
ACM TEMPLATE

UESTC_Hime

Last build at October 24, 2013

Contents

1 Datastructure	2		
1.1 KD tree	2		
1.2 Binary indexed tree	2		
1.3 Splay	2		
1.4 Dynamic tree	3		
1.5 Partition tree	4		
2 Dynamic programming	5		
2.1 RMQ	5		
2.2 2D-LIS	5		
3 Geometry	5		
3.1 2D	5		
3.1.1 Point	5		
3.1.2 Line	5		
3.1.3 Functions	5		
3.1.4 Half plane intersection	6		
3.1.5 Convex hull	6		
3.1.6 Intersections of line and polygon	7		
3.2 3D	7		
3.2.1 Point	7		
3.2.2 Functions	8		
3.2.3 Convex hull	8		
3.3 Circle	9		
3.3.1 Functions	9		
3.3.2 Union	9		
3.3.3 Area of intersection part with polygon	10		
4 Graph	10		
4.1 Sap	10		
4.2 Minimal cost maximal flow	11		
4.3 Bi-connect	12		
4.4 Cut and bridge	13		
4.5 Global cut	13		
4.6 Euler path	14		
4.7 Strongly connected component	14		
4.8 Match	14		
4.8.1 Bipartite graph	14		
4.8.2 Edmonds	15		
4.8.3 KM	16		
4.9 Clique	16		
4.10 Spanning tree	17		
		4.10.1 Count the number of spanning tree	17
		4.10.2 Spanning tree on directed graph	17
5 Math	17		
5.1 FFT	17		
5.1.1 Usage	18		
5.2 Euler function	18		
5.3 Ex-GCD	18		
5.4 Prime	19		
5.4.1 Get primes	19		
5.4.2 Get factors	19		
5.5 Simpson	20		
5.6 Chinese remainder theorem	20		
5.7 Lucas	20		
5.8 Inverse element	20		
6 Search	20		
6.1 Dancing links	20		
6.1.1 Usage	21		
6.2 Dancing links (A-star)	21		
7 String	22		
7.1 Aho-Corasick automation	22		
7.2 KMP	23		
7.3 E-KMP	23		
7.4 Manacher	23		
7.5 Suffix array	23		
7.5.1 Longest common prefix	24		
7.6 Smallest representation	24		
8 Tool	24		
8.1 Bit compression	24		
8.2 Hash map	24		
8.3 120 bit integer	25		
8.4 Bash script	25		
8.5 Codeblocks settings	25		

1 Datastructure

1.1 KD tree

```

1 bool Div[MaxN];
2 void BuildKD(int deep,int l, int r, Point p[]) {
3     if (l > r) return;
4     int mid = l + r >> 1;
5     int minX, minY, maxX, maxY;
6     minX = min_element(p + l, p + r + 1, cmpX)->x;
7     minY = min_element(p + l, p + r + 1, cmpY)->y;
8     maxX = max_element(p + l, p + r + 1, cmpX)->x;
9     maxY = max_element(p + l, p + r + 1, cmpY)->y;
10    Div[mid] = (maxX - minX >= maxY - minY);
11    nth_element(p + l, p + mid, p + r + 1, Div[mid] ? cmpX : cmpY);
12    BuildKD(l, mid - 1, p);
13    BuildKD(mid + 1, r, p);
14 }
15 long long res;
16 void Find(int l, int r, Point a, Point p[]) {
17     if (l > r) return;
18     int mid = l + r >> 1;
19     long long dist = dist2(a, p[mid]);
20     if (dist > 0)//NOTICE
21         res = min(res, dist);
22     long long d = Div[mid] ? (a.x - p[mid].x) : (a.y - p[mid].y);
23     int l1, l2, r1, r2;
24     l1 = l, l2 = mid + 1;
25     r1 = mid - 1, r2 = r;
26     if (d > 0)
27         swap(l1, l2), swap(r1, r2);
28     Find(l1, r1, a, p);
29     if (d * d < res)
30         Find(l2, r2, a, p);
31 }

```

1.2 Binary indexed tree

```

1 int read(int k) {
2     int sum = 0;
3     for (; k; k^=k&-k) sum+=tree[k];
4     return sum;
5 }
6 void update(int k, int v) {
7     for (; k<=MaxN; k+=k&-k) tree[k]+=v;
8 }
9 int find_Kth(int k) {
10    int idx = 0;
11    for(int i=20; i>=0; i--) {
12        idx |= 1 << i;
13        if(idx <= MaxN && tree[idx] < k)
14            k -= tree[idx];
15        else idx ^= 1 << i;
16    }

```

```

17    return idx + 1;
18 }

```

1.3 Splay

```

1 //Node
2 struct Node {
3     int size,key;
4     Node *c[2], *p;
5 } mem[MaxN], *cur, *nil;
6 //Initialize functions without memory pool
7 Node *newNode(int v, Node *p) {
8     cur->c[0] = cur->c[1] = nil, cur->p = p;
9     cur->size = 1;
10    cur->key = v;
11    return cur++;
12 }
13 void Init() {
14     cur = mem;
15     nil = newNode(0, cur);
16     nil->size = 0;
17 }
18 //Splay tree
19 struct SplayTree {
20     Node *root;
21     void Init() {
22         root = nil;
23     }
24     void Pushup(Node *x) {
25         if (x == nil) return;
26         Pushdown(x);
27         Pushdown(x->c[0]);
28         Pushdown(x->c[1]);
29         x->size = x->c[0]->size + x->c[1]->size + 1;
30     }
31     void Pushdown(Node *x) {
32         if (x == nil) return;
33         //do something
34     }
35     void Rotate(Node *x, int f) {
36         if (x == nil) return;
37         Node *y = x->p;
38         y->c[f ^ 1] = x->c[f], x->p = y->p;
39         if (x->c[f] != nil)
40             x->c[f]->p = y;
41         if (y->p != nil)
42             y->p->c[y->p->c[1] == y] = x;
43         x->c[f] = y, y->p = x;
44         Pushup(y);
45     }
46     void Splay(Node *x, Node *f) {
47         static Node *stack[maxn];
48         int top = 0;
49         stack[top++] = x;

```

```

50   for (Node *y = x; y != f; y = y->p)
51       stack[top++] = y->p;
52   while (top)
53       Pushdown(stack[--top]);
54   while (x->p != f) {
55       Node *y = x->p;
56       if (y->p == f)
57           Rotate(x, x == y->c[0]);
58       else {
59           int fd = y->p->c[0] == y;
60           if (y->c[fd] == x)
61               Rotate(x, fd ^ 1), Rotate(x, fd);
62           else
63               Rotate(y, fd), Rotate(x, fd);
64       }
65   }
66   Pushup(x);
67   if (f == nil)
68       root = x;
69 }
70 void Select(int k, Node *f) {
71     Node *x = root;
72     Pushdown(x);
73     int tmp;
74     while ((tmp = x->c[0]->size) != k) {
75         if (k < tmp) x = x->c[0];
76         else
77             x = x->c[1], k -= tmp + 1;
78     }
79     Pushdown(x);
80     Splay(x, f);
81 }
82 void Select(int l, int r) {
83     Select(l, nil), Select(r + 2, root);
84 }
85 Node *Make_tree(int a[], int l, int r, Node *p) {
86     if (l > r) return nil;
87     int mid = l + r >> 1;
88     Node *x = newNode(a[mid], p);
89     x->c[0] = Make_tree(a, l, mid - 1, x);
90     x->c[1] = Make_tree(a, mid + 1, r, x);
91     Pushup(x);
92     return x;
93 }
94 void Insert(int pos, int a[], int n) {
95     Select(pos, nil), Select(pos + 1, root);
96     root->c[1]->c[0] = Make_tree(a, 0, n - 1, root->c[1]);
97     Splay(root->c[1]->c[0], nil);
98 }
99 void Insert(int v) {
100     Node *x = root, *y = nil;
101     //Need pushdown
102     while (x != nil) {
103         y = x;

```

```

104         y->size++;
105         x = x->c[v >= x->key];
106     }
107     y->c[v >= y->key] = x = newNode(v, y);
108     Splay(x, nil);
109 }
110 void Remove(int l, int r) {
111     Select(l, r);
112     //Recycle(root->c[1]->c[0]);
113     root->c[1]->c[0] = nil;
114     Splay(root->c[1], nil);
115 }
116 };

```

1.4 Dynamic tree

```

1 struct SplayTree {
2     void Pushup(Node *x) {
3         if (x == nil) return;
4         Pushdown(x);
5         Pushdown(x->c[0]);
6         Pushdown(x->c[1]);
7         x->size = x->c[0]->size + x->c[1]->size + 1;
8     }
9     void Pushdown(Node *x) {
10        if (x == nil) return;
11        if (x->rev) {
12            x->rev = 0;
13            x->c[0]->rev ^= 1;
14            x->c[1]->rev ^= 1;
15            swap(x->c[0], x->c[1]);
16        }
17    }
18    bool isRoot(Node *x) {
19        return (x == nil) || (x->p->c[0] != x && x->p->c[1] != x);
20    }
21    void Rotate(Node *x, int f) {
22        if (isRoot(x)) return;
23        Node *y = x->p;
24        y->c[f ^ 1] = x->c[f], x->p = y->p;
25        if (x->c[f] != nil)
26            x->c[f]->p = y;
27        if (y != nil) {
28            if (y == y->p->c[1])
29                y->p->c[1] = x;
30            else if (y == y->p->c[0])
31                y->p->c[0] = x;
32        }
33        x->c[f] = y, y->p = x;
34        Pushup(y);
35    }
36    void Splay(Node *x) {
37        static Node *stack[MaxN];
38        int top = 0;

```

```

39 stack[top++] = x;
40 for (Node *y = x; !isRoot(y); y = y->p)
41     stack[top++] = y->p;
42 while (top)
43     Pushdown(stack[--top]);
44 while (!isRoot(x)) {
45     Node *y = x->p;
46     if (isRoot(y))
47         Rotate(x, x == y->c[0]);
48     else {
49         int fd = y->p->c[0] == y;
50         if (y->c[fd] == x)
51             Rotate(x, fd ^ 1), Rotate(x, fd);
52         else
53             Rotate(y, fd), Rotate(x, fd);
54     }
55 }
56 Pushup(x);
57 }
58 Node *Access(Node *u) {
59     Node *v = nil;
60     while (u != nil) {
61         Splay(u);
62         v->p = u;
63         u->c[1] = v;
64         Pushup(u);
65         u = (v = u)->p;
66         if (u == nil)
67             return v;
68     }
69 }
70 Node *LCA(Node *u, Node *v) {
71     Access(u);
72     return Access(v);
73 }
74 Node *Link(Node *u, Node *v) {
75     Access(u);
76     Splay(u);
77     u->rev = true;
78     u->p = v;
79 }
80 void ChangeRoot(Node *u) {
81     Access(u)->rev ^= 1;
82 }
83 Node *GetRoute(Node *u, Node *v) {
84     ChangeRoot(u);
85     return Access(v);
86 }
87 };

```

1.5 Partition tree

```

1 int n,m;
2 struct elem {

```

```

3     int v,index;
4 } a[120000];
5 int d[30][120000];
6 int s[30][120000];
7 bool cmp(elem a,elem b) {
8     if (a.v == b.v)
9         return a.index <= b.index;
10    return a.v < b.v;
11 }
12 void build(int depth,int l,int r) {
13     if (l == r)
14         return;
15     int mid = (l+r)/2;
16     int tl,tr;
17     tl = tr = 0;
18     for (int i = l; i <= r; i++) {
19         if (cmp(a[d[depth][i]],a[mid])) {
20             d[depth+1][l+tl] = d[depth][i];
21             tl++;
22         } else {
23             d[depth+1][mid+1+tr] = d[depth][i];
24             tr++;
25         }
26     }
27     s[depth][i] = tl;
28     build(depth+1,l,mid);
29     build(depth+1,mid+1,r);
30 }
31 int find(int depth,int dl,int dr,int fl,int fr,int k) {
32     if (fl == fr)
33         return a[d[depth][fl]].v;
34     int ls,rs;
35     int mid = (dl+dr)/2;
36     ls = (fl == dl)? 0 : s[depth][fl-1];
37     rs = s[depth][fr];
38     return (rs-ls < k)?
39         find(depth+1,mid+1,dr,mid+fl-dl-ls+1,mid+fr-dl-rs+1,k-(rs-ls))
40         : find(depth+1,dl,mid,dl+ls,dl+rs-1,k);
41 }
42 int main() {
43     while (scanf("%d%d",&n,&m) != EOF) {
44         for (int i = 1; i <= n; i++) {
45             scanf("%d",&a[i].v);
46             a[i].index = i;
47         }
48         sort(a+1,a+n+1,cmp);
49         for (int i = 1; i <= n; i++)
50             d[0][a[i].index] = i;
51         build(0,1,n);
52         int l,r,k;
53         for (int i = 1; i <= m; i++) {
54             scanf("%d%d%d",&l,&r,&k);
55             printf("%d\n",find(0,1,n,l,r,k));
56         }

```

```

57 }
58 return 0;
59 }

```

2 Dynamic programming

2.1 RMQ

```

1 void init() {
2     int i,j;
3     int n=N,k=1,l=0;
4     for (i=0; i<n; i++) {
5         f[i][0]=ele[i].num;
6         if (i+1>k*2) {
7             k*=2;
8             l++;
9         }
10        lent[i+1]=l;
11    }
12    for (j=1; (1<<j)-1<n; j++)
13        for (i=0; i+(1<<j)-1<n; i++)
14            f[i][j]=max(f[i][j-1],f[i+(1<<(j-1))][j-1]);
15 }
16 int fint(int x,int y) {
17     int k=lent[y-x+1];
18     return max(f[x][k],f[y-(1<<k)+1][k]);
19 }

```

2.2 2D-LIS

```

1 #include<cstdio>
2 #include<map>
3 using namespace std;
4 map<int,int> mp[100001];
5 bool check(int idx,int x,int y) {
6     if (!idx) return 1;
7     if (mp[idx].begin()->first>=x) return 0;
8     map<int,int> ::iterator it=mp[idx].lower_bound(x);
9     it--;
10    if (it->second<y) return 1;
11    else return 0;
12 }
13 int main() {
14     int n;
15     scanf("%d",&n);
16     int l=0,r=0;
17     for (int i=0; i<n; i++) {
18         int x,y;
19         scanf("%d%d",&x,&y);
20         int tl=l,tr=r;
21         while (tl<tr) {
22             int mid=(tl+tr+1)/2;
23             if (check(mid,x,y))
24                 tl=mid;

```

```

25         else
26             tr=mid-1;
27     }
28     if (tl==r) r++;
29     int idx=tl+1;
30     map<int,int> ::iterator itl=mp[idx].lower_bound(x),itr=itl;
31     while (itr!=mp[idx].end() && itr->second>y) itr++;
32     if (mp[idx].find(x)!=mp[idx].end())
33         y=min(y,mp[idx][x]);
34     if (itl!=itr) mp[idx].erase(itl,itr);
35     if (mp[idx].find(x)==mp[idx].end() || mp[idx][x]>y)
36         mp[idx][x]=y;
37 }
38 printf("%d\n",r);
39 return 0;
40 }

```

3 Geometry

3.1 2D

3.1.1 Point

```

1 //Use cross product instead of atan2
2 bool cmp(const Point& a,const Point& b) {
3     if (a.y*b.y <= 0) {
4         if (a.y > 0 || b.y > 0) return a.y < b.y;
5         if (a.y == 0 && b.y == 0) return a.x < b.x;
6     }
7     return a*b > 0;
8 }

```

3.1.2 Line

```

1 Point operator &(const Line& b) const {
2     Point res = s;
3     double t = ((s - b.s) * (b.s - b.e)) / ((s - e) * (b.s - b.e));
4     res.x += (e.x - s.x) * t;
5     res.y += (e.y - s.y) * t;
6     return res;
7 }

```

3.1.3 Functions

```

1 Point nearestPointToLine(Point P, Line L) {
2     Point result;
3     double a, b, t;
4     a = L.e.x-L.s.x;
5     b = L.e.y-L.s.y;
6     t = ( (P.x-L.s.x)*a+(P.y-L.s.y)*b )/(a*a+b*b);
7     if (t >= 0 && t <= 1) {
8         result.x = L.s.x+a*t;
9         result.y = L.s.y+b*t;
10    }

```

```

11  return result;
12  }
13  //Segment
14  bool inter(Line l1,Line l2) {
15      return
16          max(l1.s.x,l1.e.x) >= min(l2.s.x,l2.e.x) &&
17          max(l2.s.x,l2.e.x) >= min(l1.s.x,l1.e.x) &&
18          max(l1.s.y,l1.e.y) >= min(l2.s.y,l2.e.y) &&
19          max(l2.s.y,l2.e.y) >= min(l1.s.y,l1.e.y) &&
20          sgn((l2.s-l1.s)*(l1.e-l1.s))*sgn((l2.e-l1.s)*(l1.e-l1.s)) <= 0 &&
21          sgn((l1.s-l2.s)*(l2.e-l2.s))*sgn((l1.e-l2.s)*(l2.e-l2.s)) <= 0;
22  }
23  bool onSeg(Line a,Point b) {
24      return ((a.s-b)*(a.e-b) == 0 &&
25          (b.x-a.s.x)*(b.x-a.e.x) <= 0 &&
26          (b.y-a.s.y)*(b.y-a.e.y) <= 0);
27  }
28  int inPoly(Point p,Point poly[], int n) {
29      int i, count;
30      Line ray, side;
31      count = 0;
32      ray.s = p;
33      ray.e.y = p.y;
34      ray.e.x = -1;//-∞
35      for (i = 0; i < n; i++) {
36          side.s = poly[i];
37          side.e = poly[(i+1)%n];
38          if(OnSeg(p, side))
39              return 1;
40          if (side.s.y == side.e.y)
41              continue;
42          if (OnSeg(side.s, ray)) {
43              if (side.s.y > side.e.y) count++;
44          } else if (OnSeg(side.e, ray)) {
45              if (side.e.y > side.s.y) count++;
46          } else if (inter(ray, side)) {
47              count++;
48          }
49      }
50      return ((count % 2 == 1) ? 0 : 2);
51  }
52  Point centerOfPolygon(Point poly[],int n) {
53      Point p, p0, p1, p2, p3;
54      double m, m0;
55      p1 = poly[0];
56      p2 = poly[1];
57      p.x = p.y = m = 0;
58      for (int i = 2; i < n; i++) {
59          p3 = poly[i];
60          p0.x = (p1.x + p2.x + p3.x) / 3.0;
61          p0.y = (p1.y + p2.y + p3.y) / 3.0;
62          m0 = p1.x*p2.y+p2.x*p3.y+p3.x*p1.y-p1.y*p2.x-p2.y*p3.x-p3.y*p1.x;
63          if (cmp(m + m0,0.0) == 0)
64              m0 += eps;

```

```

65      p.x = (m * p.x + m0 * p0.x) / (m + m0);
66      p.y = (m * p.y + m0 * p0.y) / (m + m0);
67      m = m + m0;
68      p2 = p3;
69  }
70  return p;
71  }

```

3.1.4 Half plane intersection

```

1  bool HPIcmp(Line a, Line b) {
2      if (fabs(a.k - b.k) > EPS) return a.k < b.k;
3      return ((a.s - b.s) * (b.e - b.s)) < 0;
4  }
5  Line Q[MAXN];
6  void HPI(Line line[], int n, Point res[], int &resn) {
7      int tot = n;
8      sort(line, line + n, HPIcmp);
9      tot = 1;
10     for (int i = 1; i < n; i++)
11         if (fabs(line[i].k - line[i - 1].k) > EPS)
12             line[tot++] = line[i];
13     int head = 0, tail = 1;
14     Q[0] = line[0];
15     Q[1] = line[1];
16     resn = 0;
17     for (int i = 2; i < tot; i++) {
18         if (fabs((Q[tail].e - Q[tail].s) * (Q[tail - 1].e - Q[tail - 1].s)) <
19             EPS ||
20             fabs((Q[head].e - Q[head].s) * (Q[head + 1].e - Q[head + 1].s)) <
21             EPS)
22             return;
23         while (head < tail && (((Q[tail] & Q[tail - 1]) - line[i].s) * (line[i].
24             e - line[i].s)) > EPS)
25             tail--;
26         while (head < tail && (((Q[head] & Q[head + 1]) - line[i].s) * (line[i].
27             e - line[i].s)) > EPS)
28             head++;
29         Q[++tail] = line[i];
30     }
31     while (head < tail && (((Q[tail] & Q[tail - 1]) - Q[head].s) * (Q[head].e
32         - Q[head].s)) > EPS)
33         tail--;
34     while (head < tail && (((Q[head] & Q[head + 1]) - Q[tail].s) * (Q[tail].e
35         - Q[tail].s)) > EPS)
36         head++;
37     if (tail <= head + 1) return;
38     for (int i = head; i < tail; i++)
39         res[resn++] = Q[i] & Q[i + 1];
40     if (head < tail + 1)
41         res[resn++] = Q[head] & Q[tail];
42 }

```

3.1.5 Convex hull

```

1 bool GScmp(Point a, Point b) {
2     if (fabs(a.x - b.x) < eps)
3         return a.y < b.y - eps;
4     return a.x < b.x - eps;
5 }
6 void GS(Point p[],int n,Point res[],int &resn) {
7     resn = 0;
8     int top = 0;
9     sort(p,p+n,GScmp);
10    if (conPoint(p,n)) {
11        res[resn++] = p[0];
12        return;
13    }
14    if (conLine(p,n)) {
15        res[resn++] = p[0];
16        res[resn++] = p[n-1];
17        return;
18    }
19    for (int i = 0; i < n;)
20        if (resn < 2 ||
21            (res[resn-1]-res[resn-2])*(p[i]-res[resn-1]) > 0)
22            res[resn++] = p[i++];
23        else
24            —resn;
25    top = resn-1;
26    for (int i = n-2; i >= 0;)
27        if (resn < top+2 ||
28            (res[resn-1]-res[resn-2])*(p[i]-res[resn-1]) > 0)
29            res[resn++] = p[i—];
30        else
31            —resn;
32    resn—;
33 }

```

3.1.6 Intersections of line and polygon

```

1 //Intersecting segment between [la,lb]
2 int Gao(int la,int lb,Line line) {
3     if (la > lb)
4         lb += n;
5     int l = la,r = lb,mid;
6     while (l < r) {
7         mid = l+r+1>>1;
8         if (cmp((line.e-line.s)*(p[la]-line.s),0)*cmp((line.e-line.s)*(p[mid]-
9             line.s),0) >= 0)
10            l = mid;
11        else
12            r = mid-1;
13    }
14    return l%n;
15 }
16 double theta[maxn];
17 void Gettheta() {
18     for (int i = 0; i < n; i++) {
19         Point v = p[(i+1)%n]-p[i];

```

```

19         theta[i] = atan2(v.y,v.x);
20     }
21     for (int i = 1; i < n; i++)
22         if (theta[i-1] > theta[i]+eps)
23             theta[i] += 2*pi;
24 }
25 void Calc(Line l) {
26     double tnow;
27     Point v = l.e-l.s;
28     tnow = atan2(v.y,v.x);
29     if (cmp(tnow,theta[0]) < 0) tnow += 2*pi;
30     int pl = lower_bound(theta,theta+n,tnow)-theta;
31     tnow = atan2(-v.y,-v.x);
32     if (cmp(tnow,theta[0]) < 0) tnow += 2*pi;
33     int pr = lower_bound(theta,theta+n,tnow)-theta;
34     //Farest points with l on polygon
35     pl = pl%n;
36     pr = pr%n;
37     if (cmp(v*(p[pl]-l.s),0)*cmp(v*(p[pr]-l.s),0) >= 0)
38         return 0.0;
39     int xa = Gao(pl,pr,l);
40     int xb = Gao(pr,pl,l);
41     if (xa > xb) swap(xa,xb);
42     //Intersecting with line  $P_{xa} \rightarrow P_{xa+1}$  and  $P_{xb} \rightarrow P_{xb+1}$ 
43     if (cmp(v*(p[xa+1]-p[xa]),0) == 0) return 0.0;
44     if (cmp(v*(p[xb+1]-p[xb]),0) == 0) return 0.0;
45     Point pa,pb;
46     //Intersections
47     pa = Line(p[xa],p[xa+1])&l;
48     pb = Line(p[xb],p[xb+1])&l;
49 }

```

3.2 3D

3.2.1 Point

```

1 Point3D operator *(const Point3D& b)const {
2     return Point3D(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
3 }
4 //Rotate around V, notice that |V|=1
5 Point3D Trans(Point3D pa,Point3D V,double theta) {
6     double s = sin(theta);
7     double c = cos(theta);
8     double x,y,z;
9     x = V.x;
10    y = V.y;
11    z = V.z;
12    Point3D pp =
13        Point3D(
14            (x*x*(1-c)+c)*pa.x+(x*y*(1-c)-z*s)*pa.y+(x*z*(1-c)+y*s)*pa.z,
15            (y*x*(1-c)+z*s)*pa.x+(y*y*(1-c)+c)*pa.y+(y*z*(1-c)-x*s)*pa.z,
16            (x*z*(1-c)-y*s)*pa.x+(y*z*(1-c)+x*s)*pa.y+(z*z*(1-c)+c)*pa.z);
17    return pp;
18 }

```


3.2.2 Functions

```

1 bool lineIntersect(Line3D L1, Line3D L2) {
2     Point3D s = L1.s-L1.e;
3     Point3D e = L2.s-L2.e;
4     Point3D p = s*e;
5     if (ZERO(p)) return false;    //Parallel
6     p = (L2.s-L1.e)*(L1.s-L1.e);
7     return ZERO(p&L2.e);        //Common face
8 }
9 //Please check whether a, b, c, d on a plane first
10 bool segmentIntersect(Point a,Point b,Point c,Point d) {
11     Point ret = (a-b)*(c-d);
12     Point t1 = (b-a)*(c-a);
13     Point t2 = (b-a)*(d-a);
14     Point t3 = (d-c)*(a-c);
15     Point t4 = (d-c)*(b-c);
16     return sgn(t1&ret)*sgn(t2&ret) < 0 &&
17         sgn(t3&ret)*sgn(t4&ret) < 0;
18 }
19 //Distance from point p to line L
20 double distance(Point3D p, Line3D L) {
21     return (Norm((p-L.s)*(L.e-L.s))/Norm(L.e-L.s));
22 }
23 //Angle between line L1 and L2,  $\theta \in [0, \pi]$ 
24 double calcTheta(Line3D L1, Line3D L2) {
25     Point3D u = L1.e - L1.s;
26     Point3D v = L2.e - L2.s;
27     return acos( (u & v) / (Norm(u)*Norm(v)) );
28 }

```

3.2.3 Convex hull

```

1 struct pt {
2     double x, y, z;
3     pt() {}
4     pt(double _x, double _y, double _z): x(_x), y(_y), z(_z) {}
5     pt operator - (const pt p1) {}
6     pt operator * (pt p) {}
7     double operator ^ (pt p) {}
8 };
9 struct _3DCH {
10     struct fac {
11         int a, b, c;
12         bool ok;
13     };
14     int n;
15     pt P[MAXV];
16     int cnt;
17     fac F[MAXV*8];
18     int to[MAXV][MAXV];
19     double vlen(pt a) {
20         return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
21     }

```

```

22 double area(pt a, pt b, pt c) {
23     return vlen((b-a)*(c-a));
24 }
25 double volume(pt a, pt b, pt c, pt d) {
26     return (b-a)*(c-a)^(d-a);
27 }
28 double ptof(pt &p, fac &f) {
29     pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
30     return (m * n) ^ t;
31 }
32 void deal(int p, int a, int b) {
33     int f = to[a][b];
34     fac add;
35     if (F[f].ok) {
36         if (ptof(P[p], F[f]) > eps)
37             dfs(p, f);
38     } else {
39         add.a = b, add.b = a, add.c = p, add.ok = 1;
40         to[p][b] = to[a][p] = to[b][a] = cnt;
41         F[cnt++] = add;
42     }
43 }
44 }
45 void dfs(int p, int cur) {
46     F[cur].ok = 0;
47     deal(p, F[cur].b, F[cur].a);
48     deal(p, F[cur].c, F[cur].b);
49     deal(p, F[cur].a, F[cur].c);
50 }
51 bool same(int s, int t) {
52     pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
53     return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b, c,
54         P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c])) < eps;
55 }
56 void construct() {
57     cnt = 0;
58     if (n < 4)
59         return;
60     bool sb = 1;
61     for (int i = 1; i < n; i++) {
62         if (vlen(P[0] - P[i]) > eps) {
63             swap(P[1], P[i]);
64             sb = 0;
65             break;
66         }
67     }
68     if (sb) return;
69     sb = 1;
70     for (int i = 2; i < n; i++) {
71         if (vlen((P[0] - P[1]) * (P[1] - P[i])) > eps) {
72             swap(P[2], P[i]);
73             sb = 0;
74             break;
75         }

```

```

76 }
77 if (sb)return;
78 sb = 1;
79 for (int i = 3; i < n; i++) {
80     if (fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) > eps) {
81         swap(P[3], P[i]);
82         sb = 0;
83         break;
84     }
85 }
86 if (sb)return;
87 fac add;
88 for (int i = 0; i < 4; i++) {
89     add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;
90     if (ptof(P[i], add) > 0)
91         swap(add.b, add.c);
92     to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
93     F[cnt++] = add;
94 }
95 for (int i = 4; i < n; i++) {
96     for (int j = 0; j < cnt; j++) {
97         if (F[j].ok && ptof(P[i], F[j]) > eps) {
98             dfs(i, j);
99             break;
100         }
101     }
102 }
103 int tmp = cnt;
104 cnt = 0;
105 for (int i = 0; i < tmp; i++) {
106     if (F[i].ok) {
107         F[cnt++] = F[i];
108     }
109 }
110 }
111 double area() {
112     double ret = 0.0;
113     for (int i = 0; i < cnt; i++) {
114         ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);
115     }
116     return ret / 2.0;
117 }
118 double volume() {
119     pt O(0, 0, 0);
120     double ret = 0.0;
121     for (int i = 0; i < cnt; i++) {
122         ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);
123     }
124     return fabs(ret / 6.0);
125 }
126 int facetCnt_tri() {
127     return cnt;
128 }
129 int facetCnt() {

```

```

130 int ans = 0;
131 for (int i = 0; i < cnt; i++) {
132     bool nb = 1;
133     for (int j = 0; j < i; j++) {
134         if (same(i, j)) {
135             nb = 0;
136             break;
137         }
138     }
139     ans += nb;
140 }
141 return ans;
142 }
143 pt Fc[MAXV*8];
144 double V[MAXV*8];
145 pt Center() {
146     pt O(0,0,0);
147     for (int i = 0; i < cnt; i++) {
148         Fc[i].x = (O.x+P[F[i].a].x+P[F[i].b].x+P[F[i].c].x)/4.0;
149         Fc[i].y = (O.y+P[F[i].a].y+P[F[i].b].y+P[F[i].c].y)/4.0;
150         Fc[i].z = (O.z+P[F[i].a].z+P[F[i].b].z+P[F[i].c].z)/4.0;
151         V[i] = volume(O,P[F[i].a],P[F[i].b],P[F[i].c]);
152     }
153     pt res = Fc[0],tmp;
154     double m = V[0];
155     for (int i = 1; i < cnt; i++) {
156         if (fabs(m+V[i]) < eps)
157             V[i] += eps;
158         tmp.x = (m*res.x+V[i]*Fc[i].x)/(m+V[i]);
159         tmp.y = (m*res.y+V[i]*Fc[i].y)/(m+V[i]);
160         tmp.z = (m*res.z+V[i]*Fc[i].z)/(m+V[i]);
161         m += V[i];
162         res = tmp;
163     }
164     return res;
165 }
166 };

```

3.3 Circle

3.3.1 Functions

```

1 //Common area of two circle
2 double area(int x1,int y1,int x2,int y2,double r1,double r2) {
3     double s=dis(x2-x1,y2-y1);
4     if(r1+r2<s) return 0;
5     else if(r2-r1>s) return PI*r1*r1;
6     else if(r1-r2>s) return PI*r2*r2;
7     double q1=acos((r1*r1+s*s-r2*r2)/(2*r1*s));
8     double q2=acos((r2*r2+s*s-r1*r1)/(2*r2*s));
9     return (r1*r1*q1+r2*r2*q2-r1*s*sin(q1));
10 }

```

3.3.2 Union

```

1  for (int i = 1; i <= n; i++)
2      ans[i] = 0.0;
3  for (int i = 0; i < n; i++) {
4      tote = 0;
5      e[tote++] = Event(-pi,1);
6      e[tote++] = Event(pi,-1);
7      for (int j = 0; j < n; j++)
8          if (j != i) {
9              lab = Point(c[j].c.x-c[i].c.x,c[j].c.y-c[i].c.y);
10             AB = lab.Length();
11             AC = c[i].r;
12             BC = c[j].r;
13             if (cmp(AB+AC,BC) <= 0) {
14                 e[tote++] = Event(-pi,1);
15                 e[tote++] = Event(pi,-1);
16                 continue;
17             }
18             if (cmp(AB+BC,AC) <= 0) continue;
19             if (cmp(AB,AC+BC) > 0) continue;
20             theta = atan2(lab.y,lab.x);
21             fai = acos((AC*AC+AB*AB-BC*BC)/(2.0*AC*AB));
22             a0 = theta-fai;
23             if (cmp(a0,-pi) < 0) a0 += 2*pi;
24             a1 = theta+fai;
25             if (cmp(a1,pi) > 0) a1 -= 2*pi;
26             if (cmp(a0,a1) > 0) {
27                 e[tote++] = Event(a0,1);
28                 e[tote++] = Event(pi,-1);
29                 e[tote++] = Event(-pi,1);
30                 e[tote++] = Event(a1,-1);
31             } else {
32                 e[tote++] = Event(a0,1);
33                 e[tote++] = Event(a1,-1);
34             }
35         }
36     sort(e,e+tote,Eventcmp);
37     cur = 0;
38     for (int j = 0; j < tote; j++) {
39         if (cur != 0 && cmp(e[j].tim,pre[cur]) != 0) {
40             ans[cur] += Area(e[j].tim-pre[cur],c[i].r);
41             ans[cur] += xmult(Point(c[i].c.x+c[i].r*cos(pre[cur]),c[i].c.y+c[i].r*
42                 sin(pre[cur])),
43                 Point(c[i].c.x+c[i].r*cos(e[j].tim),c[i].c.y+c[i].r*
44                 sin(e[j].tim)))/2.0;
45         }
46         cur += e[j].typ;
47         pre[cur] = e[j].tim;
48     }
49     for (int i = 1; i < n; i++)
50         ans[i] -= ans[i+1];

```

3.3.3 Area of intersection part with polygon

```

1  bool InCircle(Point a,double r) {

```

```

2      return cmp(a.x*a.x+a.y*a.y,r*r) <= 0;
3      //e should big enough
4  }
5  double CalcArea(Point a,Point b,double r) {
6      Point p[4];
7      int tot = 0;
8      p[tot++] = a;
9      Point tv = Point(a,b);
10     Line tmp = Line(Point(0,0),Point(tv.y,-tv.x));
11     Point near = LineToLine(Line(a,b),tmp);
12     if (cmp(near.x*near.x+near.y*near.y,r*r) <= 0) {
13         double A,B,C;
14         A = near.x*near.x+near.y*near.y;
15         C = r;
16         B = C*C-A;
17         double tvl = tv.x*tv.x+tv.y*tv.y;
18         double tmp = sqrt(B/tvl);
19         p[tot] = Point(near.x+tmp*tv.x,near.y+tmp*tv.y);
20         if (OnSeg(Line(a,b),p[tot]) == true) tot++;
21         p[tot] = Point(near.x-tmp*tv.x,near.y-tmp*tv.y);
22         if (OnSeg(Line(a,b),p[tot]) == true) tot++;
23     }
24     if (tot == 3) {
25         if (cmp(Point(p[0],p[1]).Length(),Point(p[0],p[2]).Length()) > 0)
26             swap(p[1],p[2]);
27     }
28     p[tot++] = b;
29     double res = 0.0,theta,a0,a1,sgn;
30     for (int i = 0; i < tot-1; i++) {
31         if (InCircle(p[i],r) == true && InCircle(p[i+1],r) == true) {
32             res += 0.5*xmult(p[i],p[i+1]);
33         } else {
34             a0 = atan2(p[i+1].y,p[i+1].x);
35             a1 = atan2(p[i].y,p[i].x);
36             if (a0 < a1) a0 += 2*pi;
37             theta = a0-a1;
38             if (cmp(theta,pi) >= 0) theta = 2*pi-theta;
39             sgn = xmult(p[i],p[i+1])/2.0;
40             if (cmp(sgn,0) < 0) theta = -theta;
41             res += 0.5*r*r*theta;
42         }
43     }
44     return res;
45 }
46 area2 = 0.0;
47 for (int i = 0; i < resn; i++) //counterclockwise
48     area2 += CalcArea(p[i],p[(i+1)%resn],r);

```

4 Graph

4.1 Sap

```

1  const int MAXEDGE=50000;
2  const int MAXN=3000;

```

```

3  const int inf=0x3fffffff;
4  struct edges {
5      int cap,to,next,flow;
6  } edge[MAXEDGE+100];
7  struct nodes {
8      int head,label,pre,cur;
9  } node[MAXN+100];
10 int L,N;
11 int gap[MAXN+100];
12 void init(int n) {
13     L=0;
14     N=n;
15     for (int i=0; i<N; i++)
16         node[i].head=-1;
17 }
18 void add_edge(int x,int y,int z,int w) {
19     edge[L].cap=z;
20     edge[L].flow=0;
21     edge[L].to=y;
22     edge[L].next=node[x].head;
23     node[x].head=L++;
24     edge[L].cap=w;
25     edge[L].flow=0;
26     edge[L].to=x;
27     edge[L].next=node[y].head;
28     node[y].head=L++;
29 }
30 int maxflow(int s,int t) {
31     memset(gap,0,sizeof(gap));
32     gap[0]=N;
33     int u,ans=0;
34     for (int i=0; i<N; i++) {
35         node[i].cur=node[i].head;
36         node[i].label=0;
37     }
38     u=s;
39     node[u].pre=-1;
40     while (node[s].label<N) {
41         if (u==t) {
42             int min=inf;
43             for (int i=node[u].pre; i!=-1; i=node[edge[i^1].to].pre)
44                 if (min>edge[i].cap-edge[i].flow)
45                     min=edge[i].cap-edge[i].flow;
46             for (int i=node[u].pre; i!=-1; i=node[edge[i^1].to].pre) {
47                 edge[i].flow+=min;
48                 edge[i^1].flow-=min;
49             }
50             u=s;
51             ans+=min;
52             continue;
53         }
54         bool flag=false;
55         int v;
56         for (int i=node[u].cur; i!=-1; i=edge[i].next) {

```

```

57             v=edge[i].to;
58             if (edge[i].cap-edge[i].flow &&
59                 node[v].label+1==node[u].label) {
60                 flag=true;
61                 node[u].cur=node[v].pre=i;
62                 break;
63             }
64         }
65         if (flag) {
66             u=v;
67             continue;
68         }
69         node[u].cur=node[u].head;
70         int min=N;
71         for (int i=node[u].head; i!=-1; i=edge[i].next)
72             if (edge[i].cap-edge[i].flow && node[edge[i].to].label<min)
73                 min=node[edge[i].to].label;
74         gap[node[u].label]--;
75         if (!gap[node[u].label]) return ans;
76         node[u].label=min+1;
77         gap[node[u].label]++;
78         if (u!=s) u=edge[node[u].pre^1].to;
79     }
80     return ans;
81 }

```

4.2 Minimal cost maximal flow

```

1  //Use stack instead of queue when get TLE
2  int L,N;
3  int K;
4  struct edges {
5      int to,next,cap,flow,cost;
6  } edge[MAXM];
7  struct nodes {
8      int dis,pre,head;
9      bool visit;
10 } node[MAXN];
11 void init(int n) {
12     N=n;
13     L=0;
14     for (int i=0; i<N; i++)
15         node[i].head=-1;
16 }
17 void add_edge(int x,int y,int cap,int cost) {
18     edge[L].to=y;
19     edge[L].cap=cap;
20     edge[L].cost=cost;
21     edge[L].flow=0;
22     edge[L].next=node[x].head;
23     node[x].head=L++;
24     edge[L].to=x;
25     edge[L].cap=0;
26     edge[L].cost=-cost;

```

```

27 edge[L].flow=0;
28 edge[L].next=node[y].head;
29 node[y].head=L++;
30 }
31 bool spfa(int s,int t) {
32     queue<int> q;
33     for (int i=0; i<N; i++) {
34         node[i].dis=0x3fffffff;
35         node[i].pre=-1;
36         node[i].visit=0;
37     }
38     node[s].dis=0;
39     node[s].visit=1;
40     q.push(s);
41     while (!q.empty()) {
42         int u=q.front();
43         node[u].visit=0;
44         for (int i=node[u].head; i!=-1; i=edge[i].next) {
45             int v=edge[i].to;
46             if (edge[i].cap>edge[i].flow &&
47                 node[v].dis>node[u].dis+edge[i].cost) {
48                 node[v].dis=node[u].dis+edge[i].cost;
49                 node[v].pre=i;
50                 if (!node[v].visit) {
51                     node[v].visit=1;
52                     q.push(v);
53                 }
54             }
55         }
56         q.pop();
57     }
58     if (node[t].pre!=-1)
59         return 0;
60     else
61         return 1;
62 }
63 int mcmf(int s,int t,int &cost) {
64     int flow=0;
65     while (spfa(s,t)) {
66         int max=inf;
67         for (int i=node[t].pre; i!=-1; i=node[edge[i^1].to].pre) {
68             if (max>edge[i].cap-edge[i].flow)
69                 max=edge[i].cap-edge[i].flow;
70         }
71         for (int i=node[t].pre; i!=-1; i=node[edge[i^1].to].pre) {
72             edge[i].flow+=max;
73             edge[i^1].flow-=max;
74             cost+=edge[i].cost*max;
75         }
76         flow+=max;
77     }
78     return flow;
79 }

```

4.3 Bi-connect

```

1 struct edges {
2     int to,next;
3     bool cut,visit;
4 } edge[MAXM<<1];
5 int head[MAXN],low[MAXN],dpt[MAXN],L;
6 bool visit[MAXN],cut[MAXN];
7 void init(int n) {
8     L=0;
9     memset(head,-1,4*n);
10    memset(visit,0,n);
11 }
12 void add_edge(int u,int v) {
13     edge[L].cut=edge[L].visit=0;
14     edge[L].to=v;
15     edge[L].next=head[u];
16     head[u]=L++;
17 }
18 int idx;
19 stack<int> st;
20 int bcc[MAXM];
21 void dfs(int u,int fu,int deg) {
22     cut[u]=0;
23     visit[u]=1;
24     low[u]=dpt[u]=deg;
25     int tot=0;
26     for (int i=head[u]; i!=-1; i=edge[i].next) {
27         int v=edge[i].to;
28         if (edge[i].visit)
29             continue;
30         st.push(i/2);
31         edge[i].visit=edge[i^1].visit=1;
32         if (visit[v]) {
33             low[u]=dpt[v]>low[u]?low[u]:dpt[v];
34             continue;
35         }
36         dfs(v,u,deg+1);
37         edge[i].cut=edge[i^1].cut=(low[v]>dpt[u] || edge[i].cut);
38         if (u!=fu) cut[u]=low[v]>=dpt[u]?1:cut[u];
39         if (low[v]>=dpt[u] || u==fu) {
40             while (st.top()!=i/2) {
41                 int x=st.top()*2,y=st.top()*2+1;
42                 bcc[st.top()]=idx;
43                 st.pop();
44             }
45             bcc[i/2]=idx++;
46             st.pop();
47         }
48         low[u]=low[v]>low[u]?low[u]:low[v];
49         tot++;
50     }
51     if (u==fu && tot>1) cut[u]=1;
52 }

```

```

53 int main() {
54     int n,m;
55     while (scanf("%d%d",&n,&m)!=EOF) {
56         init(n);
57         for (int i=0; i<m; i++) {
58             int u,v;
59             scanf("%d%d",&u,&v);
60             add_edge(u,v);
61             add_edge(v,u);
62         }
63         idx=0;
64         for (int i=0; i<n; i++)
65             if (!visit[i])
66                 dfs(i,i,0);
67     }
68     return 0;
69 }

```

4.4 Cut and bridge

```

1 struct edges {
2     int to,next;
3     bool cut,visit;
4     int from;
5 } edge[MAXN-1<<1];
6 int head[MAXN],low[MAXN],dfn[MAXN],L;
7 bool visit[MAXN],cut[MAXN];
8 void init(int n) {
9     L=0;
10    memset(head,-1,4*n);
11    memset(cut,0,4*n);
12    memset(visit,0,4*n);
13 }
14 void add_edge(int u,int v) {
15     edge[L].from=u;
16     edge[L].cut=edge[L].visit=0;
17     edge[L].to=v;
18     edge[L].next=head[u];
19     head[u]=L++;
20 }
21 int idx;
22 void dfs(int u,int fu) {
23     visit[u]=1;
24     low[u]=dfn[u]=idx++;
25     int tot=0;
26     for (int i=head[u]; i!=-1; i=edge[i].next) {
27         int v=edge[i].to;
28         if (edge[i].visit)
29             continue;
30         edge[i].visit=edge[i^1].visit=1;
31         if (visit[v]) {
32             low[u]=dfn[v]>low[u]?low[u]:dfn[v];
33             continue;
34         }

```

```

35         dfs(v,u);
36         edge[i].cut=edge[i^1].cut=low[v]>dfn[u] || edge[i].cut;
37         if (u!=fu) cut[u]=low[v]>=dfn[u]?1:cut[u];
38         low[u]=low[v]>low[u]?low[u]:low[v];
39         tot++;
40     }
41     if (u==fu && tot>1) cut[u]=1;
42 }
43 int main() {
44     int t;
45     scanf("%d",&t);
46     while (t--) {
47         int n,m;
48         scanf("%d%d",&n,&m);
49         init(n);
50         for (int i=0; i<m; i++) {
51             int u,v;
52             scanf("%d%d",&u,&v);
53             add_edge(u,v);
54             add_edge(v,u);
55         }
56         for (int i=0; i<n; i++)
57             if (!visit[i]) {
58                 idx=0;
59                 dfs(i,i);
60             }
61     }
62     return 0;
63 }

```

4.5 Global cut

```

1 int map[maxn][maxn];
2 int n;
3 void contract(int x,int y) {
4     int i,j;
5     for (i=0; i<n; i++)
6         if (i!=x) map[x][i]+=map[y][i],map[i][x]+=map[i][y];
7     for (i=y+1; i<n; i++) for (j=0; j<n; j++) {
8         map[i-1][j]=map[i][j];
9         map[j][i-1]=map[j][i];
10    }
11    n--;
12 }
13 int w[maxn],c[maxn];
14 int sx,tx;
15 int mincut() {
16     int i,j,k,t;
17     memset(c,0,sizeof(c));
18     c[0]=1;
19     for (i=0; i<n; i++) w[i]=map[0][i];
20     for (i=1; i+1<n; i++) {
21         t=k=-1;
22         for (j=0; j<n; j++) if (c[j]==0&&w[j]>k)

```

```

23     k=w[t=j];
24     c[sx=t]=1;
25     for (j=0; j<n; j++) w[j]+=map[t][j];
26 }
27 for (i=0; i<n; i++) if (c[i]==0) return w[tx=i];
28 }
29 int main() {
30     int i,j,k,m;
31     while (scanf("%d%d",&n,&m)!=EOF) {
32         memset(map,0,sizeof(map));
33         while (m--) {
34             scanf("%d%d%d",&i,&j,&k);
35             map[i][j]+=k;
36             map[j][i]+=k;
37         }
38         int mint=999999999;
39         while (n>1) {
40             k=mincut();
41             if (k<mint) mint=k;
42             contract(sx,tx);
43         }
44         printf("%d\n",mint);
45     }
46     return 0;
47 }

```

4.6 Euler path

```

1 //Directed graph
2 void solve(int x) {
3     int i;
4     if (!match[x]) {
5         path[++l]=x;
6         return ;
7     }
8     for (i=1; i<=n; i++)
9         if (b[x][i]) {
10             b[x][i]--;
11             match[x]--;
12             solve(i);
13         }
14     path[++l]=x;
15 }
16 //Undirected graph
17 void solve(int x) {
18     int i;
19     if (!match[x]) {
20         path[++l]=x;
21         return ;
22     }
23     for (i=1; i<=n; i++)
24         if (b[x][i]) {
25             b[x][i]--;
26             b[i][x]--;

```

```

27         match[x]--;
28         match[i]--;
29         solve(i);
30     }
31     path[++l]=x;
32 }

```

4.7 Strongly connected component

```

1 int dfsnum[2000];
2 int low[2000];
3 int stack[2000];
4 int top;
5 int ans;
6 int an;
7 int be[2000];
8 int flag[2000];
9 void dfs(int x) {
10     dfsnum[x] = low[x] = ans++;
11     stack[++top] = x;
12     flag[x] = 1;
13     for (int i = head[x]; i != -1; i = edge[i].next) {
14         int y = edge[i].to;
15         if (dfsnum[y] == -1) {
16             dfs(y);
17             low[x] = min(low[x], low[y]);
18         } else if (flag[y] == 1)
19             low[x] = min(low[x], dfsnum[y]);
20     }
21     if (dfsnum[x] == low[x]) {
22         while (stack[top] != x) {
23             flag[stack[top]] = 0;
24             be[stack[top]] = an;
25             top--;
26         }
27         flag[x] = 0;
28         be[x] = an++;
29         top--;
30     }
31 }
32 void SC() {
33     memset(dfsnum,-1,sizeof(dfsnum));
34     memset(flag,0,sizeof(flag));
35     top = 0;
36     an = 0;
37     ans = 0;
38     for (int i = 0; i < n; i++)
39         if (dfsnum[i] == -1)
40             dfs(i);
41 }

```

4.8 Match

4.8.1 Bipartite graph

```

1 bool check(int u) {
2     for (int i=head[u]; i!=-1; i=edge[i].next) {
3         int v=edge[i].to;
4         if (!use[v]) {
5             use[v]=1;
6             if (pre[v]==-1 || check(pre[v])) {
7                 pre[v]=u;
8                 return 1;
9             }
10        }
11    }
12    return 0;
13 }
14 int match() {
15     int ret=0;
16     memset(pre,-1,sizeof(pre));
17     for (int u=1; u<=N; u++) {
18         memset(use,0,sizeof(use));
19         if (check(u))
20             ret++;
21     }
22     return ret;
23 }

```

4.8.2 Edmonds

```

1 int N;
2 bool Graph[MaxN+1][MaxN+1];
3 int Match[MaxN+1];
4 bool InQueue[MaxN+1], InPath[MaxN+1], InBlossom[MaxN+1];
5 int Head, Tail;
6 int Queue[MaxN+1];
7 int Start, Finish;
8 int NewBase;
9 int Father[MaxN+1], Base[MaxN+1];
10 int Count;
11 void CreateGraph() {}
12 void Push(int u) {
13     Queue[Tail] = u;
14     Tail++;
15     InQueue[u] = true;
16 }
17 int Pop() {
18     int res = Queue[Head];
19     Head++;
20     return res;
21 }
22 int FindCommonAncestor(int u, int v) {
23     memset(InPath, false, sizeof(InPath));
24     while (true) {
25         u = Base[u];
26         InPath[u] = true;
27         if (u == Start) break;
28         u = Father[Match[u]];
29     }

```

```

30     while (true) {
31         v = Base[v];
32         if (InPath[v]) break;
33         v = Father[Match[v]];
34     }
35     return v;
36 }
37 void ResetTrace(int u) {
38     int v;
39     while (Base[u] != NewBase) {
40         v = Match[u];
41         InBlossom[Base[u]] = InBlossom[Base[v]] = true;
42         u = Father[v];
43         if (Base[u] != NewBase) Father[u] = v;
44     }
45 }
46 void BlossomContract(int u, int v) {
47     NewBase = FindCommonAncestor(u, v);
48     memset(InBlossom, false, sizeof(InBlossom));
49     ResetTrace(u);
50     ResetTrace(v);
51     if (Base[u] != NewBase) Father[u] = v;
52     if (Base[v] != NewBase) Father[v] = u;
53     for (int tu = 1; tu <= N; tu++)
54         if (InBlossom[Base[tu]]) {
55             Base[tu] = NewBase;
56             if (!InQueue[tu]) Push(tu);
57         }
58 }
59 void FindAugmentingPath() {
60     memset(InQueue, false, sizeof(InQueue));
61     memset(Father, 0, sizeof(Father));
62     for (int i = 1; i <= N; i++)
63         Base[i] = i;
64     Head = Tail = 1;
65     Push(Start);
66     Finish = 0;
67     while (Head < Tail) {
68         int u = Pop();
69         for (int v = 1; v <= N; v++)
70             if (Graph[u][v] && (Base[u] != Base[v]) && (Match[u] != v)) {
71                 if ((v == Start) ||
72                     ((Match[v] > 0) && (Father[Match[v]] > 0)))
73                     BlossomContract(u, v);
74                 else if (Father[v] == 0) {
75                     Father[v] = u;
76                     if (Match[v] > 0)
77                         Push(Match[v]);
78                 } else {
79                     Finish = v;
80                     return;
81                 }
82             }
83     }

```



```

84 }
85 }
86 void AugmentPath() {
87     int u,v,w;
88     u = Finish;
89     while (u > 0) {
90         v = Father[u];
91         w = Match[v];
92         Match[v] = u;
93         Match[u] = v;
94         u = w;
95     }
96 }
97 void Edmonds() {
98     memset(Match,0,sizeof(Match));
99     for (int u = 1; u <= N; u++)
100         if (Match[u] == 0) {
101             Start = u;
102             FindAugmentingPath();
103             if (Finish > 0) AugmentPath();
104         }
105 }
106 void PrintMatch() {}
107 int main() {
108     CreateGraph();
109     Edmonds();
110     PrintMatch();
111 }

```

4.8.3 KM

```

1 bool visx[N],visy[N];
2 int lx[N],ly[N];
3 int matchy[N];
4 int map[N][N];
5 bool find(int x) {
6     visx[x]=true;
7     int t;
8     for (int y=0; y<ycnt; y++) {
9         if (!visy[y]) {
10             t=lx[x]+ly[y]-map[x][y];
11             if (t==0) {
12                 visy[y]=true;
13                 if (matchy[y]==-1 || find(matchy[y])) {
14                     matchy[y]=x;
15                     return true;
16                 }
17             } else if (lack>t) lack=t;
18         }
19     }
20     return false;
21 }
22 void KM() {
23     memset(lx,0,sizeof(lx));
24     memset(ly,0,sizeof(ly));

```

```

25     memset(matchy,-1,sizeof(matchy));
26     for (int i=0; i<xcnt; i++)
27         for (int j=0; j<ycnt; j++)
28             if (map[i][j]>lx[i])
29                 lx[i]=map[i][j];
30     for (int x=0; x<xcnt; x++) {
31         while (true) {
32             memset(visx,false,sizeof(visx));
33             memset(visy,false,sizeof(visy));
34             lack=INFI;
35             if (find(x)) break;
36             for (int i=0; i<xcnt; i++) {
37                 if (visx[i]) lx[i]-=lack;
38                 if (visy[i]) ly[i]+=lack;
39             }
40         }
41     }
42     int cost=0;
43     for (int i=0; i<ycnt; i++)
44         cost+=map[matchy[i]][i];
45 }

```

4.9 Clique

```

1 bool am[100][100];
2 int ans;
3 int c[100];
4 int U[100][100];
5 int n;
6 bool dfs(int rest,int num) {
7     if (!rest) {
8         if (num>=ans)
9             return 1;
10        else
11            return 0;
12    }
13    int pre=-1;
14    for (int i=0; i<rest && rest-i+num>=ans; i++) {
15        int idx=U[num][i];
16        if (num+c[idx]<ans)
17            return 0;
18        int nrest=0;
19        for (int j=i+1; j<rest; j++)
20            if (am[idx][U[num][j]])
21                U[num+1][nrest++]=U[num][j];
22        if (dfs(nrest,num+1))
23            return 1;
24    }
25    return 0;
26 }
27 int main() {
28     while (scanf("%d",&n),n) {
29         for (int i=0; i<n; i++)
30             for (int j=0; j<n; j++)

```

```

31     scanf("%d",&am[i][j]);
32     ans=0;
33     for (int i=n-1; i>=0; i--) {
34         int rest=0;
35         for (int j=i+1; j<n; j++)
36             if (am[i][j])
37                 U[0][rest++]=j;
38         ans+=dfs(rest,0);
39         c[i]=ans;
40     }
41     printf("%d\n",ans);
42 }
43 return 0;
44 }

```

4.10 Spanning tree

4.10.1 Count the number of spanning tree

```

1 Matrix laplacian;
2 laplacian.clear();
3 for (int i = 0; i < n; i++)
4     for (int j = 0; j < n; j++)
5         if (i != j && G[i][j]) {
6             laplacian.a[i][j] = -1;
7             laplacian.a[i][i]++;
8         }
9 printf("%d\n",laplacian.det(n-1));

```

4.10.2 Spanning tree on directed graph

```

1 struct Edge {
2     int u,v,cost;
3 };
4 Edge e[1001*1001];
5 int pre[1001],id[1001],visit[1001],in[1001];
6 int zhuliu(int root,int n,int m,Edge e[]) {
7     int res = 0,u,v;
8     while (true) {
9         for (int i = 0; i < n; i++)
10             in[i] = inf;
11         for (int i = 0; i < m; i++)
12             if (e[i].u != e[i].v && e[i].cost < in[e[i].v]) {
13                 pre[e[i].v] = e[i].u;
14                 in[e[i].v] = e[i].cost;
15             }
16         for (int i = 0; i < n; i++)
17             if (i != root)
18                 if (in[i] == inf) return -1;
19         int tn = 0;
20         memset(id,-1,sizeof(id));
21         memset(visit,-1,sizeof(visit));
22         in[root] = 0;
23         for (int i = 0; i < n; i++) {

```

```

24             res += in[i];
25             v = i;
26             while (visit[v] != i && id[v] == -1 && v != root) {
27                 visit[v] = i;
28                 v = pre[v];
29             }
30             if (v != root && id[v] == -1) {
31                 for (int u = pre[v] ; u != v ; u = pre[u])
32                     id[u] = tn;
33                 id[v] = tn++;
34             }
35         }
36         if (tn == 0) break;
37         for (int i = 0; i < n; i++)
38             if (id[i] == -1)
39                 id[i] = tn++;
40         for (int i = 0; i < m; i++) {
41             int v = e[i].v;
42             e[i].u = id[e[i].u];
43             e[i].v = id[e[i].v];
44             if (e[i].u != e[i].v)
45                 e[i].cost -= in[v];
46             else
47                 swap(e[i],e[---m]);
48         }
49         n = tn;
50         root = id[root];
51     }
52     return res;
53 }

```

5 Math

5.1 FFT

```

1 struct vir {
2     long double re, im;
3     vir(long double a = 0, long double b = 0) {
4         re = a;
5         im = b;
6     }
7     vir operator +(const vir& b) const {
8         return vir(re + b.re, im + b.im);
9     }
10    vir operator -(const vir& b) const {
11        return vir(re - b.re, im - b.im);
12    }
13    vir operator *(const vir& b) const {
14        return vir(re * b.re - im * b.im, re * b.im + im * b.re);
15    };
16 };
17 void change(vir *x, int len, int loglen) {
18     int i, j, k, t;
19     for (i = 0; i < len; i++) {

```

```

20     t = i;
21     for (j = k = 0; j < loglen; j++, t >= 1)
22         k = (k << 1) | (t & 1);
23     if (k < i) {
24         vir wt = x[k];
25         x[k] = x[i];
26         x[i] = wt;
27     }
28 }
29 }
30 void fft(vir *x, int len, int loglen) {
31     int i, j, t, s, e;
32     change(x, len, loglen);
33     t = 1;
34     for (i = 0; i < loglen; i++, t <= 1) {
35         s = 0;
36         e = s + t;
37         while (s < len) {
38             vir a, b, wo(cos(PI / t), sin(PI / t)), wn(1, 0);
39             for (j = s; j < s + t; j++) {
40                 a = x[j];
41                 b = x[j + t] * wn;
42                 x[j] = a + b;
43                 x[j + t] = a - b;
44                 wn = wn * wo;
45             }
46             s = e + t;
47             e = s + t;
48         }
49     }
50 }
51 void dit_fft(vir *x, int len, int loglen) {
52     int i, j, s, e, t = 1 << loglen;
53     for (i = 0; i < loglen; i++) {
54         t >= 1;
55         s = 0;
56         e = s + t;
57         while (s < len) {
58             vir a, b, wn(1, 0), wo(cos(PI / t), -sin(PI / t));
59             for (j = s; j < s + t; j++) {
60                 a = x[j] + x[j + t];
61                 b = (x[j] - x[j + t]) * wn;
62                 x[j] = a;
63                 x[j + t] = b;
64                 wn = wn * wo;
65             }
66             s = e + t;
67             e = s + t;
68         }
69     }
70     change(x, len, loglen);
71     for (i = 0; i < len; i++)
72         x[i].re /= len;
73 }

```

5.1.1 Usage

```

1 vir x1[MAXN], x2[MAXN];
2 void solve(long long *a, int lena, long long *b, int lenb, long long *ret,
3     int& len) {
4     int len1 = lena << 1;
5     int len2 = lenb << 1;
6     len = 1;
7     int loglen = 0;
8     while (len < len1 || len < len2) {
9         len <= 1;
10        loglen++;
11    }
12    for (int i = 0; i < lena; i++)
13        x1[i] = vir(a[i], 0);
14    for (int i = lena; i < len; i++)
15        x1[i] = vir(0, 0);
16    for (int i = 0; i < lenb; i++)
17        x2[i] = vir(b[i], 0);
18    for (int i = lenb; i < len; i++)
19        x2[i] = vir(0, 0);
20    fft(x1, len, loglen);
21    fft(x2, len, loglen);
22    for (int i = 0; i < len; i++)
23        x1[i] = x1[i] * x2[i];
24    dit_fft(x1, len, loglen);
25    for (int i = 0; i < len; i++)
26        ret[i] = (long long)(x1[i].re + 0.5);

```

5.2 Euler function

```

1 int getEuler(int x) {
2     getFactor(x);
3     int ret=x;
4     for (int i=0; i<N; i++)
5         ret = ret/fac[i]*(fac[i]-1);
6     return ret;
7 }
8 void getEuler2() {
9     memset(euler,0,sizeof(euler));
10    euler[1] = 1;
11    for (int i = 2; i <= 3000000; i++) {
12        if (!euler[i]) {
13            for (int j = i; j <= 3000000; j += i) {
14                if (!euler[j])
15                    euler[j] = j;
16                euler[j] = euler[j]/i*(i-1);
17            }
18        }
19    }
20 }

```

5.3 Ex-GCD

```

1 //Find one solution (x,y) of  $ax + by = gcd(a,b)$ 
2 long long ex_gcd(long long a,long long b,long long &x,long long &y) {
3     if (b) {
4         long long ret = ex_gcd(b,a%b,x,y),tmp = x;
5         x = y;
6         y = tmp-(a/b)*y;
7         return ret;
8     } else {
9         x = 1;
10        y = 0;
11        return a;
12    }
13 }

```

5.4 Prime

5.4.1 Get primes

```

1 int N;
2 bool isPrime[10001];
3 int prime[10000];
4 void getPrime(int n) {
5     memset(isPrime,1,++n);
6     N=0;
7     isPrime[0]=isPrime[1]=0;
8     for (int i=2; i<n; i++) {
9         if (isPrime[i])
10            prime[N++]=i;
11        for (int j=0; j<N && prime[j]*i<n; j++) {
12            isPrime[i*prime[j]]=0;
13            if (i%prime[j]==0)
14                break;
15        }
16    }
17 }

```

5.4.2 Get factors

```

1 const int TIME = 8;
2 int factor[100],fac_top = -1;
3 //GCD of bint
4 bint gcd(bint small,bint big) {
5     while(small) {
6         swap(small,big);
7         small%=big;
8     }
9     return abs(big);
10 }
11 //ret = (a*b)%n ( $n < 2^{62}$ )
12 bint muti_mod(bint a,bint b,bint n) {
13     bint exp = a%n, res = 0;
14     while(b) {
15         if(b&1) {
16             res += exp;

```

```

17         if(res>n) res -= n;
18     }
19     exp <<= 1;
20     if (exp>n) exp -= n;
21     b>>=1;
22 }
23 return res;
24 }
25 // ret = (a^b)%n
26 bint mod_exp(bint a,bint p,bint m) {
27     bint exp=a%m, res=1;
28     while(p>1) {
29         if(p&1)
30             res=muti_mod(res,exp,m);
31         exp = muti_mod(exp,exp,m);
32         p>>=1;
33     }
34     return muti_mod(res,exp,m);
35 }
36 //miller-rabin
37 bool miller_rabin(bint n, int times) {
38     if(n==2)return 1;
39     if(n<2||!(n&1))return 0;
40     bint a, u=n-1, x, y;
41     int t=0;
42     while(u%2==0) {
43         t++;
44         u/=2;
45     }
46     srand(time(0));
47     for(int i=0; i<times; i++) {
48         a = rand() % (n-1) + 1;
49         x = mod_exp(a, u, n);
50         for(int j=0; j<t; j++) {
51             y = muti_mod(x, x, n);
52             if (y == 1 && x != 1 && x != n-1 )
53                 return false; //must not
54             x = y;
55         }
56         if( y!=1) return false;
57     }
58     return true;
59 }
60 bint pollard_rho(bint n,int c) {
61     bint x,y,d,i = 1,k = 2;
62     srand(time(0));
63     x = rand()%(n-1)+1;
64     y = x;
65     while(true) {
66         i++;
67         x = (muti_mod(x,x,n) + c) % n;
68         d = gcd(y-x, n);
69         if (1 < d && d < n) return d;
70         if( y == x) return n;

```

```

71     if(i == k) {
72         y = x;
73         k <<= 1;
74     }
75 }
76 }
77 void findFactor(bint n,int k) {
78     if(n==1)return;
79     if(miller_rabin(n, TIME)) {
80         factor[++fac_top] = n;
81         return;
82     }
83     bint p = n;
84     while(p >= n)
85         p = pollard_rho(p,k--);
86     findFactor(p,k);
87     findFactor(n/p,k);
88 }

```

5.5 Simpson

```

1 double Simp(double l,double r) {
2     double h = (r-l)/2.0;
3     return h*(calc(l)+4*calc((l+r)/2.0)+calc(r))/3.0;
4 }
5 double rSimp(double l,double r) {
6     double mid = (l+r)/2.0;
7     if (abs((Simp(l,r)-Simp(l,mid)-Simp(mid,r)))/15 < eps)
8         return Simp(l,r);
9     else
10        return rSimp(l,mid)+rSimp(mid,r);
11 }

```

5.6 Chinese remainder theorem

```

1 int m[10],a[10]; //x mod m_i = a_i
2 bool solve(int &m0,int &a0,int m,int a) {
3     int y,x;
4     int g=ex_gcd(m0,m,x,y);
5     if (abs(a-a0)%g) return 0;
6     x*=(a-a0)/g;
7     x%=m/g;
8     a0=(x*m0+a0);
9     m0*=m/g;
10    a0%=m0;
11    if (a0<0) a0+=m0;
12    return 1;
13 }
14 int MLES() {
15     bool flag=1;
16     int m0=1,a0=0;
17     for (int i=0; i<n; i++)
18         if (!solve(m0,a0,m[i],a[i])) {
19             flag=0;

```

```

20         break;
21     }
22     if (flag)
23         return a0;
24     else
25         return -1;
26 }

```

5.7 Lucus

```

1 //num[i] = i!
2 int comLucus(int n,int m,int p) {
3     int ans=1;
4     for (; n && m && ans; n/=p,m/=p) {
5         if (n%p>=m%p)
6             ans = ans*num[n%p]%p*getInv(num[m%p]%p)%p
7                 *getInv(num[n%p-m%p])%p;
8         else
9             ans=0;
10    }
11    return ans;
12 }

```

5.8 Inverse element

```

1 void getInv2(int x) {
2     inv[1]=1;
3     for (int i=2; i<=x; i++)
4         inv[i]=(mod-(mod/i)*inv[mod%i]%mod)%mod;
5 }

```

6 Search

6.1 Dancing links

```

1 struct DLX {
2     int h,n,m,tot;
3     int U[MaxN*MaxM],D[MaxN*MaxM],L[MaxN*MaxM],R[MaxN*MaxM],Row[MaxN*MaxM],Col
4         [MaxN*MaxM];
5     int S[MaxM],O[MaxN];
6     bool hasans;
7     void init() {
8         h = 0;
9         hasans = false;
10        tot = m+n;
11        for (int i = 0; i <= m; i++) {
12            D[i] = U[i] = Col[i] = i;
13            Row[i] = S[i] = 0;
14            L[i] = (i+m)%(m+1);
15            R[i] = (i+1)%(m+1);
16        }
17        for (int i = 1; i <= n; i++) {
18            R[i+m] = L[i+m] = i+m;
19            Row[i+m] = i;

```

```

19     Col[i+m] = 0;
20 }
21 }
22 void insert(int x,int y) {
23     tot++;
24     Row[tot] = x;
25     Col[tot] = y;
26     S[y]++;
27     int colPos,rowPos;
28     colPos = y;
29     while (true) {
30         colPos = D[colPos];
31         if (colPos == y || Row[colPos] > x)    break;
32     }
33     colPos = U[colPos];
34     if (Row[colPos] == x)    return;
35     U[tot] = colPos;
36     D[tot] = D[colPos];
37     U[D[tot]] = D[U[tot]] = tot;
38     rowPos = x+m;
39     while (true) {
40         rowPos = R[rowPos];
41         if (rowPos == x+m || Col[rowPos] > y)    break;
42     }
43     rowPos = L[rowPos];
44     if (Col[rowPos] == y)    return;
45     L[tot] = rowPos;
46     R[tot] = R[rowPos];
47     L[R[tot]] = R[L[tot]] = tot;
48 }
49 void print(int deep) {
50     for (int i = 0; i < deep; i++)
51         printf("%d_", O[i]);
52     printf("\n");
53 }
54 void cover(int col) {
55     L[R[col]] = L[col];
56     R[L[col]] = R[col];
57     for (int i = D[col]; i != col; i = D[i])
58         for (int j = R[i]; j != i; j = R[j])
59             if (Col[j] != col) {
60                 U[D[j]] = U[j];
61                 D[U[j]] = D[j];
62                 S[Col[j]]--;
63             }
64 }
65 void resume(int col) {
66     for (int i = U[col]; i != col; i = U[i])
67         for (int j = L[i]; j != i; j = L[j])
68             if (Col[j] != col) {
69                 S[Col[j]]++;
70                 U[D[j]] = j;
71                 D[U[j]] = j;
72             }

```

```

73     L[R[col]] = col;
74     R[L[col]] = col;
75 }
76 void initDFS() {
77     for (int i = 1; i <= n; i++) {
78         L[R[i+m]] = L[i+m];
79         R[L[i+m]] = R[i+m];
80     }
81 }
82 void DFS(int deep) {
83     if (hasans == true) return;
84     if (R[0] == 0) {
85         hasans = true;
86         print(deep);
87         return;
88     };
89     int tc = R[0];
90     for (int i = R[0]; i != 0; i = R[i])
91         if (S[i] < S[tc])    tc = i;
92     cover(tc);
93     for (int i = D[tc]; i != tc; i = D[i]) {
94         int temp = O[deep];
95         O[deep] = Row[i];
96         for (int j = R[i]; j != i; j = R[j])
97             cover(Col[j]);
98         DFS(deep+1);
99         for (int j = L[i]; j != i; j = L[j])
100             resume(Col[j]);
101         O[deep] = temp;
102     }
103     resume(tc);
104 }
105 }

```

6.1.1 Usage

```

1 DLX g;
2 g.n = ROW_SIZE;
3 g.m = COL_SIZE;
4 g.init();
5 g.insert(ROW, COL);
6 g.initDFS();
7 g.DFS(0);

```

6.2 Dancing links (A-star)

```

1 namespace DLX {
2     const int MAXN = 1000;
3     const int MAXM = 400;
4     const int INF = 0x3f3f3f3f;
5     int D[MAXN * MAXM], U[MAXN * MAXM], L[MAXN * MAXM], R[MAXN * MAXM], Col[MAXN
        * MAXM], Row[MAXN * MAXM];
6     int CNT, BEG[MAXN * MAXM], END[MAXN * MAXM], ANS, USE[MAXM], _USE[MAXM];
7     int SUM[MAXM];

```

```

8  bool vis[MAXM];
9  void init(int n) {
10     memset(BEG, 0xff, sizeof(BEG));
11     for(int i = 1; i <= n; i++)
12         SUM[L[i] + 1] = R[i - 1] = D[i] = U[i] = i = 0;
13     L[L[1] = R[n] = 0] = n, CNT = n + 1;
14     ANS = n + 1;
15 }
16 void link(int r, int c) {
17     D[CNT] = D[c], U[CNT] = c, U[D[c]] = CNT, D[c] = CNT, COL[CNT] = c, ROW[
18         CNT] = r, SUM[c]++;
19     if (BEG[r] == -1) BEG[r] = END[r] = CNT;
20     R[END[r]] = CNT, L[CNT] = END[r], R[CNT] = BEG[r], L[BEG[r]] = CNT, END[r]
21         = CNT++;
22 }
23 void DLX_Remove_Repeat(int c) {
24     for (int i = D[c]; i != c; i = D[i])
25         L[R[i]] = L[i], R[L[i]] = R[i], SUM[COL[i]]--;
26 }
27 void DLX_Resume_Repeat(int c) {
28     for (int i = U[c]; i != c; i = U[i])
29         L[R[i]] = i, R[L[i]] = i, SUM[COL[i]]++;
30 }
31 int Heuristics() {
32     memset(vis, true, sizeof(vis));
33     int c, i, j, cnt=0;
34     for(c=R[0]; c; c=R[c])
35         if(vis[c])
36             for(cnt++, vis[c] = false, i = D[c]; i != c; i = D[i])
37                 for(j = R[i]; j != i; j = R[j])
38                     vis[COL[j]] = false;
39     return cnt;
40 }
41 void DLX_Dfs(int n) {
42     if (Heuristics() + n >= ANS) return;
43     if (R[0] == 0) {
44         ANS = n;
45         for (int i = 0; i < n; i++)
46             USE[i] = _USE[i];
47         return ;
48     }
49     int i,now = INF,c;
50     for (i = R[0]; i; i = R[i])
51         if (now > SUM[i])
52             now = SUM[c = i];
53     for(i = D[c]; i != c; i = D[i]) {
54         DLX_Remove_Repeat(i);
55         for(int j = R[i]; j != i; j = R[j])
56             DLX_Remove_Repeat(j);
57         _USE[n] = ROW[i];
58         DLX_Dfs(n + 1);
59         for(int j = L[i]; j != i; j = L[j])
60             DLX_Resume_Repeat(j);
61         DLX_Resume_Repeat(i);

```

```

60     }
61 }
62 void solve() {
63     //ANS = m
64     DLX_Dfs(0);
65 }
66 };

```

7 String

7.1 Aho-Corasick automation

```

1  struct Trie {
2      int next[50][10],fail[50];
3      bool end[50];
4      int L,root;
5      int newNode() {
6          for (int i = 0; i < 10; i++)
7              next[L][i] = -1;
8          end[L] = false;
9          return L++;
10     }
11     void Init() {
12         L = 0;
13         root = newNode();
14     }
15     void Insert(char s[]) {
16         int now = root;
17         for (int i = 0; s[i] != 0; i++) {
18             if (next[now][s[i]-'0'] == -1)
19                 next[now][s[i]-'0'] = newNode();
20             now = next[now][s[i]-'0'];
21         }
22         end[now] = true;
23     }
24     void Build() {
25         queue<int> Q;
26         for (int i = 0; i < 10; i++)
27             if (next[root][i] == -1)
28                 next[root][i] = root;
29             else {
30                 fail[next[root][i]] = root;
31                 Q.push(next[root][i]);
32             }
33         while (!Q.empty()) {
34             int now = Q.front();
35             Q.pop();
36             end[now] |= end[fail[now]];
37             for (int i = 0; i < 10; i++)
38                 if (next[now][i] == -1)
39                     next[now][i] = next[fail[now]][i];
40                 else {
41                     fail[next[now][i]] = next[fail[now]][i];
42                     Q.push(next[now][i]);

```

```

43     }
44 }
45 }
46 };

```

7.2 KMP

Match the suffix of $A[\dots i]$ and the prefix of B

```

1 //Self match
2 int j;
3 p[0] = j = -1;
4 for (int i = 1; i < lb; i++) {
5     while (j >= 0 && b[j + 1] != b[i]) j = p[j];
6     if (b[j + 1] == b[i]) j++;
7     p[i] = j;
8 }
9 //Match
10 j = -1;
11 for (int i = 0; i < la; i++) {
12     while (j >= 0 && b[j + 1] != a[i]) j = p[j];
13     if (b[j + 1] == a[i]) j++;
14     KMP[i] = j + 1;
15 }

```

7.3 E-KMP

Common prefix of $A[i\dots]$ and B

```

1 //Self match
2 int j = 0;
3 while (j < lb && b[j] == b[j + 1])
4     j++;
5 p[0] = lb, p[1] = j;
6 int k = 1;
7 for (int i = 2; i < lb; i++) {
8     int Len = k + p[k] - 1, L = p[i - k];
9     if (L < Len - i + 1)
10        p[i] = L;
11     else {
12         j = max(0, Len - i + 1);
13         while (i + j < lb && b[i + j] == b[j])
14             j++;
15         p[i] = j, k = i;
16     }
17 }
18 //Match
19 j = 0;
20 while (j < la && j < lb && a[j] == b[j])
21     j++;
22 eKMP[0] = j;
23 k = 0;
24 for (int i = 1; i < la; i++) {
25     int Len = k + eKMP[k] - 1, L = p[i - k];

```

```

26     if (L < Len - i + 1)
27         eKMP[i] = L;
28     else {
29         j = max(0, Len - i + 1);
30         while (i + j < la && j < lb && a[i + j] == b[j])
31             j++;
32         eKMP[i] = j, k = i;
33     }
34 }

```

7.4 Manacher

```

1 const int maxn = 110000;
2
3 char Ma[maxn*2];
4 int Mp[maxn*2];
5 void Manacher(char s[],int len) {
6     int l = 0;
7     Ma[l++] = '.';
8     Ma[l++] = ',';
9     for (int i = 0; i < len; i++) {
10        Ma[l++] = s[i];
11        Ma[l++] = ',';
12    }
13    Ma[l] = 0;
14    int pnow = 0, pid = 0;
15    for (int i = 1; i < l; i++) {
16        if (pnow > i)
17            Mp[i] = min(Mp[2*pid-i], pnow-i);
18        else
19            Mp[i] = 1;
20        for (; Ma[i-Mp[i]] == Ma[i+Mp[i]]; Mp[i]++);
21        if (i+Mp[i] > pnow) {
22            pnow = i+Mp[i];
23            pid = i;
24        }
25    }
26 }
27 /*
28 abaaba
29 . , a , b , a , a , b , a ,
30 0 1 2 1 4 1 2 7 2 1 4 1 2 1
31 */

```

7.5 Suffix array

```

1 const int maxn = 200010;
2 int wx[maxn], wy[maxn], *x, *y, wss[maxn], wv[maxn];
3
4 bool cmp(int *r, int n, int a, int b, int l) {
5     return a+l < n && b+l < n && r[a] == r[b] && r[a+l] == r[b+l];
6 }
7 void da(int str[], int sa[], int rank[], int height[], int n, int m) {

```



```

8   int *s = str;
9   int *x=wx,*y=wy,*t,p;
10  int i,j;
11  for(i=0; i<m; i++)wss[i]=0;
12  for(i=0; i<n; i++)wss[x[i]=s[i]]++;
13  for(i=1; i<m; i++)wss[i]+=wss[i-1];
14  for(i=n-1; i>=0; i--)sa[--wss[x[i]]]=i;
15  for(j=1,p=1; p<n && j<n; j*=2,m=p) {
16      for(i=n-j,p=0; i<n; i++)y[p++]=i;
17      for(i=0; i<n; i++)if(sa[i]-j>=0)y[p++]=sa[i]-j;
18      for(i=0; i<n; i++)wv[i]=x[y[i]];
19      for(i=0; i<m; i++)wss[i]=0;
20      for(i=0; i<n; i++)wss[wv[i]]++;
21      for(i=1; i<m; i++)wss[i]+=wss[i-1];
22      for(i=n-1; i>=0; i--)sa[--wss[wv[i]]]=y[i];
23      for(t=x,x=y,y=t,p=1,i=1,x[sa[0]]=0; i<n; i++)
24          x[sa[i]]=cmp(y,n,sa[i-1],sa[i],j)?p-1:p++;
25  }
26  for(int i=0; i<n; i++) rank[sa[i]]=i;
27  for(int i=0,j=0,k=0; i<n; height[rank[i++]]=k)
28      if(rank[i]>0)
29          for(k?k--:0,j=sa[rank[i]-1];
30              i+k < n && j+k < n && str[i+k]==str[j+k];
31              k++);
32  }

```

7.5.1 Longest common prefix

```

1   int lcp(int x,int y) {
2       if (x > y) swap(x,y);
3       if (x == y)
4           return len-sa[x];//NOTICE!
5       x++;
6       int k = lent[y-x+1];
7       return min(f[x][k],f[y-(1<<k)+1][k]);
8   }
9   //Interval
10  void getinterval(int pos,int comlen,int& pl,int& pr) {
11      int l,r,mid,cp;
12      l = 0;
13      r = pos;
14      while (l < r) {
15          mid = l+r>>1;
16          cp = lcp(mid,pos);
17          if (cp < comlen)
18              l = mid+1;
19          else
20              r = mid;
21      }
22      pl = l;
23      l = pos;
24      r = len-1;
25      while (l < r) {
26          mid = l+r+1>>1;
27          cp = lcp(pos,mid);

```

```

28      if (cp < comlen)
29          r = mid-1;
30      else
31          l = mid;
32  }
33  pr = l;
34  }

```

7.6 Smallest representation

```

1   int Gao(char a[],int len) {
2       int i = 0,j = 1,k = 0;
3       while (i < len && j < len && k < len) {
4           int cmp = a[(j+k)%len]-a[(i+k)%len];
5           if (cmp == 0)
6               k++;
7           else {
8               if (cmp > 0)
9                   j += k+1;
10              else
11                  i += k+1;
12              if (i == j) j++;
13              k = 0;
14          }
15      }
16      return min(i,j);
17  }

```

8 Tool

8.1 Bit compression

```

1   int bit[5];
2   inline int getbit26(int sta, int pos) {
3       return sta / bit[pos] % bit[1];
4   }
5   inline int setbit26(int sta, int pos, int val) {
6       return sta / bit[pos + 1] * bit[pos + 1] + val * bit[pos] + sta % bit[pos]
7   ];
8   //bin
9   inline int getbit(int sta, int pos) {
10      return (sta >> pos) & 1;
11  }
12  inline int setbit(int sta, int pos, int val) {
13      return ((sta >> (pos + 1)) << (pos + 1)) | (val << pos) | (sta & ((1 <<
14      pos) - 1));

```

8.2 Hash map

```

1   struct hash_map {
2       int head[MOD];
3       struct hash_tables {

```

```

4   int key1, key2;
5   long long val;
6   int next;
7 } ele[ELE];
8 int N;
9 int getHash(int key1, int key2) {
10    return (key1 * 1000000 + key2) % MOD;
11 }
12 void init() {
13     memset(head, -1, sizeof(head));
14     N = 0;
15 }
16 void clear() {
17     for (int i = 0; i < N; i++)
18         head[getHash(ele[i].key1, ele[i].key2)] = -1;
19     N = 0;
20 }
21 int fint(int key1, int key2) {
22     for (int i = head[getHash(key1, key2)]; i != -1; i = ele[i].next) {
23         if (ele[i].key1 == key1 && ele[i].key2 == key2)
24             return i;
25     }
26     return -1;
27 }
28 void insert(int key1, int key2) {
29     int tmp = getHash(key1, key2);
30     ele[N].key1 = key1;
31     ele[N].key2 = key2;
32     ele[N].val = 0;
33     ele[N].next = head[tmp];
34     head[tmp] = N++;
35 }
36 long long get(int key1, int key2) {
37     int tmp = fint(key1, key2);
38     if (tmp == -1) {
39         insert(key1, key2);
40         return ele[N - 1].val;
41     } else
42         return ele[tmp].val;
43 }
44 void set(int key1, int key2, long long val) {
45     int tmp = fint(key1, key2);
46     if (tmp == -1) {
47         insert(key1, key2);
48         ele[N - 1].val = val;
49     } else
50         ele[tmp].val = val;
51 }
52 void add(int key1, int key2, long long val) {
53     int tmp = fint(key1, key2);
54     if (tmp == -1) {
55         insert(key1, key2);
56         ele[N - 1].val += val;
57     } else

```

```

58     ele[tmp].val += val;
59 }
60 };

```

8.3 120 bit integer

```

1 struct integer {
2     long long pa, pb;
3     integer() {}
4     integer(long long _pa, long long _pb) {
5         pa = _pa;
6         pb = _pb;
7     }
8     integer negate() {
9         if (pa == 0 && pb == 0)
10             return integer(pa, pb);
11         else if (pa == 0)
12             return integer(pa, -pb);
13         else
14             return integer(-pa, pb);
15     }
16     integer operator +(const integer& b) const {
17         integer ret = integer(pa + b.pa, pb + b.pb);
18         if (ret.pb >= MOD) {
19             ret.pa += 1;
20             ret.pb -= MOD;
21         }
22         return ret;
23     }
24     bool operator <(const integer& b) const {
25         if (pa == b.pa)
26             return pb < b.pb;
27         return pa < b.pa;
28     }
29 };

```

8.4 Bash script

```

1 while true; do
2     ./gen > input
3     ./sol < input > output.sol
4     ./bf < input > output.bf
5
6     diff output.sol output.bf
7     if [ $? -ne 0 ] ; then break; fi
8 done

```

8.5 Codeblocks settings

```

1 |gnome-terminal -t $TITLE -x

```