

# ICPC TEMPLATE

BJTU\_SpadeAce 冯大落

Last build at October 23, 2015

# Contents

<b>1</b>	<b>基础</b>	<b>2</b>
1.1	头文件 . . . . .	2
1.2	二进制 . . . . .	2
1.3	稀疏矩阵乘法 . . . . .	2
<b>2</b>	<b>图论</b>	<b>3</b>
2.1	2-SAT . . . . .	3
2.2	哈密尔顿回路 . . . . .	4
2.3	欧拉回路 . . . . .	5
2.4	DINIC . . . . .	5
2.5	费用流 . . . . .	7
2.6	最小路径覆盖最小点覆盖最大独立集 . . . . .	8
2.6.1	定义 . . . . .	8
2.6.2	算法 . . . . .	8
2.6.3	其他问题 . . . . .	8
2.7	匈牙利算法 . . . . .	8
<b>3</b>	<b>数据结构</b>	<b>9</b>
3.1	LCA . . . . .	9
3.2	RMQ . . . . .	10
3.3	树链剖分 . . . . .	10
3.4	区间第 k 大 . . . . .	12
3.5	二维线段树 . . . . .	14
3.5.1	点修改区间查询 . . . . .	15
<b>4</b>	<b>字符串</b>	<b>17</b>
4.1	KMP . . . . .	17
4.2	RKHash . . . . .	17
4.3	Manacher . . . . .	18
4.4	SA . . . . .	18
4.5	最小表示法 . . . . .	19
<b>5</b>	<b>数论</b>	<b>20</b>
5.1	欧拉函数 . . . . .	20
5.2	线性逆元 . . . . .	21
5.3	高阶等差数列 . . . . .	21
<b>6</b>	<b>数表</b>	<b>22</b>
6.1	数据范围 . . . . .	22
6.2	时间复杂度 . . . . .	22

# 1 基础

## 1.1 头文件

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <set>
5  #include <map>
6  #include <vector>
7  #include <queue>
8  #include <algorithm>
9  #include <cmath>
10 #include <sstream>
11 #include <fstream>
12 using namespace std;
13 #define LL long long
14 #define u32 unsigned int
15 #define AA first
16 #define BB second
17 typedef pair<int, int> PII;
18 #define cmin(x, y) x = min(x, y)
19 #define cmax(x, y) x = max(x, y)
20 #define PB(x) push_back(x)
21 #define SZ size()
22 #define MP(a, b) make_pair(a, b)
23 #define OP begin()
24 #define ED end()
25 #define CLR clear()
26 #define INS(x) insert(x)
27 #define FOR(i, n) for(int i = 0; i < (n); i++)
28 #define FORR(i, n) for(int i = 1; i <= (n); i++)
29 #define REP(i, L, R) for(int i = (L); i <= (R); i++)
30 #define MEM(a) memset(a, 0, sizeof a)
31 #define ECH(x) for(__typeof x.OP it = x.OP; it != x.ED; it++)
32 #define ONES(x) __builtin_popcount(x)
33 /*=====*/

```

## 1.2 二进制

\_\_builtin\_clz 前导 0  
 \_\_builtin\_ctz 后缀 0  
 \_\_builtin\_popcount 1 个数

## 1.3 稀疏矩阵乘法

```

1  int n; //Matrix Size
2  void matrixMul(LL a[][], LL b[][], LL c[][])
3  {
4      /*
5       * CLEAR c[][]
6       */
7      for(int i = 0; i < n; i++)

```

```

8      {
9          for(int j = 0; j < n; j++)
10         {
11             if(0 == a[i][j]) continue;
12             for(int k = 0; k < n; k++)
13                 c[i][k] += a[i][j] * b[j][k];
14         }
15     }
16 }

```

## 2 图论

### 2.1 2-SAT

```

1  #define bf(x) ((x)<<1)
2  #define bt(x) ((x)<<1|1)
3  int ind[N], id[N];
4  int t[M], nt[M];
5  int dfn[N], low[N];
6  int s[N];
7  bool vis[N];
8  int l;
9  int n;
10 int cnt, num, idn;
11 int a[N], b[N], c[N];
12
13 void add(int i, int j)
14 {
15     cnt++;
16     t[cnt] = j;
17     nt[cnt] = ind[i];
18     ind[i] = cnt;
19 }
20
21 void tarjan(int x)
22 {
23     num++;
24     dfn[x] = low[x] = num;
25     s[++l] = x;
26     vis[x] = true;
27     for(int k = ind[x]; k != -1; k = nt[k])
28     {
29         if(dfn[t[k]] == 0) tarjan(t[k]);
30         if(vis[t[k]]) low[x] = min(low[x], low[t[k]]);
31     }
32     if(dfn[x] == low[x])
33     {
34         idn++;
35         while(true)
36         {
37             int tmp = s[l--];
38             id[tmp] = idn;
39             vis[tmp] = false;
40             if(tmp == x) break;

```

```

41     }
42 }
43 }
44
45 bool chk(int M)
46 {
47     for(int i = 0; i < 2 * n; i++) ind[i] = -1;
48     cnt = 0;
49     /*
50     * ADD Edge
51     */
52     memset(vis, 0, sizeof vis);
53     num = 0;
54     idn = 0;
55     l = 0;
56     memset(dfn, 0, sizeof dfn);
57     memset(low, 0, sizeof low);
58     memset(id, 0, sizeof id);
59     for(int i = 0; i < 2 * n; i++)
60     {
61         if(dfn[i] == 0) tarjan(i);
62     }
63     for(int i = 0; i < 2 * n; i++)
64     {
65         if(id[i] == id[i ^ 1]) return false;
66     }
67     return true;
68 }

```

## 2.2 哈密尔顿回路

设  $dp[i][j]$  表示站在  $i$  点, 盘面状态为  $j$  时的最短路, 最后答案需要遍历所有  $j == (1 \ll n) - 1$  注意有的题需要用 Floyd 预处理最短路

```

1  int f[N][N];
2  int dp[N][(1 << 18)];
3  int n, m;
4
5  int main()
6  {
7      int T; scanf("%d", &T);
8      while(T--)
9      {
10         scanf("%d%d", &n, &m);
11         for(int i = 1; i <= n; i++) for(int j = 1; j <= n; j++) f[i][j] =
            INF;
12         for(int i = 1; i <= n; i++) f[i][i] = 0;
13         for(int i = 0; i < m; i++)
14         {
15             int x, y, z; scanf("%d%d%d", &x, &y, &z);
16             f[x][y] = min(f[x][y], z);
17             f[y][x] = min(f[y][x], z);
18         }
19         for(int k = 1; k <= n; k++) for(int i = 1; i <= n; i++) for(int j
            = 1; j <= n; j++)
20             f[i][j] = min(f[i][k] + f[k][j], f[i][j]);

```

```

21
22     for(int i = 1; i <= n; i++) for(int j = 0; j < 1 << (n + 1); j++)
        dp[i][j] = INF;
23     dp[1][2] = 0;
24
25     for(int j = 0; j < (1 << (n + 1)); j++)
26     {
27         for(int i = 1; i <= n; i++)
28             if(((1 << i) & j) != 0)
29             {
30                 for(int k = 1; k <= n; k++)
31                     dp[k][((1 << k) | j)] = min(dp[i][j] + f[i][k], dp[k]
                        ][((1 << k) | j)]);
32             }
33     }
34     int ans = INF;
35     for(int i = 1; i <= n; i++) { ans = min(ans, dp[i][((1 << (n + 1))
        - 2) + f[i][1]]); }
36     printf("%d\n", ans);
37 }
38 return 0;
39 }

```

## 2.3 欧拉回路

存在条件为奇数度的点 0 个 (无向图), 所有点入度等于出度 (有向图), 求欧拉回路需要注意访问过的边需要删除

## 2.4 DINIC

```

1  int f[N];
2  int q[N];
3  bool vis[N];
4  int h[N];
5  int ind[N];
6  int t[M], c[M], nt[M], opp[M];
7  int cnt;
8  int n, m;
9  int ST, ED;
10
11 int add(int a, int b, int C)
12 {
13     t[cnt] = b;
14     nt[cnt] = ind[a];
15     ind[a] = cnt;
16     c[cnt] = C;
17     return cnt++;
18 }
19
20 void add1(int a, int b, int c)
21 {
22     static int x, y;
23     x = add(a, b, c);

```

```
24     y = add(b, a, 0);
25     opp[x] = y; opp[y] = x;
26 }
27
28 bool bfs()
29 {
30     int l = 0, r = 0;
31     q[l] = ST;
32     memset(h, 0, sizeof h);
33     h[ST] = 1;
34     while(l <= r)
35     {
36         int x = q[l++];
37         for(int k = ind[x]; k != -1; k = nt[k])
38             if(!h[t[k]] && c[k] > 0)
39             {
40                 h[t[k]] = h[x] + 1;
41                 q[++r] = t[k];
42             }
43     }
44     return h[ED] != 0;
45 }
46
47 int dfs(int x, int p)
48 {
49     if(x == ED) return p;
50     bool flg = false;
51     int tot = 0;
52     for(int k = ind[x]; k != -1; k = nt[k])
53         if(h[t[k]] == h[x] + 1)
54         {
55             int d = dfs(t[k], min(p, c[k]));
56             if(d)
57             {
58                 p -= d;
59                 tot += d;
60                 c[k] -= d;
61                 flg = true;
62                 c[opp[k]] += d;
63             }
64         }
65     return tot;
66 }
67
68 int dinic()
69 {
70     int ans = 0, tmp;
71     while(bfs())
72     {
73         while(true)
74         {
75             int fw = dfs(ST, INF);
76             if(!fw) break;
77             else ans += fw;
```

```

78     }
79 }
80     return ans;
81 }

```

## 2.5 费用流

增广路版费用流，复杂度  $O(n^2m)$

```

1  #include <iostream>
2  #include <cstdio>
3  #include <cstring>
4  #include <algorithm>
5  using namespace std;
6  #define N 3000
7  #define M 100000
8  #define INF 0x7fffffff
9  #define LL long long
10
11 int ind[N], pre[N], f[N];
12 bool vis[N];
13 int bg[M], t[M], nt[M], c[M], op[M], v[M];
14 int cnt;
15 int S, T;
16 int n, m;
17
18 int add(int a, int b, int C, int V)
19 {
20     bg[cnt] = a;
21     t[cnt] = b;
22     v[cnt] = V;
23     c[cnt] = C;
24     nt[cnt] = ind[a];
25     ind[a] = cnt;
26     return cnt++;
27 }
28
29 int ADD(int a, int b, int c, int v)
30 {
31     int x = add(a, b, c, v);
32     int y = add(b, a, 0, -v);
33     op[x] = y; op[y] = x;
34     return x;
35 }
36
37 int h[N], q[N];
38
39 bool spfa()
40 {
41     memset(vis, 0, sizeof vis);
42     for(int i = S; i <= T; i++) f[i] = INF;
43     for(int i = S; i <= T; i++) pre[i] = -1;
44     pre[S] = -1;
45     f[S] = 0;
46     int l = 0, r = 0; q[l] = S; vis[S] = true;
47     while(l <= r)

```



```

48     {
49         int x = q[l % N];
50         l++; vis[x] = false;
51         for(int k = ind[x]; k != -1; k = nt[k])
52             if(c[k] > 0 && f[t[k]] > f[x] + v[k])
53             {
54                 f[t[k]] = f[x] + v[k];
55                 pre[t[k]] = k;
56                 if(!vis[t[k]])
57                 {
58                     r++;
59                     q[r % N] = t[k];
60                     vis[t[k]] = true;
61                 }
62             }
63     }
64     return pre[T] != -1;
65 }
66
67 int dinic()
68 {
69     LL ans = 0, tmp;
70     while(spfa())
71     {
72         ans += (LL)f[T] * dfs();
73     }
74     return ans;
75 }

```

## 2.6 最小路径覆盖最小点覆盖最大独立集

### 2.6.1 定义

对于图  $G=(V,E)$  中的一个点覆盖是一个集合  $S \subseteq V$  使得每一条边至少有一个端点在  $S$  中

### 2.6.2 算法

对于任意边  $(i,j)$ , 建立  $(i,j'), (j,i')$  两条边求最大匹配, 最后答案为  $\text{ans} / 2 + \text{ans} \bmod 2$

### 2.6.3 其他问题

对于二分图

最小路径覆盖 =  $|P|$  - 最大匹配数

最小点覆盖数 = 最大匹配数

最大独立顶点集 = 总顶点数 - 最大匹配数

## 2.7 匈牙利算法

kuangbin 模板, 边方向为  $u$  到  $v$ ,  $uN$  为  $u$  类节点个数,  $vN$  为  $v$  类节点个数

```

1
2  const int MAXN=1000;
3  int uN, vN;
4  int g[MAXN][MAXN];
5  int linker[MAXN];
6  bool used[MAXN];
7  bool dfs(int u)
8  {
9      int v;
10     for(v = 0; v < vN; v++)
11         if(g[u][v] && !used[v])
12             {
13                 used[v] = true;
14                 if(linker[v] == -1 || dfs(linker[v]))
15                     {
16                         linker[v] = u;
17                         return true;
18                     }
19             }
20     return false;
21 }
22 int hungary()
23 {
24     int res = 0;
25     int u;
26     memset(linker, -1, sizeof(linker));
27     for(u = 0; u < uN; u++)
28     {
29         memset(used, 0, sizeof(used));
30         if(dfs(u)) res++;
31     }
32     return res;
33 }

```

## 3 数据结构

### 3.1 LCA

```

1  const int POW = 18;
2  void dfs(int u, int fa){
3      d[u]=d[fa]+1;
4      p[u][0]=fa;
5      for(int i=1; i<POW; i++) p[u][i]=p[p[u][i-1]][i-1];
6      int sz=edge[u].size();
7      for(int i=0; i<sz; i++){
8          int v=edge[u][i];
9          if(v==fa) continue;
10         dfs(v, u);
11     }
12 }
13 int lca( int a, int b ){
14     if( d[a] > d[b] ) a ^= b, b ^= a, a ^= b;
15     if( d[a] < d[b] ){

```

```

16         int del = d[b] - d[a];
17         for( int i = 0; i < POW; i++ ) if(del & (1 << i)) b = p[b][i];
18     }
19     if( a != b ){
20         for( int i = POW-1; i >= 0; i-- )
21             if( p[a][i] != p[b][i] )
22                 a = p[a][i] , b = p[b][i];
23         a = p[a][0], b = p[b][0];
24     }
25     return a;
26 }

```

### 3.2 RMQ

```

1  int f[N][POW];
2  for(int j = 0; j < POW; j++)
3      for(int i = 0; i < n; i++)
4          if(i + (1 << (j + 1)) <= n)
5              {
6                  f[i][j + 1] = max(f[i][j], f[i + (1 << j)][j]);
7              }
8  int k = log2(r - l + 1);
9  ans = max(ans, max(f[l][k], f[r - qpow(2, k) + 1][k]));

```

### 3.3 树链剖分

$\text{siz}[v]$  表示以  $v$  为根的子树的节点数  $\text{dep}[v]$  表示  $v$  的深度  $\text{top}[v]$  表示  $v$  所在的重链的顶端节点  $\text{fa}[v]$  表示  $v$  的父亲  $\text{son}[v]$  表示与  $v$  在同一重链上的  $v$  的儿子节点  $\text{w}[v]$  表示  $v$  与其父亲节点的连边在线段树中的位置 初始需要调用  $\text{cnt1} = \text{cnt2} = \text{cnt3} = 0$ ;  $\text{dfs1}(\text{ROOT}, 0)$ ;  $\text{dfs2}(\text{ROOT}, 1)$ ;  $\text{bt}(1, \text{cnt2})$ ; 模板为边带权值, 点带权值需要修改  $\text{query}(x, y)$ ;  $\text{update}(x, p, c)$  的  $p$  为线段树中的编号, 更新  $x$  需要调用  $\text{w}[x]$

```

1  #define MID(x, y) (((x) + (y)) >> 1)
2
3  int fa[N], top[N], w[N], son[N], dep[N], sz[N], r[N];
4  int a[N], b[N];
5  LL c[N];
6  int ind[N];
7  int t[M], nt[M];
8  int cnt1, cnt2, cnt3;
9  int n, m;
10
11 struct node
12 {
13     int l, r;
14     int a, b;
15     LL sum;
16 } f[M];
17 int rt;
18
19 void dfs1(int x, int d)
20 {
21     dep[x] = d;
22     son[x] = 0;
23     sz[x] = 1;
24     for(int k = ind[x]; k != -1; k = nt[k])

```

```

25     if(t[k] != fa[x])
26     {
27         fa[t[k]] = x;
28         dfs1(t[k], d + 1);
29         sz[x] += sz[t[k]];
30         if(sz[t[k]] > sz[son[x]]) son[x] = t[k];
31     }
32 }
33
34 void dfs2(int x, int tt)
35 {
36     w[x] = ++cnt2;
37     top[x] = tt;
38     if(son[x]) dfs2(son[x], tt);
39     for(int k = ind[x]; k != -1; k = nt[k]) if(t[k] != fa[x] && t[k] !=
        son[x])
40         dfs2(t[k], t[k]);
41 }
42
43 LL add(int a, int b)
44 {
45     t[cnt1] = b;
46     nt[cnt1] = ind[a];
47     ind[a] = cnt1++;
48 }
49
50 void update(int x)
51 {
52     f[x].sum = f[f[x].l].sum + f[f[x].r].sum;
53 }
54
55 int bt(int a, int b)
56 {
57     int x = cnt3++;
58     f[x].a = a; f[x].b = b;
59     if(a < b)
60     {
61         int mid = MID(a, b);
62         f[x].l = bt(a, mid);
63         f[x].r = bt(mid + 1, b);
64         f[x].sum = 0;
65     }
66     else
67     {
68         f[x].sum = 0;
69     }
70     return x;
71 }
72
73 // Query On ST, Do not Call Directly
74 LL query(int x, int a, int b)
75 {
76     if(a <= f[x].a && f[x].b <= b) return f[x].sum;
77     int mid = MID(f[x].a, f[x].b);

```

```

78     LL ans = 0;
79     if(a <= mid) ans += query(f[x].l, a, b);
80     if(b > mid) ans += query(f[x].r, a, b);
81     return ans;
82 }
83
84 //Modify Point
85 void update(int x, int p, int cc)
86 {
87     if(f[x].a == f[x].b) { f[x].sum = cc; return; }
88     int mid = MID(f[x].a, f[x].b);
89     if(p <= mid) update(f[x].l, p, cc);
90     else update(f[x].r, p, cc);
91     update(x);
92 }
93
94 //Query Segment
95 LL query(int x, int y)
96 {
97     int fx = top[x], fy = top[y];
98     LL sum = 0;
99     while(fx != fy)
100     {
101         if(dep[fx] < dep[fy])
102         {
103             swap(x, y);
104             swap(fx, fy);
105         }
106         sum += query(rt, w[fx], w[x]);
107         x = fa[top[x]];
108         fx = top[x];
109     }
110     if(dep[x] > dep[y]) swap(x, y);
111     if(x == y) return sum;
112     return sum + query(rt, w[son[x]], w[y]);
113 }

```

### 3.4 区间第 k 大

静态区间第 k 大模板，每次查询 [l, r] 区间

```

1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  #include <cstring>
5  using namespace std;
6  #define N 100010
7  #define NN (30 * N)
8  namespace ST
9  {
10     int T[N];
11     struct node
12     {

```

```

13     int l, r, a, b;
14     int c;
15     node() {}
16 }f[NN];
17 int cnt;
18 int lb, ub;
19
20 int add(int a, int b, int l, int r, int c)
21 {
22     f[cnt].a = a; f[cnt].b = b; f[cnt].l = l; f[cnt].r = r; f[cnt].c
        = c;
23     return cnt++;
24 }
25
26 int build(int a, int b)
27 {
28     int rt = add(a, b, -1, -1, 0);
29     if(a < b)
30     {
31         int mid = (a + b) >> 1;
32         f[rt].l = build(a, mid);
33         f[rt].r = build(mid + 1, b);
34     }
35     return rt;
36 }
37
38 void init(int a, int b) { cnt = 0; T[b + 1] = build(a, b); lb = a; ub
    = b; }
39
40 int update(int x, int p)
41 {
42     int rt = add(lb, ub, -1, -1, f[x].c + 1), tmp = rt;
43     int l = lb, r = ub;
44     while(l < r)
45     {
46         int mid = (l + r) >> 1;
47         if(p <= mid)
48         {
49             f[rt].l = add(l, mid, -1, -1, 0);
50             f[rt].r = f[x].r;
51             rt = f[rt].l; x = f[x].l;
52             r = mid;
53         }
54         else
55         {
56             f[rt].r = add(mid + 1, r, -1, -1, 0);
57             f[rt].l = f[x].l;
58             rt = f[rt].r; x = f[x].r;
59             l = mid + 1;
60         }
61         f[rt].c = f[x].c + 1;
62     }
63     return tmp;
64 }

```

```

65
66     int qry(int lx, int rx, int k)
67     {
68         int l = lb, r = ub;
69         while(l < r)
70         {
71             int mid = (l + r) >> 1;
72             if(f[f[lx].l].c - f[f[rx].l].c >= k)
73             {
74                 rx = f[rx].l; lx = f[lx].l;
75                 r = mid;
76             }
77             else
78             {
79                 k -= (f[f[lx].l].c - f[f[rx].l].c);
80                 rx = f[rx].r; lx = f[lx].r;
81                 l = mid + 1;
82             }
83         }
84         return l;
85     }
86
87     int q(int l, int r, int k)
88     {
89         return qry(T[l], T[r + 1], k);
90     }
91 }
92
93 int n, m;
94 int h[N], a[N];
95
96 int main()
97 {
98     scanf("%d%d", &n, &m);
99     for(int i = 0; i < n; i++) scanf("%d", &a[i]);
100     memcpy(h, a, sizeof a);
101     sort(h, h + n);
102     int cnt = unique(h, h + n) - h;
103     for(int i = 0; i < n; i++) a[i] = lower_bound(h, h + n, a[i]) - h;
104
105     ST::init(0, cnt - 1);
106     for(int i = n - 1; i > -1; i--) ST::T[i] = ST::update(ST::T[i + 1], a[i]);
107     while(m--)
108     {
109         int i, j, k; scanf("%d%d%d", &i, &j, &k);
110         printf("%d\n", h[ST::q(i - 1, j - 1, k)]);
111     }
112     return 0;
113 }

```

### 3.5 二维线段树

## 3.5.1 点修改区间查询

```

1  int c[N][N];
2  int lx[N], ly[N];
3  int n, Q;
4  struct Node
5  {
6      struct node
7      {
8          int a, b;
9          int Min, Max;
10     }f[N];
11     int a, b;
12
13     void update(int x)
14     {
15         f[x].Min = min(f[x << 1].Min, f[x << 1 | 1].Min);
16         f[x].Max = max(f[x << 1].Max, f[x << 1 | 1].Max);
17     }
18
19     void build(int x, int a, int b)
20     {
21         f[x].a = a; f[x].b = b;
22         if(a < b)
23         {
24             int mid = (a + b) / 2;
25             build(x << 1, a, mid);
26             build(x << 1 | 1, mid + 1, b);
27             update(x);
28         } else { f[x].Min = f[x].Max = 0; ly[a] = x; }
29     }
30
31     int queryMin(int x, int a, int b)
32     {
33         if(a <= f[x].a && f[x].b <= b) return f[x].Min;
34         else
35         {
36             int mid = (f[x].a + f[x].b) / 2;
37             if(b <= mid) return queryMin(x << 1, a, b);
38             else if(a > mid) return queryMin(x << 1 | 1, a, b);
39             else return min(queryMin(x << 1, a, b), queryMin(x << 1 | 1,
40                 a, b));
41         }
42
43     int queryMax(int x, int a, int b)
44     {
45         if(a <= f[x].a && f[x].b <= b) return f[x].Max;
46         else
47         {
48             int mid = (f[x].a + f[x].b) / 2;
49             if(b <= mid) return queryMax(x << 1, a, b);
50             else if(a > mid) return queryMax(x << 1 | 1, a, b);
51             else return max(queryMax(x << 1, a, b), queryMax(x << 1 | 1,
                a, b));

```



```

52     }
53 }
54 }f[N];
55
56 void build(int x, int a, int b, int p, int q)
57 {
58     f[x].a = a; f[x].b = b;
59     f[x].build(1, p, q);
60     if(a < b)
61     {
62         int mid = (a + b) / 2;
63         build(x << 1, a, mid, p, q);
64         build(x << 1 | 1, mid + 1, b, p, q);
65     }
66     else lx[a] = x;
67 }
68
69 void update(int x, int y, int c)
70 {
71     x = lx[x]; y = ly[y];
72     for(int X = x; X != 0; X >>= 1)
73         for(int Y = y; Y != 0; Y >>= 1)
74         {
75             if(f[X].f[Y].a == f[X].f[Y].b)
76             {
77                 if(f[X].a == f[X].b)
78                 {
79                     f[X].f[Y].Min = f[X].f[Y].Max = c;
80                 }
81                 else
82                 {
83                     f[X].f[Y].Min = min(f[X << 1].f[Y].Min, f[X << 1 |
84                                     1].f[Y].Min);
85                     f[X].f[Y].Max = max(f[X << 1].f[Y].Max, f[X << 1 |
86                                     1].f[Y].Max);
87                 }
88             }
89             else { f[X].update(Y); }
90         }
91
92 int queryMin(int x, int a, int b, int p, int q)
93 {
94     if(a <= f[x].a && f[x].b <= b) return f[x].queryMin(1, p, q);
95     else
96     {
97         int mid = (f[x].a + f[x].b) / 2;
98         if(b <= mid) return queryMin(x << 1, a, b, p, q);
99         else if(a > mid) return queryMin(x << 1 | 1, a, b, p, q);
100        else return min(queryMin(x << 1, a, b, p, q), queryMin(x << 1 |
101                                1, a, b, p, q));
102    }
103 }

```

```

103 int queryMax(int x, int a, int b, int p, int q)
104 {
105     if(a <= f[x].a && f[x].b <= b) return f[x].queryMax(1, p, q);
106     else
107     {
108         int mid = (f[x].a + f[x].b) / 2;
109         if(b <= mid) return queryMax(x << 1, a, b, p, q);
110         else if(a > mid) return queryMax(x << 1 | 1, a, b, p, q);
111         else return max(queryMax(x << 1, a, b, p, q), queryMax(x << 1 |
112             1, a, b, p, q));
113     }
114 }

```

## 4 字符串

### 4.1 KMP

```

1 void getNext()
2 {
3     int j,k;
4     j = 0;
5     k = -1;
6     next[0] = -1;
7     while(j < m)
8     {
9         if(k == -1 || b[j] == b[k])
10             next[++j] = ++k;
11         else k = next[k];
12     }
13 }
14 // get First Position
15 int KMP_Index()
16 {
17     int i = 0, j = 0;
18     getNext();
19     while(i < n && j < m)
20     {
21         if(j == -1 || a[i] == b[j])
22         {
23             i++;
24             j++;
25         }
26         else j = next[j];
27     }
28     if(j == m) return i - m + 1;
29     else return -1;
30 }

```

### 4.2 RKHash

把字符串变成 X 进制数，可以完成  $O(1)$  比较，有一定概率冲突，调用 `init()` 初始化幂，`mh(0)` 构造 hash 数组

```

1 char s[N];
2 ULL hash[N];
3 int len;
4 ULL pp[N];
5
6 void init()
7 {
8     pp[0] = 1;
9     for(int i = 1; i < N; i++) pp[i] = (pp[i - 1] * 31ULL) % MOD;
10 }
11
12 void mh(int p)
13 {
14     if(!p) { hash[0] = (s[0] - 'a'); p++; }
15     for(int i = p; i < len; i++) hash[i] = ((hash[i - 1] * 31ULL) % MOD +
16         (s[i] - 'a')) % MOD;
17 }
18
19 int cp(int x, int y, int l)
20 {
21     if(x + l > len || y + l > len) return false;
22     x--; y--;
23     if((hash[x + l] - hash[x] * pp[l] % MOD + MOD) % MOD
24     == (hash[y + l] - hash[y] * pp[l] % MOD + MOD) % MOD) return true;
25     return false;
26 }

```

### 4.3 Manacher

求以  $i$  为中心的最长回文串长度，结果保存在  $pk[i]$  中，下标  $[0, n)$  开头、末尾、字符之间插入 #，例如 ababa 变为 #a#b#a#b#a#

```

1 int pk[N];
2 void manacher()
3 {
4     int mx = 0;
5     int p;
6     for(int i = 0; i < n; i++)
7     {
8         if(i < mx) pk[i] = min(pk[2 * p - i], mx - i);
9         else pk[i] = 1;
10        while(i + pk[i] < n && i - pk[i] > -1 && a[i + pk[i]] == a[i - pk
11            [i]]) pk[i]++;
12        if(i + pk[i] > mx) { p = i; mx = i + pk[i]; }
13    }
14 }

```

### 4.4 SA

倍增算法， $r$  为待匹配数组， $n$  为总长度 +1， $m$  为字符范围，`num` 保存字符串使用时注意 `num[]` 有效位为  $[0, n)$ ，但是需要将 `num[n] = 0`，另外，对于模板的处理将空串也处理了，作为

rank 最小的串, 因此有效串为  $[0, n]$  共  $n-1$  个, 在调用 `da()` 函数时, 需要调用 `da(num, n + 1, m)` 对于 `sa[]`, `rank[]`, `height[]` 数组都将空串考虑在内, 作为 rank 最小的后缀注意 rank, height 范围从  $[0, n]$

```

1
2 namespace SA
3 {
4     int len;
5     int num[N];
6     int sa[N], rank[N], height[N]; // sa[1~n]value(0~n-1); rank[0..n-1]
        value(1..n); height[2..n]
7     int wa[N], wb[N], wv[N], wd[N];
8
9     int cmp(int *r, int a, int b, int x) {
10         return r[a] == r[b] && r[a + x] == r[b + x];
11     }
12
13     void da(int *r, int n, int m) {
14         int i, j, k, p, *x = wa, *y = wb, *t;
15         for(i = 0; i < m; i++) wd[i] = 0;
16         for(i = 0; i < n; i++) wd[x[i] = r[i]]++;
17         for(i = 1; i < m; i++) wd[i] += wd[i - 1];
18         for(i = n - 1; i >= 0; i--) sa[--wd[x[i]]] = i;
19         for(j = 1, p = 1; p < n; j <= 1, m = p) {
20             for(p = 0, i = n - j; i < n; i++) y[p++] = i;
21             for(i = 0; i < n; i++) if(sa[i] >= j) y[p++] = sa[i] - j;
22             for(i = 0; i < n; i++) wv[i] = x[y[i]];
23             for(i = 0; i < m; i++) wd[i] = 0;
24             for(i = 0; i < n; i++) wd[wv[i]]++;
25             for(i = 1; i < m; i++) wd[i] += wd[i - 1];
26             for(i = n - 1; i >= 0; i--) sa[--wd[wv[i]]] = y[i];
27             for(t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i
                ++){
28                 x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;
29             }
30         }
31
32         for(i = 0, k = 0; i < n; i++) rank[sa[i]] = i;
33         for(i = 0; i < n - 1; height[rank[i++]] = k) {
34             for(k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k
                ++);
35         }
36     }
37 }

```

## 4.5 最小表示法

求一个字符串字典序最小的开头位置, 注意要复制串两次

```

1 int smallest(char *s)
2 {
3     int i = 0, j = 1, k = 0;
4     int len = strlen(s);
5     while(i < len && j < len)
6     {

```

```

7         k=0;
8         while(k<len&&(s[i+k]==s[j+k]))
9             k++;
10        if(k>=len)
11            break;
12        if(s[i+k]>s[j+k])
13            i=max(i+k+1,j+1);
14        else
15            j=max(i+1,j+k+1);
16    }
17    return min(i,j);
18 }
19
20 int biggest(char *s)
21 {
22     int i = 0, j = 1, k = 0;
23     int len = strlen(s);
24     while(i<len&&j<len)
25     {
26         k=0;
27         while(k<len&&(s[i+k]==s[j+k]))
28             k++;
29         if(k>=len)
30             break;
31         if(s[i+k]<s[j+k])
32             i=max(i+k+1,j+1);
33         else
34             j=max(i+1,j+k+1);
35     }
36     return min(i,j);
37 }

```

## 5 数论

### 5.1 欧拉函数

对正整数  $n$ ，欧拉函数是少于或等于  $n$  的数中与  $n$  互质的数的数目  
 $P$  是素数：

若  $p$  是  $x$  的约数，则  $E(x * p) = E(x) * p$

$p$  不是  $x$  的约数，则  $E(x * p) = E(x) * E(p) = E(x) * (p - 1)$

```

1  int prime[N];
2  int phi[N];
3  bool is_prime[N];
4
5  void get_phi()
6  {
7      int i, j, k;
8      k = 0;
9      for(i = 2; i < N; i++)
10     {
11         if(is_prime[i] == false)
12         {

```

```

13         prime[k++] = i;
14         phi[i] = i - 1;
15     }
16     for(j = 0; j < k && i * prime[j] < N; j++)
17     {
18         is_prime[ i * prime[j] ] = true;
19         if( i % prime[j] == 0)
20         {
21             phi[ i * prime[j] ] = phi[i] * prime[j];
22             break;
23         }
24         else
25         {
26             phi[ i * prime[j] ] = phi[i] * ( prime[j] - 1 );
27         }
28     }
29 }
30 }

```

## 5.2 线性逆元

$\text{inv}[i] = (\text{MOD} - \text{MOD} / i) * \text{inv}[\text{MOD} \% i] \% \text{MOD}$  适用于 MOD 是质数且  $\text{MOD} > n$  的情况, 能够  $O(n)$  时间求出  $1 \sim n$  对模 MOD 的逆

## 5.3 高阶等差数列

$$\sum n = \frac{1}{2}n(n+1) \quad (1)$$

$$\sum n^2 = \frac{1}{6}n(n+1)(2n+1) \quad (2)$$

$$\sum n^3 = \left(\sum n\right)^2 \quad (3)$$

$$\sum n^4 = \left(\sum n^2\right) \frac{1}{5}(3n^2 + 3n - 1) \quad (4)$$

$$\sum n^5 = \left(\sum n\right)^2 \frac{1}{3}(2n^2 + 2n - 1) \quad (5)$$

$$\sum n^6 = \left(\sum n^2\right) \frac{1}{7}(3n^4 + 6n^3 - 3n + 1) \quad (6)$$

$$\sum n^7 = \left(\sum n\right)^2 \frac{1}{6}(3n^4 + 6n^3 - n^2 - 4n + 2) \quad (7)$$

$$\sum n^8 = \left(\sum n^2\right) \frac{1}{15}(5n^6 + 15n^5 + 5n^4 - 15n^3 - n^2 + 9n - 3) \quad (8)$$

## 6 数表

### 6.1 数据范围

类型	字节	最小值	最大值
int	4	-2147483648	2147483647
unsigned	4	0	4294967295
long long	8	-9223372036854775808	9223372036854775807
unsigned long long	8	0	18446744073709551615
float	4	1.17549e-038	3.40282e+038
double	8	2.22507e-308	1.79769e+308
long double	12	3.3621e-4932	1.18973e+4932

Table 1: 数据范围

### 6.2 时间复杂度

x	$\log_2(x)$
10	3
1e2	6
1e3	9
1e4	13
1e5	16
1e6	19
1e7	23
1e8	26
1e9	29

Table 2:  $\log_2(x)$ 

n	n!
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800

Table 3: 阶乘