1. Min Heap -> Min Priority Queue

```
#define PP(x) (((x)-1)/2)   //parent of head node
#define LL(x) ((x)*2+1)     //left child
#define RR(x) (((x)+1)*2)   //right child

struct MinHeap {
    int *A,*d,*O;   //min priority queue based on d[] array
    int size;
    MinHeap(int *B,int sz) {
        size = sz, d = B;
        A = new int [size], O = new int [size];
        while(--sz >= 0) A[sz] = O[sz] = sz;
    }
    ~MinHeap() {
        delete[] A; delete[] O;
    }
    void swp(int x,int y) {
        std::swap( O[A[x]] , O[A[y]] );
        std::swap( A[x] , A[y] );
    }
    void minheapify(int p) {
        int L = LL(p), R = RR(p), q = p; //q == least
        if(L < size && d[A[L]] < d[A[q]] ) q = L;//1. comparator, d[..] < d[..]
        if(R < size && d[A[R]] < d[A[q]] ) q = R;//2. comparator, d[..] < d[..]
        if(q != p) { //new least found
            swap(p,q);
            minheapify(q);
        }
    }
    void build() {
        int i = size / 2;
        while(i >= 0) minheapify(i--);
    }
    int exmin() {
        int ret = A[0];
        swp(0,--size);
        minheapify(0);
        return ret;
    }
    void dec(int p,int key) {
        int x = O[p];
        //assert(d[A[x]] >= key);
        d[A[x]] = key;
        while(x > 0 && d[A[PP(x)]] > d[A[x]] ) { //3. comparator, d[..] > d[..]
            swp(x, PP(x));
            x = PP(x);
        }
    }
};
```

2. Dijkstra

```
void dijkstra(int src) { //Dijkstra
    int taken = 0, i,u,v,c;

    rep(i,n) { d[i] = inf; pi[i] = -1; } d[src] = 0; //init
    MinHeap mh(d,n);
    mh.build();

    while(taken++ < n) {
        u = mh.exmin();
```

```
        rep(i,edge[u].size()) {
            v = edge[u][i].first; c = edge[u][i].second;
            //relax edge (u,v) with cost c
            if(d[v] > d[u] + tax[v] + c) {
                d[v] = d[u] + tax[v] + c; mh.dec(v,d[v]); pi[v] = u;
            }
        }
    }//d[i] contains distance of shortest path from src
}
```

3. MaxFlow – Push Relabel (N^3)

```
int h[MAXV],f[MAXV][MAXV],cf[MAXV][MAXV],c[MAXV][MAXV],e[MAXV];
vector<int> G[MAXV];
int n,s,t;

void initialize_preflow()
{
    int sz,i,v;
    memset(h,0,sizeof(int)*n);
    h[s]=n;
    memset(e,0,sizeof(int)*n);
    for(i=0;i<n;++i)
        memset(f[i],0,sizeof(int)*n);
    sz=G[s].size();
    for(i=0;i<sz;++i)
    {
        v=G[s][i];
        f[s][v]=c[s][v]; f[v][s]=-c[s][v];
        e[v]=c[s][v]; e[s]-=c[s][v];
        cf[s][v]=c[s][v]-f[s][v]; cf[v][s]=c[v][s]-f[v][s];
    }
}

void push(int u,int v)
{
    int temp=MIN(e[u],cf[u][v]);
    f[u][v]=f[u][v]+temp; f[v][u]=-f[u][v];
    e[u]-=temp; e[v]+=temp;
    cf[u][v]=c[u][v]-f[u][v]; cf[v][u]=c[v][u]-f[v][u];
}

void MAXFLOW()
{
    initialize_preflow();
    queue<int> Q;
    int l[MAXV];
    int u,v,m,i,sz;
    memset(l,0,sizeof(int)*n);
    sz=G[s].size();
    for(i=0;i<sz;++i)
    {
        if(G[s][i]!=t)
        {
            Q.push(G[s][i]);
            l[G[s][i]]=1;
        }
    }

    while(!Q.empty()) {
        u=Q.front(); m=-1; sz=G[u].size();
        for(i=0;i<sz && e[u];++i) {
            v=G[u][i];
```

```
            if(cf[u][v] > 0) {
                if(h[u] > h[v]) {
                    push(u,v);
                    if(!l[v] && v!=s && v!=t) {l[v]=1; Q.push(v);}
                }
                else if(m==-1) m=h[v];
                else if(h[v]<m) m=h[v];
            }

            if(e[u]) h[u]=1+m;
            else { l[u]=0; Q.pop(); }
        }
    }
}

int main() {
//  GRAPH();
    MAXFLOW();
    return 0;
}
```

4. Weighted Bi partite Matching (N^3)

```
#define SZ 2000

int w[SZ][SZ];
int matched1[SZ],matched2[SZ],f1[SZ],f2[SZ],gf[SZ][SZ];
int S[SZ],T[SZ],vis[SZ],p[SZ];

int n;

int WBPM()
{
    int i,j,x,cnt,d,u,t,cost;
    queue<int> Q;

    for(i=1;i<=n;i++) f2[i]=0, matched1[i]=matched2[i]=-1;
    cnt=0;
    for(i=1;i<=n;i++) {
        f1[i]=0;
        for(j=1;j<=n;j++) if(f1[i]<w[i][j]) f1[i]=w[i][j];
        for(j=1;j<=n;j++) gf[i][j]=f1[i]==w[i][j];
        for(j=1;j<=n;j++) if(gf[i][j] && matched2[j]==-1) {
matched1[i]=j; matched2[j]=i; break;
}
        cnt+=matched1[i]!=-1;
    }

    while(cnt!=n)
    {
        while(!Q.empty()) Q.pop();
        for(i=1;i<=n;i++) S[i]=T[i]=vis[i]=0;
        for(i=1;i<=n;i++)  {
            if(matched1[i]==-1) {
                for(j=1;j<=n;j++)  if(gf[i][j]) {
                    vis[j]=1; p[j]=i; Q.push(j);
                }
                S[i]=1;
                break;
            }
        }

        while(1)
```

```
        {
            if(Q.empty()) {
                d=1000000000;
                for(i=1;i<=n;i++) if(S[i])
                    for(j=1;j<=n;j++) if(!T[j])
                        if(d > f1[i]+f2[j]-w[i][j])
                            d=f1[i]+f2[j]-w[i][j];

                for(i=1;i<=n;i++) if(S[i]) f1[i]-=d;
                for(i=1;i<=n;i++) if(T[i]) f2[i]+=d;

                for(i=1;i<=n;i++)
                    for(j=1;j<=n;j++) {
                        gf[i][j]=f1[i]+f2[j]==w[i][j];
                        if(gf[i][j] && !T[j] && S[i]) {
                            vis[j]=1; p[j]=i; Q.push(j);
                        }
                    }
            }
            else
            {
                u=Q.front(); Q.pop();
                if(matched2[u]==-1) {
                    x=u;
                    while(x!=-1) {
                        t=matched1[p[x]];
                        matched2[x]=p[x];
                        matched1[p[x]]=x;
                        x=t;
                    }
                    cnt++;
                    break;
                }
                else {
                    S[matched2[u]]=1;
                    T[u]=1;
                    u=matched2[u];
                    for(j=1;j<=n;j++) if(gf[u][j] && !vis[j]) {vis[j]=1; p[j]=u;
Q.push(j);}
                }
            }
        }
    }

    cost=0;
    printf("%d",f1[1]); for(i=2;i<=n;i++) printf(" %d",f1[i]); printf("\n");
    printf("%d",f2[1]); for(i=2;i<=n;i++) printf(" %d",f2[i]); printf("\n");
    for(i=1;i<=n;i++) cost+=f1[i]+f2[i];

    return cost;
}

int main()
{
    int i,j;
    while(scanf("%d",&n)!=EOF)
    {
        for(i=1;i<=n;i++) for(j=1;j<=n;j++) scanf("%d",&w[i][j]);
        printf("%d\n",WBPM());
    } return 0;
}
```

5. Mincost Flow

```
int ans,flow;
int SOURCE,SINK;

struct EDGE
{
    int u,v,t,cf;
    EDGE(int a,int b,int c,int d) { u=a; v=b; t=c; cf=d;}
};

vector<EDGE>E;

int d[MAXN],pi[MAXN],dd[MAXN],pe[MAXN];

bool BF() {
    int i,j,ok;
    for(i=SOURCE;i<=SINK;i++) d[i] = INF;
    d[SOURCE] = 0;
    for(i=SOURCE;i<SINK;i++)  {
        ok = 0;
        for(j=0;j<E.size();j++) {
            if(d[E[j].v] > d[E[j].u] + E[j].t && E[j].cf > 0) {
                ok = 1; d[E[j].v] = d[E[j].u] + E[j].t;
                pi[E[j].v] = E[j].u; pe[E[j].v] = j;
            }
        } if(!ok) break;
    }
    return d[SINK] < INF;
}

void MINCOST_FLOW() {
    int x,mn,t,z;
    while(BF())  {

        //find min
        x = SINK; mn = INF;
        while(x) { //x!=SOURCE
            t = pi[x]; z = pe[x]; mn = _min(mn,E[z].cf);
            x = t;
        }

        //create residual network
        flow += mn; ans += d[SINK]*mn;
        x = SINK;
        while(x) {
            t = pi[x]; z = pe[x]; E[z].cf -= mn; E[z^1].cf += mn;
            x=t;
        }
    }
}

//when creating the edge list, add two entries for each edge – with indices
//consecutive i.e. 0 & 1 or 2 & 3. The cost of back edge will be the negative of the
//front edge cost. Capacities will be 'cap of the edge' and 0 respectively.
```

6. Euler Circuit

```
list< int > cyc;
void euler( list< int >::iterator i, int u ) {
    int v;
    for(v=1;v<=n;v++){
        if( mat[u][v] ) {
            mat[u][v]--;
            mat[v][u]--;
            euler( cyc.insert( i, u ) , v );
        }
    }
}

////
cyc.clear();
euler( cyc.begin(), 1 );

list<int>::iterator iter;
for( i=0,iter = cyc.begin(); iter != cyc.end(); iter++,i++ )
    ans[i] = node[*iter];
////
```

7. Strongly Connected Component

1 Call DFS(G) to compute finishing time f[u] for each vertex u

2 Compute G^T

3 Call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing f[u].

4 Output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected component.

8. Binary Search Tree

```
TreeSearch(x, k)
    if(x == NIL || k == key[x]) return x;
    if(k < key) return TreeSearch(left[x],k);
    else return TreeSearch(righ[x],k);

TreeMin(x)
    while(left[x] != NIL) x = left[x];
    return x;

TreeSuccessor(x)
    if(right[x] != NIL) return TreeMin(right[x]);
    y = p[x];
    while(y != NIL && x == right[y]) { x = y; y = p[y]; }
    return y;

TreeInsert(T,z)
    y = NIL;
    x = roor[T];
    while( x != NIL) {
        y = x;
        if(key[z] < key[x]) x = left[x]; else x = right[x];
    } p[z] = y;
    if(y == NIL) root[T] = z;
    else {
        if(key[z] < key[y]) left[y] = z;
        else right[y] = z;
    }
```

```
TreeDelete(T,z)
    if(left[z] == NIL || right[z] == NIL) y = z;
    else y = TreeSuccessor(z);
    if(left[z] != NIL) x = left[y]; else x = right[y];
    if(x != NIL) p[x] = p[y];
    if(p[y] == NIL) root[T] = x;
    else {
        if(y == left[p[y]]) left[p[y]] = x;
        else right[p[y]] = x;
    }
    if(y != z) {
        key[z] = key[y];
        copy y's satellite data into z
    } return y;
```

9. Binary Indexed Tree

1D

```
//finding f[idx]
int read(int idx){
    int sum = 0;
    while (idx > 0){
        sum += tree[idx];
        idx -= (idx & -idx);
    }
    return sum;
}

//f[idx]+=val
void update(int idx ,int val){
    while (idx <= MaxVal){
        tree[idx] += val;
        idx += (idx & -idx);
    }
}

//find f[idx] with cumiliative frequency = cumFre
int findG(int cumFre){
    int idx = 0;
    while ((bitMask != 0) && (idx < MaxVal)){
        int tIdx = idx + bitMask;
        if (cumFre >= tree[tIdx]){
            // if current cumulative frequency is equal to cumFre,
            // we are still looking for higher index (if exists)
            idx = tIdx;
            cumFre -= tree[tIdx];
        }
        bitMask >>= 1;
    }
    if (cumFre != 0)    return -1;
    else return idx;
}
```

2D

```
void update(int x , int y , int val){
    int y1;
    while (x <= max_x){
        y1 = y;
        while (y1 <= max_y){
            tree[x][y1] += val;
            y1 += (y1 & -y1);
        }
        x += (x & -x);
    }
}
```

10. Interval Tree

```
#define MAX         650010
#define ROOT            1
#define PAR(x)      ((x)/2)
#define LEFT(x)     (2*(x))
#define RIGHT(x)        (2*(x)+1)
#define X(m,n)      ((node1.s[m]*node2.s[n])%prime)
typedef long long LL;

struct Node{
    int left,right;
    LL s[5];
    Node(int _left = -1,int _right = -1){
        left = _left;right = _right;
        s[0] = 1;s[1] = 0;s[2] = 0;s[3] = 0;s[4] = 0;
    }
}node[MAX];

int n;
LL a[MAX],prime;

void cross(Node &par,Node &node1,Node &node2){
    par.s[0] = 1;
    par.s[1] = (node1.s[1] + node2.s[1]) % prime;
    par.s[2] = (node1.s[2] + X(1,1) + node2.s[2]) % prime;
    par.s[3] = (node1.s[3] + X(1,2) + X(2,1) + node2.s[3]) % prime;
    par.s[4] = (node1.s[4] + X(1,3) + X(2,2) + X(3,1) + node2.s[4]) % prime;
}

void construct(int index,Node &par,int lo,int hi){
    par = Node(lo,hi);
    Node &node1 = node[LEFT(index)];
    Node &node2 = node[RIGHT(index)];
    if(lo<hi){
        int mid = (lo+hi)/2;
        construct(LEFT(index),node1,lo,mid);
        construct(RIGHT(index),node2,mid+1,hi);
        cross(par,node1,node2);
    }
    else par.s[1] = a[lo];
}

int pos;
LL delta;

void update(int index){
    if(node[index].left == node[index].right){
        node[index].s[1] = (node[index].s[1] + delta) % prime;
```

```
        return ;
    }
    int lefti  = LEFT(index);
    int righti = RIGHT(index);
    if(pos <= node[lefti].right) update(lefti);
    else update(righti);
    cross( node[index] , node[lefti] , node[righti] );
}

//valid for L <= .. <= R
int L,R,k;
int get(int index,Node &par){
    int i;
    if( node[index].right < L || R < node[index].left){
        for(i=1;i<5;i++) par.s[i] = 0;
        par.s[0] = 1;
        return 0;
    }

    if( L <= node[index].left && node[index].right <= R ){
        par = node[index];
        return 1;
    }

    int l,r;
    Node newLeftNode;
    Node newRightNode;

    l = get( LEFT(index) , newLeftNode);
    r = get( RIGHT(index) , newRightNode);

    if(l && r)      cross( par , newLeftNode , newRightNode );
    else if(l)      par = newLeftNode;
    else if(r)      par = newRightNode;
    return 1;
}

void MOD(LL &x){
    if(x < 0){
        x = (-x) % prime;
        x = (prime - x)%prime;
    }
    else x = x % prime;
}

int main(){
    int i;
    int n,query;
    Node outNode;
    char buf[5];
    while(scanf("%d%d%I64d",&n,&query,&prime)==3){
        for(i=1;i<=n;i++){
            scanf("%I64d",&a[i]);
            MOD(a[i]);
        }
        construct(ROOT,node[ROOT],1,n);
        while(query--){
            scanf("%s",buf);
            if(buf[0]=='I'){
                scanf("%d%I64d",&pos,&delta);

                MOD(delta);
                update(ROOT);
            }
            else{
```

```
                scanf("%d%d%d",&L,&R,&k);
                get(ROOT,outNode);

                for(i=0;i<=k;i++){
                    if(i)   printf(" ");
                    printf("%I64d",outNode.s[i]);
                } printf("\n");
            }
        }
    }
    return 0;
}
```

11. Suffix Array

```
memset(A, 0, sizeof(A));
for (i = 0; i < N; ++i) A[(int)(S[i]-'A')] = 1;
for (i = 1; i < 26; ++i) A[i] += A[i-1];
for (i = 0; i < N; ++i) o[0][i] = A[(int)(S[i]-'A')];
x=0;
for (j = 0, jj = 1, k = 0; jj < N && k < 2; ++j, jj <<= 1) {
    memset(A, 0, sizeof(A));
    memset(B, 0, sizeof(B));
    for (i = 0; i < N; ++i) {
        ++A[ t[i][0] = o[x][i] ];
        ++B[ t[i][1] = (i+jj<N) ? o[x][i+jj] : 0 ];
    }
    for (i = 1; i <= 2*N; ++i) {
        A[i] += A[i-1];
        B[i] += B[i-1];
    }
    for (i = 2*N-1; i >= 0; --i) C[--B[t[i][1]]] = i;
    for (i = 2*N-1; i >= 0; --i) D[--A[t[C[i]][0]]] = C[i];
    x ^= 1;
    o[x][D[0]] = k = 1;
    for (i = 1; i < 2*N; ++i)
        o[x][D[i]]=(k += (t[D[i]][0] != t[D[i-1]][0] || t[D[i]][1] != t[D[i-1]][1]));

}
```

12. Least Common Ancestor (LCA)

```
void process3(int N, int T[MAXN], int P[MAXN][LOGMAXN])
{
    int i, j;
    //we initialize every element in P with -1
    for (i = 0; i < N; i++) for (j = 0; (1 << j) < N; j++) P[i][j] = -1;

    //the first ancestor of every node i is T[i]
    for (i = 0; i < N; i++) P[i][0] = T[i];

    //bottom up dynamic programming
    for (j = 1; (1 << j) < N; j++) for (i = 0; i < N; i++) if (P[i][j - 1] != -1)
        P[i][j] = P[P[i][j - 1]][j - 1];
}

int query(int N, int P[MAXN][LOGMAXN], int T[MAXN], int L[MAXN], int p, int q) {
    int tmp, log, i;
    //if p is situated on a higher level than q then we swap them
    if (L[p] < L[q]) tmp = p, p = q, q = tmp;

    //we compute the value of [log(L[p)]
```

```
    for (log = 1; 1 << log <= L[p]; log++) ;
    log--;

    //we find the ancestor of node p situated on the same level
    //with q using the values in P
    for (i = log; i >= 0; i--) if (L[p] - (1 << i) >= L[q]) p = P[p][i];
    if (p == q) return p;

    //we compute LCA(p, q) using the values in P
    for (i = log; i >= 0; i--) if (P[p][i] != -1 && P[p][i] != P[q][i])
        p = P[p][i], q = P[q][i];
    return T[p];
}
```

13. Union of Rectangle in Nlog(N)

```
struct Axis {
    int value,type,id;
};
bool cmp(Axis A, Axis B) {
    if(A.value < B.value) return 1;
    if(A.value == B.value && A.type < B.type) return 1;
    return 0;
}
struct Node {
    int low,high;
    int left,right;
    int range,count;
};
struct Rectangle {
    int size;
    int lx[MAX],hx[MAX],ly[MAX],hy[MAX];
    Axis H[MAX<<1],V[MAX<<1];
    int Hsize,Vsize;

    Node NODE[1<<16];
    int avail;
    int INF_COUNT;
    int root;

    Rectangle() { INF_COUNT=1000000; }
    void INIT()
    {
        int i,j;
        for(j=0,i=0;i<size;i++)
        {
            H[j].value=ly[i]; H[j].type=0; H[j].id=i;
            V[j].value=lx[i];   V[j].type=0; V[j++].id=i;
            H[j].value=hy[i]; H[j].type=1; H[j].id=i;
            V[j].value=hx[i]; V[j].type=1; V[j++].id=i;
        }
        sort(H,H+j,cmp);
        sort(V,V+j,cmp);
        Hsize=Vsize=j;
        for(j=0,i=0;i<Hsize;i++)
        {
            if(H[j].value == H[i].value) continue;
            H[++j]=H[i];
        }
        while( j&(j-1) )
        {
            H[j+1]=H[j];
            ++j;
```

```
        }
        Hsize=j+1;
    }

    int MYTREE(int from,int to) {
        int here=avail++;
        NODE[here].low=H[from-1].value;
        NODE[here].high=H[to].value;
        NODE[here].range=0;
        NODE[here].count=0;
        if(from==to)
        {
            if(NODE[here].low==NODE[here].high)
                NODE[here].count=INF_COUNT;
            NODE[here].left=NODE[here].right=-1;
            return here;
        }
        NODE[here].left=MYTREE(from,(from+to-1)>>1);
        NODE[here].right=MYTREE(((from+to-1)>>1)+1,to);
        return here;
    }

    void INSERT(int low,int high,int at) {
        if(NODE[at].low==low && NODE[at].high==high)
        {
            NODE[at].count++;
            NODE[at].range=high-low;
            return;
        }
        if(NODE[NODE[at].left].high>low)
INSERT(low,min(NODE[NODE[at].left].high,high),NODE[at].left);
        if(NODE[NODE[at].right].low<high)
INSERT(max(low,NODE[NODE[at].right].low),high,NODE[at].right);
        if(NODE[at].count==0)
            NODE[at].range=NODE[NODE[at].left].range + NODE[NODE[at].right].range;
    }

    void REMOVE(int low,int high,int at) {
        if(NODE[at].low==low && NODE[at].high==high)
        {
            NODE[at].count--;
            if(NODE[at].count==0)
            {
                if(NODE[at].left==-1) NODE[at].range=0;
                else NODE[at].range=NODE[NODE[at].left].range +
NODE[NODE[at].right].range;
            }
            return;
        }
        if(NODE[NODE[at].left].high>low)
REMOVE(low,min(NODE[NODE[at].left].high,high),NODE[at].left);
        if(NODE[NODE[at].right].low<high)
REMOVE(max(low,NODE[NODE[at].right].low),high,NODE[at].right);
        if(NODE[at].count==0)
            NODE[at].range=NODE[NODE[at].left].range + NODE[NODE[at].right].range;
    }
    int area() {
        if(size==0) return 0;
        int ans=0,now,prev,current;
        INIT();
        avail=0;
        root=MYTREE(1,Hsize-1);
        now=0;
        prev=V[0].value;
        while(now<Vsize)
```

```
        {
            current=V[now].value;
            ans+=NODE[root].range*(current-prev);
            prev=current;
            for(;V[now].value==current && now<Vsize;now++)
            {
                if(V[now].type==0)
                    INSERT(ly[V[now].id],hy[V[now].id],root);
                else
                    REMOVE(ly[V[now].id],hy[V[now].id],root);
            }
        }

        return ans;
    }
};

Rectangle R;

int main() {
    int n,i;
    scanf("%d",&n);
    R.size=n;
    for(i=0;i<n;i++)
        scanf("%d%d%d%d",&R.lx[i],&R.ly[i],&R.hx[i],&R.hy[i]);
    printf("%d\n",R.area());
    return 0;
}
```

14. Point Classification

```
/* p2 on which side of p0(origin)-p1(destination) */
int classify(Point p0,Point p1,Point p2){
    Point a,b;
    double t;
    a.x=p1.x-p0.x;
    a.y=p1.y-p0.y;
    b.x=p2.x-p0.x;
    b.y=p2.y-p0.y;
    t=a.x*b.y-a.y*b.x;
    if(t>0.0) return LEFT;
    if(t<0.0) return RIGHT;
    if((a.x*a.x+a.y*a.y)<(b.x*b.x+b.y*b.y)) return BEYOND;
    if((a.x>0.0 && b.x<0.0) || (a.x<0.0 && b.x>0.0)) return BEHIND;
    if((a.y>0.0 && b.y<0.0) || (a.y<0.0 && b.y>0.0)) return BEHIND;
    if(p0.x==p2.x && p0.y==p2.y) return ORIGIN;
    if(p1.x==p2.x && p1.y==p2.y) return DESTINATION;
    return BETWEEN;
}
```

15. Cohen-Sutherland Polygon Clipping

```
typedef struct {
    double x,y;
}Point;

enum { LEFT=0,RIGHT,BEHIND,BEYOND,ORIGIN,DESTINATION,BETWEEN};
enum { COLLINEAR=0,PARALLEL,SKEW,SKEW_CROSS,SKEW_NO_CROSS};

int intersect(Point a,Point b,Point c,Point d,double *t) {
    double denom,num;
    int aclass;
    Point n,ba,ac;
```

```
    n.x = d.y - c.y;    ba.x = b.x - a.x;   ac.x = a.x - c.x;
    n.y = c.x - d.x;    ba.y = b.y - a.y;   ac.y = a.y - c.y;
    denom = dotProduct(n,ba);
    if(is_equal(denom,0.0)) {
        aclass = classify(c,d,a);
        if(aclass == LEFT || aclass == RIGHT) return PARALLEL;
        else return COLLINEAR;
    }
    num = dotProduct(n,ac);
    *t = -num / denom;
    return SKEW;
}

double dotProduct(Point a,Point b) { return a.x * b.x + a.y * b.y; }


int lineClip(Point a,Point b,Point p[],int n) {
    Point r[SIZE];
    Point org,dest,crosspt;
    int orgInside,destInside;
    double t;
    int i,j;

    p[n] = p[0];
    for(i=j=0;i<n;i++) {
        org = p[i];     dest = p[i+1];
        orgInside = (classify(a,b,org)!=LEFT);
        destInside = (classify(a,b,dest)!=LEFT);
        if(orgInside != destInside) {
            intersect(a,b,org,dest,&t);
            crosspt.x=a.x+t*(b.x - a.x); crosspt.y=a.y + t*(b.y-a.y);
        }

        if(orgInside && destInside) r[j++] = dest;
        else if(orgInside && !destInside) {
        if((!is_equal(org.x,crosspt.x))||(!is_equal(org.y,crosspt.y)))
                r[j++] = crosspt;
        }
        else if(!orgInside && !destInside) ;
        else {
            r[j++] = crosspt;
    if((!is_equal(dest.x,crosspt.x))||(!is_equal(dest.y,crosspt.y)))
                r[j++] = dest;
        }
    }
    for(i=0;i<j;i++)
        p[i] = r[i];
    return j;
}

int polygonClip(Point subject[],int m,Point clipper[],int n) {
// polygons have to be in clockwise order
    int tm,i;
    clipper[n] = clipper[0];
    for(i=0;i<n;i++) {
        tm = lineClip(clipper[i],clipper[i+1],subject,m);
        m = tm;
    }
    return m;
}
```

16. Convexhull(Integer Calculation Only)

```
int ns,s[MAX];
void convexhull(int n,Point *p,vector<int> &vindex){
    int i;
    vindex.clear();
    if(n<=2){
        for(i=0;i<n;i++)
            vindex.push_back(i);
        return;
    }

    //lower hull : 1) lesser X, then 2) lesser Y
    ns = 1; s[0] = 0;
    for(i=1;i<n;i++){
        if(p[i]==p[i-1])
            continue;
        while( ns >= 2 && A( p[s[ns-2]],p[s[ns-1]],p[i]) <= 0 )
            ns--;
        s[ns++] = i;
    }

    for(i=0;i<ns;i++)
        vindex.push_back(s[i]);

    //upper hull : 1) greater X, then 2) greater Y
    ns = 1;
    s[0] = n-1;
    for(i=n-2;i>=0;i--){
        if(p[i]==p[i+1])
            continue;
        while( ns >= 2 && A( p[s[ns-2]],p[s[ns-1]],p[i]) <= 0 )
            ns--;
        s[ns++] = i;
    }

    if( !(p[vindex[vindex.size()-1]] == p[s[0]]) )
        vindex.push_back(s[0]);

    for(i=1;i<ns-1;i++)
        vindex.push_back(s[i]);

    if( !(p[vindex[0]] == p[s[ns-1]]) )
        vindex.push_back(s[ns-1]);
}
```

17. Closest Pair Distance nlog(n) ///// kinda naive, but OK

```
double dist(int k1,int k2){
    double d,d2 ,d3;

    if(k2-k1+1 == 1)    ////////
        return 0;
    if(k2-k1+1 == 2)
        return D1(p[k1],p[k2]);
    if(k2-k1+1 == 3){
        d  = D1(p[k1],p[k1+1]);
        d2 = D1(p[k1+1],p[k1+2]);
        d3 = D1(p[k1+2],p[k1]);
        if(d > d2)  d = d2;
        if(d > d3)  d = d3;
        return d;
    }
    int k,i,j;
```

```
    k = (k1 + k2) / 2;
    d  = dist(k1 , k);
    d2 = dist(k+1 , k2);
    if(d > d2)  d = d2;

    ns1 = 0;
    for(i = k; i>=k1 ; i--){
        if( p[k].x - p[i].x  >  d )
            break;
        s1[ ns1++ ] = p[i];
    }
    qsort( s1 , ns1 , sizeof(s1[0]) , sortY);   //ascending

    ns2 = 0;
    for(i = k+1; i<=k2 ; i++){
        if( p[i].x - p[k].x  >  d )
            break;
        s2[ ns2++ ] = p[i];
    }
    qsort( s2 , ns2 , sizeof(s2[0]) , sortY);   //ascending

    for(i=0;i<ns1;i++){
        for(j=0;j<ns2;j++){
            if(s2[j].y - s1[i].y > d)
                break;
            d2 = D1(s1[i],s2[j]);
            if(d2 < d)
                d = d2;
        }
    }
    return d;
}

////
qsort(p,n,sizeof(p[0]),sortX);  //ascending
d = dist(0,n-1);
////
```

18. Maximum Points in Circle $n^2\log(n)$

```
#define MAX 305
#define EPS 1e-4
#define _abs(x) ( ((x)>0) ? (x) : -(x) )
#define Z(x)    (_abs(x) < EPS)
#define E(x,y)  (_abs((x)-(y)) < EPS)
#define S(x)    ((x)*(x))
#define D2(a,b) (S(a.x-b.x) + S(a.y-b.y))
#define D1(a,b) (sqrt(D2(a,b)))
#define OPEN    0
#define CLOSE   1

int n;
double radius, pi = 2.*acos(0.);
struct Point{
    double x,y;
    Point(double _x=0,double _y=0){
        x = _x;
        y = _y;
    }
    void scan(){ scanf("%lf%lf",&x,&y); }
}p[MAX];
```

```
struct Event{
    double angle;
    int type,id;
    Event(double _angle=0,int _type=0,int _id=0){
        angle   = _angle;
        type    = _type;
        id      = _id;
        if(angle < 0)               angle += 2*pi;
        if(angle > 2*pi || E(angle,2*pi))   angle -= 2*pi;
    }
};

vector<Point>   vp;
vector<Event>   ve;

bool operator<(const Event &p,const Event &q){
    if( E(p.angle,q.angle) )
        return p.type < q.type; //this priority does not matter
    return p.angle < q.angle;
}

double mytan(Point p,Point p1){
    double dx = p1.x - p.x;
    double dy = p1.y - p.y;
    if(Z(dy)){
        if(dx > 0)  return 0;
        else        return pi;
    }
    if(Z(dx)){
        if(dy > 0)  return pi/2.;
        else        return 3*pi/2.;
    }
    double at = atan2(dy,dx);
    if(at < 0)  at += 2*pi;
    return at;
}

int supmax;
int taken[MAX];

void solve(Point c){
    int     i;
    int     run;
    int     smax;
    Point       center;
    double  d, theta, alpha;

    ve.clear();
    vp.clear();
    for(i=0;i<n;i++){
        d = D1(c,p[i]);
        if( !E(d,2*radius) && d > 2*radius)
            continue;

        vp.push_back(p[i]);
        if( Z(d) )
            continue;

        theta = mytan(c,p[i]);
        alpha = acos(d/2.);
        ve.push_back( Event(theta-alpha, OPEN  , vp.size()-1) );
        ve.push_back( Event(theta+alpha, CLOSE , vp.size()-1) );
    }
    sort(ve.begin(),ve.end());
```

```
    run = 0;
    //init enclosure with circle at (c.x+radius,c.y)
    center = Point(c.x + radius , c.y);
    for(i=0;i<vp.size();i++){
        d = D1(center,vp[i]);
        if( E(d,radius) || d < radius){
            run++;
            taken[i] = 1;
        }
        else
            taken[i] = 0;
    }
    smax = run;
    for(i=0;i<ve.size();i++){
        if(ve[i].type==OPEN){
            if(taken[ ve[i].id ] == 0)
                run++;
        }
        else
            run--;
        if(run > smax)  smax = run;
    }
    if(smax > supmax)
        supmax = smax;
}

int main(){
    int i;
    while(scanf("%d%lf",&n,&radius)==2 && n){
        for(i=0;i<n;i++)
            p[i].scan();
        supmax = 0;
        for(i=0;i<n;i++)
            solve(p[i]);
        printf("%d\n",supmax);
    }
    return 0;
}
```

19. Mirror Image of a Point with respect to a Line

```
void mirror(Point &p,Line l){
    double A1,B1,C1;
    double A2,B2,C2;

    A1 = l.a;   B1 =  l.b;  C1 =   l.a*p.x + l.b*p.y + 2*l.c;
    A2 = l.b;   B2 = -l.a;  C2 = - l.b*p.x + l.a*p.y;

    p.x = det(B1,B2,C1,C2) / det(A1,A2,B1,B2);
    p.y = det(C1,C2,A1,A2) / det(A1,A2,B1,B2);
}
```

20. Common(AND) area of 2 Circles

```
double area_and(Circle p,Circle q){
    double d,h,s,ta,pt,qt,ret;

    if( p.rad > q.rad )swap(p,q);
    //Now p.rad < q.rad

    d = D1(p.cen,q.cen);
    if( d < q.rad - p.rad || Z(d) || E(d,q.rad - p.rad))
        return pi*S(p.rad);
    if( d > p.rad + q.rad || E(d,p.rad + q.rad))
```

```
        return 0;

    s  = (p.rad + q.rad + d) / 2;
    ta = sqrt(s*(s-d))*sqrt((s-p.rad)*(s-q.rad));
    h  = 2*ta / d;
    pt = acos( (S(p.rad) + D2(p.cen,q.cen) - S(q.rad)) / (2.*p.rad*d) );
    qt = acos( (S(q.rad) + D2(p.cen,q.cen) - S(p.rad)) / (2.*q.rad*d) );

    return = pt*S(p.rad) + qt*S(q.rad) - h*d;
}
```

21. Line Through (x1,y1) & (x2,y2)

```
a = y1 - y2;
b = x2 - x1;
c = x1*y2 - x2*y1;
```

22. Bisector of the Line segment (x1,y1) & (x2,y2)

```
a = 2*(x1 - x2);
b = 2*(y1 - y2);
c = (x2*x2 + y2*y2) - (x1*x1 + y1*y1);
```

23. Rotation 2D - XOY to X'OY'

```
x' =  x*cos(theta) + y*sin(theta)
y' = -x*sin(theta) + y*cos(theta)
```

24. Rotation 3D - OXYZ to OX'Y'Z'

Rotation axis is **r** which is a unit vector having components rx,ry,rz and passing through the origin O.

$$\begin{bmatrix} rx*rx*V(\Phi)+C(\Phi) & rx*ry*V(\Phi)-rz*S(\Phi) & rz*rx*V(\Phi)+ry*S(\Phi) \\ rx*ry*V(\Phi)+rz*S(\Phi) & ry*ry*V(\Phi)+C(\Phi) & ry*rz*V(\Phi)-rx*S(\Phi) \\ rz*rx*V(\Phi)-ry*S(\Phi) & ry*rz*V(\Phi)+rx*S(\Phi) & rz*rz*V(\Phi)+C(\Phi) \end{bmatrix}$$

$$V(\Phi) = vers(\Phi) = 1-\cos(\Phi)$$

$$C(\Phi) = \cos(\Phi)$$

$$S(\Phi) = \sin(\Phi)$$

25. Homogeneous Matrices

Translation

$$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation(about Z)

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Global Scaling

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix}$$

s has the effect of globally reducing the coordinates if s > 1 and of enlarging the coordinates if 0 < s < 1.

26. InCircle

```
r   = area / s
I.x = (A.x*a + B.x*b + C.x) / (a+b+c)
I.y = (A.y*a + B.y*b + C.y) / (a+b+c)
```

27. CircumCircle

```
R = abc / (4*area)

//measuring the Circum_center M(x,y):
k1 = A.x*A.x - B.x*B.x + A.y*A.y - B.y*B.y;
k2 = A.x*A.x - C.x*C.x + A.y*A.y - C.y*C.y;
k3 = (A.x*C.y + B.x*A.y + C.x*B.y) - (C.x*A.y + A.x*B.y + B.x*C.y);

M.x = (k2*(A.y-B.y) - k1*(A.y-C.y))/(2.*k3);
M.y = (k1*(A.x-C.x) - k2*(A.x-B.x))/(2.*k3);
```

28. Center of Gravity

```
G.x = (A.x + B.x + C.x) / 3.;
G.y = (A.y + B.y + C.y) / 3.;
```

29. Centroid

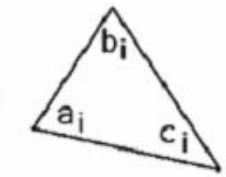
As in the calculation of the area above, xN is assumed to be x0, in other words the polygon is closed.

$$c_x = \frac{1}{6A}\sum_{i=0}^{N-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \qquad c_y = \frac{1}{6A}\sum_{i=0}^{N-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

30. Centroid of a 3D shell described by 3 vertex facets

The centroid C of a 3D object made up of a collection of N triangular faces with vertices (ai,bi,ci) is given below. Ri is the average of the vertices of the i'th face and Ai is twice the area of the i'th face. Note the faces are assumed to be thin sheets of uniform mass, they need not be connected or form a solid object.
This reduces to the equations above for a 2D 3 vertex polygon.

$$C = \frac{\sum_{i=0}^{N-1} A_i R_i}{\sum_{i=0}^{N-1} A_i}$$



$$R_i = (a_i + b_i + c_i) / 3$$

$$A_i = \|(b_i - a_i) \otimes (c_i - a_i)\|$$

31. Pick's Theorem

```
I = area + 1 - B/2,
where I = number of points inside
      B = number of points on the border.
```

32. Vector

- If 3 points A(**a**),B(**b**) & R(**r**) with common origin are collinear, scalars a,ß & ? are such that a+ß+? = 0 and **a**a+**b**ß+**r**? = 0.
- If position vectors **a**,**b**,**c**,**d** of four points A,B,C,D, no three of which are collinear and the non-zero scalars a,ß,?,d are such that a+ß+?+ d = 0 and **a**a+**b**ß+**c**?+**d**d = 0, then the four points are coplanar.
- [**a b c**]=[**b c a**]=[**c a b**]=**a**.(**b**×**c**). If vectors **a**,**b**,**c** represent the 3 sides of a parallelopiped, then its volume V = |[**abc**]|.
- Let A(**a**),B(**b**),C(**c**),D(**d**) be the 4 vertices of a tetrahedron.

Then its volume V= [(**a˜d) (b ˜d) (c ˜d)**]/6

- Let a,b,c,d are 4 vectors. Then **d** □...[**d b c**]**a** .[**d c a**]**b** ..[**d a b**]**c) / [a b c]**, [**a b c**] ? 0
- Let 2 non-coplanar straight lines be: **r** = **a**+t**b**, **r = a'+**s**b'**. Then, shortest distance between these lines = [**bb'a'˜a**] / **b** □.**b'** .
- Equation of a plane: *p* - **r**.**n** = 0. *p* is the perpendicular distance from origin. **n**

is unit normal of the plane.

- Equation of the bisectors of the angles between 2 planes $p - \mathbf{r}.\mathbf{n1} = 0$, $p - \mathbf{r}.\mathbf{n2} = 0$: If the origin & the points on the bisector lie on the same side of the given planes, the 2 perpendicular distances have the same sign. Then, the equation is $p - \mathbf{r}.\mathbf{n1} = p' - \mathbf{r}.\mathbf{n2}$ or $p - p' = \mathbf{r}.(\mathbf{n1-n2})$. When the origin & points on the other bisector are on the opposite sides of one of the given planes, then the perpendicular distances will be of opposite signs. Hence the equation is $p + p' = \mathbf{r}.(\mathbf{n1+n2})$.

33. Misc Geometric Formula

triangle	Circum Radius = a*b*c/(4*area) In Radius = area/s, where s = (a+b+b)/2 length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2 length of bisector of angle C = sqrt(ab[(a+b)*(a+b)-c*c])/(a+b)
Ellipse	Area = PI*a*b Circumference = 4a *int(0,PI/2){sqrt(1-(k*sint)*(k*sint))}dt = 2*PI*sqrt((a*a+b*b)/2) approx where k = sqrt((a*a-b*b)/a) = PI*(3*(r1+r2)-sqrt[(r1+3*r2)*(3*r1+r2)])
Spherical cap	V = (1/3)*PI*h*h*(3*r-h) Surface Area = 2*PI*r*h
Spherical Sector	V = (2/3)*PI*r*r*h
Spherical Segment	V = (1/6)*PI*h*(3*a*a+3*b*b+h*h)
Torus	V = 2*PI*PI*R*r*r
Truncated Conic	V = (1/3)*PI*h*(a*a+a*b+b*b) Surface Area = PI*(a+b)*sqrt(h*h+(b-a)*(b-a)) = PI*(a+b)*l
Pyramidal frustum	(1/3)*h*(A1+A2+sqrt(A1*A2))

34. Misc Trigonometric Functions

tan A/2 = ±sqrt((1-cos A)/(1+cos A))
= sin A / (1+cos A)
= (1-cos A) / sin A
= cosec A – cot A
sin 3A = 3*sin A – 4*sincube A
cos 3A = 4*coscube A – 3*cos A
tan 3A = (3*tan A-tancube A)/(1-3*tansq A)
sin 4A = 4*sin A*cos A – 8*sincube A*cos A
cos 4A = $8*\cos^4 A - 8*\text{cossq } A + 1$
$[r*(\text{cost}+i*\text{sint})]^p = r^p*(\cos pt+i*\sin pt)$

35. Misc Integration Formula

```
a^x => a^x/ln(a)
1/sqrt(x*x+a*a) => ln(x+sqrt(x*x+a*a))
1/sqrt(x*x-a*a) => ln(x+sqrt(x*x-a*a))
1/(x*sqrt(x*x+a*a) => -(1/a)*ln([a+sqrt(x*x+a*a)]/x)
1/(x*sqrt(a*a-x*x) => -(1/a)*ln([a+sqrt(a*a-x*x)]/x)
```

36. Misc Differentiation Formula

```
asin x =>  1 / sqrt(1 - x*x)
acos x => -1 / sqrt(1 - x*x)
atan x =>  1 / (1 + x*x)
acot x => -1 / (1 + x*x)
asec x =>  1 / [x*sqrt(x*x - 1)]
acosec x => -1 / [x*sqrt(x*x - 1)]
a^x => a^x*ln(x)
tan x => sec^2x
cot x => -cosec^2x
sec x => sec x * tan x
cosec x => -cosec x * cot x
```

37. All Real Roots of n-degree equation

```
#define INF 1000000.
#define myabs(x) ((x>0)?(x):-(x))

double f(int n,double x,double *a){
    int i;
    double s=0;
    for(i=0;i<=n;i++) s+=a[i]*pow(x,i);
    return s;
}

double bisect(int n,double *a,double lm,double um,int fx){
    double x,y,LMT = 1e-13;
    while(1){
        x=(um+lm)/2.;
        y=f(n,x,a);
        if(myabs(y) < LMT || um-lm < LMT) break;
        else{
            if(fx==1){
                if(y > LMT) um=x;
                else        lm=x;
            }
            else{
                if(y > LMT) lm=x;
                else        um=x;
            }
        }
    }
    return x;
}

void Differentiate(int n,double *a,double *b){
    int i;
    //  d/dx(f(n,x,a)) = f(n-1,x,b)
    for(i=0;i<=n;i++)
        b[i]=(i+1)*a[i+1];
    if(b[n] < 0)
        for(i=0;i<=n;i++) b[i]= -b[i];
}
```

```
int main(){
    int i,j,k,n,r[31];
    double a[30][31],x[30][31],y[31],D;
    while(scanf("%d",&n)==1 && n){
        for(i=n;i>=0;i--)
            scanf("%lf",&a[n][i]);
        if(n==1){
            printf("Equation of Degree 1\n");
            printf("Real Roots(1), Complex Roots(0)\n");
            printf("%.10lf\n",-a[1][0] / a[1][1]);
            continue;
        }
        for(i=n-1;i>=2;i--)
            Differentiate(i,a[i+1],a[i]);
        D=a[2][1]*a[2][1] - 4*a[2][2]*a[2][0];
        if(D < 0){
            x[2][0]=-INF;   x[2][1]=INF;    r[2]=2;
        }
        else{
            x[2][0]=-INF;
            x[2][1]=(-a[2][1]-sqrt(D))/(2*a[2][2]);
            x[2][2]=(-a[2][1]+sqrt(D))/(2*a[2][2]);
            x[2][3]=INF;    r[2]=4;
        }
        for(i=2;i<n;i++){
            for(j=0;j < r[i];j++)
                y[j]=f(i+1,x[i][j],a[i+1]);
            k=0;
            x[i+1][k++]=-INF;
            for(j=0;j < r[i]-1;j++){
                if( y[j] >=0 && y[j+1] <=0)
                    x[i+1][k++] = bisect( i+1 , a[i+1] , x[i][j] , x[i][j+1] , -1);
                else if( y[j] <=0 && y[j+1] >=0)
                    x[i+1][k++] = bisect( i+1 , a[i+1] , x[i][j] , x[i][j+1] ,  1);
            }
            x[i+1][k++]=INF;
            r[i+1]=k;
        }
        printf("Equation of Degree %d\n",n);
        printf("Real Roots(%d), Complex Roots(%d)\n",r[n]-2,n-r[n]+2);
        for(i=1;i<r[n]-1;i++){
            if(i>1)
                printf(" ");
            printf("%.10lf",x[n][i]);
        }
        printf("\n");
    }
    return 0;
}
```

38. Extended Euclid & Modular Linear Equation Solver

```
int extended_euclid(int a,int b,int &x,int &y) {
//finds (d,x,y) such that d = gcd(a,b) = ax + by
    if(b==0) {
        x = 1; y = 0; return a;
    }
    int xx,yy,d;
    d = extended_euclid(b,a%b,xx,yy);
    x = yy; y = xx - (a/b) * yy;
    return d;
}
```

```
bool modular_linear_eqn_solver(int a,int b,int n,int sol[],int &total) {
// solutions of ax === b (mod n)
    int x,y,d;
    d = extended_euclid(a,b,x,y);
    if(b%d==0) {
        sol[0] = (x * (b/d) ) % n;
        for(int i=0;i<d;i++)
            sol[i] = ( sol[0] + i * (n/d) ) % n;
        total = d; return 1;
    } return 0;//returns 0 if "no solution"
}
```

39. KMP Matcher

```
int lps[MAX];
void ComputePrefixFunction(int m,char *p){      //O(m)
    int k,q;
    lps[1] = 0; k = 0;
    for(q=2;q<=m;q++){
        while(k > 0 && p[k] != p[q-1]) k = lps[k];
        if(p[k]==p[q-1]) k++;
        lps[q] = k;
    }
}

vector<int> shift;
void KMP_Matcher(char *t,char *p){              //O(n)
    static int n,m,i,q;
    n = strlen(t); m = strlen(p);
    ComputePrefixFunction(m,p);
    q = 0;
    for(i=0;i<n;i++){
        while(q > 0 && p[q]!=t[i]) q = lps[q];
        if(p[q]==t[i]) q++;
        if(q==m){
            shift.push_back(1+i-m);
            q = lps[q];
        }
    }
}
```

40. Common Errors

1. Tree recursion can go very deep, so try using manual stack.
2. DP must be DP, memo must be used ;)
3. Remember using atan2(y,x) for angle / slope calculation.
   Returns angle in range –pi to pi.
4. Overflow … intermediate overflow. Use long long if not sure
5. Strategies:
   a. Use binary search to guess input size / time limit / other limit.
   b. Generate a table using inefficient code and send the table.
   c. Optimize the most critical part.
   d. Remember IDS, backtracking.