的下标1~n】

1. int nt[MAXN];//nt[i]为满足x[i-z…i-1]=x[0…z-1]的最大z值(就是相同的最大前缀后缀)
2. void get\_next1(int \*x, int m)//(预处理要查询的文本，即key)
3. {
4. int i = 0,j;
5. j = nt[0] = -1;
6. while(i < m)
7. {
8. while(~j && x[i] != x[j]) j = nt[j];
9. nt[++i] = ++j;
10. }
11. }
12. void get\_next2(int \*x, int m)//这个nt[i]表示nt’[i] = nt[nt[…nt[i]]]
13. { //直到nt'[i] < 0 或者x[nt’[i]] != x[i],这样要快一些
14. int i = 0,j;
15. j = nt[0] = -1;
16. while(i < m)
17. {
18. while(~j && x[i] != x[j]) j = nt[j];
19. if(x[++i] == x[++j]) nt[i] = nt[j];
20. else nt[i] = j;
21. }
22. }
23. //返回x在y中出现的次数,可以重叠
24. int kmp(int \*x, int m, int \*y, int n)//x是模式串,y是主串
25. {
26. int i = 0,j = 0,ans = 0;
27. get\_next(x,m);
28. while(i < n)
29. {
30. while(~j && y[i] != x[j]) j = nt[j];
31. i++,j++;
32. if(j == m)
33. {
34. ans++;
35. j = nt[j];//改为j=0，则计算不重复数量
36. }
37. }
38. return ans;
39. }

## 字符串最小(大)表示法

循环字符串的最小表示法的问题可以这样描述：

**对于一个字符串S，求S的循环的同构字符串S’中字典序最小的一个。**

设S=bcad，且S’是S的循环同构的串。S’可以是bcad或者cadb,adbc,dbca。而且最小表示的S’是adbc。

【O(n)模版】

1. int get\_min(char\*s, int len, char \*ans = NULL)//ans为最小表示后的串，不输入就不求
2. {
3. int i = 0,j = 1,k = 0,tmp;
4. while(i < len && j < len && k < len)
5. {
6. tmp = s[(i+k)%len] - s[(j+k)%len];
7. if(!tmp) k++;
8. else
9. {
10. if(tmp > 0) i += k+1;//改成<就是最大表示法
11. else j += k+1;
12. if(i == j) j++;
13. k = 0;
14. }
15. }
16. int st = min(i, j);//st-最小表示法在原串起始位置
17. if(ans)
18. {
19. rep(i,0,len) ans[i] = s[(st+i)%len];
20. ans[len] = 0;
21. }
22. return st;//返回的st是下标最小的起始位置,要求最大位置，用2倍主串的kmp判最大即可
23. }

## Manacher算法-最长回文子串

O(n)时间求最长回文子串

1. //p[i]记录i为中心的回文串半径,添加#后p[i]-1就是原串的回文长度了
2. int p[MAXN<<1];
3. char s[MAXN<<1];//中间加上'#'后的串,下标1开始的
4. int manacher(char \*a, int n, char \*outs)//a[]-原串,n原串长度,outs-第一个最长回文子串
5. {
6. //给原串加'#'
7. int len = 0;
8. s[len++] = '$';
9. s[len++] = '#';
10. rep(i,0,n)
11. {
12. s[len++] = a[i];
13. s[len++] = '#';
14. }
15. s[len] = 0;
16. //开始计算最长回文串
17. int mx = 0, id = 0;
18. rep(i,0,len)
19. {
20. p[i] = mx>i?min(p[id\*2-i],mx-i):1;
21. while(s[i+p[i]] == s[i-p[i]]) p[i]++;
22. if(i+p[i] > mx)
23. {
24. mx = i+p[i];
25. id = i;
26. }
27. }
28. //得出最长值
29. int ans = -1;
30. rep(i,0,len)
31. {
32. if(ans < p[i]-1) ans = p[i]-1, id = i;
33. }
34. p[id]--;
35. int cnt = 0;
36. for(int i=id-p[id];i<=id+p[id];i++){
37. if(s[i]=='#')continue;
38. outs[cnt++] = s[i];
39. }
40. outs[cnt] = 0;
41. return ans;
42. }

## AC自动机

多模版匹配，用struct慢一倍

1. /\*O(n)时间(预处理模版串O(m\*len))求多模版字符串匹配，这里是计算文章中出现的模版数目(有重复模版)\*/
2. const int INF = 0x3f3f3f3f, MAXN = 100000+10;//MAXN所有字符串最多可能组成几个结点(一般为n\*len)
3. struct Trie{
4. int cnt;//当前节点总数
5. int ch[MAXN][26], fail[MAXN], end[MAXN];//儿子节点，失配指针,单词结束计数
6. int newnode()
7. {
8. clc(ch[cnt],-1);
9. end[cnt++] = 0;
10. return cnt-1;
11. }
12. void init()//0为根节点
13. {
14. cnt = 0;
15. newnode();
16. }
17. void insert(char \*s)//trie的插入
18. {
19. int u = 0,len = strlen(s);
20. rep(i,0,len)
21. {
22. int& c = ch[u][s[i]-'a'];//u的儿子
23. if(-1 == c)//新建
24. c = newnode();
25. u = c;
26. }
27. end[u]++;
28. }
29. void get\_fail()//得到失配指针(前缀后缀相同)
30. {
31. queue<int> q;
32. //特殊处理第一层的儿子节点，在模版比较多时合并在while中会变慢
33. fail[0] = 0;
34. rep(i,0,26)
35. {
36. int& c = ch[0][i];
37. if(~c)//存在的节点加入队列
38. {
39. fail[c] = 0;
40. q.push(c);
41. }
42. else c = 0;//不存在的全部变成root节点，方便操作
43. }
44. while(!q.empty())
45. {
46. int u = q.front();q.pop();
47. rep(i,0,26)
48. {
49. int& c = ch[u][i];//u的儿子结点c
50. int p = ch[fail[u]][i];//u的失配节点和c相同的儿子节点
51. if(~c)//存在c儿子，则把c失配指针指向p(增加前后缀相同的长度,
52. { //如果没有不会增加还是0,因为第一层失配节点都指向了0)
53. fail[c] = p;
54. q.push(c);
55. }
56. else c = p;//不存在的话直接指向p
57. }
58. }
59. }
60. int query(const char \*s)
61. {
62. int len = strlen(s);
63. int u = 0, ans = 0;
64. rep(i,0,len)
65. {
66. u = ch[u][s[i]-'a'];
67. int temp = u;
68. while(temp)//沿着失配指针往上走，0为根结束
69. {
70. ans += end[temp];//加上当前出现的单词数
71. end[temp] = 0;//防止重复计算且算出有几个模版串出现在当前文章中(如果查询多次需要变量保存变成0的end，查完后恢复),不加可计算所有模版串在当前文章中出现的次数之和
72. temp = fail[temp];
73. }
74. }
75. return ans;
76. }
77. }ac;

## 后缀数组(DA倍增算法)

时间复杂度O((n+m)\*) n为被查找的文本串长度+1 (预处理被查找文本),m是字符最大值

1. const int INF = 0x3f3f3f3f, MAXN = 100\*1001+10;
2. struct DA{
3. int s[MAXN];//文本串(末尾加0,前面字符>0)
4. int sa[MAXN], t[MAXN], t2[MAXN], c[MAXN], n;
5. /\*sa[]是后缀数组(sa[0]为n无效,sa[1~n]存放字典序从小到大的后缀串i,后缀串i表示s[i]~s[(strlen(s)-1)]的串);
6. t[],t2[]是辅助空间,c[]是桶排序的数组桶
7. n是文本串长度(加完字符后)\*/
8. void bulid\_sa(int m) //m是出现字符的范围+1(也就是最大的那个字符大小+1) O(nlog(n))
9. {
10. int \*x = t, \*y = t2, p;//这样交换t1[],t2[]只需要更换指针
11. //第一轮基数排序,如果m很大可用sort+HASH实现n\*logn的预处理，总复杂度不变
12. memset(c,0,sizeof(int)\*(m));//初始化c[0]~c[m-1]为0
13. rep(i,0,n) c[x[i] = s[i]]++;
14. rep(i,1,m) c[i] += c[i-1];
15. per(i,n-1,0) sa[--c[x[i]]] = i;
16. for(int k = 1; k <= n; k <<= 1, m = p)
17. {
18. p = 0;
19. //直接利用sa[]排序第二关键字
20. rep(i,n-k,n) y[p++] = i;
21. rep(i,0,n) if(sa[i] >= k) y[p++] = sa[i]-k;
22. //基数排序第一关键字
23. memset(c,0,sizeof(int)\*m);
24. rep(i,0,n) c[x[y[i]]]++;
25. rep(i,1,m) c[i] += c[i-1];
26. per(i,n-1,0) sa[--c[x[y[i]]]] = y[i];
27. //根据sa和y数组计算新的x数组
28. swap(x,y);
29. p = 1;x[sa[0]] = 0;
30. rep(i,1,n) x[sa[i]] = y[sa[i-1]] == y[sa[i]] && y[sa[i-1]+k] == y[sa[i]+k]?p-1:p++;
31. if(p >= n) break;//p是rank(y)值不相同的数量,p>=n说明排序编号都不同了，不需要继续排下去
32. }
33. }
34. int rank[MAXN], height[MAXN];
35. //rank[i]表示后缀i在sa[]中的下标(就是后缀i排第几)，rank[0~n-1]存放后缀i排第几(1~n),rank[n]为0无效
36. //height[i]定义为sa[i-1]和sa[i]的最长公共前缀(LCP)长度,height[2~n]为有效值
37. void get\_height() //O(n)
38. {
39. int k = 0;
40. rep(i,0,n) rank[sa[i]] = i;//获取rank值
41. rep(i,0,n)
42. {
43. if(k) k--;
44. int j = sa[rank[i]-1];
45. while(s[i+k] == s[j+k])
46. k++;
47. height[rank[i]] = k;
48. }
49. }
50. int dp[MAXN][20];
51. void rmq\_init()//初始化rmq O(nlog(n))
52. {
53. repe(i,1,n) dp[i][0] = height[i];
54. for(int j = 1; (1<<j)<=n; j++)
55. {
56. for(int i = 1; i+(1<<j)-1 <= n; i++)
57. dp[i][j] = min(dp[i][j-1],dp[i+(1<<(j-1))][j-1]);
58. }
59. }
60. int ask\_rmq(int x, int y)
61. {
62. int k = 0;
63. while((1<<(k+1)) <= y-x+1) k++;
64. return min(dp[x][k], dp[y-(1<<k)+1][k]);
65. }
66. int lcp(int x, int y)//求出后缀x和y的最长公共前缀 O(1)
67. {
68. x = rank[x], y = rank[y];
69. return x<y?ask\_rmq(x+1,y):ask\_rmq(y+1,x);
70. }
71. }da;

# 数学

## 乘法逆元

**mod是素数时a和b可以是任何正数。**

()%mod = a \* ()%mod

**否则，mod必须与b互质(gcd(mod,b) == 1)。**

则此时为()%mod = a\*%mod

## 快速乘法和快速幂

都是的

【快速乘法(乘法变加法)】可用于爆long long的乘法(比如MOD为)

1. LL mul\_mod(LL a, LL b)//a\*b对MOD取余
2. {
3. a %= MOD;
4. LL ans = 0;
5. while(b)
6. {
7. if(b&1)
8. {
9. ans += a;
10. if(ans >= MOD) ans -= MOD;
11. }
12. a <<= 1;
13. if(a >= MOD) a -= MOD;
14. b >>= 1;
15. }
16. return ans;
17. }

【快速幂】

1. LL pow\_mod(LL x, LL n)
2. {
3. LL ans = 1;
4. while(n)
5. {
6. if(n&1) ans = mul\_mod(ans,x);
7. x = mul\_mod(x,x);
8. n >>= 1;
9. }
10. return ans;
11. }

## 矩阵快速幂

### 优化递推式

**矩阵快速幂时间复杂度O()**

【描述】

给你一个递推公式：

f(x)=a\*f(x-2)+b\*f(x-1)+c

并给你f(1),f(2)的值，请求出f(n)的值，由于f(n)的值可能过大，求出f(n)对1000007取模后的值。

其中0<=f(1),f(2)<100,-100<=a,b,c<=100,1<=n<=100000000 (10^9)

【解法】构造矩阵

X

{0 a 0

1 b 0

0 1 1}

使得{f(x-2),f(x-1),c} \* X = {f(x-1),a\*f(x-2)+b\*f(x-1)+c, c} == {f(x-2),f(x),c}

{f(1),f(2),c}乘上N-1次X后就可得出{f(n),f(n+1),c} f(n)就是最终答案

也就{f(1),f(2),c} \* X^n-1 = {f(n),f(n+1),c};

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long LL;
4. #define rep(i,a,n) for(int i = a; i < n; i++)
5. #define repe(i,a,n) for(int i = a; i <= n; i++)
6. #define per(i,n,a) for(int i = n; i >= a; i--)
7. #define clc(a,b) memset(a,b,sizeof(a))
8. const int INF = 0x3f3f3f3f, MOD = 1000007;
9. struct MATRIX{//矩阵
10. LL num[3][3];
11. MATRIX(LL a1,LL a2,LL a3, LL b1, LL b2, LL b3, LL c1, LL c2, LL c3)
12. {
13. num[0][0] = a1, num[0][1] = a2, num[0][2] = a3;
14. num[1][0] = b1, num[1][1] = b2, num[1][2] = b3;
15. num[2][0] = c1, num[2][1] = c2, num[2][2] = c3;
16. }
17. MATRIX operator \*(const MATRIX& b)//矩阵a\*b
18. {
19. MATRIX ans=MATRIX(0,0,0,0,0,0,0,0,0);
20. rep(i,0,3)
21. {
22. rep(j,0,3)
23. {
24. rep(k,0,3)
25. ans.num[i][j] = (ans.num[i][j]+num[i][k]\*b.num[k][j])%MOD;
26. }
27. }
28. return ans;
29. }
30. MATRIX operator ^ (LL n)//快速幂
31. {
32. MATRIX ans = MATRIX(1,0,0,0,1,0,0,0,1);//任何3\*3矩阵乘以这个单位矩阵不会改变值
33. MATRIX x = \*this;
34. while(n)
35. {
36. if(n&1) ans = ans\*x;
37. x = x\*x;
38. n >>= 1;
39. }
40. return ans;
41. }
42. };
43. int main()
44. {
45. #ifdef SHY
46. freopen("d:\\1.txt", "r", stdin);
47. #endif
48. int t;
49. scanf("%d", &t);
50. while(t--)
51. {
52. int f1,f2,a,b,c;
53. LL n;
54. scanf("%d %d %d %d %d %lld", &f1, &f2, &a, &b, &c, &n);
55. MATRIX x = MATRIX(0,a,0,1,b,0,0,1,1);//构造出的矩阵
56. x = x^(n-1);
57. int ans = (f1\*x.num[0][0] + f2\*x.num[1][0] + c\*x.num[2][0])%MOD;
58. //只要求出[0][0]即可
59. printf("%d\n", ans);
60. }
61. return 0;
62. }

### 优化动态规划

【题目描述】求在所有长度在[l, r]之间的能被7整除且相邻数位之和不为k的正整数有多少个。l,r <= 10^9; 结果模1e9+7。

【DP方程】**dp[i+1][l][(k\*10+l)%7] += dp[i][j][k] (0 <= l <= 9)**

**dp[i][j][k]表示长度为i位的数最低位为j且值%7=k**

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long LL;
4. #define rep(i,a,n) for(int i = a; i < n; i++)
5. #define repe(i,a,n) for(int i = a; i <= n; i++)
6. #define per(i,n,a) for(int i = n; i >= a; i--)
7. #define clc(a,b) memset(a,b,sizeof(a))
8. const int INF = 0x3f3f3f3f, MAXN = 71;
9. LL MOD = 1e9+7;
10. struct MATRIX{
11. LL num[MAXN][MAXN];
12. MATRIX(){
13. clc(num,0);
14. }
15. MATRIX operator\*(const MATRIX& b)const{
16. MATRIX ans;
17. rep(i,0,MAXN)
18. {
19. rep(k,0,MAXN)
20. {
21. rep(j,0,MAXN)
22. {
23. ans.num[i][j] = (ans.num[i][j]+num[i][k]\*b.num[k][j])%MOD;
24. }
25. }
26. }
27. return ans;
28. }
29. MATRIX operator ^(int n)
30. {
31. MATRIX ans, x = \*this;
32. if(n < 0) return ans;
33. rep(i,0,MAXN) ans.num[i][i] = 1;
34. while(n)
35. {
36. if(n&1) ans = ans\*x;
37. x = x\*x;
38. n >>= 1;
39. }
40. return ans;
41. }
42. };
43. LL st[MAXN],ed[MAXN];//初始状态和最后状态的各个结果，0~69就是对应dp的70个状态，70对应dp[i-1]所有%7=0的方案之和
44. inline int mhash(int x, int y){return x\*7+y;}//mhash(x,y)表示把最低位为x且%7为y的状态
45. int main()
46. {
47. #ifdef SHY
48. freopen("d:\\1.txt", "r", stdin);
49. #endif
50. int t;
51. scanf("%d", &t);
52. rep(i,1,10) st[mhash(i,i%7)] = 1;//dp[1][i][i%7]=1;
53. while(t--)
54. {
55. int l,r,m;
56. scanf("%d%d%d", &l,&r,&m);
57. //构造转移矩阵x
58. MATRIX x;
59. rep(j,0,10)
60. {
61. rep(l,0,10)
62. {
63. if(j+l == m) continue;
64. rep(k,0,7)
65. {
66. x.num[mhash(j,k)][mhash(l,(k\*10+l)%7)]++;
67. //dp[i+1][l][(k\*10+l)%7] += dp[i][j][k];
68. }
69. }
70. }
71. rep(j,0,10) x.num[mhash(j,0)][70] = 1;//累加dp[i-1]中%7=0的所有状态
72. x.num[70][70] = 1;//x\*x过程中始终是1,是为了最后乘上st矩阵时ed[70]不为0
73. MATRIX ret = x^(r-1);
74. //初始矩阵乘上x^(r-1)则就是dp[r]的所有结果
75. clc(ed,0);
76. rep(i,0,MAXN)
77. {
78. rep(j,0,MAXN)
79. ed[i] = (ed[i]+st[j]\*ret.num[j][i])%MOD;
80. }
81. LL ans = ed[70];//sum{dp[1 ~ i-1];}
82. rep(j,0,10) ans = (ans+ed[mhash(j,0)])%MOD;//加上当前dp[r]中所有方案数
83. //初始矩阵乘上x^(l-2)则就是dp[l-1]的所有结果
84. ret = x^(l-2);
85. clc(ed,0);
86. rep(i,0,MAXN)
87. {
88. rep(j,0,MAXN)
89. ed[i] = (ed[i]+st[j]\*ret.num[j][i])%MOD;
90. }
91. LL ans2 = ed[70];
92. rep(j,0,10) ans2 = (ans2+ed[mhash(j,0)])%MOD;
93. printf("%lld\n", (ans-ans2+MOD)%MOD);
94. }
95. return 0;
96. }

## 快速查找素数

都是O()，判断单个n是素数：不能被2~中所有**素数**整除

### 直接bool数组判断是否是prime

1. void get\_prime()//没判0和1
2. {
3. clc(isprime,1);isprime[1] = false;
4. int len = sqrt(MAXNUM+0.5);
5. repe(i,2,len)
6. {
7. if(!isprime[i]) continue;
8. for(int j = i\*i; j <= MAXNUM; j += i) isprime[j] = false;
9. }
10. }

### 数组顺序储存每一个素数并在prime[0]保存2~n范围内有几个素数

1. void getprime()
2. {
3. clc(prime,0);
4. int len = sqrt(MAXNUM+0.5);
5. repe(i,2,MAXNUM)
6. {
7. if(!prime[i])prime[++prime[0]] = i;
8. if(i > len || prime[i]) continue;
9. for(int j = i\*i; j <= MAXNUM; j += i) prime[j] = 1;
10. }
11. }

### Miller Rabin素数测试

可以判断64位无符号正整数是否素数，有一定误差但很小，可以通过加大S来减小误差

1. const int S = 8; //随机算法判定次数，一般8~10就够了
2. LL mul\_mod(LL a, LL b, LL MOD)//a\*b对MOD取余(无符号32位内不需要这个)
3. {
4. a %= MOD;
5. LL ans = 0;
6. while(b)
7. {
8. if(b&1)
9. {
10. ans += a;
11. if(ans >= MOD) ans -= MOD;
12. }
13. a <<= 1;
14. if(a >= MOD) a -= MOD;
15. b >>= 1;
16. }
17. return ans;
18. }
19. LL pow\_mod(LL x, LL n, LL MOD)
20. {
21. LL ans = 1;
22. while(n)
23. {
24. if(n&1) ans = mul\_mod(ans,x,MOD);//u\_int32以内直接乘
25. x = mul\_mod(x,x,MOD);
26. n >>= 1;
27. }
28. return ans;
29. }
30. // 通过 a^(n-1)=1(mod n)来判断n是不是素数,n-1 = x\*2^t 中间使用二次判断
31. // 是合数返回true, 不一定是合数返回false
32. bool check(LL a,LL n,LL x,LL t)
33. {
34. LL ans = pow\_mod(a,x,n);
35. LL last = ans;
36. repe(i,1,t)
37. {
38. ans = mul\_mod(ans,ans,n);
39. if(ans == 1 && last != 1 && last != n-1)return true;//合数
40. last = ans;
41. }
42. if(ans != 1)return true;
43. else return false;
44. }
45. // 是素数返回true,(可能是伪素数),不是素数返回false
46. bool Miller\_Rabin(LL n)
47. {
48. if(n < 2)return false;
49. if(2 == n)return true;
50. if(!(n&1))return false;//偶数
51. LL x = n - 1;
52. LL t = 0;
53. while(!(x&1)){x >>= 1; t++;}
54. srand(time(NULL));
55. for(int i = 0;i < S;i++)
56. {
57. LL a = rand()%(n-1) + 1;
58. if(check(a,n,x,t)) return false;
59. }
60. return true;
61. }

## 统计每个数的约数个数

筛法：O(),比筛法略微大点

1. void get\_divisor()
2. {
3. clc(isprime,1);isprime[1] = false;
4. repe(i,2,MAXNUM)
5. {
6. if(!isprime[i]) continue;
7. for(int j = i; j <= MAXNUM; j += i)
8. {
9. int tmp = j;
10. while(tmp%i == 0)
11. {
12. sum[j]++;
13. tmp /= i;
14. }
15. isprime[j] = false;
16. }
17. }
18. }

统计单个数的话直接使用2.6的dfq()

## 分解质因数求最小公倍数(LCM)

LCM可以用GCD求：LCM(a,b) = a/gcd(a,b)\*b

多个数的LCM就是求n次LCM；

但是如果结果非常大需要取模那么不能这么算了；需要用分解质因数：

把所有数分解质因素用cnt[]记录每个素数在单个数字中出现的最多次数，最后把所有分解出的素数乘起来就是所有数字的LCM。

### 单个数字的分解质因数

求cnt[x],是出现过的数中因子为x的单个数最大次数(需要两种筛法预处理，1~n总复杂度O(n\*))：

1. void dfq(int x)
2. {
3. if(used[x]) return;
4. used[x] = true;
5. if(isprime[x])
6. {
7. cnt[x] = max(cnt[x],(short)1);
8. return;
9. }
10. repe(i,1,prime[0])
11. {
12. int tmp = 0,p = prime[i];
13. if(0 == x%p)
14. {
15. while(0 == x%p) x /= p, tmp++;
16. cnt[p] = max(cnt[p],(short)tmp);
17. }
18. else if(isprime[x] || x < prime[i]) break;
19. }
20. if(x > 1) cnt[x] = max(cnt[x],(short)1);
21. }

求最后1~n的LCM：

1. repe(i,2,mx) dqf(i);
2. LL ans = 1;
3. repe(i,2,mx) while(cnt[i]--) ans = ans\*i%MOD;

### 筛法求多个数的分解质因数

复杂度:O(n\*)

小于223,092,870的数最多只有8种(它自己9种)不同素数,小于6,469,693,230最多9种(它10种)

6469693230= 2\*3\*5\*7\*11\*13\*17\*19\*23,所以一般能用筛法的话最多不超过9种不同的分解质因子

1. int prime[MAXN];
2. int p[MAXN][9],plen[MAXN];
3. void get\_dqf()
4. {
5. int len = sqrt(MAXNUM+0.5);
6. repe(i,2,MAXNUM)
7. {
8. if(prime[i]) continue;
9. for(int j = i; j <= MAXNUM; j += i) prime[j] = 1,p[j][plen[j]++] = i;
10. }
11. }

## 互质统计

### 求1~n与n互质个数(欧拉函数)

对一个正整数n,求小于n且与n互质（包括1）的个数。 互质:gcd(a,b)==1

【求1~n中所有数的欧拉phi函数值】O(n\*)

1. void get\_phi()
2. {
3. memset(phi,0,sizeof(phi));
4. phi[1] = 1;
5. for(int i = 2; i <= MAXN; i++)
6. {
7. if(phi[i]) continue;
8. for(int j = i; j <= MAXN; j += i)
9. {
10. if(!phi[j]) phi[j] = j;
11. phi[j] -= phi[j]/i;
12. }
13. }
14. }

【求单个数的欧拉函数】

第一种: O()

1. long long eular(long long n)
2. {
3. long long ans = n;
4. int m = sqrt(n+0.5);
5. for (int i = 2; i <= m; i++)
6. {
7. if (n % i == 0)
8. {
9. ans -= ans / i;
10. while (n % i == 0)
11. n /= i;
12. }
13. }
14. if (n > 1)ans -= ans / n;
15. return ans;
16. }

第二种:素数预处理, O()，单次查询O(K)，K是素数个数(一般还要小于K)

1. /\*MAXNUM是最大n的开根号\*/
2. void getprime()
3. {
4. clc(prime,0);
5. int len = sqrt(MAXNUM+0.5);
6. repe(i,2,MAXNUM)
7. {
8. if(!prime[i])prime[++prime[0]] = i;
9. if(i > len || prime[i]) continue;
10. for(int j = i\*i; j <= MAXNUM; j += i) prime[j] = 1;
11. }
12. }
13. LL eular(LL n)
14. {
15. LL ans = n;
16. int m = sqrt(n+0.5);
17. for(int i = 1; i <= prime[0] && (LL)prime[i]\*prime[i] <= n;i++)
18. {
19. if(n % prime[i] == 0)
20. {
21. ans = ans/prime[i]\*(prime[i]-1);
22. while (n % prime[i] == 0)
23. n /= prime[i];
24. }
25. }
26. if (n > 1) ans = ans / n\*(n-1);
27. return ans;
28. }

### 求x~y中与n互质的个数(容斥原理)

单次复杂度O(),可以用双素数表优化使得整体O(n\*)，详情见[2.5章](#_最小公倍数(LCM)分解质因数解法)

求这些数的和只需要把count()函数用等差数列求和公式即可

1. inline LL count(LL x, LL y, LL c)
2. {
3. return y/c-(x-1)/c;
4. }
5. vector<int> p;
6. LL sloved(LL x, LL y, LL n)//分解质因素后，用容斥原理-二进制(奇数个+ 偶数个-)
7. {
8. p.clear();
9. int mx = sqrt(n+0.5);
10. repe(i,2,mx)
11. {
12. if(n % i == 0)
13. {
14. p.push\_back(i);
15. while(n%i == 0) n /= i;
16. }
17. }
18. if(n > 1) p.push\_back(n);
19. int sz = p.size(),all = 1<<sz;
20. LL ans = 0;
21. rep(s,1,all)
22. {
23. int cnt = 0;
24. LL tmp = 1;
25. rep(i,0,sz)
26. {
27. if(s&(1<<i)) cnt++,tmp \*= p[i];
28. }
29. if(cnt&1) ans += count(x,y,tmp);
30. else ans -= count(x,y,tmp);
31. }
32. return y-x+1-ans;
33. }

## 扩展GCD

【题意】求出X\*a + Y\*b = 1的最小的非负整数x时的一个解 X，Y；如果不存在解输出sorry；

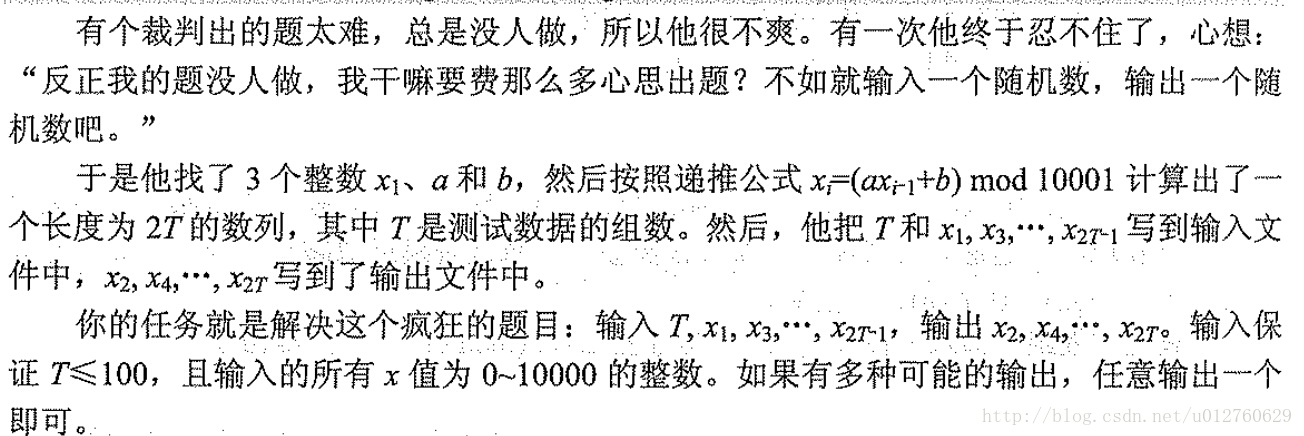
【分析】扩展GCD模版，紫书上的蛮简洁的。

【AC CODE】O()

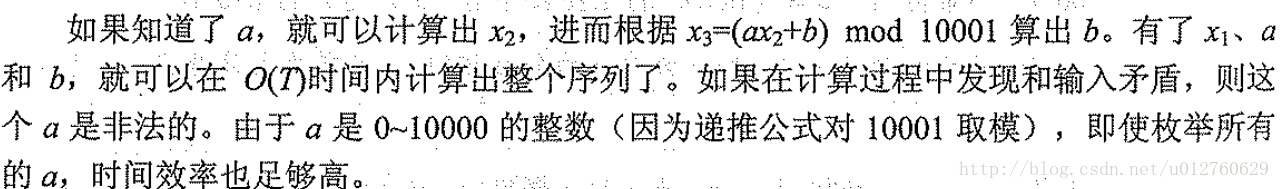
1. #include <cstdio>
2. #include <cstring>
3. #include <cstdlib>
4. typedef long long LL;
5. void e\_gcd(LL a, LL b, LL& d, LL& x, LL& y)
6. {
7. if (!b) d = a, x = 1, y = 0;
8. else e\_gcd(b, a%b, d, y, x), y -= x\*(a / b);
9. }
10. int main()
11. {
12. #ifdef SHY
13. freopen("e:\\1.txt", "r", stdin);
14. #endif
15. LL a, b, d, x, y;
16. while (~scanf("%I64d %I64d%\*c", &a, &b))
17. {
18. e\_gcd(a, b, d, x, y);
19. LL c = 1;
20. if (0 == c%d)
21. {
22. x \*= c / d; //求出x0后要乘上c/d倍,y也一样:y \*= c/d;
23. /\*此时的x,y才是其中一个解,下面可以用通项公式推出所有解\*/
24. LL gx = abs(b/d);//通项公式:(x0+k\*gx,y0-k\*gy)；gx = b/gcd(a,b),gy = a/gcd(a,b)互素,k为任意整数
25. x = (x%gx+gx)%gx;//求出>=0的最小的x,如果要求>=0的最小的y则只需要y(y%gy+gy)%gy;
26. /\*也可以下面这样循环求，效率低
27. if(gx < 0) gx = -gx, gy = - gy;//防止负数，但是这里都是整数其实可以去掉
28. while (x > 0) x -= gx, y += gy;
29. while (x < 0) x += gx, y -= gy;
30. printf("%I64d %I64d\n", x, y);
31. \*/
32. printf("%I64d %I64d\n", x, (c-a\*x)/b);
33. }
34. else puts("sorry");
35. }
36. return 0;
37. }

## 求解模线性方程

【题意】



【分析】



如果不看上面的分析，最先想到的肯定枚举a和b，因为最多只有100个数字，又只有一组数据，UVA3000ms这样肯定可以过，但在HDU1000ms，如果写的戳一点就过不了了(判断合法写好了肯定可以过，本渣渣想不到)，然后只能按照书上的，只枚举a，而b直接用模方程计算出来(因为都是取模运算，不能直接用方程解，一开始这样写的，后来突然发现了)；

那么就剩下怎样计算出b了；

首先可以得出:

x[2]≡a\*x[1]+b (mod 10001); x[3]≡a\*x[2]+b (mod 10001);

然后把x[2]带入第二个等式得到 x[3]≡a^2 \* x[1]+(a+1)b (mod 10001);

移项得(a+1)b≡x[3]-a^2x[1] (mod 10001);

这样其实就得到了一个形如ax≡b (mod n);的模方程了，接下来就是求解模方程了,具体看代码注释；

【AC CODE】0ms(HDU) 15ms(UVA)

1. /\*
2. 枚举a求b，模方程:x[3]≡a^2 \* x[1]+ab+b (mod 10001)
3. 由"a≡b(mod n)使得a-b是n的倍数"，令倍数为y，则 (a+1)b+10001y = x[3]-a^2 \* x[1];
4. 此时可以直接用扩展GCD求出b和y;因为题意说输出任意一个满足的即可，所以只要求出0~10000中一个b
5. \*/
6. #include <cstdio>
7. #include <cstring>
8. #include <cstdlib>
9. #define MAXN 105
10. #define MOD 10001
11. int in[MAXN],ans[MAXN],n;
12. void e\_gcd(int a, int b, int& d, int& x, int& y)
13. {
14. if(!b) d = a, x = 1, y = 0;
15. else e\_gcd(b,a%b,d,y,x), y -= a/b\*x;
16. }
17. void sloved()
18. {
19. for (int a = 0; a <= 10000; a++)
20. {
21. int c = (in[1]-(a\*a%MOD)\*in[0]%MOD+MOD)%MOD,d,b,y;
22. e\_gcd(a+1,-MOD,d,b,y);
23. if(c%d) continue;//无解
24. b \*= c/d;//任意一个b
25. //求出0~10000中最小的b
26. int gb = abs(-MOD/d);
27. b = (b%gb+gb)%gb;
28. for (int j = 0; j < n; j++)
29. {
30. ans[j] = (a\*in[j]%MOD+b)%MOD;
31. if(j == n-1)
32. {
33. for(int i = 0; i < n; i++)
34. printf("%d\n", ans[i]);
35. return;
36. }
37. if((a\*ans[j]%MOD+b)%MOD != in[j+1]) break;
38. }
39. }
40. }
41. int main()
42. {
43. #ifdef SHY
44. freopen("e:\\1.txt","r",stdin);
45. #endif
46. scanf("%d%\*c", &n);
47. for(int i = 0; i < n; i++)
48. scanf("%d%\*c", &in[i]);
49. sloved();
50. return 0;
51. }

## 费马小定理+指数循环节公式

【题意】就是求% 1000000007 (n <= 10^100000)

【分析】由于n有100000位，所以直接快速幂也肯定超时，需要优化。

这里要使用到费马小定理；

费马小定理: gcd(a,m)=1(a,m互质) && m为素数，则≡1 (mod m)

还需要用到一个公式（指数循环节）:≡ (mod M) (n >= Phi(M));Phi(M)在M为素数时为M-1，n>=Phi(M) (n<Phi(M)的话完全可以直接用快速幂了)。

有关这个公式的证明可以参考(数学渣表示看不懂)http://hi.baidu.com/aekdycoin/item/e493adc9a7c0870bad092fd9

这样在这里这个公式可以写成: ≡ (mod m)(直接用这个公式复杂度已经很低了)

然后 因为1000000007是素数，并且gcd(2,1000000007) = 1，所以可以结合费马小定理化简上式：

≡ \* ≡(mod m)

这样求 %1000000007只要求出(%(1000000006)) %1000000007就可以了

【AC CODE(费马小定理+循环节公式) O ()】0ms

1. #include <cstdio>
2. #include <cstring>
3. #include <cmath>
4. typedef long long LL;
5. #define MAXN 100010
6. #define MOD 1000000007
7. char c[MAXN];
8. int pow\_mod(LL a, int n)
9. {
10. LL ans = 1;
11. while(n)
12. {
13. if(n&1) ans = ans\*a%MOD;
14. a = a\*a%MOD;
15. n >>= 1;
16. }
17. return ans;
18. }
19. int main()
20. {
21. #ifdef SHY
22. freopen("e:\\1.txt","r",stdin);
23. #endif
24. while(~scanf("%s%\*c", c))
25. {
26. LL n = 0;
27. for(int i = 0;c[i];i++)
28. n = (n\*10%(MOD-1)+c[i]-'0')%(MOD-1);
29. printf("%d\n", pow\_mod(2,(n-1+MOD-1)%(MOD-1)));
30. }
31. return 0;
32. }

## 高斯消元(求解方程组)

### 浮点数(整数)

可以求类似 a[1][1]\*x[1]+a[1][2]\*x[2]+…+a[1][m]\*x[m] = b[1];

a[2][1]\*x[1]+a[2][2]\*x[2]+…+a[2][m]\*x[m] = b[2];

…

a[n][1]\*x[1]+a[n][2]\*x[2]+…+a[n][m]\*x[m] = b[n];

这样的方程组，已知所有a和b，求x[1]~x[m];

复杂度O(n\*) n是方程数,m是未知数个数

1. const double eps = 1e-8;
2. double a[MAXN][MAXM],x[MAXN];
3. //a[][]方程左边的矩阵,x[]-方程右边的值以及最后的未知数答案
4. inline bool zero(double num){return fabs(num) < eps;}
5. //n-方程个数,m-未知数个数
6. int gauss(int n, int m)//返回-1无解,0有一个解并存在x[],否则返回自由变元个数
7. {
8. int row,col;
9. for(row = 0,col = 0; row < n && col < m; row++,col++)
10. {
11. int mx = row;//找出绝对值最大的作为当前行
12. rep(i,row+1,n) if(fabs(a[i][col]) > fabs(a[mx][col])) mx = i;
13. if(mx != row)
14. {
15. rep(j,col,m) swap(a[mx][j],a[row][j]);
16. swap(x[mx],x[row]);
17. }
18. if(zero(a[row][col]))
19. {
20. row--;
21. continue;//有多个解
22. }
23. rep(i,row+1,n)
24. {
25. if(zero(a[i][col])) continue;
26. double tmp = a[i][col]/a[row][col];
27. a[i][col] = 0;
28. rep(j,col+1,m) a[i][j] -= tmp\*a[row][j];
29. x[i] -= tmp\*x[row];
30. }
31. }
32. rep(i,row,n) if(!zero(x[i])) return -1;//i行所有a为0，而右边值x不为0，无解
33. if(row < m) return m-row;//迭代次数小于未知数个数则有多解(每个自由元可能的状态^(m-row))
34. per(now,m-1,0)//从下往上推,利用已知求未知
35. {
36. rep(j,now+1,m) x[now] -= a[now][j]\*x[j];
37. x[now] /= a[now][now];//输出整数解可能需要(int)(x[i]+0.5)
38. }
39. return 0;
40. }

### 开关问题(异或方程组)

基本和浮点数类似，a[i][j]表示j点能否影响i点(1-能，0-不能)，右边表示各个最终状态，x[]是01解。

方程(有n个)：**(a[1] \* x[1]) xor (a[2] \* x[2]) xor ... xor (a[m] \* x[m]) = 最终状态xor 初始状态**

x[]开始是最终状态xor 初始状态,最后是解

1. int a[MAXN][MAXM],x[MAXN];
2. //a[][]方程左边的矩阵,x[]-方程右边的值以及最后的未知数答案
3. //n-方程个数,m-未知数个数
4. int gauss(int n, int m)//返回-1无解,0有一个解并存在x[],否则返回自由变元个数
5. {
6. int row,col;
7. for(row = 0,col = 0; row < n && col < m; row++,col++)
8. {
9. int mx = row;//找出绝对值最大的作为当前行
10. rep(i,row+1,n) if(abs(a[i][col]) > abs(a[mx][col])) mx = i;
11. if(mx != row)
12. {
13. rep(j,col,m) swap(a[mx][j],a[row][j]);
14. swap(x[mx],x[row]);
15. }
16. if(!a[row][col])
17. {
18. row--;
19. continue;//有多个解
20. }
21. rep(i,row+1,n)
22. {
23. if(!a[i][col]) continue;
24. a[i][col] = 0;
25. rep(j,col+1,m) a[i][j] ^= a[row][j];
26. x[i] ^= x[row];
27. }
28. }
29. rep(i,row,n) if(x[i]) return -1;//i行所有a为0，而右边值x不为0，无解
30. if(row < m) return m-row;//迭代次数小于未知数个数则有多解(每个自由元可能的状态^(m-row))
31. per(now,m-1,0)//从下往上推,利用已知求未知
32. {
33. rep(j,now+1,m) x[now] ^= (a[now][j]&x[j]);
34. x[now] &= a[now][now];
35. }
36. return 0;
37. }

## lucas定理(c(n,m) mod p)

Lucas定理是用来求 mod p，p为**素数**。对于n和m很大的时候可以快速求解.

时间家复杂度O(p\*)，如果n,m是LL，需要套快速乘法,复杂度再乘上一个

1. LL mul\_mod(LL a, LL b, LL MOD)
2. {
3. a %= MOD;
4. LL ans = 0;
5. while(b)
6. {
7. if(b&1)
8. {
9. ans += a;
10. if(ans >= MOD) ans -= MOD;
11. }
12. a <<= 1;
13. if(a >= MOD) a -= MOD;
14. b >>= 1;
15. }
16. return ans;
17. }
18. LL pow\_mod(LL x,LL n, LL MOD)
19. {
20. LL ans = 1;
21. while(n)
22. {
23. if(n&1) ans = mul\_mod(ans,x,MOD);
24. x = mul\_mod(x,x,MOD);
25. n >>= 1;
26. }
27. return ans;
28. }
29. LL getc(LL n,LL m, LL MOD)
30. {
31. if(n<m)return 0;
32. if(m>n-m)m=n-m;
33. LL s1=1,s2=1;
34. rep(i,0,m)
35. {
36. s1=s1\*(n-i)%MOD;
37. s2=s2\*(i+1)%MOD;
38. }
39. return s1\*pow\_mod(s2,MOD-2,MOD)%MOD;
40. }
41. LL lucas(LL n,LL m, LL MOD)
42. {
43. if(m==0)return 1;
44. return getc(n%MOD,m%MOD,MOD)\*lucas(n/MOD,m/MOD,MOD)%MOD;
45. }

## 中国剩余定理(CRT)

### 求x%(m[1]\*m[2]\*…m[n])的值

求x%()的值,分解成方程组有cnt个x%m[i]=a[i]。(m[i]不需要互质)

时间复杂度O(cnt\*)

1. void e\_gcd(LL a, LL b, LL& d, LL& x, LL& y)
2. {
3. if (!b) d = a, x = 1, y = 0;
4. else e\_gcd(b, a%b, d, y, x), y -= x\*(a / b);
5. }
6. LL crt(LL cnt, LL \*m, LL \*a)//cnt是m和a的个数,m[]是除数,a[]是余数
7. {
8. LL aa = a[0],mm = m[0];
9. rep(i,0,cnt)
10. {
11. LL sub = (a[i] - aa),d,x,y;
12. e\_gcd(mm,m[i],d,x,y);
13. if(sub % d) return -1;//不存在
14. LL new\_m = m[i] / d;
15. new\_m = (sub/d\*x%new\_m + new\_m)%new\_m;
16. aa = mm\*new\_m + aa;
17. mm = mm\*m[i]/d;
18. }
19. aa = (aa + mm) % mm;
20. return aa;
21. }

### 求出x的值

其实上面aa的值就是x，只是当aa为0时答案是所有mi的lcm；m不需要互质

1. void e\_gcd(LL a, LL b, LL& d, LL& x, LL& y)
2. {
3. if (!b) d = a, x = 1, y = 0;
4. else e\_gcd(b, a%b, d, y, x), y -= x\*(a / b);
5. }
6. LL gcd(LL a, LL b)
7. {
8. while(b)
9. {
10. a %= b;
11. if(a < b) swap(a,b);
12. }
13. return a;
14. }
15. LL crt(LL cnt, LL \*m, LL \*a)//cnt是m和a的个数,m[]是除数,a[]是余数
16. {
17. LL aa = a[0],mm = m[0],lcm = 1;
18. rep(i,0,cnt) lcm = lcm/gcd(lcm,m[i])\*m[i];
19. rep(i,0,cnt)
20. {
21. LL sub = (a[i] - aa),d,x,y;
22. e\_gcd(mm,m[i],d,x,y);
23. if(sub % d) return -1;//不存在
24. LL new\_m = m[i] / d;
25. new\_m = (sub/d\*x%new\_m + new\_m)%new\_m;
26. aa = mm\*new\_m + aa;
27. mm = mm\*m[i]/d;
28. }
29. if(0 == aa) return lcm;
30. return aa;
31. }

## 博弈

### 巴什博奕（Bash Game）

【问题描述】只有一堆n个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取m个。最后取光者得胜。

【先手必胜策略】必须要让所剩下的物品数量为m+1的倍数，否则就输了

【模版】

1. inline bool Bash(int n, int m)//返回1先取者必胜，0必败
2. {
3. return n%(m+1);
4. }

### 威佐夫博奕（Wythoff Game）

【问题描述】：有两堆各若干个物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

【先手必胜策略】必须每次要让对手面临奇异局势。奇异局势判定模版在下方

【公式模版】：

1. bool Wythoff(int a, int b)//返回1先取者必胜，0必输
2. {
3. /\*两个人如果都采用正确操作，那么面对非奇异局势，先拿者必胜；反之，则后拿者取胜。\*/
4. /\*a,b为两堆石子(a<=b)\*/
5. if(a > b) swap(a,b);
6. int k = b-a;
7. int tmp = k\*(sqrt(5.0)+1.0)/2.0;
8. return tmp != a;//先取者为奇异局势则必输，否则则必胜
9. }

### 尼姆博奕（Nimm Game）

【问题描述】任给N堆石子,两人轮流从任一堆中任取(每次只能取自一堆),取最后一颗石子的人获胜，问先取的人如何获胜？先取者负输出No.先取者胜输出Yes,然后输出先取者第1次取子的所有方法.如果从有a个石子的堆中取若干个后剩下b个后会胜就输出a b；

【先手必胜策略】必须每次要让对手面临奇异局势。奇异局势(就是必输局势)判定：每堆数量异或值为0

【模版】

1. int k = a[0];
2. rep(i,1,n) k ^= a[i];
3. if(k)//k为所有堆的异或值==0必输，!=0必胜
4. {
5. puts("Yes");
6. rep(i,0,n)
7. {
8. //tmp是除了第i堆以外的其他m-1堆的异或值
9. int tmp = k^a[i];
10. /\*tmp也表示在第i堆取完若干个后变成奇异状态第i堆还剩下几个
11. 同时tmp<a[i]才存在方案\*/
12. if(tmp < a[i])
13. {
14. printf("%d %d\n", a[i], tmp);
15. }
16. }
17. }
18. else puts("No");

### 斐波那契博弈(Fibonacci Game)

【问题描述】1堆石子有n个,两人轮流取.先取者第1次可以取任意多个，但不能全部取完.以后每次取的石子数不能超过上次取子数的2倍。取完者胜.先取者负输出"Second win".先取者胜输出"First win".

【先手必胜策略】先手必须要留给对手一个斐波那契数就必胜，否则只要对手采取这个策略先手必输

【模版】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. const int INF = 0x3f3f3f3f, MAXN = 46;
6. int f[MAXN];
7. int main()
8. {
9. #ifdef SHY
10. freopen("e:\\1.txt", "r", stdin);
11. #endif
12. f[0] = 1, f[1] = 1;
13. rep(i,2,MAXN)
14. f[i] = f[i-1]+f[i-2];
15. int n;
16. while(~scanf("%d%\*c", &n) && n)
17. {
18. if(binary\_search(f,f+MAXN,n)) puts("Second win");
19. else puts("First win");
20. }
21. return 0;
22. }

# 数据结构

## Hash表

手写Hash表

1. const int INF = 0x3f3f3f3f, MAXN = 10000+10, MOD = 1000007;
2. struct HASH{
3. int head[MOD], next[MAXN], sum[MAXN], cnt, node[MAXN];
4. void clear(){
5. cnt = 0;
6. clc(head,-1);
7. clc(sum,0);
8. }
9. int hash(int s){
10. if(s < 0) s = -s;
11. return s%MOD;
12. }
13. void insert(int s){
14. int id = hash(s);
15. int u = head[id];
16. while(~u){
17. if(s == node[u])
18. {
19. sum[u]++;
20. return;
21. }
22. u = next[u];
23. }
24. sum[cnt] = 1;
25. node[cnt] = s;
26. next[cnt] = head[id];
27. head[id] = cnt++;
28. }
29. int find(int s){
30. int id = hash(s);
31. int u = head[id];
32. while(~u){
33. if(node[u] == s) return sum[u];
34. u = next[u];
35. }
36. return 0;
37. }
38. }vis;

## 树状数组(fenwick树)

【树状数组模版】

注意：树状数组x必须>0，因为lowbit(0) = 0，会死循环;

【单点更新区间查询】在内修改单点，求出cnt[x] = a[1]~a[x]之和

求区间[a,b]之和只要求出 sum(b)-sum(a-1)即可

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define INF 0x3f3f3f3f
6. typedef long long LL;
7. #define MAXN 100010
8. int cnt[MAXN],mx, a[MAXN];
9. inline int lowbit(int x){return x&-x;}
10. int sum(int x)
11. {
12. int ans = 0;
13. while(x > 0)
14. {
15. ans += cnt[x];
16. x -= lowbit(x);
17. }
18. return ans;
19. }
20. void update(int x, int num)
21. {
22. while(x <= mx)
23. {
24. cnt[x] += num;
25. x += lowbit(x);
26. }
27. }

【区间更新单点查询】在内修改区间a[1]~a[x]覆盖次数，求出单点被覆盖的次数

要**修改**区间[a,b]的覆盖次数+1，可以 update(a,1), update(b+1,-1);**查询**点x就是sum(x)

代码和单点更新区间查询一样。

## 静态RMQ问题ST-在线算法

【ST-在线算法 预处理O(n) 查询O(1)】

用于查询静态区间[a,b]的最值(max换成min就是最小值)

每次合并上来的区间可能会有重复段，所以不能查询区间和

1. void st\_init(int n)
2. {
3. repe(i,1,n) dp[i][0] = a[i];//下标从1开始的
4. for(int j = 1; (1<<j) <= n; j++)
5. {
6. for(int i = 1; i+(1<<j)-1 <= n; i++)//下标从1开始的
7. dp[i][j] = max(dp[i][j-1], dp[i+(1<<(j-1))][j-1]);
8. }
9. }
10. int st(int x, int y)
11. {
12. int k = 0;
13. while((1<<(k+1)) <= y-x+1) k++;
14. return max(dp[x][k], dp[y-(1<<k)+1][k]);
15. }

## 线段树

【线段树单点更新区间查询】查询修改复杂度都是，建树复杂度2n(有2n-1个结点)

合并上来的区间不会有重复段

【离散区间和连续区间的区别】

连续型线段(比如海报，尺上的刻度)，而一般的线段树只对应离散型(每个点都是真实独立存在的)区间上的点，连续型的[x,y]实际长度为y-x,而离散型()的是y-x+1,所以有两种方法解决：

1.改线段树：用(i,i+1)表示叶子，(x,m)和(m,y)表示左右儿子。(这样改不能用id(x,y) = x+y|x!=y了)

2.改线段：把每个连续区间(x,y)，改为(x+1,y)或者(x,y-1)。

### 优化空间写法

x+y|x!=y作为id，id(L,R)为根(0,n-1下标都可以)，只需要2n个节点

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define id(x,y) (x+y|x!=y)
6. #define lc id(x,m)
7. #define rc id(m+1,y)
8. const int INF = 0x3f3f3f3f, MAXN = 100000+10;
9. int a[MAXN];
10. LL sum[MAXN<<1];
11. int p;
12. inline void push\_up(int x, int y, int m)
13. {
14. sum[id(x,y)] = sum[lc]+sum[rc];
15. }
16. void update(int x, int y)
17. {
18. if(x == y)
19. {
20. sum[id(x,y)]++;
21. return;
22. }
23. int m = (x+y)>>1;
24. if(p <= m) update(x,m);
25. else update(m+1,y);
26. push\_up(x,y,m);
27. }
28. int ql, qr;
29. LL cnt;
30. void query(int x, int y)
31. {
32. if(ql <= x && y <= qr)
33. {
34. cnt += sum[id(x,y)];
35. return;
36. }
37. int m = (x+y)>>1;
38. if(ql <= m) query(x,m);
39. if(qr > m) query(m+1,y);
40. }

id(x,y)可获取任意节点在数组的编号

### ZKW线段树(从下往上更新)

【单点更新区间查询】

1. /\* zkw线段树根节点为1,需要空间为最大的M\*2 \*/
2. #define rep(i,a,n) for(int i = a; i < n; i++)
3. #define repe(i,a,n) for(int i = a; i <= n; i++)
4. #define per(i,n,a) for(int i = n; i >= a; i--)
5. #define clc(a,b) memset(a,b,sizeof(a))
6. #define lc u<<1
7. #define rc u<<1|1
8. const int INF = 0x3f3f3f3f, MAXN = 200000, MAXM = 262144;
9. int mx[MAXM<<1],M, a[MAXN];//叶子层的mx[]就是值，省去一个空间
10. void get\_m(int n)//M是叶子层的结点数
11. {
12. M = 1;
13. while(M-2 < n) M <<= 1;//需要建左右两个虚节点,所以少了2个空间
14. }
15. void push\_up(int u)
16. {
17. mx[u] = max(mx[lc],mx[rc]);
18. }
19. void bulid(int n)//线段树叶子层1~n相对于a[]的1~n
20. {
21. get\_m(n);
22. repe(i,1,n) mx[i+M] = a[i];//叶子层
23. mx[M] = mx[M+n+1] = -INF;//叶子边界
24. per(i,M-1,1) push\_up(i);//向上更新每一层
25. }
26. int query(int x, int y)//查询区间[x,y]的最大值
27. {
28. int ans = -INF;
29. //要把[x,y]变成(x,y)
30. for(x += M-1, y += M+1; x^y^1; x >>= 1, y >>= 1)//x^y^1 == (y-x)!=1(即x!=2&&y!=3)
31. {
32. if(~x&1) ans = max(ans,mx[x^1]);//~x&1就是x结点是否为左儿子
33. if(y&1) ans = max(ans,mx[y^1]);//y&1就是y结点是否为右儿子
34. }
35. return ans;
36. }
37. void update(int p, int v)//单点更新a[p]为v
38. {
39. //从下往上修改,u>>1为父亲结点
40. for(mx[p+=M] = v, p >>= 1;p;p >>= 1) push\_up(p);
41. }

### 单调性修改线段树

标记单调性线段树均摊统计,修改区间比v大的数变成v. HDU5306

1. int a[MAXN];
2. inline int id(int x, int y){return x+y|x!=y;}
3. int mx[MAXN<<1],tag[MAXN<<1],cov[MAXN<<1];//mx-最值,tag-lazy标记,cov-当前区间被tag影响的个数
4. LL sum[MAXN<<1];
5. void push\_up(int x, int y, int m)
6. {
7. int u = id(x,y), l = id(x,m), r = id(m+1,y);
8. mx[u] = max(mx[l],mx[r]);
9. sum[u] = sum[l]+sum[r];
10. cov[u] = cov[l]+cov[r];
11. }
12. void down(int x, int y, int v)
13. {
14. int u = id(x,y);
15. if(~tag[u] && tag[u] <= v) return;
16. tag[u] = v;
17. if(y-x+1 != cov[u])
18. {
19. mx[u] = v;
20. sum[u] += (LL)v\*(y-x+1-cov[u]);
21. cov[u] = y-x+1;
22. }
23. }
24. void push\_down(int x, int y, int m)//不需要下方后清空，还有用
25. {
26. int u = id(x,y);
27. if(~tag[u]) down(x,m,tag[u]), down(m+1,y,tag[u]);
28. }
29. void fix(int x, int y, int v)//把>=v的tag全部清除,统计出cov,单次O(n),均摊O(logn)
30. {
31. int u = id(x,y);
32. if(mx[u] <= v) return;//不加就成每次O(n)了
33. if(tag[u] >= v) tag[u] = -1;//封闭结点
34. if(x == y)
35. {
36. sum[u] = mx[u] = ~tag[u]?tag[u]:0, cov[u] = bool(~tag[u]);
37. return;
38. }
39. int m = (x+y)>>1;
40. push\_down(x,y,m);
41. fix(x,m,v);fix(m+1,y,v);
42. push\_up(x,y,m);
43. }
44. void bulid(int x, int y)
45. {
46. if(x == y)
47. {
48. int u = id(x,y);
49. mx[u] = sum[u] = tag[u] = a[x];
50. cov[u] = 1;
51. return;
52. }
53. int m = (x+y)>>1;
54. bulid(x,m);
55. bulid(m+1,y);
56. push\_up(x,y,m);
57. }
58. void update(int x, int y, int ql, int qr, int v)//修改区间[x,y]中所有大于v的数字为v
59. {
60. if(mx[id(x,y)] <= v) return;
61. int m = (x+y)>>1;
62. if(ql <= x && y <= qr)
63. {
64. fix(x,y,v);//统计出子树的cov等
65. if(x == y) sum[id(x,y)] = mx[id(x,y)] = tag[id(x,y)] = v, cov[id(x,y)] = 1;
66. else push\_up(x,y,m);
67. down(x,y,v);
68. return;
69. }
70. push\_down(x,y,m);
71. if(ql <= m) update(x,m,ql,qr,v);
72. if(qr > m) update(m+1,y,ql,qr,v);
73. push\_up(x,y,m);
74. }
75. int get\_mx(int x, int y, int ql, int qr)//区间最值
76. {
77. if(ql <= x && y <= qr) return mx[id(x,y)];
78. int m = (x+y)>>1,ans = -INF;
79. push\_down(x,y,m);
80. if(ql <= m) ans = max(ans,get\_mx(x,m,ql,qr));
81. if(qr > m) ans = max(ans,get\_mx(m+1,y,ql,qr));
82. push\_up(x,y,m);
83. return ans;
84. }
85. LL get\_sum(int x, int y, int ql, int qr)//区间和
86. {
87. if(ql <= x && y <= qr) return sum[id(x,y)];
88. int m = (x+y)>>1;
89. LL ans = 0;
90. push\_down(x,y,m);
91. if(ql <= m) ans += get\_sum(x,m,ql,qr);
92. if(qr > m) ans += get\_sum(m+1,y,ql,qr);
93. push\_up(x,y,m);
94. return ans;
95. }

### 二维线段树

树套树实现的；n\*m矩阵查询修改复杂度log(n)\*log(m)

【单点修改子矩阵查询】

/\*树套树；一维线段树每个结点都是一颗树\*/

1. int a[MAXN][MAXN],n;//这个n是对应我习惯的y的
2. int mx[MAXN<<1][MAXN<<1], mi[MAXN<<1][MAXN<<1];
3. inline int id(int x, int y){return x+y|x!=y;}
4. void push\_up(int x, int y, int m, int u)
5. {
6. mx[u][id(x,y)] = max(mx[u][id(x,m)], mx[u][id(m+1,y)]);
7. mi[u][id(x,y)] = min(mi[u][id(x,m)], mi[u][id(m+1,y)]);
8. }
9. void bulid2(int x, int y, int ux, int uy, bool yz)
10. {
11. if(x == y)
12. {
13. int u = id(ux,uy);
14. /\*只有当两层都是叶子时才是一个点，否则还需要分解\*/
15. if(yz) mx[u][id(x,y)] = mi[u][id(x,y)] = a[ux][x];
16. else
17. {
18. int m = (ux+uy)>>1;
19. mx[u][id(x,y)] = max(mx[id(ux,m)][id(x,y)], mx[id(m+1,uy)][id(x,y)]);
20. mi[u][id(x,y)] = min(mi[id(ux,m)][id(x,y)], mi[id(m+1,uy)][id(x,y)]);
21. }
22. return;
23. }
24. int m = (x+y)>>1;
25. bulid2(x,m,ux,uy,yz);
26. bulid2(m+1,y,ux,uy,yz);
27. push\_up(x,y,m,id(ux,uy));
28. }
29. void bulid(int x, int y)
30. {
31. if(x == y)
32. {
33. bulid2(1,n,x,y,1);
34. return;
35. }
36. int m = (x+y)>>1;
37. bulid(x,m);
38. bulid(m+1,y);
39. bulid2(1,n,x,y,0);
40. }
41. void update2(int x, int y, int p, int v,int ux, int uy, bool yz)
42. {
43. if(x == y)
44. {
45. int u = id(ux,uy);
46. if(yz) mx[u][id(x,y)] = mi[u][id(x,y)] = v;
47. else
48. {
49. int m = (ux+uy)>>1;
50. mx[u][id(x,y)] = max(mx[id(ux,m)][id(x,y)], mx[id(m+1,uy)][id(x,y)]);
51. mi[u][id(x,y)] = min(mi[id(ux,m)][id(x,y)], mi[id(m+1,uy)][id(x,y)]);
52. }
53. return;
54. }
55. int m = (x+y)>>1;
56. if(p <= m) update2(x,m,p,v,ux,uy,yz);
57. else update2(m+1,y,p,v,ux,uy,yz);
58. push\_up(x,y,m,id(ux,uy));
59. }
60. void update(int x, int y, int p, int p2, int v)
61. {
62. if(x == y)
63. {
64. update2(1,n,p2,v,x,y,1);
65. return;
66. }
67. int m = (x+y)>>1;
68. if(p <= m) update(x,m,p,p2,v);
69. else update(m+1,y,p,p2,v);
70. update2(1,n,p2,v,x,y,0);
71. }
72. int qmx,qmi;
73. void query2(int x, int y, int ql, int qr, int u)
74. {
75. if(ql <= x && y <= qr)
76. {
77. qmx = max(qmx,mx[u][id(x,y)]);
78. qmi = min(qmi,mi[u][id(x,y)]);
79. return;
80. }
81. int m = (x+y)>>1;
82. if(ql <= m) query2(x,m,ql,qr,u);
83. if(qr > m) query2(m+1,y,ql,qr,u);
84. }
85. void query(int x, int y, int ql, int qr, int sl, int sr)
86. {
87. if(ql <= x && y <= qr)
88. {
89. query2(1,n,sl,sr,id(x,y));
90. return;
91. }
92. int m = (x+y)>>1;
93. if(ql <= m) query(x,m,ql,qr,sl,sr);
94. if(qr > m) query(m+1,y,ql,qr,sl,sr);
95. }
96. /\*用法\*/
97. while(q--)
98. {
99. int op;
100. scanf("%d", &op);
101. if(1 == op)
102. {
103. int x1,y1,x2,y2;
104. scanf("%d%d%d%d", &x1,&y1,&x2,&y2);
105. qv = 0;
106. query(1,n,x1,x2,y1,y2);
107. if(qv) puts("Yes");
108. else puts("No");
109. }
110. else
111. {
112. int x,y,v;
113. scanf("%d %d %d", &x, &y, &v);
114. update(1,n,x,y,v);
115. }
116. }

## 划分树-(静态区间第k大,k小和)

预处理O(n)—有logn层，每一层访问的结点总数是n ;查询q\*

1. int a[MAXN], sot[MAXN];//a[]是输入序列;sot[]是a[]排序后的序
2. int tree[20][MAXN], tolleft[20][MAXN];
3. //tree[dep][i]表示dep层所有值,tolleft[dep][i]表示dep层从1~i分入左子树的个数
4. void bulid(int x, int y, int dep)
5. {
6. if(x == y) return;
7. int m = (x+y)>>1;
8. int same = m-x+1;//表示等于中间值而且需要被分入左子树的个数
9. repe(i,x,y) if(tree[dep][i] < sot[m]) same--;
10. int lpos = x, rpos = m+1;
11. repe(i,x,y)
12. {
13. if(tree[dep][i] < sot[m])
14. tree[dep+1][lpos++] = tree[dep][i];
15. else if(tree[dep][i] == sot[m] && same > 0)
16. tree[dep+1][lpos++] = tree[dep][i],same--;
17. else tree[dep+1][rpos++] = tree[dep][i];
18. tolleft[dep][i] = tolleft[dep][x-1]+lpos-x;
19. }
20. bulid(x,m,dep+1);
21. bulid(m+1,y,dep+1);
22. }
23. void init(int n)
24. {
25. clc(tolleft,0);
26. repe(i,1,n) tree[0][i] = a[i];
27. sort(sot+1,sot+1+n);
28. bulid(1,n,0);
29. }
30. LL lsum, lnum;//[ql,qr]中[ql,ql+k-1]（前k-1大）的所有数字之和以及个数
31. int query(int x, int y, int ql, int qr, int dep, int k)
32. {
33. if(ql == qr) return tree[dep][ql];
34. int m = (x+y)>>1, cnt = tolleft[dep][qr]-tolleft[dep][ql-1];//cnt- [ql,qr]有几个分在左子树数
35. if(cnt >= k)//左边比k小的数>=k个,往左子树走
36. {
37. //找到[ql,qr]区间在左子树的对应区间
38. int nx = x+tolleft[dep][ql-1]-tolleft[dep][x-1];//下一层的左端点需要在x往右退不在查询区间的[x,ql-1]被分在左子树的个数
39. int ny = nx+cnt-1;//右区间直接加上需要查询区间在左子树的个数
40. return query(x,m,nx,ny,dep+1,k);
41. }
42. //往右子树, 找到[ql,qr]区间在右子树的对应区间
43. int ny = qr+tolleft[dep][y]-tolleft[dep][qr];//右端点需要加上[ql+1,y]被分在左子树的个数(虽然不是要下一层所在范围，但是在右边的放到了左边就要往后推了)
44. int nx = ny-(qr-ql-cnt);//左端点就是右端点减去[ql,qr]中分到右子树的个数
45. lnum += cnt;//累加往左边走的即可
46. lsum += sum[dep][qr]-sum[dep][ql-1];
47. return query(m+1,y,nx,ny,dep+1,k-cnt);
48. }

## 最近公共祖先(LCA)

树上的最近公共祖先

树上任意两点(u,v)的最短路 = dis[u]+dis[v]-2\*dis[LCA(u,v)];dis[i]为根到i的距离

### 离线 Tarjan 算法

复杂度O（n+m+2q）n是点数，m是边数，q是询问数，询问数要建双向边，所以需要2q

1. struct Edge{//邻接表
2. int head[MAXN], tol, next[MAXM], to[MAXM], num[MAXM];//num询问的编号(用于离线)
3. void init(){
4. clc(head,-1);
5. tol = 0;
6. }
7. void add\_edge(int u, int v){
8. next[tol] = head[u], to[tol] = v;
9. head[u] = tol++;
10. }
11. void add\_edge(int u, int v, int id){
12. next[tol] = head[u], to[tol] = v, num[tol] = id;
13. head[u] = tol++;
14. }
15. }g, q;//g-存树, q-所有询问
16. int f[MAXN];
17. int find(int x){return f[x] == x?x:f[x] = find(f[x]);}
18. void bing(int a, int b)//把b所在集合并到a所在集合
19. {
20. int x = find(a), y = find(b);
21. if(x != y) f[y] = x;
22. }
23. bool root[MAXN], vis[MAXN];
24. int ans[MAXQ];//最后答案，顺序输出即可
25. void lca(int u)
26. {
27. vis[u] = true;
28. for(int i = g.head[u]; ~i; i = g.next[i])
29. {
30. int v = g.to[i];
31. if(vis[v]) continue;
32. lca(v);
33. bing(u,v);//u,v不能换,否则要加一个数组记录每个结点当前所在根
34. }
35. /\*找出当前结点有关的询问\*/
36. for(int i = q.head[u]; ~i; i = q.next[i])
37. {
38. int v = q.to[i];
39. if(vis[v])//访问过了说明v在u子树中,或者v是u的祖先，则这时候v所在的集合编号(也是最近的祖先)就是LCA
40. ans[q.num[i]] = find(v);
41. }
42. }

### 在线转RMQ算法

预处理O(n),查询O(1)

可以把ST改成线段树，复杂度变成预处理O(n+m)，查询O()

1. /\*LCA在线算法核心思想:
2. 从树的根节点开始进行深度优先搜索，每次经过某一个点——无论是从它的父亲节点进入这个点，还是从它的儿子节点返回这个点，
3. 都按顺序记录下来。这样，就把一棵树转换成了一个数组。而找到树上两个节点的最近公共祖先，
4. 无非就是找到这两个节点第一次出现在数组中的位置所囊括的一段区间中深度最小的那个点。变成了RMQ问题
5. \*/
6. int head[MAXN], tol,nxt[MAXM],to[MAXM],sum[MAXN];
7. inline void add\_edge(int u, int v)
8. {
9. nxt[tol] = head[u], to[tol] = v;
10. head[u] = tol++;
11. }
12. int cnt, ft[MAXN], d[MAXN<<1],num[MAXN<<1];//d数组点数为2\*n-1
13. //ft[i]是i点第一次在d中出现的下标,d[]是所有点进入或者返回到时都加入的点的深度(最多有2\*n-1个)，num是d对应的点编号
14. void dfs(int u, int deep, int fa)//找到每个点进入和返回时的所有ft,d,num
15. {
16. ft[u] = cnt;
17. d[cnt] = deep, num[cnt++] = u;
18. for(int i = head[u]; ~i; i = nxt[i])
19. {
20. int v = to[i];
21. if(fa == v) continue;
22. dfs(v,deep+1,u);
23. d[cnt] = deep, num[cnt++] = u;
24. }
25. }
26. int dp[MAXN<<1][21], dp\_num[MAXN<<1][21];//dp\_num记录dp对应的点的编号
27. void st\_init()
28. {
29. rep(i,0,cnt) dp[i][0] = d[i], dp\_num[i][0] = num[i];
30. for(int j = 1; (1<<j) <= cnt; j++)
31. {
32. for(int i = 0; i+(1<<j)-1 < cnt; i++)
33. {
34. if(dp[i][j-1] < dp[i+(1<<(j-1))][j-1])
35. {
36. dp[i][j] = dp[i][j-1];
37. dp\_num[i][j] = dp\_num[i][j-1];
38. }
39. else
40. {
41. dp[i][j] = dp[i+(1<<(j-1))][j-1];
42. dp\_num[i][j] = dp\_num[i+(1<<(j-1))][j-1];
43. }
44. }
45. }
46. }
47. int st\_query(int x, int y)//查询d[x~y]中最小值对应的点编号
48. {
49. if(x > y) swap(x,y);
50. int k = 0;
51. while((1<<(k+1)) <= y-x+1) k++;
52. if(dp[x][k] <= dp[y-(1<<k)+1][k]) return dp\_num[x][k];
53. return dp\_num[y-(1<<k)+1][k];
54. }
55. /\*LCA初始化和查询\*/
56. void lca\_init(int rt)//初始化O(n\*log(n))
57. {
58. cnt = 0;
59. dfs(rt,0,-1);
60. st\_init();
61. }
62. int lca\_query(int x, int y)//x,y的LCA, O(1)
63. {
64. return st\_query(ft[x],ft[y]);
65. }

## 树上DFS序和LCA的深度序区别

【定义】

DFS序：每次入栈该点记录，所有儿子返回后再记录一下，序列长度=点数\*2,即2n。

LCA深度序：每次入栈该点记录，每个儿子返回后都要记录下该点，序列长度=边数\*2+1(由于是树，所以是2n-1)

【用途】

DFS序：可以把树转化为序列(从根开始的序列)；用线段树等记录值，那么第一次出现在序列中的点+1(即插入)，第二次出现-1(即删除)，则从根到该点经过的所有点就是序列1~该点第一次在序列出现的位置。不能直接求出任意两点之间的所有点，可能会缺少祖先结点，但是如果两点属于祖先-子孙关系（即同一条链）则两点第一次出现位置之间的序列就是这两点路径。可以配合LCA最短路求出树上任意两点间所有路径的值。

LCA深度序：现在只发现用来求LCA；即记录任意两个点之间深度最值。不能直接算出任意两点之间的所有点，可能会多出这两点的某些子孙点。

## 主席树(函数式/可持久化线段树)

这里的第k大都是从小到大(第k小)

### 静态区间第k大-HDU2665

每棵树维护区间[1,i]的数字出现次数的前缀和 O(n\*+q\*)

1. /\*主席树-求区间第k大\*/
2. const int INF = 0x3f3f3f3f, MAXN = 100000+10, M = MAXN\*30;//M = 2\*N\*log(N)
3. int l[M],r[M],sum[M], cnt;//sum是出现次数,cnt占用结点计数
4. void bulid(int &u, int x, int y)//建空树0
5. {
6. sum[cnt] = 0;u = cnt++;
7. if(x == y) return;
8. int m = (x+y)>>1;
9. bulid(l[u],x,m);
10. bulid(r[u],m+1,y);
11. }
12. void insert(int &u, int x, int y, int p)
13. {
14. l[cnt] = l[u],r[cnt] = r[u],sum[cnt] = sum[u];//复制老版本信息
15. sum[u=cnt++]++;//新版本计数+1
16. if(x == y) return;
17. int m = (x+y)>>1;
18. if(p <= m) insert(l[u],x,m,p);
19. else insert(r[u],m+1,y,p);
20. }
21. int a[MAXN],sot[MAXN],n, root[MAXN];//a[]原序列,sot[]排序后离散化,root[i]表示[1,i]每个数出现的次数的线段树
22. void init()
23. {
24. cnt = 0;
25. bulid(root[0],1,n);//第0颗空树，这里初始都是0，所以可以只用一个结点0,初始cnt = 1,因为lc[0]=0
26. sort(sot+1,sot+1+n);
27. repe(i,1,n)
28. {
29. root[i] = root[i-1];//借用第i-1颗树
30. insert(root[i],1,n,lower\_bound(sot+1,sot+1+n,a[i])-sot);
31. }
32. }
33. int query(int a, int b, int x, int y, int k)//查询区间[a+1,b]的第k大数
34. {
35. if(x == y) return x;
36. int d = sum[l[b]]-sum[l[a]];
37. int m = (x+y)>>1;
38. if(d >= k) return query(l[a],l[b],x,m,k);
39. return query(r[a],r[b],m+1,y,k-d);
40. }
41. int main()
42. {
43. #ifdef SHY
44. freopen("d:\\1.txt", "r", stdin);
45. #endif
46. int t;
47. scanf("%d", &t);
48. while(t--)
49. {
50. int q;
51. scanf("%d %d", &n, &q);
52. repe(i,1,n) scanf("%d",&a[i]), sot[i] = a[i];
53. init();
54. while(q--)
55. {
56. int x,y,k;
57. scanf("%d%d%d",&x,&y,&k);
58. printf("%d\n", sot[query(root[x-1],root[y],1,n,k)]);
59. }
60. }
61. return 0;
62. }

### 单点修改的区间第K大-ZOJ2112

预处理用n\*建立静态前缀和主席树，用树状数组套主席树，每次修改不存在的结点再新建，修改

1. /\*离线+离散处理的 O(logn\*n + q\*logn\*logn)\*/
2. const int INF = 0x3f3f3f3f, MAXN = 50000+10, MAXQ = 10000+10,MAXM = MAXN\*30;
3. struct IN{
4. bool f;
5. int a,b,k;
6. }in[MAXQ];
7. int lc[MAXM], rc[MAXM], sum[MAXM], tol;//主席树内存池
8. int num[MAXN+MAXQ], sot[MAXN+MAXQ], cnt;//离散
9. int rt[MAXN<<1], a[MAXN], n;//rt[]保存各个版本的线段树根,rt[1~n]保存树状数组套的主席树,后面的保存初始的n个前缀主席树
10. inline int mhash(int v){return lower\_bound(sot+1,sot+1+cnt,v)-sot;}
11. void init\_hash(int cc)
12. {
13. sort(num+1,num+1+cc);
14. sot[1] = num[1];
15. cnt = 1;
16. repe(i,2,cc) if(num[i] != num[i-1]) sot[++cnt] = num[i];
17. }
18. inline int newnode(int \_lc, int \_rc, int \_sum)
19. {
20. lc[tol] = \_lc, rc[tol] = \_rc, sum[tol] = \_sum;
21. return tol++;
22. }
23. void bulid(int &u, int x, int y, int p)//在前一个版本基础上添加一个新的结点v,成为一棵新树
24. {
25. u = newnode(lc[u],rc[u],sum[u]+1);
26. if(x == y) return;
27. int m = (x+y)>>1;
28. if(p <= m) bulid(lc[u],x,m,p);
29. else bulid(rc[u],m+1,y,p);
30. }
31. void t\_update(int &u, int x, int y, int p, int v)//树状数组中每个结点所套的主席树,更新到才新建，省内存
32. {
33. if(!u) u = newnode(0,0,0);
34. sum[u] += v;
35. if(x == y) return;
36. int m = (x+y)>>1;
37. if(p <= m) t\_update(lc[u],x,m,p,v);
38. else t\_update(rc[u],m+1,y,p,v);
39. }
40. /\*树状数组记录区间[1,n]每个数字出现的次数,次数用值域主席树记录\*/
41. inline int lowbit(int x){return x&-x;}
42. void add(int x, int v)//树状数组添加或删除元素a[x]
43. {
44. int p = a[x];
45. while(x <= n)
46. {
47. t\_update(rt[x],1,cnt,p,v);
48. x += lowbit(x);
49. }
50. }
51. int tx[50], ty[50], len[2];//查询用到的所有树
52. int t\_query(int x, int y, int k)
53. {
54. if(x == y) return x;
55. int d = 0;
56. rep(i,0,len[1]) d += sum[lc[ty[i]]];
57. rep(i,0,len[0]) d -= sum[lc[tx[i]]];
58. int m = (x+y)>>1;
59. if(d >= k)
60. {
61. rep(i,0,len[1]) ty[i] = lc[ty[i]];
62. rep(i,0,len[0]) tx[i] = lc[tx[i]];
63. return t\_query(x,m,k);
64. }
65. rep(i,0,len[1]) ty[i] = rc[ty[i]];
66. rep(i,0,len[0]) tx[i] = rc[tx[i]];
67. return t\_query(m+1,y,k-d);
68. }
69. /\*====== 调用操作 ======\*/
70. void init(int cc)//logn\*n新建初始前缀树
71. {
72. init\_hash(cc);
73. repe(i,1,n) a[i] = mhash(a[i]);//把所有a[i]变成离散值
74. tol = 1;
75. clc(rt,0);
76. rt[0] = sum[0] = lc[0] = rc[0] = 0;//只需要用一个结点代表第0颗树
77. repe(i,1,n)
78. {
79. rt[i+n] = rt[i+n-1];
80. bulid(rt[i+n],1,cnt,a[i]);
81. }
82. }
83. void update(int p, int v)//更新a[p] -> v
84. {
85. add(p,-1);
86. a[p] = mhash(v);
87. add(p,1);
88. }
89. int query(int x, int y, int k)//查询区间[x,y]的第k大元素
90. {
91. /\*把所有表示区间y的树减去区间x-1的树存起来查询\*/
92. len[0] = len[1] = 1;
93. tx[0] = rt[x-1?x-1+n:0], ty[0] = rt[y+n];//先把初始的一棵树加进去
94. for(int i = x-1; i > 0; i -= lowbit(i)) tx[len[0]++] = rt[i];//树状数组求区间[x,y]和就是sum(y)-sum(x-1)则，把多个这样的点一起在主席树中算就可以了
95. for(int i = y; i > 0; i -= lowbit(i)) ty[len[1]++] = rt[i];
96. return t\_query(1,cnt,k);
97. }
98. int main()
99. {
100. #ifdef SHY
101. freopen("d:\\1.txt", "r", stdin);
102. #endif
103. int t;
104. scanf("%d", &t);
105. while(t--)
106. {
107. int q;
108. scanf("%d %d", &n, &q);
109. int cc = 0;//所有值没去重的数量
110. repe(i,1,n) scanf("%d", &a[i]), num[++cc] = a[i];
111. char op[10];
112. rep(i,0,q)
113. {
114. scanf("%s", op);
115. if('Q' == op[0])
116. {
117. in[i].f = 0;
118. scanf("%d %d %d",&in[i].a, &in[i].b, &in[i].k);
119. }
120. else
121. {
122. in[i].f = 1;
123. scanf("%d %d", &in[i].a, &in[i].b);
124. num[++cc] = in[i].b;
125. }
126. }
127. init(cc);
128. rep(i,0,q)
129. {
130. if(!in[i].f) printf("%d\n", sot[query(in[i].a,in[i].b,in[i].k)]);
131. else update(in[i].a,in[i].b);
132. }
133. }
134. return 0;
135. }

### 树上第K大-bzoj1146

在每个点建一棵到根的前缀可持久化线段树，则查询任意两点间(x,y)的第k大可以在主席树T=a[x]+a[y]-2\*[LCA(x,y)];由于是点所以变成T = a[x]+a[y]-a[LAC(x,y)]-a[fa[LCA(x,y)]];这个查询可以在主席树带入这四个点实现O()的查询。

如果是可修改的，用DFS序(和计算LCA的不同)：对于每一个点，在其入栈的时候加上，在出栈的时候减去。那么如果查询一个点入栈位置的前缀和，就是它到根的路径的信息。这样就转变成了区间可修改第k大+静态树上K大，查询时带入树状数组表示的值树按照这个公式T=a[x]+a[y]-2\*[LCA(x,y)]计算即可。查询复杂度O(),可以配合静态n\*预处理加上另外维护修改可以把空间复杂度降到n\*+n\*^2

下面是静态第k大的代码：

1. const int INF = 0x3f3f3f3f, MAXN = 100000+10, MAXM = MAXN\*30;
2. int lc[MAXM],rc[MAXM],sum[MAXM],tol;
3. int rt[MAXN], a[MAXN],num[MAXN],sot[MAXN],cnt;
4. int head[MAXN], nxt[MAXN<<1], to[MAXN<<1],e;
6. inline int mhash(int v){return lower\_bound(sot+1,sot+1+cnt,v)-sot;}
7. inline void add\_edge(int u, int v)
8. {
9. nxt[e] = head[u], to[e] = v;
10. head[u] = e++;
11. }
12. void insert(int &u, int x, int y, int p)
13. {
14. lc[++tol] = lc[u], rc[tol] = rc[u], sum[tol] = sum[u]+1;
15. u = tol;
16. if(x == y) return;
17. int m = (x+y)>>1;
18. if(p <= m) insert(lc[u],x,m,p);
19. else insert(rc[u],m+1,y,p);
20. }
21. int all,ft[MAXN],d[MAXN<<1],id[MAXN<<1], fa[MAXN];
22. void dfs(int u, int dep)//建所有根到点的前缀和树以及LCA序列初始化
23. {
24. rt[u] = rt[fa[u]];
25. insert(rt[u],1,cnt,mhash(a[u]));
26. ft[u] = all;
27. d[all] = dep, id[all++] = u;
28. for(int i = head[u]; ~i; i = nxt[i])
29. {
30. int v = to[i];
31. if(fa[u] == v) continue;
32. fa[v] = u;
33. dfs(v,dep+1);
34. d[all] = dep, id[all++] = u;
35. }
36. }
37. int dp[MAXN<<1][20], dp\_num[MAXN<<1][20];
38. void st\_init()
39. {
40. rep(i,0,all) dp[i][0] = d[i], dp\_num[i][0] = id[i];
41. for(int j = 1; (1<<j) <= all; j++)
42. {
43. for(int i = 0; i+(1<<j)-1 < all; i++)
44. {
45. if(dp[i][j-1] < dp[i+(1<<(j-1))][j-1])
46. {
47. dp[i][j] = dp[i][j-1];
48. dp\_num[i][j] = dp\_num[i][j-1];
49. }
50. else
51. {
52. dp[i][j] = dp[i+(1<<(j-1))][j-1];
53. dp\_num[i][j] = dp\_num[i+(1<<(j-1))][j-1];
54. }
55. }
56. }
57. }
58. int st\_query(int x, int y)//查询d[x~y]中最小值对应的点编号
59. {
60. if(x > y) swap(x,y);
61. int k = 0;
62. while((1<<(k+1)) <= y-x+1) k++;
63. if(dp[x][k] <= dp[y-(1<<k)+1][k]) return dp\_num[x][k];
64. return dp\_num[y-(1<<k)+1][k];
65. }
66. /\*LCA初始化和查询\*/
67. void lca\_init(int rt)//初始化O(n\*log(n))
68. {
69. tol = all = 0;
70. fa[rt] = 0;
71. dfs(rt,1);
72. st\_init();
73. }
74. int lca\_query(int x, int y)//x,y的LCA, O(1)
75. {
76. return st\_query(ft[x],ft[y]);
77. }
78. int query(int a, int b, int lca, int lcaf, int x, int y, int k)
79. {
80. if(x == y) return x;
81. int m = (x+y)>>1, ls = sum[lc[a]]+sum[lc[b]]-sum[lc[lca]]-sum[lc[lcaf]];
82. if(ls >= k) return query(lc[a],lc[b],lc[lca],lc[lcaf],x,m,k);
83. return query(rc[a],rc[b],rc[lca],rc[lcaf],m+1,y,k-ls);
84. }
86. int main()
87. {
88. #ifdef SHY
89. freopen("d:\\1.txt", "r", stdin);
90. #endif
91. int n,q;
92. scanf("%d %d", &n, &q);
93. repe(i,1,n) scanf("%d", &a[i]), num[i] = a[i];
94. sort(num+1,num+1+n);
95. sot[cnt=1] = num[1];
96. repe(i,2,n) if(num[i] != num[i-1]) sot[++cnt] = num[i];
97. e = 0;
98. clc(head,-1);
99. rep(i,1,n)
100. {
101. int u,v;
102. scanf("%d %d", &u, &v);
103. add\_edge(u,v);
104. add\_edge(v,u);
105. }
106. lca\_init(1);
107. int last = 0;
108. while(q--)
109. {
110. int u,v,k;
111. scanf("%d %d %d", &u, &v, &k);
112. u ^= last;
113. int lca = lca\_query(u,v);
114. int ans = sot[query(rt[u],rt[v],rt[lca],rt[fa[lca]],1,cnt,k)];
115. printf("%d", ans);
116. last = ans;
117. if(q) putchar('\n');
118. }
119. return 0;
120. }

## 树链剖分

树链剖分可以解决树中任意两点的最值查询，sum查询，以及点值修改，两点之间点值批量修改等；

把树剖分成重链(各个点儿子最多的点连成的)和轻链(其实只有一条边)；

然后把各个重链轻链连续映射到线段树或者其他维护区间的数据结构上面

剖分后有两个性质保证了每次查询修改复杂度保持在(本身logn在乘区间维护数据结构的)：

【性质1】：如果(u,v)为轻边，则sz[v] \* 2 < sz[u]；

【性质2】：从根到某一点的路径上轻链、重链的个数都不大于

### 点权(线段树)

1. /\*树链剖分-点权O(q\*logn^2) \*/
2. int head[MAXN], tol, nxt[MAXM], to[MAXM];
3. int a[MAXN];//每个点初始权值
4. inline void add\_edge(int u, int v)
5. {
6. nxt[tol] = head[u], to[tol] = v;
7. head[u] = tol++;
8. }
9. struct TCP{
10. private:
11. int fa[MAXN];//fa[u]表示u的父亲
12. int dep[MAXN];//dep[u]表示u的深度
13. int sz[MAXN];//sz[u]表示以u为根的子树有几个结点
14. int son[MAXN];//son[u]表示u的所有儿子结点中sz[]最大的那个点(重儿子)
15. int top[MAXN];//top[u]表示u所在的链的顶端节点
16. int tree[MAXN];//tree[u]表示节点u在线段树中的编号
17. int pre[MAXN];//pre[x]表示线段树中编号为x的点在树中的点(和tree相反)
18. int cnt;//线段树中点的数量(其实就是n)
19. void dfs1(int u)//预处理fa[],dep[],sz[],son[]
20. {
21. int num = 0;
22. sz[u] = 1;
23. for(int i = head[u]; ~i; i = nxt[i])
24. {
25. int v = to[i];
26. if(v == fa[u]) continue;
27. fa[v] = u;dep[v] = dep[u]+1;
28. dfs1(v);
29. if(sz[v] > num) num = sz[v], son[u] = v;
30. sz[u] += sz[v];
31. }
32. }
33. void dfs2(int u, int num)//预处理top[],tree[],pre[]
34. {
35. tree[u] = cnt, pre[cnt++] = u,top[u] = num;
36. if(-1 == son[u]) return;
37. dfs2(son[u],num);//必须点递归重儿子，否则重链在线段树的编号会不连续
38. for(int i = head[u]; ~i; i = nxt[i])
39. {
40. int v = to[i];
41. if(v != fa[u] && son[u] != v) dfs2(v,v);
42. }
43. }
44. /\*--线段树(区间更新单点查询)--s\*/
45. int add[MAXN<<1];
46. inline int id(int x, int y){return x+y|x!=y;}
47. void bulid(int x, int y)
48. {
49. if(x == y)
50. {
51. add[id(x,y)] = a[pre[x]];
52. return;
53. }
54. int m = (x+y)>>1;
55. bulid(x,m);
56. bulid(m+1,y);
57. }
58. void push\_down(int x, int y, int m)
59. {
60. int u = id(x,y);
61. if(add[u])
62. {
63. int l = id(x,m), r = id(m+1,y);
64. add[l] += add[u];add[r] += add[u];
65. add[u] = 0;
66. }
67. }
68. void update(int x, int y, int ql, int qr, int v)
69. {
70. if(ql <= x && y <= qr)
71. {
72. add[id(x,y)] += v;
73. return;
74. }
75. int m = (x+y)>>1;
76. push\_down(x,y,m);
77. if(ql <= m) update(x,m,ql,qr,v);
78. if(qr > m) update(m+1,y,ql,qr,v);
79. }
80. int query(int x, int y, int p)
81. {
82. if(x == y) return add[id(x,y)];
83. int m = (x+y)>>1;
84. push\_down(x,y,m);
85. if(p <= m) return query(x,m,p);
86. return query(m+1,y,p);
87. }
88. /\*--线段树--e\*/
89. public:
90. void init(int rt)
91. {
92. clc(son,-1);
93. dep[rt] = 0, fa[rt] = -1;
94. dfs1(rt);
95. cnt = 0;
96. dfs2(rt,rt);
97. clc(add,0);
98. bulid(0,cnt-1);
99. }
100. void tcp\_update(int x, int y, int v)//树上点x~y之间的权值都加v
101. {
102. int f1 = top[x], f2 = top[y];
103. while(f1 != f2)
104. {
105. if(dep[f1] < dep[f2]) swap(f1,f2), swap(x,y);
106. update(0,cnt-1,tree[f1],tree[x],v);
107. x = fa[f1], f1 = top[x];
108. }
109. if(dep[x] > dep[y]) swap(x,y);
110. update(0,cnt-1,tree[x],tree[y],v);
111. }
112. int tcp\_query(int x)//查询点x的权值
113. {
114. return query(0,cnt-1,tree[x]);
115. }
116. }tcp;

### 边权(线段树)

把边转换为以深度较深的点放进线段树

1. /\*树链剖分-边权O(q\*logn^2) \*/
2. int head[MAXN], tol, nxt[MAXM<<1],to[MAXM<<1];//树中边-邻接表
3. int cost[MAXM], e[MAXM][2];//按照输入顺序的边(边的编号1~n-1,无向边)
4. inline void add\_edge(int u, int v)
5. {
6. nxt[tol] = head[u], to[tol] = v;
7. head[u] = tol++;
8. }
9. struct TCP{
10. private:
11. int fa[MAXN];//fa[u]表示u的父亲
12. int dep[MAXN];//dep[u]表示u的深度
13. int sz[MAXN];//sz[u]表示以u为根的子树有几个结点
14. int son[MAXN];//son[u]表示u的所有儿子结点中sz[]最大的那个点(重儿子)
15. int top[MAXN];//top[u]表示u所在的链的顶端节点
16. int tree[MAXN];//tree[u]表示节点u和他父亲的连边在线段树中的编号
17. int pre[MAXN];//pre[x]和tree相反
18. int bnum[MAXN];//每个边转化为深度较深的点之后对应的边号
19. int cnt;//线段树中点的数量(其实就是n)
20. void dfs1(int u)//预处理fa[],dep[],sz[],son[]
21. {
22. int num = 0;
23. sz[u] = 1;
24. for(int i = head[u]; ~i; i = nxt[i])
25. {
26. int v = to[i];
27. if(v == fa[u]) continue;
28. fa[v] = u;dep[v] = dep[u]+1;
29. dfs1(v);
30. if(sz[v] > num) num = sz[v], son[u] = v;
31. sz[u] += sz[v];
32. }
33. }
34. void dfs2(int u, int num)//预处理top[],tree[],pre[]
35. {
36. tree[u] = cnt, pre[cnt++] = u,top[u] = num;
37. if(-1 == son[u]) return;
38. dfs2(son[u],num);//必须点递归重儿子，否则重链在线段树的编号会不连续
39. for(int i = head[u]; ~i; i = nxt[i])
40. {
41. int v = to[i];
42. if(v != fa[u] && son[u] != v) dfs2(v,v);
43. }
44. }
45. /\*--线段树--s\*/
46. int sum[MAXN<<1];
47. inline int id(int x, int y){return x+y|x!=y;}
48. inline void push\_up(int x, int y, int m){
49. sum[id(x,y)] = sum[id(x,m)]+sum[id(m+1,y)];
50. }
51. void bulid(int x, int y)
52. {
53. if(x == y)
54. {
55. sum[id(x,y)] = cost[bnum[pre[x]]];
56. return;
57. }
58. int m = (x+y)>>1;
59. bulid(x,m);
60. bulid(m+1,y);
61. push\_up(x,y,m);
62. }
63. void update(int x, int y, int p, int v)
64. {
65. if(x == y)
66. {
67. sum[id(x,y)] = v;
68. return;
69. }
70. int m = (x+y)>>1;
71. if(p <= m) update(x,m,p,v);
72. else update(m+1,y,p,v);
73. push\_up(x,y,m);
74. }
75. int query(int x, int y, int ql, int qr)
76. {
77. if(ql <= x && y <= qr) return sum[id(x,y)];
78. int m = (x+y)>>1, ans = 0;
79. if(ql <= m) ans += query(x,m,ql,qr);
80. if(qr > m) ans += query(m+1,y,ql,qr);
81. return ans;
82. }
83. /\*--线段树--e\*/
84. public:
85. void init(int rt)
86. {
87. clc(son,-1);
88. dep[rt] = 0, fa[rt] = -1;
89. dfs1(rt);
90. cnt = 0;
91. dfs2(rt,rt);
92. clc(sum,0);
93. rep(i,1,cnt)//所有边
94. {
95. if(dep[e[i][0]] > dep[e[i][1]]) swap(e[i][0],e[i][1]);//每条边在树中下面的点作为标识(边转点)
96. bnum[e[i][1]] = i;
97. }
98. bulid(0,cnt-1);
99. }
100. void tcp\_update(int c, int v)//把第c条边权修改为v
101. {
102. update(0,cnt-1,tree[e[c][1]],v);//直接用e[c][1](每条边深度较深的点)作为该边的标识
103. }
104. int tcp\_sum(int x, int y)//求点x~y之间所有边权和
105. {
106. int f1 = top[x], f2 = top[y], ans = 0;
107. while(f1 != f2)
108. {
109. if(dep[f1] < dep[f2]) swap(f1,f2), swap(x,y);
110. ans += query(0,cnt-1,tree[f1], tree[x]);
111. x = fa[f1], f1 = top[x];
112. }
113. if(x == y) return ans;//由于是边，点相同没有权值
114. if(dep[x] > dep[y]) swap(x,y);
115. ans += query(0,cnt-1,tree[son[x]],tree[y]);
116. return ans;
117. }
118. }tcp;

### 有序合并

有些问题要求树链合并的时候需要有序，即按照两点(x,y)路径走(分别从x->lca和y->lca注意这里两个方向相反)

1. /\*树链剖分左右按顺序合并\*/
2. int tcp\_query(int x, int y)
3. {
4. NODE ans,ax,ay;
5. int f1 = top[x],f2 = top[y];
6. while(f1 != f2)
7. {
8. if(dep[top[x]] < dep[top[y]])//LCA(x,y)右边,从y到LCA方向
9. {
10. merge(ay,query(0,cnt-1,tree[top[y]],tree[y],1),ay,1);
11. y = fa[f2];f2 = top[y];
12. }
13. else//LCA(x,y)左边,从x到LCA方向
14. {
15. merge(ax,query(0,cnt-1,tree[top[x]],tree[x],0),ax,0);
16. x = fa[f1], f1 = top[x];
17. }
18. }
19. if(dep[x] < dep[y]) merge(ay,query(0,cnt-1,tree[x],tree[y],1),ay,1);
20. else merge(ax,query(0,cnt-1,tree[y],tree[x],0),ax,0);
21. merge(ans,ax,ay,1);//把左右总的合并起来
22. return ans.ans;
23. }

## 扫描线(线段树辅助)

由于这里的线段树删除区间都是出现过的区间并且查询只有根(两个条件缺一不可)，所以不需要lazy向下传递，当然传了也不会错，只是稍微麻烦点。

### 矩形面积并

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long LL;
4. #define rep(i,a,n) for(int i = a; i < n; i++)
5. #define repe(i,a,n) for(int i = a; i <= n; i++)
6. #define per(i,n,a) for(int i = n; i >= a; i--)
7. #define clc(a,b) memset(a,b,sizeof(a))
8. /\*扫描线+线段树辅助
9. 把矩形的上下边存起来，并且标记为-1和1;
10. 从下往上每次扫描到一条边的时候计算横坐标覆盖的x长度(用线段树维护)\*当前边和下一条边的高度差
11. 就是不重复分一小块面积，全部扫描完就是所有矩形面积的并
12. 线段树记录的是点x~y+1的并，而线段少了个端点,所以加入删除线段x,y的话对应线段树操作x~y-1
13. \*/
14. typedef LL type;
15. const int INF = 0x3f3f3f3f, MAXN = 200000+10;
16. struct NODE{//扫描线段
17. type l,r,h;
18. int v;
19. bool operator<(const NODE &t) const{
20. return h < t.h;
21. }
22. }p[MAXN];
23. int tol;//线段数量
24. type pos[MAXN], tmp[MAXN];//离散x坐标,二分找pos
25. int cnt;//x坐标离散后的数量
26. int cov[MAXN<<1], lz[MAXN<<1];//cov记录该段完全覆盖次数,lz是cov的lazy标记(可以不用)
27. type sum2[MAXN<<1], sum1[MAXN<<1], one[MAXN<<1];
28. //sum2记录被覆盖2次及以上的区间长度，sum1是一次以上的，one只覆盖一次的长度
29. void init()
30. {
31. clc(sum2,0);
32. clc(sum1,0);
33. clc(cov,0);
34. clc(lz,0);
35. }
36. inline int id(int x, int y){return x+y|x!=y;}
37. void push\_up(int x, int y, int m)
38. {
39. /\*计算覆盖一次以及以上的长度\*/
40. int u = id(x,y), l = id(x,m), r = id(m+1,y);
41. if(cov[u]) sum1[u] = pos[y+1]-pos[x];//完全覆盖
42. else if(x == y) sum1[u] = 0;//叶子结点
43. else sum1[u] = sum1[l]+sum1[r];
44. /\*计算覆盖2次以及2次以上的长度\*/
45. if(2 <= cov[u]) sum2[u] = pos[y+1]-pos[x],sum1[u];
46. else if(x == y) sum2[u] = 0;
47. else if(1 == cov[u]) sum2[u] = sum1[l]+sum1[r];//当前覆盖了1次只要加上覆盖1次的儿子就是覆盖2次的值
48. else sum2[u] = sum2[l]+sum2[r];
49. /\*减一下就是只覆盖一次的长度了\*/
50. one[u] = sum1[u]-sum2[u];
51. }
52. /\*这个向下传递可以不用\*/
53. void push\_down(int x, int y, int m)
54. {
55. int u = id(x,y);
56. if(lz[u])
57. {
58. int l = id(x,m), r = id(m+1,y);
59. cov[l] += lz[u], cov[r] += lz[u];
60. lz[l] += lz[u], lz[r] += lz[u];
61. lz[u] = 0;
62. }
63. }
64. void update(int x, int y, int ql, int qr, int v)
65. {
66. int m = (x+y)>>1;
67. if(ql <= x && y <= qr)
68. {
69. cov[id(x,y)] += v;
70. push\_up(x,y,m);
71. return;
72. }
73. push\_down(x,y,m);
74. if(ql <= m) update(x,m,ql,qr,v);
75. if(qr > m) update(m+1,y,ql,qr,v);
76. push\_up(x,y,m);
77. }
78. int main()
79. {
80. #ifdef SHY
81. freopen("e:\\1.txt", "r", stdin);
82. #endif
83. int t,count = 0;
84. scanf("%d%\*c", &t);
85. while(t--)
86. {
87. int n;
88. scanf("%d%\*c", &n);
89. cnt = tol = 0;
90. int nn = 0;
91. rep(i,0,n)
92. {
93. type x1,y1,x2,y2;
94. scanf("%I64d %I64d %I64d %I64d%\*c", &x1, &y1, &x2, &y2);
95. tmp[nn++] = x1, tmp[nn++] = x2;
96. p[tol].h = y1, p[tol].l = x1, p[tol].r = x2, p[tol++].v = 1;
97. p[tol].h = y2, p[tol].l = x1, p[tol].r = x2, p[tol++].v = -1;
98. }
99. sort(p,p+tol);
100. sort(tmp,tmp+nn);
101. pos[cnt++] = tmp[0];
102. rep(i,1,nn)//tmp去重
103. {
104. if(tmp[i] != tmp[i-1]) pos[cnt++] = tmp[i];
105. }
106. type ans = 0;
107. init();
108. rep(i,0,tol)
109. {
110. int x = lower\_bound(pos,pos+cnt,p[i].l)-pos, y = lower\_bound(pos,pos+cnt,p[i].r)-pos-1;
111. update(0,cnt-1,x,y,p[i].v);
112. ans += one[id(0,cnt-1)]\*(p[i+1].h-p[i].h);
113. }
114. printf("Case %d: %I64d\n",++count, ans);
115. }
116. return 0;
117. }

### 多个矩形总周长

就是在求面积的时候每次求和，需要多加三个数组，lc[],rc[],num[],lc和rc记录左右端点是否被覆盖，num记录区间有多少分开的覆盖段；则公式为ans += abs(sum[id(1,n)]-lsum)+2\*(p[i+1].h-p[i].h)\*num[id(1,n)];其中lsum是上一次的x轴覆盖总长度(就是sum[id(1,n)]);

Push\_up改为(未离散)：

1. void push\_up(int x, int y, int m)
2. {
3. int u = id(x,y);
4. if(cov[u]) sum[u] = y+1-x, num[u] = 1, lc[u] = rc[u] = 1;
5. else if(x == y) sum[u] = num[u] = lc[u] = rc[u] = 0;
6. else
7. {
8. int l = id(x,m), r = id(m+1,y);
9. sum[u] = sum[l]+sum[r];
10. lc[u] = lc[l], rc[u] = rc[r];
11. num[u] = num[l]+num[r];
12. if(rc[l] && lc[r]) num[u]--;
13. }
14. }

Ans统计改为(未离散):

1. sort(p,p+tol);
2. init();
3. int ans = 0, lsum = 0;
4. rep(i,0,tol)
5. {
6. update(0,cnt-1,p[i].l,p[i].r-1,p[i].v);
7. if(i < tol-1)
8. ans += abs(sum[id(0,cnt-1)]-lsum)+2\*(p[i+1].h-p[i].h)\*num[id(0,cnt-1)];
9. else ans += lsum; //要加上最后一条上边的长度
10. lsum = sum[id(0,cnt-1)];
11. }

### 矩形能包含多少点

【问题】给出N个二维坐标点，和一个大小固定的矩形，任意放置该矩形最多能包含多少个点(边上的点也算)

把每个点坐标变成一个矩形的左下角，根据给出的矩形把每个点变成一个矩形，最后答案就是覆盖最多的次数，可以用线段树维护覆盖最多次数，加上lazy操作就可以了。

线段树就是标准的求区间最值的写法，注意一点：因为边上的点也算，所以在计算每个点的右上角的时候横纵坐标都要+1(这样才能保证两个矩形相切的时候也算进去)。

如果不包含边上的点，则需要在计算右上角的时候-1。

给出ans统计：

1. int ans = 0;
2. clc(mx,0);
3. clc(lz,0);
4. rep(i,0,tol)
5. {
6. int x = lower\_bound(pos,pos+cnt,p[i].l)-pos;
7. int y = lower\_bound(pos,pos+cnt,p[i].r)-pos;
8. update(0,cnt-1,x,y,p[i].v);
9. ans = max(ans, mx[id(0,cnt-1)]);
10. }

## Trie树

前缀树(字典树)模版(O(len) len是字符串长度)

【len叉树】空间复杂度(n\*len\*len) n是单词总数，len是字符串长度

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define INF 0x3f3f3f3f
6. #define MAXN 500010
7. struct NODE{//trie树结点
8. NODE \*child[26];//儿子结点
9. char val[15];//节点字符串
10. NODE(){
11. clc(child,0);
12. val[0] = 0;
13. }
14. };
15. void insert(NODE \*u, char \*s, char \*v)//插入key为s，值为v
16. {
17. int n = strlen(s);//从根节点(0)开始
18. rep(i,0,n)
19. {
20. int c = s[i]-'a';
21. if(!u->child[c])//不存在的前缀，新建结点
22. u->child[c] = new NODE;
23. u = u->child[c];
24. }
25. strcpy(u->val,v);//叶子节点赋值
26. }
27. bool query(NODE \*u, char \*s, char \*v)//根据s查找,成功返回true,并且存到v
28. {
29. int n = strlen(s);
30. rep(i,0,n)
31. {
32. int c = s[i]-'a';
33. if(!u->child[c])//不存在的key
34. return false;
35. u = u->child[c];
36. }
37. if(!u->val[0]) return false;//不是叶子
38. strcpy(v,u->val);
39. }
40. void clear(NODE \*u)//释放空间
41. {
42. rep(i,0,26)
43. {
44. if(u->child[i])
45. clear(u->child[i]);
46. }
47. free(u);
48. }

【左儿子右兄弟建树】时间复杂度略高，空间复杂度为n\*len（对于len比较长的必须这样存）

1. const int MAXN = 10\*50000;//每个单词长度\*单词总数
2. struct NODE{//trie树结点
3. int son, next;//左儿子，右兄弟
4. char val[15], ch;//叶子节点字符串,每个节点字符
5. }node[MAXN];
6. int sz;
7. void init()
8. {
9. sz = 1;
10. node[0].next = node[0].son = -1;
11. }
12. void insert(char \*s, char \*v)//插入key为s，值为v
13. {
14. int u = 0,n = strlen(s);//从根节点(0)开始
15. rep(i,0,n)
16. {
17. int v;
18. bool finded = false;
19. for(v = node[u].son; ~v; v = node[v].next)//查找字符为s[i]结点
20. {
21. if(s[i] == node[v].ch)
22. {
23. finded = true;
24. break;
25. }
26. }
27. if(!finded)//不存在的前缀，新建结点
28. {
29. v = sz++;
30. //初始化新节点
31. node[v].ch = s[i];
32. node[v].son = -1;
33. node[v].next = node[u].son;
34. node[v].val[0] = 0;
35. //v结点添加为u的儿子
36. node[u].son = v;
37. }
38. u = v;
39. }
40. strcpy(node[u].val,v);//叶子节点赋值
41. }
42. bool query(char \*s, char \*v)//根据s查找,成功返回true,并且存到v
43. {
44. int u = 0,n = strlen(s);
45. rep(i,0,n)
46. {
47. int v;
48. bool finded = false;
49. for(v = node[u].son; ~v; v = node[v].next)//查找字符为s[i]结点
50. {
51. if(s[i] == node[v].ch)
52. {
53. finded = true;
54. break;
55. }
56. }
57. if(!finded) return false;//不存在
58. u = v;
59. }
60. if(!node[u].val[0]) return false;//不是叶子
61. strcpy(v,node[u].val);
62. }

## 平衡树-Treap

1. /\*
2. Treap同时有二叉搜索树(BST,键值:左儿子<根<右儿子)和堆的性质(根节点的优先级大于儿子节点)
3. 这样能保证在优先级唯一且不相等的情况下树是唯一的
4. 这里的r(优先级)是用rand()生成的，能证明查找，插入，删除，找第K大，名次的时间复杂度都是log(n)
5. 注意:rand()最大值是RAND\_MAX(系统不同值不同，可以输出看看),所以不够的话不能直接用
6. rand()是在0~RAND\_MAX中的数按照某个顺序输出的，在输出前RAND\_MAX+1中是不会重复的
7. \*/
8. struct NODE{
9. NODE \*ch[2];
10. int r;//随机优先级
11. int v;//键值
12. int sum;//以当前节点为根的总结点数
13. NODE(int a){
14. v = a,ch[0] = ch[1] = NULL, r = rand(), sum = 1;
15. }
16. int cmp(int x){
17. if(x == v) return -1;
18. return x<v?0:1;
19. }
20. void push\_up(){
21. sum = 1;
22. if(ch[0]) sum += ch[0]->sum;
23. if(ch[1]) sum += ch[1]->sum;
24. }
25. };
26. void clear(NODE\* &u)//清空树
27. {
28. if(!u) return;
29. if(u->ch[0]) clear(u->ch[0]);
30. if(u->ch[1]) clear(u->ch[1]);
31. delete u;
32. u = NULL;
33. }
34. void rotate(NODE\* &u, int d)//旋转，d = 0左旋,d = 1右旋
35. {
36. NODE \*k = u->ch[d^1];
37. u->ch[d^1] = k->ch[d], k->ch[d] = u;
38. u->push\_up(), k->push\_up(); //更新sum,先更新u
39. u = k;
40. }
41. void insert(NODE\* &u, int v)//插入键值v(不需要保证v是否存在)
42. {
43. if(!u) u = new NODE(v);
44. else
45. {
46. int d = v < u->v?0:1;//不要使用cmp(),可能已经存在相同的v
47. insert(u->ch[d], v);
48. if(u->r < u->ch[d]->r) //如果儿子的r比当前大则需要旋转
49. rotate(u,d^1);//左儿子则右旋，右儿子左旋
50. }
51. u->push\_up();//更新sum
52. }
53. void del(NODE\* &u, int v)//删除键值v
54. {
55. int d = u->cmp(v);
56. if(~d)//没找到继续往下找
57. del(u->ch[d],v);
58. else
59. {
60. NODE \*o = u;
61. if(u->ch[0] && u->ch[1])//左右儿子都存在
62. {
63. int d2 = u->ch[0]->r > u->ch[1]->r?1:0;//优先级大的儿子旋转为根
64. rotate(u,d2);
65. del(u->ch[d2],v);
66. }
67. else
68. {
69. if(!u->ch[0]) u = u->ch[1];//只有右儿子
70. else u = u->ch[0];//只有左儿子
71. delete o;
72. //u = NULL;
73. }
74. }
75. if(u) u->push\_up();
76. }
77. int kth(NODE\* u, int k)//寻找第k大键值
78. {
79. if(!u || k <= 0 || k > u->sum) return 0;//不存在
80. int d = k-(u->ch[0]?u->ch[0]->sum:0);
81. if(d == 1) return u->v;//找到第k大值
82. if(d <= 0) return kth(u->ch[0],k);
83. return kth(u->ch[1],k-(u->ch[0]?u->ch[0]->sum:0)-1);
84. }
85. void merge(NODE\* &u, NODE \* &to)//把u树合并到to树,O(n1\*log(n2)) n1,n2是u和to中的节点数
86. {
87. if(u->ch[0]) merge(u->ch[0],to);
88. if(u->ch[1]) merge(u->ch[1],to);
89. insert(to,u->v);
90. delete u;
91. u = NULL;
92. }

## 伸展树Splay-Tree

【从下往上伸展(可以伸展任意结点)】

左右各添加一个结点可以更加方便处理；记得把null各项值初始化，比如sum=0,mx = -INF;

1. /\*
2. splay单次可能会是O(n)但是平摊下来为log(n)
3. SplayTree可以是左儿子<根<右儿子；也可以不这样,存一个数组,中序遍历顺序和原数组相同即可
4. \*/
5. struct NODE{
6. NODE \*ch[2], \*fa;
7. int sz,v,sum, lmx, rmx, mx;//sz-节点总数,v-当前节点值，lmx-包括左端点的最大连续和，rmx-包括右端点的最大连续和，mx-当前区间的最大连续和,sum-当前区间和
8. bool f;//lazy是否翻转
9. int setv;//lazy
10. int chd(){//当前儿子是否是左儿子(0-左，1-右)
11. return this == fa->ch[1];
12. }
13. };
14. NODE \*null = new NODE;
15. NODE\* newnode(int v)
16. {
17. NODE \*tmp = new NODE();
18. tmp->sz = 1;tmp->v = v;
19. tmp->ch[0] = tmp->ch[1] = tmp->fa = null;
20. return tmp;
21. }
22. void push\_up(NODE \*u){//向上更新
23. u->sz = u->ch[0]->sz+u->ch[1]->sz+1;
24. u->sum = u->ch[0]->sum+u->ch[1]->sum+u->v;
25. u->lmx = max(u->ch[0]->lmx, u->ch[0]->sum + u->v + max(u->ch[1]->lmx,0));
26. u->rmx = max(u->ch[1]->rmx, u->ch[1]->sum + u->v + max(u->ch[0]->rmx,0));
27. u->mx = max(u->ch[0]->rmx,0) + max(u->ch[1]->lmx,0) + u->v;
28. u->mx = max(max(u->ch[0]->mx,u->ch[1]->mx), u->mx);
29. }
30. inline void pd\_set(NODE \*u, int v)//向下更新单点修改
31. {
32. if(null == u)return;
33. u->v = v;
34. u->sum = u->sz\*v;
35. u->lmx = u->rmx = u->mx = max(u->sum,v);
36. u->setv = v;
37. }
38. inline void pd\_rev(NODE \*u)//向下更新单点翻转
39. {
40. if(null == u)return;
41. swap(u->ch[0],u->ch[1]);
42. swap(u->lmx,u->rmx);
43. u->f ^= 1;
44. }
45. void push\_down(NODE \*u){//向下更新
46. if(-INF != u->setv){
47. pd\_set(u->ch[0],u->setv);
48. pd\_set(u->ch[1],u->setv);
49. u->setv = -INF;
50. }
51. if(u->f)
52. {
53. pd\_rev(u->ch[0]);
54. pd\_rev(u->ch[1]);
55. u->f = false;
56. }
57. }
58. void free\_all(NODE \*u)//释放内存
59. {
60. if(null == u) return;
61. free\_all(u->ch[0]);
62. free\_all(u->ch[1]);
63. delete u;
64. }
65. inline void setch(NODE \*fa, NODE \*u, int p)//重置关联的儿子和父亲,p=0表示u是fa的左儿子1是右儿子
66. {
67. fa->ch[p] = u, u->fa = fa;
68. }
69. void rot(NODE \*u)//旋转
70. {
71. NODE\*y = u->fa;
72. int d = u->chd();//决定是左旋还是右旋d=0左儿子对应右旋,d=1右儿子对应左旋
73. setch(y->fa,u,y->chd());//把y的父亲(就是u父亲的父亲)的y儿子变为u =====把u放到当前根
74. setch(y,u->ch[d^1],d);//把u的对应儿子放到u所在位置(此时变成两颗树了 y成另外一颗树的根)
75. setch(u,y,d^1);//把y放到u的对应儿子
76. push\_up(y), push\_up(u);//先更新y再更新u(y在u下面了)
77. }
78. /\* =================== splay1->start ===========================================\*/
79. void dfs\_down(NODE \*x)//先把根到x所用到的所有的标记下放
80. {
81. if(x == null)return;
82. dfs\_down(x->fa);
83. push\_down(x);//保证从上往下push\_down
84. }
85. void splay(NODE\* &rt,NODE \*x, NODE \*p)//把rt树的x节点伸展到p下方(即p的儿子),从下往上伸展
86. {
87. dfs\_down(x);
88. if(x == p) return;
89. while(x->fa != p)
90. {
91. NODE \*y = x->fa;
92. if(x->fa->fa != p && x->chd() == y->chd()) rot(y);
93. rot(x);
94. }
95. if(p == null) rt = x;
96. }
97. /\* =================== splay1->end ===========================================\*/
98. /\* =================== splay2->start ===========================================\*/
99. NODE\* find\_kth(NODE \*u, int k)//寻找第k小的结点(数列从左往右第k个),u为splay的根
100. {
101. push\_down(u);
102. int d = k - u->ch[0]->sz;
103. if(1 == d) return u;//找到
104. if(d <= 0) return find\_kth(u->ch[0],k);//往左
105. return find\_kth(u->ch[1],k-u->ch[0]->sz-1);//往右
106. }
107. void splay(NODE\* &rt,int k, NODE \*p)//把rt树第k大结点(数列从左往右第k个)伸展到p下面
108. {
109. NODE \*x = find\_kth(rt,k);
110. if(x == p) return;
111. while(x->fa != p)
112. {
113. NODE \*y = x->fa;
114. if(x->fa->fa != p && x->chd() != y->chd()) rot(y);
115. rot(x);
116. }
117. if(p == null) rt = x;
118. }
119. /\* =================== splay2->end ===========================================\*/
120. void insert(NODE \*&rt, int k, NODE\* x)//把x结点插入第k个后面
121. {
122. splay(rt,k,null);
123. if(rt->ch[1] == null) setch(rt,x,1);
124. else
125. {
126. NODE \*u = rt->ch[1];
127. push\_down(u);
128. while(u->ch[0] != null) u = u->ch[0],push\_down(u);
129. setch(u,x,0);
130. while(x->fa != null) push\_up(x), x = x->fa;//向上更新sz
131. }
132. push\_up(rt);
133. }
134. NODE\* del(NODE \*&rt, int k)//删除第k个结点,需要左右都加一个虚拟结点,返回删除结点
135. {
136. splay(rt,k,null);
137. NODE \*u = rt->ch[0], \*v = rt->ch[1], \*ans = rt;
138. if(null == u) v->fa = null, rt = v;
139. else if(null == v) u->fa = null, rt = u;
140. else
141. {
142. splay(rt,rt->ch[0]->sz+2,rt);//把rt右儿子子树最小的伸展到rt->ch[1],这样rt->ch[1]没有左儿子
143. setch(rt->ch[1],rt->ch[0],0);
144. rt = rt->ch[1], rt->fa = null;
145. push\_up(rt);
146. }
147. ans->ch[0] = ans->ch[1] = ans->fa = null;
148. ans->sz = 1;
149. return ans;
150. }
151. /\*合并left和right，假定left所有元素都比right小；right可以是NULL，但left不可以\*/
152. NODE\* merge(NODE \*left, NODE \*right)
153. {
154. splay(left,left->sz,null);//把left中序顺序最后一个结点 伸展到根,这样left就没有右子树了
155. setch(left,right,1);//把right接到left的右子树
156. push\_up(left);//更新根节点需要维护的值
157. return left;
158. }
159. /\*分裂rt,把rt前k小结点放在left,其他放在right; 1<= k <=rt->sum当k==rt->sum时right=NULL,rt不能空\*/
160. void split(NODE\* &rt, int k, NODE\* &left, NODE\* &right)
161. {
162. splay(rt,k,null);//把rt的第k小的结点伸展到u的根
163. right = rt->ch[1];//rt的右儿子就是right
164. right->fa = null;//right的fa需要变为null
165. left = rt;//rt的左儿子以及rt就是left，右儿子改为null
166. rt->ch[1] = null;
167. push\_up(left);//只需要更新左儿子，右儿子不需要更新
168. }
169. /\*修改[x,y]的值,lazy标记,rt != NULL,把x-1伸展到根，y+1伸展到根的右儿子，那么根的右儿子的左儿子就是区间[x,y]，这里左右各增加了一个虚拟节点方便处理\*/
170. void update(NODE\* &rt, int x, int y, int v)
171. {
172. splay(rt,x-1,null);
173. splay(rt,y+1,rt);
174. NODE \*&u = rt->ch[1]->ch[0];
175. pd\_rev(u);
176. pd\_set(u,v);
177. /\*如果没有删除添加新树，只是修改标记的话可以不需要往上更新\*/
178. push\_up(rt->ch[1]);
179. push\_up(rt);
180. }
181. int find\_small\_num(NODE \*u, int v)//找到比v小的元素数量
182. {
183. if(u == null) return 0;
184. if(v < u->v) return find\_small\_num(u->ch[0],v);
185. return find\_small\_num(u->ch[1],v)+u->ch[0]->sz+1;
186. }
187. void bulid(NODE \*&u, int x, int y, int \*a,NODE \*fa)
188. {
189. if(x > y)
190. {
191. u = null;
192. return;
193. }
194. int m = (x+y)>>1;
195. u = new NODE();
196. u->v = a[m];
197. u->f = 0;
198. u->fa = fa;
199. u->rmx = u->lmx = u->mx = u->setv = -INF;
200. bulid(u->ch[0],x,m-1,a,u);
201. bulid(u->ch[1],m+1,y,a,u);
202. push\_up(u);
203. }
204. void debug(NODE \*u)
205. {
206. if(u == null) return;
207. push\_down(u);
208. debug(u->ch[0]);
209. printf("%d ", u->v);
210. debug(u->ch[1]);
211. push\_up(u);
212. if(u->fa == null) putchar('\n');
213. }

## 动态树-LCT(link-cut-tree)

动态树问题，复杂度均摊为O()；

**有根树不能使用make\_root(),无根树必须使用make\_root()来确定边的方向。无根树没有LCA，两者不能混用；**

**而没有link和cut操作的静态树，则无根和有根都可以用。**

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long LL;
4. #define rep(i,a,n) for(int i = a; i < n; i++)
5. #define repe(i,a,n) for(int i = a; i <= n; i++)
6. #define per(i,n,a) for(int i = n; i >= a; i--)
7. #define clc(a,b) memset(a,b,sizeof(a))
8. const int INF = 0x3f3f3f3f, MAXN = 300000+10;
9. int par[MAXN],ch[MAXN][2],fa[MAXN],fz[MAXN],val[MAXN],add[MAXN],mx[MAXN];
10. //par是真实树的父亲，fa是虚树(splay)的父亲
11. void init()//下标1~n
12. {
13. clc(par,0);clc(ch,0);clc(fa,0);
14. clc(fz,0);clc(val,0);clc(add,0);clc(mx,0);
15. }
16. inline int chd(int u){return ch[fa[u]][1] == u;}
17. inline void setch(int f, int u, int d){ch[f][d] = u,fa[u] = f;}
18. inline void push\_up(int u)
19. {
20. mx[u] = max(mx[ch[u][0]],max(mx[ch[u][1]],val[u]));
21. }
22. inline void rot(int u)
23. {
24. int d = chd(u),y = fa[u];
25. setch(fa[y],u,chd(y));
26. setch(y,ch[u][d^1],d);
27. setch(u,y,d^1);
28. push\_up(y);push\_up(u);
29. }
30. inline void rev(int u)
31. {
32. swap(ch[u][0],ch[u][1]);
33. fz[u] ^= 1;
34. }
35. inline void one\_add(int u, int v)
36. {
37. if(!u) return;
38. add[u] += v;
39. val[u] += v;
40. mx[u] += v;
41. }
42. inline void push\_down(int u)
43. {
44. if(add[u])
45. {
46. one\_add(ch[u][0],add[u]);one\_add(ch[u][1],add[u]);
47. add[u] = 0;
48. }
49. if(fz[u])
50. {
51. rev(ch[u][0]),rev(ch[u][1]);
52. fz[u] = 0;
53. }
54. }
55. void dfs\_down(int u)
56. {
57. if(fa[u]) dfs\_down(fa[u]);
58. push\_down(u);
59. }
60. void splay(int u)
61. {
62. dfs\_down(u);
63. int rt = u;
64. while(fa[rt]) rt = fa[rt];
65. if(rt == u) return;
66. par[u] = par[rt],par[rt] = 0;
67. while(fa[u])
68. {
69. if(fa[fa[u]] && chd(u) == chd(fa[u])) rot(fa[u]);
70. rot(u);
71. }
72. }
73. void expose(int u)
74. {
75. for(int now = u,la = 0;now;la = now,now = par[now])
76. {
77. splay(now);
78. par[ch[now][1]] = now;fa[ch[now][1]] = par[la] = 0;
79. setch(now,la,1);
80. push\_up(now);
81. }
82. splay(u);
83. }
84. int find\_root(int u)//找到splay最左边点，即当前连通树的根
85. {
86. expose(u);
87. while(ch[u][0]) u = ch[u][0];
88. return u;
89. }
90. /\*====================无根树============================\*/
91. void make\_root(int u)//把u点变成当前splay的根(有根树不能使用)
92. {
93. expose(u);
94. rev(u);
95. }
96. bool link(int u, int v)//如果x(以下x都是当前节点),y不在同一颗子树中，则通过在x,y之间连边的方式，连接这两颗子树
97. {
98. if(find\_root(u) == find\_root(v)) return false;
99. make\_root(u);
100. par[u] = v;
101. return true;
102. }
103. void cut(int u)//有根树单个cut和这个一样
104. {
105. expose(u);
106. par[u] = fa[ch[u][0]] = 0;
107. ch[u][0] = 0;
108. push\_up(u);
109. }
110. bool cut(int u, int v)//如果x,y在同一颗子树中，且x!=y,则将x视为这颗子树的根以后，切断y与其父亲结点的连接
111. {
112. if(u == v || find\_root(u) != find\_root(v)) return false;
113. make\_root(u);
114. cut(v);
115. return true;
116. }
117. bool update(int x, int y, int v)//如果x,y在同一颗子树中，则将x,y之间路径上所有点的点权增加v
118. {
119. if(find\_root(x) != find\_root(y)) return false;
120. make\_root(x);
121. expose(y);
122. one\_add(y,v);
123. return true;
124. }
125. int query(int x, int y)//如果x,y在同一颗子树中，返回x,y之间路径上点权的最大值
126. {
127. if(find\_root(x) != find\_root(y)) return -1;
128. make\_root(x);
129. expose(y);
130. return mx[y];
131. }
132. /\*=======================有根树============================\*/
133. bool link\_2(int u, int v)//如果x,y不在同一颗子树中,把u所在子树接到v下面
134. {
135. if(find\_root(u) == find\_root(v)) return false;
136. expose(u);
137. par[u] = v;
138. return true;
139. }
140. int query\_2(int x, int y)//如果x,y在同一颗子树中，返回x,y之间路径上点权的最大值
141. {
142. if(find\_root(x) != find\_root(y)) return -1;
143. expose(x);
144. for(int now = y,la = 0;now; la = now,now = par[now])
145. {
146. splay(now);
147. if(!par[now]) return max(mx[ch[now][1]],max(mx[la],val[now]));//now就是LCA(x,y)
148. par[ch[now][1]] = now;fa[ch[now][1]] = par[la] = 0;
149. setch(now,la,1);
150. push\_up(now);
151. }
152. }
153. int find\_last(int u)//用来查找真实树中的上个结点，只在下面分成两段使用
154. {
155. while(ch[u][1]) u = ch[u][1];
156. return u;
157. }
158. void update(int x, int y, int v) //如果x,y在同一颗子树中，则将x,y之间路径上所有点的点权增加v
159. {
160. expose(x);
161. for(int now = y,la = 0;now;la = now,now = par[now])
162. {
163. splay(now);
164. if(!par[now])
165. {
166. val[now] += v;//分成三块：单点LCA，ch[now][1]:(lca->x]，la:(lca->y]
167. one\_add(ch[now][1],v);
168. one\_add(la,v);
169. /\*下面分成两段:now:[lca->x]，la:(lca->y]\*/
170. /\*fa[ch[now][0]] = 0;par[now] = find\_last(ch[now][0]);
171. ch[now][0] = 0;
172. one\_add(now,v);
173. push\_down(now);
174. one\_add(la,v);\*/
175. return;
176. }
177. par[ch[now][1]] = now;fa[ch[now][1]] = par[la] = 0;
178. setch(now,la,1);
179. }
180. }
181. int main()
182. {
183. #ifdef SHY
184. freopen("d:\\1.txt", "r", stdin);
185. #endif
186. int n;
187. while(~scanf("%d", &n))
188. {
189. init();
190. rep(i,1,n)
191. {
192. int u,v;
193. scanf("%d %d", &u, &v);
194. link(u,v);
195. }
196. repe(i,1,n)
197. {
198. int v;
199. scanf("%d", &v);
200. update(i,i,v);
201. }
202. int q;
203. scanf("%d", &q);
204. while(q--)
205. {
206. int op,x,y;
207. scanf("%d %d %d", &op, &x, &y);
208. if(1 == op)
209. {
210. if(!link(x,y)) puts("-1");
211. }
212. else if(2 == op)
213. {
214. if(!cut(x,y)) puts("-1");
215. }
216. else if(3 == op)
217. {
218. int z;
219. scanf("%d", &z);
220. if(!update(y,z,x)) puts("-1");
221. }
222. else printf("%d\n", query(x,y));
223. }
224. putchar('\n');
225. }
226. return 0;
227. }

## 莫队算法

说白了就是有技巧的暴力：

分块+离线按询问左端点所在块以及右端点升序排序；

对于上一次查询区间[a,b]和这次需要查询的区间[c,d];如果a<c则删除区间(a,c]，否则加上区间[c,a);如果b<d则加上区间[b,d),否则删除区间(d,b];

可以证明复杂度是N\*的：右区间在同一块中单调递增，最多移动n次；而一共只有块，而如果不同块向左移动由于只有块所以右区间移动总复杂度：N\*;而左区间在同一块左右滑动不超过即使跨越两块也不超过\*2;一共Q次询问所以总复杂度2\*\*q;极端情况从第一块跨越到最后一块，那么说明中间没有查询了，怎样复杂度才O(n);而随着中间查询的增多复杂度越来越接近q\*

整个算法复杂度为: **n\*+ 2\*\*q**;

【BZOJ2038】

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef unsigned int LL;
4. #define rep(i,a,n) for(int i = a; i < n; i++)
5. #define repe(i,a,n) for(int i = a; i <= n; i++)
6. #define per(i,n,a) for(int i = n; i >= a; i--)
7. #define clc(a,b) memset(a,b,sizeof(a))
8. const int INF = 0x3f3f3f3f, MAXN = 50000+10,SIZE = 500;
9. struct NODE{
10. LL x,y;
11. int id,block;
12. bool operator<(const NODE&t)const{
13. if(block != t.block) return block < t.block;
14. return y < t.y;
15. }
16. }in[MAXN];
17. int a[MAXN];
18. LL ans1[MAXN],ans2[MAXN],cnt[MAXN];
19. LL gcd(LL a, LL b)
20. {
21. while(b)
22. {
23. a %= b;
24. if(a < b) swap(a,b);
25. }
26. return a;
27. }
28. int main()
29. {
30. #ifdef SHY
31. freopen("d:\\1.txt", "r", stdin);
32. #endif
33. int n,q;
34. scanf("%d %d", &n, &q);
35. rep(i,0,n) scanf("%d", &a[i]);
36. rep(i,0,q)
37. {
38. scanf("%d %d", &in[i].x, &in[i].y);
39. in[i].x--;in[i].y--;
40. in[i].id = i,in[i].block = in[i].x/SIZE;
41. }
42. sort(in,in+q);
43. LL sum = 0,x = 0,y = 0;cnt[a[0]]++;
44. rep(i,0,q)
45. {
46. while(x < in[i].x)
47. {
48. sum -= (--cnt[a[x++]]);
49. }
50. while(x > in[i].x)
51. {
52. sum += (cnt[a[--x]]++);
53. }
54. while(y < in[i].y)
55. {
56. sum += (cnt[a[++y]]++);
57. }
58. while(y > in[i].y)
59. {
60. sum -= (--cnt[a[y--]]);
61. }
62. LL d = (y-x+1)\*(y-x)/2;
63. LL g = gcd(sum,d);
64. ans1[in[i].id] = sum/g;ans2[in[i].id] = d/g;
65. }
66. rep(i,0,q) printf("%u/%u\n", ans1[i],ans2[i]);
67. return 0;
68. }

# 图论

## 图中是否有环(圈)的判定

环(圈)的概念：任意点出发能回到该点的回路(每条边只能走一次,每个点可以通过多次)

【1】有向图判是否有环：拓扑排序(有拓扑排序的图肯定无环，否则有环);或者dfs三色判环int[](只有能回到当前正在访问还没退出的点时才有环，即祖先节点)

【2】无向图判是否有环：并查集(边两端点在同一集合说明有环，否则无环)；或者dfs标记bool[]能否回到访问过点(除父亲节点)

【3】混合图(既有有向边又有无向边)是否有环：不能直接把无向边转换两条有向边；先判断无向边是否有环，然后把无向边用并查集缩点，则新图就是有向图是否有环。

【4】无向图判是否有奇环(边为奇数)：二分图染色(二分图中必定不存在奇环，而非二分图中必定存在奇环)

【5】无向图判是否有偶环(边为偶数)：在二分图染色的基础上，如果遇到两点都染色并且颜色不同说明存在偶环，但是这样如果先判了奇环可能导致偶环判断不到，需要回溯一下重新判断另外一条路径，加个vis[]保证边只走了一次就可以了。

代码(O(n))：

1. /\*找出无向图上是否分别存在奇环和偶环\*/
2. int head[MAXN], tol, to[MAXM], nxt[MAXM];
3. inline void add\_edge(int u, int v)
4. {
5. nxt[tol] = head[u], to[tol] = v;
6. head[u] = tol++;
7. }
8. int col[MAXN];
9. bool ou,ji, vis[MAXM];
10. void dfs(int u, int fa)
11. {
12. for(int i = head[u]; ~i; i = nxt[i])
13. {
14. if(-1 == col[u]) return;//当前边还没染色直接退出
15. int v = to[i];
16. if(fa == v) continue;//防止同一条边被走两次误判成偶环
17. if(~col[v])
18. {
19. if(col[v] == col[u]) ji = true;//颜色相同存在奇环
20. else ou = true;//颜色不同存在偶环
21. if(ji && ou) return;//剪枝
22. }
23. if(!vis[i])//边是否走过
24. {
25. vis[i] = true;
26. col[v] = col[u]^1, dfs(v,u);
27. }
28. }
29. col[u] = -1;//重新开始染色
30. }
31. int n;
32. void sloved()
33. {
34. ou = ji = 0;
35. clc(col,-1);
36. clc(vis,0);
37. repe(i,1,n)
38. {
39. if(~col[i]) continue;
40. col[i] = 0;
41. dfs(i,-1);
42. if(ji && ou) return;
43. }
44. }

## 欧拉路径和欧拉回路判定

在做一些图类时经常要用到欧拉路，比如近期的单词连接和涂彩棒等

欧拉通路: 通过图中每条边且只通过一次，并且经过每一顶点的通路。

欧拉回路: 通过图中每条边且只通过一次，并且经过每一顶点的回路。

【无向图是否具有欧拉通路或回路的判定】:

欧拉通路:**图连通**,图中只有0个或2个度为奇数的节点

欧拉回路:**图连通**,图中所有节点度均为偶数

(度：就是无向图的边u-v, du[u]++,du[v]++)

【有向图是否具有欧拉通路或回路的判定】:

欧拉通路:**图连通**,除2个端点外其余节点入度=出度；1个端点入度比出度大1；一个端点入度比出度小1 或 所有节点入度等于出度

欧拉回路:**图连通**,所有节点入度等于出度

【混合欧拉回路(路径)的判定】就是既有有向边，又有无向边，需要给无向边定向，原本的有向边无视

1.首先判断图的连通性，若不连通，无解。

2.然后任意定向无向边，计算每个点i的入度和出度之差deg[i]。若deg[i]为奇数，无解。

3.设立源点s和汇点t，若某点i入度<出度，连边(s,i,-deg[i]/2)，若入度>出度，连边(i,t,deg[i]/2)；对于任意定向的无向边(i,j,1)。

4.若有两个度数为奇数的点，假设存在欧拉路径，添加一条容量为1的边，构成欧拉回路，不影响结果。若全为偶数，直接最大流。

5.若从S发出的边全部满流(即最大流等于所有s出去的边权值之和)，证明存在欧拉回路(路径)，否则不存在。

ps：若要求输出路径，将网络中有(无)流量的边反向，加上原图的有向边，用套圈算法即可。

【一个图能最少用几笔画成(每条边必须只经过一次且每条边都需要访问)】=有奇数点的连通块的度为奇数的点的数量/2+没有度数为奇数的点的连通块数量

（还要看孤立点是不是算笔画，这里是不算的）

## 平面图-完全子图性质

完全子图定义：集合中任意两个点都有一条线段直接相连。

而这里是基于下面的限制的

平面图上N个点，满足**任意三点不共线**。把一些点用线段连起来，任意两条线段**不会在端点以外相交**的线段最多有**3\*N-6**条。这样可以忽略点的位置，按线段建图即可。

如果一个点的集合中**任意两个点都有一条线段直接相连**，则这样的集合中点的数量最多有**4**个，全部判断可以DFS回溯枚举，按点编号升序可去重。复杂度N^2(实测接近n^1.5)。

HDU5277就是求这样的最大点数集合(最大团)

## 树的直径(树的最长路)

有两种方法求树的直径：

1. 利用两次DFS(或者BFS)：先求出任意一点到所有点的一个最远点u，然后在求u的最远点v，则u到v的距离就是最长路(树的直径)。
2. 树形DP，求每个结点u为根的子树的最远点mx[i],以及不和最远点在同一子树的次远距离second，则最终答案就是MAX{mx[i]+second};转移就是mx[u] = max{mx[i]}+1;叶子结点mx[]和second都为0；

## 无向图的割顶和桥

### 无向图的割顶(Tarjan)

【定义】对于无向图G，如果删除某个点u后，连通分量数目增加，称u为图的割顶(关键点)。对于连通图，割顶就是删除之后图不再连通的点。

【判断那些点是割顶用iscut表示】

**不带**重边处理O(n+m)

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define clc(a,b) memset(a,b,sizeof(a))
4. #define min(a,b) (a>b?b:a)
5. #define MAXN 110
6. vector<int> g[MAXN];
7. int n,clock, pre[MAXN], low[MAXN];//pre[]记录每个点的dfs进入计数
8. bool iscut[MAXN];
9. //low[u]记录u和u的后代能回到的最早的祖先的pre值
10. void dfs(int u, int fa)
11. {
12. low[u] = pre[u] = ++clock;
13. int sz = g[u].size(), child = 0;
14. rep(i,0,sz)
15. {
16. int v = g[u][i];
17. if(v == fa) continue;
18. if(!pre[v])
19. {
20. child++;
21. dfs(v,u);
22. low[u] = min(low[u],low[v]);
23. if(low[v] >= pre[u]) iscut[u] = true;
24. }
25. else if(pre[v] < pre[u]) low[u] = min(low[u],pre[v]);
26. }
27. if(fa < 0 && child == 1) iscut[u] = 0;//根节点是否只有一个儿子,写在外面要判断重边
28. }
29. int sloved(int s)
30. {
31. clc(pre,0);
32. clc(iscut,0);
33. clock = 0;
34. dfs(s,-1);
35. if(clock != n) return -1;//原图不连通
36. int ans = 0;
37. repe(i,1,n) if(iscut[i]) ans++;
38. return ans;
39. }

### 无向图的桥(Tarjan)

带**重边**处理，O(n+m)

1. /\*找到最小权值的桥,带重边处理\*/
2. const int INF = 0x3f3f3f3f, MAXN = 1000+10, MAXM = MAXN\*MAXN\*2;
3. int head[MAXN],nxt[MAXM],to[MAXM],tol, cost[MAXN][MAXN];//二维cost为了处理重边,N大了需要hash或map
4. int n,m,clock, pre[MAXN],low[MAXN];//pre[]记录每个点的dfs进入计数
5. //low[u]记录u和u的后代能回到的最早的祖先的pre值
6. void add\_edge(int u, int v, int w)//无向图需要反向再调用一次
7. {
8. nxt[tol] = head[u], to[tol] = v;
9. head[u] = tol++;
10. if(~cost[u][v]) cost[u][v] = INF;//如果u-v之间有多条变，直接把他们的权值变成INF(即u-v不是桥,计数的话需要dfs中去掉)
11. else cost[u][v] = w;
12. }
13. int mi;
14. void dfs(int u, int fa)
15. {
16. low[u] = pre[u] = ++clock;
17. for(int i = head[u]; ~i; i = nxt[i])
18. {
19. int v = to[i];
20. if(v == fa)continue;//因为是dfs树，不能往回走
21. if(!pre[v])
22. {
23. dfs(v,u);
24. low[u] = min(low[u],low[v]);
25. if(low[v] > pre[u])//u-v是桥
26. mi = min(mi,cost[u][v]);
27. }
28. else if(pre[v] < pre[u]) low[u] = min(low[u],pre[v]);//v是u的祖先
29. }
30. }
31. int sloved()
32. {
33. clc(pre,0);
34. clock = 0;
35. mi = INF;
36. dfs(1,-1);
37. if(clock != n) return 0;//原图不连通
38. if(INF == mi) return -1;//没有桥
39. return max(mi,1);//mi为0还需要一个人
40. }

## 有向图强连通分量(SCC)

【Tarjan算法(O(n+m))】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define clc(a,b) memset(a,b,sizeof(a))
4. #define min(a,b) (a>b?b:a)
5. #define MAXN 10010
6. int n, pre[MAXN], sccno[MAXN], dfs\_clock, scc\_cnt;
7. //sccno[1~n]标记每个点在第几个SCC中,scc\_cnt是SCC的计数,dfs\_clock递归时间戳
8. vector<int> g[MAXN];
9. stack<int> s;
10. int dfs(int u)
11. {
12. int lowu = pre[u] = ++dfs\_clock, sz = g[u].size();
13. s.push(u);
14. rep(i,0,sz)
15. {
16. int v = g[u][i];
17. if(!pre[v])
18. {
19. int lowv = dfs(v);
20. lowu = min(lowu, lowv);
21. }
22. else if(!sccno[v]) lowu = min(lowu, pre[v]);
23. }
24. if(lowu == pre[u])//u的子孙最早只能回到u，说明找到一个SCC分量
25. {
26. int x;
27. scc\_cnt++;
28. do{
29. x = s.top();s.pop();
30. sccno[x] = scc\_cnt;
31. }while(x != u);
32. /\*可以对这个强连通分量操作\*/
33. }
34. return lowu;
35. }
36. void find\_scc()
37. {
38. dfs\_clock = scc\_cnt = 0;
39. clc(pre,0);
40. clc(sccno,0);
41. repe(i,1,n) if(!pre[i]) dfs(i);
42. }

【Kosaraju算法(O(n+m)常数比Tarjan大)】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define min(a,b) (a>b?b:a)
6. #define MAXN 10010
7. int sccno[MAXN], vis[MAXN], scc\_cnt, n;
8. vector<int> g[MAXN], gt[MAXN], s;
9. //gt[]是g[]的相反边的图
10. void dfs1(int u)//获取拓扑序
11. {
12. vis[u] = true;
13. int sz = g[u].size();
14. rep(i,0,sz) if(!vis[g[u][i]]) dfs1(g[u][i]);
15. s.push\_back(u);
16. }
17. void dfs2(int u)
18. {
19. sccno[u] = scc\_cnt;
20. int sz = gt[u].size();
21. rep(i,0,sz) if(!sccno[gt[u][i]]) dfs2(gt[u][i]);
22. }
23. void find\_scc()
24. {
25. scc\_cnt = 0;
26. s.clear();
27. clc(sccno,0);
28. clc(vis,0);
29. repe(i,1,n) if(!vis[i]) dfs1(i);
30. per(i,n-1,0) if(!sccno[s[i]]) scc\_cnt++, dfs2(s[i]);
31. }

## 拓扑排序模版

【使用队列(可以用优先队列限制某些状态) (O(n+m))】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define INF 0x3f3f3f3f
6. typedef long long LL;
7. #define MAXN 50
8. vector<int> g[MAXN];
9. int in[MAXN], tmp[MAXN],ans[MAXN], t, n;
10. bool ct;
11. int topo()//返回0说明存在多个排序，返回1表示找到求只有一个topo排序,-1表示没找到
12. {
13. int p = 0, c = 1;
14. queue<int> q;
15. rep(i,0,n) if(!in[i]) q.push(i);
16. while(!q.empty())
17. {
18. if(q.size() > 1) c = 0;
19. int u = q.front();q.pop();
20. int sz = g[u].size();
21. rep(i,0,sz)
22. {
23. int v = g[u][i];
24. if(!--in[v]) q.push(v);
25. }
26. ans[p++] = u;
27. }
28. if(p < n) return -1;
29. return c;
30. }

## 2-sat问题

有n个布尔变量xi,另有m个需要满足的条件，每个条件的形式都是”xi为真/假或者xj为真/假”。这里的或是指两个条件至少有一个条件是正确的。2-sat问题的目标就是给每个变量赋值，使得所有条件都得到满足。

O(n+m)

1. const int MAXN = 2000+10, MAXM = MAXN\*2\*MAXN\*2+10;
2. struct Twosat{
3. //id&1 = 1表示真，否则假(就是奇数编号为真，偶数编号为假)
4. //下标0~2n-1
5. int n, to[MAXM], next[MAXM],head[MAXN<<1],tol;
6. void init(int n)
7. {
8. this->n = n;
9. clc(head,-1);
10. tol = 0;
11. }
12. void add\_edge(int u, int v)
13. {
14. next[tol] = head[u],to[tol] = v;
15. head[u] = tol++;
16. }
17. void add\_clause(int x, int xv, int y, int vv)//添加子句x(xv)或者y(vv)至少满足一个
18. //一般是冲突的关系(例如x不能选择1并且y不能0，就是(x-1,y-0)不能同时存在)，则只需要添加addclause(x,1^1,y,0^1),就是都取反向状态->(x,0,y,1)，这样就表示x为0或者y为1至少满足一个，就不可能存在冲突了。
19. {
20. x = (x<<1)+xv;y = (y<<1)+vv;
21. add\_edge(x^1,y);
22. add\_edge(y^1,x);
23. }
24. bool vis[MAXN<<1];//从小到达扫描vis就是最小字典序
25. int s[MAXN<<1], c;
26. bool dfs(int x)
27. {
28. if(vis[x^1]) return false;
29. if(vis[x]) return true;
30. vis[x] = true;
31. s[c++] = x;
32. for(int i = head[x]; ~i; i = next[i])
33. if(!dfs(to[i])) return false;
34. return true;
35. }
36. bool sloved()
37. {
38. clc(vis,0);
39. for(int i = 0; i < n<<1; i += 2)
40. {
41. if(!vis[i] && !vis[i^1])
42. {
43. c = 0;
44. if(!dfs(i))
45. {
46. while(c > 0) vis[s[--c]] = false;
47. if(!dfs(i^1)) return false;//i和i^1都不能选说明不存在2-sat的解
48. }
49. }
50. }
51. return true;
52. }
53. }twosat;

## 最小生成树两种模板(MST)

n个节点的完全树有个最小生成树)

最小生成树是无向图的

### kruskal算法

【kruskal算法 O(k\*m) k为常数，并查集耗时】

1. #include <cstdio>
2. #include <cstring>
3. #include <cmath>
4. #include <algorithm>
5. using namespace std;
6. #define MAXN 105
7. #define dis(a,b) (sqrt((double)(x[a]-x[b])\*(x[a]-x[b])+(y[a]-y[b])\*(y[a]-y[b])))
8. struct NODE{
9. int a, b;
10. double v;
11. bool operator <(const NODE& t)const {return v < t.v;}
12. }p[MAXN\*MAXN];
13. int x[MAXN], y[MAXN], f[MAXN], n,m;
14. int find(int x){return f[x] == x ? x : f[x] = find(f[x]);}
15. double kruskal()
16. {
17. for(int i = 0; i <= n; i++)f[i] = i;
18. double ans = 0;
19. int cnt = 1;
20. sort(p,p+m);
21. for(int i = 0; i < m; i++)
22. {
23. int a = find(p[i].a), b = find(p[i].b);
24. if(a != b) ans += p[i].v, f[a] = b, cnt++;
25. if(cnt == n) break;
26. }
27. if(cnt < n) return -1;
28. return ans;
29. }
30. int main()
31. {
32. #ifdef SHY
33. freopen("e:\\1.txt","r",stdin);
34. #endif
35. int t;
36. scanf("%d%\*c", &t);
37. while(t--)
38. {
39. m = 0;
40. scanf("%d%\*c", &n);
41. for(int i = 0; i < n; i++)
42. scanf("%d %d%\*c", &x[i], &y[i]);
43. for(int i = 0; i < n; i++)
44. {
45. for(int j = i+1; j < n; j++)
46. {
47. p[m].v= dis(i,j);
48. if(10.0 <= p[m].v && 1000.0 >= p[m].v)
49. p[m].a = i, p[m++].b = j;
50. }
51. }
52. double ans = kruskal();
53. if(ans > 0.0)
54. printf("%.1lf\n", ans\*100.0);
55. else puts("oh!");
56. }
57. return 0;
58. }

### prim算法

【prim算法 O()】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define INF 0x3f3f3f3f
6. typedef long long LL;
7. #define MAXN 110
8. int cost[MAXN][MAXN], n, low[MAXN], f[MAXN];
9. bool vis[MAXN];
10. /\*f[]记录路径(父亲节点)，\*/
11. int prim()
12. {
13. int ans = 0;
14. clc(vis,0);
15. vis[1] = true;//从1开始查找
16. repe(i,2,n) low[i] = cost[1][i], f[i] = 1;
17. repe(i,2,n)
18. {
19. int mi = INF, p = -1;
20. repe(j,1,n)//找到没有选择点的最小的距离
21. {
22. if(!vis[j] && mi > low[j])
23. {
24. mi = low[j];
25. p = j;
26. }
27. }
28. if(INF == mi) return -1;//原图不连通
29. ans += mi;
30. vis[p] = true;
31. repe(j,1,n)
32. {
33. if(!vis[j] && low[j] > cost[p][j])//更新距离和f[]
34. low[j] = cost[p][j], f[j] = p;
35. }
36. }
37. return ans;
38. }

【Prim堆优化 O(M\*)】

1. const int INF = 0x3f3f3f3f, MAXN = 100000+10, MAXM = 1000000\*2+10;
2. int dis[MAXN];
3. struct NODE{
4. int p, dis;
5. bool operator<(const NODE&t) const{
6. return dis > t.dis;
7. }
8. NODE(int a, int b){
9. p = a, dis = b;
10. }
11. };
12. int n,m, head[MAXN], tol, nxt[MAXM], to[MAXM],cost[MAXM];
13. inline void add\_edge(int u, int v, int c)
14. {
15. nxt[tol] = head[u], to[tol] = v, cost[tol] = c;
16. head[u] = tol++;
17. }
18. bool vis[MAXN];
19. int prim()
20. {
21. priority\_queue<NODE> q;
22. clc(dis,0x3f);
23. clc(vis,0);
24. int ans = 0;
25. dis[1] = 0;
26. q.push(NODE(1,0));
27. repe(i,2,n) q.push(NODE(i,INF));
28. repe(i,1,n)
29. {
30. NODE now = q.top();q.pop();
31. while(vis[now.p])//保证每个点只算了一次
32. {
33. now = q.top();
34. q.pop();
35. }
36. ans += now.dis;
37. vis[now.p] = true;
38. for(int i = head[now.p]; ~i; i = nxt[i])
39. {
40. int v = to[i];
41. if(!vis[v] && dis[v] > cost[i])
42. {
43. dis[v] = cost[i];
44. q.push(NODE(v,dis[v]));
45. }
46. }
47. }
48. return ans;
49. }

## 最小树形图

朱刘算法是**有向图**的最小生成树算法，Prim和Kruskal是**无向图**的最小生成树算法。一般而言，这两类算法**不能通用**。根结点不能随便选，因为有向图不一定能回到根

【朱刘算法】O(N\*M)

1. /\*朱刘最小树形图算法 O(N\*M)\*/
2. struct Edge{
3. int u,v;
4. double cost;
5. }edge[MAXM];
6. int pre[MAXN], vis[MAXN], id[MAXN];
7. double in[MAXN];
8. double zl\_mst(int rt, int n, int m)//根，点数0~n，边数0~n
9. {
10. double ans = 0;
11. while(1)
12. {
13. //1.确定最小入边集
14. rep(i,0,n) in[i] = INF;
15. rep(i,0,m)
16. {
17. int u = edge[i].u, v = edge[i].v;
18. if(u != v && edge[i].cost < in[v])//忽略自环
19. {
20. in[v] = edge[i].cost;
21. pre[v] = u;
22. }
23. }
24. //检查除根之外是否有不可达顶点
25. in[rt] = 0;
26. rep(i,0,n) if(in[i] == INF) return -1;
27. //2.找环
28. clc(vis,-1);
29. clc(id,-1);
30. int cnt\_node = 0;//把一个环看成一个点，环外的一个点还是一个点
31. rep(i,0,n)
32. {
33. ans += in[i];//计算当前最小入边集的权值和
34. int v = i;
35. //对每个点回溯。若最后v回到了根，说明这个点不在环里；若最后v没回到根，说明v在环里。
36. while(vis[v] != i && -1 == id[v] && v != rt)
37. {
38. vis[v] = i;
39. v = pre[v];
40. }
41. if(v != rt && -1 == id[v])//v没回到根，说明v在环里
42. {
43. for(int u = pre[v]; u != v; u = pre[u]) id[u] = cnt\_node;//标记环上的所有顶点为同一序号
44. id[v] = cnt\_node++;
45. }
46. }
47. if(!cnt\_node) break;//没有环（每次都回到根，跳过上一个for循环，cnt\_node总是0），那么当前ans值就是最小树形图的边权和
48. rep(i,0,n)//给孤立点赋予点(连通块)编号
49. {
50. if(-1 == id[i]) id[i] = cnt\_node++;
51. }
52. //3.重新构图，准备开始下一轮迭代
53. rep(i,0,m)
54. {
55. int v = edge[i].v;
56. edge[i].u = id[edge[i].u];
57. edge[i].v = id[edge[i].v];
58. if(edge[i].u != edge[i].v)
59. edge[i].cost -= in[v];//放弃最小边（如果放弃是因为它在环里）所需要付出的代价，就是增量。这些增量又构成了一张新图，下一轮迭代处理这张新图。
60. /\*注意：
61. Here's the tricky part！还记得算法本身是怎样做的吗？
62. 求出环上每个点若放弃最小入边所产生的增量，形成一个增量集合，选择集合中最小的增量。
63. 这里把环上的所有点都标成同一个序号，看成一个点，
64. 下一次迭代过程的第一步就是求这个“点”入边的最小权值，
65. 实际上求的就是环上所有点增量集合的最小值，也就是最小增量！\*/
66. }
67. n = cnt\_node;
68. rt = id[rt];
69. }
70. return ans;
71. }

## 次小生成树

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. const int MAXN = 1000+10;
6. const double INF = 1e14;
7. double cost[MAXN][MAXN], low[MAXN];//cost[][]边权,low[]->DP的记录当前点到其他所有点的距离
8. double mx[MAXN][MAXN];//mx[u][v]表示唯一路径u到v上的最大边权(用来替换)
9. bool vis[MAXN], intree[MAXN][MAXN];
10. int f[MAXN], n;//f[]记录路径,上一个结点
11. struct NODE{
12. int x,y;
13. double a;
14. }p[MAXN];
15. double prim()
16. {
17. double ans = 0;
18. clc(vis,0);
19. clc(intree,0);
20. clc(mx,0);
21. vis[0] = true;//从结点0开始找
22. low[0] = 0;
23. rep(i,1,n) low[i] = cost[0][i],f[i] = 0;
24. rep(i,1,n)
25. {
26. double mi = INF;
27. int p = -1;
28. rep(j,0,n)
29. {
30. if(!vis[j] && low[j] < mi) mi = low[j],p = j;
31. }
32. if(-1 == p) continue;//图不连通
33. ans += mi;
34. vis[p] = true;
35. intree[f[p]][p] = intree[p][f[p]] = true;
36. rep(j,0,n)
37. {
38. if(vis[j] && j != p) mx[j][p] = mx[p][j] = max(mx[j][f[p]], low[p]);
39. if(!vis[j] && low[j] > cost[p][j]) low[j] = cost[p][j], f[j] = p;
40. }
41. }
42. /\*到这里为止的ans是最小生成树,下面是枚举替换边(枚举不在最小生成树中的边uv替换掉mx[u][v]的最小值)\*/
43. /\*这里枚举的是HDU4081的，不是标准最小生成树\*/
44. double ret = -1;
45. rep(i,0,n)
46. {
47. rep(j,0,n)
48. {
49. if(i == j) continue;
50. if(intree[i][j])//在最小生成树中的边
51. ret = max(ret,(p[i].a+p[j].a)/(ans-cost[i][j]));
52. else
53. ret = max(ret,(p[i].a+p[j].a)/(ans-mx[i][j]));
54. }
55. }
56. return ret;
57. }

## 最短路算法模板

bfs可以求无权的图，DP可以求有向无环图的最短路。来回最短路可用逆向建边

### dijkstra

【下面是两种dijkstra】求没有负权图(可以有环)

【O()】

1. /\*dp[ed]就是终点最短路，dp[ed] == INF时无路径\*/
2. void dijkstra(int st)
3. {
4. memset(dp,0x3f,sizeof(dp));
5. memset(vis,0,sizeof(vis));
6. dp[st] = 0;
7. for(int i = 0; i < n; i++)
8. {
9. int x, minn = INF;
10. for(int j = 0; j < n; j++) if(!vis[j] && dp[j] <= minn) minn = dp[x=j];
11. if(INF == minn) return;//小优化
12. vis[x] = true;
13. for(int j = 0; j < n; j++) dp[j] = min(dp[j], dp[x]+cost[x][j]);
14. }
15. }

【O(m)】m是边数，n是点数

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. const int INF = 0x3f3f3f3f, MAXN = 100+10, MAXM = 20000+10;
6. struct Edge{
7. int next,to,cost;
8. Edge(int a = 0, int b = 0, int c = 0){
9. next = a, to = b, cost = c;
10. }
11. }edge[MAXM];
12. struct NODE{
13. int dis;//这里的dis不一定是dis[u]
14. int u;
15. NODE(int a, int b){ dis = a, u = b;}
16. bool operator <(const NODE& a) const {return dis > a.dis;}
17. };
18. int tol,head[MAXN], dis[MAXN];
19. bool vis[MAXN];
20. void add\_edge(int u, int v, int cost)
21. {
22. edge[tol] = Edge(head[u],v,cost);
23. head[u] = tol++;
24. }
25. void dijkstra(int st)
26. {
27. priority\_queue<NODE> q;
28. clc(dis, 0x3f);
29. clc(vis, 0);
30. dis[st] = 0;
31. q.push(NODE(dis[st], st));
32. while(!q.empty())
33. {
34. NODE now = q.top(); q.pop();
35. int u = now.u;
36. if(dis[u] > now.dis) dis[u] = now.dis;
37. if(vis[u]) continue;
38. vis[u] = true;
39. for(int i = head[u]; ~i; i = edge[i].next)
40. {
41. int v = edge[i].to, cost = edge[i].cost;
42. if(dis[v] > now.dis+cost) q.push(NODE(now.dis+cost, v));
43. }
44. }
45. }

### SPFA

【下面是SPFA(就是bellman\_ford改进的) O(k\*m，k是常数)】可以求有负环的图

这样写一般情况下远小于 O（n\*m），但卡的话会使得复杂度变成n\*m，**求两点之间最短路不能在ed出队就返回答案，而要在队列为空返回dis[ed]**

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. const int INF = 0x3f3f3f3f, MAXN = 200+10,MAXM = 2000+10;
6. struct Edge{
7. int next,to,cost;
8. Edge(int a = 0, int b = 0, int c = 0){
9. next = a, to = b, cost = c;
10. }
11. }edge[MAXM];
12. int tol,head[MAXN], dis[MAXN], cnt[MAXN], n;
13. bool inq[MAXN];
14. void add\_edge(int u, int v, int cost)
15. {
16. edge[tol] = Edge(head[u],v,cost);
17. head[u] = tol++;
18. }
19. int spfa(int s, int ed)
20. {
21. queue<int> q;
22. clc(dis,0x3f);
23. clc(inq,0);
24. clc(cnt,0);
25. q.push(s);
26. inq[s] = true;
27. dis[s] = 0;
28. while(!q.empty())
29. {
30. int u = q.front();q.pop();
31. inq[u] = false;
32. for(int i = head[u]; ~i; i = edge[i].next)
33. {
34. int v = edge[i].to, cost = edge[i].cost;
35. if(INF != dis[u] && dis[v] > dis[u]+cost)
36. {
37. dis[v] = dis[u]+cost;
38. if(!inq[v])
39. {
40. q.push(v);
41. inq[v] = true;
42. if(++cnt[v] > n) return -1;//发现负圈(可以无限小)退出（也不一定返回false，需要根据题目意思），如果求最长路，发现正圈(可以无限大)时需要判断e.to能不能到达终点再返回；
43. }
44. }
45. }
46. }
47. if(INF == dis[ed]) return -1;
48. return dis[ed];
49. }

### floyd

【下面是floyd O()】用来求任意两点之间的最短路，上面的都是求单源最短路

把最外层的k去掉，每次遇到新的点N^2更新可以变成在线的floyd(HDU3631)

【求最短路】

1. void floyd(int s)
2. {
3. for(int i = 0; i < n; i++) dp[i][i] = 0;
4. for (int k = 0; k < n; k++)
5. {
6. for (int i = 0; i < n; i++)
7. {
8. for (int j = 0; j < n; j++)
9. {
10. dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);
11. }
12. }
13. }
14. }

【求有向图的传递闭包】就是两点之间有无通路

1. void floyd(int s)
2. {
3. for(int i = 0; i < n; i++) dp[i][i] = 1;
4. for (int k = 0; k < n; k++)
5. {
6. for (int i = 0; i < n; i++)
7. {
8. for (int j = 0; j < n; j++)
9. {
10. dp[i][j] = dp[i][j] || (dp[i][k]&&dp[k][j]);
11. }
12. }
13. }
14. }

## 最大流

最小割就是删掉权值最小的边让源点和汇点分别分在两个不连通的集合(就是把图分成两个集合，保证源点和汇点不连通，可以解决删掉权值最小的边刻意让某些点和另外点孤立，也就是堵住前面点到汇点的去路)

求最优值下边的限制(比如边尽量少可以让每条边cost\*(m+1)+1，这样maxflow/(m+1)就是最大流，maxflow%(m+1)就是最大流下最少边数，二分匹配尽量保持i-i匹配也可以这样做)

### 最大流(最小割)-dinic算法（非递归）

(\*m) Dinic实现二分图匹配复杂度为O(sqrt(n)\*m)

1. typedef LL type;
2. struct Dinic{
3. struct Edge{
4. int next,u,v;
5. type cap;//cap是剩余流量
6. }edge[MAXM<<1];
7. int head[MAXN], tol, s,t,d[MAXN];
8. void init()
9. {
10. clc(head,-1);
11. tol = 0;
12. }
13. void add\_edge(int u, int v, type cap, type cap2 = 0)//有向图传前3个，无向图传4个
14. {
15. Edge e1 = {head[u],u,v,cap};
16. edge[tol] = e1;
17. head[u] = tol++;
18. Edge e2 = {head[v],v,u,cap2};//无向图把这里的cap2=cap可以只建一条边(add\_edge一次)
19. edge[tol] = e2;
20. head[v] = tol++;
21. }
22. bool bfs()
23. {
24. clc(d,-1);
25. queue<int> q;
26. q.push(s);
27. d[s] = 0;
28. while(!q.empty())
29. {
30. int u = q.front();q.pop();
31. for(int i = head[u]; ~i; i = edge[i].next)
32. {
33. int v = edge[i].v;
34. if(-1 == d[v] && edge[i].cap)
35. {
36. d[v] = d[u]+1;
37. q.push(v);
38. if(~d[t]) return true;
39. }
40. }
41. }
42. return false;
43. }
44. int stack[MAXN],cur[MAXN],top;
45. type maxflow(int s, int t)
46. {
47. this->s = s, this->t = t;
48. type ans = 0;
49. while(bfs())
50. {
51. memcpy(cur,head,sizeof(head));
52. int u = s;
53. top = 0;
54. while(1)
55. {
56. if(u == t)
57. {
58. type flow = INF;
59. int loc;//loc 表示 stack 中 cap 最小的边
60. rep(i,0,top)
61. {
62. if(flow > edge[stack[i]].cap)
63. {
64. flow = edge[stack[i]].cap;
65. loc = i;
66. }
67. }
68. rep(i,0,top)
69. {
70. edge[stack[i]].cap -= flow;
71. edge[stack[i]^1].cap += flow;
72. }
73. ans += flow;
74. top = loc;
75. u = edge[stack[top]].u;
76. }
77. for(int i = cur[u]; ~i; cur[u] = i = edge[i].next) //cur[u] 表示u所在能增广的边的下标
78. if(edge[i].cap && (d[u]+1 == d[edge[i].v])) break;
79. if(~cur[u])
80. {
81. stack[top++] = cur[u];
82. u = edge[cur[u]].v;
83. }
84. else
85. {
86. if(!top) break;
87. d[u] = -1;
88. u = edge[stack[--top]].u;
89. }
90. }
91. }
92. return ans;
93. }
94. }dinic;

### 最大流(最小割)-EdmondsKarp

【增广路EdmondsKarp算法(n\*)】n是点数，m是有向边数

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define clc(a,b) memset(a,b,sizeof(a))
3. #define MAXN 410
4. #define INF 0x3f3f3f3f
5. struct EK{
6. struct E{
7. int from, to, cap, flow;
8. E(int a, int b, int c, int d){
9. from = a, to = b, cap = c, flow = d;
10. }
11. };
12. int a[MAXN], p[MAXN];
13. vector<int> g[MAXN];
14. vector<E> edge;
15. void init(int n)
16. {
17. edge.clear();
18. rep(i,0,n) g[i].clear();
19. }
20. void add\_edge(int from, int to, int cap)
21. {
22. edge.push\_back(E(from, to, cap, 0));
23. edge.push\_back(E(to, from, 0, 0));//反向限制为0
24. int num = edge.size();
25. g[from].push\_back(num - 2);
26. g[to].push\_back(num - 1);
27. }
28. int e\_k(int st, int ed)
29. {
30. int ans = 0;
31. while (1)
32. {
33. clc(a,0);
34. queue<int> q;
35. q.push(st);
36. a[st] = INF;
37. while (!q.empty())
38. {
39. int x = q.front(); q.pop();
40. int len = g[x].size();
41. rep(i,0,len)
42. {
43. E& e = edge[g[x][i]];
44. if (!a[e.to] && e.cap > e.flow)
45. {
46. p[e.to] = g[x][i];
47. a[e.to] = min(a[x], e.cap - e.flow);
48. q.push(e.to);
49. }
50. }
51. if (a[ed]) break;
52. }
53. if (!a[ed]) break;
54. for (int u = ed; u != st; u = edge[p[u]].from)
55. {
56. edge[p[u]].flow += a[ed];
57. edge[p[u] ^ 1].flow -= a[ed];
58. }
59. ans += a[ed];
60. }
61. return ans;
62. }
63. }ek;

### 最大流(最小割)-isap

【递归简洁版(\*m)】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define INF 0x3f3f3f3f
6. typedef long long LL;
7. #define MAXN 20
8. #define MAXM 2010//记得两倍 有反向边
9. struct Isap{
10. int n,s,t,d[MAXN],num[MAXN];
11. //n是所有节点数量,s源点，t汇点
12. struct Edge{//不需要flow，cap即可，最后的flow=原来的cap-现在的cap
13. int next, v, cap;
14. Edge(int a = 0, int b = 0, int c = 0){next = a, v = b, cap = c;}
15. }edge[MAXM<<1];
16. int head[MAXN], tol;
17. void init(int n)
18. {
19. this->n = n;
20. clc(num,0);
21. clc(d,0);
22. clc(head,-1);
23. tol = 0;
24. }
25. void add\_edge(int u, int v, int cap)
26. {
27. edge[tol] = Edge(head[u], v, cap);
28. head[u] = tol++;
29. edge[tol] = Edge(head[v], u, 0);
30. head[v] = tol++;
31. }
32. int dfs(int u, int a)
33. {
34. if(t == u) return a;
35. int delta, mi = n-1,ans = 0;
36. for(int i = head[u]; ~i; i = edge[i].next)
37. {
38. Edge& e = edge[i];
39. if(!e.cap) continue;
40. if(d[e.v]+1 == d[u])
41. {
42. delta = dfs(e.v, min(a,e.cap));
43. if(delta)
44. {
45. e.cap -= delta;
46. edge[i^1].cap += delta;
47. ans += delta;
48. a -= delta;
49. }
50. else if(d[s] >= n || !a)
51. return ans;
52. }
53. mi = min(mi, d[e.v]);
54. }
55. if(!ans)
56. {
57. if(!--num[d[u]])
58. {
59. d[s] = n;
60. return 0;
61. }
62. num[d[u] = mi+1]++;
63. }
64. return ans;
65. }
66. int maxflow(int s, int t)
67. {
68. this->s = s,this->t = t;
69. int flow = 0;
70. num[0] = n;
71. while(d[s] < n) flow += dfs(s,INF);
72. return flow;
73. }
74. }isap;

【ISAP + BFS初始化 + 栈优化(\*m)】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define clc(a,b) memset(a,b,sizeof(a))
3. #define MAXN 55
4. #define INF 0x3f3f3f3f
5. struct ISAP{
6. struct Edge{
7. int from, to, cap, flow;
8. Edge(int a, int b, int c, int d){
9. from = a, to = b, cap = c, flow = d;
10. }
11. };
12. int p[MAXN]/\*可增广的上一条弧\*/, num[MAXN]/\*距离标号计数\*/, d[MAXN]/\*起点到i距离\*/,cur[MAXN]/\*当前弧下标\*/;
13. int n, s, t;//节点数，边数，起点，终点
14. vector<int> g[MAXN];//邻接表
15. vector<Edge> edge;
16. bool vis[MAXN];
17. void init(int n)
18. {
19. this->n = n;
20. edge.clear();
21. rep(i,0,n) g[i].clear();
22. clc(num,0);
23. clc(cur,0);
24. }
25. void add\_edge(int from, int to, int cap)
26. {
27. edge.push\_back(Edge(from,to,cap,0));
28. edge.push\_back(Edge(to,from,0,0));
29. int m = edge.size();
30. g[from].push\_back(m-2);
31. g[to].push\_back(m-1);
32. }
33. int augment()
34. {
35. int x = t, a = INF;
36. while(x != s)
37. {
38. Edge& e = edge[p[x]];
39. a = min(a, e.cap-e.flow);
40. x = edge[p[x]].from;
41. }
42. x = t;
43. while(x != s)
44. {
45. edge[p[x]].flow += a;
46. edge[p[x]^1].flow -= a;
47. x = edge[p[x]].from;
48. }
49. return a;
50. }
51. bool bfs()
52. {
53. memset(vis,0,sizeof(vis));
54. queue<int> q;
55. q.push(t);
56. d[t] = 0;
57. vis[t] = 1;
58. while(!q.empty())
59. {
60. int x = q.front(); q.pop();
61. int sz = g[x].size();
62. rep(i,0,sz)
63. {
64. Edge& e = edge[g[x][i]];
65. if(!vis[e.to])
66. {
67. vis[e.to] = 1;
68. d[e.to] = d[x]+1;
69. q.push(e.to);
70. }
71. }
72. }
73. return vis[s];
74. }
75. int maxflow(int s, int t)
76. {
77. this->s = s, this->t = t;
78. int flow = 0;
79. bfs();//bfs可以不加;需要clc(d,0),并且下面一行换成num[0] = n;
80. rep(i,0,n) num[d[i]]++;
81. int x = s;
82. while(d[s] < n)
83. {
84. if(x == t)
85. {
86. flow += augment();
87. x = s;
88. continue;
89. }
90. int ok = 0,sz = g[x].size();
91. rep(i,cur[x],sz)
92. {
93. Edge& e = edge[g[x][i]];
94. if(e.cap > e.flow && d[x] == d[e.to]+1)
95. {
96. ok = 1;
97. p[e.to] = g[x][i];
98. cur[x] = i;
99. x = e.to;
100. break;
101. }
102. }
103. if(!ok)
104. {
105. int sum = n-1;
106. rep(i,0,sz)
107. {
108. Edge& e = edge[g[x][i]];
109. if(e.cap > e.flow) sum = min(sum,d[e.to]);
110. }
111. if(--num[d[x]] == 0) return flow;//gap优化
112. num[d[x] = sum+1]++;
113. cur[x] = 0;
114. if(x != s) x = edge[p[x]].from;
115. }
116. }
117. return flow;
118. }
119. }isap;

## 最小费用最大流

**每条边的费用是指单位流量产生的费用，不是满流时的费用。**

**最后结果为满流时的总费用。**

### 普通SPFA版

1. struct MCMF{
2. int s,t;
3. struct Edge{
4. int next, to, cap, cost;
5. }edge[MAXM<<1];//记得多开一倍边数(有反向边)
6. int tol,head[MAXN];
7. bool inq[MAXN];
8. int d[MAXN]/\*spfa\*/, p[MAXN]/\*上一条边\*/,pre[MAXN]/\*上一个点\*/;
9. void init(){
10. clc(head,-1);
11. tol = 0;
12. }
13. void add\_edge(int from, int to, int cap, int cost)
14. {
15. Edge e = {head[from],to,cap,cost};
16. edge[tol] = e;
17. head[from] = tol++;
18. Edge e2 = {head[to],from,0,-cost};
19. edge[tol] = e2;
20. head[to] = tol++;
21. }
22. bool spfa()
23. {
24. queue<int> q;
25. clc(d,0x3f);
26. clc(inq,0);
27. d[s] = 0, inq[s] = true,p[s] = 0;
28. q.push(s);
29. while(!q.empty())
30. {
31. int u = q.front();q.pop();
32. inq[u] = false;
33. for(int i = head[u]; ~i; i = edge[i].next)
34. {
35. int v = edge[i].to, c = edge[i].cost;
36. if(edge[i].cap > 0 && d[v] > d[u]+c)
37. {
38. d[v] = d[u]+c;
39. p[v] = i;
40. pre[v] = u;
41. if(!inq[v]) q.push(v), inq[v] = true;
42. }
43. }
44. }
45. return INF != d[t];
46. }
47. //需要保证初始网络没有负圈，返回最小花费，flow是最大流量
48. //如需要求最大费用最大流，则只要把边权改为负的，最后答案为-ans即可
49. int mincostmaxflow(int st, int ed,int &flow)
50. {
51. s = st, t = ed;
52. flow = 0;
53. int cost = 0;
54. while(spfa())
55. {
56. int tmp = INF;
57. for(int i = t; i != s; i = pre[i]) tmp = min(tmp,edge[p[i]].cap);
58. flow += tmp;
59. cost += d[t]\*tmp;
60. for (int i = t ; i != s; i = pre[i])
61. {
62. edge[p[i]].cap -= tmp;
63. edge[p[i]^1].cap += tmp;
64. }
65. }
66. return cost;
67. }
68. }mcmf;

### ZKW费用流

1. struct ZKW\_MinCostMaxFlow
2. {
3. private:
4. int st, ed, ecnt, n;
5. int head[MAXN],cap[MAXM<<1],cost[MAXM<<1],to[MAXM<<1],nxt[MAXM<<1];//有反向边，记得两倍
6. int dis[MAXN];
7. void SPFA()
8. {
9. repe(i,0,n) dis[i] = INF;
10. priority\_queue<pair<int, int> > q;
11. dis[st] = 0;
12. q.push(make\_pair(0, st));
13. while(!q.empty()){
14. int u = q.top().second, d = -q.top().first;
15. q.pop();
16. if(dis[u] != d) continue;
17. for(int p = head[u]; p!=-1; p = nxt[p]){
18. int &v = to[p];
19. if(cap[p] && dis[v] > d + cost[p]){
20. dis[v] = d + cost[p];
21. q.push(make\_pair(-dis[v], v));
22. }
23. }
24. }
25. repe(i,0,n) dis[i] = dis[ed] - dis[i];
26. }
27. int minCost, maxFlow;
28. bool use[MAXN];
29. int add\_flow(int u, int flow)
30. {
31. if(u == ed)
32. {
33. maxFlow += flow;
34. minCost += dis[st] \* flow;
35. return flow;
36. }
37. use[u] = true;
38. int now = flow;
39. for(int p = head[u]; p!=-1; p = nxt[p])
40. {
41. int &v = to[p];
42. if(cap[p] && !use[v] && dis[u] == dis[v] + cost[p])
43. {
44. int tmp = add\_flow(v, min(now, cap[p]));
45. cap[p] -= tmp;
46. cap[p^1] += tmp;
47. now -= tmp;
48. if(!now) break;
49. }
50. }
51. return flow - now;
52. }
53. bool modify\_label()
54. {
55. int d = INF;
56. repe(u,0,n) if(use[u])
57. for(int p = head[u]; p!=-1; p = nxt[p])
58. {
59. int &v = to[p];
60. if(cap[p] && !use[v]) d = min(d, dis[v] + cost[p] - dis[u]);
61. }
62. if(d == INF) return false;
63. repe(i,0,n) if(use[i]) dis[i] += d;
64. return true;
65. }
66. public:
67. void init()
68. {
69. clc(head,-1);
70. ecnt = 2;
71. }
72. void add\_edge(int u, int v, int cc, int ww)//点u-v,cc-容量,ww-单位流量费用
73. {
74. cap[ecnt] = cc; cost[ecnt] = ww; to[ecnt] = v;
75. nxt[ecnt] = head[u]; head[u] = ecnt++;
76. cap[ecnt] = 0; cost[ecnt] = -ww; to[ecnt] = u;
77. nxt[ecnt] = head[v]; head[v] = ecnt++;
78. }
79. int zkw(int ss, int tt, int nn, int &flow)//ss-源点，tt-汇点,nn-点的总数(编号0~n)
80. {
81. st = ss, ed = tt, n = nn;
82. minCost = maxFlow = 0;
83. SPFA();
84. while(true)
85. {
86. while(true)
87. {
88. repe(i,0,n) use[i] = 0;
89. if(!add\_flow(st, INF)) break;
90. }
91. if(!modify\_label()) break;
92. }
93. flow = maxFlow;
94. return minCost;
95. }
96. }zkwflow;

## 二分图匹配模板

### 无权图最大匹配

1）一个二分图中的最大匹配数=这个图中的最小点覆盖数

König定理是一个二分图中很重要的定理，它的意思是，一个二分图中的最大匹配数等于这个图中的最小点覆盖数。如果你还不知道什么是最小点覆盖，我也在这里说一下：假如选了一个点就相当于覆盖了以它为端点的所有边，你需要选择最少的点来覆盖所有的边。

2）最小路径覆盖＝｜G｜－最大匹配数

在一个N\*N的有向图中，路径覆盖就是在图中找一些路经，使之覆盖了图中的所有顶点，且任何一个顶点有且只有一条路径与之关联；（如果把这些路径中的每条路径从它的起始点走到它的终点，那么恰好可以经过图中的每个顶点一次且仅一次）；如果不考虑图中存在回路，那么每每条路径就是一个弱连通集．由上面可以得出：

1.一个单独的顶点是一条路径；

2.如果存在一路径p1,p2,......pk，其中p1 为起点，pk为终点，那么在覆盖图中，顶点p1,p2,......pk不再与其它的 顶点之间存在有向边． 最小路径覆盖就是找出最小的路径条数使之成为G的一个路径覆盖． 路径覆盖与二分图匹配的关系：最小路径覆盖＝｜G｜－最大匹配数；

3）二分图最大独立集=顶点数-二分图最大匹配

独立集：图中任意两个顶点都不相连的顶点集合。

【dinic算法O（\*m）】

Dinic实现二分图匹配复杂度为O(\*m) 二分图一般情况是有向的(s->左->右->t)cap = = 1

所有正向边 cap变为0则说明使用了该边，遍历出这些边就确定了哪些点匹配

【匈牙利算法(邻接表)(n\*m)】n表示顶点数，m表示边数

1. bool vis[MAXN];
2. int lin[MAXN], n;
3. vector<int> g[MAXN];
4. //n是左边点数的数量，g[i]：i表示左边的点标号，g[i]里面的是右边的点标号
5. bool dfs(int u)
6. {
7. for(int i = 0; i < g[u].size(); i++)
8. {
9. int v = g[u][i];
10. if(!vis[v])
11. {
12. vis[v] = true;
13. if(-1 == lin[v] || dfs(lin[v]))
14. {
15. lin[v] = u;
16. return true;
17. }
18. }
19. }
20. return false;
21. }
22. int hungary()
23. {
24. int ans = 0;
25. memset(lin,-1,sizeof(lin));
26. for(int i = 0; i < n; i++)
27. {
28. memset(vis,0,sizeof(vis));
29. if(dfs(i)) ans++;
30. }
31. return ans;
32. }

【Hopcroft-Carp算法(\*m)】

1. #define rep(i,a,n) for(int i = a; i < n; i++)
2. #define repe(i,a,n) for(int i = a; i <= n; i++)
3. #define per(i,n,a) for(int i = n; i >= a; i--)
4. #define clc(a,b) memset(a,b,sizeof(a))
5. #define INF 0x3f3f3f3f
6. typedef long long LL;
7. #define MAXN 510
8. #define MAXM 250100
9. struct Edge{
10. int next, v;
11. Edge(int a = 0, int b = 0){next = a, v = b;}
12. }edge[MAXM];
13. int head[MAXN], tol, n, dis, dx[MAXN], dy[MAXN], mx[MAXN], my[MAXN];
14. bool vis[MAXN];
15. void add\_edge(int u, int v)
16. {
17. edge[tol] = Edge(head[u], v);
18. head[u] = tol++;
19. }
20. bool bfs()
21. {
22. queue<int> q;
23. dis = INF;
24. clc(dx,-1);
25. clc(dy,-1);
26. rep(i,0,n)
27. {
28. if(-1 == mx[i])
29. {
30. q.push(i);
31. dx[i] = 0;
32. }
33. }
34. while(!q.empty())
35. {
36. int u = q.front();q.pop();
37. if(dx[u] > dis) break;
38. for(int i = head[u]; ~i; i = edge[i].next)
39. {
40. int v = edge[i].v;
41. if(-1 == dy[v])
42. {
43. dy[v] = dx[u]+1;
44. if(-1 == my[v]) dis = dy[v];
45. else
46. {
47. dx[my[v]] = dy[v]+1;
48. q.push(my[v]);
49. }
50. }
51. }
52. }
53. return dis != INF;
54. }
55. bool dfs(int u)
56. {
57. for(int i = head[u]; ~i; i = edge[i].next)
58. {
59. int v = edge[i].v;
60. if(!vis[v] && dx[u]+1 == dy[v])
61. {
62. vis[v] = true;
63. if(-1 != my[v] && dy[v] == dis) continue;
64. if(-1 == my[v] || dfs(my[v]))
65. {
66. my[v] = u;
67. mx[u] = v;
68. return true;
69. }
70. }
71. }
72. return false;
73. }
74. int maxmatch()
75. {
76. int ans = 0;
77. clc(mx,-1);
78. clc(my,-1);
79. while(bfs())
80. {
81. clc(vis,0);
82. rep(i,0,n)
83. {
84. if(-1 == mx[i] && dfs(i))
85. ans++;
86. }
87. }
88. return ans;
89. }

### 有权图最佳完美匹配

图中圈的并就是二分图匹配(左右两边都是i点，看成拆点(网络流容易超时))

【KM算法 矩阵()】

1. int g[MAXN][MAXN];//二分图描述,纵坐标为左边，横坐标为右边，这里下标从1开始的
2. struct KM{
3. struct NODE{
4. int to,w;
5. NODE(int a, int b){
6. to = a, w = b;
7. }
8. };
9. int nx,ny;//两边的点数
10. int linker[MAXN],lx[MAXN],ly[MAXN];//y中各点匹配状态，x,y中的点标号
11. int slack[MAXN];
12. bool visx[MAXN],visy[MAXN];
13. bool dfs(int x)
14. {
15. visx[x] = true;
16. repe(y,1,ny)
17. {
18. if(visy[y]) continue;
19. int tmp = lx[x]+ly[y]-g[x][y];
20. if(!tmp)
21. {
22. visy[y] = true;
23. if(-1 == linker[y] || dfs(linker[y]))
24. {
25. linker[y] = x;
26. return true;
27. }
28. }
29. else if(slack[y] > tmp)
30. slack[y] = tmp;
31. }
32. return false;
33. }
34. int max\_km(int nx, int ny)//传入两边点数
35. {
36. this->nx = nx, this->ny = ny;
37. clc(linker,-1);
38. clc(ly,0);
39. repe(i,1,nx)
40. {
41. lx[i] = -INF;
42. repe(j,1,ny)
43. if(g[i][j] > lx[i])
44. lx[i] = g[i][j];
45. }
46. repe(x,1,nx)
47. {
48. clc(slack,0x3f);
49. while(1)
50. {
51. clc(visx,0);
52. clc(visy,0);
53. if(dfs(x))break;
54. int d = INF;
55. repe(i,1,ny)
56. {
57. if(!visy[i] && d > slack[i])
58. d = slack[i];
59. }
60. repe(i,1,nx) if(visx[i]) lx[i] -= d;
61. repe(i,1,ny)
62. {
63. if(visy[i]) ly[i] += d;
64. else slack[i] -= d;
65. }
66. }
67. }
68. //最后统计最大值，加负号就是最小值，然后要把所有g[][]\*-1
69. int ans = 0;
70. repe(i,1,ny)
71. {
72. //如果求最小值就是-INF，最大值才是INF
73. if(-INF == g[linker[i]][i]) return -1;//不存在完美匹配，返回-1，注释掉这句可以求不完全匹配下的最小/大费用
74. if(~linker[i]) ans += g[linker[i]][i];
75. }
76. return -ans;
77. }
78. }km;

## 差分约束

【题意】给你一个N\*M的矩阵，求两列数a1,a2,a3...an 和 b1,b2.....bm使得对矩阵中的每个数进行下面的操作之后的值在[L,U]之间，操作为：a[i] \* m[i][j] / b[j]。  N,M<=400

【分析】第一题差分约束，学习了链式前向星存图(比vector快很多这题可以快200ms+)

[差分约束的描述](http://www.cnblogs.com/void/archive/2011/08/26/2153928.html)： （注意：查分约束对于每个约束 u-v<=c 可以建v->u花费为c的边,<=spfa求最小值而实际问题的最大值，>=spfa求最大值而实际问题的最小值，如果是<或者>则必须通过加减一来变成<=或者>=）

由题意可知，对于矩阵中的每个元素要满足的条件是：L <= a[i] \* m[i][j] / b[j] <= U ，这样我们就可以得到下面的两个式子：L\*b[j] <= a[i]  \* m[i][j]  和 a[i] \* m[i][j]  <= U\*b[j] ，因为差分约束中dis[]前面没有系数，为了把系数取消掉，我们可以用对式子两边取对数，就可以得到：log(b[j])  - log( a[i] ) <= log(U/m[i][j]) ，同理可以得到另外一个：log(b[j]) - log(a[i]) <= -log(L/m[i][j])最后用spfa判负环就可以得出答案了

注意：

判断有无解（负环）的时候，如果用spfa，不能用入队次数大于N来判断，会超时。

有如下两种比较可靠的方法（一般情况下）

1：某个点入队次数大于sqrt(N)的时候

2：所有入队次数大于T \*（N + M），其中T一般取2

【AC CODE】484ms

1. #include <bits/stdc++.h>
2. using namespace std;
3. #define rep(i,a,n) for(int i = a; i < n; i++)
4. #define repe(i,a,n) for(int i = a; i <= n; i++)
5. #define per(i,n,a) for(int i = n; i >= a; i--)
6. #define clc(a,b) memset(a,b,sizeof(a))
7. #define INF 0x3f3f3f3f
8. typedef long long LL;
9. #define MAXN 810
10. struct Edge{
11. int next,to;
12. double cost;
13. Edge(int a = 0,int b = 0, double c = 0){next = a, to = b, cost = c;}
14. }edge[MAXN\*MAXN];
15. int head[MAXN], cnt[MAXN], tol, n, m;
16. double dis[MAXN];
17. bool inq[MAXN];
18. void add\_edge(int from, int to, double cost)
19. {
20. edge[tol] = Edge(head[from],to,cost);
21. head[from] = tol++;
22. }
23. bool spfa(int s)
24. {
25. queue<int> q;
26. int nn = 2\*(n+m), cnt = 0;
27. rep(i,0,n+m) dis[i] = INF;
28. clc(inq,0);
29. inq[s] = true;
30. dis[s] = 0;
31. q.push(s);
32. while(!q.empty())
33. {
34. int u = q.front();q.pop();
35. inq[u] = false;
36. for(int i = head[u]; ~i; i = edge[i].next)
37. {
38. int v = edge[i].to;
39. double cost = edge[i].cost;
40. if(dis[u] < INF && dis[v] > dis[u]+cost)
41. {
42. dis[v] = dis[u]+cost;
43. if(!inq[v])
44. {
45. inq[v] = true;
46. q.push(v);
47. if(++cnt > nn) return false;
48. }
49. }
50. }
51. }
52. return true;
53. }
54. int main()
55. {
56. #ifdef SHY
57. freopen("e:\\1.txt","r",stdin);
58. #endif
59. double l,u;
60. while(~scanf("%d %d %lf %lf%\*c", &n, &m, &l, &u))
61. {
62. int a;
63. tol = 0;
64. clc(head,-1);
65. rep(i,0,n)
66. {
67. rep(j,0,m)
68. {
69. scanf("%d%\*c", &a);
70. add\_edge(i,j+n,-log(l/a));
71. add\_edge(j+n,i,log(u/a));
72. }
73. }
74. if(spfa(0)) puts("YES");
75. else puts("NO");
76. }
77. return 0;
78. }

# 动态规划

## 可以找零的多重背包+完全背包问题

1. const int INF = 0x3f3f3f3f, MAXN = 100+10, MAXM = 120\*120+10000+10;
2. /\*POJ3260;付款最多不超过maxv\*maxv+t(maxv是货币种类的最大值，t是购买物品的值)\*/
3. int v[MAXN],c[MAXN],dp[MAXM];
4. int main()
5. {
6. #ifdef SHY
7. freopen("e:\\1.txt", "r", stdin);
8. #endif
9. int n,t;
10. while(~scanf("%d %d%\*c", &n, &t))
11. {
12. int mv = -1;
13. rep(i,0,n)
14. {
15. scanf("%d%\*c", &v[i]);
16. if(mv < v[i]) mv = v[i];
17. }
18. int m = mv\*mv+t+1;//鸽巢定理
19. rep(i,0,n) scanf("%d%\*c", &c[i]);
20. /\*付款的混合背包\*/
21. clc(dp,0x3f);
22. dp[0] = 0;
23. rep(i,0,n)
24. {
25. if(c[i]\*v[i] >= m)
26. repe(j,v[i],m) dp[j] = min(dp[j],dp[j-v[i]]+1);
27. else
28. {
29. for(int j = 1; j <= c[i]; c[i] -= j, j<<=1)
30. {
31. int w = j\*v[i];
32. per(k,m,w) dp[k] = min(dp[k],dp[k-w]+j);
33. }
34. if(c[i] > 0)
35. {
36. int w = c[i]\*v[i];
37. per(k,m,w) dp[k] = min(dp[k],dp[k-w]+c[i]);
38. }
39. }
40. }
41. /\*倒过来的完全背包，用于找零;
42. 也可以用另外一个数组r[]记录正常的完全背包，最后扫一遍min{dp[i+m]+r[i]}\*/
43. rep(i,0,n)
44. {
45. per(j,m-v[i],0)
46. dp[j] = min(dp[j], dp[j+v[i]]+1);
47. }
48. if(INF != dp[t]) printf("%d\n", dp[t]);
49. else puts("-1");
50. }
51. return 0;
52. }

## LCS最长公共子序列

【我的记忆化搜索和递推模版】

[INPUT]

GCCCTAGCG

GCGCAATG

[OUTPUT]

5

GCCTG

O(len(a)\*len(b))

1. #include <bits/stdc++.h>
2. using namespace std;
3. #define MAXN 100
4. char a[MAXN], b[MAXN];
5. int lena, lenb, dp[MAXN][MAXN];//dp[i][j]记录字符串a[0~i-1]和字符串b[0~j-1]的最长子序列长度
6. struct NODE{
7. int x, y;
8. }f[MAXN][MAXN];
9. //f[i][j]记录dp[i][j]的父亲节点，也就是他是由max(dp[i-1][j],dp[i][j-1],dp[i-1][j-1]+sum(当a[i-1] == b[j-1]时sum = 1,否则为0))继承下来的
10. //【记忆化】寻找两个字符串的最大子序列长度，并记录路径
11. int dfs(int i, int j)
12. {
13. if (-1 != dp[i][j])
14. return dp[i][j];
15. if (0 == i || 0 == j)
16. return dp[i][j] = 0;
17. if (a[i-1] == b[j-1])
18. {
19. dp[i][j] = dfs(i-1,j-1)+1;
20. f[i][j].x = i-1;
21. f[i][j].y = j-1;
22. }
23. else
24. {
25. int buf1 = dfs(i-1,j), buf2 = dfs(i,j-1);
26. if (buf1 < buf2)
27. dp[i][j] = buf2,f[i][j].x = i, f[i][j].y = j-1;
28. else
29. dp[i][j] = buf1, f[i][j].x = i-1, f[i][j].y = j;
30. }
31. return dp[i][j];
32. }
33. //【递推】
34. int sloved(int la, int lb)
35. {
36. memset(dp,0,sizeof(dp));
37. for (int i = 1; i <= la; i++)
38. {
39. for (int j = 1; j <= lb; j++)
40. {
41. if (a[i-1] == b[j-1])
42. dp[i][j] = dp[i-1][j-1]+1, f[i][j].x = i-1, f[i][j].y = j-1;
43. else if(dp[i-1][j] < dp[i][j-1])
44. {
45. dp[i][j] = dp[i][j-1];
46. f[i][j].x = i;
47. f[i][j].y = j-1;
48. }
49. else
50. {
51. dp[i][j] = dp[i-1][j];
52. f[i][j].x = i-1;
53. f[i][j].y = j;
54. }
55. }
56. }
57. return dp[la][lb];
58. }
59. //输出其中一个最长子序列
60. void printans(int i, int j)
61. {
62. if (!i || !j)
63. return;
64. int x = f[i][j].x, y = f[i][j].y;
65. printans(x, y);
66. if (dp[x][y]+1 == dp[i][j])
67. printf ("%c", a[i-1]);
68. }
69. int main ()
70. {
71. #ifdef SHY
72. freopen("e:\\1.txt", "r", stdin);
73. #endif
74. while(gets(a))
75. {
76. gets(b);
77. lena = strlen(a);
78. lenb = strlen(b);
79. memset(dp,-1,sizeof(dp));
80. printf ("%d\n", sloved(lena, lenb));
81. printans(lena, lenb);
82. printf ("\n");
83. }
84. return 0;
85. }

## LIS(最长上升子序列两种算法模板)

【最长上升子序列的各种模板】

有两种复杂度的算法，的算法可以算出长度的同时记录每个选择的点，而n的算法只能算出长度

【的模板】

1. #include <cstdio>
2. #include <cstring>
3. #include <algorithm>
4. using namespace std;
5. #define MAXN 5010
6. int a[MAXN], dp[MAXN], f[MAXN];
7. //dp[i]表示以a[i]为终点的最长上升子序列的长度,f[i]记录最优解父亲节点
8. void pt(int s)//输出一种最长上升子序列
9. {
10. if (-1 == s)
11. return;
12. pt(f[s]);
13. printf ("%d ", a[s]);
14. }
15. int main ()
16. {
17. #ifdef SHY
18. freopen("e:\\1.txt", "r", stdin);
19. #endif
20. int n;
21. scanf ("%d%\*c", &n);
22. memset(f,-1,sizeof(f));
23. dp[0] = 1;//把只有一个数的序列最长子序列长度设置为1,边界处理
24. for (int i = 0; i < n; i++)
25. scanf ("%d%\*c", &a[i]);
26. int ans = 1, ss;
27. for (int i = 1; i < n; i++)
28. {
29. dp[i] = 1;
30. for (int j = 0; j < i; j++)//枚举终点不算的时候左边最长的子序列长度
31. if (a[j] < a[i] && dp[j]+1 > dp[i])
32. dp[i] = dp[j]+1, f[i] = j;
33. if (ans < dp[i])//因为最后一个点不一定是最长子序列的终点，所以要去所有点为终点的最长的那一个
34. ans = dp[i], ss = i;
35. }
36. printf ("%d\n", ans);
37. pt(ss);
38. return 0;
39. }

【n模板】可以用线段树离散化维护a值在用思想快速查找，这样也是n

1. int q[MAXN], a[MAXN], n;
2. //q[]是一个单调递增队列,用来贪心的记录当前下标的长度下,可以得到的一个末尾数字最小的子序列
3. int LIS()
4. {
5. int ans = 0;
6. clc(q,0x3f);
7. rep(i,0,n)
8. {
9. int k = lower\_bound(q,q+n,a[i])-q;//改成upper\_bound()就是最长不下降(允许相等)
10. q[k] = a[i];
11. ans = max(ans,k+1);//d[i] = k+1;d[i]表示以[0,i]的LIS且以i结尾
12. }
13. return ans;
14. }

## LCS转LIS

【题意】

求最大公共子序列(LCS)

【分析】

直接求LCS，n^2超时，所以要优化，因为序列中每个元素不重复，所以直接用个数字保存第一个序列每个元素的下标，然后只要记录第二个序列中每个元素在第一个序列中的下标，如果在第一个序列中没出现直接不用记录，这样就有一个由第一个序列和第二个序列共同元素组成的下标的一个新序列，然后对于原来两个序列的LCS其实就是这个新序列的LIS了。

## LCIS最长公共上升子序列模板

【分析】

（1）n^3模板

dp[i][j]表示序列a前i个和序列b前j个字符，并且以b[j]作为LCIS序列的末尾的LCIS的长度

转移方程

1) dp[i][j] = dp[i-1][j] (a[i] != b[j])//a[i] != a[j]时不满足两个序列公共，所以不会增加长度，又因为dp[][]是以b[j]作

为结尾的新序列下的长度，所以直接等于dp[i-1][j]的值

2) dp[i][j] = {dp[i-1][k] | 1 <= k < j} (a[i] == b[j])//只有a[i] == b[j]时才会有可能使得长度增加,然后对b[](其实主要

是对a[k] == b[m](k < j)的序列)进行LIS就可

（2）n^2模版

看看LIS中的条件：b[k]<b[j](1<=k<=j-1)&&a[i]==b[j] 可以推出a[i]>b[j](这里的j相当于下个循环的枚举的k)，

用一个maxn变量来记录dp[i-1][1~j]的最大值，可以省去LIS中的每次都重复查找最大值，这样可以把复杂度降

低一维。

（3）n^2滚动数组模板

因为dp[i][..]使用的都是上一次的dp[i-1][...]，当然可以像01背包那样用滚动数组优化，第二个循环不是倒序，因

为这里有点像完全背包那样，本次循环更新之后，还可以在本次基础上再更新；使用滚动数组就没办法输出路径

了，因为路径需要以前的状态才能输出。

【算法ZOJ】30ms//利用了下标1开始，所以下标为0的一直不会被改变始终是初始化中的0

1. #include <cstdio>
2. #include <cstring>
3. #include <algorithm>
4. using namespace std;
5. #define MAXN 503
6. struct NODE{
7. int x, y;
8. }f[MAXN][MAXN];
9. int a[MAXN], b[MAXN], dp[MAXN][MAXN], ex, ey;
10. void pt(int i, int j)
11. {
12. if (!i || !j)
13. return;
14. int x = f[i][j].x, y = f[i][j].y;
15. pt(x, y);
16. if (dp[x][y]+1 == dp[i][j])
17. printf ("%d ", b[j]);
18. }
19. int main ()
20. {
21. #ifdef SHY
22. freopen("e:\\1.txt", "r", stdin);
23. #endif
24. int t;
25. scanf ("%d%\*c", &t);
26. while(t--)
27. {
28. int na, nb, ans = 0;
29. ex = 0, ey = 0;
30. scanf ("%d%\*c", &na);
31. for (int i = 1; i <= na; i++)
32. scanf ("%d%\*c", &a[i]);
33. scanf ("%d%\*c", &nb);
34. for (int i = 1; i <= nb; i++)
35. scanf ("%d%\*c", &b[i]);
36. for (int i = 1; i <= na; i++)
37. {
38. int maxn = 0, x = 0, y = 0;
39. for (int j = 1; j <= nb; j++)
40. {
41. if (a[i] == b[j])
42. {
43. dp[i][j] = maxn+1;
44. f[i][j].x = x, f[i][j].y = y;
45. if (ans < dp[i][j])
46. ans = dp[i][j], ex = i, ey = j;
47. }
48. else
49. {
50. dp[i][j] = dp[i-1][j], f[i][j].x = i-1, f[i][j].y = j;
51. if (a[i] > b[j] && maxn < dp[i-1][j])
52. maxn = dp[i-1][j], x = i, y = j;
53. }
54. }
55. }
56. printf ("%d\n", ans);
57. pt(ex,ey);
58. printf ("\n");
59. if (t)
60. printf ("\n");
61. }
62. return 0;
63. }

## 数位DP

### 统计个数

统计区间[x,y]中满足位之间某种关系的数的个数，复杂度O()(b为进制数)

1. /\*这个是求：按位(10进制)出现，奇数出现偶数次，偶数出现奇数次的数的个数\*/
2. const int INF = 0x3f3f3f3f, MAXN = 21;
3. LL dp[MAXN][1<<10][1<<10];
4. int bit[MAXN];
5. bool ok(int odd, int even)//判断最终是否符合条件
6. {
7. rep(i,0,10)
8. {
9. if(0 == (odd&(1<<i)) && 0 == (even&(1<<i))) continue;
10. if((odd&(1<<i)) && (i&1)) return false;
11. if((even&(1<<i)) && !(i&1)) return false;
12. }
13. return true;
14. }
15. //len-数字位数(高到低),odd&even-dp参数,ismax是否当前查询数上限,z是否为0(包括前导零)
16. LL dfs(int len, int odd, int even, bool ismax, bool z)
17. {
18. if(!len) return ok(odd,even);
19. LL &ans = dp[len][odd][even];
20. if(!ismax && ~ans) return ans;
21. LL sum = 0;
22. int mx = ismax?bit[len]:9;
23. repe(i,0,mx)
24. {
25. int nxodd = odd,nxeven = even;
26. if(!z || i)//去除0以及前导0，但不能continue，后面还需要
27. {
28. if((0 == (odd&(1<<i)) && 0 == (even&(1<<i))))
29. nxodd |= 1<<i;
30. else if(odd&(1<<i))
31. nxodd ^= 1<<i, nxeven |= 1<<i;
32. else
33. nxeven ^= 1<<i, nxodd |= 1<<i;
34. }
35. sum += dfs(len-1,nxodd,nxeven,ismax&&i==mx,z&&0==i);
36. }
37. return ismax?sum:ans=sum;
38. }
39. LL sloved(LL n)//[0~n]满足条件的个数
40. {
41. int len = 0;
42. while(n)
43. {
44. bit[++len] = n%10;
45. n /= 10;
46. }
47. return dfs(len,0,0,1,1);
48. }

### 统计和(平方和)

1. /\*统计与7无关数的平方和(与7有关的数为：1、整数中某一位是7；2、整数的每一位加起来的和是7的整数倍；3、这个整数是7的整数倍；符合一个即可)\*/
2. const int INF = 0x3f3f3f3f, MAXN = 20;
3. const LL MOD = 1000000007;
4. struct NODE{
5. LL n,sum,qsum;//n-个数，sum-和，qsum-平方和
6. NODE(){
7. n = sum = qsum = 0;
8. }
9. }dp[MAXN][7][7];
10. bool vis[MAXN][7][7];
11. LL z[MAXN];
12. int bit[MAXN];
13. NODE dfs(int len, int s, int c,bool ismax)
14. {
15. if(!len)
16. {
17. NODE tmp;
18. if(s&&c) tmp.n = 1;
19. return tmp;
20. }
21. NODE &ans = dp[len][s][c];
22. if(!ismax && vis[len][s][c]) return ans;
23. NODE cnt;
24. int mx = ismax?bit[len]:9;
25. repe(i,0,mx)
26. {
27. if(7 == i) continue;
28. NODE tmp = dfs(len-1,(s+i)%7,(c\*10+i)%7,ismax&&i==mx);
29. LL p = i\*z[len]%MOD;
30. cnt.n = (cnt.n+tmp.n)%MOD;
31. cnt.sum = (cnt.sum+tmp.sum+ p\*tmp.n%MOD)%MOD;
32. cnt.qsum = (cnt.qsum+tmp.qsum+ p\*p%MOD\*tmp.n%MOD+2\*p\*tmp.sum%MOD)%MOD;
33. }
34. if(!ismax)
35. {
36. ans=cnt;
37. vis[len][s][c] = 1;
38. }
39. return cnt;
40. }
41. NODE sloved(LL n)
42. {
43. int len = 0;
44. while(n)
45. {
46. bit[++len] = n%10;
47. n /= 10;
48. }
49. return dfs(len,0,0,1);
50. }
51. int main()
52. {
53. #ifdef SHY
54. freopen("d:\\1.txt", "r", stdin);
55. #endif
56. int t;
57. scanf("%d", &t);
58. clc(vis,0);
59. z[1] = 1;
60. rep(i,2,20) z[i] = (z[i-1]\*10)%MOD;
61. while(t--)
62. {
63. LL x,y;
64. scanf("%lld %lld",&x, &y);
65. printf("%lld\n", (sloved(y).qsum-sloved(x-1).qsum+MOD)%MOD);
66. }
67. return 0;
68. }

## 树形DP

### 每个人最多只有一个上级，但可以有多个下,求最大价值

【题意】有一个公司举办一次party，每个人都有一个上下级关系，要求来到聚会的所有人不能有直接的上下级关系，也就是请来了a就不能请他的上级以及下级(每个人最多只有一个上级，但可以有多个下级)，人事处已评估每位员工的欢乐，请你求出选择哪些员工来能使欢乐度最大。

【分析】经典的树形DP，其实树形DP基本上和线性的DP差别不大，无非是它要你在树上进行DP，必须把树建立出来才能DP，然后就是根据儿子节点推算出父节点。

关于DP方程：由于每个员工只有两种选择--来或者不来，那么只要建立两个dp[i][2];dp[i][1]用来记录选择i节点的时候最大获得的价值，dp[i][0]记录不选择i节点的最大价值；

然后对于dp[i][1]，因为选择了i结点，所以不能再选择他的儿子节点，所以只要加上他的所有儿子节点不选择的时候的价值；也就是dp[i][1] += {dp[j][0]}+val[i](j是i的所有儿子节点)

对于dp[i][0]，因为没有选择i结点，所以他的所有儿子节点选择和不选择都可以，所以要加上它所有儿子结点选择与不选择的较大的那个；也就是dp[i][0] += {max(dp[j][0],dp[j][1])}

1. #include <bits/stdc++.h>
2. using namespace std;
3. #define MAXN 6005
4. vector <int> s[MAXN];//记录每个结点的所有儿子结点
5. int f[MAXN], dp[MAXN][2], val[MAXN];
6. void dfs(int root)
7. {
8. dp[root][1] = val[root];
9. for (int i = 0; i < s[root].size(); i++)//先把所有儿子节点的值算出来
10. dfs(s[root][i]);
11. for (int i = 0; i < s[root].size(); i++)//然后再计算当前节点选择和不选择的最大值
12. {
13. int j = s[root][i];
14. dp[root][0] += max(dp[j][0], dp[j][1]);
15. dp[root][1] += dp[j][0];
16. }
17. }
18. int main ()
19. {
20. #ifdef SHY
21. freopen("e:\\1.txt", "r", stdin);
22. #endif
23. int n;
24. while(~scanf ("%d%\*c", &n))
25. {
26. int a, b;
27. for (int i = 1; i <= n; i++)
28. {
29. scanf ("%d%\*c", &val[i]);
30. s[i].clear();
31. }
32. memset(dp,0,sizeof(dp));
33. memset(f,-1,sizeof(f));
34. while(~scanf ("%d %d%\*c", &a, &b) && a+b)
35. {
36. s[b].push\_back(a);
37. f[a] = b;
38. }
39. for (a = 1; a <= n && f[a] != -1; a++);//找到根节点
40. dfs(a);
41. printf ("%d\n", max(dp[a][1],dp[a][0]));
42. }
43. return 0;
44. }

### 结点占领花费取价值最大

【题意】

有一棵树根节点为1，占领节点有一个花费值wi和获得的价值vi，给你m个人(每个人相当于20点花费值)，并且必须放置的人\*20要大于等于当前节点wi才能占领获得价值vi；求最大能获得多少价值

【分析】

应该还算是比较简单的树形DP。具体看注释。

DP方程:dp[root][j] = max{dp[root][k] + dp[g[root].s[i]][j - k]};

1. /\*
2. dp[i][j]表示以i为根的子树派j个人能获得的最大的价值
3. 转移方程：
4. dp[root][j] = max{dp[root][k] + dp[g[root].s[i]][j - k]};
5. \*/
6. #include <bits/stdc++.h>
7. using namespace std;
8. #define MAXN 110
9. int w[MAXN], v[MAXN], dp[MAXN][MAXN];
10. struct NODE{
11. int len, s[MAXN];
12. }g[MAXN];
13. void dfs(int root, int fa, int sum)//root表示当前子树根节点，fa表示root的父亲节点用来防止往上走，sum表示当前能用的总人数
14. {
15. int a = w[root] / 20 + (w[root] % 20 ? 1 : 0);//计算当前占领当前节点需要多少几个人
16. if (a <= sum)
17. {
18. for (int i = a; i <= sum; i++)//赋给当前节点占领后最少人数到当前可以用的sum人之间的初始价值v[root]
19. dp[root][i] = v[root];
20. }
21. else//当前人数不够占领这个节点，直接退出
22. return;
23. for (int i = 0; i < g[root].len; i++)
24. {
25. if (fa == g[root].s[i]) continue;
26. dfs(g[root].s[i], root, sum - a);
27. for (int j = sum; j >= a; j--)//寻找root节点有j个人人时最大能获得多少价值
28. {
29. for (int k = a; k < j; k++)//分配给root节点j个人，给当前儿子节点j-k个人，寻找最大值
30. dp[root][j] = max(dp[root][j], dp[root][k] + dp[g[root].s[i]][j - k]);
31. }
32. }
33. }
34. int main()
35. {
36. #ifdef SHY
37. freopen("e:\\1.txt", "r", stdin);
38. #endif
39. int n, m;
40. while (~scanf("%d %d%\*c", &n, &m))
41. {
42. if (-1 == n && -1 == m)
43. break;
44. int a, b;
45. for (int i = 1; i <= n; i++)
46. g[i].len = 0;
47. memset(dp, 0, sizeof(dp));
48. for (int i = 1; i <= n; i++)
49. scanf("%d %d%\*c", &w[i], &v[i]);
50. for (int i = 1; i < n; i++)
51. {
52. scanf("%d %d%\*c", &a, &b);
53. g[a].s[g[a].len++] = b;
54. g[b].s[g[b].len++] = a;
55. }
56. if (!m)
57. {
58. puts("0");
59. continue;
60. }
61. dfs(1, -1, m);
62. printf("%d\n", dp[1][m]);
63. }
64. return 0;
65. }

## 最大连续子串(子矩阵)和

子串:

【方法一O(n)】: 扫描法:子串和为sum[j]-sum[i-1] (i <= j), 则维护min{sum[i-1]},每次比较ans = max(ans,sum[j]-mi)

【方法二O(n)】:动态规划:转移方程：dp[i] = max(dp[i-1]+a[i], a[i]);

====================================================

子矩阵(两种方法sum是不同的):

【方法一O()】扫描法:就是用二维前缀和sum[i][j] = sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]+a[i][j];然后枚举上下边界和左右向终点，线性扫描1~n，维护最小(才能保证最大)前缀mi，则答案为max{sum-mi},sum-mi就是一段区间的和。如果限制了区间长度(比如环面,因为要扩展\*4而最多只能用n\*n)则需要用单调队列，复杂度还是不变的。

1. repe(i,1,n)
2. {
3. repe(j,1,m)
4. {
5. scanf("%d%\*c", &a[i][j]);
6. sum[i][j] = sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1]+a[i][j];
7. }
8. }
9. int ans = -INF;
10. repe(i,1,n)
11. {
12. repe(j,i,n)
13. {
14. int mi = 0;//需要初始为0而不是第一格
15. repe(k,1,m)//这里就是一维情况的做法
16. {
17. int v = sum[j][k]-sum[i-1][k];//不是标准的子矩阵和了，因为矩阵左边是起始点了
18. ans = max(ans,v-mi);
19. mi = min(mi,v);
20. }
21. }
22. }

【方法二O()】动态规划:转移方程：dp[i] = max(dp[i-1]+sum[i][ed]-sum[i][st-1],sum[i][ed]-sum[i][st-1]);这里的dp[i]已经是压缩空间之后的，每次保留下来上一次的值；还是需要枚举上下边界以及左右向终点，这里的sum[i][j]表示i列上的前缀和(不是矩形的前缀和)；sum[i][j] = sum[i][j-1]+a[i][j];

1. repe(i,1,n)
2. {
3. repe(j,1,n)
4. {
5. scanf("%d%\*c", &a[i][j]);
6. sum[i][j] = sum[i][j-1]+a[i][j];
7. }
8. }
9. int ans = -0x3f3f3f3f;
10. repe(st,1,n)
11. {
12. repe(ed,st,n)
13. {
14. repe(i,1,n) //这里就是一维情况的做法
15. {
16. dp[i] = max(dp[i-1]+sum[i][ed]-sum[i][st-1],sum[i][ed]-sum[i][st-1]);
17. ans = max(ans,dp[i]);
18. }
19. }
20. }

## 最大M子段和

【题意】求最大M子段和

【分析】dp[i][j]表示前j项中i个子段和的最大值，且第i个子段包含a[j]

(a[j]可能包含在第i子段;也可能第i子段只有a[j]一个元素,那么只有把a[j]加到每一个dp[i-1][k]后面找最大值)

所以状态转移方程:dp[i][j] = max(dp[i][j-1], dp[i-1][k])(i-1 <= k <= j-1)。

可以写出：

1. for (int i = 1; i <= m; i++)
2. {
3. for (int j = 1; j <= n; j++)
4. {
5. dp[i][j] = dp[i][j - 1];
6. for (int k = i - 1; k < j; k++)
7. dp[i][j] = max(dp[i][j], dp[i - 1][k]);
8. dp[i][j] += a[j];
9. }
10. }

但是这样复杂度m\*n^2肯定超时，于是要优化。

这里发现for (int k = i-1; k < j; k++) dp[i][j] = max(dp[i][j], dp[i-1][k]);这个算max{dp[i-1][k]}在前面已经算出来了，只要保存下来就直接可以花费O(1)调用了；所以可以用另外一个数组last[]来分别保存上一次每个i-1~k的最大值max{dp[i-1][]}，在用一个变量maxn来辅助计算当前到j时的最大值。这样就可以写成如下O(N\*M)算法了:

1. for (int i = 1; i <= m; i++)
2. {
3. maxn = -INF;
4. for (int j = i; j <= n; j++)
5. {
6. dp[i][j] = max(dp[i][j - 1], last[j - 1]) + a[j];
7. last[j - 1] = maxn;
8. maxn = max(maxn, dp[i][j]);
9. }
10. }

又发现dp[i][j]的第一维没有用，利用滚动数组思想可以把第一维直接去掉，最终变成如下AC代码：

【AC代码】104ms(NYOJ) 640ms(HDU)

1. #include <cstdio>
2. #include <cstring>
3. #define MAXN 1000010
4. #define INF 0x3f3f3f3f
5. #define max(a,b) (a>b?a:b)
6. int ch, ans, f, a[MAXN], dp[MAXN], last[MAXN];
7. int in()
8. {
9. if((ch = getchar()) == '-') f = -1;
10. else f = 1;
11. while(ch < '0' || '9' < ch) ch = getchar();
12. ans = ch-'0';
13. while((ch = getchar()) >= '0' && '9' >= ch) ans = ans\*10+ch-'0';
14. return ans\*f;
15. }
16. int main()
17. {
18. #ifdef SHY
19. freopen("e:\\1.txt","r",stdin);
20. #endif
21. int t;
22. t = in();
23. while(t--)
24. {
25. int m = in(), n = in(), maxn;
26. for (int i = 1; i <= n; i++)
27. a[i] = in();
28. memset(dp,-0x3f,sizeof(dp));
29. memset(last,0,sizeof(last));
30. for (int i = 1; i <= m; i++)
31. {
32. maxn = -INF;
33. for (int j = i; j <= n; j++)
34. {
35. dp[j] = max(dp[j-1], last[j-1])+a[j];
36. last[j-1] = maxn;
37. maxn = max(maxn,dp[j]);
38. }
39. }
40. printf("%d\n", maxn);
41. }
42. return 0;
43. }

## 状压DP+插头DP

【题意】用1\*2或者L型的地板铺满n\*m的地面一共有多少中方案

【分析】

主要是枚举出两行之间的合法状态，很像炮兵摆放，但是这里的“炮兵”不是只占一个位置，所以不能通过简单的位运算判断是否合法，需要用dfs来判断。然后用dp[i][s]表示前i-1行都填满后i行的状态为s时可能的方案数，转移方程很简单:dp[i][now] += dp[i-1][last]。用dfs()枚举出所有合法的last能够用1\*2或者L型的地板填充到状态now，关于dfs的具体细节请看代码中的注释。

附上两组数据:

输入

9 9

2 5

输出

38896105985522272

24

1. #include <cstdio>
2. #include <cstring>
3. #define MAXN 10
4. typedef long long LL;
5. struct NDOE{//保存枚举的任意两行之间合法的状态
6. int now, last;
7. }p[79500];
8. int h,w,len;
9. LL dp[MAXN+1][1<<9];
10. //dp[i][s]表示前i-1行全部填满第i行状态为s时有多少种方案
11. //dfs()生成出:i行和i-1行还没被填满，1~i-2行已经被填满时的i行和i-1行的状态使得1~i行都被填满
12. //b1:i行当前这一列有没有被占，b2:i-1行当前这一列有没有被占
13. void dfs(int j, int now, int last, int b1, int b2)
14. {
15. if (j == w)
16. {
17. if (!b1 && !b2)
18. p[len].now = now, p[len++].last = last;
19. return;
20. }
21. dfs(j+1,now<<1|b1,last<<1|1^b2,0,0);/\*不放;i行可以不被填满(因为可以被i+1行填满)->now继承状态b1
22. 但是i-1行必须要被填满了,所以last填上1，但如果b2=1说明当前i-1行已经可以被前面填满,则这个last状态
23. 填上0，可以想像成还空这个位置去填补它，而这里枚举的是填补以前的，所以也就是中间会空出来。
24. 也就是有两种填入方式:一种是本来就填好了前面状态已经记录过，还有一种则需要现在去填,结果都要使得全部填满
25. 可以想象成你现在在生成一个地形，中间有些还没被填满的空格都是1\*2和L型的，最终可以用这两种填满它\*/
26. /\*一句话，当前该位置如果已经填了就不能再填，如果没填就可以填\*/
27. if (!b1)
28. {
29. dfs(j+1,now<<1|1,last<<1|1^b2,1,0);//1\*2横放
30. dfs(j+1,now<<1|1,last<<1|1^b2,1,1);//缺左上角
31. if(!b2)
32. {
33. dfs(j+1,now<<1|1,last<<1,0,0);//1\*2竖放
34. dfs(j+1,now<<1|1,last<<1,1,0);//缺右上角
35. dfs(j+1,now<<1|1,last<<1,0,1);//缺右下角
36. }
37. }
38. if (!b2)
39. dfs(j+1,now<<1|b1,last<<1,1,1);//缺左下角
40. }
41. int main()
42. {
43. #ifdef SHY
44. freopen("e:\\1.txt","r",stdin);
45. #endif
46. while(~scanf("%d %d%\*c", &h, &w))
47. {
48. if (h < w)
49. {
50. h ^= w;
51. w ^= h;
52. h ^= w;
53. }
54. len = 0;
55. dfs(0,0,0,0,0);
56. memset(dp,0,sizeof(dp));
57. dp[0][(1<<w)-1] = 1;
58. for (int i = 1; i <= h; i++)
59. {
60. for (int j = 0; j < len; j++)
61. dp[i][p[j].now] += dp[i-1][p[j].last];
62. }
63. printf("%lld\n", dp[h][(1<<w)-1]);
64. }
65. return 0;
66. }

## 第k大01背包

【题意】给定每个物品的价值和费用，求出在V费用下能获得的第k大的价值是多少

【分析】这是经典的第k大01背包，一开始天真的认为只要在原来01背包基础上增加一个额外的递减队列，然后每次出现新的价值时维护更新这个队列就可以求出第k大值，交上去直接WA，后来核对几组数据后发现01背包只记录最大的那个值，而过程中不会出现所有可以得到的价值(想得到这个应该只有枚举吧),然后只能增加一维记录前k大了。

【AC CODE】109ms

1. #include <cstdio>
2. #include <cstring>
3. #define MAXN 1002
4. #define MAXM 1002
5. #define MAXK 32
6. #define max(a,b) (a>b?a:b)
7. int v[MAXN],w[MAXN],dp[MAXM][MAXK],a[MAXK],b[MAXK],p;
8. /\*dp[j][k]表示前i个物品背包容量为j时第k大价值；
9. a[],b[]分别记录选择当前物品和不选择当前物品时候的前k大价值，然后每次合并两个序列的前k大即可\*/
10. int main()
11. {
12. #ifdef SHY
13. freopen("e:\\1.txt","r",stdin);
14. #endif
15. int t;
16. scanf("%d%\*c", &t);
17. while(t--)
18. {
19. int n,m;
20. scanf("%d %d %d%\*c", &n, &m, &p);
21. for(int i = 1; i <= n; i++)
22. scanf("%d%\*c", &v[i]);
23. for(int i = 1; i <= n; i++)
24. scanf("%d%\*c", &w[i]);
25. memset(dp,0,sizeof(dp));
26. memset(a,0,sizeof(a));
27. for(int i = 1; i <= n; i++)
28. {
29. for(int j = m; j >= w[i]; j--)
30. {
31. for(int k = 1; k <= p; k++)
32. {
33. a[k] = dp[j-w[i]][k]+v[i];
34. b[k] = dp[j][k];
35. }
36. //合并
37. int k = 1,p1 = 1,p2 = 1;
38. while(k <= p && (p1 <= p || p2 <= p))
39. {
40. if(a[p1] > b[p2]) dp[j][k] = a[p1++];
41. else dp[j][k] = b[p2++];
42. if(dp[j][k] != dp[j][k-1]) k++;
43. }
44. }
45. }
46. printf("%d\n", dp[m][p]);
47. }
48. return 0;
49. }

## 斜率DP

把具有单调性的转移决策使用单调队列优化，使得n^2的dp变成n；

例如：dp[i] = min{dp[j]+(sum[i]-sum[j])^2+M};

设决策k比决策j更优，则dp[k]+(sum[i]-sum[k])^2+M <= dp[j]+(sum[i]-sum[j])^2+M

化成j,k在左边，i在右边的斜率方程为 (dp[k]-dp[j]+sum[k]^2-sum[j]^2)/(2\*(sum[k]-sum[j])) <= s[i]

就能使用单调队列优化。

需要满足:决策sum[i]有关变量需要有单调性

1. #include <bits/stdc++.h>
2. using namespace std;
3. typedef long long LL;
4. #define rep(i,a,n) for(int i = a; i < n; i++)
5. #define repe(i,a,n) for(int i = a; i <= n; i++)
6. #define per(i,n,a) for(int i = n; i >= a; i--)
7. #define clc(a,b) memset(a,b,sizeof(a))
8. const int INF = 0x3f3f3f3f, MAXN = 500000+10;
9. int s[MAXN],dp[MAXN],q[MAXN];
10. inline int f(int k, int j){ return (dp[k]-dp[j]+s[k]\*s[k]-s[j]\*s[j]);}
11. inline int down(int k, int j){ return 2\*(s[k]-s[j]);}
12. int main()
13. {
14. int n,m;
15. while(~scanf("%d %d", &n, &m))
16. {
17. int a,x = 1, y = 1;
18. repe(i,1,n) scanf("%d", &a),s[i] = s[i-1]+a;
19. repe(i,1,n)
20. {
21. while(x < y && f(q[x+1],q[x]) <= s[i]\*down(q[x+1],q[x])) x++;
22. int j = q[x];
23. dp[i] = dp[j]+(s[i]-s[j])\*(s[i]-s[j])+m;
24. while(x < y && f(i,q[y])\*down(q[y],q[y-1]) <= f(q[y],q[y-1])\*down(i,q[y])) y--;
25. q[++y] = i;
26. }
27. printf("%d\n", dp[n]);
28. }
29. return 0;
30. }

# 搜索

## Dancing Links(DLX舞蹈链)

### 精确覆盖

精确覆盖就是一些行可以覆盖多个列，求没列都被覆盖，且只被一行覆盖一次的所有行。

1. /\*行标，列标都是1~n,和1~m
2. 需要倒过来恢复，因为顺着恢复的话后一个不在链表内，后一个的前指针指向他是错误的
3. 当然可以正着来，和删除一个顺序，但是会很慢，很多重复恢复，cnt计数会出错\*/
4. struct DLX{
5. private:
6. int lt[M],rt[M],up[M],down[M],row[M],col[M],tol;//舞蹈链(交叉十字循环双向链)内存池(大小是列数+1的数量)lt,rt,up,down是链表指针，row[i]结点i所在行，col[i]结点i所在列
7. int cnt[MAXN];//每一列还有多少个结点,用来优化搜索
8. void remove(int c)//删除c列含有1的全部行
9. {
10. lt[rt[c]] = lt[c];rt[lt[c]] = rt[c];
11. for(int i = down[c]; i != c; i = down[i])
12. {
13. for(int j = rt[i]; j != i; j = rt[j])
14. {
15. down[up[j]] = down[j];up[down[j]] = up[j];
16. --cnt[col[j]];
17. }
18. }
19. }
20. void reset(int c)//恢复被删除的c列含有1的全部行
21. {
22. for(int i = up[c]; i != c; i = up[i])
23. {
24. for(int j = lt[i]; j != i; j = lt[j])
25. {
26. down[up[j]] = up[down[j]] = j;
27. ++cnt[col[j]];
28. }
29. }
30. lt[rt[c]] = rt[lt[c]] = c;
31. }
32. public:
33. void init(int m)//总共有m列
34. {
35. repe(i,0,m)//第0行的虚拟结点
36. {
37. lt[i] = i-1,rt[i] = i+1;
38. up[i] = down[i] = i;
39. }
40. lt[0] = m;rt[m] = 0;//循环
41. tol = m+1;
42. clc(cnt,0);
43. }
44. void add\_row(int r,int len, int \*a)//向舞蹈链r行添加一整行结点
45. {
46. int ft = tol;
47. rep(i,0,len)
48. {
49. int c = a[i];
50. row[tol] = r,col[tol] = c;
51. lt[tol] = tol-1,rt[tol] = tol+1,up[tol] = up[c],down[tol] = c;
52. down[up[c]] = tol,up[c] = tol;
53. cnt[c]++,tol++;
54. }
55. if(ft != tol)
56. lt[ft] = tol-1, rt[tol-1] = ft;
57. }
58. int ansd,ans[MAXN];
59. bool dance(int d)
60. {
61. if(0 == rt[0])//所有列都被覆盖,找到答案,如要找能覆盖的最小长度，可以加估价函数f()强力剪枝,f()在重复覆盖模板中||直接在下面加if(d>=mi)return;
62. {
63. ansd = d;
64. return true;
65. }
66. int c = rt[0];//第一个没被覆盖的列c
67. for(int i = rt[0]; i; i= rt[i]) if(cnt[i] < cnt[c]) c = i;//优化，选择列上行数最小的那列，可以减少重复覆盖的概率，加快搜索
68. remove(c);//删除所有能覆盖c列的行
69. for(int i = down[c];i != c; i = down[i])//选择row[i]行来覆盖c列,并删除所有冲突行
70. {
71. ans[d] = row[i];
72. for(int j = rt[i]; j != i; j = rt[j]) remove(col[j]);//删除所有能覆盖col[j]列的行
73. if(dance(d+1)) return true;
74. for(int j = lt[i]; j != i; j = lt[j]) reset(col[j]);
75. }
76. reset(c);
77. return false;
78. }
79. /\*debug\*/
80. int out[MAXN][MAXN];
81. void pt()
82. {
83. clc(out,0);
84. for(int c = rt[0];c;c = rt[c])
85. {
86. for(int i = down[c];i!=c;i = down[i])
87. out[row[i]][c] = 1;
88. }
89. repe(i,1,n)
90. {
91. repe(j,1,m) printf("%d ", out[i][j]);
92. putchar('\n');
93. }
94. putchar('\n');
95. }
96. }dlx;

### 重复覆盖(不冲突子集)

【求集合中最大(小)互相**不冲突**子集】只要把序列建成n\*n的矩阵，1表示互相**冲突**(**自己和自己建1**)，求最大(小)重复覆盖就是答案、

1. /\*和精确覆盖大致一样，只是删除的时候不需要删除选择行所覆盖的所有点的其他行，只要删除选中行覆盖的所有列即可
2. 在找最小情况时，需要加上A\*启发函数f()来强力剪枝，否则时间巨慢FZU1686\*/
3. struct DLX
4. {
5. private:
6. int lt[M],rt[M],up[M],down[M],row[M],col[M],tol;
7. int cnt[MAXN];
8. void remove(int c)//删除列c所有元素
9. {
10. for(int i = down[c]; i != c; i = down[i])
11. lt[rt[i]] = lt[i],rt[lt[i]] = rt[i],--cnt[col[i]];
12. }
13. void reset(int c)//恢复列c
14. {
15. for(int i = up[c]; i != c; i = up[i])
16. lt[rt[i]] = rt[lt[i]] = i,++cnt[col[i]];
17. }
18. bool vis[MAXN];
19. int f()//最少还需要几行才能覆盖，最小值剪枝
20. {
21. int ret = 0;
22. for(int c = rt[0];c;c = rt[c]) vis[c] = true;
23. for(int c = rt[0];c;c = rt[c])
24. {
25. if(vis[c])
26. {
27. ret++;
28. vis[c] = false;
29. for(int i = down[c]; i != c; i = down[i])
30. for(int j = rt[i]; j != i; j = rt[j])
31. vis[col[j]] = false;
32. }
33. }
34. return ret;
35. }
36. int f\_max()//距离全部覆盖最多还需要几行，用来求最大值的f()剪枝
37. {
38. int ans = 0;
39. for(int c = rt[0];c;c = rt[c]) ans++;
40. return ans;
41. }
42. public:
43. void init(int maxc)
44. {
45. repe(i,0,maxc)
46. lt[i] = i-1,rt[i] = i+1,up[i] = down[i] = col[i] = i;
47. lt[0] = maxc,rt[maxc] = 0;
48. tol = maxc+1;
49. clc(cnt,0);
50. }
51. void add\_row(int r,int len, int \*a)
52. {
53. if(!len) return;
54. int ft = tol;
55. rep(i,0,len)
56. {
57. int c = a[i];
58. row[tol] = r,col[tol] = c;
59. lt[tol] = tol-1,rt[tol] = tol+1,up[tol] = up[c],down[tol] = c;
60. down[up[c]] = tol, up[c] = tol;
61. cnt[c]++,tol++;
62. }
63. if(ft != tol)
64. lt[ft] = tol-1,rt[tol-1] = ft;
65. }
66. int mi;
67. void dance(int d)
68. {
69. if(d+f() >= mi) return;//估价剪枝
70. if(0 == rt[0])
71. {
72. if(mi > d) mi = d;
73. return;
74. }
75. int c = rt[0];
76. for(int i = rt[0];i;i = rt[i]) if(cnt[c] > cnt[i]) c = i;
77. for(int i = down[c];i != c; i = down[i])//选择行row[i],并删除他所能覆盖的所有列
78. {
79. remove(i);
80. for(int j = rt[i]; j != i; j = rt[j]) remove(j);//删除行row[i]能覆盖的所有列
81. dance(d+1);
82. for(int j = lt[i]; j != i; j = lt[j]) reset(j);
83. reset(i);
84. }
85. }
86. }dlx;

# 其他

## 一些注意事项

1. double类型输入用%lf,输出用%f
2. for,while等循环内变量不要和循环外面变量重名，有些编译器会冲突(不报编译错误，运行时值会错)

## 解决递归爆栈问题

【强制扩栈】

【C++ (windows)(放在头文件前面)】

1. #pragma comment(linker, "/STACK:1024000000,1024000000")

【G++ (linux/windows上的g++)（放在主函数）】和系统有关 POJ,以及基于HUSTOJ的都可用

1. int \_\_size\_\_ = 256<<20;
2. char \*\_\_p\_\_ = (char \*)malloc(\_\_size\_\_)+\_\_size\_\_;
3. \_\_asm\_\_("mov %0,%%esp\n"::"r"(\_\_p\_\_));

【模拟栈(通用方法)】

比如这个dfs可以修改成如下模拟栈

1. void dfs1(int u)
2. {
3. int num = 0;
4. sz[u] = 1;
5. for(int i = head[u]; ~i; i = nxt[i])
6. {
7. int v = to[i];
8. if(v == fa[u]) continue;
9. fa[v] = u;dep[v] = dep[u]+1;
10. dfs1(v);
11. if(sz[v] > num) num = sz[v], son[u] = v;
12. sz[u] += sz[v];
13. }
14. }
15. stack<int> s;
16. void dfs1(int u)
17. {
18. while(!s.empty()) s.pop();
19. clc(vis,0);
20. s.push(u);
21. while(!s.empty())
22. {
23. int u = s.top();
24. if(vis[u])//从儿子结点返回
25. {
26. s.pop();
27. if(sz[u] > num[fa[u]]) num[fa[u]] = sz[u], son[fa[u]] = u;
28. sz[fa[u]] += sz[u];
29. continue;
30. }
31. /\*第一次进入结点u\*/
32. vis[u] = true;
33. sz[u] = 1;
34. num[u] = 0;
35. for(int i = head[u]; ~i; i = nxt[i])
36. {
37. int v = to[i];
38. if(v == fa[u]) continue;
39. fa[v] = u;dep[v] = dep[u]+1;
40. s.push(v);
41. }
42. }
43. }

## IO输入输出挂

由于windows下存换行标准是’\r\n’,而linux下是只有一个’\n’的，MAC是’\r’。

所以windows需要以二进制(比如freopen的’rb’,‘wb’)读入文件，然而很多windows的OJ都是以字符方式(‘r’,’w’)读入的,会把连续的”\r\n”变成单个’\n’, 在MAXIN>=4096时会在末尾加入所有读入的”\r\n”出现的次数cnt个字符,去掉最后cnt个字符才是’正确’的读入；(不完全一样,把所有”\r\n”都变成了’\n’了)

windows下制作的数据不加b输出的话会把’\n’变成”\r\n”。 linux下无影响。

### fread正负数

不能使用ps-1,可能会被截断,ps+1则可能越界

1. char buf[MAXIN], \*ps = buf, \*pe = buf+1;
2. inline void rnext(){
3. if(++ps == pe) pe = (ps = buf)+fread(buf,sizeof(char),sizeof(buf)/sizeof(char),stdin);
4. }
5. template <class T>
6. inline bool in(T &ans)
7. {
8. ans = 0;
9. T f = 1;
10. if(ps == pe) return false;
11. do{ rnext(); if('-' == \*ps) f = -1;} while(!isdigit(\*ps) && ps != pe);
12. if(ps == pe) return false;
13. do{ ans = (ans<<1)+(ans<<3)+\*ps-48;rnext();}while(isdigit(\*ps) && ps != pe);
14. ans \*= f;
15. return true;
16. }

### putchar输出(非负整数)

1. template <class T>
2. inline void out(T x) {
3. if(x>9) out(x/10);
4. putchar(x%10+'0');
5. }

### fwrite输出(字符，字符串，正负整数)

1. char bufout[MAXOUT], outtmp[50],\*pout = bufout, \*pend = bufout+MAXOUT;
2. inline void write(){ fwrite(bufout,sizeof(char),pout-bufout,stdout);pout = bufout;}/\*必须在程序结束时加 write()\*/
3. inline void out\_char(char c){ \*(pout++) = c;if(pout == pend) write();}
4. inline void out\_str(char \*s)
5. {
6. while(\*s){ \*(pout++) = \*(s++); if(pout == pend) write(); }
7. }
8. template <class T>
9. inline void out\_int(T x) {
10. if(!x){ out\_char('0');return;}
11. if(x < 0) x = -x,out\_char('-');
12. int len = 0;
13. while(x){ outtmp[len++] = x%10+48; x /= 10;}
14. outtmp[len] = 0;
15. for(int i = 0, j = len-1; i < j; i++,j--) swap(outtmp[i],outtmp[j]);
16. out\_str(outtmp);
17. }

### JAVA输入 大数A+B

1. **import** java.util.\*;
2. **import** java.math.BigInteger;
3. **import** java.io.\*;
4. public class Main {
5. **static** BufferedReader *in* = **new** BufferedReader(**new** InputStreamReader(System.*in*));
6. **static** BufferedWriter *out* = **new** BufferedWriter(**new** OutputStreamWriter(System.*out*));
7. **public** **static** **void** main(String args[])**throws** Exception{
8. **int** t = Integer.*parseInt*(*in*.readLine());
9. **for** (**int** i = 0; i < t; i++){
10. StringTokenizer st = **new** StringTokenizer(*in*.readLine());
11. BigInteger a = **new** BigInteger(st.nextToken());
12. BigInteger b = **new** BigInteger(st.nextToken());
13. *out*.write(a.add(b).toString());
14. *out*.newLine();
15. }
16. *out*.flush();//必须加
17. }
18. }

## 高精度

### C++手写BigInteger类

\* 完全大数模板

\* 输出a.print();

\* 注意这个输入不能自动去掉前导0的，可以先读入到char数组，去掉前导0，再用构造函数。

1. const int MAXN = 9999,MAXSIZE = 1010,DLEN = 4;//大数一个int存几位,一般不改,改动下面的4都要改
2. const int LEN = 54;//可以控制大数位数=DLEN\*LEN
3. struct BigInter{
4. int a[LEN];
5. int len;
6. BigInter(){ len = 1; memset(a, 0, sizeof(a)); } //构造函数
7. BigInter(const int); //将一个int类型的变量转化成大数
8. BigInter(const char\*); //将一个字符串类型的变量转化为大数
9. BigInter(const BigInter &); //拷贝构造函数
10. BigInter &operator=(const BigInter &); //重载赋值运算符，大数之间进行赋值运算
11. BigInter operator+(const BigInter &)const; //重载加法运算符，两个大数之间的相加运算
12. BigInter operator-(const BigInter &)const; //重载减法运算符，两个大数之间的相减运算
13. BigInter operator\*(const BigInter &)const; //重载乘法运算符，两个大数之间的相乘运算
14. BigInter operator/(const int &)const; //重载除法运算符，大数对一个整数进行相除运算
15. BigInter operator^(const int &)const; //大数的n次方运算
16. int operator%(const int &)const; //大数对一个int类型的变量进行取模运算
17. bool operator>(const BigInter &T)const; //大数和另一个大数的大小比较
18. bool operator>(const int &t)const; //大数和一个int类型的变量的大小比较
19. void print(); //输出大数
20. };
21. BigInter::BigInter(const int b) //将一个int类型的变量转化为大数
22. {
23. int c,d = b;
24. len = 0;
25. clc(a,0);
26. while(d > MAXN)
27. {
28. c = d-(d/(MAXN+1))\*(MAXN+1);
29. d = d/(MAXN+1);
30. a[len++] = c;
31. }
32. a[len++] = d;
33. }
34. BigInter::BigInter(const char \*s) //将一个字符串类型的变量转化为大数
35. {
36. int t,k,index,L;
37. clc(a,0);
38. L = strlen(s);
39. len = L/DLEN;
40. if(L%DLEN)len++;
41. index = 0;
42. for(int i = L-1; i >= 0; i -= DLEN)
43. {
44. t = 0;
45. k = i-DLEN+1;
46. if(k < 0)k = 0;
47. repe(j,k,i) t = t\*10+s[j]-'0';
48. a[index++] = t;
49. }
50. }
51. BigInter::BigInter(const BigInter &T) :len(T.len) //拷贝构造函数
52. {
53. clc(a,0);
54. rep(i,0,len) a[i] = T.a[i];
55. }
56. BigInter & BigInter::operator=(const BigInter &n) //重载赋值运算符，大数之间赋值运算
57. {
58. len = n.len;
59. clc(a,0);
60. rep(i,0,len) a[i] = n.a[i];
61. return \*this;
62. }
63. BigInter BigInter::operator+(const BigInter &T)const //两个大数之间的相加运算
64. {
65. BigInter t(\*this);
66. int i, big;
67. big = T.len > len ? T.len : len;
68. for (i = 0; i < big; i++)
69. {
70. t.a[i] += T.a[i];
71. if (t.a[i] > MAXN)
72. {
73. t.a[i + 1]++;
74. t.a[i] -= MAXN + 1;
75. }
76. }
77. if (t.a[big] != 0)
78. t.len = big + 1;
79. else t.len = big;
80. return t;
81. }
82. BigInter BigInter::operator-(const BigInter &T)const //两个大数之间的相减运算
83. {
84. bool flag;
85. BigInter t1, t2;
86. if (\*this > T)
87. {
88. t1 = \*this;
89. t2 = T;
90. flag = 0;
91. }
92. else
93. {
94. t1 = T;
95. t2 = \*this;
96. flag = 1;
97. }
98. int big = t1.len;
99. rep(i,0,big)
100. {
101. if(t1.a[i]<t2.a[i])
102. {
103. int j = i + 1;
104. while(t1.a[j] == 0) j++;
105. t1.a[j--]--;
106. while(j>i) t1.a[j--] += MAXN;
107. t1.a[i] += MAXN + 1 - t2.a[i];
108. }
109. else t1.a[i] -= t2.a[i];
110. }
111. t1.len = big;
112. while(t1.a[t1.len-1] == 0 && t1.len > 1)
113. {
114. t1.len--;
115. big--;
116. }
117. if(flag) t1.a[big - 1] = 0-t1.a[big - 1];
118. return t1;
119. }
120. BigInter BigInter::operator\*(const BigInter &T)const //两个大数之间的相乘
121. {
122. BigInter ret;
123. int i, j, up;
124. int temp, temp1;
125. for (i = 0; i < len; i++)
126. {
127. up = 0;
128. for (j = 0; j < T.len; j++)
129. {
130. temp = a[i] \* T.a[j] + ret.a[i + j] + up;
131. if (temp > MAXN)
132. {
133. temp1 = temp - temp / (MAXN + 1)\*(MAXN + 1);
134. up = temp / (MAXN + 1);
135. ret.a[i + j] = temp1;
136. }
137. else
138. {
139. up = 0;
140. ret.a[i + j] = temp;
141. }
142. }
143. if (up != 0)
144. ret.a[i + j] = up;
145. }
146. ret.len = i + j;
147. while (ret.a[ret.len - 1] == 0 && ret.len > 1)ret.len--;
148. return ret;
149. }
150. BigInter BigInter::operator/(const int &b)const //大数对一个整数进行相除运算
151. {
152. BigInter ret;
153. int down = 0;
154. per(i,len-1,0)
155. {
156. ret.a[i] = (a[i] + down\*(MAXN + 1)) / b;
157. down = a[i] + down\*(MAXN + 1) - ret.a[i] \* b;
158. }
159. ret.len = len;
160. while (ret.a[ret.len - 1] == 0 && ret.len > 1)
161. ret.len--;
162. return ret;
163. }
164. int BigInter::operator%(const int &b)const //大数对一个 int类型的变量进行取模
165. {
166. int d = 0;
167. per(i,len-1,0) d = (((LL)d\*(MAXN + 1)) % b + a[i]) % b;
168. return d;
169. }
170. BigInter BigInter::operator^(const int &n)const //大数的n次方运算
171. {
172. BigInter t, ret(1);
173. int i;
174. if (n < 0)exit(-1);
175. if (n == 0)return 1;
176. if (n == 1)return \*this;
177. int m = n;
178. while(m > 1)
179. {
180. t = \*this;
181. for (i = 1; (i << 1) <= m; i <<= 1)
182. t = t\*t;
183. m -= i;
184. ret = ret\*t;
185. if (m == 1)ret = ret\*(\*this);
186. }
187. return ret;
188. }
189. bool BigInter::operator>(const BigInter &T)const //大数和另一个大数的大小比较
190. {
191. int ln;
192. if (len > T.len)return true;
193. else if (len == T.len)
194. {
195. ln = len - 1;
196. while (a[ln] == T.a[ln] && ln >= 0)
197. ln--;
198. if (ln >= 0 && a[ln] > T.a[ln])
199. return true;
200. else
201. return false;
202. }
203. else
204. return false;
205. }
206. bool BigInter::operator>(const int &t)const //大数和一个int类型的变量的大小比较
207. {
208. BigInter b(t);
209. return \*this > b;
210. }
211. void BigInter::print() //输出大数
212. {
213. int i;
214. printf("%d", a[len - 1]);
215. for (i = len - 2; i >= 0; i--)
216. printf("%04d", a[i]);
217. printf("\n");
218. }

### JAVA -BigInteger

BigInteger add(BigInteger val) 返回其值为 (this + val) 的 BigInteger。

BigInteger subtract(BigInteger val) 返回其值为 (this - val) 的 BigInteger。

BigInteger multiply(BigInteger val) 返回其值为 (this \* val) 的 BigInteger。

BigInteger divide(BigInteger val) 返回其值为 (this / val) 的 BigInteger。

BigInteger remainder(BigInteger val) 返回其值为 (this % val) 的 BigInteger。

BigInteger[] divideAndRemainder(BigInteger val) 返回包含 (this / val) 后跟 (this % val) 的两个 BigInteger 的数组。

BigInteger abs() 返回其值是此 BigInteger 的绝对值的 BigInteger。

BigInteger and(BigInteger val) 返回其值为 (this & val) 的 BigInteger。

BigInteger andNot(BigInteger val) 返回其值为 (this & ~val) 的 BigInteger。

int bitCount() 返回此 BigInteger 的二进制补码表示形式中与符号不同的位的数量。

int bitLength() 返回此 BigInteger 的最小的二进制补码表示形式的位数，不包括 符号位。

BigInteger clearBit(int n) 返回其值与清除了指定位的此 BigInteger 等效的 BigInteger。

int compareTo(BigInteger val) 将此 BigInteger 与指定的 BigInteger 进行比较。

double doubleValue() 将此 BigInteger 转换为 double。

boolean equals(Object x) 比较此 BigInteger 与指定的 Object 的相等性。

BigInteger flipBit(int n) 返回其值与对此 BigInteger 进行指定位翻转后的值等效的 BigInteger。

float floatValue() 将此 BigInteger 转换为 float。

BigInteger gcd(BigInteger val) 返回一个 BigInteger，其值是 abs(this) 和 abs(val) 的最大公约数。

int getLowestSetBit() 返回此 BigInteger 最右端（最低位）1 比特的索引（即从此字节的右端开始到本字节中最右端 1 比特之间的 0 比特的位数）。

int hashCode() 返回此 BigInteger 的哈希码。

int intValue() 将此 BigInteger 转换为 int。

boolean isProbablePrime(int certainty) 如果此 BigInteger 可能为素数，则返回 true，如果它一定为合数，则返回 false。参数是对不确定性的度量，越大越精确。

long longValue() 将此 BigInteger 转换为 long。

BigInteger max(BigInteger val) 返回此 BigInteger 和 val 的最大值。

BigInteger min(BigInteger val) 返回此 BigInteger 和 val 的最小值。

BigInteger mod(BigInteger m) 返回其值为 (this mod m) 的 BigInteger。

BigInteger modInverse(BigInteger m) 返回其值为 (this-1 mod m) 的 BigInteger。

BigInteger modPow(BigInteger exponent, BigInteger m) 返回其值为 (thisexponent mod m) 的 BigInteger。

BigInteger negate() 返回其值是 (-this) 的 BigInteger。

BigInteger nextProbablePrime() 返回大于此BigInteger的可能为素数的第一个整数。

BigInteger not() 返回其值为 (~this) 的 BigInteger。

BigInteger or(BigInteger val) 返回其值为 (this | val) 的 BigInteger。

BigInteger pow(int exponent) 返回其值为 (thisexponent) 的 BigInteger。

static BigInteger probablePrime(int bitLength, Random rnd) 返回有可能是素数的、具有指定长度的正 BigInteger。

BigInteger setBit(int n) 返回其值与设置了指定位的此 BigInteger 等效的 BigInteger。

BigInteger shiftLeft(int n) 返回其值为 (this << n) 的 BigInteger。

BigInteger shiftRight(int n) 返回其值为 (this >> n) 的 BigInteger。

int signum() 返回此 BigInteger 的正负号函数。

boolean testBit(int n) 当且仅当设置了指定的位时，返回 true。

byte[] toByteArray() 返回一个字节数组，该数组包含此 BigInteger 的二进制补码表示形式。

String toString() 返回此 BigInteger 的十进制字符串表示形式。

String toString(int radix) 返回此 BigInteger 的给定基数的字符串表示形式。

static BigInteger valueOf(long val) 返回其值等于指定 long 的值的 BigInteger。

BigInteger xor(BigInteger val) 返回其值为 (this ^ val) 的 BigInteger。

### JAVA -BigDecimal

String a = in.next();

BigDecimal num = new BigDecimal(a);

res = num.setScale(2,BigDecimal.ROUND\_HALF\_EVEN).toString();//四舍五入保留两位小数，大多数和BigInteger差不多

equals()方法认为0.1和0.1是相等的，返回true，而认为0.10和0.1是不等的，结果返回false。方法compareTo()则认为0.1与0.1相等，0.10与0.1也相等。所以在从数值上比较两个BigDecimal值时，应该使用compareTo()而不是 equals()。

## 公式计算，给定年月日输出星期几

【公式(输入年月日获得星期几)】 只能计算1582年10.15到9999年12.31日的日期

1582年的10月5日-14日这十天历史上不存在

1. inline int get\_date(int y, int m, int d){//获得星期几
2. if(1 == m || 2 == m) y--,m+=12;//一定要加
3. return ((d+2\*m+3\*(m+1)/5+y+y/4-y/100+y/400)+1)%7;
4. }

## n!的位数

1. int fac\_len(int n)
2. {
3. double ans = 0;
4. repe(i,1,n) ans += log10(i);
5. return (int)ans+1;
6. }

## 时针和分针角度计算

【只有时针和分针，秒针始终为0】

设时间为h时m分,以12时0分开始为0度参考,分针的角度为m/60\*360度=6m度；时针除考虑h外,也要考虑m,角度应是h /12\*360度+m/60/12\*360度=(30h+0.5m)度,所以夹角便是两者的差=6m-(30h+0.5m)度= **abs(5.5m-30h)** 度(按360度算的角).如果需要变成劣角(0~180)则需要把大于180的用360减去。去掉abs就是可以有负角度的。

可以乘上12000转化为整数。

计算秒的话算到分针里面即可。

【时针分针秒针三个互相夹角】

时针和分针(h,m)：abs( (3600\*h-660\*m-11\*s)/120 )

时针和秒针(h,s)：abs( (3600\*h+60\*m-719\*s)/120 )

分针和秒针(m,s)：abs( (60\*m-59\*s)/10 )

算0~180度需要：while(ans > 180) ans = abs(ans-360);

## 整点正多边形判断(只有正方形)

整数点只能形成正4边形，其他都不能形成。

正方形判定: 4条边相等，且两个对角线相等(有正和斜两种情况)。

## STL

### 优先队列 priority\_queue

empty() 如果队列为空返回真

pop() 删除对顶元素

push() 加入一个元素

size() 返回优先队列中拥有的元素个数

top() 返回优先队列队顶元素

在默认的优先队列中，优先级高的先出队。在默认的 int 型中先出队的为较大的

数。

【小根堆】

1. priority\_queue<int, vector<int>, greater<int>> q;

【自定义比较函数：】

1. struct cmp
2. {
3. bool operator () (int x, int y)
4. {
5. return x > y; // x小的优先级高
6. //也可以写成其他方式，如： return p[x] > p[y]; 表示p[i] 小的优先级高
7. }
8. };
9. priority\_queue<int, vector<int>, cmp>q; //定义方法
10. //其中，第二个参数为容器类型。第三个参数为比较函数。

结构体排序：

1. struct node
2. {
3. int x, y;
4. friend bool operator < (node a, node b)
5. {
6. return a.x > b.x; //结构体中， x小的优先级高
7. }
8. };
9. priority\_queue<node>q; //定义方法
10. //在该结构中， y为值, x为优先级。
11. //通过自定义operator<操作符来比较元素中的优先级。
12. //在重载”<”时，最好不要重载”>”，可能会发生编译错误

### set,multiset,map…

set 和 multiset 用法一样，就是 multiset 允许重复元素。

元素放入容器时，会按照一定的排序法则自动排序，默认是按照 less<>排序规则来排序。不

能修改容器里面的元素值，只能插入和删除。

自定义 int 排序函数：（默认的是从小到大的，下面这个从大到小）

1. struct classcomp {
2. bool operator() (const int& lhs, const int& rhs) const
3. {
4. return lhs > rhs;
5. }
6. }; //这里有个逗号的，注意
7. multiset<int, classcomp> fifth; // class as Compare

上面这样就定义成了从大到小排列了。

结构体自定义排序函数：

（定义 set 或者 multiset 的时候定义了排序函数，定义迭代器时一样带上排序函数）

1. struct Node
2. {
3. int x, y;
4. };
5. struct classcomp//先按照 x 从小到大排序， x相同则按照y从大到小排序
6. {
7. bool operator() (const Node &a, const Node &b) const
8. {
9. if (a.x != b.x) return a.x < b.x;
10. else return a.y > b.y;
11. }
12. }; //注意这里有个逗号
13. multiset<Node, classcomp>mt;
14. multiset<Node, classcomp>::iterator it;

主要函数：

begin() 返回指向第一个元素的迭代器

clear() 清除所有元素

count() 返回某个值元素的个数

empty() 如果集合为空，返回 true

end() 返回指向最后一个元素的迭代器

erase() 删除集合中的元素 (参数是一个元素值，或者迭代器)

find() 返回一个指向被查找到元素的迭代器

insert() 在集合中插入元素

size() 集合中元素的数目

lower\_bound() 返回指向大于（或等于）某值的第一个元素的迭代器

upper\_bound() 返回大于某个值元素的迭代器

equal\_range() 返回集合中与给定值相等的上下限的两个迭代器

(注意对于 multiset 删除操作之间删除值会把所以这个值的都删掉，删除一个要用迭代器)

Map, multimap类似，只是map是要两个map<int,int>对应。

至于C++11的unordered\_set,unordered\_map, unordered\_multiset, unordered\_multimap只是内部实现用了hash表，所以是无序的，但是速度快很多，空间小很多。

非C++还可以用<ext/hash\_map> <ext/hash\_set>等调用hash\_map，还需要using namespace \_\_gnu\_cxx;

Unordered\_set和hash\_set都如下

1. struct NODE{
2. int a[6];
3. };
4. struct HASH{//自己定义的hash函数
5. int operator()(const NODE &s)const{
6. int ans = 0;
7. rep(i,0,6)
8. ans += s.a[i];
9. return ans%MOD;
10. }
11. };
12. struct cmp{//比较函数
13. bool operator()(const NODE& u, const NODE &t)const{
14. return !memcmp(u.a, t.a,sizeof(int)\*6);
15. }
16. };
17. unordered\_map<NODE,bool, HASH,cmp> vis;

### lower\_bound，upper\_bound, binary\_search

lower\_bound(d,d+len,a[i]),返回a[i]在d[]的位置(返回的是地址)，要是不存在a[i],返回a[i]应该插入的位置

upper\_bound(d,d+len,a[i]),返回a[i]在d[]的位置的下一个位置，要是不存在a[i],返回a[i]应该插入的位置

binary\_search(d,d+len,a[i]),查找a[i]是否在d[]中，存在返回true，不存在返回false

上面是数组，如果是容器的话一样，返回的也是容器。都是二分查找log(n)的时间

### nth\_element(第k小元素)

把第k小元素放到位置k，并且比k小的元素全部放在k左边，比k大的都在右边，但不是都有序的。

复杂度O(n)

下面是例子，数组0~n-1

1. nth\_element(a,a+k-1,a+n);
2. printf("%d\n", a[k-1]);

第k大则加个cmp()函数

1. bool cmp(int x, int y){return x > y;}
2. nth\_element(a,a+k-1,a+n,cmp);
3. printf("%d\n", a[k-1]);

### bitset-超大位数位运算容器

定义：bitset<N> b;(N是位数)，并且两个bitset之间支持各种位运算。

|  |  |
| --- | --- |
| b.any() | b中是否存在置为1的二进制位？ |
| b.none() | b中不存在置为1的二进制位吗？ |
| b.count() | b中置为1的二进制位的个数 |
| b.size() | b中二进制位的个数 |
| b[pos] | 访问b中在pos处的二进制位 |
| b.test(pos) | b中在pos处的二进制位是否为1？ |
| b.set() | 把b中所有二进制位都置为1 |
| b.set(pos) | 把b中在pos处的二进制位置为1 |
| b.reset() | 把b中所有二进制位都置为0 |
| b.reset(pos) | 把b中在pos处的二进制位置为0 |
| b.flip() | 把b中所有二进制位逐位取反 |
| b.flip(pos) | 把b中在pos处的二进制位取反 |
| b.to\_ulong() | 用b中同样的二进制位返回一个unsigned long值 |
| os << b | 把b中的位集输出到os流 |

### 非标准库rope(可持久化平衡树)

头文件：#include <ext/rope>

命名空间：using namespace \_\_gnu\_cxx;

只支持G++；

定义: rope<char> rp;

部分操作(几乎都是O(logn)的)：

push\_back(x)：在末尾添加x (x是char)

insert(pos,x)：在pos插入x (x是字符串,x后面加个int参数可以只能x中插入几个)

erase(pos,x): 从pos开始删除x个

replace(pos,x): 从pos开始换成x(x是字符串,x后面加个int参数可以只能x中的前几个)

substr(pos,x)：提取pos开始x个

copy(x)：复制rope中所有内容到x字符串

at(x)/[x]：访问第x个元素

【实现可持久化(O(1)记录历史版本)】

1. rope<char> \*his[maxn];
2. his[0]=new rope<char>();
3. his[i]=new rope<char>(\*his[i-1]);

就可以实现可持久化了，每次new都是O(1)的，只复制了树根

## VS配置

如果没有代码段管理选项

1、打开vs2005，选择[工具]-[自定义]菜单

2、再选中[命令]选项卡，单击[重排命令(R)...]按钮

3、在弹出的"重排命令"对话框里，菜单栏下拉列表选择"工具"，再单击[添加(A)...]按钮

4、在弹出的"添加命令"对话框中，在"类别"中选择"工具",再在"命令"中找出"代码段管理器..."，单击[确定]返回，再[关闭]，[关闭。

5、这样[工具]菜单中就出现了[代码段管理器(S)...]菜单了.

【工具——代码段管理器——添加init.Snippet代码段文件夹，Snippet (init.Snippet)文件配置】

1. <?xml version="1.0" encoding="utf-8"?>
2. <CodeSnippets xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
3. <CodeSnippet Format="1.0.0">
4. <Header>
5. <Title>inith</Title>
6. <Shortcut>inith</Shortcut>
7. </Header>
8. <Snippet>
9. <Code Language="cpp">
10. <![CDATA[#include <cstdio>
11. #include <cstring>
12. #include <cctype>
13. #include <cmath>
14. #include <set>
15. #include <bitset>
16. //#include <unordered\_set>
17. #include <queue>
18. #include <stack>
19. #include <vector>
20. #include <string>
21. #include <algorithm>
22. using namespace std;
23. typedef long long LL;
24. #define rep(i,a,n) for(int i = a; i < n; i++)
25. #define repe(i,a,n) for(int i = a; i <= n; i++)
26. #define per(i,n,a) for(int i = n; i >= a; i--)
27. #define clc(a,b) memset(a,b,sizeof(a))
28. const int INF = 0x3f3f3f3f;
29. int main()
30. {
31. #ifdef SHY
32. freopen("d:\\1.txt","r",stdin);
33. #endif
34. return 0;
35. }
36. $selected$ $end$]]>
37. </Code>
38. </Snippet>
39. </CodeSnippet>
40. </CodeSnippets>

**<bits/stdc++.h>包含c++所有头文件，目前HDU，HUSTOJ，ZOJ，CF都支持，POJ不支持**