

## 图论

1. 2sat 2. block\_forest\_data\_structure 3. blossom algorithm 4. euler\_tour  
 5. 仙人掌 DP 6. 倍增lca 7. 有向图强联通 tarjan 8. 构造圆方树 9. 点双联通 tarjan  
 10. 边双联通 tarjan 11. 最小树形图：朴素算法 12. 最小树形图：Tarjan 算法

## 二分图匹配

13. hungarian

## 计算几何

14. largest\_empty\_convex 15. minkowski\_sum 16. 最小圆覆盖 17. 向量  
 18. 圆的切线

## 数论

19. Pohlig-Hellman 离散对数 20. Pohlig\_Hellman 21. continued\_fraction  
 22. min\_25\_seive\_lattice\_on\_circle 23. min\_25\_sieve 24. 幂级数前缀和  
 25. 类 Euclid 算法 26. 线性筛 & 杜教筛

## 多项式

27. fft 28. ntt 29. polynomial\_all\_star

## 常识

30. Havel-Hakimi 定理 31. Josephus\_problem

## 网络流

32. dinic 33. 费用流

## 未分类

34. Xorshift 35. 三分\_上凸函数 36. 单纯型 37. 线性空间求交

## 字符串

38. AC 自动机 39. KMP 40. PAM 41. SA 42. manacher  
 43. palindrome\_partition 44. pam 45. 回文自动机 46. 树上后缀数组  
 47. 后缀排序：倍增算法 48. 后缀排序：DC3 49. 后缀排序：SA-IS 50. 后缀树  
 51. fwt 52. lct 53. lichao\_segment\_tree 54. xihe\_tree 55. xihe\_tree\_counting  
 56. 二项堆 57. 左偏树 58. 序列splay 59. 权值splay 60. Link-Cut Tree (splay)  
 61. Link-Cut Tree (treap)

## 其它文档

## 1. 2sat [lmj/2sat.cpp]

## Priest John's Busiest Day

## Description

John is the only priest in his town. September 1st is the John's busiest day in a year because there is an old legend in the town that the couple who get married on that day will be forever blessed by the God of Love. This year  $N$  couples plan to get married on the blessed day. The  $i$ -th couple plan to hold their wedding from time  $S_i$  to time  $T_i$ . According to the traditions in the town, there must be a special ceremony on which the couple stand before the priest and accept blessings. The  $i$ -th couple need  $D_i$  minutes to finish this ceremony. Moreover, this ceremony must be either at the beginning or the ending of the wedding (i.e. it must be either from  $S_i$  to  $S_i + D_i$ , or from  $T_i - D_i$  to  $T_i$ ). Could you tell John how to arrange his schedule so that he can present at every special ceremonies of the weddings.

Note that John can not be present at two weddings simultaneously.

## Input

The first line contains a integer  $N$  ( $1 \leq N \leq 1000$ ).

The next  $N$  lines contain the  $S_i$ ,  $T_i$  and  $D_i$ .  $S_i$  and  $T_i$  are in the format of  $hh:mm$ .

## Output

The first line of output contains "YES" or "NO" indicating whether John can be present at every special ceremony. If it is "YES", output another  $N$  lines describing the starting time and finishing time of all the ceremonies.

## Sample Input

```
2
08:00 09:00 30
08:15 09:00 20
```

## Sample Output

```
YES
08:00 08:30
08:40 09:00
```

## Task Description

- 有  $n$  个 Boolean 变量  $\{x_i\}$ , 以及一个有如下形式的逻辑表达式:  $(x_i \vee x_j) \wedge (\neg x_p \vee x_q) \wedge \dots$ 。(大概就是一堆 or 的 and, 然后每个括号里面有两个变量)

## Solution

- 显然我们可以建一个  $2n$  个点的有向图。对于每个变量  $x_i$  我们开两个变量分别表示  $x_i$  为真和  $x_i$  为假。

对于每组  $(i \vee j)$ , 我们连两条边  $\neg i \implies j$  以及  $\neg j \implies i$  (因为至少要选一个)。

现在对于每个变量  $x$  有四种情况:

- $x$  和  $\neg x$  毫无关系: 爱取啥取啥
- $x \implies \neg x$ : 那么我们取  $x$  为假
- $\neg x \implies x$ : 那么我们取  $x$  为真
- $x \implies \neg x$  并且  $\neg x \implies x$ : 那么无解

注意到只有第四种情况是无解的, 这对应了有向图中  $x$  和  $\neg x$  是在同一个强连通分量中。于是我们可以跑一遍 Tarjan, 判断是否有一个变量的两点在同一个强连通分量中。

以上我们学习了如何判断 2SAT 是否有解，那么如何给出一组可行解呢？

首先我们用 Tarjan 强连通缩点，现在的图变成了一个 DAG。假设我们有任意一组这个 DAG 的拓扑序，如果拓扑序中  $x$  在  $\neg x$  前面，那么我们设  $x$  为假，否则为真。

<https://zhuanlan.zhihu.com/p/50211772> by 后缀自动机 · 张

```

1 const int maxn = 1200;
2 struct node {
3     int v;
4     node *next;
5 } pool[maxn*maxn*4] , *g[maxn*2] , *g1[maxn*2];
6 int top;
7 int n;
8 char z1[200] , z2[200];
9 int start[maxn] , end[maxn] , last[maxn];
10 int low[maxn*2] , dfn[maxn*2] , index;
11 int st[maxn*2] , ins[maxn*2] , tops;
12 int col[maxn*2] , color;
13 int block[maxn*2] , l[maxn*2] , r[maxn*2] , topb;
14 int in[maxn*2] , taboo[maxn*2];
15 int ans[maxn*2];
16 queue < int > q;
17 int get ( int x ) {
18     if ( x % 2 == 0 ) return x - 1;
19     else return x + 1;
20 }
21 void add ( int u , int v ) {
22     node *tmp = &pool[++top];
23     tmp->v = v; tmp->next = g[u]; g[u] = tmp;
24     //printf ( "%d %d\n" , u , v );
25 }
26 void add1 ( int u , int v ) {
27     node *tmp = &pool[++top];
28     tmp->v = v; tmp->next = g1[u]; g1[u] = tmp;
29     //printf ( "%d %d\n" , u , v );
30 }
31 int check ( int l1 , int r1 , int l2 , int r2 ) {
32     //printf ( "%d %d %d %d\n" , l1 , r1 , l2 , r2 );
33     if ( r1 <= l2 || r2 <= l1 ) return 0;
34     return 1;
35 }
36 void tarjan ( int i ) {
37     dfn[i] = low[i] = ++index;
38     ins[i] = 1;
39     st[++tops] = i;
40     for ( node *j = g[i] ; j ; j = j->next ) {
41         if ( !dfn[j->v] ) {
42             tarjan ( j->v );
43             low[i] = min ( low[i] , low[j->v] );
44         }
45         else if ( ins[j->v] ) low[i] = min ( low[i] , dfn[j->v] );
46     }
47     if ( low[i] == dfn[i] ) {
48         color++;
49         l[color] = topb + 1;
50         while ( st[tops] != i ) {
51             col[st[tops]] = color;
52             block[++topb] = st[tops];

```

```

53         ins[st[tops]] = 0;
54         tops--;
55     }
56     col[i] = color;
57     block[++topb] = i;
58     r[color] = topb;
59     ins[i] = 0;
60     tops--;
61 }
62 }
63 void dfs ( int i ) {
64     taboo[i] = 2;
65     for ( node *j = g1[i] ; j ; j = j->next ) {
66         if ( !taboo[j->v] ) dfs ( j->v );
67     }
68 }
69 void work () {
70     int i , k , t1 , t2 , t3 , t4;
71     scanf ( "%d" , &n );
72     for ( i = 1 ; i <= n ; i++ ) {
73         scanf ( "%s%sd" , z1 + 1 , z2 + 1 , &last[i] );
74         start[i] = (z1[1] - '0') * 600 + (z1[2] - '0') * 60 + (z1[4] - '0') * 10 + (z1[5] - '0');
75         end[i] = (z2[1] - '0') * 600 + (z2[2] - '0') * 60 + (z2[4] - '0') * 10 + (z2[5] - '0');
76     }
77     for ( i = 1 ; i <= n ; i++ ) {
78         for ( int j = i + 1 ; j <= n ; j++ ) {
79             t1 = check ( start[i] , start[i] + last[i] , start[j] , start[j] + last[j] );
80             t2 = check ( start[i] , start[i] + last[i] , end[j] - last[j] , end[j] );
81             t3 = check ( end[i] - last[i] , end[i] , start[j] , start[j] + last[j] );
82             t4 = check ( end[i] - last[i] , end[i] , end[j] - last[j] , end[j] );
83             if ( t1 == 1 ) {
84                 add ( i * 2 - 1 , j * 2 );
85                 add ( j * 2 - 1 , i * 2 );
86             }
87             if ( t2 == 1 ) {
88                 add ( i * 2 - 1 , j * 2 - 1 );
89                 add ( j * 2 , i * 2 );
90             }
91             if ( t3 == 1 ) {
92                 add ( i * 2 , j * 2 );
93                 add ( j * 2 - 1 , i * 2 - 1 );
94             }
95             if ( t4 == 1 ) {
96                 add ( i * 2 , j * 2 - 1 );
97                 add ( j * 2 , i * 2 - 1 );
98             }
99         }
100     }
101     for ( i = 1 ; i <= n * 2 ; i++ ) if ( !dfn[i] ) tarjan ( i );
102     //printf ( "%d\n" , tops );
103     //for ( i = 1 ; i <= n * 2 ; i++ ) printf ( "%d %d\n" , dfn[i] , low[i] );
104     //printf ( "%d\n" , color );
105     /*for ( i = 1 ; i <= color ; i++ ) {

```

```

106         for ( int j = l[i] ; j <= r[i] ; j++ ) printf ( "%d " , b
lock[j] );
107         printf ( "\n" );
108     }*/
109     for ( i = 1 ; i <= n ; i++ ) {
110         if ( col[i*2] == col[i*2-1] ) {
111             printf ( "NO\n" );
112             return ;
113         }
114     }
115     for ( i = 1 ; i <= n * 2 ; i++ )
116         for ( node *j = g[i] ; j ; j = j -> next ) {
117             if ( col[i] != col[j->v] ) {
118                 add1 ( col[j->v] , col[i] );
119                 in[col[i]]++;
120             }
121         }
122     for ( i = 1 ; i <= color ; i++ ) if ( in[i] == 0 ) q.push ( i );
123     while ( q.size () ) {
124         //for ( i = 1 ; i <= color ; i++ ) printf ( "%d " , in[i] );
125         //printf ( "\n" );
126         k = q.front ();
127         q.pop ();
128         if ( taboo[k] ) continue;
129         taboo[k] = 1;
130         for ( i = l[k] ; i <= r[k] ; i++ ) {
131             //printf ( "%d %d\n" , k , col[get(block[i])] );
132             dfs ( col[get(block[i])] );
133             ans[block[i]] = 1;
134         }
135         for ( node *j = g1[k] ; j ; j = j -> next ) if ( !taboo[j->v] ) {
136             in[j->v]--;
137             if ( in[j->v] == 0 ) q.push ( j -> v );
138         }
139     }
140     printf ( "YES\n" );
141     for ( i = 1 ; i <= n ; i++ ) {
142         if ( ans[i*2-1] ) {
143             printf ( "%02d:%02d %02d:%02d\n" ,
144                 start[i]/60 , start[i]%60 ,
145                 (start[i]+last[i])/60 , (start[i]+last[i])%60 );
146         }
147         else {
148             printf ( "%02d:%02d %02d:%02d\n" ,
149                 (end[i]-last[i])/60 , (end[i]-last[i])%60 ,
150                 end[i]/60 , end[i]%60 );
151         }
152     }
153 }

```

## 2. block\_forest\_data\_structure [lmj/block\_forest\_data\_structure.cpp]

又叫圆方树

这个代码用来构造仙人掌的圆方树，两个点一条边的双联通分量不会被处理为圆点 + 方点，而是两个圆点直接相连，kind = 0 为圆点。tot 是圆点 + 方点的数量。注意数组大小要开两倍来维护方点。

gt 是造好的圆方树，如果还是从 1 号点开始遍历树的话，那么方点的边表中，就是按照 dfn 顺序的那些点，也就是按照环的顺序排序的，开头是与 1 号点最近的点，可以方便地处理环。

```

1 struct node {
2     int v , u ; node *next;
3 } pooln[maxn*4] , *gn[maxn];
4 struct tree {
5     int v ; tree *next;
6 } poolt[maxn*4] , *gt[maxn*2];
7 int topt , topn;
8 int n , m , tot;
9 int kind[maxn*2] , dfn[maxn] , low[maxn] , index;
10 stack <node*> st;
11 void add ( int u , int v ) {
12     node *tmp = &pooln[++topn];
13     tmp -> v = v ; tmp -> u = u ; tmp -> next = gn[u] ; gn[u] = tmp;
14 }
15 void addt ( int u , int v ) {
16     tree *tmp = &poolt[++topt];
17     tmp -> v = v ; tmp -> next = gt[u] ; gt[u] = tmp;
18 }
19 void tarjan ( int i , int from ) {
20     dfn[i] = low[i] = ++index;
21     for ( node *j = gn[i] ; j ; j = j -> next ) if ( j -> v != from ) {
22         if ( !dfn[j->v] || dfn[i] > dfn[j->v] ) st.push(j);
23         if ( !dfn[j->v] ) {
24             tarjan ( j -> v , i );
25             low[i] = min ( low[i] , low[j->v] );
26             if ( low[j->v] >= dfn[i] ) {
27                 if ( st.top() == j ) {
28                     addt ( i , j -> v , j -> prob );
29                     addt ( j -> v , i , j -> prob );
30                     st.pop();
31                 } else {
32                     tot++;
33                     kind[tot] = 1;
34                     while ( st.top() != j ) {
35                         node *k = st.top ();
36                         st.pop();
37                         addt ( tot , k -> u , k -> prob );
38                         addt ( k -> u , tot , k -> prob );
39                     }
40                     addt ( tot , i , j -> prob );
41                     addt ( i , tot , j -> prob );
42                     st.pop();
43                 }
44             } else low[i] = min ( low[i] , dfn[j->v] );
45         }
46     }
47     void work () {
48         int i , u , v , a , b;
49         scanf ( "%d%d" , &n , &m );
50         for ( i = 1 ; i <= m ; i++ ) {
51             scanf ( "%d%d%d%d" , &u , &v , &a , &b );
52             add ( u , v );
53             add ( v , u );
54         }
55         tot = n;
56         for ( i = 1 ; i <= n ; i++ ) kind[i] = 0;

```

```
56 tarjan ( 1 , -1 );
57 }
```

### 3. blossom algorithm [lmj/blossom\_algorithm.cpp]

```
1 const int maxn = 510;
2 struct node {
3     int v;
4     node *next;
5 } pool[maxn*maxn*2], *g[maxn];
6 int top, n, m, match[maxn];
7 int kind[maxn], pre[maxn], vis[maxn], c[maxn];
8 queue < int > q;
9 int f[maxn], ans;
10 void add ( int u , int v ) {node *tmp = &pool[++top]; tmp -> v = v; tmp ->
    next = g[u]; g[u] = tmp;}
11 int find ( int x ) {int i, t; for ( i = x ; c[i] > 0 ; i = c[i] ) ; while
    ( c[x] > 0 ) {t = c[x]; c[x] = i; x = t;} return i;}
12 void getpath ( int x , int tar , int root ) {
13     int t;
14     while ( x != root ) {t = match[x]; match[tar] = x; match[x] = tar; tar = t;
    x = pre[t];}
15     match[tar] = x; match[x] = tar;
16 }
17 int lca ( int u , int v , int root ) {
18     int i; for ( i = 1 ; i <= n ; i++ ) f[i] = 0;
19     while ( find ( u ) != root ) {u = find ( u ); f[u] = 1; if ( !match[u] )
    break; u = pre[match[u]];}
20     f[root] = 1;
21     while ( find ( v ) != root ) {v = find ( v ); if ( f[v] == 1 ) return v;
    if ( !match[v] ) break; v = pre[match[v]];}
22     return root;
23 }
24 void blossom ( int x , int y , int l ) {
25     while ( find ( x ) != l ) {pre[x] = y; y = match[x]; if ( kind[match[x]]
    == 2 ) {kind[match[x]] = 1; q.push ( match[x] );} if ( find ( x ) == x ) c[
    find(x)] = l; if ( find ( match[x] ) == match[x] ) c[find(match[x])] = l; x
    = pre[y];}
26 }
27 void bfs ( int x ) {
28     int k, i, z;
29     for ( i = 1 ; i <= n ; i++ ) {
30         kind[i] = pre[i] = vis[i] = 0; c[i] = -1;
31     }
32     while ( q.size () ) q.pop (); q.push ( x ); kind[x] = 1; vis[x] = 1;
33     while ( q.size () ) {
34         k = q.front (); q.pop ();
35         for ( node *j = g[k] ; j ; j = j -> next ) {
36             if ( !vis[j->v] ) {
37                 if ( !match[j->v] ) {
38                     getpath ( k , j -> v , x );
39                     return ;
40                 }
41             } else {
42                 kind[j->v] = 2;
43                 kind[match[j->v]] = 1;
44                 pre[j->v] = k;
45                 vis[j->v] = 1; vis[match[j->v]] = 1;
46                 q.push ( match[j->v] );
```

```
47     }
48     else {
49         if ( find ( k ) == find ( j -> v ) ) continue;
50         if ( kind[find(j->v)] == 1 ) {
51             z = lca ( k , j -> v , x );
52             blossom ( k , j -> v , z );
53             blossom ( j -> v , k , z );
54         }
55     }
56 void work () {
57     int i, u, v;
58     scanf ( "%d%d", &n, &m );
59     for ( i = 1 ; i <= m ; i++ ) {
60         scanf ( "%d%d", &u, &v );
61         add ( u , v ); add ( v , u );
62     }
63     for ( i = 1 ; i <= n ; i++ ) {
64         if ( !match[i] ) bfs ( i );
65     }
66     for ( i = 1 ; i <= n ; i++ ) if ( match[i] ) ans++;
67     printf ( "%d\n", ans / 2 );
68     for ( i = 1 ; i <= n ; i++ ) printf ( "%d%c", match[i], i==n?'\\n':' '
    );
69 }
```

### 4. euler\_tour [lmj/euler\_tour.cpp]

```
1 stack < int > s;
2 void dfs ( int i ) {
3     for ( node *j = g[i] ; j ; j = j -> next ) if ( !j -> taboo ) {
4         s.push ( j -> f );
5         j -> taboo = 1;
6         dfs ( j -> v );
7         ans[++index] = s.top ();
8         s.pop ();
9     }
10 }
```

### 5. 仙人掌 DP [xzl/仙人掌 DP, 图论.cpp]

重复使用时，只需清空 dfn、fa 和 now。每次扫出的环按一定顺序存放在 a 数组中，a[1] 是环的根。

```
1 int dfn[NMAX + 10], low[NMAX + 10], now, cnt;
2 int ed[NMAX + 10], fa[NMAX + 10], a[NMAX + 10];
3 void dfs(int x) {
4     dfn[x] = low[x] = ++now;
5     for (int v : G[x]) if (v != fa[x]) {
6         if (dfn[v]) {
7             ed[v] = x, low[x] = min(low[x], dfn[v]);
8             continue;
9         } fa[v] = x;
10        dfs(v);
11        if (low[v] > dfn[x]) ; // 割边
12        else if (low[v] == dfn[x]) {
13            a[1] = x;
14            for (cnt = 1, v = ed[x]; v != x; v = fa[v])
15                a[++cnt] = v;
16            // 环 a[1]...a[cnt]
17        } else low[x] = min(low[x], low[v]);
18    }
```

## 6. 倍增lca [sll/lca.cpp]

```

1 int lca(int x, int y) {
2     if (deep[x] < deep[y]) swap(x, y);
3     int t = deep[x] - deep[y];
4     for (int i = 0; bin[i] <= t; i++)
5         if (t & bin[i]) x = fa[x][i];
6     for (int i = 16; i >= 0; i--)
7         if (fa[x][i] != fa[y][i])
8             x = fa[x][i], y = fa[y][i];
9     if (x == y) return x;
10    return fa[x][0];
11 }

```

## 7. 有向图强联通 tarjan [sll/tarjan(SCC).cpp]

```

1 int n, m;
2 int head[N], pos;
3 struct edge { int to, next; } e[N < 1];
4 void add(int a, int b)
5 { pos++; e[pos].to = b, e[pos].next = head[a], head[a] = pos; }
6 int dfn[N], low[N], SCC;
7 bool in[N];
8 int st[N], top, T;
9 vector<int> G[N];
10 void tarjan(int u) {
11     st[++top] = u; in[u] = 1;
12     dfn[u] = low[u] = ++T;
13     for (int i = head[u]; i; i = e[i].next) {
14         int v = e[i].to;
15         if (!dfn[v]) {
16             tarjan(v);
17             low[u] = min(low[u], low[v]);
18         }
19         else if (in[v]) low[u] = min(low[u], dfn[v]);
20     }
21     if (low[u] == dfn[u]) {
22         int v;
23         ++SCC;
24         do {
25             v = st[top--];
26             in[v] = false;
27             G[SCC].push_back(v);
28         } while (v != u);
29     }
30 }
31 int main() {
32     scanf("%d%d", &n, &m);
33     for (int i = 1; i <= m; i++) {
34         int x, y;
35         scanf("%d%d", &x, &y);
36         add(x, y);
37     }
38     for (int i = 1; i <= n; i++) if (!dfn[i]) tarjan(i);

```

## 8. 构造圆方树 [xzl/biconnected.cpp]

G 用于存图，T 是构造的圆方树。只有一个点的点双没有添加方点。

```
1 static vector<int> G[NMAX + 10], T[NMAX + 10];
```

```

2 void bcc(int u, int f = 0) {
3     static stack<Pair> stk;
4     static bool marked[NMAX + 10];
5     static int in[NMAX + 10], low[NMAX + 10], cur;
6     in[u] = low[u] = ++cur;
7     for (int v : G[u]) {
8         if (v == f) f = 0; // 应对重边
9         else if (in[v]) low[u] = min(low[u], in[v]);
10        else {
11            stk.push(Pair(u, v)); // stk 内存储 DFS 树上的边
12            bcc(v, u);
13            low[u] = min(low[u], low[v]);
14            if (low[v] > in[u]) { // 割边 u - v
15                T[u].push_back(v);
16                T[v].push_back(u);
17                stk.pop();
18            } else if (low[v] >= in[u]) { // 可能有点双了
19                cnt++;
20                int linked = 0, p = n + cnt; // linked 点数, p 圆方树上的新方点
21                auto add = [p, &linked](int x) {
22                    if (!marked[x]) {
23                        marked[x] = true;
24                        T[p].push_back(x);
25                        T[x].push_back(p);
26                        linked++;
27                    }
28                };
29                while (!stk.empty()) {
30                    Pair x = stk.top();
31                    stk.pop();
32                    add(x.u);
33                    add(x.v);
34                    if (x.u == u && x.v == v) break;
35                }
36                for (int v : T[p]) marked[v] = false;
37                if (linked == 0) cnt--; // 假点双
38            }
39        }
40    }
41 }

```

## 9. 点双联通 tarjan [sll/点双连通分量.cpp]

```

1 void tarjan(int u) {
2     dfn[u] = low[u] = ++tim;
3     sz[u] = 1;
4     ll t = 0;
5     for (ll i = head[u]; i; i = e[i].next) {
6         ll v = e[i].to;
7         if (dfn[v]) low[u] = min(low[u], dfn[v]);
8         else {
9             tarjan(v);
10            sz[u] += sz[v];
11            low[u] = min(low[u], low[v]);
12            if (dfn[u] <= low[v]) {
13                ans[u] += t * sz[v];
14                t += sz[v];
15            }
16        }
17     }
18     ans[u] += t * (n - t - 1);

```

## 10. 边双联通 tarjan [sll/边双连通分量.cpp]

```
1 void tarjan(int x, int f) {
```

```

2  dfn[x] = low[x] = ++tot;
3  sta[++tp] = x; vis[x] = true;
4  for (int i = head[x]; i; i = e[i].next) if (i != (f ^ 1)) {
5      if (!dfn[e[i].to]) {
6          tarjan(e[i].to, i);
7          low[x] = min(low[x], low[e[i].to]);
8      } else if (vis[e[i].to])
9          low[x] = min(low[x], dfn[e[i].to]);
10 }
11 if (dfn[x] == low[x]) {
12     scc++;
13     while(1) {
14         int pos = sta[tp--];
15         vis[pos] = false;
16         belong[pos] = scc;
17         if (pos == x) break;
18     }}

```

### 11. 最小树形图：朴素算法 [xzl/mdst-nm.cpp]

给定一张  $n$  个点  $m$  条边的带权有向图，求以  $r$  为根的最小树形图上的边权总和，如果不存在输出 -1。时间复杂度为  $O(nm)$ 。调用 `mdst(r)` 获得答案，调用前需清空 `id` 数组。如要求不定根的最小树形图，可以额外添加一个节点，向原图中的每个点连接一条边权为  $\infty$  的边。

```

1 static int n, m, G[NMAX + 10], nxt[MMAX + 10];
2 static struct Edge { int u, v, w; } E[MMAX + 10], *in[NMAX + 10];
3 static int id[NMAX + 10], mark[NMAX + 10];
4 int find(int x) { return id[x] ? id[x] = find(id[x]) : x; }
5 int dfs(int x) {
6     mark[x] = 1; int ret = 1;
7     for (int i = G[x]; i; i = nxt[i])
8         if (!mark[E[i].v]) ret += dfs(E[i].v);
9     return ret;
10 }
11 inline int detect(int x) {
12     mark[x] = x;
13     for (int y = in[x]->u; in[y]; y = in[y]->u)
14         if (mark[y]) return mark[y] == x ? y : 0;
15     else mark[y] = x;
16     return 0;
17 }
18 int mdst(int r) {
19     if (dfs(r) < n) return -1;
20     int ret = 0;
21     while (true) {
22         memset(in, 0, sizeof(in));
23         memset(mark, 0, sizeof(mark));
24         for (auto *e = E + 1; e <= E + m; e++)
25             if (e->u != e->v && e->v != r && (!in[e->v] || e->w < in[e->v]->w))
26                 in[e->v] = e;
27         int p = 0, t = 0;
28         for (int x = 1; x <= n; x++, t |= p) if (!mark[x] && in[x]) {
29             if (!(p = detect(x))) continue;
30             ret += in[p]->w;
31             for (int x = in[p]->u; x != p; x = in[x]->u)
32                 id[find(x)] = p, ret += in[x]->w;
33             for (auto *e = E + 1; e <= E + m; e++) {
34                 int u = find(e->u), v = find(e->v);

```

```

35         if (u != p && v == p) e->w -= in[e->v]->w;
36         e->u = u; e->v = v;
37     }}
38     if (!t) break;
39 }
40 for (int x = 1; x <= n; x++) if (in[x]) ret += in[x]->w;
41 return ret;
42 }

```

### 12. 最小树形图：Tarjan 算法 [xzl/最小树形图：Tarjan 算法, 图论.cpp]

使用可并堆优化的 Chu-Liu 算法，这里使用左偏树。in 存储原图的入边。contract 会生成一棵 contraction 树，树根为  $n$ 。Contraction 树上每个节点的所有儿子构成一个环，环上每个点的入边存放在 `ed` 内。使用 `expand(r, n)` 从节点  $r$  处展开以  $r$  为根的最小树形图，如果返回 INF 则表示不存在树形图。contract 过程会增加节点并且改动边权，故使用 `w0` 保存原始边权。注意点数  $n$  应该开到两倍。重复使用时注意收缩完后 `fa[n]` 和 `nxt[n]` 置零。contract 时间复杂度为  $O(m \log n)$ ，expand 时间复杂度为  $\Theta(n)$ ，实测随机数据下只有边数  $m$  达到  $5 \times 10^5$  级别时才比朴素算法快。

```

1 #define INF 0x3f3f3f3f
2 struct Edge { int u, v, w, w0; };
3 struct Heap {
4     Heap(Edge *_e) : e(_e), rk(1), sum(0), lch(NULL), rch(NULL) {}
5     Edge *_e; int rk, sum;
6     Heap *_lch, *_rch;
7     void push() {
8         if (lch) lch->sum += sum;
9         if (rch) rch->sum += sum;
10        e->w += sum; sum = 0;
11    };
12    inline Heap *_meld(Heap *_x, Heap *_y) {
13        if (!x) return y;
14        if (!y) return x;
15        if (x->e->w + x->sum > y->e->w + y->sum)
16            swap(x, y);
17        x->push();
18        x->rch = meld(x->rch, y);
19        if (!x->lch || x->lch->rk < x->rch->rk)
20            swap(x->lch, x->rch);
21        x->rk = x->rch ? x->rch->rk + 1 : 1;
22        return x;
23    }
24    inline Edge *_extract(Heap *_x) {
25        Edge *_r = x->e;
26        x->push();
27        x = meld(x->lch, x->rch);
28        return r;
29    }
30    static vector<Edge> in[NMAX + 10];
31    static int n, m, fa[2 * NMAX + 10], nxt[2 * NMAX + 10];
32    static Edge *_ed[2 * NMAX + 10];
33    static Heap *_Q[2 * NMAX + 10];
34    static UnionFind id; // id[] & id.fa
35    void contract() {
36        static bool mark[2 * NMAX + 10];
37        //memset(mark + 1, 0, 2 * n);

```

```

38 //id.clear(2 * n);
39 for (int i = 1; i <= n; i++) {
40     queue<Heap*> q;
41     for (int j = 0; j < in[i].size(); j++)
42         q.push(new Heap(&in[i][j]));
43     while (q.size() > 1) {
44         Heap *u = q.front(); q.pop();
45         Heap *v = q.front(); q.pop();
46         q.push(meld(u, v));
47     } Q[i] = q.front();
48 } mark[1] = true;
49 for (int u = 1, u0 = 1, p; Q[u]; mark[u0 = u] = true) {
50     do u = id[ed[u] = extract(Q[u]) -> u];
51     while (u == u0 && Q[u]);
52     if (u == u0) break;
53     if (!mark[u]) continue;
54     for (u0 = u, n++; u != n; u = p) {
55         id.fa[u] = fa[u] = n;
56         if (Q[u]) Q[u] -> sum -= ed[u] -> w;
57         Q[n] = meld(Q[n], Q[u]);
58         p = id[ed[u] -> u];
59         nxt[p == n ? u0 : p] = u;
60     }}
61 i64 expand(int, int);
62 i64 _expand(int x) {
63     i64 r = 0;
64     for (int u = nxt[x]; u != x; u = nxt[u])
65         if (ed[u] -> w0 >= INF) return INF;
66     else r += expand(ed[u] -> v, u) + ed[u] -> w0;
67     return r;
68 }
69 i64 expand(int x, int t) {
70     i64 r = 0;
71     for (; x != t; x = fa[x])
72         if ((r += _expand(x)) >= INF) return INF;
73     return r;
74 }
75 //contract();
76 //i64 ans = expand(rt, n);

```

### 13. hungarian [lmj/hungarian.cpp]

求一定/可能出现在匹配中的边/点

#### 3.2一些拓展

任意图中,设 $M, M'$ 为两个不同的最大匹配,令 $D$ 为 $M, M'$ 的对称差

则 $D$ 由若干交替环、交替链组成,且由于 $M, M'$ 都是最大匹配,环、链长度都为偶数(长为奇数的交替链代表有增广路)。

这说明,任意两个最大匹配,一个都可以通过异或上若干不相交的偶链、偶环得到另一个。

特别地,两个完备匹配的对称差只有若干不相交的交替环。因为某个匹配异或长度为偶数的交替链后,链的某端点将成为未盖点,说明不是完备匹配。

#### 3.2.1二分图最大匹配关键点

关键点是指的一定在最大匹配中的点

由于二分图左右两侧是对称的,我们只考虑找左侧的关键点。

先求任意一个最大匹配 $M$ ,要找的关键点此时一定都是匹配点。考虑 $M$ 中的一个匹配点 $p$ ,设 $M'$ 为某个不包含 $p$ 的最大匹配,对称差 $D=M\oplus M'$ ,则 $D$ 中一定存在一条以 $p$ 为端点的偶交替链,这条链另一端不在 $M$ 中。

那么一个匹配点 $s$ 能变成非匹配点,当且仅当从这个点出发能找一条以匹配边出发的交替链,使得终点是某个未盖点 $t$ 。由于链长为偶数, $t$ 和 $s$ 属于同一侧(左侧)。

我们倒过来考虑,先给二分图定向:匹配边从右到左、非匹配边从左到右,从左侧每个未盖点出发DFS,给到达那些点打上标记。最终左侧每个没有标记的匹配点即为关键点。因为只关心可达性,显然每个点只需访问至多一次,复杂度 $O(n+m)$

#### 3.2.2二分图最大匹配关键边

我们需要找到哪些边一定在最大匹配中

同样地,先求任意一个最大匹配 $M$ 。

关键边一定是匹配 $M$ 中的边,对于一条边 $e(e\in M)$ ,假设存在另一个不包含 $e$ 的最大匹配 $M'$ 。

$M, M'$ 的对称差中, $e$ 一定存在,则要么属于一个偶交替链、要么属于一个偶交替环。

对于偶交替链的情况,链的某一端一定是未盖点,说明 $e$ 的某个端点能通过交替路走到未盖点,即某个端点是非关键点。因此只需判定 $e$ 两端是否存在非关键点。

对于偶交替环的情况,我们给二分图定向:匹配边从左到右、非匹配边从右到左,再检测 $e$ 是否在某个环中(因为不存在奇环)。用 Tarjan 算法求强连通分量即可。

考虑复杂度,需要先 $O(n+m)$ 求关键点,再 $O(n+m)$ 用 Tarjan 算法求强连通分量。

#### 2015集训队论文陈胤伯

可能出现的点: 如果一个点 $u$  (在左边) 不在匹配里但是指向一个以匹配的点 $v$ , 可以选择这条边, 然后原来的 $v$ 的匹配边删除即可。

代码: 二分图点编号为左边 $1, 3, 5, \dots$ , 右边 $2, 4, 6, \dots$ , 输入为左边第 $u$ 个点 to 右边第 $v$ 个点连边,  $key$ 数组存每个点是不是一定在最大匹配里。 $1$ 表示是关键点

```

1 const int maxn = 120000;
2 struct node {
3     int v, f;
4     node *next, *rev;
5 } pool[maxn*10], *g[maxn*2];
6 int top;
7 int n, m;
8 int S, T;
9 int match[maxn*2];
10 int level[maxn*2];
11 int key[maxn*2], vis[maxn*2];
12 int ans[maxn], cnt;
13 void add ( int u, int v ) {
14     node *tmp1 = &pool[++top], *tmp2 = &pool[++top];
15     tmp1 -> v = v; tmp1 -> f = 1; tmp1 -> next = g[u]; g[u] = tmp1; tmp1 ->
    rev = tmp2;
16     tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> next = g[v]; g[v] = tmp2; tmp2 ->
    rev = tmp1;
17 }
18 bool makelevel () {
19     int i, k;
20     queue < int > q;
21     for ( i = S ; i <= T ; i++ ) {
22         level[i] = -1;

```

```

23 }
24 level[S] = 0;
25 q.push ( S );
26 while ( q.size () ) {
27     k = q.front (); q.pop ();
28     for ( node *j = g[k] ; j ; j = j -> next )
29         if ( j -> f && level[j->v] == -1 ) {
30             level[j->v] = level[k] + 1;
31             q.push ( j -> v );
32             if ( j -> v == T ) return true;
33         }
34 }
35 return false;
36 }
37 int find ( int k , int key ) {
38     if ( k == T ) return key;
39     int i , s = 0;
40     for ( node *j = g[k] ; j ; j = j -> next ) {
41         if ( j -> f && level[j->v] == level[k] + 1 && s < key ) {
42             i = find ( j -> v , min ( key - s , j -> f ) );
43             j -> f -= i;
44             j -> rev -> f += i;
45             s += i;
46         }
47     }
48     if ( s == 0 ) level[k] = -1;
49     return s;
50 }
51 int dinic () {
52     int flow = 0;
53     while ( makelevel () == true ) flow += find ( S , 9999999 );
54     return n - 1 - flow;
55 }
56 void dfs ( int i ) {
57     key[i] = 0;
58     vis[i] = 1;
59     for ( node *j = g[i] ; j ; j = j -> next ) if ( j -> v != S && j -> v != T ) {
60         if ( j -> v != match[i] ) {
61             if ( vis[match[j->v]] ) continue;
62             dfs ( match[j->v] );
63         }
64     }
65 }
66 void work () {
67     int i , u , v;
68     scanf ( "%d%d" , &n , &m );
69     S = 0; T = 2 * n + 1;
70     for ( i = 1 ; i <= n ; i++ ) {
71         add ( S , 2 * i - 1 );
72         add ( i * 2 , T );
73     }
74     for ( i = 1 ; i <= m ; i++ ) {
75         scanf ( "%d%d" , &u , &v );
76         add ( 2 * u - 1 , 2 * v );
77     }
78     int out = dinic ();
79     printf ( "%d\n" , out );

```

```

80 for ( i = 1 ; i <= n * 2 ; i += 2 ) {
81     for ( node *j = g[i] ; j ; j = j -> next ) {
82         if ( j -> v >= 1 && j -> f == 0 ) {
83             match[i] = j -> v;
84             match[j->v] = i;
85         }
86     }
87 }
88 for ( i = 1 ; i <= 2 * n ; i++ ) key[i] = 1;
89 for ( i = 1 ; i <= n * 2 ; i++ ) {
90     if ( !match[i] && key[i] == 1 ) {
91         dfs ( i );
92     }
93 }
94 //for ( i = 1 ; i <= n * 2 ; i++ ) {
95 //    printf ( "%d\n" , key[i] );
96 //}
97 }

```

#### 14. largest\_empty\_convex [lmj/largest\_empty\_convex.cpp]

##### 算法一

先枚举每个点作为凸包的最下面的点O，不考虑它下方的点，其余点以O为原点极角排序。

再枚举凸包最后一个点i与O相连，设dp[i][j]表示组成凸包的最后一个三角形是Oij的最大凸包面积，那么容易得到dp方程：dp[i][j]=max(SΔOij+dp[j][k]),(ΔOij内无点且边Oi上无点且k在ij→右侧保证凸性)；

注意到如果边Oi上有点，它就只能作为凸包的边界，所以不能更新dp[i][j]，但还是要用max(SΔOij+dp[j][k])更新最终答案，否则用dp[i][j]更新答案。但这样单次dp的复杂度是O(n<sup>3</sup>)的，考虑优化这个过程，所以有了算法二。

##### 算法二

还是枚举凸包最后一个点i，dp意义也相同。可以发现，合法的j可用以下方法得到：1.令j1=i-1,可以保证ΔOij合法；如果j1在Oi上，那么从大到小枚举第一个不在Oi上的点作为j1，但这样Oi上就有点了，处理方法同上。

2.令j2为最大的在j1→右侧的点，j3为最大的在j2→右侧的点.....可以证明这样可以找出所有合法的j。

3.注意到找出的j序列是满足凸性的，且dp[i][jn]的第一个可以转移来的点就是jn+1,所以可以用g[i][j]表示max(dp[i][k],1≤k≤j)，那么就有dp[i][jn]=SΔOij+g[i][jn+1]注：1-j中不合法的点dp值为0，不影响g的更新，所以g数组一定满足凸性；但如果Oi上有点还是不更新dp和g，但还是要更新最终答案。

这样枚举原点是O(n)的，枚举凸包最后一点i是O(n)的，枚举合法的j是O(n)的，转移是O(1)的，更新g数组是O(n)的，所以总复杂度是O(n<sup>3</sup>)的。

代码里求的是面积

<https://blog.csdn.net/cdsszjj/article/details/79366813>

```

1 const int N=105;
2 struct point
3 {
4     double x,y;
5     point(){}

```



```

6   point(double _x,double _y):x(_x),y(_y){}
7   inline friend point operator - (const point &a,const point &b)
8   {return point(a.x-b.x,a.y-b.y);}
9   inline friend double operator * (const point &a,const point &b)
10  {return a.x*b.y-a.y*b.x;}
11  inline double dis(){return x*x+y*y;}
12 }a[N],p[N],0;
13 int T,n,m;
14 double dp[N][N],ans;
15
16 inline bool cmp(const point &a,const point &b)
17 {
18     double res=(a-0)*(b-0);
19     if(res)return res>0;
20     return (a-0).dis()<(b-0).dis();
21 }
22
23 void solve()
24 {
25     memset(dp,0,sizeof(dp));
26     sort(p+1,p+m+1,cmp);
27     for(int i=1;i<=m;i++)
28     {
29         int j=i-1;
30         while(j&&!((p[i]-0)*(p[j]-0)))j--;
31         bool bz=(j==i-1);
32         while(j)
33         {
34             int k=j-1;
35             while(k&&(p[i]-p[k])*(p[j]-p[k])>0)k--;
36             double area=fabs((p[i]-0)*(p[j]-0))/2;
37             if(k)area+=dp[j][k];
38             if(bz)dp[i][j]=area;
39             ans=max(ans,area),j=k;
40         }
41         if(bz)for(int j=1;j<i;j++)dp[i][j]=max(dp[i][j],dp[i][j-1]);
42     }
43 }
44
45 int main()
46 {
47     //freopen("lx.in","r",stdin);
48     T=getint();
49     while(T--)
50     {
51         n=getint();ans=0;
52         for(int i=1;i<=n;i++)a[i].x=getint(),a[i].y=getint();
53         for(int i=1;i<=n;i++)
54         {
55             0=a[i],m=0;
56             for(int j=1;j<=n;j++)
57             if(a[j].y>a[i].y||a[j].y==a[i].y&&a[j].x>a[i].x)p[++m]=a[j];
58             solve();
59         }
60         printf("%.11f\n",ans);
61     }
62     return 0;

```


63 }

## 15. minkowski\_sum [lmj/minkowski\_sum.cpp]

求  $C = \{x + y | x \in A, y \in B\}$

如果B只有一个点，相当于A向某个方向平移，然后我们现在想要求出C的凸包，首先原来不在A的凸包上的点平移之后也肯定不在凸包上，所以我们可以先求出A的凸包和B的凸包。

考虑合并两个凸壳的情况

 minkowski\_sum

对于图中每个点，如果它是i和j加起来得到的话，我们就把它标号为(i,j)，然后把这张图变成一个表格。然后我们发现，凸包中的点构成一个从(1,1)到(|A|,|B|)的路径，而且只能往上或往右走，那么我们就可以用双指针来维护了，设当前在(i,j)，每次看一下(i+1,j)和(i,j+1)两个点哪个在新的凸包上，然后走过去

为了避免我们求出的凸包是个有三点共线的假凸包，再对C跑一个凸包就好了

<https://www.cnblogs.com/bztMinamoto/p/10511031.html>

```

1  typedef long long LL;
2  struct node {
3      LL x , y;
4      node () : x(0),y(0){}
5      node ( LL xx , LL yy ) {x = xx; y = yy;}
6  } p[4][220000] , tt[220000];
7  LL n[4];
8  LL ans;
9  node operator + ( node x1 , node x2 ) {
10     x1.x += x2.x; x1.y += x2.y;
11     return x1;
12 }
13 node operator - ( node x1 , node x2 ) {
14     x1.x -= x2.x; x1.y -= x2.y;
15     return x1;
16 }
17 bool cmp ( node x1 , node x2 ) {
18     if ( x1.x == x2.x ) return x1.y < x2.y;
19     return x1.x < x2.x;
20 }
21 LL det ( node x1 , node x2 , node x3 ) {
22     x3 = x3 - x1; x2 = x2 - x1;
23     return x2.x * x3.y - x2.y * x3.x;
24 }
25 void convex ( node *x , LL &num ) {
26     LL i , cnt , old;
27     sort ( x + 1 , x + 1 + num , cmp );
28     tt[cnt=1] = x[1];
29     for ( i = 2 ; i <= num ; i++ ) {
30         while ( cnt >= 2 && det ( tt[cnt-1] , tt[cnt] , x[i] ) >= 0 ) cnt--;
31         tt[++cnt] = x[i];
32     }
33     old = cnt;
34     for ( i = num - 1 ; i >= 1 ; i-- ) {
35         while ( cnt >= old + 1 && det ( tt[cnt-1] , tt[cnt] , x[i] ) >= 0 ) cnt--;
36         tt[++cnt] = x[i];
37     }
38     if ( cnt != 1 ) cnt--;

```

```

39 for ( i = 1 ; i <= cnt ; i++ )
40   x[i] = tt[i];
41 num = cnt;
42 //the first point doesn't repeat in the end
43 }
44 void merge ( node *y , node *x1 , node *x2 , LL num1 , LL num2 , LL &num3
45 ) {
46   LL i , j;
47   y[num3+1] = x1[1] + x2[1];
48   x1[num1+1] = x1[1]; x2[num2+1] = x2[1];
49   for ( i = 1 , j = 1 ; i <= num1 && j <= num2 ; ) {
50     if ( det (node(0,0),x1[i+1]-x1[i],x2[j+1]-x2[j]) > 0 )
51       y[++num3] = x1[i] + x2[++j];
52     else y[++num3] = x1[++i] + x2[j];
53   }
54   for ( ; i <= num1 ; i++ ) y[++num3] = x1[i] + x2[j];
55   for ( ; j <= num2 ; j++ ) y[++num3] = x1[i] + x2[j];
56 }
57 void work () {
58   LL i;
59   scanf ( "%lld%lld" , &n[1] , &n[2] );
60   for ( i = 1 ; i <= n[1] ; i++ )
61     scanf ( "%lld%lld" , &p[1][i].x , &p[1][i].y );
62   for ( i = 1 ; i <= n[2] ; i++ )
63     scanf ( "%lld%lld" , &p[2][i].x , &p[2][i].y );
64   convex ( p[1] , n[1] );
65   convex ( p[2] , n[2] );
66   merge ( p[3] , p[1] , p[2] , n[1] , n[2] , n[3] );
67   convex ( p[3] , n[3] );
68   p[3][n[3]+1] = p[3][1];
69   for ( i = 1 ; i <= n[3] ; i++ ) {
70     ans += det ( node(0,0) , p[3][i+1] , p[3][i] );
71   } //calculating 2*area
72   printf ( "%lld\n" , ans );
73 }

```

## 16. 最小圆覆盖 [lmj/minimal\_circle\_cover.cpp]

```

1 const int maxn = 120000;
2 struct point {
3   double x , y;
4 } a[maxn] , c , tmp1 , tmp2;
5 int n;
6 double r;
7 double tmp;
8 double dis ( point x1 , point x2 ) {return sqrt ( (x1.x-x2.x)*(x1.x-x2.x)
9 + (x1.y-x2.y)*(x1.y-x2.y) );}
10 double det ( point x1 , point x2 , point x3 ) {return (x2.x-x1.x) * (x3.y
11 -x1.y) - (x3.x-x1.x) * (x2.y-x1.y);}
12 double abs ( double x ) {if ( x < 0 ) return -x;return x;}
13 point getcen ( point x1 , point x2 , point x3 ) {
14   double A , B , C , D , E , F;point ret;
15   if ( x1.x == x2.x ) A = 0.0, B = 1.0, C = (x1.y+x2.y)/2.0;
16   else {
17     A = 1.0/((x1.y-x2.y) / (x1.x-x2.x));B = 1.0;
18     C = -(x1.y+x2.y)/2.0 - A * (x1.x+x2.x)/2.0;
19   }
20   if ( x1.x == x3.x ) D = 0.0, E = 1.0, F = (x1.y+x3.y)/2.0;
21   else {

```

```

22   D = 1.0/((x1.y-x3.y) / (x1.x-x3.x));E = 1.0;
23   F = -(x1.y+x3.y)/2.0 - D * (x1.x+x3.x)/2.0;
24 }
25 ret.x = (B * F - C * E) / (A * E - B * D);
26 ret.y = (A * F - C * D) / (B * D - A * E);
27 return ret;
28 }
29 void work () {
30   int i , j , k;
31   srand(67890);
32   scanf ( "%d" , &n );
33   for ( i = 1 ; i <= n ; i++ ) scanf ( "%lf%lf" , &a[i].x , &a[i].y );
34   random_shuffle ( a + 1 , a + 1 + n );
35   if ( n == 2 ) {
36     printf ( "%.3lf\n" , dis ( a[1] , a[2] ) / 2.0 );
37     return ;
38   }
39   c.x = a[1].x;c.y = a[1].y;r = 0.0;
40   for ( i = 2 ; i <= n ; i++ ) {
41     if ( dis ( c , a[i] ) - r > 1e-9 ) {
42       c.x = a[i].x;c.y = a[i].y;r = 0.0;
43       for ( j = 1 ; j < i ; j++ ) {
44         if ( dis ( c , a[j] ) - r > 1e-9 ) {
45           c.x = (a[i].x + a[j].x) / 2.0;
46           c.y = (a[i].y + a[j].y) / 2.0;
47           r = dis ( a[i] , a[j] ) / 2.0;
48           tmp = r; tmp1 = c;
49           for ( k = 1 ; k <= j - 1 ; k++ ) {
50             if ( dis ( tmp1 , a[k] ) - tmp > 1e-9 ) {
51               if ( abs(det ( a[i] , a[j] , a[k] )) < 1e-9 ) continue;
52               tmp2 = getcen ( a[i] , a[j] , a[k] );
53               tmp = dis ( tmp2 , a[i] );
54               tmp1 = tmp2;
55             }
56           }
57           c = tmp1; r = tmp;
58         }
59       }
60     }
61   }
62   printf ( "%.3lf\n" , r );
63 }

```

## 17. 向量 [xzl/vector.cpp]

```

1 typedef double ld;
2 #define EPS 1e-8
3 inline bool eq(ld x, ld y) { return x - EPS < y && y < x + EPS; }
4 inline ld sqrt_s(ld x) { return sqrt(max(0.0, x)); }
5 struct vec {
6   vec() : x(0), y(0) {}
7   vec(ld _x, ld _y) : x(_x), y(_y) {}
8   ld x, y;
9   ld len() const { return hypot(x, y); }
10  ld len2() const { return x * x + y * y; }
11  vec norm() const { ld l = len(); return vec(x / l, y / l); }
12  vec cw() const { return vec(y, -x); }
13  vec cw(ld t) const { ld c = cos(t), s = sin(t); return vec(-c * x + s *
14 y, -s * x - c * y); }
15  vec ccw() const { return vec(-y, x); }
16  vec ccw(ld t) const { ld c = cos(t), s = sin(t); return vec(c * x - s *
17 y, s * x + c * y); }
18  vec operator+(const vec &z) const { return vec(x + z.x, y + z.y); }

```

```

17 vec operator~(const vec &z) const { return vec(x - z.x, y - z.y); }
18 vec operator-() const { return vec(-x, -y); }
19 friend vec operator*(ld k, const vec &z);
20 vec operator*(ld k) const { return vec(x * k, y * k); }
21 vec operator/(ld k) const { return vec(x / k, y / k); }
22 vec &operator+=(const vec &z) { x += z.x; y += z.y; return *this; }
23 vec &operator-=(const vec &z) { x -= z.x; y -= z.y; return *this; }
24 vec &operator*=(ld k) { x *= k; y *= k; return *this; }
25 vec &operator/=(ld k) { x /= k; y /= k; return *this; }
26 bool operator==(const vec &z) const {
27     return x - EPS < z.x && z.x < x + EPS &&
28         y - EPS < z.y && z.y < y + EPS;
29 }
30 bool operator!=(const vec &z) const {
31     return x - EPS >= z.x || z.x >= x + EPS ||
32         y - EPS >= z.y || z.y >= y + EPS;
33 };
34 inline vec operator*(ld k, const vec &z) { return vec(z.x * k, z.y * k); }
35 inline ld dot(const vec &u, const vec &v) { return u.x * v.x + u.y * v.y; }
36 inline ld cross(const vec &u, const vec &v) { return u.x * v.y - u.y * v.x; }

```

## 18. 圆的切线 [xzl/circle-tangents.cpp]

注意需要保证切线存在算法才能正常运作。并且注意使用 cmath 中的函数的时候防止定义域溢出导致的 nan 问题。

```

1 struct seg {
2     vec u, v;
3     ld len() const { return (u - v).len(); }
4 };
5 struct cir {
6     vec p; ld r;
7 };
8 // 点与圆的切点
9 inline void ptan(const vec &p, const cir &c, vec &t1, vec &t2) {
10     vec v = p - c.p;
11     ld d = v.len(), l = sqrt_s(d * d - c.r * c.r);
12     ld h = c.r * l / d, s = c.r * c.r / d;
13     v /= d; vec u = c.p + v * s; v = v.cw() * h;
14     t1 = u + v; t2 = u - v;
15 }
16 // 外公切线
17 inline void c2tan1(const cir &c1, const cir &c2, seg &t1, seg &t2) {
18     vec v = c1.p - c2.p;
19     ld dr = abs(c1.r - c2.r), d = v.len();
20     ld l = sqrt_s(d * d - dr * dr);
21     ld h1 = l * c1.r / d, s1 = dr * c1.r / d;
22     ld h2 = l * c2.r / d, s2 = dr * c2.r / d;
23     v = (c1.r > c2.r ? -v : v) / d;
24     vec u = v.cw(), p1 = c1.p + v * s1, p2 = c2.p + v * s2;
25     t1 = seg(p1 + u * h1, p2 + u * h2);
26     t2 = seg(p1 - u * h1, p2 - u * h2);
27 }
28 // 内公切线
29 inline void c2tan2(const cir &c1, const cir &c2, seg &t1, seg &t2) {
30     vec v = c1.p - c2.p;

```

```

31     ld d = v.len();
32     ld d1 = d * c1.r / (c1.r + c2.r), d2 = d * c2.r / (c1.r + c2.r);
33     ld l1 = sqrt_s(d1 * d1 - c1.r * c1.r), l2 = sqrt_s(d2 * d2 - c2.r * c2.r);
34     ld h1 = c1.r * l1 / d1, h2 = c2.r * l2 / d2;
35     ld s1 = c1.r * c1.r / d1, s2 = c2.r * c2.r / d2;
36     v /= d; vec u = v.cw();
37     vec p1 = c1.p - v * s1, p2 = c2.p + v * s2;
38     t1 = seg(p1 + u * h1, p2 - u * h2);
39     t2 = seg(p1 - u * h1, p2 + u * h2);
40 }

```

## 19. Pohlig-Hellman 离散对数 [xzl/Pohlig-Hellman 离散对数, 数论.cpp]

Pohlig-Hellman 离散对数算法，求解同余方程  $a^x \equiv b \pmod{m}$  的最小解  $x$  或者报告无解，要求  $m$  为质数。ord 用于求出  $a$  关于  $m$  的阶数。算法需要实现快速幂  $\text{qpow}(a, k, m)$ 、快速乘  $\text{qmul}(a, b, m)$ 、素数判定  $\text{isprime}(n)$  和使用扩展 Euclid 算法求出的逆元  $\text{inv}(x, m)$ 。p0、k0、c0 存放的是  $m-1$  的质因数分解，p1、k1、c1 存放的是  $a$  关于  $m$  的阶数的质因数分解。factor 是 Pollard- $\rho$  质因数分解算法。设阶数的质因数分解为  $p_1^{k_1} p_2^{k_2} \dots p_n^{k_n}$ ，则时间复杂度为  $O(\sum_{i=1}^n k_i (\log m + \sqrt{p_i}))$ 。

```

1 #define KMAX 64
2 static i64 p0[KMAX], p1[KMAX];
3 static int k0[KMAX], k1[KMAX], c0, c1;
4 inline i64 _f(i64 x, i64 m) {
5     i64 r = qmul(x, x, m) + 1;
6     if (r >= m) r -= m;
7     return r;
8 }
9 inline void factor(i64 n) {
10     if (n == 2 || isprime(n)) p0[c0++] = n;
11     else if (!(n & 1)) factor(2), factor(n >> 1);
12     else {
13         for (int i = 1; ; i++) {
14             i64 x = i, y = _f(x, n), p = __gcd(y - x, n);
15             while (p == 1) {
16                 x = _f(x, n);
17                 y = _f(_f(y, n), n);
18                 p = __gcd((y - x + n) % n, n) % n;
19             }
20             if (p != 0 && p != n) {
21                 factor(p), factor(n / p);
22                 return;
23             }
24         }
25     }
26 }
27 inline void psort(i64 *p, int *k, int &c) {
28     sort(p, p + c);
29     int t = 0;
30     for (int i = 0, j; i < c; i = j) {
31         for (j = i; j < c && p[i] == p[j]; j++);
32         p[t] = p[i];
33         k[t++] = j - i;
34     }
35     c = t;
36 }
37 void ord(i64 a, i64 m, int p = 0, i64 cur = 1) {
38     static i64 tmp[KMAX + 10], mi;
39     static int t;
40     if (p == 0) mi = LLONG_MAX;

```

```

37 if (p == c0 && qpow(a, cur, m) == 1 && cur < mi) {
38     mi = cur;
39     memcpy(p1, tmp, sizeof(i64) * t);
40     c1 = t;
41 } else if (p != c0) {
42     int t0 = t;
43     for (int k = 0; k <= k0[p] && cur < mi; k++, cur *= p0[p]) {
44         if (k) tmp[t++] = p0[p];
45         ord(a, m, p + 1, cur);
46     } t = t0;
47 }
48 inline i64 log(i64 a, i64 b, i64 m, i64 p, int k) {
49     typedef unordered_map<i64, i64> Map;
50     static Map tb;
51     i64 pw = 1, bc, bt = 1, s = 1;
52     for (int i = 1; i < k; i++) pw *= p;
53     i64 g = qpow(a, pw, m), ai = inv(a, m), x = 0;
54     for (bc = g; s * s <= p; s++) bc = qmul(bc, g, m);
55     tb.clear();
56     for (i64 i = 1, t = bc; i <= s; i++, t = qmul(t, bc, m))
57         tb.insert(make_pair(t, i));
58     for (int i = 0; i < k; i++, pw /= p, bt *= p) {
59         i64 b0 = qpow(qmul(b, qpow(ai, x, m), m), pw, m), d = -1;
60         for (i64 j = 0, t = b0; j < s; j++, t = qmul(t, g, m)) {
61             Map::iterator it = tb.find(t);
62             if (it != tb.end()) {
63                 d = it->second * s - j;
64                 if (d >= p) d -= p;
65                 break;
66             }
67             if (d == -1) return -1;
68             x += bt * d;
69         } return x;
70 }
71 inline i64 log(i64 a, i64 b, i64 m) {
72     if (a == 1) return b == 1 ? 0 : -1;
73     i64 m0 = 1, x = 0;
74     for (int i = 0; i < c1; i++)
75         for (int j = 0; j < k1[i]; j++) m0 *= p1[i];
76     for (int i = 0; i < c1; i++) {
77         i64 pw = p1[i];
78         for (int j = 1; j < k1[i]; j++) pw *= p1[i];
79         i64 mi = m0 / pw, r = log(qpow(a, mi, m), qpow(b, mi, m), m, p1[i], k
80 1[i]);
81         if (r == -1) return -1;
82         x = (x + qmul(qmul(r, mi, m0), inv(mi, pw), m0)) % m0;
83     } return x < 0 ? x + m0 : x;
84 }
85 //factor(m - 1);
86 //psort(p0, k0, c0);
87 //ord(a, m);
88 //psort(p1, k1, c1);
89 //i64 ans = log(a, b, m);

```

## 20. Pohlig\_Hellman [lmj/Pohlig\_Hellman.cpp]

用来对 smooth 的模数  $p$  求离散对数。如果  $p-1$  的质因数分解中最大的素因子比较小可以使用。getlog 用来取对数，getroot 算原根，枚举时，只需要判断这个数的  $(p-1)/\text{prime factor}$

次幂是不是全部都不是 1 就可以。考虑  $n = p^e$  阶循环群，原根  $g$  是生成元，现在要找  $g^x = h$ 。

1. 令  $x_0 = 0$
2. 计算  $r = g^{p^{e-1}}$ ，这个元素阶数为  $p$
3. 对  $k = 0 \dots e-1$  计算
  1.  $h_k = (g^{-x_k} h)^{p^{e-1-k}}$ ，这个元素同样是  $p$  阶的，在  $\langle r \rangle$  中
  2. 用 BSGS（或者暴力）求出  $d_k \in \{0, \dots, p-1\}$  满足  $r^{d_k} = h_k$
  3. 令  $x_{k+1} = x_k + p^k d_k$
4. 返回  $x_e$

即设  $x = c_0 + c_1 p + c_2 p^2 + \dots + c_{e-1} p^{e-1}$ ，每一次进行的  $p^{e-1-k}$  次方可以令之后的数都是  $p^e$  的倍数，从而都是 1，只留下  $g^{c_k p^{e-1}}$  这一项（之前的项被 3.1. 中的逆元消去了），然后计算这一项。如果  $n = p_1^{e_1} p_2^{e_2}$  这样，考虑对每个质因数，调用一次  $g_i = g^{n/p_i^{e_i}}$ ， $h_i = h^{n/p_i^{e_i}}$ ，得到  $x \equiv x_i \pmod{p_i^{e_i}}$ ，CRT 求解就可以了。这一步同样是把其他无关的素因子的阶通过高次幂消去。ExGCD 似乎过程是不会爆 long long 的 (?)。复杂度  $O(\sum e_i (\log n + \sqrt{p_i}))$ 。

```

1 typedef long long LL;
2 LL pfactor[1200], totf;
3 LL gene[1200];
4 void exgcd(LL a, LL b, LL &x, LL &y) {
5     if (b == 0) { x = 1; y = 0; return; }
6     exgcd(b, a % b, x, y);
7     LL tp = x;
8     x = y; y = tp - a / b * y;
9 }
10 LL inv ( LL a , LL mod ) {
11     LL x , y;
12     exgcd ( a , mod , x , y );
13     return (x%mod+mod)%mod;
14 }
15 LL qmul ( LL a , LL b , LL m ) {
16     a %= m; b %= m;
17     LL r = a*b, s = (long double)(a)*b/m;
18     return ((r-m*s)%m+m)%m;
19 }
20 LL fast_mul ( LL a , LL b , LL c , LL mod ) {
21     return qmul(qmul(a,b,mod),c,mod);
22 }
23 pair<LL,LL> crt ( pair<LL,LL> a , pair<LL,LL> b ) {
24     if ( a.first == -1 ) return b;
25     a.first = fast_mul(a.first,b.second,inv(b.second,a.second),a.second*b.seco
26     nd) + fast_mul(b.first,a.second,inv(a.second,b.second),a.second*b.seco
27     nd);
28     a.second *= b.second;
29     a.first %= a.second;
30     return a;
31 }
32 LL mpow ( LL f , LL x , LL mod ) {
33     LL s = 1;
34     while ( x ) {
35         if ( x % 2 ) s = qmul(s,f,mod);
36         f = qmul(f,f,mod); x >>= 1;
37     } return s;
38 }

```

```

37 pair<LL,LL> solve ( LL g , LL h , LL mod , LL prime , LL e , LL p ) { // mod = prime^e
38     LL j , k , r = mpow ( g , mod / prime , p ) , x = 0 , hh;
39     LL ret = 0 , nowp = mod / prime , pp = 1;
40     gene[0] = 1;
41     for ( k = 1 ; k <= prime - 1 ; k++ ) {
42         gene[k] = qmul ( gene[k-1],r,p);
43     }
44     for ( k = 0 ; k <= e - 1 ; k++ ) {
45         h = qmul(h,inv(mpow(g,x,p),p),p);
46         hh = mpow ( h , nowp , p );
47         for ( j = 0 ; j <= prime - 1 ; j++ ) {
48             if ( gene[j] == hh ) break;
49         }
50         nowp = nowp / prime;
51         x = j * pp;
52         ret += x;
53         pp = pp * prime;
54     } return make_pair(ret,mod);
55 }
56 LL getlog ( LL a , LL root , LL p ) {
57     LL i , j , tp , tmp;
58     pair<LL,LL> ret , rem;
59     tp = p - 1;
60     rem.first = -1;
61     for ( i = 2 ; tp != 1 ; i++ ) {
62         if ( tp % i == 0 ) {
63             tmp = 1; j = 0;
64             while ( tp % i == 0 ) {
65                 tmp = tmp * i;
66                 j++; tp /= i;
67             }
68             ret = solve ( mpow ( root , p / tmp , p ) , mpow ( a , p / tmp , p ) , tmp , i , j , p );
69             rem = crt ( rem , ret );
70         } return rem.first;
71     }
72 LL getroot ( LL p ) {
73     LL i , j , tp = p - 1;
74     totf = 0;
75     for ( i = 2 ; tp != 1 ; i++ ) {
76         if ( tp % i == 0 ) {
77             pfactor[++totf] = i;
78             while ( tp % i == 0 ) tp /= i;
79         }
80     }
81     for ( i = 2 ; i < p ; i++ ) {
82         for ( j = 1 ; j <= totf ; j++ ) {
83             if ( mpow ( i , (p-1)/pfactor[j] , p ) == 1 ) break;
84         }
85         if ( j == totf + 1 ) return i;
86     } return -1;
87 }
88 LL work ( LL p , LL a , LL b ) { // return x, such that a^x = b (mod p)
89     LL i , j , rt , la , lb , x , y , g;
90     rt = getroot ( p );
91     la = getlog ( a , rt , p ); // rt^la = a (mod p)
92     lb = getlog ( b , rt , p );
93     // x*la = lb (mod p-1)

```

```

93     g = __gcd ( la , p - 1 );
94     exgcd ( la , p - 1 , x , y );
95     if ( lb % g != 0 ) return -1;
96     x = (x*(p-1)+(p-1))%(p-1);
97     return qmul ( x , (lb/g) , (p - 1)/__gcd(la,p-1) );
98 }

```

## 21. continued\_fraction [lmj/continued\_fraction.cpp]

连分数相关/最佳分数逼近

这个代码用来处理  $\frac{a}{b} < \frac{x}{y} < \frac{c}{d}$ ，给出一组  $x, y$  最小的解，注意， $x$  最小就对应了  $y$  最小，二者是等价的。（请自行保证  $\frac{a}{b} < \frac{c}{d}$ ）

结果为 num/dom，过程中，dec 保存了两个分数的连分数展开，len 是两个数组的长度。例如  $[4; 1, 2]$  表示的分数是  $4 + \frac{1}{1+\frac{1}{2}}$ 。

连分数的一些性质  $[4; 1, 4, 3] = [4; 1, 4, 2, 1] = [4; 1, 4, 3, \infty]$ ，可以在最后加一个 1 上去（只能有一个，因为 1 不能再减 1 了），完成了之后，后面可以认为有无穷个  $\infty$ 。

求一个分数的连分数展开：把整数部分减掉，放到答案数组里，然后把剩下的真分数去倒数，重复做到  $\frac{0}{x}$  就是结果。无理数类似，但是想办法存数值。

代码中求的是两个公共前缀，在第一个不同处取  $\min\{a_i, b_i\} + 1$  就是分子分母最小的解。

复杂度与辗转相除类似， $O(\log n)$ 。

如果要求的是和一个分数最接近的数，即限制了分子，分母有一个界，那么同样求出这个分数的连分数表示，然后考虑每一个前缀截断一下，并且把最后一个数字  $-1, +0, +1$  分别求一下看看哪个最接近。复杂度  $O(\log^2 n)$ ，卡时间的话可以尝试二分一下，变成  $O(\log n \log \log n)$ 。

（此段的代码没实现过，不保证正确性）（理论大概是连分数展开是最优的分数逼近，所以可以这样搞（不会证，不记得对不对））

```

1 Long Long dec1[1200] , dec2[1200] , len1 , len2;
2 Long Long num , dom;
3 void getfrac ( Long Long *d , Long Long &l , Long Long a , Long Long b )
4 {
5     l = 1;
6     d[1] = a / b;
7     a %= b;
8     while ( a != 0 ) {
9         swap ( a , b );
10        d[++l] = a / b;
11        a %= b;
12    }
13    void work () {
14        Long Long i;
15        getfrac ( dec1 , len1 , a , b );
16        getfrac ( dec2 , len2 , c , d );
17        dec1[len1+1] = 2147483647777777777ll;
18        dec2[len2+1] = 2147483647777777777ll;
19        for ( i = 1 ; i <= len1 && i <= len2 ; i++ ) {
20            if ( dec1[i] != dec2[i] ) break;
21        }
22        dec1[i] = min ( dec1[i] , dec2[i] ) + 1;
23        num = dec1[i]; dom = 1;
24        for ( i-- ; i >= 1 ; i-- ) {
25            swap ( num , dom );
26            num = num + dom * dec1[i];

```

```

26 }
27 printf ( "%lld %lld\n" , num , dom );
28 }

```

## 22. min\_25\_seive\_lattice\_on\_circle [lmj/min\_25\_seive\_lattice\_on\_circle.cpp]

求  $a^2 + b^2 = c^2$  对所有  $c \leq n$ , 并满足  $1 \leq a \leq b$ 。

就是(圆上的整点数量-4)/8

如果是  $a^2 + b^2 = c$  可以直接枚举  $a$ 。

```

1 #include <stdio.h>
2 #include <algorithm>
3
4 using namespace std;
5
6 typedef long long LL;
7 const LL NN = 420000;
8 const LL block = 100000;
9 LL n , p[1200000] , prime[NN] , tot;
10 LL value[NN] , cnt , limit , pos[NN];
11 LL sumh[NN] , sumh1[NN] , h0[NN] , h1[NN];
12 LL h[NN] , g[NN];
13 void clear () {
14     int i;
15     for ( i = 0 ; i <= NN - 1 ; i++ ) {
16         value[i] = sumh1[i] = pos[i] = sumh[i] = h0[i] = h1[i] = h[i] = g[i]
= 0;
17         prime[i] = 0;
18     }
19     tot = 0; cnt = limit = 0;
20 }
21 LL getpos ( LL x ) {
22     return x<=limit?x:pos[n/x];
23 }
24 void predo () {
25     LL i , j;
26     for ( i = 2 ; i <= block ; i++ ) {
27         if ( !p[i] )
28             prime[++tot] = i;
29         for ( j = 1 ; j <= tot && i * prime[j] <= block ; j++ ) {
30             p[i*prime[j]] = 1;
31             if ( i % prime[j] == 0 ) break;
32         }
33     }
34     cnt = 0;
35     for ( i = 1 ; i * i <= n ; i++ ) value[++cnt] = i;
36     i--; limit = i;
37     for ( ; i >= 1 ; i-- ) if ( n / i != value[cnt] ) {
38         value[++cnt] = n / i;
39         pos[i] = cnt;
40     }
41     LL tmp;
42     for ( i = 1 ; i <= tot ; i++ ) {
43         if ( prime[i] % 4 == 1 ) tmp = 3;
44         else tmp = 1;
45         if ( i == 1 ) tmp = 0;
46         sumh[i] = (sumh[i-1] + tmp);
47         if ( i != 1 )

```

```

48         sumh1[i] = sumh1[i-1] + 1;
49     }
50     for ( i = 1 ; i <= cnt ; i++ ) { //cal h from 2 to i
51         if ( value[i] <= 2 ) continue;
52         h0[i] = value[i] - value[i] % 4; //modulo before multiply
53         if ( value[i] % 4 >= 1 ) h0[i] += 3;
54         if ( value[i] % 4 == 3 ) h0[i] += 1;
55         h0[i] -= 4;
56         if ( value[i] >= 2 ) h0[i] += 1;
57         if ( value[i] == 1 ) h1[i] = 0;
58         else if ( value[i] == 2 ) h1[i] = 0;
59         else h1[i] = ((value[i]+1)/2-1);
60     }
61     for ( i = 2 ; i <= tot ; i++ ) {
62         for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- ) {
63             if ( prime[i] % 4 == 1 )
64                 h0[j] = (h0[j] - 1 * (h0[getpos(value[j]/prime[i])]-sumh[i-1])) ;
65             else
66                 h0[j] = (h0[j] - ((h1[getpos(value[j]/prime[i])]-sumh1[i-1]) * 4
- (h0[getpos(value[j]/prime[i])]-sumh[i-1])) ) );
67         }
68         for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- ) {
69             h1[j] = ( h1[j] - 1 * (h1[getpos(value[j]/prime[i])]-sumh1[i-1]) )
);
70         }
71     }
72     for ( i = 1 ; i <= cnt ; i++ )
73         h[i] = h0[i] + (i>1?1:0);
74 }
75 LL getf ( LL p , LL e ) {
76     if ( p % 4 == 1 )
77         return 2 * e + 1;
78     else return 1;
79 }
80 void min25 () {
81     LL i , j , e , now , tmp;
82     for ( j = cnt ; j >= 1 ; j-- ) g[j] = h[j];
83     for ( i = tot ; i >= 1 ; i-- )
84         for ( j = cnt ; prime[i]*prime[i]<=value[j];j-- )
85             for ( e = 1 , now = prime[i];now*prime[i]<=value[j];e++,now=now*pri
me[i] ) {
86                 g[j]=(g[j]+getf(prime[i],e)*(g[getpos(value[j]/now)]-h[prime[i]]
+getf(prime[i],e+1)) );
87             }
88     printf ( "%lld\n" , ((g[cnt]+1)-n)/2 );
89 }
90 void work () {
91     scanf ( "%lld" , &n );
92     predo ();
93     min25 ();
94     clear ();
95 }
96 int main () {
97     int t;
98     scanf ( "%d" , &t );
99     while ( t-- ) work ();
100     return 0;
101 }

```

### 23. min\_25\_sieve [lmj/min\_25\_sieve.cpp]

记号同whzzt18年集训队论文。 $f(x)$  表示被求和的积性函数，并且在质数点值是一个低阶多项式。 $h(n) = \sum_{2 \leq p \leq n, p \text{ 是质数}} f(p)$ ,  $h_{n,m} = \sum_{\substack{p \leq n \\ p \nmid m}} f(p)$ ,  $g_{n,m} = \sum_{\substack{p \leq n \\ p \nmid m}} f(p)$ , 注意从 2 开始。考虑线性筛的过程，每次筛掉一个最小的质数。对于  $h(n,m)$  和  $g(n,m)$  进行筛法时，考虑枚举  $i$  的最小质因子，并且合数的最小质因子不超过  $\sqrt{n}$ 。

其中  $h(n) = h(n,0)$ ,  $h(n,m)$  是筛  $h(n)$  的过程， $g(n,0)$  就是答案。

从而写出递推式（假设质数点值  $f(p) = p^k$ ）（这时候  $f(p)$  是完全积性的，对下面的扣掉的性质特别好，可以在  $x \in [\frac{n}{p_j}]$  与  $p_j$  不互质的时候也能这样直接扣掉）

$$h(n,j) = h(n,j-1) - p_j^k \left[ h\left(\left\lfloor \frac{n}{p_j} \right\rfloor, j-1\right) - h(p_{j-1}) \right]$$

其中  $p_{j-1} \leq \sqrt{n}$  可以把  $h(p_{j-1})$  打表，扣掉是要把最小质因子小的去掉，并且只有  $p_j^2 \leq n$  时转移不为 0。从小到大按层转移。

$$g(n,i) = g(n,i+1) + \sum_{e \geq 1, p_i^{e+1} \leq n} \left[ f(p_i^e) \left[ g\left(\left\lfloor \frac{n}{p_i^e} \right\rfloor, i+1\right) - h(p_i) \right] + f(p_i^{e+1}) \right]$$

同样的，只有  $p_i^2 \leq n$  是存在转移，分层计算即可。

初值  $h(n,0) = \sum_{i=1}^n i^k$  全都算上，然后把不是质数的点值筛出去， $g(n,m) = h(n)$ ，先只计算质数上的点值，然后把合数的点值逐个加入到  $g$  中。

最后的答案是  $g(n,0) + f(1)$ 。

<https://blog.csdn.net/HOWARLI/article/details/80339931>

```
1 typedef Long Long LL;
2 const LL NN = 420000;
3 const LL block = 100000;
4 const LL mod = 1000000007;
5 const LL inv2 = 500000004;
6 LL n;
7 LL p[120000], prime[NN], tot;
8 LL value[NN], cnt, limit, pos[NN];
9 LL sumh[NN], h0[NN], h1[NN];
10 LL h[NN]; // sum of h[1..value[x]]
11 LL g[NN];
12 LL getpos ( LL x ) {
13     return x<=limit?x:pos[n/x];
14 }
15 void predo () {
16     LL i, j;
17     for ( i = 2 ; i <= block ; i++ ) {
18         if ( !p[i] )
19             prime[++tot] = i;
20         for ( j = 1 ; j <= tot && i * prime[j] <= block ; j++ ) {
21             p[i*prime[j]] = 1;
22             if ( i % prime[j] == 0 ) break;
23         }
24     }
25     cnt = 0;
26     for ( i = 1 ; i * i <= n ; i++ ) value[++cnt] = i;
```

```
27     i--; limit = i;
28     for ( ; i >= 1 ; i-- ) if ( n / i != value[cnt] ) {
29         value[++cnt] = n / i;
30         pos[i] = cnt;
31     }
32     for ( i = 1 ; i <= tot ; i++ )
33         sumh[i] = (sumh[i-1] + prime[i]) % mod;
34     for ( i = 1 ; i <= cnt ; i++ ) { // cal h from 2 to i
35         h0[i] = ((value[i]-1)%mod*((value[i]+2)%mod)%mod*inv2) % mod; // modulo
        before multiply
36         h1[i] = (value[i] - 1) % mod;
37     }
38     for ( i = 1 ; i <= tot ; i++ ) {
39         for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- ) {
40             h0[j] = ( (h0[j] - prime[i] * (h0[getpos(value[j]/prime[i]])-sumh[i-1]) ) % mod );
41             if ( h0[j] < 0 ) h0[j] += mod;
42             h1[j] = ( (h1[j] - 1 * (h1[getpos(value[j]/prime[i]])-(i-1)) ) % mod );
43             if ( h1[j] < 0 ) h1[j] += mod;
44         }
45     }
46     for ( i = 1 ; i <= cnt ; i++ ) // f(p)=p-1
47         h[i] = ( h0[i] - h1[i] + mod ) % mod;
48 }
49 LL getf ( LL p , LL e ) {
50     return p ^ e;
51 }
52 void min25 () {
53     LL i, j, e, now, tmp;
54     for ( j = cnt ; j >= 1 ; j-- ) g[j] = h[j];
55     for ( i = tot ; i >= 1 ; i-- )
56         for ( j = cnt ; prime[i] * prime[i] <= value[j] ; j-- )
57             for ( e = 1, now = prime[i] ; now * prime[i] <= value[j] ; e++, now = now * prime[i] )
58                 g[j] = ( g[j] + getf(prime[i],e) * (g[getpos(value[j]/now)]-h[prime[i]]+mod) + getf(prime[i],e+1) ) % mod;
59     printf ( "%lld\n" , (g[cnt] + 1) % mod );
60 }
61 void work () {
62     scanf ( "%lld" , &n );
63     predo ();
64     min25 ();
65 }
```

### 24. 幂级数前缀和 [xzl/power-series.cpp]

KMAX 表示插值多项式次数最大值，MOD 为模数，要求为质数。qpow 是快速幂，add 是取模加法。 $f[0]$  到  $f[K+1]$  存放的是前缀和函数的取值，下面的预处理是暴力快速幂求出的，如果要线性复杂度请换成线性筛。插值方法为 Lagrange 插值法，单次计算复杂度为  $\Theta(K)$ 。注意计算结果可能为负数。使用时可以开一个 PowerSeries 的数组。

```
1 static bool _initialized;
2 static int cnt;
3 static i64 _fi[KMAX + 10], _tmp[KMAX + 10];
4 struct PowerSeries {
5     static void init() {
6         _fi[0] = 1;
```

```

7   for (int i = 2; i <= KMAX + 1; i++) _fi[0] = _fi[0] * i % MOD;
8   _fi[KMAX + 1] = qpow(_fi[0], MOD - 2);
9   for (int i = KMAX; i >= 0; i--) _fi[i] = _fi[i + 1] * (i + 1) % MOD;
10  _initialized = true;
11  }
12  int K; i64 *f;
13  PowerSeries() : PowerSeries(cnt++) {}
14  PowerSeries(int _K) : K(_K) {
15      if (!_initialized) init();
16      f = new i64[K + 2]; f[0] = 0;
17      for (int i = 1; i <= K + 1; i++) f[i] = (f[i - 1] + qpow(i, K)) % MOD;
18  }
19  ~PowerSeries() { delete[] f; }
20  i64 operator()(i64 n) const {
21      n %= MOD; _tmp[K + 2] = 1;
22      for (int i = K + 1; i >= 1; i--) _tmp[i] = _tmp[i + 1] * (n - i) % MOD;
23  D;
24      i64 ret = 0, pre = 1;
25      for (int i = 0, b = K & 1 ? 1 : -1; i <= K + 1; i++, b = -b) {
26          add(ret, b * f[i] * pre % MOD * _tmp[i + 1] % MOD * _fi[i] % MOD *
27          _fi[K + 1 - i] % MOD);
28          pre = pre * (n - i) % MOD;
29      } return ret;
30  };

```

## 25. 类 Euclid 算法 [xzl/sim-euclid.cpp]

类 Euclid 算法在模意义下计算：

$$\sum_{k=0}^n k^p \left[ \frac{ak+b}{c} \right]^q$$

其中所有参数非负，在计算过程中始终保证  $K = p + q$  不减， $a, c \geq 1$  且  $b \geq 0$ 。需要 Bernoulli 数 ( $B_1 = +1/2$ ) 来计算自然数幂前缀和  $S_p(x) = \sum_{k=1}^x k^p = \sum_{k=1}^{p+1} a_k^{(p)} x^k$ ，其中  $a_k^{(p)} = \frac{1}{p+1} \binom{p+1}{k} B_{p+1-k}$ 。代码中 has 为访问标记数组，每次使用前需清空，val 为记忆化使用的数组，qpow 是快速幂，S 是自然数幂前缀和，A 记录了  $a_k^{(p)}$ ，C 是组合数。时空复杂度为  $O(K^3 \log \max\{a, c\})$ 。注意参数的范围防止整数溢出。如果只是计算直线下整点数量，则主算法部分只用被注释掉的四句话。

算法主要分为三个情况，其中  $a \geq c$  和  $b \geq c$  的情况比较简单。当  $a, b < c$  时，用  $j = \lfloor (ak+b)/c \rfloor$  进行代换，注意最终要转化为  $\lfloor (c(j-1) + c - b - 1)/a \rfloor < k \leq \lfloor (cj + c - b - 1)/a \rfloor$ ，再进行一次分部求和即可。注意处理  $k \leq n$  这个条件。

$n$	0	1	2	4	6	8
$B_n$	1	$\frac{1}{2}$	$\frac{1}{6}$	$-\frac{1}{30}$	$\frac{1}{42}$	$-\frac{1}{30}$
$n$	10	12	14	16	18	20
$B_n$	$\frac{5}{66}$	$-\frac{691}{2730}$	$\frac{7}{6}$	$-\frac{3617}{510}$	$\frac{43867}{798}$	$-\frac{174611}{330}$

```

1 i64 F(i64 n, i64 a, i64 b, i64 c, int p, int q, int d = 0) {
2     if (n < 0) return 0;
3     if (has[d][p][q]) return val[d][p][q];

```

```

4     has[d][p][q] = true;
5     i64 &ret = val[d][p][q] = 0; // 后面的 d 均加 1
6     if (!q) ret = S(n, p) + (!p); // 注意 p = 0 的边界情况
7     else if (!a) {
8         ret = qpow(b / c, q) * (S(n, p) + (!p)) % MOD;
9         //return b / c * (n + 1) % MOD;
10    } else if (a >= c) {
11        i64 m = a / c, r = a % c, mp = 1;
12        for (int j = 0; j <= q; j++, mp = mp * m % MOD)
13            add(ret, C[q][j] * mp % MOD * F(n, r, b, c, p + j, q - j, d) % MOD);
14    } //return (F(n, a % c, b, c) + a / c * n % MOD * (n + 1) % MOD * INV2)
15    % MOD;
16    } else if (b >= c) {
17        i64 m = b / c, r = b % c, mp = 1;
18        for (int j = 0; j <= q; j++, mp = mp * m % MOD)
19            add(ret, C[q][j] * mp % MOD * F(n, a, r, c, p, q - j, d) % MOD);
20    } //return (F(n, a, b % c, c) + b / c * (n + 1)) % MOD;
21    } else {
22        i64 m = (a * n + b) / c;
23        for (int k = 0; k < q; k++) {
24            i64 s = 0;
25            for (int i = 1; i <= p + 1; i++)
26                add(s, A[p][i] * F(m - 1, c, c - b - 1, a, k, i, d) % MOD);
27            add(ret, C[q][k] * s % MOD);
28        }
29        ret = (qpow(m, q) * S(n, p) - ret) % MOD;
30        //return (m * n - F(m - 1, c, c - b - 1, a)) % MOD;
31    } return ret;
32 }

```

## 26. 线性筛 & 杜教筛 [xzl/dyh.cpp]

计算积性函数  $f(n)$  的前缀和  $F(n) = \sum_{k=1}^n f(k)$ ：先选定辅助函数  $g(n)$  进行 Dirichlet 卷积，得到递推公式：

$$F(n) = \frac{1}{g(1)} \left( \sum_{k=1}^n (f \times g)(k) - \sum_{k=2}^n g(k) F\left(\left\lfloor \frac{n}{k} \right\rfloor\right) \right)$$

对于 Euler 函数  $\varphi(n)$ ，选定  $g(n) = 1$ ，得：

$$\Phi(n) = \frac{n(n+1)}{2} - \sum_{k=2}^n \Phi\left(\left\lfloor \frac{n}{k} \right\rfloor\right)$$

对于 Mobius 函数  $\mu(n)$ ，选定  $g(n) = 1$ ，得：

$$M(n) = 1 - \sum_{k=2}^n M\left(\left\lfloor \frac{n}{k} \right\rfloor\right)$$

如果没有预处理，时间复杂度为  $\Theta(n^{3/4})$ ，空间复杂度为  $\Theta(\sqrt{n})$ 。如果预处理前  $\Theta(n^{2/3})$  项前缀和，则时空复杂度均变为  $\Theta(n^{2/3})$ 。下面的代码以 Euler 函数为例，能够在 1s 内计算  $10^{10}$  内的数据。可以多次调用。

```

1 #define S 17000000 // for F(10^10)
2 static int pc, pr[S + 10];
3 static i64 phi[S + 10];
4 static unordered_map<i64, i64> dat;
5 inline void sub(i64 &a, i64 b) { a -= b; if (a < 0) a += MOD; }
6 inline i64 c2(i64 n) { n %= MOD; return n * (n + 1) % MOD * INV2 % MOD; }

```



```

7 i64 F(i64 n) { // 杜教筛
8   if (n <= S) return phi[n];
9   if (dat.count(n)) return dat[n];
10  i64 &r = dat[n] = c2(n);
11  for (i64 i = 2, l; i <= n; i = l + 1) {
12    i64 p = n / i;
13    l = n / p;
14    sub(r, (l - i + 1) * F(p) % MOD); // (1 - i + 1) % MOD?
15  }
16  return r;
17 }
18 phi[1] = 1; // 线性筛
19 for (int i = 2; i <= S; i++) {
20   if (!phi[i]) {
21     pr[pc++] = i;
22     phi[i] = i - 1;
23   }
24   for (int j = 0; pr[j] * i <= S; j++) {
25     int p = pr[j];
26     if (i % p) phi[i * p] = phi[i] * (p - 1);
27     else {
28       phi[i * p] = phi[i] * p;
29       break;
30     }
31   }
32 }
33 for (int i = 2; i <= S; i++) add(phi[i], phi[i - 1]);

```

## 27. fft [lmj/fft.cpp]

```

1 const int maxn = 120000;
2 const double pi = acos(-1);
3 struct complex {
4   double r, i;
5 } a[maxn*4], b[maxn*4], c[maxn*4], d[maxn*4];
6 complex operator + (complex x1, complex x2) {complex y; y.r = x1.r + x2.r; y.i = x1.i + x2.i; return y;}
7 complex operator - (complex x1, complex x2) {complex y; y.r = x1.r - x2.r; y.i = x1.i - x2.i; return y;}
8 complex operator * (complex x1, complex x2) {complex y; y.r = x1.r * x2.r - x1.i * x2.i; y.i = x1.r * x2.i + x1.i * x2.r; return y;}
9 int n, m, N;
10 int rev (int x) {int i, y; i = 1; y = 0; while (i < N) {y = y * 2 + (x % 2); x >>= 1; i <<= 1; } return y;}
11 void br (complex *x) {int i; for (i = 0; i < N; i++) d[rev(i)] = x[i]; for (i = 0; i < N; i++) x[i] = d[i];}
12 void FFT (complex *x, int f) {
13   int i, j, s, k;
14   complex w, wm, u, t;
15   br (x);
16   for (s = 2; s <= N; s *= 2) {
17     k = s / 2;
18     wm.r = cos(2*pi/s); wm.i = sin(2*pi/s) * f;
19     for (i = 0; i < N; i += s) {
20       w.r = 1.0; w.i = 0.0;
21       for (j = 1; j <= k; j++) {
22         u = x[i+j-1]; t = x[i+j-1+k] * w;
23         x[i+j-1] = u + t;
24         x[i+j-1+k] = u - t;
25         w = w * wm;
26       }
27     }
28   }
29 }

```

```

27 if (f == -1) for (i = 0; i < N; i++) x[i].r = x[i].r / N;
28 }
29 void work () {
30   int i;
31   scanf ("%d%d", &n, &m);
32   N = 1;
33   while (N < n + m + 2) N = N * 2;
34   for (i = 0; i <= n; i++) scanf ("%lf", &a[i].r);
35   for (i = 0; i <= m; i++) scanf ("%lf", &b[i].r);
36   FFT (a, 1); FFT (b, 1);
37   for (i = 0; i < N; i++) c[i] = a[i] * b[i];
38   FFT (c, -1);
39   for (i = 0; i <= n + m; i++) printf ("%d%c", int (c[i].r + 0.5), i==n+m?'\n':' ');
40 }

```

## 28. ntt [lmj/ntt.cpp]

```

1 const Long Long maxn = 120000;
2 const Long Long mod = 998244353;
3 const Long Long omega = 3;
4 Long Long a[maxn*4], b[maxn*4], c[maxn*4], d[maxn*4];
5 Long Long n, m, N, in;
6 Long Long pow (Long Long f, Long Long x) {Long Long s = 1; while (x) {if (x % 2) s = (s*f) % mod; f = (f*f) % mod; x >>= 1; } return s;}
7 Long Long inv (Long Long x) {return pow (x, mod - 2);}
8 Long Long rev (Long Long x) {Long Long i, y; i = 1; y = 0; while (i < N) {y = y * 2 + (x % 2); i <<= 1; x >>= 1; } return y;}
9 void br (Long Long *x) {Long Long i; for (i = 0; i < N; i++) d[rev(i)] = x[i]; for (i = 0; i < N; i++) x[i] = d[i];}
10 void FFT (Long Long *x, Long Long f) {
11   Long Long i, j, s, k;
12   Long Long w, wm, u, t;
13   br (x);
14   for (s = 2; s <= N; s *= 2) {
15     k = s / 2;
16     wm = pow (omega, (mod-1) / s);
17     if (f == -1) wm = inv (wm);
18     for (i = 0; i < N; i += s) {
19       w = 1;
20       for (j = 1; j <= k; j++) {
21         u = x[i+j-1]; t = (x[i+j-1+k]*w) % mod;
22         x[i+j-1] = (u + t) % mod;
23         x[i+j-1+k] = (u - t + mod) % mod;
24         w = (w*wm) % mod;
25       }
26     }
27   }
28   if (f == -1) for (i = 0; i < N; i++) x[i] = (x[i] * in) % mod;
29 }
30 void work () {
31   Long Long i;
32   scanf ("%lld%lld", &n, &m);
33   N = 1;
34   while (N < n + m + 2) N = N * 2;
35   for (i = 0; i <= n; i++) scanf ("%lld", &a[i]);
36   for (i = 0; i <= m; i++) scanf ("%lld", &b[i]);
37   in = inv (N);
38   FFT (a, 1); FFT (b, 1);
39   for (i = 0; i < N; i++) c[i] = (a[i]*b[i]) % mod;
40   FFT (c, -1);

```

```

39  for ( i = 0 ; i <= n + m ; i++ ) printf ( "%lld%c" , c[i] , i==n+m?'\\n'
    : ' ');
40 }

```

## 29. polynomial\_all\_star [lmj/polynomial\_all\_star.cpp]

```

1 #include <bits/stdc++.h>
2 #define ll long long
3 #define vi vector<int>
4 #define pb push_back
5 #define mod 998244353
6 using namespace std;
7 inline int add(int a, int b) {
8     a += b;
9     return (a < mod) ? a : (a - mod);
10 }
11 inline int sub(int a, int b) {
12     a -= b;
13     return (a < 0) ? a + mod : a;
14 }
15 inline int mul(int a, int b) {
16     return (ll)a * b % mod;
17 }
18 int power(int a, int b) {
19     if (!b) return 1;
20     int u = power(a, b >> 1);
21     u = mul(u, u);
22     if (b & 1) u = mul(u, a);
23     return u;
24 }
25 vi rev, roots;
26 int mx = -1;
27 void init() {
28     mx = 23;
29     roots.resize(1 << mx);
30     for (int j = 1; j <= mx; j++) {
31         int mn = power(3, (mod - 1) >> j);
32         for (int i = 0; i < (1 << (j - 1)); i++) {
33             int npl = (1 << (j - 1)) | i;
34             if (i == 0) roots[npl] = 1;
35             else roots[npl] = mul(mn, roots[npl - 1]);
36         }
37     }
38 }
39 void calr(int m) {
40     rev.resize(1 << m);
41     if (mx == -1) init();
42     rev[0] = 0;
43     for (int j = 0; j < m; j++)
44         for (int k = 0; k < (1 << j); k++)
45             rev[k | (1 << j)] = rev[k] + (1 << (m - j - 1));
46 }
47 void dft(vi& a) {
48     int n = a.size(), r = 0; // 保证 a 是 2 幂
49     while ((1 << r) < n) r++;
50     calr(r);
51     for (int i = 0; i < n; i++)
52         if (rev[i] > i)
53             swap(a[rev[i]], a[i]);

```

```

54     for (int i = 1; i < n; i <= 1) {
55         for (int j = 0; j < n; j += (i << 1)) {
56             for (int k = 0; k < i; k++) {
57                 int mr = mul(a[i + j + k], roots[i + k]);
58                 a[i + j + k] = sub(a[j + k], mr);
59                 a[j + k] = add(a[j + k], mr);
60             }
61         }
62     }
63 }
64 vi operator * (vi a, vi b) {
65     int l = a.size() + b.size() - 1;
66     if ((ll)a.size() * b.size() < 100) {
67         vi fn(a.size() + b.size() - 1);
68         for (int i = 0; i < a.size(); i++)
69             for (int j = 0; j < b.size(); j++)
70                 fn[i + j] = add(fn[i + j], mul(a[i], b[j]));
71         return fn;
72     }
73     int r = 0;
74     while ((1 << r) < l) r++;
75     a.resize(1 << r), b.resize(1 << r);
76     dft(a);
77     dft(b);
78     int bk = power(1 << r, mod - 2);
79     for (int i = 0; i < (1 << r); i++)
80         a[i] = mul(mul(a[i], b[i]), bk);
81     reverse(a.begin() + 1, a.end());
82     dft(a);
83     a.resize(l);
84     return a;
85 }
86 vi inv(vi a) {
87     int n = a.size(), m = (n + 1) >> 1;
88     if (n == 1)
89         return vi(1, power(a[0], mod - 2));
90     vi k = inv(vi(a.begin(), a.begin() + m));
91     int r = 0;
92     while ((1 << r) < (2 * n)) r++;
93     int sz = (1 << r), inv_sz = power(sz, mod - 2);
94     calr(r);
95     a.resize(sz), k.resize(sz);
96     dft(a), dft(k);
97     for (int i = 0; i < sz; i++)
98         k[i] = mul(k[i], sub(2, mul(a[i], k[i])));
99     reverse(k.begin() + 1, k.end());
100    dft(k);
101    for (int i = 0; i < sz; i++)
102        k[i] = mul(k[i], inv_sz);
103    k.resize(n);
104    return k;
105}
106void otp(vi a) {
107    for (int i = 0; i < a.size(); i++)
108        printf("%d ", a[i]);
109    printf("\\n");
110}
111vi operator / (vi a, vi b) {

```

```

112 int n = a.size(), m = b.size();
113 if (n < m) return vi(1, 0);
114 reverse(a.begin(), a.end());
115 reverse(b.begin(), b.end());
116 a.resize(n - m + 1);
117 b.resize(n - m + 1);
118 a = a * inv(b);
119 a.resize(n - m + 1);
120 reverse(a.begin(), a.end());
121 return a;
122}
123vi operator - (vi a, vi b) {
124 if (a.size() < b.size())
125     a.resize(b.size());
126 for (int i = 0; i < b.size(); i++)
127     a[i] = sub(a[i], b[i]);
128 return a;
129}
130vi operator + (vi a, vi b) {
131 if (a.size() < b.size())
132     a.resize(b.size());
133 for (int i = 0; i < b.size(); i++)
134     a[i] = add(a[i], b[i]);
135 return a;
136}
137vi operator % (vi a, vi b) {
138 vi f = a - b * (a / b);
139 f.resize(b.size() - 1);
140 return f;
141}
142vi dw(vi f) {
143 //求导
144 int n = f.size();
145 if (n == 1) return vi{0};
146 vi g(n - 1);
147 for (int i = 0; i < n - 1; i++)
148     g[i] = mul(f[i + 1], (i + 1));
149 return g;
150}
151vi bk; // reverse
152vi up(vi f) {
153 //积分
154 int n = f.size();
155 vi g(n + 1);
156 if (bk.size() <= n) {
157     int ur = bk.size();
158     bk.resize(n + 1);
159     for (int i = max(ur, 1); i <= n; i++) {
160         if (i == 1) bk[i] = 1;
161         else bk[i] = mul(mod / i + 1, bk[mul(i, mod / i + 1)]);
162         // if (mul(i, bk[i]) != 1) cout << "WAAA" << endl;
163     }
164 }
165 for (int i = 1; i <= n; i++)
166     g[i] = mul(f[i - 1], bk[i]);
167 return g;
168}
169vi caly(vi f, vi x) {

```

```

170 //多点求值
171 int n = x.size();
172 vector<vi> mt(2 * n);
173 for (int i = 0; i < n; i++) {
174     mt[i + n].resize(2);
175     mt[i + n][0] = sub(0, x[i]), mt[i + n][1] = 1;
176     // cout << mt[i + n].size() << endl;
177 }
178 for (int i = n - 1; i >= 1; i--)
179     mt[i] = mt[i * 2] * mt[i * 2 + 1];
180 vector<vi> nf(2 * n);
181 nf[1] = f % mt[1];
182 for (int i = 2; i < 2 * n; i++)
183     nf[i] = nf[i / 2] % mt[i];
184 vi y(n);
185 for (int i = 0; i < n; i++)
186     y[i] = nf[i + n][0];
187 return y;
188}
189vi calf(vi x, vi y) {
190 //插值
191 int n = x.size();
192 vector<vi> mt(2 * n);
193 for (int i = 0; i < n; i++) {
194     mt[i + n].resize(2);
195     mt[i + n][0] = sub(0, x[i]), mt[i + n][1] = 1;
196 }
197 for (int i = n - 1; i >= 1; i--)
198     mt[i] = mt[i * 2] * mt[i * 2 + 1];
199 vi u = dw(mt[1]), v = caly(u, x);
200 vector<vi> sm(2 * n);
201 for (int i = 0; i < n; i++)
202     sm[i + n] = vi(1, mul(y[i], power(v[i], mod - 2)));
203 for (int i = n - 1; i >= 1; i--)
204     sm[i] = mt[i * 2 + 1] * sm[i * 2] + mt[i * 2] * sm[i * 2 + 1];
205 return sm[1];
206}
207vi ln(vi x) {
208 // 保证 x[0] = 1
209 int n = x.size();
210 vi t = up(dw(x) * inv(x));
211 t.resize(n);
212 return t;
213}
214vi exp(vi f) {
215 vi x = vi(1, 1);
216 while (x.size() < f.size()) {
217     int er = min(x.size() * 2, f.size());
218     vi q = x, r = vi(f.begin(), f.begin() + er);
219     r[0] = add(r[0], 1);
220     x.resize(er);
221     r = r - ln(x);
222     r = r * q;
223     r.resize(x.size()), x = r;
224 }
225 return x;
226}
227vi ez_pow(vi f, int b) {

```

```

228 f = ln(f);
229 for (int j = 0; j < f.size(); j++)
230     f[j] = mul(f[j], b);
231 return exp(f);
232}
233vi power(vi f, int b) {
234 // b > mod 且常数项不是1 时需要改动 (把int 改成ll ,因为b只能对mod * (mod - 1)
    取模)
235 int pl = f.size();
236 for (int j = 0; j < f.size(); j++)
237     if (f[j]) {
238         pl = j;
239         break;
240     }
241 if (1ll * b * pl >= f.size()) return vi(f.size(), 0);
242 int mv = b * pl, mt = power(f[pl], mod - 2), ca = power(f[pl], b);
243 vi g = vi(f.begin() + pl, f.end());
244 for (int j = 0; j < g.size(); j++)
245     g[j] = mul(g[j], mt);
246 g = ez_pow(g, b);
247 vi fn(f.size());
248 for (int j = 0; j < f.size(); j++)
249     if (j < mv) fn[j] = 0;
250     else fn[j] = mul(ca, g[j - mv]);
251 return fn;
252}
253int main() {
254 return 0;
255}

```

### 30. Havel-Hakimi 定理 [xzl/graph-realization/havel-hakimi.cpp]

给定度数列，要求构造一张简单无向图。设度数为  $d_1, d_2, \dots, d_n$  且按降序排列，则从中任取一个点  $u$ ，向剩下的度数列中按度数从大到小选点连边。如果这个递归过程能顺利结束，则可以构造，否则不能构造。realize 中令  $K = 1$  可以给出另一组解。过程中的排序应该可用双队列优化。

证明 如果不是选取度数最大的，可假设没选  $i$  而选了  $j$ ，且  $d_i > d_j$ 。由于  $i$  的度数比  $j$  大，所以肯定有一个点  $k$ ，满足  $k$  与  $i$  连边却没有和  $j$  连边，因此我们可以把  $k$  改连向  $j$ ，从而空出一个位置给  $i$ 。

Erdős-Gallai 定理 度数列  $d_1 \geq d_2 \geq d_3 \geq \dots \geq d_n$  对应于简单无向图当且仅当：

- $\sum_{k=1}^n d_k$  为偶数；
- $\forall m = 1, 2, \dots, n: \sum_{k=1}^m d_k \leq m(m-1) + \sum_{k=m+1}^n \min\{d_k, m\}$ 。即前缀点的度数之和不能超过它们可连的边的上界。

```

1 #define MULTIPLE 1
2 #define UNIQUE 0
3 #define IMPOSSIBLE -1
4 #define NMAX 100
5 #define MMAX (NMAX * NMAX)
6 static int n, m1, m2;
7 static int d[NMAX + 10], d0[NMAX + 10];
8 static Edge { int u, v; } e1[MMAX + 10], e2[MMAX + 10];
9 int realize(Edge *e, int &m, int K = 0) {
10     static int q1[NMAX + 10], q2[NMAX + 10];

```

```

11 int ret = UNIQUE;
12 for (int i = 1; i <= n; i++) id[i] = i;
13 for (int i = 1; i < n; i++) {
14     int u = id[i];
15     sort(id + i + 1, id + n + 1, [](int x, int y) {
16         return d[x] > d[y];
17     }); // 可以考虑换成双队列实现
18     if (i + d[u] > n) return IMPOSSIBLE;
19     for (int j = i + 1; j <= i + d[u]; j++) {
20         int v = id[j];
21         if (d[v] == 0) return IMPOSSIBLE;
22         if (j == i + d[u] && j < n && d[v] == d[id[j + 1]]) {
23             ret = MULTIPLE;
24             if (K--) v = id[++j];
25         }
26         e[++m] = {u, v};
27         d[v]--;
28     }
29     if (d[id[n]] != 0) return IMPOSSIBLE;
30     return ret;
31 }
32 m1 = m2 = 0;
33 for (int i = 1; i <= n; i++)
34     scanf("%d", d0 + i);
35 memcpy(d, d0, sizeof(d));
36 int ret = realize(e1, m1);
37 if (ret == MULTIPLE) {
38     memcpy(d, d0, sizeof(d));
39     realize(e2, m2, 1);
40 }
41 if (ret == IMPOSSIBLE) puts("IMPOSSIBLE");
42 else {
43     puts(ret == UNIQUE ? "UNIQUE" : "MULTIPLE");
44     print(e1, m1);
45     if (ret == MULTIPLE) print(e2, m2);
46 }

```

### 31. Josephus\_problem [lmj/Josephus\_problem.cpp]

约瑟夫环

可以处理  $n$  个人，报到  $k$  出局，问第  $m$  个人。（从1编号）

复杂度  $O(\min(m, k))$

容易得出  $O(m)$  的递推公式  $f[n][m] = (f[n-1][m-1] + k - 1) \% n + 1$ ，初始状态  $f[n-m+1][1]$  容易得出，当  $m$  小的时候用该公式计算。考虑  $k$  小  $m$  大的情况下，递推式的取膜很多情况下没有用到，可以用乘法代替加法加速递推的过程：

当前状态为  $f[a][b] = c$ ，经过  $x$  次加法后的状态为  $f[a+x][b+x] = c + k * x$ ，假设经过  $x$  次加法之后需要取模，有

$$c + k * x > a + x \rightarrow x > (a - c) / (k - 1)$$

得到该不等式后便可以计算出另一种情况了，还要注意  $k = 1$  需要特判。

递推公式考虑如果直到  $m-1$  的结果，那么在  $f(n, m)$  时让第  $k$  个人出局，然后从第  $k+1$  个人开始重新编号，就是  $f(n-1, m-1)$ ，然后把编号返回去就是在  $f(n-1, m-1)$  编号  $+k$

在k小的时候, 每次跳大概可以跳 $n/k$ 个人, 所以跳  $m/(n/k) = mk/n$  次, 就是 $O(k)$ 次。也可能是 $(1 + 1/k)^x > m$ , 找x这样的取log的感觉。

<https://www.cnblogs.com/scaulok/p/9911819.html>

```
1 long long n, m, k;
2 void work () {
3     long long i, x, t;
4     scanf ( "%lld%lld%lld" , &n , &m , &k );
5     if ( k == 1 ) {
6         printf ( "%lld\n" , m );
7         return ;
8     }
9     if ( m < k ) {
10        x = (k - 1) % (n-m+1) + 1;
11        for ( i = 2 ; i <= m ; i++ ) {
12            x = (x + k - 1) % (n-m+i) + 1;
13        }
14        printf ( "%lld\n" , x );
15    }
16    else {
17        x = (k - 1) % (n-m+1) + 1;
18        i = 1;
19        while ( i < m ) {
20            //printf ( "%lld %lld\n" , i , x );
21            if ( (x+k-1) >= (n-m+i) ) {
22                i++;
23                x = (x + k - 1) % (n-m+i) + 1;
24            }
25            else {
26                t = (n-m+i-1-x)/(k-1);
27                if ( i + t < m ) {
28                    x += t * k;
29                    i += t;
30                }
31                else {
32                    x += (m-i) * k;
33                    i = m;
34                }
35            }
36        }
37        printf ( "%lld\n" , x );
38    }
39 }
```

### 32. dinic [lmj/dinic.cpp]

```
1 void add ( int u , int v , int f ) {
2     node *tmp1 = &pool[++top] , *tmp2 = &pool[++top];
3     tmp1 -> v = v; tmp1 -> f = f; tmp1 -> next = g[u]; g[u] = tmp1; tmp1 ->
    rev = tmp2;
4     tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> next = g[v]; g[v] = tmp2; tmp2 ->
    rev = tmp1;
5 }
6 bool makelevel () {
7     int i, k;
8     queue < int > q;
9     for ( i = 1 ; i <= 1 + n + n + 1 ; i++ ) level[i] = -1;
10    level[1] = 1; q.push ( 1 );
```

```
11    while ( q.size () != 0 ) {
12        k = q.front (); q.pop ();
13        for ( node *j = g[k] ; j ; j = j -> next )
14            if ( j -> f && level[j->v] == -1 ) {
15                level[j->v] = level[k] + 1;
16                q.push ( j -> v );
17                if ( j -> v == 1 + n + n + 1 ) return true;
18            }
19        return false;
20    }
21    int find ( int k , int key ) {
22        if ( k == 1 + n + n + 1 ) return key;
23        int i, s = 0;
24        for ( node *j = g[k] ; j ; j = j -> next )
25            if ( j -> f && level[j->v] == level[k] + 1 && s < key ) {
26                i = find ( j -> v , min ( key - s , j -> f ) );
27                j -> f -= i;
28                j -> rev -> f += i;
29                s += i;
30            }
31        if ( s == 0 ) level[k] = -1;
32        return s;
33    }
34    void dinic () {
35        int ans = 0;
36        while ( makelevel () == true ) ans += find ( 1 , 99999 );
37        //printf ( "%d\n" , ans );
38        if ( ans == sum ) printf ( "^_^\n" );
39        else printf ( "T_T\n" );
40    }
```

### 33. 费用流 [lmj/min\_cost\_max\_flow.cpp]

```
1 void add ( int u , int v , int f , int c ) {
2     node *tmp1 = &pool[++top] , *tmp2 = &pool[++top];
3     tmp1 -> v = v; tmp1 -> f = f; tmp1 -> c = c; tmp1 -> next = g[u]; g[u]
    = tmp1; tmp1 -> rev = tmp2;
4     tmp2 -> v = u; tmp2 -> f = 0; tmp2 -> c = -c; tmp2 -> next = g[v]; g[v]
    = tmp2; tmp2 -> rev = tmp1;
5 }
6 bool spfa () {
7     int i, k;
8     queue < int > q;
9     for ( i = 1 ; i <= 1 + n*m*3 + 1 ; i++ ) dis[i] = 9999999, f[i] = 0;
10    dis[1] = 0; f[1] = 1; q.push ( 1 );
11    while ( q.size () != 0 ) {
12        k = q.front (); q.pop (); f[k] = 0;
13        for ( node *j = g[k] ; j ; j = j -> next )
14            if ( j -> f && dis[j->v] > dis[k] + j -> c ) {
15                dis[j->v] = dis[k] + j -> c;
16                from[j->v] = k;
17                if ( f[j->v] == 0 ) q.push ( j -> v );
18                f[j->v] = 1;
19            }
20    }
21    if ( dis[1+n*m*3+1] != 9999999 ) return true;
22    return false;
23 }
24 int find () {
```

```

25  int i , f = 999999 , s = 0;
26  for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] -> rev -> v ) f = min ( f ,
    from[i] -> f );
27  flow += f;
28  for ( i = 1+n*m*3+1 ; i != 1 ; i = from[i] -> rev -> v ) from[i] -> f -
    = f, from[i] -> rev -> f += f;
29  return f * dis[1+n*m*3+1];
30 }
31 void dinic () {
32  int ans = 0;
33  while ( spfa () == true ) ans += find ();
34  //printf ( "%d\n" , flow );
35  if ( flow == sum && sum == sum1 ) printf ( "%d\n" , ans );
36  else printf ( "-1\n" );
37 }

```

#### 34. Xorshift [xzl/Xorshift.cpp]

```

1  inline u32 mrand32() {
2  static u32 x = 19260817;
3  x ^= x << 13;
4  x ^= x >> 17;
5  x ^= x << 5;
6  return x;
7 }
8  inline u64 mrand64() {
9  static u64 x = 0x19260817deedbeef;
10 x ^= x << 13;
11 x ^= x >> 7;
12 x ^= x << 17;
13 return x;
14 }

```

#### 35. 三分\_上凸函数 [sll/三分\_上凸函数.cpp]

```

1  double solve() {
2  while(l+eps<r) {
3      double mid=(l+r)/2.0;
4      double mmid=(mid+r)/2.0;
5      if(cal(mid)>cal(mmid))r=mmid;
6      else l=mid;
7  }
8  if(cal(l)<cal(r))return r;
9  else return l;
10 }

```

#### 36. 单纯型 [xzl/simplex.cpp]

```

1  #define EPS 1e-10
2  #define INF 1e100
3
4  class Simplex {
5  public:
6  void initialize() {
7      scanf("%d%d%d", &n, &m, &t);
8      memset(A, 0, sizeof(A));
9      for (int i = 1; i <= n; i++) {
10         idx[i] = i;
11         scanf("%Lf", A[0] + i);
12     }

```

```

13     for (int i = 1; i <= m; i++) {
14         idy[i] = n + i;
15         for (int j = 1; j <= n; j++) {
16             scanf("%Lf", A[i] + j);
17             A[i][j] *= -1;
18         }
19         scanf("%Lf", A[i]);
20     }
21 }
22 void solve() {
23     srand(time(0));
24     while (true) {
25         int x = 0, y = 0;
26         for (int i = 1; i <= m; i++)
27             if (A[i][0] < -EPS && (!y || (rand() & 1))) y = i;
28         if (!y) break;
29         for (int i = 1; i <= n; i++)
30             if (A[y][i] > EPS && (!x || (rand() & 1))) x = i;
31         if (!x) {
32             puts("Infeasible");
33             return;
34         }
35         pivot(x, y);
36     }
37     while (true) {
38         double k = INF;
39         int x, y;
40         for (x = 1; x <= n; x++)
41             if (A[0][x] > EPS) break;
42         if (x > n) break;
43         for (int i = 1; i <= m; i++) {
44             double d = A[i][x] > -EPS ? INF : -A[i][0] / A[i][x];
45             if (d < k) {
46                 k = d;
47                 y = i;
48             }
49         }
50         if (k >= INF) {
51             puts("Unbounded");
52             return;
53         }
54         pivot(x, y);
55     }
56     printf("%.10Lf\n", A[0][0]);
57     if (t) {
58         static double ans[NMAX + 10];
59         for (int i = 1; i <= m; i++)
60             if (idy[i] <= n) ans[idy[i]] = A[i][0];
61         for (int i = 1; i <= n; i++)
62             printf("%.10Lf ", ans[i]);
63         printf("\n");
64     }
65 }
66 private:
67 void pivot(int x, int y) {
68     swap(idx[x], idy[y]);
69     double r = -A[y][x];
70     A[y][x] = -1;
71     for (int i = 0; i <= n; i++) A[y][i] /= r;
72     for (int i = 0; i <= m; i++) {
73         if (i == y) continue;

```

```

71     r = A[i][x];
72     A[i][x] = 0;
73     for (int j = 0; j <= n; j++)
74         A[i][j] += r * A[y][j];
75 }
76 int n, m, t;
77 double A[NMAX + 10][NMAX + 10];
78 int idx[NMAX + 10], idy[NMAX + 10];
79 };

```

### 37. 线性空间求交 [xzl/vector-space-intersect.cpp]

设两个线性空间  $U$ 、 $V$  的基分别为  $u_1, u_2, \dots, u_n$  和  $v_1, v_2, \dots, v_m$ 。考虑同时求出  $U + V$  和  $U \cap V$  的基：逐次将  $u_i$  加入。设当前扩展到  $v_1, \dots, v_m, u'_1, \dots, u'_j$ ，若  $u_i$  不能被它们线性表出，则令  $u'_{j+1} = u_i$ 。否则  $u_i = \sum a_j u'_j + \sum b_j v_j$ ，即  $u_i - \sum a_j u'_j = \sum b_j v_j$ ，那么等式左边可以直接加入交空间。时间复杂度  $\Theta(nm)$ 。代码是异或线性空间的求交。

```

1 #define SMAX 32
2 typedef unsigned int u32;
3 struct Basis {
4     u32 v[SMAX];
5     auto operator[](const size_t i) -> u32& {
6         return v[i];
7     };
8     auto intersect(Basis &u, Basis v) -> Basis {
9         Basis z, r;
10        for (int i = 0; i < SMAX; i++) if (u[i]) {
11            u32 x = u[i], y = u[i];
12            for (int j = 0; j < SMAX; j++) if ((x >> j) & 1) {
13                if (v[j] x ^= v[j], y ^= r[j];
14                else {
15                    v[j] = x, r[j] = y;
16                    break;
17                }
18            } if (!x) z.add(y);
19        } return z;
20 }

```

### 38. AC 自动机 [xzl/ac-automaton.cpp]

时间复杂度  $O(n + m + z + n|\Sigma|)$ ， $n$  是模板串总长度， $m$  是目标串长度， $z$  是总匹配次数， $\Sigma$  是字符集。如果想移掉  $n|\Sigma|$  这一项，需要使用哈希表。传入的字符串下标从 0 开始。

```

1 struct Node {
2     Node() : mark(false), suf(NULL), nxt(NULL) {
3         memset(ch, 0, sizeof(ch));
4     }
5     bool mark;
6     Node *suf, *nxt, *ch[SIGMA];
7 };
8 void insert(Node *x, char *s) {
9     for (int i = 0; s[i]; i++) {
10        int c = s[i] - 'a';
11        if (!x->ch[c]) x->ch[c] = new Node;
12        x = x->ch[c];
13    }
14    x->mark = true;
15 }

```

```

16 void build_automaton(Node *r) {
17     queue<Node *> q;
18     for (int c = 0; c < SIGMA; c++) {
19         if (!r->ch[c]) continue;
20         r->ch[c]->suf = r;
21         q.push(r->ch[c]);
22     }
23     while (!q.empty()) {
24         Node *x = q.front();
25         q.pop();
26         for (int c = 0; c < SIGMA; c++) {
27             Node *v = x->ch[c]; if (!v) continue;
28             Node *y = x->suf;
29             while (y != r && !y->ch[c]) y = y->suf;
30             if (y->ch[c]) y = y->ch[c];
31             v->suf = y;
32             if (y->mark) v->nxt = y;
33             else v->nxt = y->nxt;
34             q.push(v);
35         }
36     }
37     void search(Node *x, char *s) {
38         for (int i = 0; s[i]; i++) {
39             int c = s[i] - 'a';
40             while (x->suf && !x->ch[c]) x = x->suf;
41             if (x->ch[c]) x = x->ch[c];
42             if (x->mark) print(i + 1, x->data);
43             for (Node *y = x->nxt; y; y = y->nxt) print(i + 1, y->data);
44         }
45     }

```

### 39. KMP [sll/KMP.cpp]

```

1 int p[101];
2 int main() {
3     string a, b;
4     cin >> a >> b;
5     int n = a.length(), m = b.length();
6     a = " " + a; b = " " + b;
7     int j = 0;
8     for (int i = 2; i <= m; i++) {
9         while (j > 0 && b[j+1] != b[i]) j = p[j];
10        if (b[j+1] == b[i]) j++;
11        p[i] = j;
12    }
13    j = 0;
14    for (int i = 1; i <= n; i++) {
15        while (j > 0 && b[j+1] != a[i]) j = p[j];
16        if (b[j+1] == a[i]) j++;
17        if (j == m) { printf("%d", i - m + 1); break; }
18    }
19    return 0;
20 }

```

### 40. PAM [sll/PAM, 字符串.cpp]

```

1 #define N 500020
2 int val[N], head[N], pos;
3 struct edge { int to, next; } e[N << 1];
4 void add(int a, int b) { pos++; e[pos].to = b, e[pos].next = head[a], head[a] = pos; }
5 struct Tree {

```

```

6  char ch[N];
7  int now,cnt,odd,even;
8  int fail[N],len[N],go[N][26];
9  void init() {
10     now=cnt=0;
11     odd=++cnt,even=++cnt;
12     len[odd]=-1,len[even]=0;
13     fail[odd]=fail[even]=odd;
14     now=even;add(odd,even);
15 }
16 void insert(int pos,char c) {
17     while(ch[pos-1-len[now]]!=c)now=fail[now];
18     if(!go[now][c-'a']){
19         go[now][c-'a']=++cnt;
20         len[cnt]=len[now]+2;
21         if(now==odd)fail[cnt]=even;
22         else {
23             int t=fail[now];
24             while(ch[pos-1-len[t]]!=c)t=fail[t];
25             fail[cnt]=go[t][c-'a'];
26         }
27         add(fail[cnt],cnt);
28     }
29     now=go[now][c-'a'];
30     val[now]++;
31 }
32 void dfs(int u) {
33     for(int i=head[u];i;i=e[i].next) {
34         int v=e[i].to;
35         dfs(v);
36         val[u]+=val[v];
37     }
38 }
39 Long Long cal() {
40     Long Long ret=0;
41     for(int i=3;i<=cnt;i++){
42         ret=max(ret,1ll*len[i]*val[i]);
43     }
44     return ret;
45 }
46 int main() {
47     tree.init();
48     scanf("%s",tree.ch+1);
49     int len=strlen(tree.ch+1);
50     for(int i=1;i<=len;i++){
51         tree.insert(i,tree.ch[i]);
52     }
53     tree.dfs(1);
54     printf("%lld\n",tree.cal());
55 }

```

#### 41. SA [sll/SA, 字符串.cpp]

```

1 #define N 200020
2 int wa[N],wb[N],ws[N],wv[N],sa[N],rank[N];
3 void cal_sa(int *r,int n,int m) {
4     int *x=wa,*y=wb,*t;
5     for(int i=0;i<m;i++)ws[i]=0;
6     for(int i=0;i<n;i++)ws[x[i]=r[i]]++;
7     for(int i=1;i<m;i++)ws[i]+=ws[i-1];
8     for(int i=n-1;i>=0;i--)sa[--ws[x[i]]]=i;
9     for(int j=1,p=1;p<n;j<=1,m=p) {

```

```

10     p=0;
11     for(int i=n-j;i<n;i++)y[p++]=i;
12     for(int i=0;i<n;i++)if(sa[i]>=j)y[p++]=sa[i]-j;
13     for(int i=0;i<n;i++)wv[i]=x[y[i]];
14     for(int i=0;i<m;i++)ws[i]=0;
15     for(int i=0;i<n;i++)ws[wv[i]]++;
16     for(int i=1;i<m;i++)ws[i]+=ws[i-1];
17     for(int i=n-1;i>=0;i--)sa[--ws[wv[i]]]=y[i];
18     t=x,x=y,y=t,p=1;x[sa[0]]=0;
19     for(int i=1;i<n;i++){
20         x[sa[i]]=(y[sa[i-1]]==y[sa[i]]&&y[sa[i-1]+j]==y[sa[i]+j])?p-1:p++;
21     }
22     int height[N];
23     void cal_h(int *r,int *sa,int n) {
24         int k=0;
25         for(int i=1;i<=n;i++)rank[sa[i]]=i;
26         for(int i=0;i<n;i++){
27             int j=sa[rank[i]-1];if(k>0)k--;
28             while(r[j+k]==r[i+k])k++;
29             height[rank[i]]=k;
30         }
31     }
32     char ch[N]; int r[N];
33     int main() {
34         std::cin>>ch;
35         int n=strlen(ch);
36         for(int i=0;i<n;i++)r[i]=ch[i];r[n]=0;
37         cal_sa(r,n+1,128);
38         cal_h(r,sa,n);
39         for(int i=1;i<=n;i++)printf("%d ",sa[i]+1);puts("");
40         for(int i=2;i<=n;i++)printf("%d ",height[i]);
41     }

```

#### 42. manacher [sll/Manacher.cpp]

```

1 void manacher() {
2     //max(p[i])-1即为最大回文子串长
3     int mx=0,id=0;n=strlen(ch);
4     for(int i=n;i;i--)ch[i]=ch[i-1];
5     for(int i=1;i<=n;i++)c[i<<1]=ch[i],c[i<<1|1]='#';
6     m=n<<1|1;c[0]='-',c[1]='#',c[m+1]='+';
7     for(int i=1;i<=m;i++){
8         if(mx>i)p[i]=min(p[2*id-i],mx-i);
9         while(c[p[i]+i]==c[i-p[i]])p[i]++;
10        if(i+p[i]>mx)mx=i+p[i],id=i;
11    }

```

#### 43. palindrome\_partition [lmj/palindrome\_partition.cpp]

作者: XLor

#### 问题

给定一个字符串  $s$  ( $1 \leq |s| \leq 10^5$ ), 求最小的  $k$ , 使得存在  $s_1, s_2, \dots, s_k$ , 满足  $s_i$  ( $1 \leq i \leq k$ ) 均为回文串, 且  $s_1, s_2, \dots, s_k$  依次连接后得到的字符串等于  $s$ 。

#### 暴力做法

考虑动态规划, 记  $dp[i]$  表示  $s$  长度为  $i$  的前缀的最小划分数, 转移只需要枚举以第  $i$  个字符结尾的所有回文串

由于一个字符串最多会有  $O(n^2)$  个回文子串, 因此上述算法的时间复杂度为  $O(n^2)$ 。



## 引理与证明

### 定义

由于一个字符串最多会有  $O(n^2)$  个回文子串，因此上述算法的时间复杂度为  $O(n^2)$ ，无法接受，为了优化转移过程，下面给出一些引理。

记字符串  $s$  长度为  $i$  的前缀为  $pre(s, i)$ ，长度为  $i$  的后缀为  $suf(s, i)$ 。

周期：若  $0 < p \leq |s|$ ， $\forall 1 \leq i \leq |s| - p, s[i] = s[i + p]$ ，就称  $p$  是  $s$  的周期。

border：若  $0 \leq r < |s|$ ， $pre(s, r) = suf(s, r)$ ，就称  $pre(s, r)$  是  $s$  的 border。

### 周期和 border 的关系

$t$  是  $s$  的 border，当且仅当  $|s| - |t|$  是  $s$  的周期。

证明：

若  $t$  是  $s$  的 border，那么  $pre(s, |t|) = suf(s, |t|)$ ，因此  $\forall 1 \leq i \leq |t|, s[i] = s[|s| - |t| + i]$ ，所以  $|s| - |t|$  就是  $s$  的周期。

若  $|s| - |t|$  为  $s$  周期，则  $\forall 1 \leq i \leq |s| - (|s| - |t|) = |t|, s[i] = s[|s| - |t| + i]$ ，因此  $pre(s, |t|) = suf(s, |t|)$ ，所以  $t$  是  $s$  的 border。

### 引理 1

$t$  是回文串  $s$  的后缀， $t$  是  $s$  的 border 当且仅当  $t$  是回文串。

证明：

对于  $1 \leq i \leq |t|$ ，由  $s$  和  $t$  为回文串，因此有  $s[i] = s[|s| - i + 1] = s[|s| - |t| + i]$ ，所以  $t$  是  $s$  的 border。

对于  $1 \leq i \leq |t|$ ，由  $t$  是  $s$  的 border，有  $s[i] = s[|s| - |t| + i]$ ，由  $s$  是回文串，有  $s[i] = s[|s| - i + 1]$ ，因此  $s[|s| - i + 1] = s[|s| - |t| + i]$ ，所以  $t$  是回文串。

下图中，相同颜色的位置表示字符对应相同。



### 引理 2

$t$  是回文串  $s$  的 border ( $|s| \leq 2|t|$ )， $s$  是回文串当且仅当  $t$  是回文串。

证明：

若  $s$  是回文串，由引理 1， $t$  也是回文串。

若  $t$  是回文串，由  $t$  是  $s$  的 border，因此  $\forall 1 \leq i \leq |t|, s[i] = s[|s| - |t| + i] = s[|s| - i + 1]$ ，因为  $|s| \leq 2|t|$ ，所以  $s$  也是回文串。

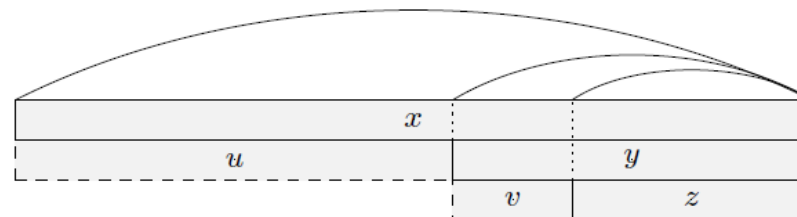
### 引理 3

$t$  是字符串  $s$  的 border，则  $|s| - |t|$  是  $s$  的周期， $|s| - |t|$  为  $s$  的最小周期，当且仅当  $t$  是  $s$  的最长回文真后缀。

## 引理 4

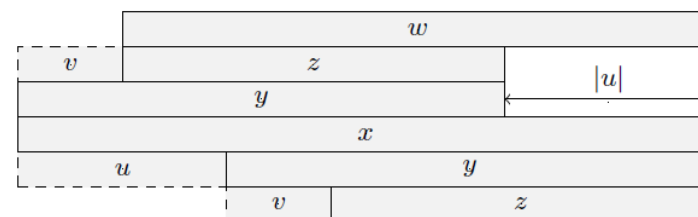
$x$  是一个回文串， $y$  是  $x$  的最长回文真后缀， $z$  是  $y$  的最长回文真后缀。令  $u, v$  分别为满足  $x = uy, y = vz$  的字符串，则有以下三条性质

- $|u| \geq |v|$ ；
- 如果  $|u| > |v|$ ，那么  $|u| > |z|$ ；
- 如果  $|u| = |v|$ ，那么  $u = v$ 。



证明：

- 由引理 3 的推论， $|u| = |x| - |y|$  是  $x$  的最小周期， $|v| = |y| - |z|$  是  $y$  的最小周期。考虑反证法，假设  $|u| < |v|$ ，因为  $y$  是  $x$  的后缀，所以  $u$  既是  $x$  的周期，也是  $y$  的周期，而  $|v|$  是  $y$  的最小周期，矛盾。所以  $|u| \geq |v|$ 。
- 因为  $y$  是  $x$  的 border，所以  $v$  是  $x$  的前缀，设字符串  $w$ ，满足  $x = vw$ （如下图所示），其中  $z$  是  $w$  的 border。考虑反证法，假设  $|u| \leq |z|$ ，那么  $|zu| \leq 2|z|$ ，所以由引理 2， $w$  是回文串，由引理 1， $w$  是  $x$  的 border，又因为  $|u| > |v|$ ，所以  $|w| > |y|$ ，矛盾。所以  $|u| > |z|$ 。
- $u, v$  都是  $x$  的前缀， $|u| = |v|$ ，所以  $u = v$ 。



## 推论

$s$  的所有回文后缀按照长度排序后，可以划分成  $\log |s|$  段等差数列。

证明：

设  $s$  的所有回文后缀长度从小到大排序为  $l_1, l_2, \dots, l_k$ 。对于任意  $2 \leq i \leq k - 1$ ，若  $l_i - l_{i-1} = l_{i+1} - l_i$ ，则  $l_{i-1}, l_i, l_{i+1}$  构成一个等差数列。否则  $l_i - l_{i-1} \neq l_{i+1} - l_i$ ，由引理 4，有  $l_{i+1} - l_i > l_i - l_{i-1}$ ，且  $l_{i+1} - l_i > l_{i-1}$ ， $l_{i+1} > 2l_{i-1}$ 。因此，若相邻两对回文后缀的长度之差

发生变化，那么这个最大长度一定会相对于最小长度翻一倍。显然，长度翻倍最多只会发生  $O(\log |s|)$  次，也就是  $s$  的回文后缀长度可以划分成  $\log |s|$  段等差数列。

该推论也可以通过使用弱周期引理，对  $s$  的最长回文后缀的所有 border 按照长度  $x$  分类， $x \in [2^0, 2^1), [2^1, 2^2), \dots, [2^k, n)$ ，考虑这  $\log |s|$  组内每组的最长 border 进行证明。详细证明可以参考金策的《字符串算法选讲》和陈孙立的 2019 年 IOI 国家候选队论文《子串周期查询问题的相关算法及其应用》。

## 做法

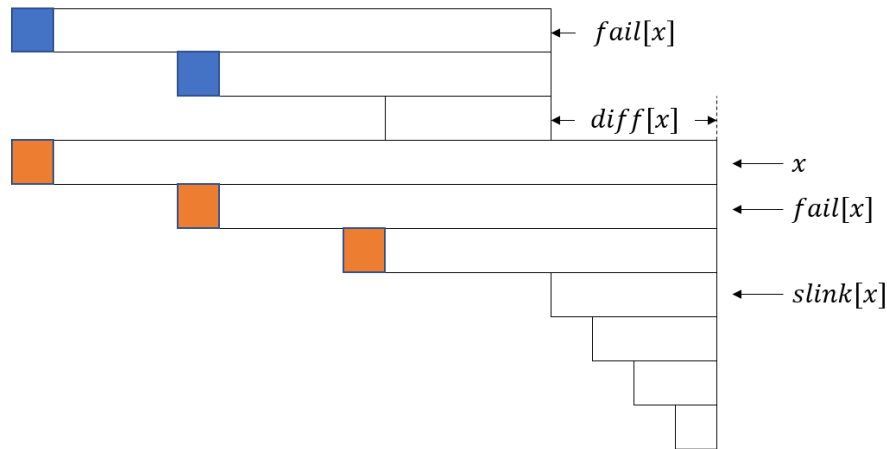
有了上述结论后，我们现在可以考虑如何优化  $dp$  的转移。

回文树上的每个节点  $u$  需要多维护两个信息， $diff[u]$  和  $slink[u]$ 。 $diff[u]$  表示节点  $u$  和  $fail[u]$  所代表的回文串的长度差，即  $len[u] - len[fail[u]]$ 。 $slink[u]$  表示  $u$  一直沿着 fail 向上跳到第一个节点  $v$ ，使得  $diff[v] \neq diff[u]$ ，也就是  $u$  所在等差数列中长度最小的那个节点。

根据上面证明的结论，如果使用  $slink$  指针向上跳的话，每向后添加一个字符，只需要向上跳  $O(\log |s|)$  次。因此，可以考虑将一个等差数列表示的所有回文串的  $dp$  值之和（在原问题中指  $\min$ ），记录到最长的那一个回文串对应节点上。

$g[v]$  表示  $v$  所在等差数列的  $dp$  值之和，且  $v$  是这个等差数列中长度最长的节点，则  $g[v] = \sum_{slink[x]=v} dp[i - len[x]]$ 。

下面我们考虑如何更新  $g$  数组和  $dp$  数组。以下图为例，假设当前枚举到第  $i$  个字符，回文树上对应节点为  $x$ 。 $g[x]$  为橙色三个位置的  $dp$  值之和（最短的回文串  $slink[x]$  算在下一个等差数列中）。 $fail[x]$  上一次出现位置是  $i - diff[x]$ （在  $i - diff[x]$  处结束）， $g[fail[x]]$  包含的  $dp$  值是蓝色位置。因此， $g[x]$  实际上等于  $g[fail[x]]$  和多出来一个位置的  $dp$  值之和，多出来的位置是  $i - (slink[x] + diff[x])$ 。最后再用  $g[x]$  去更新  $dp[i]$ ，这部分等差数列的贡献就计算完毕了，不断跳  $slink[x]$ ，重复这个过程即可。具体实现方式可参考例题代码。



最后，上述做法的正确性依赖于：如果  $x$  和  $fail[x]$  属于同一个等差数列，那么  $fail[x]$  上一次出现位置是  $i - diff[x]$ 。

证明：

根据引理 1， $fail[x]$  是  $x$  的 border，因此其在  $i - diff[x]$  处出现。

假设  $fail[x]$  在  $(i - diff[x], i)$  中的  $j$  位置出现。由于  $x$  和  $fail[x]$  属于同一个等差数列，因此  $2|fail[x]| \geq x$ 。多余的  $fail[x]$  和  $i - diff[x]$  处的  $fail[x]$  有交集，记交集为  $w$ ，设串  $u$  满足  $uw = fail[x]$ 。用类似引理 1 的方式可以证明， $w$  是回文串，而  $x$  的前缀  $s[i - len[x] + 1..j] = wuw$  也是回文串，这与  $fail[x]$  是  $x$  的最长回文前缀（后缀）矛盾。

## Codeforces 932G Palindrome Partition

构造字符串  $t = s[0]s[n-1]s[1]s[n-2]s[2]s[n-3] \dots s[n/2-1]s[n/2]$ ，问题等价于求  $t$  的偶回文划分方案数，把上面的转移方程改成求和形式并且只在偶数位置更新  $dp$  数组即可。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 const int mod = 1e9 + 7;
5 const int maxn = 1000000 + 5;
6
7 inline int add(int x, int y) {
8     x += y;
9     return x >= mod ? x - mod : x;
10 }
11
12 namespace pam {
13     int sz, tot, last;
14     int ch[maxn][26], len[maxn], fail[maxn];
15     int cnt[maxn], dep[maxn], dif[maxn], slink[maxn];
16     char s[maxn];
17     int node(int l) {
18         sz++;
19         memset(ch[sz], 0, sizeof(ch[sz]));
20         len[sz] = l;
21         fail[sz] = 0;
22         cnt[sz] = 0;
23         dep[sz] = 0;
24         return sz;
25     }
26     void clear() {
27         sz = -1; last = 0;
28         s[tot = 0] = '$';
29         node(0); node(-1);
30         fail[0] = 1;
31     }
32     int getfail(int x) {
33         while (s[tot - len[x] - 1] != s[tot]) x = fail[x];
34         return x;
35     }
36     void insert(char c) {
37         s[++tot] = c;
38         int now = getfail(last);
39         if (!ch[now][c - 'a']) {
40             int x = node(len[now] + 2);
41             fail[x] = ch[getfail(fail[now])][c - 'a'];
```

```

42     dep[x] = dep[fail[x]] + 1;
43     ch[now][c - 'a'] = x;
44
45     dif[x] = len[x] - len[fail[x]];
46     if (dif[x] == dif[fail[x]]) slink[x] = slink[fail[x]];
47     else slink[x] = fail[x];
48 }
49 last = ch[now][c - 'a'];
50 cnt[last]++;
51 }
52 }
53 using pam::len;
54 using pam::fail;
55 using pam::slink;
56 using pam::dif;
57
58 int n, dp[maxn], g[maxn]; char s[maxn], t[maxn];
59
60 int main() {
61     pam::clear();
62     scanf("%s", s + 1);
63     n = strlen(s + 1);
64     for (int i = 1, j = 0; i <= n; i++) t[++j] = s[i], t[++j] = s[n - i + 1];
65     dp[0] = 1;
66     for (int i = 1; i <= n; i++) {
67         pam::insert(t[i]);
68         for (int x = pam::last; x > 1; x = slink[x]) {
69             g[x] = dp[i - len[slink[x]] - dif[x]];
70             if (dif[x] == dif[fail[x]]) g[x] = add(g[x], g[fail[x]]);
71             if (i % 2 == 0) dp[i] = add(dp[i], g[x]);
72         }
73     }
74     printf("%d", dp[n]);
75     return 0;
76 }

```

#### 44. pam [lmj/pam.cpp]

```

1 const int NN = 310000;
2 struct node {
3     int len, cnt, ch[30], fail;
4 } p[NN];
5 int top, n, last;
6 char z[NN];
7 long long ans;
8 void work () {
9     int i, tmp;
10    scanf ("%s", z + 1);
11    n = strlen (z + 1);
12    top = 2;
13    p[1].fail = 2; p[2].fail = 1;
14    p[1].len = 0; p[2].len = -1;
15    z[0] = '$';
16    last = 1;
17    for (i = 1; i <= n; i++) {
18        while (z[i] != z[i - p[last].len - 1]) last = p[last].fail;
19        if (!p[last].ch[z[i] - 'a' + 1]) {
20            p[last].ch[z[i] - 'a' + 1] = ++top;

```

```

21        p[top].len = p[last].len + 2;
22        tmp = p[last].fail;
23        while (z[i] != z[i - p[tmp].len - 1]) tmp = p[tmp].fail;
24        if (p[top].len > 1 && p[tmp].ch[z[i] - 'a' + 1]) p[top].fail = p[tmp];
25        else p[top].fail = 1;
26    }
27    last = p[last].ch[z[i] - 'a' + 1];
28    p[last].cnt++;
29 }
30 for (i = top; i >= 1; i--) p[p[i].fail].cnt += p[i].cnt;
31 for (i = 1; i <= top; i++) {
32     //printf ("%d %d\n", p[i].len, p[i].cnt);
33     ans = max (ans, (long long)p[i].len * p[i].cnt);
34 }
35 printf ("%lld\n", ans);
36 }

```

#### 45. 回文自动机 [sll/回文自动机.cpp]

```

1 int val[N];
2 int head[N], pos;
3 struct edge {int to, next; } e[N << 1];
4 void add(int a, int b)
5 {pos++; e[pos].to = b, e[pos].next = head[a], head[a] = pos;}
6 struct Tree {
7     char ch[N];
8     int now, cnt, odd, even;
9     int fail[N], len[N], go[N][26];
10    void init() {
11        now = cnt = 0;
12        odd = ++cnt, even = ++cnt;
13        len[odd] = -1, len[even] = 0;
14        fail[odd] = fail[even] = odd;
15        now = even; add(odd, even);
16    }
17    void insert(int pos, char c) {
18        while (ch[pos - 1 - len[now]] != c) now = fail[now];
19        if (!go[now][c - 'a']) {
20            go[now][c - 'a'] = ++cnt;
21            len[cnt] = len[now] + 2;
22            if (now == odd) fail[cnt] = even;
23            else {
24                int t = fail[now];
25                while (ch[pos - 1 - len[t]] != c) t = fail[t];
26                fail[cnt] = go[t][c - 'a'];
27            }
28            add(fail[cnt], cnt);
29        }
30        now = go[now][c - 'a'];
31        val[now]++;
32    }
33    void dfs(int u) {
34        for (int i = head[u]; i; i = e[i].next) {
35            int v = e[i].to;
36            dfs(v);
37            val[u] += val[v];
38        }
39        long long cal() {

```

```

40  Long Long ret=0;
41  for(int i=3;i<=cnt;i++)
42      ret=max(ret,1ll*len[i]*val[i]);
43  return ret;
44  }
45 }tree;

```

#### 46. 树上后缀数组 [xzl/sa-on-tree.cpp]

对树上每个节点到根的路径上所有边上的字符依次接成的字符串排序。倍增算法，时空复杂度均为  $\Theta(n \log n)$ 。字符视为存在边上，0 号节点为根节点，默认表示空字符“\0”。各种数组的下标从 0 开始，但是由于第一个总是节点 0，因此实际上也是从 1 开始的。fa 是倍增跳表，rk 是倍增 rank 数组，可以当 Hash 用。注意 lcp 如果传入的两个后缀相同，会返回一个极大值，因此计算 h 时要和节点深度取最小值。suffix\_sort 传入的参数是字符集大小，字符集为  $\{0, 1, 2, \dots, m\}$ 。

```

1 #define NMAX 300000
2 #define LOGN 20
3 static int n;
4 static char s[NMAX + 10];
5 static int fa[LOGN][NMAX + 10];
6 static int rk[LOGN][NMAX + 10];
7 static int sa[NMAX + 10], h[NMAX + 10];
8 void suffix_sort(int m, int i = 0) {
9     static int cnt[NMAX + 10], y[NMAX + 10], b[NMAX + 10];
10 #define SORT(_y) \
11     memset(cnt, 0, sizeof(int) * (m + 1)); \
12     for (i = 0; i <= n; i++) cnt[_y[i]]++; \
13     for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1]; \
14     for (i = n; i >= 0; i--) b[--cnt[_y[sa[i]]]] = sa[i]; \
15     memcpy(sa, b, sizeof(int) * (n + 1));
16 #define RANK(_rk, _x, _y) \
17     for (i = 0, m = 0; i <= n; i++) { \
18         m += i && (_y[sa[i]] != _y[sa[i - 1]] || _x[sa[i]] != _x[sa[i - 1]]); \
19         _rk[sa[i]] = m; \
20     }
21     for (i = 0; i <= n; i++) sa[i] = i;
22     SORT(s);
23     RANK(rk[0], s, s);
24     for (int k = 0; k < LOGN - 1; k++) {
25         for (i = 0; i <= n; i++) fa[k + 1][i] = fa[k][fa[k][i]];
26         for (i = 0; i <= n; i++) y[i] = rk[k][fa[k][i]];
27         SORT(y);
28         SORT(rk[k]);
29         RANK(rk[k + 1], rk[k], y);
30     }
31 int lcp(int x, int y, int r = 0) {
32     for (int k = LOGN - 1; k >= 0; k--)
33         if (rk[k][x] == rk[k][y]) {
34             x = fa[k][x];
35             y = fa[k][y];
36             r += 1 << k;
37         }
38     return r;
39 }
40 int main() {
41     //s[0] = 0;

```

```

41 //fa[0][0] = fa[0][1] = 0;
42 for (int v = 2; v <= n; v++) {
43     scanf("%d", &fa[0][v]);
44     dep[v] = dep[fa[0][v]] + 1;
45 }
46 suffix_sort(255);
47 for (int i = 1; i <= n; i++)
48     h[i] = min(dep[sa[i]], lcp(sa[i - 1], sa[i]));
49 }

```

#### 47. 后缀排序：倍增算法 [xzl/sa-nlogn.cpp]

倍增法后缀排序，时间复杂度为  $\Theta(n \log n)$ 。suffix\_sort 是本体，结果输出到 sa 数组和 rk 数组（排名数组）。参数 s 是字符串，下标从 0 开始，n 是字符串长度（包括末尾添加的保留字符 \$），m 是字符集大小（一般为 255，字符集为  $\Sigma = \{0, 1, 2, \dots, m\}$ ，0 是保留的 \$ 字符）。算法运行完毕后 sa 数组里面存的是从 0 开始的下标，rk 数组里面存的是从 1 开始的排名值，两个数组均从 0 开始索引。如果要多次使用请注意清空 cnt 数组。

另外附带一个线性求 lcp 数组的代码。lcp 数组下标从 1 开始，实际上只有在 2 到 n 范围内的才是有效值。参数意义与 suffix\_sort 相同。

```

1 static int sa[NMAX + 10], rk[NMAX + 10], lcp[NMAX + 10];
2 void suffix_sort(const char *s, int n, int m) {
3     static int x[NMAX + 10], y[NMAX + 10], cnt[NMAX + 10], i;
4     //memset(cnt, 0, sizeof(int) * (m + 1));
5     for (i = 0; i < n; i++) cnt[s[i]]++;
6     for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
7     for (i = 0; i < n; i++) sa[--cnt[s[i]]] = i;
8     for (i = 1, m = 1, rk[sa[0]] = 1; i < n; i++) {
9         if (s[sa[i - 1]] != s[sa[i]]) m++;
10        rk[sa[i]] = m;
11    }
12    for (int l = 1; l < n; l <= 1) {
13        memset(cnt, 0, sizeof(int) * (m + 1));
14        for (i = 0; i < n; i++) cnt[y[i] = i + l < n ? rk[i + l] : 0]++;
15        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
16        for (i = n - 1; i >= 0; i--) x[--cnt[y[i]]] = i;
17        memset(cnt, 0, sizeof(int) * (m + 1));
18        for (i = 0; i < n; i++) cnt[rk[i]]++;
19        for (i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
20        for (i = n - 1; i >= 0; i--) sa[--cnt[rk[x[i]]]] = x[i];
21        for (i = 1, m = 1, x[sa[0]] = 1; i < n; i++) {
22            if (rk[sa[i - 1]] != rk[sa[i]] || y[sa[i - 1]] != y[sa[i]]) m++;
23            x[sa[i]] = m;
24        }
25        memcpy(rk, x, sizeof(int) * n);
26    }
27 void compute_lcp(const char *s, int n) {
28     int j = 0, p;
29     for (int i = 0; i < n; i++, j = max(0, j - 1)) {
30         if (rk[i] == 1) {
31             j = 0;
32             continue;
33         }
34         p = sa[rk[i] - 2];
35         while (p + j < n && i + j < n && s[p + j] == s[i + j]) j++;
36         lcp[rk[i]] = j;

```

```
37 } }
```

#### 48. 后缀排序: DC3 [xzl/dc3.cpp]

DC3 后缀排序算法, 时空复杂度  $\Theta(n)$ 。字符串本体  $s$  数组、 $sa$  数组和  $rk$  数组都要求 3 倍空间。下标从 0 开始, 字符串长度为  $n$ , 字符集  $\Sigma$  为  $[0, m]$ 。partial\_sum 需要标准头文件 `numeric`。

```
1 #define CH(i, n) i < n ? s[i] : 0
2 static int ch[NMAX + 10][3], seq[NMAX + 10];
3 static int arr[NMAX + 10], tmp[NMAX + 10], cnt[NMAX + 10];
4 inline bool cmp(int i, int j) {
5     return ch[i][0] == ch[j][0] && ch[i][1] == ch[j][1] && ch[i][2] == ch[j][2];
6 }
7 inline bool sufcmp(int *s, int *rk, int n, int i, int j) {
8     if (s[i] != s[j]) return s[i] < s[j];
9     if ((i + 1) % 3 && (j + 1) % 3) return rk[i + 1] < rk[j + 1];
10    if (s[i + 1] != s[j + 1]) return s[i + 1] < s[j + 1];
11    return rk[i + 2] < rk[j + 2];
12 }
13 void radix_sort(int n, int m, int K, bool init = true) {
14     if (init) for (int i = 0; i < n; i++) arr[i] = i;
15     int *a = arr, *b = tmp;
16     for (int k = 0; k < K; k++) {
17         memset(cnt, 0, sizeof(int) * (m + 1));
18         for (int i = 0; i < n; i++) cnt[ch[a[i]][k]]++;
19         partial_sum(cnt, cnt + m + 1, cnt);
20         for (int i = n - 1; i >= 0; i--) b[--cnt[ch[a[i]][k]]] = a[i];
21         swap(a, b);
22     }
23     if (a != arr) memcpy(arr, tmp, sizeof(int) * n);
24 }
25 void suffix_sort(int *s, int n, int m, int *sa, int *rk) {
26     s[n] = 0; n++;
27     int p = 0, q = 0;
28     for (int i = 1; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
29         ch[p][2 - j] = CH(i + j, n);
30     for (int i = 2; i < n; i += 3, p++) for (int j = 0; j < 3; j++)
31         ch[p][2 - j] = CH(i + j, n);
32     radix_sort(p, m, 3);
33     for (int i = 0; i < p; i++) {
34         if (!q || (q && !cmp(arr[i - 1], arr[i]))) q++;
35         s[n + arr[i]] = q;
36     }
37     if (q < p) suffix_sort(s + n, p, q, sa + n, rk + n);
38     else {
39         for (int i = 0; i < p; i++) sa[n + s[n + i] - 1] = i;
40         for (int i = 0; i < p; i++) rk[n + sa[n + i]] = i + 1;
41     }
42     m = max(m, p);
43     p = q = 0;
44     for (int i = 1; i < n; i += 3, p++) rk[i] = rk[n + p];
45     for (int i = 2; i < n; i += 3, p++) rk[i] = rk[n + p];
46     for (int i = 0; i < n; i++) if (i % 3) seq[rk[i] - 1] = i;
47     for (int i = 0; i < n; i += 3, q++) {
48         ch[i][0] = i + 1 < n ? rk[i + 1] : 0;
49         ch[i][1] = s[i];
```

```
50     arr[q] = i;
51 }
52 radix_sort(q, m, 2, false);
53 for (int i = seq[0] == n - 1, j = arr[0] == n - 1, k = 0; i < p || j <
54     q; k++) {
55     if (i == p) sa[k] = arr[j++];
56     else if (j == q) sa[k] = seq[i++];
57     else if (sufcmp(s, rk, n, seq[i], arr[j])) sa[k] = seq[i++];
58     else sa[k] = arr[j++];
59 }
60 for (int i = 0; i < n - 1; i++) rk[sa[i]] = i + 1;
61 }
```

#### 49. 后缀排序: SA-IS [xzl/sais.cpp]

SA-IS 后缀数组排序。字符串存在  $str$  中, 下标从 1 开始, 长度为  $n$ , 并且  $str[n + 1]$  为哨兵字符, 编号为 1。后缀数组放在  $sa$  中, 下标从 1 开始。时空复杂度为  $\Theta(n)$ 。其中使用了 `vector<bool>` 来优化缓存命中率。

```
1 #define rep(i, l, r) for (register int i = (l); i <= (r); ++i)
2 #define rrep(i, r, l) for (register int i = (r); i >= (l); --i)
3 #define PUTS(x) sa[cur[tr[x]]--] = x
4 #define PUTL(x) sa[cur[tr[x]]++] = x
5 #define LMS(x) (!type[x - 1] && type[x])
6 #define RESET memset(sa + 1, 0, sizeof(int) * (n + 1)); \
7     memcpy(cur + 1, cnt + 1, sizeof(int) * m);
8 #define INDUCE rep(i, 1, m) cur[i] = cnt[i - 1] + 1; \
9     rep(i, 1, n + 1) if (sa[i] > 1 && !type[sa[i] - 1]) PUTL(sa[i] - 1); \
10    memcpy(cur + 1, cnt + 1, sizeof(int) * m); \
11    rrep(i, n + 1, 1) if (sa[i] > 1 && type[sa[i] - 1]) PUTS(sa[i] - 1);
12 void sais(int n, int m, int *str, int *sa) {
13     static int id[NMAX + 10];
14     vector<bool> type(n + 2);
15     type[n + 1] = true;
16     rrep(i, n, 1) type[i] = str[i] == str[i + 1] ? type[i + 1] : str[i] < s
17     tr[i + 1];
18     int cnt[m + 1], cur[m + 1], idx = 1, y = 0, rt, lrt, *ns = str + n + 2,
19     *nsa = sa + n + 2;
20     memset(cnt, 0, sizeof(int) * (m + 1));
21     rep(i, 1, n + 1) cnt[tr[i]]++;
22     rep(i, 1, m) cnt[i] += cnt[i - 1];
23     RESET rep(i, 2, n + 1) if (LMS(i)) PUTS(i); INDUCE
24     memset(id + 1, 0, sizeof(int) * n);
25     rep(i, 2, n + 1) if (LMS(sa[i])) {
26         register int x = sa[i];
27         for (rt = x + 1; !LMS(rt); rt++);
28         id[x] = y && rt + y == lrt + x && !memcmp(str + x, str + y, sizeof(in
29         t) * (rt - x + 1)) ? idx : ++idx;
30         y = x, lrt = rt;
31     }
32     int len = 0, pos[(n >> 1) + 1];
33     rep(i, 1, n) if (id[i]) {
34         ns[++len] = id[i];
35         pos[len] = i;
36     }
37     ns[len + 1] = 1, pos[len + 1] = n + 1;
38     if (len == idx - 1) rep(i, 1, len + 1) nsa[ns[i]] = i;
39     else sais(len, idx, ns, nsa);
```

```

37 RESET rrep(i, len + 1, 1) PUTS(pos[nsa[i]]); INDUCE
38 }
39 static int str[NMAX * 3 + 10], sa[NMAX * 3 + 10];

```

## 50. 后缀树 [xzl/后缀树, 字符串.cpp]

Ukkonen 在线添加尾部字符的后缀树构建算法。后缀树即后缀 Trie 的虚树，树上节点数不超过两倍的字符串总长。State 是后缀树上的节点。Trans 是后缀树的边，记录了一个区间  $[l, r]$  表示边所对应的子串。根节点没有 fail 指针。原字符串 str 的下标从 1 开始，字符串的最后一个字符是 EOFC，该字符不一定要字典序最大。注意  $n$  比原长多 1。字符集的第一个字母为 0，字符集  $\Sigma$  大小由 SIGMA 确定。添加字符串前需调用 `_append::reset`。时间复杂度为  $\Theta(n)$ ，空间复杂度为  $\Theta(n|\Sigma|)$ 。大字符集请使用 `unordered_map`。

```

1 #define SIGMA 27
2 #define EOFC (SIGMA - 1)
3 struct State {
4     struct Trans {
5         Trans(int _l, int _r, State *_nxt)
6             : l(_l), r(_r), nxt(_nxt) {}
7         int l, r; State *nxt;
8         int len() const { return r - l + 1; }
9     };
10    State() : fail(NULL) { memset(ch, 0, sizeof(ch)); }
11    State *fail; Trans *ch[SIGMA];
12 };
13 typedef State::Trans Trans;
14 static State *rt;
15 static char str[NMAX + 10];
16 static int n;
17 namespace _append {
18     static char dir;
19     static int len, cnt, cur;
20     static State *ap;
21     void reset() {
22         dir = -1; ap = rt;
23         len = cnt = cur = 0;
24     }
25     inline void append(char c) {
26         using namespace _append;
27         cnt++; cur++;
28         State *x, *y = NULL;
29         while (cnt) {
30             if (cnt <= len + 1) {
31                 len = cnt - 1;
32                 dir = len ? str[cur - len] : -1;
33             }
34             while (dir >= 0 && len >= ap->ch[dir]->len()) {
35                 len -= ap->ch[dir]->len();
36                 ap = ap->ch[dir]->nxt;
37                 dir = len ? str[cur - len] : -1;
38             }
39             if ((dir >= 0 && str[ap->ch[dir]->l + len] == c) ||
40                 (dir < 0 && ap->ch[c])) {
41                 if (dir < 0) dir = c;
42                 if (y) y->fail = ap;
43                 len++; return;
44             }

```

```

45         if (dir < 0) {
46             ap->ch[c] = new Trans(cur, n, new State);
47             x = ap;
48         } else {
49             Trans *t = ap->ch[dir];
50             x = new State;
51             x->ch[c] = new Trans(cur, n, new State);
52             x->ch[str[t->l + len]] = new Trans(t->l + len, t->r, t->nxt);
53             t->r = t->l + len - 1;
54             t->nxt = x;
55         }
56         if (y) y->fail = x;
57         if (ap != rt) ap = ap->fail;
58         y = x; cnt--;
59     }
60     inline void initialize() {
61         rt = new State;
62         _append::reset();
63         n = strlen(str) + 1;
64         for (int i = 1; i < n; i++) {
65             str[i] -= 'a';
66             append(str[i]);
67         }
68         str[n] = EOFC;
69         append(EOFC);
70     }

```

## 51. fwt [lmj/fwt.cpp]

```

1 Long Long m;
2 Long Long lim[5][5];
3 Long Long a[5][5000];
4 Long Long b[5][5000], x[5000], last[5000];
5 unsigned Long Long ans;
6 void clear() {
7     int i;
8     for (i = 0; i <= 2048; i++) {
9         a[1][i] = a[2][i] = a[3][i] = 0;
10        b[1][i] = b[2][i] = b[3][i] = 0;
11        last[i] = x[i] = 0;
12    }
13    ans = 0;
14 }
15 void FWT ( Long Long *x, Long Long f ) {
16     Long Long i, j, s, k;
17     Long Long u, t;
18     for (s = 2; s <= m; s = s * 2) {
19         k = s / 2;
20         for (i = 0; i < m; i += s) {
21             for (j = 1; j <= k; j++) {
22                 u = x[i+j-1]; t = x[i+j-1+k];
23                 x[i+j-1] = (u + t);
24                 x[i+j-1+k] = (u - t);
25                 if (f == -1) {
26                     x[i+j-1] = (x[i+j-1]/2);
27                     x[i+j-1+k] = (x[i+j-1+k]/2);
28                 }
29             }
30         }
31     }
32     Long Long chk ( Long Long x, Long Long lim ) {
33         if (0 <= x && x <= lim) return 1;

```

```

31 return 0;
32 }
33 void work () {
34     Long Long i , j , k;
35     m = 2048;
36     clear ();
37     for ( i = 1 ; i <= 2 ; i++ ) {
38         for ( j = 1 ; j <= 3 ; j++ ) {
39             scanf ( "%lld" , &lim[i][j] );
40         }
41     }
42     for ( i = 0 ; i <= 2000 ; i++ ) {
43         for ( j = 1 ; j <= 3 ; j++ ) {
44             for ( k = 0 ; k <= lim[1][j] ; k++ ) {
45                 if ( chk ( k + i , lim[2][j] ) )
46                     a[j][k^(k+i)]++;
47                 if ( i != 0 && chk ( k - i , lim[2][j] ) )
48                     a[j][k^(k-i)]++;
49             }
50         }
51         for ( j = 1 ; j <= 3 ; j++ ) {
52             for ( k = 0 ; k <= 2047 ; k++ ) {
53                 b[j][k] = a[j][k];
54             }
55         }
56         FWT ( b[1] , 1 );
57         FWT ( b[2] , 1 );
58         FWT ( b[3] , 1 );
59         for ( k = 0 ; k <= 2047 ; k++ ) {
60             x[k] = b[1][k] * b[2][k] * b[3][k];
61         }
62         FWT ( x , -1 );
63         for ( k = 0 ; k <= 2047 ; k++ ) {
64             ans += (k^i)*(x[k]-last[k]);
65             last[k] = x[k];
66         }
67     }
68     printf ( "%llu\n" , ans );
69 }

```

## 52. lct [lmj/lct.cpp]

```

1 struct node {
2     Long Long x;
3     Long Long lm , lp , rev;
4     Long Long s , siz;
5     Long Long ch[4] , fa;
6 } p[maxn];
7 void cut ( Long Long x , Long Long kind ) {
8     p[p[x].ch[kind]].fa *= -1;
9     p[x].ch[kind] = 0;
10    update ( x );
11 }
12 void down ( Long Long x ) {
13     if ( p[x].fa > 0 ) down ( p[x].fa );
14     pushdown ( x );
15 }
16 void rotate ( Long Long x , Long Long kind ) {
17     Long Long y = p[x].fa;

```

```

18     if ( p[y].fa > 0 ) p[p[y].fa].ch[y==p[p[y].fa].ch[1]] = x;
19     p[x].fa = p[y].fa;
20     if ( p[x].ch[kind^1] ) p[p[x].ch[kind^1]].fa = y;
21     p[y].ch[kind] = p[x].ch[kind^1];
22     p[y].fa = x;
23     p[x].ch[kind^1] = y;
24     update ( y ); update ( x );
25 }
26 void splay ( Long Long x ) {
27     down ( x );
28     for ( ; p[x].fa > 0 ; rotate ( x , x==p[p[x].fa].ch[1] ) )
29         if ( p[p[x].fa].fa > 0 && (x==p[p[x].fa].ch[1]) == (p[x].fa==p[p[p[x].fa].fa].ch[1]) )
30             rotate ( p[x].fa , x==p[p[x].fa].ch[1] );
31 }
32 void access ( Long Long x ) {
33     splay ( x );
34     cut ( x , 1 );
35     for ( ; p[x].fa != 0 ; ) {
36         splay ( -p[x].fa );
37         cut ( -p[x].fa , 1 );
38         p[-p[x].fa].ch[1] = x;
39         update ( -p[x].fa );
40         p[x].fa *= -1;
41         splay ( x );
42     }
43 }
44 void makeroot ( Long Long x ) {
45     access ( x );
46     p[x].rev ^= 1;
47     swap ( p[x].ch[0] , p[x].ch[1] );
48 }
49 void link ( Long Long x , Long Long y ) {
50     makeroot ( y );
51     p[y].fa = -x;
52 }

```

## 53. lichao\_segment\_tree [lmj/lichao\_segment\_tree.cpp]

用一个线段树维护凸壳（线段的最大高度），高度就是这条线在某个x坐标的y值，每个线段树节点维护一个这个区间的优势线段，就是中点的高度最大的那个点，插入线段的时候分类讨论，x1,x2,x3是新插入线段的l,mid,r的高度，t1,t2,t3是原来的优势线段，如果x1和t1，x3和t3的优劣情况相同，可以直接return或者直接把当前优势线段替换成x后return。否则考虑x2和t2，让高的那个当成现在节点的优势线段，然后低的那个向下递归，如果交点在左儿子就递归到左儿子，反之右儿子。因为没递归的儿子肯定有新的优势线段比被递归掉的那个线段优秀，只有递归的儿子需要进一步讨论。

```

1 const int maxn = 51000;
2 struct line {
3     int x1 , y1 , x2 , y2;
4 };
5 struct node {
6     int l , r , cov;
7     line t;
8     node *ll , *rr;
9 } pool[maxn*4] , *t;
10 int top;

```

```

11 int n , m;
12 void buildtree ( node *id , int l , int r ) {
13     id -> l = l; id -> r = r;
14     if ( l == r ) return ;
15     int mid = (id->l+id->r)/2;
16     id -> ll = &pool[++top]; id -> rr = &pool[++top];
17     buildtree ( id -> ll , l , mid ); buildtree ( id -> rr , mid + 1 , r );
18 }
19 double geth ( line l , int x ) {
20     if ( l.x1 == l.x2 ) return l.y1;
21     double ret = ( ((double)x) - l.x1 ) * (l.y2-l.y1) / (l.x2-l.x1) + l.y1;
22     return ret;
23 }
24 void add ( node *id , int l , int r , line x ) {
25     int mid = (id->l+id->r)/2;
26     if ( id -> l == l && id -> r == r ) {
27         if ( id -> cov == 0 ) {
28             id -> cov = 1;
29             id -> t = x;
30         }
31         else {
32             double x1 , x2 , x3 , t1 , t2 , t3;
33             line y;
34             x1 = geth ( x , l ); x2 = geth ( x , mid ); x3 = geth ( x , r );
35             t1 = geth ( id -> t , l ); t2 = geth ( id -> t , mid ); t3 = geth (
id -> t , r );
36             if ( t1 >= x1 && t3 >= x3 ) return ;
37             if ( t1 <= x1 && t3 <= x3 ) {
38                 id -> t = x;
39                 return ;
40             }
41             if ( x2 >= t2 ) swap ( id -> t , x );
42             if ( (t1>=x1) == (t2>=x2) ) {
43                 add ( id -> rr , mid + 1 , r , x );
44             }
45             else add ( id -> ll , l , mid , x );
46         }
47     }
48     return ;
49     if ( r <= mid ) add ( id -> ll , l , r , x );
50     else {
51         if ( l > mid ) add ( id -> rr , l , r , x );
52         else {
53             add ( id -> ll , l , mid , x );
54             add ( id -> rr , mid + 1 , r , x );
55         }
56     }
57 }
58 double query ( node *id , int x ) {
59     if ( id -> l == id -> r ) {
60         if ( id -> cov == 1 ) return geth ( id -> t , x );
61         else return -1000000000.0;
62     }
63     int mid = (id->l+id->r)/2;
64     double tmp;
65     if ( id -> cov == 1 ) tmp = geth ( id -> t , x );
66     else tmp = -1000000000.0;
67     if ( x <= mid ) return max ( tmp , query ( id -> ll , x ) );

```

```

68     else return max ( tmp , query ( id -> rr , x ) );
69 }
70 void work () {
71     int i , op , x;
72     double ans;
73     line tmp;
74     scanf ( "%d%d" , &n , &m );
75     t = &pool[++top];
76     buildtree ( t , 1 , 100000 );
77     for ( i = 1 ; i <= n ; i++ ) {
78         scanf ( "%d%d%d%d" , &tmp.x1 , &tmp.y1 , &tmp.x2 , &tmp.y2 );
79         if ( tmp.x1 > tmp.x2 ) {
80             swap ( tmp.x1 , tmp.x2 );
81             swap ( tmp.y1 , tmp.y2 );
82         }
83         if ( tmp.x1 == tmp.x2 ) {
84             tmp.y1 = tmp.y2 = max ( tmp.y1 , tmp.y2 );
85         }
86         if ( tmp.x1 > 100000 || tmp.x2 < 1 ) continue;
87         add ( t , max ( 1 , tmp.x1 ) , min ( 100000 , tmp.x2 ) , tmp );
88     }
89     for ( i = 1 ; i <= m ; i++ ) {
90         scanf ( "%d" , &op );
91         if ( op == 0 ) {
92             scanf ( "%d%d%d%d" , &tmp.x1 , &tmp.y1 , &tmp.x2 , &tmp.y2 );
93             if ( tmp.x1 > tmp.x2 ) {
94                 swap ( tmp.x1 , tmp.x2 );
95                 swap ( tmp.y1 , tmp.y2 );
96             }
97             if ( tmp.x1 == tmp.x2 ) {
98                 tmp.y1 = tmp.y2 = max ( tmp.y1 , tmp.y2 );
99             }
100             if ( tmp.x1 > 100000 || tmp.x2 < 1 ) continue;
101             add ( t , max ( 1 , tmp.x1 ) , min ( 100000 , tmp.x2 ) , tmp );
102         }
103         else {
104             scanf ( "%d" , &x );
105             ans = query ( t , x );
106             //printf ( "%lf\n" , query ( t , x ) );
107             if ( ans < -1000000000.0 ) ans = 0.0;
108             printf ( "%lf\n" , ans );
109         }
110     }
111 }

```

#### 54. xihe\_tree [lmj/xihe\_tree.cpp]

析合树，用来处理连续段的数据结构

代码求包含 $[l,r]$ 的最小连续段（相当于lca）

son里面的儿子按顺序是节点编号从小到大。

<https://oi-wiki.org/ds/divide-combine/>

解释一下本文可能用到的符号：  $\wedge$  逻辑与，  $\vee$  逻辑或。

#### 关于段的问题

我们由一个小清新的问题引入：



对于一个  $1 - n$  的排列，我们称一个值域连续的区间为段。问一个排列的段的个数。  
比如，  $5, 3, 4, 1, 2$  的段有：  $[1, 1], [2, 2], [3, 3], [4, 4], [5, 5], [2, 3], [4, 5], [1, 3], [2, 5], [1, 5]$ 。

看到这个东西，感觉要维护区间的值域集合，复杂度好像挺不友好的。线段树可以查询某个区间是否为段，但不太能统计段的个数（也可能是因为我太菜了不会用线段树）

这里我们引入这个神奇的数据结构——析合树！

## 连续段

在介绍析合树之前，我们先做一些前提条件的限定。鉴于 LCA 的课件的定义十分玄乎，为保证读者的身心健康，我就口糊一些人性化的定义吧。

排列与连续段

**排列**：定义一个  $n$  阶排列  $P$  是一个大小为  $n$  的序列，使得  $P_i$  取遍  $1, 2, \dots, n$ 。说得形式化一点， $n$  阶排列  $P$  是一个有序集合满足：

1.  $|P| = n$  .
2.  $\forall i, P_i \in [1, n]$  .
3.  $\nexists i, j \in [1, n], P_i = P_j$  .

**连续段**：对于排列  $P$ ，定义连续段  $(P, [l, r])$  表示一个区间  $[l, r]$ ，要求  $P_{l \sim r}$  值域是连续的。说得更形式化一点，对于排列  $P$ ，连续段表示一个区间  $[l, r]$  满足：

$$(\nexists x, z \in [l, r], y \notin [l, r], P_x < P_y < P_z)$$

特别地，当  $l > r$  时，我们认为这是一个空的连续段，记作  $(P, \emptyset)$ 。

我们称排列  $P$  的所有连续段的集合为  $I_P$ ，并且我们认为  $(P, \emptyset) \in I_P$ 。

连续段的运算

连续段是依赖区间和值域定义的，于是我们可以定义连续段的交并差的运算。

定义  $A = (P, [a, b]), B = (P, [x, y])$ ，且  $A, B \in I_P$ 。于是连续段的关系和运算可以表示为：

1.  $A \subseteq B \Leftrightarrow x \leq a \wedge b \leq y$  .
2.  $A = B \Leftrightarrow a = x \wedge b = y$  .
3.  $A \cap B = (P, [\max(a, x), \min(b, y)])$  .
4.  $A \cup B = (P, [\min(a, x), \max(b, y)])$  .
5.  $A \setminus B = (P, i | i \in [a, b] \wedge i \notin [x, y])$  .

其实这些运算就是普通的集合交并差放在区间上而已。

连续段的性质

连续段的一些显而易见的性质。我们定义  $A, B \in I_P$ ，那么有  $A \cup B, A \cap B, A \setminus B, B \setminus A \in I_P$ 。

证明？证明的本质就是集合的交并差的运算。

## 析合树

好的，现在讲到重点了。你可能已经猜到了，析合树正是由连续段组成的一棵树。但是要知道一个排列可能有多达  $O(n^2)$  个连续段，因此我们就要抽出其中更基本的连续段组成析合树。

本原段

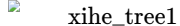
其实这个定义全称叫作 **本原连续段**。但笔者认为本原段更为简洁。

对于排列  $P$ ，我们认为一个本原段  $M$  表示在集合  $I_P$  中，不存在与之相交且不包含的连续段。形式化地定义，我们认为  $X \in I_P$  且满足  $\forall A \in I_P, X \cap A = (P, \emptyset) \vee X \subseteq A \vee A \subseteq X$ 。

所有本原段的集合为  $M_P$ 。显而易见， $(P, \emptyset) \in M_P$ 。

显然，本原段之间只有相离或者包含关系。并且你发现一个连续段可以由几个互不相交的本原段构成。最大的本原段就是整个排列本身，它包含了其他所有本原段，因此我们认为本原段可以构成一个树形结构，我们称这个结构为析合树。更严格地说，排列  $P$  的析合树由排列  $P$  的所有本原段组成。

前面干讲这么多的定义，不来点图怎么行。考虑排列  $P = 9, 1, 10, 3, 2, 5, 7, 6, 8, 4$ 。它的本原段构成的析合树如下：

 xihe\_tree1

在图中我们没有标明本原段。而图中每个结点都代表一个本原段。我们只标明了每个本原段的值域。举个例子，结点  $[5, 8]$  代表的本原段就是  $(P, [6, 9]) = 5, 7, 6, 8$ 。于是这里就有一个问题：什么是析点合点？

析点与合点

这里我们直接给出定义，稍候再来讨论它的正确性。

1. **值域区间**：对于一个结点  $u$ ，用  $[u_l, u_r]$  表示该结点的值域区间。
2. **儿子序列**：对于析合树上的一个结点  $u$ ，假设它的儿子结点是一个有序序列，该序列是以值域区间为元素的（单个的数  $x$  可以理解为  $[x, x]$  的区间）。我们把这个序列称为儿子序列。记作  $S_u$ 。
3. **儿子排列**：对于一个儿子序列  $S_u$ ，把它的元素离散化成正整数后形成的排列称为儿子排列。举个例子，对于结点  $[5, 8]$ ，它的儿子序列为  $[5, 5], [6, 7], [8, 8]$ ，那么把区间排序标个号，则它的儿子排列就为  $1, 2, 3$ ；类似的，结点  $[4, 8]$  的儿子排列为  $2, 1$ 。结点  $u$  的儿子排列记为  $P_u$ 。
4. **合点**：我们认为，儿子排列为顺序或者逆序的点为合点。形式化地说，满足  $P_u = 1, 2, \dots, |S_u|$  或者  $P_u = |S_u|, |S_u| - 1, \dots, 1$  的点称为合点。叶子结点没有儿子排列，我们也认为它是合点。
5. **析点**：不是合点的就是析点。

从图中可以看到，只有  $[1, 10]$  不是合点。因为  $[1, 10]$  的儿子排列是  $3, 1, 4, 2$ 。

析点与合点的性质

析点与合点的命名来源于他们的性质。首先我们有一个非常显然的性质：对于析合树中任何的结点  $u$ ，其儿子序列区间的并集就是结点  $u$  的值域区间。即  $\bigcup_{i=1}^{|S_u|} S_u[i] = [u_l, u_r]$ 。

对于一个合点  $u$ ：其儿子序列的任意子区间都构成一个连续段。形式化地说， $\forall S_u[l \sim r]$ ，有  $\bigcup_{i=l}^r S_u[i] \in I_P$ 。

对于一个析点  $u$ ：其儿子序列的任意长度大于 1（这里的长度是指儿子序列中的元素数，不是下标区间的长度）的子区间都不构成一个连续段。形式化地说， $\forall S_u[l \sim r], l < r$ ，有  $\bigcup_{i=l}^r S_u[i] \notin I_P$ 。

合点的性质不难口糊证明。因为合点的儿子排列要么是顺序，要么是倒序，而值域区间也是首位相接，因此只要是连续的一段子序列（区间）都是一个连续段。

对于析点的性质可能很多读者就不太能理解了：为什么任意长度大于1的子区间都不构成连续段？

使用反证法。假设对于一个点  $u$ ，它的儿子序列中有一个最长的区间  $S_u[l \sim r]$  构成了连续段。那么这个  $A = \bigcup_{i=l}^r S_u[i] \in I_P$ ，也就意味着  $A$  是一个本原段！（因为  $A$  是儿子序列中最长的，因此找不到一个与它相交又不包含的连续段）于是你就没有使用所有的本原段构成这个析合树。矛盾。

#### 析合树的构造

前面讲了这么多零零散散的东西，现在就来具体地讲如何构造析合树。LCA 大佬的线性构造算法我是没看懂的，今天就讲一下比较好懂的  $O(n \log n)$  的算法。

#### 增量法

我们考虑增量法。用一个栈维护前  $i-1$  个元素构成的析合森林。在这里我需要着重强调，析合森林的意思是，在任何时候，栈中结点要么是析点要么是合点。现在考虑当前结点  $P_i$ 。

1. 我们先判断它能否成为栈顶结点的儿子，如果能就变成栈顶的儿子，然后把栈顶取出，作为当前结点。重复上述过程直到栈空或者不能成为栈顶结点的儿子。
2. 如果不能成为栈顶的儿子，就看能不能把栈顶的若干个连续的结点都合并成一个结点（判断能否合并的方法在后面），把合并后的点，作为当前结点。
3. 重复上述过程直到不能进行为止。然后结束此次增量，直接把当前结点压栈。

接下来我们仔细解释一下。

#### 具体的策略

我们认为，如果当前点能够成为栈顶结点的儿子，那么栈顶结点是一个合点。如果是析点，那么你合并后这个析点就存在一个子连续段，不满足析点的性质。因此一定是合点。

如果无法成为栈顶结点的儿子，那么我们就看栈顶连续的若干个点能否与当前点一起合并。我们预处理一个数组  $L$ ， $L_i$  表示右端点下标为  $i$  的连续段中，左端点的最小值。当前结点为  $P_i$ ，栈顶结点记为  $t$ 。

1. 如果  $t_l < L_i$  那么显然当前结点无法合并；
2. 如果  $t_l = L$ ，那么这就是两个结点合并，合并后就是一个合点；
3. 如果在栈中存在一个点  $t'$  的左端点  $t'_l = L_i$ ，那么一定可以从当前结点合并到  $t'$  形成一个析点；
4. 否则，我们找到栈中的一个点  $t'$  使得  $t'_l < L_i \leq t'_r$ 。由连续段的差运算可知  $(P, [t'_r + 1, i])$  也是连续段，于是合并  $t'$  之后的结点到当前结点成一个析点即可。

#### 判断能否合并

最后，我们考虑如何处理  $L$  数组。事实上，一个连续段  $(P, [l, r])$  等价于区间极差与区间长度 -1 相等。即

$$\max_{l \leq i \leq r} P_i - \min_{l \leq i \leq r} P_i = r - l$$

而且由于  $P$  是一个排列，因此对于任意的区间  $[l, r]$  都有

$$\max_{l \leq i \leq r} P_i - \min_{l \leq i \leq r} P_i \geq r - l$$

于是我们就维护  $\max_{l \leq i \leq r} P_i - \min_{l \leq i \leq r} P_i - (r - l)$ ，那么要找到一个连续段相当于查询一个最小值！

有了上述思路，不难想到这样的算法。对于增量过程中的当前的  $i$ ，我们维护一个数组  $Q$  表示区间  $[j, i]$  的极差减长度。即

$$Q_j = \max_{j \leq k \leq i} P_k - \min_{j \leq k \leq i} P_k - (i - j), \quad 0 < j < i$$

现在我们想知道在  $1 \sim i-1$  中是否存在一个最小的  $j$  使得  $Q_j = 0$ 。这等价于求  $Q_{1 \sim i-1}$  的最小值。求得最小的  $j$  就是  $L_i$ 。如果没有，那么  $L_i = i$ 。

但是当第  $i$  次增量结束时，我们需要快速把  $Q$  数组更新到  $i+1$  的情况。原本的区间从  $[j, i]$  变成  $[j, i+1]$ ，如果  $P_{i+1} > \max$  或者  $P_{i+1} < \min$  都会造成  $Q_j$  发生变化。如何变化？如果  $P_{i+1} > \max$ ，相当于我们把  $Q_j$  先减掉  $\max$  再加上  $P_{i+1}$  就完成了  $Q_j$  的更新； $P_{i+1} < \min$  同理，相当于  $Q_j = Q_j + \min - P_{i+1}$ 。

那么如果对于一个区间  $[x, y]$ ，满足  $P_{x \sim i}, P_{x+1 \sim i}, P_{x+2 \sim i}, \dots, P_{y \sim i}$  的区间  $\max$  都相同呢？你已经发现了，那么相当于我们在做一个区间加的操作；同理，当  $P_{x \sim i}, P_{x+1 \sim i}, \dots, P_{y \sim i}$  的区间  $\min$  都想同时也是一个区间加的操作。同时， $\max$  和  $\min$  的更新是相互独立的，因此可以各自更新。

因此我们对  $Q$  的维护可以这样描述：

1. 找到最大的  $j$  使得  $P_j > P_{i+1}$ ，那么显然， $P_{j+1 \sim i}$  这一段数全部小于  $P_{i+1}$ ，于是就需要更新  $Q_{j+1 \sim i}$  的最大值。由于  $P_i, \max(P_i, P_{i-1}), \max(P_i, P_{i-1}, P_{i-2}), \dots, \max(P_i, P_{i-1}, \dots, P_{j+1})$  是（非严格）单调递增的，因此可以每一段相同的  $\max$  做相同的更新，即区间加操作。
2. 更新  $\min$  同理。
3. 把每一个  $Q_j$  都减 1。因为区间长度加 1。
4. 查询  $L_i$ ：即查询  $Q$  的最小值的所在的下标。

没错，我们可以使用线段树维护  $Q$ ！现在还有一个问题：怎么找到相同的一段使得他们的  $\max / \min$  都相同？使用单调栈维护！维护两个单调栈分别表示  $\max / \min$ 。那么显然，栈中以相邻两个元素为端点的区间的  $\max / \min$  是相同的，于是在维护单调栈的时候顺便更新线段树即可。

#### xihe\_tree2

```
1 const int maxn = 120000;
2 struct tree {
3     int l, r, lazy;
4     int mn;
5     tree *ll, *rr;
6 } poolt[maxn*2], *t;
7 struct edge {
8     int v;
9     edge *next;
10 } poole[maxn*2];
11 struct node {
12     int fa, l, r, kl, kr; // [l,r] represents its range in a[], [kl,kr] represents its range of value
13     int kind; // 0 for xi, 1 for he
14     edge *son; // sons from left to right
15 } p[maxn*2];
16 int topt, tope, tot; // tot: number of nodes in xihe-tree
17 int n, m, root;
```

```

18 int a[maxn];
19 int id[maxn]; //id[i]=label of (a[i],a[i]) in xihe-tree
20 int lst[maxn*2]; //if p[i] is he, lst[i]=the last son -> l, lst[i]=0 if p
[i] is xi
21 int s1[maxn], s2[maxn], tp1, tp2;
22 int st[maxn*2], tops;
23 int dep[maxn*2], go[18][maxn*2];
24 void add ( int u, int v ) {
25     edge *tmp = &poole[++tope];
26     tmp->v = v; tmp->next = p[u].son; p[u].son = tmp;
27     //printf ( "%d %d\n", u, v );
28 }
29 void reverse ( int i ) {
30     edge *tmp = NULL, *t2;
31     while ( p[i].son ) {
32         t2 = p[i].son->next;
33         p[i].son->next = tmp;
34         tmp = p[i].son;
35         p[i].son = t2;
36     }
37     p[i].son = tmp;
38 }
39 void buildtree ( tree *id, int l, int r ) {
40     id->l = l; id->r = r;
41     if ( l == r ) return;
42     int mid = (l+r)/2;
43     id->ll = &poolt[++topt]; id->rr = &poolt[++topt];
44     buildtree ( id->ll, l, mid ); buildtree ( id->rr, mid+1, r );
45 }
46 void pushdown ( tree *id ) {
47     if ( id->lazy ) {
48         id->ll->lazy += id->lazy;
49         id->ll->mn += id->lazy;
50         id->rr->lazy += id->lazy;
51         id->rr->mn += id->lazy;
52         id->lazy = 0;
53     }
54 }
55 void change ( tree *id, int l, int r, int x ) {
56     //printf ( "%d %d %d %d %d\n", id->l, id->r, l, r, x );
57     if ( id->l == l && id->r == r ) {
58         id->mn += x;
59         id->lazy += x;
60         return;
61     }
62     pushdown ( id );
63     int mid = (id->l+id->r)/2;
64     if ( r <= mid ) change ( id->ll, l, r, x );
65     else {
66         if ( l > mid ) change ( id->rr, l, r, x );
67         else {
68             change ( id->ll, l, mid, x );
69             change ( id->rr, mid+1, r, x );
70         }
71     }
72     id->mn = min ( id->ll->mn, id->rr->mn );
73 }
74 int querypoint ( tree *id, int l ) {

```

```

75     if ( id->l == id->r ) return id->mn;
76     pushdown ( id );
77     int mid = (id->l+id->r)/2;
78     if ( l <= mid ) return querypoint ( id->ll, l );
79     else return querypoint ( id->rr, l );
80 }
81 int query ( tree *id ) {
82     if ( id->l == id->r ) return id->l;
83     pushdown ( id );
84     if ( id->ll->mn == 0 ) return query ( id->ll );
85     else return query ( id->rr );
86 }
87 bool chk ( int x ) {
88     if ( x == 0 || querypoint ( t, x ) != 0 ) return false;
89     return true;
90 }
91 void dfs ( int i ) {
92     /*printf ( "%d:", i );
93     for ( edge *j = p[i].son; j; j = j->next ) {
94         printf ( "%d ", j->v );
95     }
96     printf ( "\n" );*/
97     for ( edge *j = p[i].son; j; j = j->next ) {
98         p[j->v].fa = i;
99         go[0][j->v] = i;
100         dep[j->v] = dep[i] + 1;
101         dfs ( j->v );
102         p[i].kl = min ( p[i].kl, p[j->v].kl );
103         p[i].kr = max ( p[i].kr, p[j->v].kr );
104     }
105 }
106 int lca ( int u, int v ) {
107     int i;
108     if ( dep[u] < dep[v] ) swap ( u, v );
109     for ( i = 17; i >= 0; i-- ) if ( dep[go[i][u]] >= dep[v] ) u = go[i][
u];
110     if ( u == v ) return u;
111     for ( i = 17; i >= 0; i-- ) if ( go[i][u] != go[i][v] ) {
112         u = go[i][u];
113         v = go[i][v];
114     }
115     //printf ( "%d %d\n", u, v );
116     return go[0][u];
117 }
118 int jump ( int u, int x ) {
119     int i;
120     for ( i = 0; i <= 17; i++ ) {
121         if ( x % 2 ) u = go[i][u];
122         x >>= 1;
123     }
124     return u;
125 }
126 void work () {
127     int i, j, now, tmp, u, v, l;
128     scanf ( "%d", &n );
129     for ( i = 1; i <= n; i++ ) {
130         scanf ( "%d", &a[i] );
131     }

```

```

132 t = &poolt[++tot];
133 buildtree ( t , 1 , n );
134 tops = 0;
135 for ( i = 1 ; i <= n ; i++ ) {
136     while ( tp1 && a[s1[tp1]] > a[i] ) {
137         change ( t , s1[tp1-1] + 1 , s1[tp1] , a[s1[tp1]] );
138         tp1--;
139     }
140     while ( tp2 && a[s2[tp2]] < a[i] ) {
141         change ( t , s2[tp2-1] + 1 , s2[tp2] , -a[s2[tp2]] );
142         tp2--;
143     }
144     change ( t , s1[tp1] + 1 , i , -a[i] ); s1[++tp1] = i;
145     change ( t , s2[tp2] + 1 , i , a[i] ); s2[++tp2] = i;
146
147     id[i] = ++tot; p[tot].l = p[tot].r = i; p[tot].kind = 0;
148     now = tot;
149     tmp = query ( t );
150     //printf ( "%d %d %d\n" , i , now , tmp );
151     while ( tops && tmp <= p[st[tops]].l ) {
152         if ( p[st[tops]].kind == 1 && chk ( lst[st[tops]] ) ) {
153             p[st[tops]].r = i;
154             add ( st[tops] , now );
155             now = st[tops--];
156         }
157         else {
158             if ( tops && chk ( p[st[tops]].l ) ) {
159                 p[++tot].kind = 1;
160                 p[tot].l = p[st[tops]].l; p[tot].r = i; lst[tot] = p[now].l;
161                 add ( tot , st[tops--] ); add ( tot , now );
162                 now = tot;
163             }
164             else {
165                 add ( ++tot , now );
166                 do {
167                     add ( tot , st[tops--] );
168                 } while ( tops && !chk ( p[st[tops]].l ) );
169                 p[tot].l = p[st[tops]].l; p[tot].r = i;
170                 add ( tot , st[tops--] );
171                 now = tot;
172             }
173         }
174     }
175     st[++tops] = now;
176     change ( t , 1 , i , -1 );
177 }
178 root = st[1];
179 for ( i = 1 ; i <= tot ; i++ ) {
180     if ( p[i].kind == 1 ) {
181         reverse ( i );
182     }
183     p[i].kl = n + 1;
184 }
185 for ( i = 1 ; i <= n ; i++ ) {
186     p[id[i]].kl = p[id[i]].kr = a[i];
187 }
188 dep[root] = 1;
189 dfs ( root );

```

```

190
191 for ( i = 1 ; i <= 17 ; i++ )
192     for ( j = 1 ; j <= tot ; j++ )
193         go[i][j] = go[i-1][go[i-1][j]];
194 scanf ( "%d" , &m );
195 for ( i = 1 ; i <= m ; i++ ) {
196     scanf ( "%d%d" , &u , &v );
197     u = id[u]; v = id[v];
198     l = lca ( u , v );
199     //printf ( "%d %d %d\n" , u , v , l );
200     if ( p[l].kind == 1 ) {
201         u = jump ( u , dep[u] - dep[l] - 1 );
202         v = jump ( v , dep[v] - dep[l] - 1 );
203         printf ( "%d %d\n" , p[u].l , p[v].r );
204     }
205     else printf ( "%d %d\n" , p[l].l , p[l].r );
206 }
207 }

```

### 55. xihe\_tree\_counting [lmj/xihe\_tree\_counting.cpp]

代码中  $f(x)$  为  $n$  个叶子的析合树的形态数量，也是合法连续段不同的排列种类（不同当且仅当存在一个区间，在一个排列里是连续段，在另一个里不是）。

考虑根节点是析点还是合点，然后枚举子树个数，要求子树叶子和是  $n$ 。

$$f(n) = \sum_{k \geq 2} \prod_{\sum_{i=1}^k a_i = n} f(a_i) + \sum_{k \geq 4} \prod_{\sum_{i=1}^k a_i = n} f(a_i)$$

```

1 LL n , mod;
2 LL dp[5][5100];
3 LL f[5100];
4 void work () {
5     LL i , j , k;
6     scanf ( "%lld%lld" , &n , &mod );
7     f[1] = dp[1][1] = 1;
8     for ( i = 2 ; i <= n ; i++ ) {
9         for ( j = 1 ; j < i ; j++ ) {
10             for ( k = 4 ; k >= 2 ; k-- ) {
11                 dp[k][i] = (dp[k][i]+f[j]*dp[k-1][i-j]) % mod;
12                 if ( k == 4 ) {
13                     dp[k][i] = (dp[k][i]+f[j]*dp[k][i-j]) % mod;
14                 }
15             }
16         }
17         f[i] = (dp[2][i]+dp[3][i]+dp[4][i]+dp[4][i]) % mod;
18         dp[1][i] = f[i];
19     }
20     for ( i = 1 ; i <= n ; i++ ) {
21         printf ( "%lld\n" , f[i] );
22     }
23 }

```

### 56. 二项堆 [xzl/二项堆, 数据结构.cpp]

```

1 #define LOGN 17
2 template <typename T>
3 struct Node {
4     Node(const T &val) : val(_val), ch(NULL), nxt(NULL) {}
5     T val; Node *ch, *nxt;

```

```

6 };
7 template <typename T>
8 struct Heap {
9     typedef Node<T> *TNode;
10    Heap() { memset(tr, 0, sizeof(tr)); }
11    TNode tr[LOGN];
12    auto operator[](int i) -> TNode& { return tr[i]; }
13    auto operator[](int i) const -> TNode { return tr[i]; }
14 };
15 template <typename TNode>
16 auto fuse(TNode x, TNode y) -> TNode {
17     if (y->val < x->val) swap(x, y);
18     y->nxt = x->ch;
19     x->ch = y;
20     return x;
21 }
22 template <typename THeap>
23 auto meld(THeap &u, THeap &v) -> THeap {
24     THeap z;
25     for (int i = 0; i < LOGN; i++) {
26         if (z[i] && u[i]) {
27             z[i+1] = fuse(z[i], u[i]);
28             z[i] = NULL;
29         } else if (u[i]) z[i] = u[i];
30         if (z[i] && v[i]) {
31             z[i+1] = fuse(z[i], v[i]);
32             z[i] = NULL;
33         } else if (v[i]) z[i] = v[i];
34     } return z;
35 }
36 template <typename T>
37 struct HeapInterface {
38     typedef Heap<T> THeap;
39     typedef typename THeap::TNode TNode;
40     HeapInterface() : size(0) {}
41     THeap q; int size;
42     void pop() {
43         int t = _top(), i;
44         TNode x = q[t];
45         q[t] = NULL;
46         THeap p;
47         for (i = 0, x = x->ch; x; x = x->nxt, i++) p[i] = x;
48         reverse(&p[0], &p[i]);
49         q = meld(q, p);
50         size--;
51     }
52     void push(const T &x) {
53         THeap p;
54         p[0] = new typename remove_pointer<TNode>::type(x);
55         q = meld(q, p);
56         size++;
57     }
58     auto _top() const -> int {
59         int t = 0;
60         for (int i = 1; i < LOGN; i++)
61             if (!q[t] || (q[i] && q[i]->val < q[t]->val))
62                 t = i;
63         return t;

```

```

64     }
65     T top() const { return q[_top()]->val; }
66     bool empty() const { return size == 0; }
67 };

```

## 57. 左偏树 [lmj/leftist\_tree.cpp]

核心操作split和merge, merge时候让小的当堆顶, 继续合并右子树和另外一棵树, 之后维护左偏性质。

```

1 struct node {
2     int x, i, dist;
3     node *ll, *rr;
4 } pool[maxn], *t[maxn];
5 int n, m;
6 int a[maxn];
7 int c[maxn], f[maxn];
8 int getdist ( node *id ) {
9     if ( id == NULL ) return -1;
10    return id -> dist;
11 }
12 node *merge ( node *id1, node *id2 ) {
13     if ( id1 == NULL ) return id2;
14     if ( id2 == NULL ) return id1;
15     if ( id1 -> x > id2 -> x ) swap ( id1, id2 );
16     id1 -> rr = merge ( id1 -> rr, id2 );
17     if ( getdist ( id1 -> ll ) < getdist ( id1 -> rr ) ) swap ( id1 -> ll,
18         id1 -> rr );
19     id1 -> dist = getdist ( id1 -> rr ) + 1;
20     return id1;
21 }
22 int find ( int x ) {
23     int i, t;
24     for ( i = x; c[i] > 0; i = c[i] );
25     while ( x != i ) {
26         t = c[x];
27         c[x] = i;
28         x = t;
29     }
30     return i;
31 }
32 void Union ( int x, int y ) {
33     t[x] = merge ( t[x], t[y] );
34     c[x] += c[y];
35     c[y] = x;
36 }

```

## 58. 序列splay [sll/区间splay.cpp]

```

1 int n, m, sz, rt;
2 char ch[10];
3 int tr[N][2], fa[N], v[N], sum[N];
4 int mx[N], lx[N], rx[N];
5 int st[N], size[N], top, tag[N];
6 bool rev[N];
7 void pushup(int u) {
8     size[u] = 1, sum[u] = v[u]; int l = tr[u][0], r = tr[u][1];
9     if (l) size[u] += size[l], sum[u] += sum[l];
10    if (r) size[u] += size[r], sum[u] += sum[r];
11    mx[u] = v[u]; if (l) mx[u] = max(mx[u], mx[l]); if (r) mx[u] = max(mx[u], mx[r]);

```

```

12  if(L&&R)mx[u]=max(mx[u],rx[L]+v[u]+lx[r]);
13  else if(L)mx[u]=max(mx[u],rx[L]+v[u]);
14  else if(R)mx[u]=max(mx[u],v[u]+lx[r]);
15  lx[u]=0;if(L)lx[u]=lx[L];rx[u]=0;if(R)rx[u]=rx[R];
16  if(!L)lx[u]=max(lx[u],v[u]);if(!R)rx[u]=max(rx[u],v[u]);
17  if(!L&&R)lx[u]=max(lx[u],v[u]+lx[r]);if(!R&&L)rx[u]=max(rx[u],v[u]+rx[L]);
18  if(L)lx[u]=max(lx[u],sum[L]+v[u]);if(R)rx[u]=max(rx[u],sum[R]+v[u]);
19  if(L&&R)lx[u]=max(lx[u],sum[L]+v[u]+lx[r]);rx[u]=max(rx[u],sum[R]+v[u]+rx[L]);
20 }
21 void work(int k,int c) {
22     tag[k]=c,v[k]=c,sum[k]=size[k]*c;
23     mx[k]=(c>0?c*size[k]:c),lx[k]=rx[k]=(c>0?c*size[k]:0);
24 }
25 void rve(int k) {
26     rev[k]^=1;
27     swap(lx[k],rx[k]);
28     swap(tr[k][0],tr[k][1]);
29 }
30 void pushdown(int u) {
31     int l=tr[u][0],r=tr[u][1];
32     if(tag[u]!=12345) {
33         if(L)work(l,tag[u]);if(R)work(r,tag[u]);
34         tag[u]=12345;
35     }
36     if(rev[u]) {
37         if(L)rve(L);if(R)rve(R);
38         rev[u]^=1;
39     }
40 void rotate(int x,int &k) {
41     int y=fa[x],z=fa[y];
42     int l=(tr[y][1]==x),r=L^1;
43     if(y==k)k=x;
44     else tr[z][tr[z][1]==y]=x;
45     fa[x]=z,fa[y]=x,fa[tr[x][r]]=y;
46     tr[y][l]=tr[x][r],tr[x][r]=y;
47     pushup(y);pushup(x);
48 }
49 void splay(int x,int &k) {
50     while(x!=k) {
51         int y=fa[x],z=fa[y];
52         if(y!=k) {
53             if(tr[y][0]==x^tr[z][0]==y)
54                 rotate(x,k);
55             else rotate(y,k);
56         }
57         rotate(x,k);
58     }
59 int find(int k,int rk) {
60     pushdown(k);
61     int l=tr[k][0],r=tr[k][1];
62     if(size[L]>=rk)return find(L,rk);
63     else if(size[L]+1==rk)return k;
64     else return find(r,rk-size[L]-1);
65 }
66 int split(int l,int r) {
67     int x=find(rt,l),y=find(rt,r+2);

```

```

68     splay(x,rt),splay(y,tr[x][1]);
69     return tr[y][0];
70 }
71 int a[N];
72 void newnode(int k,int c)
73 {v[k]=sum[k]=c,mx[k]=c,tag[k]=12345,lx[k]=rx[k]=(c>0?c:0),size[k]=1,rev[k]=0;}
74 int build(int l,int r) {
75     if(L>R)return 0;int mid=(L+R)>>1,now;
76     now=++sz;newnode(now,a[mid-1]);
77     tr[now][0]=build(l,mid-1);if(tr[now][0])fa[tr[now][0]]=now;
78     tr[now][1]=build(mid+1,r);if(tr[now][1])fa[tr[now][1]]=now;
79     pushup(now);return now;
80 }
81 int Build(int l,int r) {
82     if(L>R)return 0;int mid=(L+R)>>1,now;
83     if(top)now=st[top--];else now=++sz;newnode(now,a[mid]);
84     tr[now][0]=Build(l,mid-1);if(tr[now][0])fa[tr[now][0]]=now;
85     tr[now][1]=Build(mid+1,r);if(tr[now][1])fa[tr[now][1]]=now;
86     pushup(now);return now;
87 }
88 void insert(int x,int tot) {
89     for(int i=0;i<=tot+2;i++)a[i]=0;
90     for(int i=1;i<=tot;i++)a[i]=read();
91     int l=find(rt,x+1),r=find(rt,x+2);
92     splay(l,rt),splay(r,tr[l][1]);
93     tr[r][0]=Build(1,tot),fa[tr[r][0]]=r;
94     pushup(r),splay(r,rt);
95 }
96 void clr(int k){tag[k]=12345,tr[k][0]=tr[k][1]=fa[k]=rev[k]=v[k]=sum[k]=mx[k]=lx[k]=rx[k]=size[k]=0;}
97 void rec(int k) {
98     if(!k)return;
99     rec(tr[k][0]);rec(tr[k][1]);
100    st[++top]=k,clr(k);
101}
102void del(int x,int tot) {
103    int l=x,r=x+tot-1,k=split(l,r);
104    int fk=fa[k];tr[fk][0]=fa[k]=0;rec(k);
105    splay(fk,rt);
106}
107void make_same(int x,int tot,int c)
108{int l=x,r=x+tot-1,k=split(l,r);work(k,c);if(fa[k])splay(fa[k],rt);}
109void rever(int x,int tot)
110{int l=x,r=x+tot-1,k=split(l,r);rve(k);if(fa[k])splay(fa[k],rt);}
111int get_sum(int x,int tot) {
112    int l=x,r=x+tot-1,k=split(l,r);
113    return sum[k];
114}

```

## 59. 权值splay [sll/权值splay.cpp]

```

1  ll n,kind,rt,sz,fa[N],num[N];
2  ll tr[N][2],size[N],v[N],ans;
3  void pushup(ll k){size[k]=size[tr[k][0]]+size[tr[k][1]]+num[k];}
4  void rotate(ll x,ll &k) {
5      ll y=fa[x],z=fa[y],l,r;
6      l=tr[y][1]==x;r=L^1;
7      if(y==k)k=x;

```

```

8   else tr[z][tr[z][1]==y]=x;
9   fa[x]=z, fa[tr[x][r]]=y, fa[y]=x;
10  tr[y][l]=tr[x][r], tr[x][r]=y;
11  pushup(y); pushup(x);
12 }
13 void splay(ll x, ll &k) {
14   while(x!=k) {
15     ll y=fa[x], z=fa[y];
16     if(y!=k) {
17       if(tr[y][0]==x^tr[z][0]==y)
18         rotate(x, k);
19       else rotate(y, k);
20     } rotate(x, k);
21 }
22 void insert(ll &k, ll x, ll last) {
23   if(!k){k=++sz; v[k]=x; size[k]=num[k]=1; fa[k]=last; splay(k, rt); return ;}
24   if(x==v[k]) num[k]++;
25   else if(x>v[k]) insert(tr[k][1], x, k);
26   else insert(tr[k][0], x, k);
27 }
28 ll t1, t2;
29 ll find(ll x, ll k) {
30   if(!k) return 0;
31   if(x==v[k]) return k;
32   else if(x>v[k]) return find(x, tr[k][1]);
33   else return find(x, tr[k][0]);
34 }
35 void ask_before(ll x, ll k) {
36   if(!k) return ;
37   if(v[k]<x){t1=k; ask_before(x, tr[k][1]);}
38   else ask_before(x, tr[k][0]);
39 }
40 void ask_after(ll x, ll k) {
41   if(!k) return ;
42   if(v[k]>x){t2=k; ask_after(x, tr[k][0]);}
43   // else if(v[k]==x) return ;
44   else ask_after(x, tr[k][1]);
45 }
46 void del(ll x, ll k) {
47   if(num[k]>1) {
48     num[k]--, size[k]--;
49     splay(k, rt); return;
50   }
51   t1=t2=-1;
52   ask_before(x, rt);
53   ask_after(x, rt);
54   if(t1!=-1&t2!=-1) {
55     if(num[rt]==1) rt=0;
56     else size[rt]--, num[rt]--;
57   }
58   else if(t1!=-1) {
59     splay(t2, rt);
60     tr[rt][0]=0;
61     pushup(rt);
62   }
63   else if(t2!=-1) {
64     splay(t1, rt);
65     tr[rt][1]=0;

```

```

66   pushup(rt);
67 }
68 else {
69   splay(t1, rt);
70   splay(t2, tr[t1][1]);
71   tr[t2][0]=0;
72   pushup(t2); pushup(t1);
73 }}

```

## 60. Link-Cut Tree (splay) [xzl/lct-splay.cpp]

```

1 static struct Node {
2   int w, sum; //optional
3   int fa, lch, rch; bool rev;
4 } m[NMAX + 1];
5 inline void push(int x) {
6   if (m[x].rev) {
7     swap(m[x].lch, m[x].rch);
8     m[m[x].lch].rev ^= 1;
9     m[m[x].rch].rev ^= 1;
10    m[x].rev = 0;
11 }
12 inline void update(int x) { m[x].sum = m[x].w + m[m[x].lch].sum + m[m[x].rch].sum; }
13 inline void lrot(int x) {
14   int y = m[x].lch;
15   m[m[y].rch].fa = x;
16   m[x].lch = m[y].rch;
17   m[y].rch = x;
18   if (m[x].fa > 0) {
19     int p = m[x].fa;
20     if (m[p].lch == x) m[p].lch = y;
21     else m[p].rch = y;
22   }
23   m[y].fa = m[x].fa;
24   m[x].fa = y;
25   m[y].sum = m[x].sum;
26   update(x); //update(y);
27 }
28 inline void rrot(int x) {
29   int y = m[x].rch;
30   m[m[y].lch].fa = x;
31   m[x].rch = m[y].lch;
32   m[y].lch = x;
33   if (m[x].fa > 0) {
34     int p = m[x].fa;
35     if (m[p].lch == x) m[p].lch = y;
36     else m[p].rch = y;
37   }
38   m[y].fa = m[x].fa;
39   m[x].fa = y;
40   m[y].sum = m[x].sum;
41   update(x); //update(y);
42 }
43 inline void access(int x) {
44   if (m[x].fa > 0) access(m[x].fa);
45   push(x);
46 }
47 inline void splay(int x, bool accessed = false) {

```

```

48  if (!accessed) access(x);
49  while (m[x].fa > 0) {
50      int p = m[x].fa, p2 = m[p].fa;
51      if (p2 > 0) {
52          if (m[p].lch == x && m[p2].lch == p) lrot(p2);
53          else if (m[p].rch == x && m[p2].rch == p) rrot(p2);
54      }
55      if (m[p].lch == x) lrot(p);
56      else rrot(p);
57  }}
58  auto splice(int x) -> int {
59      int p = -m[x].fa;
60      splay(p);
61      m[m[p].rch].fa = -p;
62      m[p].rch = x;
63      m[x].fa = p;
64      update(p);
65      return p;
66  }
67  void expose(int x) {
68      splay(x);
69      m[m[x].rch].fa = -x;
70      m[x].rch = 0;
71      update(x);
72      while (m[x].fa) x = splice(x);
73  }
74  void link(int x, int y) {
75      splay(y);
76      m[y].fa = -x;
77      //expose(y);
78  }
79  void fastcut(int x) {
80      splay(x); //假定父亲已被 expose
81      m[x].fa = 0;
82  }
83  void cut(int x) {
84      expose(x);
85      splay(x);
86      int y = m[x].lch;
87      if (!y) return;
88      push(y);
89      while (m[y].rch) {
90          y = m[y].rch;
91          push(y);
92      }
93      splay(y, true);
94      m[m[y].rch].fa = 0;
95      m[y].rch = 0;
96      update(y);
97  }
98  void evert(int x) {
99      expose(x);
100     splay(x);
101     m[x].rev ^= 1;
102 }

```

## 61. Link-Cut Tree (treap) [xzl/lct-treap.cpp]

用处不大，主要是有 Treap 的 2-way join2(x, y)、3-way join(x, u, y) 和 3-way split(x)。注意初始化每个节点的 wt 和 size，以及 split 后节点 x 数据的重设。mrand 是 Xorshift 算法，比 C 标准库的 rand 快。

```

1  #define STACKSIZE 64
2  static struct Node {
3      int val, mx, pos; //optional
4      int wt, size, fa, lch, rch; bool rev;
5  } m[NMAX + 1];
6  inline void push(int x) {
7      if (m[x].rev) {
8          swap(m[x].lch, m[x].rch);
9          m[m[x].lch].rev ^= 1;
10         m[m[x].rch].rev ^= 1;
11         m[x].rev = 0;
12     }}
13  inline auto update(int x) -> int { /*...*/; return x; }
14  static auto join2(int x, int y) -> int {
15      if (!x) return y;
16      if (!y) return x;
17      int w = mrand(m[x].size + m[y].size);
18      if (w < m[x].size) {
19          push(x);
20          m[x].rch = join2(m[x].rch, y);
21          m[m[x].rch].fa = x;
22          return update(x);
23      }
24      push(y);
25      m[y].lch = join2(x, m[y].lch);
26      m[m[y].lch].fa = y;
27      return update(y);
28  }
29  static auto join(int x, int u, int y) -> int {
30      if (!x && !y) return u;
31      int w = mrand(m[x].size + m[u].wt + m[y].size);
32      if (w < m[x].size) {
33          push(x);
34          m[x].rch = join(m[x].rch, u, y);
35          m[m[x].rch].fa = x;
36          return update(x);
37      } else if (w >= m[x].size + m[u].wt) {
38          push(y);
39          m[y].lch = join(x, u, m[y].lch);
40          m[m[y].lch].fa = y;
41          return update(y);
42      }
43      m[u].lch = x;
44      m[u].rch = y;
45      m[x].fa = m[y].fa = u;
46      return update(u);
47  }
48  struct Triple { int l, r, p; };
49  static auto split(int x) -> Triple {
50      static int stk[STACKSIZE], tail = 0, y = x;
51      do {
52          stk[tail++] = y;
53          y = m[y].fa;

```



```

54 } while (y > 0);
55 for (int i = tail - 1; i >= 0; i--) push(stk[i]);
56 int l = m[x].lch, r = m[x].rch, t = m[stk[tail - 1]].fa;
57 for (int i = 1; i < tail; i++) {
58     int u = stk[i];
59     if (stk[i - 1] == m[u].lch) {
60         m[u].lch = r;
61         r = m[r].fa = u;
62     } else {
63         m[u].rch = l;
64         l = m[l].fa = u;
65     } update(u);
66 }
67 m[l].fa = m[r].fa = m[x].lch = m[x].rch = m[x].fa = 0;
68 //m[x].size = m[x].wt;
69 m[x].mx = m[x].val;
70 m[x].pos = x;
71 return {l, r, t};
72 }
73 #define REWEIGHT(x, d) \
74     m[x].wt += d; \
75     m[x].size = m[x].wt;
76 inline void reweight(int x, int d) {
77     auto t = split(x);
78     REWEIGHT(x, d);
79     x = join(t.l, x, t.r);
80     m[x].fa = t.p;
81 }
82 auto splice(int x) -> int {
83     int p = -m[x].fa;
84     auto t = split(p);
85     m[t.r].fa = -p;
86     REWEIGHT(p, m[t.r].size - m[x].size);
87     x = join(t.l, p, x);
88     m[x].fa = t.p;
89     return x;
90 }
91 void expose(int x) {
92     auto t = split(x);
93     m[t.r].fa = -x;
94     REWEIGHT(x, m[t.r].size);
95     x = join2(t.l, x);
96     m[x].fa = t.p;
97     while (m[x].fa) x = splice(x);
98 }
99 void link(int x, int y) {
100     while (m[y].fa > 0) y = m[y].fa;
101     expose(x);
102     m[y].fa = -x;
103     reweight(x, m[y].size);
104     //expose(y);
105 }
106 void fastcut(int x) {
107     while (m[x].fa > 0) x = m[x].fa;
108     if (m[x].fa) reweight(-m[x].fa, -m[x].size);
109     m[x].fa = 0;
110 }
111 void cut(int x) {

```

```

112 expose(x);
113 split(x);
114 m[x].size = m[x].wt;
115 }
116 void evert(int x) {
117     expose(x);
118     while (m[x].fa) x = m[x].fa;
119     m[x].rev ^= 1;
120 }

```

◦ xzl/preparation.md

试机时干什么

- 有题做题
- 抄模板测速
  - 浮点数运算速度 FFT
  - 取模速度 NTT
  - 后缀数组/后缀自动机
  - 快读 int
- 测试下评测机是如何工作的，是全部跑完再返回结果还是评测中途就会返回
- 试一下 `__int128_t`
- 试一下有没有 Presentation Error
- 询问：
  - 评测机配置
  - 栈空间限制
- 抄好的模板放 /tmp

◦ xzl/bashrc.md

```

alias cl="g++ -Wall -Wextra -Wno-char-subscripts -fmax-errors=6 -fsanitize=
undefined -std=c++11"
alias gdb="gdb -q"
alias bc="bc -q"

```

◦ xzl/manhattan.md

Manhattan 距离最小生成树：每  $45^\circ$  一个象限，对每个点找到每个象限中离它最近的点连边，然后做最小生成树。

优化：只用写找直线  $y = x$  与直线  $x = 0$  之间的最近点的代码，然后依次交换  $x$  和  $y$ 、取反  $y$ 、交换  $x$  和  $y$  一共做 4 次扫描线即可。

◦ xzl/maxdn.md

表格内的数据表示最坏情况。

$\log_{10} n$	1	2	3	4	5	6
$\omega(n)$	2	3	4	5	6	7
$d(n)$	4	12	32	64	128	240
$\log_{10} n$	7	8	9	10	11	12

$\log_{10} n$	7	8	9	10	11	12
$\omega(n)$	8	9	9	10	10	11
$d(n)$	448	768	1344	2304	4032	6720
$\log_{10} n$	13	14	15	16	17	18
$\omega(n)$	12	12	13	13	14	15
$d(n)$	10752	17280	26880	41472	64512	103680

- xzl/polar-sort.md
  - 极角排序：先按象限分后用叉积判断顺序。注意要分四个象限。
- xzl/spfa-opt.md
  - SPFA 优化。均为玄学，该卡掉的都可以卡掉。费用流时可以考虑一下。
  - SLF: 如果入队元素 `dist` 小于队首元素 `dist`，则加入队首。使用 `deque`。
  - SLF-swap: 如果入队后发现队尾元素 `dist` 小于队首元素 `dist`，则交换队首和队尾。避免使用双端队列。
  - LLL: 入队时与队内 `dist` 平均值做比较来决定是进队首或者队尾。使用 `deque`。（效果甚微）
- xzl/vimrc.md

```
set nocompatible
set smartindent
set tabstop=4
set softtabstop=4
set shiftwidth=4
set cursorline
set lines=50
set columns=100
set go=
set backspace=2
set mouse=a
set nu
set clipboard=unnamedplus
set ignorecase
set smartcase
map ; :
noremap H h
noremap J gj
noremap K gk
noremap L l
```
- xzl/fwt.md
  - FWT 算法：分治  $A \rightarrow A_1, A_2$ ，线性变换  $T$ ，合并时  $A = T[A_1, A_2]^T$ 。逆变换时取  $T$  的逆矩阵即可。

卷积类型	变换
------	----

卷积类型	变换
异或卷积	$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix}$
或卷积	$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$
和卷积	$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$

或卷积就是子集和变换。通过按子集大小分类可在  $O(n \log^2 n)$  时间内计算子集卷积：

```
for i = 0 → n - 1: // 按大小分类
    F[c][i] = f[i]
    G[c][i] = g[i]
for i = 0 → k - 1: // 提前计算 FWT
    F[i] = fwt(F[i])
    G[i] = fwt(G[i])
for i + j = k: // 卷积
    H[k] += F[i] · G[j]
for i in xrange(k): // FWT 逆变换
    H[i] = rfwt(H[i])
for all subset S: // 得到卷积结果
    R[i] = H[popcount(S)][i]
```

- lmj/Mo's\_algorithm.md
  - 带修莫队：把时间当成一维，排序时左右端点的块和时间一起排序，模拟时间。
  - 树上莫队：按照欧拉序，如果询问  $x, y$ ，若  $\text{lca}(x, y) = x$ ，则查询  $\text{st}[x]$  到  $\text{st}[y]$ ，否则  $\text{ed}[x], \text{st}[y]$ ，再加上  $\text{lca}$ ，出现两次的点不算。
- lmj/todo.md
  - 斯坦纳树
  - 连分数展开，在  $a/b$  到  $c/d$  之间分母最小的有理数，  
[https://en.wikipedia.org/wiki/Continued\\_fraction](https://en.wikipedia.org/wiki/Continued_fraction)  
[https://blog.csdn.net/lanchunhui/article/details/51719743?utm\\_source=blogxgwz4](https://blog.csdn.net/lanchunhui/article/details/51719743?utm_source=blogxgwz4)  
最佳渐进分数，Best rational within an interval，分子分母最小
  - 最大空凸包  
<http://picks.logdown.com/posts/209226-newtons-method-of-polynomial>
  - 牛顿迭代
  - exCRT
  - 线性基取交
  - 交也是线性空间。
  - 引理：如果  $V1, V2$  是线性空间，且  $B1, B2$  是基， $W = B2 \cap V1$ ，若  $B1 \cup (B2 - W)$  线性无关，则  $W$  是  $V1 \cap V2$  的基。
  - 证明：考虑任意的  $v \in V1 \cap V2$ ，那么  $v$  可以被  $B1, B2$  线性表示。考虑证明  $v$  可以被  $W$  线性表示。不妨令  $v$  可以被  $S$  和  $T$  共同线性表示，其中  $S \in W, T \in B - W$ ，显然，由于  $S$  可以被  $B1$

线性表示，如果  $T$  不为空，则  $T$  与  $B_1$  显然线性相关，与题目不符。因此  $v$  可以直接由  $W$  表示出。

但是显然， $B_1 \cup (B_2 \setminus W)$  有可能线性相关。于是其实我们只要换一组基，即把  $B_2$  换一下即可。

考虑  $b_i$  表示  $B_2$  中前  $i$  个向量组成的基，令我们新构造的基为  $\gamma$ ，第  $i$  个向量为  $\gamma_i$ ，则若  $B_2$  中第  $i$  个向量能够被  $b_i \setminus B_1$  表示出，不妨令它能够拆分为  $S+T$ ，其中  $S \setminus b_i \setminus T \setminus B_1$ ，我们令  $\gamma_i = S$ （或者  $T$ ）；否则的话  $\gamma_i$  就是  $B_2$  中的第  $i$  个向量。

显然，这样构造出的基满足  $B_1 \cup (\gamma \setminus W)$  线性无关，因此问题得以在  $O(d^3)$  的时间内解决。其中  $d$  为向量维数。

- 常系数线性递推  $k \log k \log n / k^2 \log n$   
<http://picks.logdown.com/posts/197262-polynomial-division>  
<https://wenku.baidu.com/view/bac23be1c8d376eeafaa3111.html>  
<https://www.cnblogs.com/Troywar/p/9078013.html>
- 数列前  $2k$  项可以暴力求，或者生成函数，然后算出闭合式之后多项式求逆
- [min\\_25筛](#)
- [定期重构](#)
- [拉格朗日插值](#)
- [lucas/exLucas](#)
- [pick定理](#)  
[https://en.wikipedia.org/wiki/Gaussian\\_binomial\\_coefficient](https://en.wikipedia.org/wiki/Gaussian_binomial_coefficient)
- [扩展欧拉定理](#)
- [lmj/number\\_theory.md](#) ■  
[反演/筛](#)
- [lmj/matrix\\_tree\\_theorem.md](#) ■  
 $K$ =度数矩阵-邻接矩阵， $K$ 的任意代数余子式（一般删最后一行一列，取正号）即为生成树数量。
- [lmj/dominator\\_tree.md](#) ■
- [lmj/virtual\\_tree.md](#) ■  
把需要的点按照dfs序排序，把相邻的lca求出来，塞进去重新排序，之后按照顺序维护当前的链，如果不是链就pop当前的点，在虚树上面加边。
- [lmj/bounded\\_flow.md](#) ■  
[无源汇可行流](#)  
[建模方法：](#)  
首先建立一个源 $ss$ 和一个汇 $tt$ ，一般称为附加源和附加汇。  
对于图中的每条弧，假设它容量上界为 $c$ ，下界 $b$ ，那么把这条边拆为三条只有上界的弧。  
一条为，容量为 $b$ ；  
一条为，容量为 $b$ ；  
一条为，容量为 $c-b$ 。  
其中前两条弧一般称为附加弧。

然后对这张图跑最大流，以 $ss$ 为源，以 $tt$ 为汇，如果所有的附加弧都满流，则原图有可行流。

这时，每条非附加弧的流量加上它的容量下界，就是原图中这条弧应该有的流量。  
[理解方法：](#)  
对于原图中的每条弧，我们把 $c-b$ 称为它的自由流量，意思就是只要它流满了下界，这些流多少都没问题。  
既然如此，对于每条弧，我们强制给 $v$ 提供 $b$ 单位的流量，并且强制从 $u$ 那里拿走 $b$ 单位的流量，这一步对应着两条附加弧。  
如果这一系列强制操作能完成的话，也就是有一组可行流了。  
[注意：](#)这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

---

[有源汇可行流](#)  
[建模方法：](#)  
建立弧，容量下界为 $0$ ，上界为 $\infty$ 。  
然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模，如果所有有附加弧满流，则存在可行流。  
[求原图中每条边对应的实际流量的方法](#)，同无源汇可行流，只是忽略掉弧就好。  
而且这时候弧的流量就是原图的总流量。  
[理解方法：](#)  
有源汇相比无源汇的不同就在于，源和汇是不满足流量平衡的，那么连接之后，源和汇也满足了流量平衡，就可以直接按照无源汇的方式建模。  
[注意：](#)这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

---

[有源汇最大流](#)  
[建模方法：](#)  
首先按照有源汇可行流的方法建模，如果不存在可行流，更别提什么最大流了。  
如果存在可行流，那么在运行过有源汇可行流的图上（就是已经存在流量的那张图，流量不要清零），跑一遍从 $s$ 到 $t$ 的最大流（这里的 $s$ 和 $t$ 是原图的源和汇，不是附加源和附加汇），就是原图的最大流。  
[理解方法：](#)  
为什么要在那个已经有了流量的图上跑最大流？因为那张图保证了每条弧的容量下界，在这张图上跑最大流，实际上就是在容量下界全部满足的前提下尽量多得获得“自由流量”。  
[注意：](#)在这张已经存在流量的图上，弧也是存在流量的，千万不要忽略这条弧。因为它的相反弧的流量为的流量的相反数，且的容量为 $0$ ，所以这部分的流量也是会被算上的。

---

[有源汇最小流](#)  
有源汇最小流的常见建模方法比较多，我就只说我常用的一种。  
[建模方法：](#)

首先按照有源汇可行流的方法建模，但是不要建立这条弧。

然后在这个图上，跑从附加源ss到附加汇tt的最大流。

这时候再添加弧，下界为0，上界为 $\infty$ 。

在现在的这张图上，从ss到tt的最大流，就是原图的最小流。

理解方法：

我们前面提到过，有源汇可行流的流量只是对应一组可行流，并不是最大或者最小流。

并且在跑完有源汇可行流之后，弧的流量就是原图的流量。

从这个角度入手，我们想让弧的流量尽量小，就要尽量多的消耗掉那些“本来不需要经过”的流量。

于是我们在添加之前，跑一遍从ss到tt的最大流，就能尽量多的消耗那些流量啦QwQ。

<https://www.cnblogs.com/mlystdcall/p/6734852.html>

• [lmj/cdq.md](#)

• [lmj/idea.md](#)

启发式合并

离线

hash

数据结构上跑图论算法

给每个点随机染色

mod合数时候可以尝试crt合并答案

曼哈顿距离和切比雪夫距离转换

$(x,y) \rightarrow (x+y,x-y)$  曼哈顿转切比雪夫,  $(x,y) \rightarrow ((x+y)/2,(x-y)/2)$  切比雪夫转曼哈顿

跑最短路时，如果边权一样的话，一定是先访问到的点松弛成功，后访问的点松弛失败。

(例如把一条边拆成多条边，如果终点一样的话，那么这条边只有第一次可能可以成功松弛)

平面图转对偶图，判连通性

【题目描述】：(opentrains10234b,bzoj4423)

比特哈顿镇有 $n*n$ 个格点，形成了一个网格图。一开始整张图是完整的。

有 $k$ 次操作，每次会删掉图中的一条边 $(u,v)$ ，你需要回答在删除这条边之后 $u$ 和 $v$ 是否仍然连通。

【题目解法】：

1.平面图

2.只有删边操作

转化为对偶图，即将每四个点包围的区域作为一个点

这样我们发现对于删边操作，相当于这些区域之间的加边操作

我们不妨给每一块区域编号，对于整个网格外面的部分把他们变成一样的号码

然后删边：

1.删除 $(x,y)$ 和 $(x,y+1)$ 的边，相当于把 $(x-1,y)$ 和 $(x,y)$ 这两个区域连起来

2.删除 $(x,y)$ 和 $(x+1,y)$ 的边，相当于把 $(x,y-1)$ 和 $(x,y)$ 这两个区域连起来

然后什么时候会导致两个点不联通呢？区域形成了环

我们发现如果区域成环，一定会把其中一个点包含在里面，区域的环相当于原图中这些边全部

断掉，那么这个点与环外界的点不联通，并查集维护即可

复杂度 $O(N^2)$

各种东西转成一般图最大匹配，也可能是费用流

Farey级数，stern-brocot树，具体数学上P96（中文版）

线段树区间线性基是三个log的！！

可以考虑线性基的前缀和，额外记录一下这个数被放进来的位置。

考虑线性基的插入过程，如果线性基当前位上已经有值，我们就不能把待插入的值放入这一位，因此线性基上每一位的数，都是对应位上在原数列最左侧的数字。现在我们改变策略，使得线性基上每一位的数，都变成对应位上在原数列最右侧的数字。实现这个策略的方法是：我们额外保存线性基上每一位数在原数列中的位置，插入的时候，如果对应位上的数在原数列中更靠左，就用待插入的数和它交换。基于这种策略，我们在查询区间 $[L, R]$ 时，可以在区间 $[1, R]$ 对应的线性基中查询，对于线性基上每一位的数，如果它在原数组中出现的位置比 $L$ 更靠右，就考虑它对答案的贡献，否则直接跳过这一位。

这个做法的正确性也很显然，通过改变策略，使线性基上每一位数变成对应位上在原数列最右侧的数字，可以看成线性基插入数字的顺序变反，完全不影响线性基的性质。同时，将线性基上所有在原数组中的位置比 $x$ 更靠左的数字删除，可以视为区间 $[1, L-1]$ 的数字还没有被插入线性基。

复杂度 $O(n \log n)$

<https://blog.csdn.net/tyxacm/article/details/97156620>

循环多项式和牛顿多项式，参考复旦高代书

写分数加法时候注意不要把分母乘起来，要先除掉gcd再算，防止爆longlong

• [lmj/gaussian\\_elimination.md](#)

在解非满秩的情况时，向下消完还要向上面消，这样消剩下的梯形矩阵里，如果一行只有一个未知数和变量就是有确定解，如果是几个变量之和，就是不能确定。同时也可以每次碰到的不能确定的变量当成自由元，这样剩下的变量可以由常数和自由元解出。注意现在消第 $i$ 行的时候不一定找第 $i$ 列的元素，可能是更大的某一列才有值。有些细节。

• [lmj/sam.md](#)

• [lmj/nim\\_k.md](#)

nim-k游戏

有 $n$ 堆石子，每次可以从至多 $k$ 堆中拿走任意数量的石子。（可以每堆拿的不一样）

不能拿的输。

先手必胜条件：把 $n$ 堆石子用二进制表示，统计每一位上面的1的个数，如果每一位1的个数 $\bmod (k+1)$ 全为0，则先手必败。否则先手必胜。

证明：类比一堆石子共 $n$ 个，每次去 $1-m$ 个，不能动为输。

（比较显然，考虑不全为0的时候，从高位到低位取成0，如果这位的数比之前少，那么可以在前面不全去完，而是剩一些，在这里加个1之类的，比之前多就把新的弄小

• [lmj/tree\\_divide\\_and\\_conquer\(edge\\_and\\_node\).md](#)

• [lmj/game.md](#)

各种游戏题

$n$  数码问题, 考虑把 0 去掉之后的逆序对数量, 如果是  $n \times n$ ,  $n$  为偶数的话, 还要加上每个数到正确的行需要的步数和。是偶数就可以恢复。

◦ [lmj/treehash.md](#)

$$\text{hash}[x] = A \cdot \prod_{v \text{ 是 } x \text{ 的儿子}} (\text{hash}[v] \oplus B) \pmod{C}$$

◦ [sll/扩展网络流.md](#)

无源汇有上下界可行流:

建图:

$M[i]$  = 流入  $i$  点的下界流量 - 流出  $i$  点的下界流量

$S \rightarrow i, c = M[i]$  ( $M[i] \geq 0$ )

$i \rightarrow T, c = -M[i]$

流程:

$S \rightarrow T$  跑最大流, 当  $S$  连出去的边满流是存在可行流

有源汇上下界最大流:

建图:

$T \rightarrow S$ , 流量限制为 (0, 无穷大), 转化成无源汇

增设  $ST$  和  $SD$ , 像无源汇那样连边

流程:

1.  $ST \rightarrow SD$  跑最大流, 判断是否满流, 不满流则无解

2. 去掉  $ST, SD$ , 从  $S \rightarrow T$  跑最大流, 两遍流量和为有源汇最大流量

有源汇上下界最小流:

建图: 同最大流

流程: 1. 同最大流

1. 去掉  $ST, SD, T \rightarrow S$  跑最大流, 两次流量之差为有源汇最小流

■ 最大权闭合子图:

问题描述: 求最大权值和的点集, 使得这个点集里的任一点的后继也在该点集中

建图: 原图中的  $(u \rightarrow v)$ , 建边  $(u \rightarrow v, \text{inf})$

对于  $c[u] > 0$  建边  $(S \rightarrow u, c[u])$

对于  $c[u] < 0$  建边  $(u \rightarrow T, -c[u])$

流程: 建图后跑  $S \rightarrow T$  的最小割,  $\sum c[u] (c[u] > 0)$  - 最小割即为答案

◦ [sll/欧拉路径.md](#)

欧拉路径:

```
def work(u):
    global e, top
    i = head[u]
    while i > 0:
        #print(str(i)+str(e[i].next))
        if e[i].c == 0:
            i = e[i].next
            continue
        v = e[i].to
        e[i].c = e[i^1].c = 0
        work(v)
        i = e[i].next
    st.append(u)
    top += 1
```