# *Contest Notebook Of*

***Rashedul Hasan Rijul***

## Table of Contents :

## 2-Sat :

```
//  1- based.....
struct two_sat{
int n,nn,m;    vector<int>G[maxm],GT[maxm],DAG[maxm],C[maxm],topo; // G=
graph.. GT= transpose graph.......
    int col[maxm],comp,comp_no[maxm],soln[maxm];
    // comp= total number component;
    // comp_no[i]= component no of node i
    // soln[i]= truth symbol of node i;
    two_sat(){}
    void init(){
        for(int i=0;i<=n+n;i++){
            G[i].clear();
            GT[i].clear();
            DAG[i].clear();
                        C[i].clear();
        }
        topo.clear();
        memset(col,0,sizeof(col));
        memset(comp_no,0,sizeof(comp_no));
        memset(soln,-1,sizeof(soln));
                comp=0;
    }
    int inv(int no){
        if(no<=n) return no+n;
        return no-n;
    }
    void OR(int u,int v){
        G[inv(v)].push_back(u);
        G[inv(u)].push_back(v);
    }
    void AND(int u,int v){
        G[u].push_back(v);
        G[v].push_back(u);
    }
    void XOR(int u,int v){
        G[inv(v)].push_back(u);
        G[u].push_back(inv(v));
        G[inv(u)].push_back(v);
        G[v].push_back(inv(u));
    }
    void XNOR(int u,int v){
        G[u].push_back(v);
        G[v].push_back(u);
        G[inv(u)].push_back(inv(v));
        G[inv(v)].push_back(inv(u));
    }
    // problem  Dependent.......
    void build_graph(){
        int u,v,op;
        nn=n+n;
        for(int i=0;i<m;i++){
            scanf("%d %d %d",&u,&v,&op);
            if(op==1) XNOR(u,v);
            else if(op==0) XOR(u,v);
        }
    }
```

```
    void make_reverse(){
        for(int i=1;i<=nn;i++){
            for(int j=0;j<G[i].size();j++){
                GT[G[i][j]].push_back(i);
            }
        }
    }
    int check_solution(){
        // Build  scc..........
        build_scc();
        for(int i=1;i<=n;i++){
            if(comp_no[i]==comp_no[inv(i)]) return 0;
        }
        return 1;
    }
    void find_solution(vector<int>&res){
        int i,j,i_p;
        for(i=1;i<=comp;i++){
            if(soln[i]==-1){
                soln[i]=0;
                i_p=comp_no[inv(C[i][0])];
                soln[i_p]=1;
                for(j=0;j<C[i_p].size();j++){
                    if(C[i_p][j]<=n) res.push_back(C[i_p][j]);
                }
            }
        }
    }

    void build_dag(){
        int i,j;
        for(i=1;i<=nn;i++){
            for(j=0;j<G[i].size();j++){
                if(comp_no[i]==comp_no[G[i][j]]) continue;
                DAG[comp_no[i]].push_back(comp_no[G[i][j]]);
            }
        }
    }
    void build_scc(){
        make_reverse();
        int i;
        for(i=1;i<=nn;i++){
            if(!col[i]) dfs(i);
        }
        for(i=topo.size()-1;i>=0;i--){
            if(!comp_no[topo[i]]){
                scc(topo[i],++comp);
            }
        }
    }
        void dfs(int s){
                if(col[s]) return ;
                col[s]=1;

                for(int i=0;i<G[s].size();i++){
                        dfs(G[s][i]);
                }
                topo.push_back(s);
        }
```

```
        void scc(int s,int comp){
            if(comp_no[s]) return ;
            comp_no[s]=comp;
                C[comp].push_back(s);

            for(int i=0;i<GT[s].size();i++){
                scc(GT[s][i],comp);
            }
        }
};
two_sat T_sat;
vector<int>res;
int main(){
    int n,m;
    while(scanf("%d",&n)==1){
        scanf("%d",&m);
        T_sat.n=n; T_sat.m=m;
        T_sat.init();
        T_sat.build_graph();
        int ans=T_sat.check_solution();
        if(ans){
            res.clear();
            T_sat.find_solution(res);
            printf("%d\n",res.size());
            for(i=0;i<res.size();i++){
                if(i) printf(" ");
                printf("%d",res[i]);
            }
            puts("");
        }
        else{
            printf("Impossible\n");
        }
    }
    return 0;
}
```

## Biconnected Component :

```
stack<pii>st_pii;
set<int>sets[maxm];
void bi_comp(int u,int v){
    while(!st_pii.empty()){
        pii now=st_pii.top(); st_pii.pop();
        sets[tot].insert(now.uu);
        sets[tot].insert(now.vv);

        if(now.uu==u && now.vv==v) break;
        if(now.uu==v && now.vv==u) break;
    }
    tot++;
}
void dfs(int s,int pre,int root){

    if(vis[s]) return;
    vis[s]=1;
    low[s]=dep[s]=tim++;
    // bi-connected with a single vertex
```

```
    if(G[s].size()==0){
        sets[tot++].insert(s);
        return ;
    }

    int i,j,k,c=0;
    for(i=0;i<G[s].size();i++){
        int d=G[s][i];
        if(d==pre) continue;
        if(vis[d] && dep[d]<dep[s]){
            st_pii.push(mp(s,d));
            low[s]=mini(low[s],dep[d]);
        }
        else if(!vis[d]){
            st_pii.push(mp(s,d));

            dfs(d,s,root); c++;

            if(low[d]>=dep[s]){
                bi_comp(s,d);
                if(s!=root){
                    is_cut[s]=1;
                }
            }
            low[s]=mini(low[s],low[d]);
        }
    }

    if(s==root && c>1){
        is_cut[s]=1;
    }
}
```

# Floyd Warshall:

```
/*
w : edge weights
d : distance matrix
p : predecessor matrix
w[i][j] = length of direct edge between i and j
d[i][j] = length of shortest path between i and j
p[i][j] = on a shortest path from i to j, p[i][j] is the last node before j.
*/
// Initialization …..
. .. . . . . ..
// Algorithm

for (k=0;k<n;k++) /* k -> is the intermediate point */
for (i=0;i<n;i++) /* start from i */
for (j=0;j<n;j++) /* reaching j */
        /* if i-->k + k-->j is smaller than the original i-->j */
```

```
        if (d[i][k] + d[k][j] < d[i][j]) {
                /* then reduce i-->j distance to the smaller one i->k->j */
                d[i][j] = d[i][k]+ d[k][j];
                /* and update the predecessor matrix */
                p[i][j] = p[k][j];
        }

void print_path (int i, int j) {
        if (i!=j) print_path(i,p[i][j]);
        print(j);
}
```

## Bellman Ford :

```
struct edge{
        int u, v,cost;};
edge edges[maxe]; int d[maxm],flag[maxm];
void bellman(int s,int n,int e){
        int i,j,k,l,u,v;
        for(i=1;i<=n;i++){
                flag[i]=0;
                d[i]=inf;
        }
        d[s]=0;
        for(i=1;i<=n+5;i++){
                for(j=0;j<e;j++){
                        u=edges[j].u;
                        v=edges[j].v;
                        if(d[v]>d[u]+edges[j].cost){
                                d[v]=d[u]+edges[j].cost;
                if(i>n){
                    // negative cycle .....
                    flag[v]=1; //  node v is in negative cycle
                }}}}
```

## Shortest Path on A DAG :

```
Void relax(Node u,Node v,double w[][]){
        if d[v] > d[u] + w[u,v] then
        d[v]:=d[u] + w[u,v] // d= distance from source.
        pi[v]:=u            // pi[i]= parent of i'th node .
}
Void DAG_SHORTEST_PATHS(Graph G,double w[][],Node s){
        topologically sort the vertices of G  // O(V+E)
        initialize_single_source(G,s)
        for each vertex u taken in topologically sorted order
        for each vertex v which is Adjacent with u
```

relax(u,v,w)
}

## Minimum Spanning Tree :

```
struct edge{
        int u,v,w;
};
edge edges[maxe]; int pre[maxm];
bool comp(edge a,edge b){
        return a.w>b.w;
}
int find(int x){
        if(pre[x]==x) return x;
        else return pre[x]=find(pre[x]);
}
        sort(edges,edges+m,comp);
        int sum=0;
        for(i=0;i<m;i++){
            k=find(edges[i].u); l=find(edges[i].v);
            if(k==l) continue;
            sum+=edges[i].w;
        }
        printf("%d\n",sum);
```

## Euler Curcuit Print :

```
// A C++ program print Eulerian Trail in a given Eulerian or Semi-Eulerian
Graph
// A class that represents an undirected graph
class Graph
{
  int V;    // No. of vertices
  list<int> *adj;    // A dynamic array of adjacency lists
public:
    // Constructor and destructor
  Graph(int V)   { this->V = V;  adj = new list<int>[V];  }
  ~Graph()       { delete [] adj;  }

  // functions to add and remove edge
  void addEdge(int u, int v) {  adj[u].push_back(v); adj[v].push_back(u); }
  void rmvEdge(int u, int v);

  // Methods to print Eulerian tour
  void printEulerTour();
  void printEulerUtil(int s);

  // This function returns count of vertices reachable from v. It does DFS
  int DFSCount(int v, bool visited[]);

  // Utility function to check if edge u-v is a valid next edge in
  // Eulerian trail or circuit
  bool isValidNextEdge(int u, int v);
};

/* The main function that print Eulerian Trail. It first finds an odd
```

```
   degree vertex (if there is any) and then calls printEulerUtil()
   to print the path */
void Graph::printEulerTour()
{
  // Find a vertex with odd degree
  int u = 0;
  for (int i = 0; i < V; i++)
      if (adj[i].size() & 1)
        {   u = i; break;  }

  // Print tour starting from oddv
  printEulerUtil(u);
  cout << endl;
}

// Print Euler tour starting from vertex u
void Graph::printEulerUtil(int u)
{
  // Recur for all the vertices adjacent to this vertex
  list<int>::iterator i;
  for (i = adj[u].begin(); i != adj[u].end(); ++i)
  {
      int v = *i;

      // If edge u-v is not removed and it's a a valid next edge
      if (v != -1 && isValidNextEdge(u, v))
      {
          cout << u << "-" << v << "  ";
          rmvEdge(u, v);
          printEulerUtil(v);
      }
  }
}

// The function to check if edge u-v can be considered as next edge in
// Euler Tout
bool Graph::isValidNextEdge(int u, int v)
{
  // The edge u-v is valid in one of the following two cases:

  // 1) If v is the only adjacent vertex of u
  int count = 0;  // To store count of adjacent vertices
  list<int>::iterator i;
  for (i = adj[u].begin(); i != adj[u].end(); ++i)
     if (*i != -1)
        count++;
  if (count == 1)
    return true;


  // 2) If there are multiple adjacents, then u-v is not a bridge
  // Do following steps to check if u-v is a bridge

  // 2.a) count of vertices reachable from u
  bool visited[V];
  memset(visited, false, V);
  int count1 = DFSCount(u, visited);

  // 2.b) Remove edge (u, v) and after removing the edge, count
```

```
  // vertices reachable from u
  rmvEdge(u, v);
  memset(visited, false, V);
  int count2 = DFSCount(u, visited);

  // 2.c) Add the edge back to the graph
  addEdge(u, v);

  // 2.d) If count1 is greater, then edge (u, v) is a bridge
  return (count1 > count2)? false: true;
}

// This function removes edge u-v from graph.  It removes the edge by
// replacing adjcent vertex value with -1.
void Graph::rmvEdge(int u, int v)
{
  // Find v in adjacency list of u and replace it with -1
  list<int>::iterator iv = find(adj[u].begin(), adj[u].end(), v);
  *iv = -1;

  // Find u in adjacency list of v and replace it with -1
  list<int>::iterator iu = find(adj[v].begin(), adj[v].end(), u);
  *iu = -1;
}
// A DFS based function to count reachable vertices from v
int Graph::DFSCount(int v, bool visited[])
{
  // Mark the current node as visited
  visited[v] = true;
  int count = 1;

  // Recur for all vertices adjacent to this vertex
  list<int>::iterator i;
  for (i = adj[v].begin(); i != adj[v].end(); ++i)
      if (*i != -1 && !visited[*i])
          count += DFSCount(*i, visited);

  return count;
}
// Driver program to test above function
int main()
{
  // Let us first create and test graphs shown in above figure
  Graph g1(4);
  g1.addEdge(0, 1);
  g1.addEdge(0, 2);
  g1.addEdge(1, 2);
  g1.addEdge(2, 3);
  g1.printEulerTour();

  Graph g2(3);
  g2.addEdge(0, 1);
  g2.addEdge(1, 2);
  g2.addEdge(2, 0);
  g2.printEulerTour();

  Graph g3(5);
  g3.addEdge(1, 0);
  g3.addEdge(0, 2);
```

```
  g3.addEdge(2, 1);
  g3.addEdge(0, 3);
  g3.addEdge(3, 4);
  g3.addEdge(3, 2);
  g3.addEdge(3, 1);
  g3.addEdge(2, 4);
  g3.printEulerTour();

  return 0;
}
```

## Maximum Bipartite Matching :

```
vector<int>v[maxm];
int lefts[maxm],rights[maxm];
bool col[maxm];
// Number of bipartite matching …….
int match(int n){
        memset(lefts,-1,sizeof(lefts));
        memset(rights,-1,sizeof(rights));
        int i,j,k,l,done=0;
        do{
                memset(col,0,sizeof(col));
                done=1;
                for(i=1;i<=n;i++){
                        if(rights[i]==-1 &&dfs(i)) done=0;
                }

        }while(!done);
        k=0;
        for(i=1;i<=n;i++){
                if(rights[i]!=-1) k++;
        }
        return k;
}
bool dfs(int s){
        if(col[s]) return 0;
        col[s]=1;
        int i,j,k,l;
        for(i=0;i<v[s].size();i++){
                k=v[s][i];
                if(lefts[k]==-1){
                        rights[s]=k;
                        lefts[k]=s;
                        return 1;
                }
                else if(dfs(lefts[k])){
```

```
                                rights[s]=k;
                                lefts[k]=s;
                                return 1;
                        }
                }
                return 0;}
```

## Stable Marriage Problem :

```
/*
Problem : Loj 1400 ( Employment).
*/
vector<int>v[maxm];
int left[maxm],right[maxm],mat[maxm][maxm],matt[maxm][maxm],n,col[maxm];
// mat=left matrix , matt= right matrix ..
void match(int n){
        memset(col,0,sizeof(col));
        memset(left,-1,sizeof(left));
        memset(right,-1,sizeof(right));
        int i,j,k,done;
        do{
                memset(col,0,sizeof(col));
                done=1;
                for(i=1;i<=n;i++){
                        if(right[i]==-1&&dfs(i)) done=0;
                }
        }while(!done);

        for(i=1;i<=n;i++){
                printf(" (%d %d)",i,right[i]);
        }
        printf("\n");
}
bool dfs(int s){
        if(col[s]) return 0;
        col[s]=1;
        int i,j,k;
        for(j=0;j<v[s].size();j++){
                i=v[s][j];
                if(left[i]==-1){
                        left[i]=s;
                        right[s]=i;
                        return 1;
                }
                else{
                        k=left[i];
```

```
                        if(matt[i][k]>matt[i][s]){
                                right[k]=-1;
                                right[s]=i;
                                left[i]=s;
                                return 1;
                        }
                }
        }
        return 0;
}
```

## Max – Flow :

*Problem : Uva 10480 Sabotage .*
*Algo : Max-flow/Min-cut .*

```
struct node{
        int no;
        int cost;
};
int n,m,tot,mat[maxm][maxm],pre[maxm],cap[maxm][maxm],col[maxm];
queue<int>q;
int in(int x){
        return 2*x-1;
}
int out(int x){
        return 2*x;
}
int comp(int x){
        if(x%2) return ((x+1)/2);
        else return x/2;
}
void ford(int s,int t);
int bfs(int s,int t);
int main(){
        int i,j,k,l,test,t=1;
        while(scanf("%d %d",&n,&m)==2){
                if(!n&&!m) break;
                memset(mat,0,sizeof(mat));
                memset(cap,0,sizeof(cap));
                for(i=1;i<=m;i++){
                        scanf("%d %d %d",&k,&l,&j);
                        mat[k][l]=mat[l][k]=j;
                        cap[k][l]=cap[l][k]=1;
                }
                mat[2][n+1]=inf;
                mat[n+1][2]=inf;
```

```
                ford(1,n+1);
                printf("\n");
        }
        return 0;
}
int bfs(int s,int t){
        memset(pre,-1,sizeof(pre));
        memset(col,0,sizeof(col));
        int i,j,k,l;
        while(!q.empty()) q.pop();
        q.push(s);
        col[s]=1;
        while(!q.empty()){
                i=q.front(); q.pop();
                if(i==t) break;
                for(j=s;j<=t;j++){
                        if(col[j]==0&&mat[i][j]>0){
                                pre[j]=i;
                                q.push(j);
                                col[j]=1;
                                if(j==t) break;
                        }
                }
        }
        int wh,path,prev;
        path=inf;
        wh=t;
        while(pre[wh]!=-1){
                prev=pre[wh];
                path=mini(path,mat[prev][wh]);
                wh=prev;
        }
        wh=t;
        while(pre[wh]!=-1){
                prev=pre[wh];
                mat[prev][wh]-=path;
                mat[wh][prev]+=path;
                wh=prev;
        }
        if(path==inf )return 0;
        return path;
}
void ford(int s,int t){
        int ret=0,i,j;
        while(1){
                int fl=bfs(s,t);
```

```
                    if(fl) ret+=fl;
                    else break;
            }
//          printf("ret - %d\n",ret);
            int flag[maxm][maxm];
            memset(flag,0,sizeof(flag));
            for(i=1;i<=n;i++){
                    if(!col[i]) continue;
                    for(j=1;j<=n;j++){
                            if(col[j]) continue;
                            if(cap[i][j])printf("%d %d\n",i,j);
                    }
            }
}
```

## Min-cost Flow :

```
/*
Problem :  Loj 1222 Gift Packing .
Algo    :  Min-cost Flow        .
*/
int in(int x){
            if(x%2) return x+1;
            return x-1;
};
struct node{
            int no;
            int cost;
};
struct edge{
            int u,v,cost,cap,next;
};
edge edges[maxe];
struct path{
            int a,b,c;
};
path paths[maxe];
priority_queue<node>pq;
int prev[maxm],pre[maxm],d[maxm],n,m,e,cas=1;
int mat[maxm][maxm];
bool operator<(const node &a,const node &b){
            return a.cost>b.cost;
}
void add(int u,int v,int cost,int cap){
            edges[e].u=u; edges[e].v=v;
            edges[e].cost=cost; edges[e].cap=cap;
```

```
        edges[e].next=prev[u];
        prev[u]=e++;
}
int mini(int a,int b){
        if(a<b) return a;
        return b;
}
bool dij(int s);
void ford(int s,int t);
int main(){
        int i,j,k,l,t=1,test,tot;
        scanf("%d",&test);
        while(test--){
                memset(prev,-1,sizeof(prev));
                scanf("%d",&n);
                tot=1;
                k=1;
                e=0;
                for(i=1;i<=n;i++){
                        for(j=1;j<=n;j++){
                                scanf("%d",&mat[i][j]);
                                paths[k].a=i; paths[k].b=n+j; paths[k].c=mat[i][j];
                                k++;
                        }
                }
                tot=k;
                for(i=1;i<=n;i++){
                        add(0,i,0,1);
                   add(i,0,0,0);

                        add(n+i,2*n+1,0,1);
                add(2*n+1,n+i,0,0);
                }
                for(i=1;i<tot;i++){
                        k=paths[i].a; l=paths[i].b; j=paths[i].c;
                        add(k,l,-j,1);
                        add(l,k,j,0);
                }
                ford(0,2*n+1);
        }
        return 0;
}
bool dij(int s){
        int i,j,k,l;
        node temp,temp1;
        memset(pre,-1,sizeof(pre));
```

```
        for(i=0;i<=2*n+10;i++){
                d[i]=inf;
        }
        d[s]=0;
        temp.cost=0;
        temp.no=s;
        pq.push(temp);
        while(!pq.empty()){
                temp=pq.top(); pq.pop();
                j=temp.no;

                for(i=prev[j];i!=-1;i=edges[i].next){
                        k=edges[i].v;
                        if(edges[i].cap>0 && d[k]>d[j]+edges[i].cost){
                                d[k]=d[j]+edges[i].cost;
                                temp1.cost=d[k]; temp1.no=k;
                                pre[k]=i;
                                pq.push(temp1);
                        }

                }
        }
        return d[2*n+1]!=inf;
}
void ford(int s,int t){
        int i,j,k,l,wh,fl,ret,ans;
        fl=0;ans=0;
        while(dij(s)){
                wh=pre[t];
                ret=inf;
                while(wh!=-1){
                        ret=mini(ret,edges[wh].cap);
                        wh=pre[edges[wh].u];
                }
                wh=pre[t];
                while(wh!=-1){
                        edges[wh].cap-=ret;
                        edges[wh^1].cap+=ret;
                        wh=pre[edges[wh].u];
                }
                fl+=ret;
                ans+=(ret*d[t]);
        }
        printf("Case %d: %d\n",cas++,ans*-1);
}
```

## Hierholzer's algorithm ( Euler Cuircuit Print ):

Hierholzer's 1873 paper provides a different method for finding Euler cycles that is more efficient than Fleury's algorithm:

- Choose any starting vertex *v*, and follow a trail of edges from that vertex until returning to *v*. It is not possible to get stuck at any vertex other than *v*, because the even degree of all vertices ensures that, when the trail enters another vertex *w* there must be an unused edge leaving *w*. The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
- As long as there exists a vertex *v* that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from *v*, following unused edges until returning to *v*, and join the tour formed in this way to the previous tour.

By using a data structure such as a doubly linked list to maintain the set of unused edges incident to each vertex, to maintain the list of vertices on the current tour that have unused edges, and to maintain the tour itself, the individual operations of the algorithm (finding unused edges exiting each vertex, finding a new starting vertex for a tour, and connecting two tours that share a vertex) may be performed in constant time each, so the overall algorithm takes linear time.

## Erdos and Gallai Theorem:

```
// Given the degrees of the vertices of a graph, is it possible to construct such graph Input - the deg[] array
int deg[MM], n, degSum[MM], ind[MM], minVal[MM];
bool ErdosGallai() { // 1 indexed
        bool poss = true;
        int i, sum = 0, j, r;
        for( i = 1; i <= n; i++ ) {
                if( deg[i] >= n ) poss = false;
                sum += deg[i];
        }
        //Summation of degrees has to be ODD and all degrees has to be < n - 1
        if( !poss || ( sum & 1 ) || ( n == 1 && deg[1] > 0 ) ) return false;
        sort( deg + 1, deg + n + 1, greater <int>() );
        degSum[0] = 0;
        j = n;
        for( i = 1; i <= n; i++ ) {
                degSum[i] = degSum[i-1] + deg[i];      //CONSTRUCTING: degSum
                for( ; j >= 1 && deg[j] < i; j-- );  //CONSTRUCTING: ind
                ind[i] = j+1;
```

```
        }
        //CONSTRUCTING : minVal
        for(r = 1; r < n; r++) {
                j = ind[r];
                if( j == n+1 ) minVal[r]=( n - r ) * r;
                else if( j <= r ) minVal[r] = degSum[n] - degSum[r];
                else {
                        minVal[r] = degSum[n] - degSum[j-1];
                        minVal[r] += (j-r-1)*r;
                }
        }
        //Checking : Erdos & Gallai Theorem
        for( r = 1; r < n; r++ ) if( degSum[r] > ( r * (r-1) + minVal[r] ) ) return false;
        return true;
}
```

# Dynamic Programming

## LIS(nlog(n)):

```
int in[maxim],L[maxim],p[maxim];
bool com(int a,int value)
{
        if(value>in[a])  return true;     //for strictly increasing LIS...ex - 1 2 2 ... ans - 2
        //if(value>=in[a])  return true;  //for non-decreasing LIS...ex - 1 2 2 ... ans - 3
        return false;
}
void print_lis(int pos)
{
        if(p[pos])        print_lis(p[pos]);
        printf("%d\n",in[pos]);
}

int main()
{
        int n,i,l,pos;
        bool f;
        while(scanf("%d",&n)==1)
        {
                in[0]=-2147483648;
                l=1;
                L[0]=0;
                n++;
```

```
                rep(i,1,n)  scanf("%d",&in[i]);
                for(i=1;i<n;i++)
                {
                        pos = lower_bound(L,L+l,in[i],com) - L;
                        f = (pos==l);
                        if( f || in[ L[pos] ] > in[i] )
                        {
                                p[i] = L[pos-1];
                                L[pos] = i;
                                if(f)     l++;
                        }
                }
                l--;
                printf("%d\n",l);
                printf("-\n");
                print_lis(L[l]);
        }
        return 0;
}
```

## Convex Hull Trick 1:

```
/*
ID: brian_bi21
PROG: acquire (Usaco Mar 08).
Algo: Convex Hull Trick.
*/
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int pointer; //Keeps track of the best line from previous query
vector<long long> M; //Holds the slopes of the lines in the envelope
vector<long long> B; //Holds the y-intercepts of the lines in the envelope
//Returns true if either line l1 or line l3 is always better than line l2
bool bad(int l1,int l2,int l3)
{
        /*
        intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
        intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
        set the former greater than the latter, and cross-multiply to
        eliminate division
        */
        return (B[l3]-B[l1])*(M[l1]-M[l2])<(B[l2]-B[l1])*(M[l1]-M[l3]);
}
```

```
//Adds a new line (with lowest slope) to the structure
void add(long long m,long long b)
{
	//First, let's add it to the end
	M.push_back(m);
	B.push_back(b);
	//If the penultimate is now made irrelevant between the antepenultimate
	//and the ultimate, remove it. Repeat as many times as necessary
	while (M.size()>=3&&bad(M.size()-3,M.size()-2,M.size()-1))
	{
		M.erase(M.end()-2);
		B.erase(B.end()-2);
	}
}
//Returns the minimum y-coordinate of any intersection between a given vertical
//line and the lower envelope
long long query(long long x)
{
	//If we removed what was the best line for the previous query, then the
	//newly inserted line is now the best for that query
	if (pointer>=M.size())
		pointer=M.size()-1;
	//Any better line must be to the right, since query values are
	//non-decreasing
	while (pointer<M.size()-1&&
	  M[pointer+1]*x+B[pointer+1]<M[pointer]*x+B[pointer])
		pointer++;
	return M[pointer]*x+B[pointer];
}
int main()
{
	int M,N,i;
	pair<int,int> a[50000];
	pair<int,int> rect[50000];
	freopen("acquire.in","r",stdin);
	freopen("acquire.out","w",stdout);
	scanf("%d",&M);
	for (i=0; i<M; i++)
		scanf("%d %d",&a[i].first,&a[i].second);
	//Sort first by height and then by width (arbitrary labels)
	sort(a,a+M);
	for (i=0,N=0; i<M; i++)
	{
		/*
		When we add a higher rectangle, any rectangles that are also
```

```
                    equally thin or thinner become irrelevant, as they are
                    completely contained within the higher one; remove as many
                    as necessary
                    */
                    while (N>0&&rect[N-1].second<=a[i].second)
                            N--;
                    rect[N++]=a[i]; //add the new rectangle
            }
            long long cost;
            add(rect[0].second,0);
            //initially, the best line could be any of the lines in the envelope,
            //that is, any line with index 0 or greater, so set pointer=0
            pointer=0;
            for (i=0; i<N; i++) //discussed in article
            {
                    cost=query(rect[i].first);
                    if (i<N)
                            add(rect[i+1].second,cost);
            }
            printf("%lld\n",cost);
            return 0;
}
```

## Divide and Conquer Optimization :

```
/*
Author : rng_58.
Sufficient Condition : pre[i][j]< pre[i][j+1] < pre[i][j+2].
Pre = Optimal path tracker .
*/

REP(i,N+1) dp[1][i] = get_cost(0, i);
for(i=1;i<K;i++) func(i, 0, N+1, 0, N);

void func(int d, int l, int r, int sepl, int sepr){
   int i;
   if(r-l == 1) return;
   int m = (l + r) / 2;
   int sep = -1;
   dp[d+1][m] = INF;
   for(i=sepl;i<=sepr;i++) if(i <= m){
      int tmp = dp[d][i] + get_cost(i, m);
      if(tmp < dp[d+1][m]){
         dp[d+1][m] = tmp;
         sep = i;
```

```
        }
    }

    func(d, l, m, sepl, sep);
    func(d, m, r, sep, sepr);
}
```

## Knuth Optimization :

Let F[a][b] be the minimum cost to make all cuts from a to b inclusive.
In the standart n^3 solution :

F[a][b] = min(F[a][c-1] + F[c+1][b] + length(a, b)) - for every c from a to b;

Let P[a][b] be the c for which F[a][b] is minimized. It can be shown that:

F[a][b] = min(F[a][c-1] + F[c+1][b] + length(a, b)) - for every c from
P[a][b - 1] to P[a + 1][b];

# GEOMETRY

## Macro :

```
//Macro.....
#define ii int
#define maxm 100100
#define pi acos(-1.0)
#define eps 1e-9
#define sq(a) ((a)*(a))
#define dist(a,b) (sq(a.x-b.x) + sq(a.y-b.y))
#define iseq(a,b) (fabs(a-b)<eps)
#define eq(a,b) iseq(a,b)
#define area_t(x1,y1,x2,y2,x3,y3) ( x1*(y2-y3) + x2*(y3-y1) + x3*(y1-y2) )
#define spDist(lat1,long1,lat2,long2,r) ( r * acos( sin(lat1) * sin(lat2) + cos(lat1) *
cos(lat2) * cos(long1-long2) ) )

// Template.....
template< class T > bool inside(T a, T b, T c) { return a<=b && b<=c; }
ii mini(ii a,ii b){
        if(a<b) return a; return b;
}
```

```
ii maxi(ii a,ii b){
        if(a>b) return a; return b;
}
```

**Structure Declaration :**

```
// Structure....
struct point { // Creates normal 2D point
   double x, y;
   point() {}
   point( double xx, double yy ) { x = xx, y = yy; }
   // Operator overloading.....
   bool operator <(point b)const{
                if(!eq(x,b.x) )       return x < b.x;
                return y < b.y;
        }
        bool operator == (point b) const{
                if(eq(x,b.x) && eq(y,b.y))   return true;
                return false;
        }

};

struct point3D { // Creates normal 3D point
   double x, y, z;
};
struct line { // Creates a line with equation ax + by + c = 0
   double a, b, c;
   line() {}
   line( point p1,point p2 ) {
      a = p1.y - p2.y;
      b = p2.x - p1.x;
      c = p1.x * p2.y - p2.x * p1.y;
   }
};
struct circle { // Creates a circle with point 'center' as center and r as radius
   point center;
   double r;
   circle() {}
   circle( point P, double rr ) { center = P; r = rr; }
};
struct segment { // Creates a segment with two end points -> A, B
   point A, B;
   segment() {}
   segment( point P1, point P2 ) { A = P1, B = P2; }
```

```
    bool operator < (const segment &a)const{
        return A<a.A;
    }
};
struct quad { // quadrilateral  with four points .. counterclock wise...
    point p[5];
    quad(){}
    quad(point a,point b,point c,point d){
        p[0]=a; p[1]=b; p[2]=c; p[3]=d;
    }
};
struct tri{ //  Triangle..... should be clock_wise...
    point p1,p2,p3;
    tri(){}
    tri(point _p1,point _p2,point _p3){
        p1=_p1; p2=_p2; p3=_p3;
    }
};
```

## Function :

### Essential Function

**# cross product = p0p1 * p0p2 :**

```
// cross product = p0p1 * p0p2..
inline double cross( point p0, point p1, point p2 ) {
    return( ( p1.x - p0.x ) * ( p2.y - p0.y ) - ( p2.x - p0.x ) * ( p1.y - p0.y ) );
}
```

**# cross product p1*p2 :**

```
inline double cross(point p1, point p2 ) {
    return( ( p1.x * p2.y ) - ( p2.x *  p1.y ) );
}
```

### Find Distance

**# distance between point to point :**

```
// distance between point to point...
inline double distancepp( point a, point b ) {
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ) );
}
```

**# distance between 3D point to point :**

```
inline double distancepp( point3D a, point3D b ) {
   return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ) + ( a.z - b.z ) * ( a.z - b.z
) );
}
```

**# square distance between point to point :**

```
// square distance between point to point.
inline double sq_distance( point a, point b ) {
   return ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y );
}
```

**# distance between point to line :**

```
// distance between point to line....
inline double distancepl( point P, line L ) {
   return fabs( L.a * P.x + L.b * P.y + L.c ) / sqrt( L.a * L.a + L.b * L.b );
}
```

**#Distance - Point, Segment:**

```
//Distance - Point, Segment:
inline double distanceps( point P, segment S ) {
        line L1 = line(S.A,S.B), L2; point P1;
        L2 = findPerpendicularLine( L1, P );
        if( intersection( L1, L2, P1 ) )
                if( eq ( distancepp( S.A, P1 ) + distancepp( S.B, P1 ), distancepp(
S.A, S.B ) ) )
                        return distancepl(P,L1);
        return mini ( distancepp( S.A, P), distancepp( S.B, P) );
}
```

## Intersection

**Intersection - Line, Line:**

```
inline bool intersection( line L1, line L2, point &p ) {
        double det = L1.a * L2.b - L1.b * L2.a;
        if( eq ( det, 0 ) ) return false;
        p.x = ( L1.b * L2.c - L2.b * L1.c ) / det;
        p.y = ( L1.c * L2.a - L2.c * L1.a ) / det;
        return true;
}
```

**Intersection - Segment, Segment:**

```
inline bool intersection( segment L1, segment L2, point &p ) {
        if( !intersection( line( L1.A, L1.B ), line( L2.A, L2.B ), p) ) {
                return false; // can lie on another, just check their equations, and check
overlap
        }
        return(eq(distancepp(L1.A,p)+distancepp(L1.B,p),distancepp(L1.A,L1.B))
&&
                eq(distancepp(L2.A,p)+distancepp(L2.B,p),distancepp(L2.A,L2.B)));
}
```

## Intersecting point between circle and line :

```
inline bool intersectioncl(circle C,line L,point &p1,point &p2) {
        if( distancepl( C.center, L ) > C.r + eps ) return false;
        double a, b, c, d, x = C.center.x, y = C.center.y;
        d = C.r*C.r - x*x - y*y;
        if( eq( L.a, 0) ) {
                p1.y = p2.y = -L.c / L.b;
                a = 1;
                b = 2 * x;
                c = p1.y * p1.y - 2 * p1.y * y - d;
                d = b * b - 4 * a * c;
                d = sqrt( fabs (d) );
                p1.x = ( b + d ) / ( 2 * a );
                p2.x = ( b - d ) / ( 2 * a );
        }
        else {
                a = L.a *L.a + L.b * L.b;
                b = 2 * ( L.a * L.a * y - L.b * L.c - L.a * L.b * x);
                c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
                d = b * b - 4 * a * c;
                d = sqrt( fabs(d) );
                p1.y = ( b + d ) / ( 2 * a );
                p2.y = ( b - d ) / ( 2 * a );
                p1.x = ( -L.b * p1.y -L.c ) / L.a;
                p2.x = ( -L.b * p2.y -L.c ) / L.a;
        }
        return true;
}
```

**//Intersection Area between Two Circles:**

```
inline double intersectionArea2C( circle C1, circle C2 ) {
        C2.center.x = distancepp( C1.center, C2.center );
        C1.center.x = C1.center.y = C2.center.y = 0;
        if( C1.r < C2.center.x - C2.r + eps ) return 0;
        if( -C1.r + eps > C2.center.x - C2.r ) return pi * C1.r * C1.r;
        if( C1.r + eps > C2.center.x + C2.r ) return pi * C2.r * C2.r;
        double c, CAD, CBD, res;
        c = C2.center.x;
        CAD = 2 * acos( (C1.r * C1.r + c * c - C2.r * C2.r) / (2 * C1.r * c) );
        CBD = 2 * acos( (C2.r * C2.r + c * c - C1.r * C1.r) / (2 * C2.r * c) );
        res=C1.r * C1.r * ( CAD - sin( CAD ) ) + C2.r * C2.r * ( CBD - sin ( CBD ) );
        return .5 * res;
}
```

## conversion

### # radian to degree :

```
double convrd(double theta){
   double ret=180; ret/=pi; return ret*theta;
}
```

### # degree to radian :

```
double convdr(double theta){
   double ret=pi; ret/=(double)180.0; return ret*theta;
}
```

### # convert spherical to cartesian co-ordinate.....

```
// convert spherical to cartesian co-ordinate.....
void sph_to_cartesian(double R,double lat,double lng,point3D &p){

   lat=convdr(lat);
   lng=convdr(lng);

   p.x=R*sin(lat)*cos(lng);
   p.y=R*sin(lat)*sin(lng);
   p.z=R*cos(lat);
}
```

### # convert longitude/latitude to cartesian co-ordinate.....

```
// convert longitude/latitude to cartesian co-ordinate.....
void earth_to_cartesian(double R,double lat,double lng,point3D &p){

   lat=convdr(lat);
   lng=convdr(lng);

   p.x=R*cos(lat)*cos(lng);
   p.y=R*cos(lat)*sin(lng);
   p.z=R*sin(lat);
}
```

**# convert cartesian co-ordinate  to longitude/latitude**

```
lat = asin(z / R)
lon = atan2(y, x)
```

# Inside Function

```
//  check whether a  point inside a Segment ....
bool inside_segment(segment S,point P){
        if( eq ( distancepp( S.A, P ) + distancepp( S.B, P ), distancepp( S.A, S.B ) ) )
return 1;
        return 0;
}
//  check whether a  point inside a triangle ....
bool inside_tri(tri t,point p){

   point p1=t.p1,p2=t.p2,p3=t.p3;

   //  check  for boundary.....
   if(iseq(cross(p,p1,p2),0) && inside_segment(segment(p1,p2),p)) return 1;
   if(iseq(cross(p,p2,p3),0) && inside_segment(segment(p2,p3),p)) return 1;
   if(iseq(cross(p,p1,p3),0) && inside_segment(segment(p1,p3),p)) return 1;
   // .....

   if(cross(p,p1,p2)*cross(p3,p1,p2)<0) return 0;
   if(cross(p,p2,p3)*cross(p1,p2,p3)<0) return 0;
   if(cross(p,p1,p3)*cross(p2,p1,p3)<0) return 0;

   return 1;

}
```

**Point Inside a Convex Polygon(O(lgn)) :**

```
/*C[] array of points of convex polygon in ccw order,  nc number of points in C, p target points.
returns true if p is inside C (including edge) or false otherwise.  complexity O(lg n) */
int triArea2(const point &a, const point &b, const point &c)   {
        return (a.x*(b.y-c.y) + b.x*(c.y-a.y) + c.x*(a.y-b.y));
}
bool inConvexPoly(point *C, int nc, const point &p)  {
        int st = 1, en = nc - 1, mid;
        while(en - st > 1) {
                mid = (st + en)>>1;
                if(triArea2(C[0], C[mid], p) < 0)       en = mid;
                else   st = mid;
        }
        //  for point in border.....
        if(iseq(triArea2(C[0], C[1], p),0.0)) return false;
        if(iseq(triArea2(C[0], C[nc-1], p),0.0)) return false;
        if(iseq(triArea2(C[nc-1], C[nc-2], p),0.0)) return false;
        // finish.....
        if(triArea2(C[0], C[st], p) < 0 ) return false;
        if(triArea2(C[st], C[en], p) < 0 || iseq(triArea2(C[st], C[en], p) , 0.0) ) return
false;  // iseq() for border testing .....
        if(triArea2(C[en], C[0], p) < 0 ) return false;
        return true;
}
```

## AREA

```
//  area of  polygon.....
double areaPoly(point P[],int n){
   double area=0;
   for( int i = 0, j = n - 1; i < n; j = i++ ) area += P[j].x * P[i].y - P[j].y * P[i].x;
   return fabs(area)*.5;
}
```

## Convex Hull :

```
//  convex Hull = graham scan O(nlgn)
bool sort_x(point a,point b){
   if(iseq(a.x,b.x)) return a.y<b.y;
   return a.x<b.x;
}
bool sort_y(point a,point b){
   if(iseq(a.y,b.y)) return a.x<b.x;
   return a.y<b.y;
}
point p[maxm]; // p=points for  convex hull...
bool normal(const point &a, const point &b) {   return (iseq(a.x,b.x) ? a.y < b.y : a.x <
b.x);}
bool issame(const point &a, const point &b) {   return (iseq(a.x,b.x) && iseq(a.y,b.y));}
```

```
void makeUnique(point p[],int &np) {  sort(&p[0], &p[np], normal);    np =
unique(&p[0], &p[np], issame) - p;}
//sort by polar angle>>>(convex_hull)
bool comp(point a,point b){
   double d = cross(p[0], a, b);
   if(d<0) return false;
   if(iseq(d,0) && dist(p[0], b) < dist(p[0], a)) return false;
   return true;
}
void convex_hull(point ans[],point p[],int &n,int &nc){
   makeUnique(p,n);
   int i,pos = 0;
   for(i=1; i<n; i++)
      if(p[i].y<p[pos].y || (p[i].y==p[pos].y && p[i].x<p[pos].x))
         pos = i;
   swap(p[0], p[pos]);
   sort(p+1, p+n, comp);

   ans[0] = p[0];
   if(n>=2)    ans[1] = p[1];
   for(i=nc=2; i<n; i++)
   {
      while(nc>=2 && cross(ans[nc-2], ans[nc-1], p[i])<0||iseq(cross(ans[nc-2], ans[nc-1],
p[i]),0)) nc--;
      ans[nc++] = p[i];
   }
   if(n==1)       nc=1;
   else if(nc==2)
   {
      if(p[0].x == p[1].x && p[0].y == p[1].y)   nc=1;
   }

}
```

# Important Formulas

## Area of a triangle :

Let *K* be the triangle's area and let *a*, *b* and *c*, be the lengths of its sides. By Heron's Formula , the area of the triangle is

K = sqrt( s * (s-a) * (s-b) * (s –c) ).

where **S** is the semiperimeter .

$$s = \tfrac{1}{2}(a+b+c).$$

**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) .

## Radius of a In-cicle:

The radius of the incircle (also known as the **inradius**, *r* ) is

$$r = \frac{2K}{P} = \sqrt{\frac{(s-a)(s-b)(s-c)}{s}}.$$

Thus, the area *K* of a triangle may be found by multiplying the inradius by the semiperimeter:

$$K = rs.$$

**Regular Polygon :**

a regular polygon is a Polygon that is equiangular (all angles are equal in measure) and equilateral (all sides have the same length).

## Angle :

For a regular convex *n*-gon, each interior angle has a measure of:

$$(n-2) \times \frac{180}{n}$$ degrees .

**Apothem:** The **apothem** of a regular polygon is a line segment from the center to the midpoint of one of its sides. Equivalently, it is the line drawn from the center of the polygon that **is perpendicular to one of its sides.**

## Circumradius:

The **circumradius** from the center of a regular polygon to one of the vertices is related to the side **length *s*** or to the **apothem *a*** by

$$r = \frac{s}{2\sin\frac{\pi}{n}} = \frac{a}{\cos\frac{\pi}{n}}.$$

## Area :

The **area *A*** of a convex regular *n*-sided polygon
having **Side *s***, **circumradius *r***, **apothem *a***, and **perimeter *p*** is given by

$$A = \tfrac{1}{2}nsa = \tfrac{1}{2}pa = \tfrac{1}{4}ns^2\cot\tfrac{\pi}{n} = na^2\tan\tfrac{\pi}{n} = \tfrac{1}{2}nr^2\sin\tfrac{2\pi}{n}$$

## Centroid of a 2D polygon:

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

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

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

# Searching

## Ternary Search :

```
double ts(){
        double min=0;
        double max=1;
        int c=100; //for higher precision have to increase
        double k,l,f,g;
        while(c--){

                f=min+(max-min)/(double)3.0;
                g=min+(double)2.0*((max-min)/(double)3.0);
                k=fun(f); l=fun(g);
                if(k<l){
                        max=g;
                }
                else{
                        min=f;
                }
        }

        return (min+max)/2.0 ;
}
// problem dependent . . . .
double fun(double piv){
}
```

# Game Theory

## Nim – Game :

```
/*
Author  : Rashedul Hasan Rijul .
problem : Uva - 10165 ( stone Games) .
```

```
Algo    : Nim .
*/
#define ii long long int
int n;
int main(){
        int i,j,k,l,test,t=1;
        while(scanf("%d",&n)==1){
                if(!n) break;
                ii ans=0;
                for(i=1;i<=n;i++){
                        scanf("%d",&k);
                        ans=ans^k;
                }
                if(ans){
                        printf("Yes\n");
                }
                else {
                        printf("No\n");
                }
        }
        return 0;
}
```

## Misere – Nim Game :

```
/*
Author  : Rashedul Hasan Rijul .
problem : Light OJ - 1253 ( Misere Nim) .
Algo    : Misere-Nim .
*/
#define maxm 1000
#define ii int
int a[maxm];
int main(){
        int i,j,k,l,test,t=1,n;
        scanf("%d",&test);
        while(test--){
                scanf("%d",&n);
                ii ans=0,ans1;
                bool fl=0;
                l=0;
                for(i=1;i<=n;i++){
                        scanf("%d",&k);
                        if(k==1){
                                l++;
                                ans^=1;
```

```
                        }
                        else{
                                ans^=k;
                                fl=1;
                        }
                }
                // Alice play first....
                if(!fl){
                        if(l%2==1) printf("Case %d: Bob\n",t++);
                        else printf("Case %d: Alice\n",t++);
                        continue;
                }
                if(ans) printf("Case %d: Alice\n",t++); // Alice play first....
                else  printf("Case %d: Bob\n",t++);
        }
        return 0;
}
```

## Sprunge – Grundy Number :

*problem : Light oj 1315 ( Game of Hyper Knight) .*

```
int dp[maxm][maxm];
int dx[]={-1,-1,1,-2,-2,-3};
int dy[]={-2,-3,-2,-1,1,-1};
int cal(int i,int j){
        if(dp[i][j]!=-1) return dp[i][j];
        set<int>s;
        int ret=0,i1,k,l,j1,val;
        for(i1=0;i1<6;i1++){
                k=i+dx[i1]; l=j+dy[i1];
                if(k>=0&&l>=0){
                        s.insert(cal(k,l));
                }
        }
        while(s.find(ret)!=s.end()){
                ret++;
        }
        return dp[i][j]=ret;
}
```

## Green Hackenbush :

```
/*
Author  : misof
Problem : ipsc 2003 G [hackenbush] (c) misof
Algo    : Green Hackenbush .
*/
```

```
#define min(x,y) ((x)<(y))?(x):(y)

int Cases,N,M;
vector< list<int> > G,G2;
vector<int> GV;
vector<int> visited,from,time_disc,time_up;
int DFStime;

void DFS_Visit(int v){
  int edges_to_parent=0;
  visited[v]=1; time_disc[v]=time_up[v]=++DFStime;
  for (list<int>::iterator start=G[v].begin();start!=G[v].end();start++) {
    if (!visited[*start]) { from[*start]=v; DFS_Visit(*start);
time_up[v]=min(time_up[v],time_up[*start]); }
    else {
      if ((*start)!=from[v]) { time_up[v]=min(time_up[v],time_disc[*start]); }
      else {
        if (edges_to_parent) { time_up[v]=min(time_up[v],time_disc[*start]); }
        edges_to_parent++;
      }
    }
  }
}

void FindBridges(void){
  time_disc.clear(); time_up.clear(); visited.clear(); from.clear();
  visited.resize(N+3,0); time_disc.resize(N+3,0); time_up.resize(N+3,0);
from.resize(N+3,0);
  from[1]=1; DFStime=0;
  DFS_Visit(1);
}

int IsBridge(int v_lo, int v_high) {
  if (v_high!=from[v_lo]) return 0;
  return ( time_disc[v_lo]==time_up[v_lo] );
}

void ContractGraph(void){
  vector<int> color(N+3,0);
  int colors=1;
  color[1]=1;

  list<int> Q;
  Q.clear(); Q.push_back(1);
  while (!Q.empty()) {
    int where=Q.front(); Q.pop_front();
```

```
    for (list<int>::iterator it=G[where].begin(); it!=G[where].end(); it++) if (!color[*it]) {
      if (IsBridge(*it,where)) color[*it]=++colors; else color[*it]=color[where];
      visited[*it]=1; Q.push_back(*it);
    }
  }

  G2.clear(); G2.resize(N+3);
  for (int i=1;i<=N;i++)
    for (list<int>::iterator it=G[i].begin(); it!=G[i].end(); it++)
      G2[color[i]].push_back(color[*it]);
}

int GrundyValue(int v){
  int loops=0,gv=0;

  if (GV[v]!=-1) return GV[v]; GV[v]=1000000000;

  for (list<int>::iterator start=G2[v].begin(); start!=G2[v].end(); start++) {
    if ((*start)==v) loops++; else if (GV[*start]!=1000000000)
gv^=(1+GrundyValue(*start));
  }
  loops/=2; if (loops%2) gv^=1;
  return GV[v]=gv;
}

int main(void){
  int v1,v2;

  //freopen("g1.in","r",stdin);
  //freopen("out.txt","w",stdout);

  cin >> Cases;
  while (Cases--) {
    // read graph dimensions
    cin >> N >> M;
    // read the graph
    G.clear(); G.resize(N+3);
    for (int i=0;i<M;i++) { cin >> v1 >> v2; G[v1].push_back(v2); G[v2].push_back(v1);
}
    // collapse all circuits in the graph
    FindBridges();
    ContractGraph();
    // compute the SG value
    GV.clear(); for (int i=0;i<=N;i++) GV.push_back(-1);
    int result=GrundyValue(1);
    if (result) cout << "Alice\n"; else cout << "Bob\n";    //cout << result << "\n";
```

```
  }
  return 0;
}
```

## Red_blue Hacken Bush (stalk Only):

```
/*
problem: codechef (chef game) .
Algo : red-blue hackenbush.
*/
#define MAXN 55
typedef long long int64;

/*
   Problem can be reduced to red-black hackenbush
   http://en.wikipedia.org/wiki/Hackenbush
   Each pile represent a hackenbush stalk
   Game value cooresponding to hackenbush stalk is easy to find.
   Please refer here : http://www.geometer.org/mathcircles/hackenbush.pdf.
   For hackebush games value of two disjoint game is equal to sum of individual game value.
   (http://www-math.mit.edu/~rstan/transparencies/games.pdf)

*/

int t,n,tcase;
int arr[MAXN];

int64 calculate(){
  int64 res = 0; int64 value = 1LL<<48;
  res = (arr[0]%2==0)?value:-value;
  bool is_changed = false;
  for(int i=1; i<n; ++i){
    assert(arr[i]!=arr[i-1]);
    if(arr[i]%2 != arr[i-1]%2){
      is_changed = true;
    }
    if(is_changed) value /= 2;
    res += (arr[i]%2==0)?value:-value;
  }
  return res;
}

int main(){
 for(scanf("%d",&tcase); tcase; tcase-=1){
   scanf("%d",&t);
```

```
    int64 res = 0;
    for(int i=0; i<t; ++i){
      scanf("%d",&n);
      for(int j=0; j<n; ++j) scanf("%d",&arr[j]);
      sort(arr,arr+n);
      res += calculate();
    }
    if(res > 0 ) printf("FIRST\n");
    else if(res < 0 ) printf("SECOND\n");
    else printf("DON'T PLAY\n");
  }
  return 0;
}
```

# Matrix

## Gaussian Elimination :

*Problem : LOJ 1151 - Snakes and Ladders*

```
#define maxm 110

int n,m;
int pos[maxm];

///Gaussian  Elimination.....................................................
#define eps (1e-9)
#define iseq(a,b) (fabs(a-b)<eps)

//  a1x1+a2x2+.....=b1 .....
double a[maxm][maxm],b[maxm],x[maxm];

void gauss(int r,int c){

    int i,j,k,l;
    double val;

    i=j=0;

    while(i<r&&j<c){

        for(k=i;k<r;k++){
            if(iseq(a[k][j],0.0)) continue;
```

```
            for(l=j;l<c;l++){
                swap(a[i][l],a[k][l]);
            }
            swap(b[i],b[k]);
            break;
        }
        if(k==r){
            j++; continue;
        }
        // Making jth col of every row  from (i+1)th to rth row  into  zero.....
        for(k=i+1;k<r;k++){
            val=a[k][j]/a[i][j];
            for(l=j;l<c;l++){
                a[k][l]-=(a[i][l]*val);
            }
            b[k]-=(b[i]*val);

        }

        i++; j++;
    }

    /// Additional  information.....

    /*
    rep(k,i,r)
    {
    rep(j,0,c)      if(!(fabs(a[k][j])<eps))   goto stop ;

     if(!(fabs(b[k])<eps))            return -1 ;      // no solution

       stop : ;
       }

        if(i>c)          return -1 ; // no solution
        if(i==c)         return 0 ; // unique solution
        if(i<c)          return 1 ; // multiple solution
    */

    /// .................

    for(i=c-1;i>=0;i--){
        x[i]=b[i];
        for(k=i+1;k<c;k++){
            x[i]-=(a[i][k]*x[k]);
        }
```

```
        if(!iseq(a[i][i],0.0)) x[i]/=a[i][i];
    }
}
///  Gaussian  Elimination  Finish.....................

int main(){

    int i,j,k,l,test,t=1;
    scanf("%d",&test);
    while(test--){

        for(i=0;i<=100;i++){
            pos[i]=i;
        }

        scanf("%d",&m);

        for(i=1;i<=m;i++){
            scanf("%d %d",&k,&l);
            k--; l--;
            pos[k]=l;
        }


        n=100;
        memset(a,0.0,sizeof(a));

        for(i=0;i<n;i++){

            a[i][i]=1.0;

            if(pos[i]!=i){
                a[i][pos[i]]=-1.0;
                b[i]=0.0;
                continue;
            }

            if(i==n-1){  b[i]=0;  continue; }
            else b[i]=1;

            // prob= probabilty.....
            double prob=(double) 1.0/ (double) 6.0;

            for(j=1;j<=6;j++){
                k=(i+j);
                if(k>99) k=i;
```

```
            k=pos[k];
            a[i][k]-=(prob);
          }

      }

      gauss(n,n);

      printf("Case %d: %.8lf\n",t++,x[0]);

   }

   return 0;
}
```

## Matrix Exponentiation :

*Problem : Uva 12470 (Tribonacci).*

```
#define maxm 10
#define ii long long int

ii n,mod;
ii base[3][3]={{1,1,1},{1,0,0},{0,1,0}};
ii unit[3][3]={{1,0,0},{0,1,0},{0,0,1}},res[3][3];

void cal(ii a[3][3],ii b[3][3]){

            ii ret[3][3]; int i,j,k;
            memset(ret,0,sizeof(ret));

            for(i=0;i<3;i++){
                        for(j=0;j<3;j++){
                                    for(k=0;k<3;k++){
                                                ret[i][j]+=(a[i][k]*b[k][j]);
                                                ret[i][j]%=mod;
                                    }
                        }
            }

            memcpy(a,ret,sizeof(ret));

}
```

```
void exp(ii r[3][3],ii n){

        ii b[3][3];
        memcpy(r,unit,sizeof(unit));
        memcpy(b,base,sizeof(base));

        while(n>0){
                if(n%2==1) cal(r,b);
                n/=2;
                cal(b,b);
        }
}

int main(){

        mod=1000000009;
                if(!n) break;
                if(n==1){
                        printf("0\n"); continue;
                }
                if(n==2){
                        printf("1\n"); continue;
                }
                if(n==3){
                        printf("2\n"); continue;
                }

                exp(res,n-3);
                ii ans=0;
                ans+=(res[0][0]*2+res[0][1]*1);
                ans%=mod;

                printf("%lld\n",ans);
```

## Memorization Technique in Matrix Expo :

```
memcpy(mem[0],base,sizeof(base));

for(i=1;i<=62;i++){
   cal(mem[i-1],mem[i-1],mem[i]);
}

void exp(ii r[4][4],ii n){

        ii b[4][4];
```

```
        memcpy(r,unit,sizeof(unit));
        memcpy(b,base,sizeof(base));

        int j=0;
        while(n>0){
                if(n%2==1) cal(r,mem[j],r);
                n/=2; j++;
                //cal(b,b);
        }
}
```

# Number Theory

## Prime Generation (Sieve ) + Factoriation :

```
#define maxm 10000600
#define ii long long int

bool p[maxm];
int prie[664610],c,tot,totn;

void fact(ii n);
void take(int n);
void gen(int n);
int main(){
        int i,j,k,l,test,t=1;
        gen(maxm-90);
        take(maxm-90);
        return 0;
}
// Factoriation . . . .
void fact(ii n){
        int i,j,k,l;
        node temp;
        ii sq;
        double nd=n;
        sq=sqrt(nd);
        v.clear();
        for(i=0;prime[i]<=sq;i++){
                if(n%prime[i]) continue;
                k=0;
                while(n%prime[i]==0){
                        n/=prime[i];
                        k++;
```

```
                }
                sq=sqrt(n);
                temp.count=k; temp.num=prime[i];
                v.push_back(temp);
                if(n==1) break;
        }

        if(n>1){
                temp.count=1; temp.num=n;
                v.push_back(temp);
        }
}
void take(int n){
        prime[c++]=2;
        for(int i=3;i<=n;i++){
                if(!p[i]) prime[c++]=i;
        }
        tot=c;
}
void gen(int n){
        int i,j,k,l,sq;
        p[0]=p[1]=1;
        sq=sqrt(n);
        for(i=4;i<=n;i+=2) p[i]=1;
        for(i=3;i<=sq;i+=2){
                if(p[i]) continue;
                for(j=i*i;j<=n;j+=(2*i)){
                        p[j]=1;
                }
        }}
```

## Segmented Seive :

```
/*
Author  : Rashedul Hasan Rijul
Problem : LOJ 1197 Help Hanzo
Algo    : Segmented Seive
*/
#define maxm 10111100
#define ii long long int

bool p[1000000+100],segment[maxm];
void gen(int n){
   int i,j,k,l,sq;
   sq=sqrt(n);
   p[0]=1; p[1]=1;
   for(i=4;i<=n;i+=2) p[i]=1;
```

```
    for(i=3;i<=sq;i+=2){
        if(p[i]) continue;
        for(j=i*i;j<=n;j+=(2*i)){
            p[j]=1;
        }
    }
}
int maxi(int a,int b){
    if(a>b) return a;
    return b;
}
int f(int l,int i){
    if(l%i==0) return maxi(l,i*i);
    return maxi(l+(i-(l%i)),i*i);
}

int main(){

    int i,j,k,l,test,t=1,h;

    freopen("in.txt","r",stdin);
    gen(1000000);
    scanf("%d",&test);
    while(test--){
        scanf("%d %d",&l,&h);
        memset(segment,0,sizeof(segment));
        if(l==1) segment[0]=1;
        int sq=sqrt(h);
        for(i=2;i<=sq;i++){
            if(p[i]) continue;
            for(j=f(l,i);j>=l&&j<=h;j+=i){
                segment[j-l]=1;
            }
        }

        int ans=0;
        ii i1;
        for(i1=l;i1<=h;i1++){
            if(!segment[i1-l]) ans++;
        }
        printf("Case %d: %d\n",t++,ans);
    }
}
```

## Bitwise Seive :

```
#define maxm 100000000
int p[(maxm/32)+10],tot,prime[(maxm/10)+1000];
```

```
int on(int n,int k){
   return (n|(1<<k));
}
bool chck(int n,int k){
   return (bool)(n&(1<<k));
}
void gen(int n){

   int i,j,k,l,sq;

   sq=sqrt(n);

   for(i=3;i<=sq;i+=2){
      if(chck(p[i>>5],i&31)) continue;
      for(j=(i*i);j<=n;j+=(i<<1)){
         p[j>>5]=on(p[j>>5],j&31);
      }
   }

   // takine prime into array>>>>>>>>>
   prime[tot++]=2;
   printf("%d\n",2);
   for(i=3;i<=n;i+=2){
      if(!chck(p[i>>5],i&31)){
                                 prime[tot++]=i;
                                 //if((tot-1)%100==0) printf("%d\n",i);
                        }
   }
   printf("%d\n",tot);
}
```

## Euler Phi :

```
#define s 50100
double phi[s];
bool prime[s];
void geneuler(int n){
            double temp;
            phi[0]=0;
            phi[2]=1;
            int sq=sqrt(n);
            int i,j;
            for(i=4;i<=n;i+=2){
                        prime[i]=1;
                        temp=i;
```

```
                phi[i]=temp*.5;
        }
        for(i=3;i<=n;i+=2) phi[i]=i;
        for(i=3;i<=n;i+=2){
                if(prime[i]==0){
                        phi[i]=i-1;
                        for(j=2*i;j<=n;j+=i){
                                prime[j]=1;
                                temp=i;
                                phi[j]*=((temp-1)/temp);
                        }
                }
        }

}
```

## Primality Test:

```
/* this function calculates (a*b)%c taking into account that a*b might overflow */
ii mulmod(ii a,ii b,ii c){
   ii x = 0,y=a%c;
   while(b > 0){
      if(b%2 == 1){
         x = (x+y)%c;
      }
      y = (y*2)%c;
      b /= 2;
   }
   return x%c;}

/* Miller-Rabin primality test, iteration signifies the accuracy of the test */
bool Miller(long long p,int iteration){
   if(p<2){
      return false;
   }
   if(p!=2 && p%2==0){
      return false;
   }
   long long s=p-1;
   while(s%2==0){
      s/=2;
```

```
    }
    for(int i=0;i<iteration;i++){
        long long a=rand()%(p-1)+1,temp=s;
        long long mod=big_mod(a,temp,p);
        while(temp!=p-1 && mod!=1 && mod!=p-1){
            mod=mulmod(mod,mod,p);
            temp *= 2;
        }
        if(mod!=p-1 && temp%2==0){
            return false;
        }
    }
    return true;
}
```

## Big Mod :

```
#define ii __int64
ii  big_mod(int b,int p,int m){
    if(p==0) return 1;
    if(p==1) return b%m;
    ii ret;
    ret=big_mod(b,p/2,m);
    ret*=ret; ret%=m;
    if(p%2) ret*=b;
    ret%=m;
    return ret;
}
```

## Modular Inverse :

```
// Extended Euclid .....  for  finding  Modular inverse
struct node{
            ii x,y,g;
            node(){};
            node(ii xx,ii yy,ii gg){ x=xx; y=yy; g=gg;};
};
//  ax+by=g where g=gcd(a,b)....
node euclid(ii a,ii b);
///////////////*******\\\\\\\\\\\\\\\\\\
node euclid(ii a,ii b){
            if(!b) return node(1,0,a);
            node r=euclid(b,a%b);
            return node(r.y,r.x-(a/b)*r.y,r.g);
}
ii mod_inv(ii n,ii m){
    node t=euclid(n,m);
    if(t.g>1) return 0;
```

```
    ii ret=t.x%m;
    if(ret<0) ret+=m;
    return ret;
}
```

## Extended Euclid :

```
/*
Problem : LOJ 1306 (Solutions to an Equation ).
Algo    : Extended Euclid ( Number of solution of a Linear Diaphontine equation in a given
range ).
*/

// Extended Euclid .....
struct node{
        ii x,y,g;
        node(){};
        node(ii xx,ii yy,ii gg){ x=xx; y=yy; g=gg;};
};
//  ax+by=g where g=gcd(a,b)....
node euclid(ii a,ii b){
        if(!b) return node(1,0,a);
        node r=euclid(b,a%b);
        return node(r.y,r.x-(a/b)*r.y,r.g);
}
//...........//
ii A,B,C,xl,xh,yl,yh;
ii find_lo(ii x0,ii y0,ii ag,ii bg);
int valid_lo(ii x0,ii t,ii bg,ii lo,ii hi);
ii find_hi(ii x0,ii y0,ii ag,ii bg);
int valid_hi(ii x0,ii t,ii bg,ii lo,ii hi);
ii common(ii a,ii b,ii c,ii d){
    if(b<c || d<a) return 0;
    if(a>=c && b<=d) return (b-a+1);
    if(c>=a && d<=b) return (d-c+1);
    if(b>=c && a<=c) return (b-c+1);
    if(a<=d && b>d)  return (d-a+1);
    return 0;
}
ii find_ans(){
    node piv=euclid(A,B);
    if(!piv.g){
        if(C) return 0;
        return (xh-xl+1)*(yh-yl+1);
    }
```

```
    if(C%piv.g) return 0;

    ii x0=piv.x,y0=piv.y;
    x0*=(C/piv.g); y0*=(C/piv.g);

    ii ag=A/piv.g,bg=B/piv.g;

    // x= x0 - t*bg , y= y0 + t*ag;
    ii lo1=find_lo(x0,bg,xl,xh);
    ii lo2=find_lo(y0,-ag,yl,yh);

    ii hi1=find_hi(x0,bg,xl,xh);
    ii hi2=find_hi(y0,-ag,yl,yh);

    return common(lo1,hi1,lo2,hi2);
}

      scanf("%lld %lld %lld %lld %lld %lld %lld",&A,&B,&C,&xl,&xh,&yl,&yh);
       C=-C;
       printf("Case %d: %lld\n",t++,find_ans());

ii find_lo(ii x0,ii bg,ii lox,ii hix){
  // x= x0 - t*bg , y= y0 + t*ag;
  ii lo=-inf,hi=inf;
  ii mid;
  while(lo<hi){
     mid=lo+hi; mid/=2;
     if(valid_lo(x0,mid,bg,lox,hix)){
        if(hi==mid){
           if(valid_lo(x0,mid-1,bg,lox,hix)) return mid-1;
           return mid;
        }
        hi=mid;
     }
     else{
        lo=mid+1;
     }
  }
  return hi;
}
ii find_hi(ii x0,ii bg,ii lox,ii hix){
  // x= x0 - t*bg , y= y0 + t*ag;
  ii lo=-inf,hi=inf;
  ii mid;
  while(lo<hi){
```

```
      mid=lo+hi; mid/=2;
      if(valid_hi(x0,mid,bg,lox,hix)){
         if(lo==mid){
            if(valid_hi(x0,mid+1,bg,lox,hix)) return mid+1;
            return mid;
         }
         lo=mid;
      }
      else{
         hi=mid-1;
      }

   }
   return lo;
}

int valid_lo(ii x0,ii t,ii bg,ii lo,ii hi){
   // check increasing .....
   if(bg<0){
      if(x0-(t*bg)<lo) return 0;
      return 1;
   }
   else{
      if(x0-(t*bg)>hi) return 0;
      return 1;
   }
}
int valid_hi(ii x0,ii t,ii bg,ii lo,ii hi){
   // check increasing .....
   if(bg<0){
      if(x0-(t*bg)>hi) return 0;
      return 1;
   }
   else{
      if(x0-(t*bg)<lo) return 0;
      return 1;
   }
}
```

**DigitCount of N!:**

```
digitCountEfficient( N ) {
        double logVal = 0;
        for( i = 0; prime[ i ] <= N; i++ ){
                logVal += pPowers[ i ] * log10( prime[ i ] );
        }
        return ( int )logVal + 1;
```

```
}
```

**Most Significant digit of N!**

```
mostSignificantDigit( N ){
        double logVal = 0;
        for( i = 0; prime[ i ] <= N; i++ ){
                logVal += pPowers[ i ] * log10( prime[ i ] );
        }
        fractionPart = logVal - ( int )logVal; //get the fractional part

        return ( int )pow( 10, fractionPart);
}
```

**First k digits of N!, k<=12**

```
firstKDigits( N , K ){
        long double logVal = 0; //long double precision
        for( i = 0; prime[ i ] <= N; i++ ){
                logVal += pPowers[ i ] * log10( prime[ i ] ); //after factoring
        }
        fractionPart = logVal - ( int )logVal + K – 1 ; //get the fractional part

printf( "%I64d\n" ,(__int64) powl ( 10.0, fractionPart ));
}
```

**Number of rightMost zeros in N!**

```
numberOfRightMostZeros( N ){
        return min( pPowers[0] , pPowers[2] ); //in array, prime[0]=2, prime[2]=5
}
```

**Last non-zero digit of N!**

```
lastNonZeroDigit( N ){
        __int64 prod = 1;
        for(int i = 1 ; i <= N; i++ ){
                __int64 f = i;
                while( f % 5 == 0 ){
                        f /= 5;
                        prod /= 2;
                }
                prod = (prod % 100000)* f;
        }
        return ( int )( prod % 10);
}
```

# Data Structure

## Segment Tree :

```
/*
Problem : LOJ 1183 ( Computing Fast Average) .
```

```
*/
struct tree{
	int sum,fl;
};
tree m[4*maxm];
int n,q;

int gcd(int a,int b){
	if(a%b==0) return b;
	return gcd(b,a%b);
}

int query(int node,int b,int e,int x,int y){

	if(b==e){
		return m[node].sum;
	}

	int k=e-b+1,l;
	if(b==x&&e==y){
		return m[node].sum;
	}

	int left,right,mid,ret=0;

	left=node<<1; right=left+1; mid=b+e; mid/=2;

	if(m[node].fl==1){
		l=m[node].sum/k; m[node].fl=0;
		update(left,b,mid,b,mid,l);
		update(right,mid+1,e,mid+1,e,l);
	}

	if(y<=mid){
		return query(left,b,mid,x,y);
	}
	else if(x>mid){
		return query(right,mid+1,e,x,y);
	}
	else{
		ret+=query(left,b,mid,x,mid);
		ret+=query(right,mid+1,e,mid+1,y);
	}

	return ret;
}
```

```
void update(int node,int b,int e,int x,int y,int v){

        if(b==e){
                m[node].fl=0;
                m[node].sum=v;
                return ;
        }

        int k=e-b+1,l;
        if(b==x&&e==y){
                m[node].sum=k*v; m[node].fl=1;
                return ;
        }

        int left,right,mid;
        left=node<<1; right=left+1; mid=b+e; mid/=2;

        if(m[node].fl==1){
                l=m[node].sum/k; m[node].fl=0;
                update(left,b,mid,b,mid,l);
                update(right,mid+1,e,mid+1,e,l);
        }

        if(y<=mid){
                update(left,b,mid,x,y,v);
        }
        else if(x>mid){
                update(right,mid+1,e,x,y,v);
        }
        else{
                update(left,b,mid,x,mid,v);
                update(right,mid+1,e,mid+1,y,v);
        }
        m[node].sum=m[left].sum+m[right].sum;
}

void init(int node,int b,int e){

        if(b==e){
                m[node].fl=m[node].sum=0;
                return ;
        }

        int left,right,mid;
```

```
        left=node<<1; right=left+1; mid=b+e; mid/=2;

        init(left,b,mid);
        init(right,mid+1,e);

        m[node].fl=m[node].sum=0;
}
```

## LCA ( Lowest Common Ancestor ) :

```
/*
Algo   : LCA O(sqrt) per query..
Problem: Min-Max Roads (light oj )
*/

#define maxm 100010
#define inf (1<<28)

struct node{
   int min1,max1;
   node(){}
   node(int a,int b){ min1=a; max1=b;}
};

// n=no of node, nr = sqrt of max heigt.....
int n,nr,T[maxm],L[maxm],costT[maxm],P[maxm],costP1[maxm],costP2[maxm];
// v for storing graph , w =weight of the edge ......
vector<int>v[maxm],w[maxm];

int mini(int a,int b){
   if(a<b) return a; return b;
}
int maxi(int a,int b){
   if(a>b) return a; return b;
}
// for calculate P .........
void dfs(int node,int val1,int val2);
// for calculate L and T.
void dfs1(int s,int lev,int pre);
// finding LCA .....
node lca(int x,int y);

int main(){

   int i,j,k,l,test,t=1;
```

```
  //freopen("in.txt","r",stdin);

   scanf("%d",&test);

   while(test--){

      scanf("%d",&n);

      for(i=0;i<=n;i++){
         v[i].clear();w[i].clear();
      }

      for(i=1;i<n;i++){
         scanf("%d %d %d",&k,&l,&j);
         v[k].push_back(l); w[k].push_back(j);
         v[l].push_back(k); w[l].push_back(j);
      }

      nr=0;
      dfs1(1,0,1);
      // fix height sqrt
      k=sqrt(nr); if(k*k!=nr) k++; nr=k;

      dfs(1,0,0);

      int q;
      scanf("%d",&q);
      printf("Case %d:\n",t++);
      for(i=1;i<=q;i++){
         scanf("%d %d",&k,&l);
         node ans=lca(k,l);
         printf("%d %d\n",ans.min1,ans.max1);
      }


   }

   return 0;
}
node lca(int x,int y){

   node ret=node(inf,-inf);

   while(P[x]!=P[y]){
      if(L[x]>L[y]){
```

```
            ret.min1=mini(ret.min1,costP1[x]);
            ret.max1=maxi(ret.max1,costP2[x]);
            x=P[x];
        }
        else{
            ret.min1=mini(ret.min1,costP1[y]);
            ret.max1=maxi(ret.max1,costP2[y]);
            y=P[y];
        }
    }
    while(x!=y){
        if(L[x]>L[y]){
            ret.min1=mini(ret.min1,costT[x]);
            ret.max1=maxi(ret.max1,costT[x]);
            x=T[x];
        }
        else{
            ret.min1=mini(ret.min1,costT[y]);
            ret.max1=maxi(ret.max1,costT[y]);
            y=T[y];
        }
    }

    // Lca node = x
            return ret;

}

//val1= min cost of edge;
//val2= max cost of edge;
void dfs(int node,int val1,int val2){

    int i,j,k,l;

    if(L[node]<nr) P[node]=1;
    else{
        if(!(L[node]%nr)){
                                    val1=val2=costT[node];
            P[node]=T[node];
        }
        else{
            P[node]=P[T[node]];
        }
    }
    costP1[node]=val1;
    costP2[node]=val2;
```

```
    for(i=0;i<v[node].size();i++){
        k=v[node][i];
        if(L[k]<=L[node]) continue;
        dfs(k,mini(val1,w[node][i]),maxi(val2,w[node][i]));
    }
}

void dfs1(int s,int lev,int pre){

    int i,j,k,l;
    T[s]=pre;
    L[s]=lev;
    nr=maxi(nr,lev);

    for(i=0;i<v[s].size();i++){
        k=v[s][i]; if(k==pre) continue;
        costT[k]=w[s][i];
        dfs1(k,lev+1,s);
    }

}
```

## BIT :

```
/*
Author : Rashedul Hasan Rijul
Algo   : BIT
*/

#define ii long long int
#define mod 1000000009
#define maxval 262150

struct BIT{

    ii tree[maxval+100];
    ii n;

    void init(int n1){
        n=n1;
        memset(tree,0,sizeof(tree));
    }
    void clear(){
        memset(tree,0,sizeof(tree));
```

```
    }

    ii read(int idx){
        ii sum=0;
        while(idx>0){
            sum+=tree[idx];
            idx-=(idx& -idx);
            sum%=mod;
        }
        return sum;
    }
    void update(int idx ,int val){

        while (idx <= n){
            tree[idx] += val;
            tree[idx]%=mod;
            idx += (idx & -idx);
        }
    }
    ii read(int beg,int end){
        ii ret=read(end)-read(beg-1);
        if(ret<0) ret+=mod;
        return ret;
    }
};


// BIT  finish .....

BIT bit;
```

## 2-D Bit :

```
/*
Author  : Rashedul Hasan Rijul (silent_coder).
Problem : LOJ 1266 ( points in rectangle ).
Algo    : 2-D bit
*/

#define maxm 1010
#define max_v 1005
int tree[maxm][maxm];
bool fl[maxm][maxm];

void updatey(int x,int y,int val){
```

```
		while(y<=max_v){
			tree[x][y]+=val;
			y+=(y & -y);
		}
}
void updatex(int x,int y,int val){
		while(x<=max_v){
			updatey(x,y,val);
			x+=(x & -x);
		}
}
int ready(int x,int y){
		int ret=0;
		while(y>0){
			ret+=tree[x][y];
			y-=(y & -y);
		}
		return ret;
}
int readx(int x,int y){
		int ret=0;
		while(x>0){
			ret+=ready(x,y);
			x-=(x & -x);
		}
		return ret;
}

int main(){

		int i,j,k,l,k1,l1,test,t=1,q,ans;

		//freopen("in.txt","r",stdin);

		scanf("%d",&test);

		while(test--){

			memset(tree,0,sizeof(tree));
			memset(fl,0,sizeof(fl));
			scanf("%d",&q);
			printf("Case %d:\n",t++);
			for(i=1;i<=q;i++){
				scanf("%d",&j);
				if(j==0){
					scanf("%d %d",&k,&l);
```

```
                        k++; l++;
                        if(fl[k][l]) continue;
                        fl[k][l]=1;
                        updatex(k,l,1);
                }
                else{
                        scanf("%d %d %d %d",&k,&l,&k1,&l1);
                        k++; l++; k1++; l1++;
                        ans=readx(k1,l1);
                        ans-=readx(k1,l);
                        ans-=readx(k,l1);
                        ans+=readx(k,l);
                        for(j=k;j<=k1;j++){
                                if(fl[j][l]) ans++;
                        }
                        for(j=l;j<=l1;j++){
                                if(fl[k][j]) ans++;
                        }
                        if(fl[k][l]) ans--;
                        //ans-=readx(k,l-1);
                        //ans+=readx(k-1,l-1);
                        printf("%d\n",ans);
                }
        }

        }

        return 0;
}
```

## Heavy-Light Decomposition :

```
/*
Problem : Spoj - Query on a tree
*/
#define maxm 200100
#define lg_maxm 20

struct tree{
    int mx_cost;
};
tree seg_T[4*maxm];

int n,m;
vector<int>G[maxm],W[maxm],edge_ind[maxm];
int L[maxm],T[maxm],P[maxm][lg_maxm],subtree_size[maxm];
int edge[maxm],ptr;
```

```
int chain_head[maxm],chain_ind[maxm],chain_no;
int pos_in_base[maxm],base_arr[maxm];

// fixing parent,size and level
void dfs(int s,int pre,int lev){

    T[s]=pre;
    L[s]=lev;
    subtree_size[s]=1;

    for(int i=0;i<G[s].size();i++){
        if(G[s][i]==pre) continue;
        edge[edge_ind[s][i]]=G[s][i];
        dfs(G[s][i],s,lev+1);
        subtree_size[s]+=subtree_size[G[s][i]];
    }

}

// Updating sparse table for lca . . .
void init_sparse(){

    int i,j;
    for(i=0;i<=n;i++){
        for(j=0;(1<<j)<n;j++){
            P[i][j]=-1;
        }
    }

    // the first ancestor ..
    for(i=1;i<=n;i++){
        P[i][0]=T[i];
    }

    // sparse table ..
    for(j=1;(1<<j)<n;j++){
        for(i=1;i<=n;i++){
            if(P[i][j-1]!=-1){
                P[i][j]=P[P[i][j-1]][j-1];
            }
        }
    }
}

/*
* Actual HL-Decomposition part
* Initially all entries of chainHead[] are set to -1.
* So when ever a new chain is started, chain head is correctly assigned.
* As we add a new node to chain, we will note its position in the baseArray.
* In the first for loop we find the child node which has maximum sub-tree
size.
* The following if condition is failed for leaf nodes.
* When the if condition passes, we expand the chain to special child.
* In the second for loop we recursively call the function on all normal
nodes.
* chainNo++ ensures that we are creating a new chain for each normal child.
*/
void heavy_light(int s,int pre,int curr_cost){
```

```
    if(chain_head[chain_no]==-1){
        chain_head[chain_no]=s; // Assign chain head
    }

    chain_ind[s]=chain_no;
    pos_in_base[s]=++ptr; // Position of this node in baseArray which we will
use in Segtree
    base_arr[ptr]=curr_cost;

    int heavy_child=-1,heavy_cost=0,heavy_size=0,i;

    // Loop to find heavy child
    for(i=0;i<G[s].size();i++){
        if(G[s][i]==pre) continue;
        if(subtree_size[G[s][i]]>heavy_size){
            heavy_size=subtree_size[G[s][i]];
            heavy_child=G[s][i];
            heavy_cost=W[s][i];
        }
    }

    if(heavy_child!=-1){
        // Expand the chain
        heavy_light(heavy_child,s,heavy_cost);
    }

    for(i=0;i<G[s].size();i++){
        if(G[s][i]==pre || G[s][i]==heavy_child) continue;
        // light node . . . .
        chain_no++;
        heavy_light(G[s][i],s,W[s][i]);
    }

}

void init_segtree(int node,int b,int e){

    if(b>e) return ;
    if(b==e){
        seg_T[node].mx_cost=base_arr[b];
        return ;
    }

    int left=node<<1,right=left+1,mid=b+e;
    mid/=2;

    init_segtree(left,b,mid);
    init_segtree(right,mid+1,e);

    seg_T[node].mx_cost=maxi(seg_T[left].mx_cost,seg_T[right].mx_cost);
}

int seg_query(int node,int b,int e,int k,int l){

    if(b>e) return 0;

    if(b==k && e==l) return seg_T[node].mx_cost;
```

```
    int left=node<<1,right=left+1,mid=b+e;
    mid/=2;

    if(l<=mid) return seg_query(left,b,mid,k,l);
    else if(k>mid) return seg_query(right,mid+1,e,k,l);
    else{
        return
maxi(seg_query(left,b,mid,k,mid),seg_query(right,mid+1,e,mid+1,l));
    }
}

void seg_update(int node,int b,int e,int ind,int v){

    if(b>e) return ;

    if(b==e){
        seg_T[node].mx_cost=v;
        return ;
    }

    int left=node<<1,right=left+1,mid=b+e;
    mid/=2;

    if(ind<=mid)  seg_update(left,b,mid,ind,v);
    else        seg_update(right,mid+1,e,ind,v);

    seg_T[node].mx_cost=maxi(seg_T[left].mx_cost,seg_T[right].mx_cost);
}


int lca(int p,int q){

     int log, i;

     //if p is situated on a higher level than q then we swap them
     if (L[p] < L[q])
         swap(p,q);

      //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];

}
```

```
/*
* query_up:
* It takes two nodes u and lc, condition is that lc is an ancestor of u
* We query the chain in which u is present till chain head, then move to next
chain up
* We do that way till u and lc are in the same chain, we query for that part
of chain and break
*/
int query_up(int u,int lc){

    if(u==lc) return 0;
    int u_chain,lc_chain,ret=-1;

    lc_chain=chain_ind[lc];
    while(1){

        if(u==lc) break;

        u_chain=chain_ind[u];

        if(u_chain==lc_chain){
           // Both u and lc are in the same chain, so we need to query from u
to lc, update ret and break.
           // We break because we came from u up till v, we are done

ret=maxi(ret,seg_query(1,1,ptr,pos_in_base[lc]+1,pos_in_base[u]));

            return ret;
        }


ret=maxi(ret,seg_query(1,1,ptr,pos_in_base[chain_head[u_chain]],pos_in_base[u
]));
        // Above is call to segment tree query function. We do from chainHead
of u till u. That is the whole chain from
        // start till head. We then update the answer

        u=chain_head[u_chain]; // move u to u's chainHead
        u=T[u]; //Then move to its parent, that means we changed chains
    }

    return ret;
}


int query(int u,int v){

    int lc=lca(u,v);

    int ret=maxi(query_up(u,lc),query_up(v,lc));

    return ret;
}

void update(int u,int v){

    seg_update(1,1,ptr,pos_in_base[u],v);
}
```

```
int main(){

    int i,j,k,l,test,t=1;

    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    scanf("%d",&test);

    while(test--){

        scanf("%d",&n);

        // init_all
        ptr=0,chain_no=1;
        for(i=0;i<=n;i++){
            G[i].clear();
            W[i].clear();
            edge_ind[i].clear();
            chain_head[i]=-1;
        }


        for(i=1;i<n;i++){
            scanf("%d %d %d",&k,&l,&j);

            G[k].push_back(l);
            W[k].push_back(j);
            edge_ind[k].push_back(i);

            G[l].push_back(k);
            W[l].push_back(j);
            edge_ind[l].push_back(i);
        }

        // init . . .
        dfs(1,1,1);
        init_sparse();
        heavy_light(1,1,0);

        // seg-tree
        init_segtree(1,1,ptr);

        char qs[10];
        int u,v;
        while(1){
            scanf("%s",qs);
            if(qs[0]=='D') break;

            scanf("%d %d",&u,&v);
            if(qs[0]=='Q'){
                printf("%d\n",query(u,v));
            }
            else{
                u=edge[u];
                update(u,v);
            }
```

```
        }
    }

    return 0;}
```

# String Algorithm

## KMP:

S= string , p =pattern.

```
int kmp(){
        int i,j,k,l,ret=0;
        int n,m;
        n=strlen(s); m=strlen(p);
        prefix();
        k=0;

        for(i=1;i<=n;i++){
                while(k>0 && s[i-1]!=p[k]) k=pre[k];
                if(s[i-1]==p[k]) k++;
                if(k==m){
                        ret++;
                        k=pre[k];
                }
        }

        return ret;
}

void prefix(){
        int i,j,k,l;
        l=strlen(p);
        pre[1]=0;
        k=0;

        for(i=2;i<=l;i++){
                while(k>0 && p[k]!=p[i-1]) k=pre[k];
                if(p[k]==p[i-1]) k++;
                pre[i]=k;
        }

}
```

## Aho-corasick :

```
/*
Problem : Beaver (Codeforces Round 71 - problem C ).
Algo    : Aho corasick , DP , Trie
*/
#define maxm 100100
#define inf (1<<29)
int maxi(int a,int b){
   if(a>b) return a;
   return b;
}
int mini(int a,int b){
   if(a<b) return a;
   return b;
}
/////****  Trie + Aho Corasick  ********//////

//  maxc= query...
#define maxc 15
//  maxl = length of query  string...
#define maxl 20
//  maxn =required trie  node ....
#define maxn ((maxc*maxl)+10)
#define cn 64

struct trie{
   //vector<int>v; // for keeping track of patterns ends here ...
   int edges[cn+3],ind;
   int pat_no;  // highest lenght pattern ends at this node...
   int pat_len; // minimum lenght pattern ends at this node...
};
trie Tri[maxn],root;
int len[maxc]; // len[i]= length of pattern i ...
int tot,f[maxn],n,pos[maxc]; //  f=failure , pos[i]=position of ith pattern in trie node .
int c[maxn]; // c[i]  = (number of occurence) count of ith node of trie in string s .
int fin[maxm]; // fin[i]  = minimum lenght of pattern finish at pos i of string s

int getid(char ch){
   if(ch>='a' && ch<='z') return ch-'a';
   else if(ch>='A' && ch<='Z') return ch-'A'+26;
   else if(ch>='0' && ch<='9') return ch-'0'+52;
   if(ch=='_') return cn-1;
   return ch-'a';
}

void init(trie *a,int ind){
   a->ind=ind;
```

```
    a->pat_no=0;
    a->pat_len=inf;
            //a->v.clear();
    memset(a->edges,-1,sizeof(a->edges));
}

void add(trie *a,char *s,int ind){

    int i,l,id;
    l=strlen(s);

    for(i=0;i<=l;i++){
        if(i==l){
            pos[ind]=a->ind;
            a->pat_no=ind;
            a->pat_len=mini(a->pat_len,len[ind]);
            //a->v.push_back(ind);
            continue;
        }
        id=getid(s[i]);
        if(a->edges[id]==-1){
            a->edges[id]=tot;
            init(&Tri[a->edges[id]],tot++);
        }
        a=&Tri[a->edges[id]];
    }

}
void build(){
    int i,j,piv;
    trie *a=&Tri[0];
    for(i=0;i<=cn;i++){
        if(a->edges[i]==-1) a->edges[i]=0;
    }
    // Failure Function .....
    queue<int>q;
    for(i=0;i<=cn;i++){
        if(a->edges[i]){
            f[a->edges[i]]=0;
            q.push(a->edges[i]);
        }
    }
    while(!q.empty()){
        int state=q.front(); q.pop();

                        a=&Tri[state];
```

```
                    //sort(a->v.begin(),a->v.end());
                    //unique(a->v.begin(),a->v.end());

        for(i=0;i<=cn;i++){
            if(a->edges[i]==-1) continue;
            int failure=f[state];
            while(Tri[failure].edges[i]==-1){
                failure=f[failure];
            }
            failure=Tri[failure].edges[i];
            piv=Tri[state].edges[i];
            f[piv]=failure;

            Tri[piv].pat_len=mini(Tri[piv].pat_len,Tri[failure].pat_len);
            /*
            trie *a1=&Tri[failure];
            trie *a2=&Tri[piv];
            for (int ind=0; ind<a1->v.size(); ind++){
                a2->v.push_back(a1->v[ind]);
            }
            */
                                    q.push(piv);
        }
    }
}
void match(char *s);

//////***************  END  **************/////

char s[maxm];
char pat[maxl];
// Problem depenedent.....
int can[maxm]; // can[i] = minimum index of string s from pos i for which it is valid ...
int dp[maxm];  // dp[i]  = store minimum index for which string S obtaining from s[dp[i]]
to s[i] is valid .
int main(){

    int i,j,k,l,test,t=1;
    scanf("%s",s);
    scanf("%d",&n);
    //Init.....
    c[0]=0;
    root=Tri[0];
    init(&Tri[0],tot++);
    for(i=1;i<=n;i++){
```

```
    scanf("%s",pat);
    len[i]=strlen(pat);
    //printf("%s\n",pat);
    add(&Tri[0],pat,i);
}
build(); // building automata
int m;
match(s); // match string

queue<int>q;
vector<int>v;
q.push( 0 );
while( !q.empty()){
    int now = q.front();
    q.pop();
    v.push_back(now);
    for( i=0;i<=cn;i++ ){
        if( Tri[now].edges[i]!=-1 && Tri[now].edges[i]!=0 ) q.push(Tri[now].edges[i]);
    }
}
for( i=v.size()-1;i>=0;i-- ){
    c[f[v[i]]] += c[v[i]];
}
/*

for(i=1;i<=n;i++){
    printf("%d\n",c[pos[i]]);
}

for(i=0;s[i];i++){
    //if(fin[i]==inf) fin[i]=-1;
    printf("%2d ",i);
}
puts("");
*/
int ans=0,mark=0,ans1;
for(i=0;s[i];i++){
    //if(fin[i]==inf) fin[i]=-1;
    can[i]=maxi(i-fin[i]+2,0);
    dp[i]=can[i];
    if(i) dp[i]=maxi(dp[i],dp[i-1]);
    ans1=i-dp[i]+1;
    if(ans1>ans){
        ans=ans1;
        mark=dp[i];
    }
```

```
        //printf("%2d ",can[i]);
    }
    //puts("");
    /*
    for(i=0;s[i];i++){
        //if(fin[i]==inf) fin[i]=-1;
        printf("%2d ",dp[i]);
    }
    puts("");

    for(i=0;s[i];i++){
        //if(fin[i]==inf) fin[i]=-1;
        printf("%2c ",s[i]);
    }
    puts("");
    */
    printf("%d %d\n",ans,mark);
    return 0;
}
int find_next(int curr,char ch){
    int id=getid(ch);
    //printf("curr =%d %c-%d\n",curr,ch,id);
    while(Tri[curr].edges[id]==-1) curr=f[curr];
    //printf("curr =%d %c-%d %d\n",curr,ch,id,Tri[curr].edges[id]);
    return Tri[curr].edges[id];
}
void match(char *s){
    int i,j,l;
    l=strlen(s);
    int curr=0;
    for(i=0;i<l;i++){
        curr=find_next(curr,s[i]);
        c[curr]++;
        fin[i]=Tri[curr].pat_len;
        /*
        trie *a=&Tri[curr];
        for (it=a->v.begin(); it!=a->v.end(); ++it){
            c[*it]++;
        }
        */
    }
}
```

## Suffix Array :

```
const int MAXN = 2005;
```

```
const int MAXL = 22;
int n ,stp,mv,suffix[MAXN],tmp[MAXN];
int sum[MAXN],cnt[MAXN],rank[MAXL][MAXN];
char str[MAXN];
int LCP(int u,int v){
        int ret=0,i;
        for(i = stp; i >= 0; i--){
                if(rank[i][u]==rank[i][v]){
                        ret += 1<<i;
                        u += 1<<i;
                        v += 1<<i;
                }
        }
        return ret;
}
bool equal(int u,int v){
        if(!stp)return str[u]==str[v];
        if(rank[stp-1][u]!=rank[stp-1][v]) return false;
        int a = u + mv < n ? rank[stp-1][u+mv] : -1;
        int b = v + mv < n ? rank[stp-1][v+mv] : -1;
        return a == b ;
}
void update(){
        int i;
        for(i = 0;i < n; i ++) sum[ i ] = 0;

        int rnk = 0;
        for(i = 0;i < n;i++){
                suffix[ i ] = tmp[ i ];
                if( i&&!equal(suffix[i],suffix[i-1])){
                        rank[stp][suffix[i]]=++rnk;
                        sum[rnk+1]=sum[rnk];
                }
                else rank[stp][suffix[i]]=rnk;
                sum[rnk+1]++;
        }
}
void Sort(){
        int i;
        for(i = 0; i < n; i ++ ) cnt[ i ] = 0;
        memset(tmp,-1,sizeof tmp);
        for(i = 0 ; i < mv; i ++){
                int idx = rank[ stp - 1 ][ n-i-1 ];
                int x = sum[ idx ];
                tmp[ x + cnt[ idx ] ] = n-i-1;
                cnt[ idx ]++;
```

```
        }
        for(i = 0;i < n; i ++ ){
                int idx = suffix[ i ] - mv;
                if(idx<0)continue;
                idx = rank[stp-1][idx];
                int x = sum[ idx ];
                tmp[ x + cnt[ idx ] ] = suffix[ i ] - mv;
                cnt[idx]++;
        }
        update();
        return;
}
bool cmp(const int &a,const int &b){
        if(str[a]!=str[b]) return str[a]<str[b];
        return false;
}
int main(){
        scanf("%d", &n);
        scanf ( "%s", str  );
        int i;
        for(i = 0;i < n;i++) tmp[ i ] = i ;

        sort(tmp,tmp+n,cmp);
        stp = 0;
        update();
        ++stp;
        for( mv = 1; mv < n;  mv <<= 1){
                Sort();
                stp++;
        }
        stp--;
        for(i = 0;i<=stp; i++)          rank[ i ][ n ] = -1;
        int res=0;
        for(i = 1; i < n; i ++)
                res=max(res,LCP(suffix[i],suffix[i-1]));
        printf("%d\n",res);
        return 0;
}
```

## Manachar's algorithm:

```
/*
Manacher algorithm implementation.
Application, largest palindromic substring, largest palindromic suffix
*/
```

```
int lengths[MAX<<1];
int manacher(char *buff, int len) {
   int i, k, pallen, found, d, j, s, e;
   k = pallen = 0;
   for(i = 0; i < len; ) {
      if(i > pallen && buff[i-pallen-1] == buff[i]) {
         pallen += 2, i++;
         continue;
      }
      lengths[k++] = pallen;
      s = k - 2, e = s - pallen, found = 0;
      for(j = s; j > e; j--) {
         d = j - e - 1;
         if(lengths[j] == d) {
            pallen = d;
            found = 1;
            break;
         }
         lengths[k++] = (d < lengths[j]? d : lengths[j]);
      }
      if(!found) { pallen = 1; i++; }
   }
   lengths[k++] = pallen;
   return lengths[k-1];
}
```

# Miscellaneous

## Fast Reader :

```
// Fast..... reader.....

const int BUFFSIZE = 10240;
char BUFF[BUFFSIZE + 1], *ppp = BUFF;
int RR, CHAR, SIGN, BYTES = 0;
#define GETCHAR(c) { \
      if(ppp-BUFF==BYTES && (BYTES==0 || BYTES==BUFFSIZE)) { BYTES =
fread(BUFF,1,BUFFSIZE,stdin); ppp=BUFF; } \
      if(ppp-BUFF==BYTES && (BYTES>0 && BYTES<BUFFSIZE)) { BUFF[0] = 0;
ppp=BUFF; } \
      c = *ppp++; \
}

#define DIGIT(c) (((c) >= '0') && ((c) <= '9'))
```

```
#define MINUS(c) ((c)== '-')
#define GETNUMBER(n) { \
      n = 0; SIGN = 1; do { GETCHAR(CHAR); } while(!(DIGIT(CHAR) ||
MINUS(CHAR))); \
      if(MINUS(CHAR)) { SIGN = -1; GETCHAR(CHAR); } \
      while(DIGIT(CHAR)) { n = 10*n + CHAR-'0'; GETCHAR(CHAR); } if(SIGN == -
1) { n = -n; } \
}
/////////*******************************/////////
```

## Knight Distance (infinite Board ) :

```
/*
NK is the size of grid you want to precalculate
NK/2,NK/2 will be considered origin
Calculates minimum knight distance from 0,0 to x,y
*/
const int KN = 101;
i64 dk[KN][KN];
int dx[] = {-1, -1, 1, 1, -2, -2, 2, 2};
int dy[] = {-2, 2, -2, 2, -1, 1, -1, 1};
void precalc() {
   int x, y, x1, y1, i;
   queue< int > Q;
   memset(dk, 0x3f, sizeof dk);
   x = y = (KN >> 1);
   dk[x][y] = 0;
   Q.push(x); Q.push(y);
   while(!Q.empty()) {
      x = Q.front(); Q.pop();
      y = Q.front(); Q.pop();
      for(i = 0; i < 8; i++) {
         x1 = x + dx[i], y1 = y + dy[i];
         if(0 <= x1 && x1 < KN && 0 <= y1 && y1 < KN) {
            if(dk[x1][y1] > dk[x][y] + 1) {
               dk[x1][y1] = dk[x][y] + 1;
               Q.push(x1); Q.push(y1);
            }
         }
      }
   }
}
i64 knight(i64 x, i64 y) {
   i64 step, res = 0;
   if(x < y) swap(x, y);
   while((x<<1) > KN) {
```

```
        step = x / 2 / 2; res += step;
        x -= step * 2; y -= step;
        if(y < 0) y = ((y % 2) + 2) % 2;
        if(x < y) swap(x, y);
    }
    res += dk[x+(KN>>1)][y+(KN>>1)];
    return res;
}
```

## Compress a array :

```
int t[maxm];
int compress(int* a, int n){
  int i, m;
  for(i = 0; i < n; i++) t[i] = a[i];
  sort(t, t+n);
  m = unique(t,t+n)-t;
  for(i = 0; i < n; i++)
    a[i] = lower_bound(t,t+m,a[i])-t;
  return m;
}
```

## Shank's Algorithm:

This algorithm finds **x** ( 0 <= x <= p - 2 ) **for** the equation

b = ax mod p where b, a, p are known

Using the fact that x can be expressed as jm + i, where 0 <= i <= m – 1, 0 <= j < p/m, and m = **ceil**(**sqrt**( p - 1 ))

So, the equation can be written as

b = amj+i mod p

b = amj ai mod p

ba-i = amj mod p

If two lists of ordered **pairs** (i, ba-i) **and** (j, amj), ordered by their second components are built, then it is possible to find one pair from each list that have equal second components. Then x = mj + i, where i and j are the first elements of the matching pairs.

**Code:**

```
/*
Shanks baby step giant step - discrete logarithm algorithm
for the equation: b = a^x % p where a, b, p known, finds x
works only when p is an odd prime
*/

int shank(int a, int b, int p) {
    int i, j, m;
    long long c, aj, ami;
```

```
    map< long long, int > M;
    map< long long, int > :: iterator it;
    m = (int)ceil(sqrt((double)(p)));
    M.insert(make_pair(1, 0));
    for(j = 1, aj = 1; j < m; j++) {
        aj = (aj * a) % p;
        M.insert(make_pair(aj, j));
    }
    ami = modexp(modinv(a, p), m, p);
    for(c = b, i = 0; i < m; i++) {
        it = M.find(c);
        if(it != M.end()) return i * m + it->second;
        c = (c * ami) % p;
    }
    return 0;
}
```

## Negative Base:

```
string negaBase(int n,int b){
        int i,tmp;
        string a;
        for(i=0;n;i++){
                tmp=n%b; n=n/b;
                if(tmp<0) {     tmp+= (-b),     n++;            }
                a+='0'+tmp;
        }
        for(n=0;n<(i/2);n++) swap(a[n],a[i -n - 1]);
        if(i) return a;
        return "0";
}
```

## Double Hashing :

```
/*
M > N and should be close, better both be primes.
M should be as much large as possible, not exceeding array size.
HKEY is the Hash function, change it if necessary.
*/

#define NIL -1
#define M 1021
#define N 1019
#define HKEY(x,i) ((x)%M+(i)*(1+(x)%N))%M
```

```
int a[M+1];

inline int hash(int key) {
   int i = 0, j;
   do {
      j = HKEY(key, i);
      if(a[j]==NIL) { a[j] = key; return j; }
      i++;
   } while(i < M);
   return -1;
}

inline int find(int key) {
   int i = 0, j;
   do {
      j = HKEY(key, i);
      if(a[j]==key) return j;
      i++;
   } while(a[j]!=NIL && i < M);
   return -1;
}
```

## Joseph:

```
int joseph(int n,int k){
            if(n==1) return 0;
            return ((joseph(n-1,k)+k)%n);
}
```

## GEO Template:

```
// Geometry Templates>>>>>>>>>>>>
// Header File
//Macro.....
// Structure....

// distance between point to point...
inline double distancepp( point a, point b ) {
   return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ) );
}
```

```
// distance between point to point...
inline double distancepp( point3D a, point3D b ) {
   return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ) + ( a.z - b.z ) * ( a.z -
b.z ) );
}
// square distance between point to point.
inline double sq_distance( point a, point b ) {
   return ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y );
}
// distance between point to line....
inline double distancepl( point P, line L ) {
   return fabs( L.a * P.x + L.b * P.y + L.c ) / sqrt( L.a * L.a + L.b * L.b );
}
//  cross product = p0p1 * p0p2..
inline double cross( point p0, point p1, point p2 ) {
   return( ( p1.x - p0.x ) * ( p2.y - p0.y ) - ( p2.x - p0.x ) * ( p1.y - p0.y ) );
}
// cross product
inline double cross(point p1, point p2 ) {
   return( ( p1.x * p2.y ) - ( p2.x *  p1.y ) );
}

//Intersection - Line, Line:
inline bool intersection( line L1, line L2, point &p ) {
            double det = L1.a * L2.b - L1.b * L2.a;
            if( eq ( det, 0 ) ) return false;
            p.x = ( L1.b * L2.c - L2.b * L1.c ) / det;
            p.y = ( L1.c * L2.a - L2.c * L1.a ) / det;
            return true;
}

//Intersection - Segment, Segment:
inline bool intersection( segment L1, segment L2, point &p ) {
            if( !intersection( line( L1.A, L1.B ), line( L2.A, L2.B ), p) ) {
                        return false; // can lie on another, just check their equations, and check
overlap
            }
            return(eq(distancepp(L1.A,p)+distancepp(L1.B,p),distancepp(L1.A,L1.B))
&&
                        eq(distancepp(L2.A,p)+distancepp(L2.B,p),distancepp(L2.A,L2.B)));
}
//Perpendicular Line of a Given Line Through a Point:
inline line findPerpendicularLine( line L, point P ) {
            line res; //line perpendicular to L, and intersects with P
            res.a = L.b, res.b = -L.a;
```

```
        res.c = -res.a * P.x - res.b * P.y;
        return res;
}
//Distance - Point, Segment:
inline double distanceps( point P, segment S ) {
        line L1 = line(S.A,S.B), L2; point P1;
        L2 = findPerpendicularLine( L1, P );
        if( intersection( L1, L2, P1 ) )
                if( eq ( distancepp( S.A, P1 ) + distancepp( S.B, P1 ), distancepp(
S.A, S.B ) ) )
                        return distancepl(P,L1);
        return mini ( distancepp( S.A, P), distancepp( S.B, P) );
}
//  area of  polygon.....
double areaPoly(point P[],int n){
   double area=0;
   for( int i = 0, j = n - 1; i < n; j = i++ ) area += P[j].x * P[i].y - P[j].y * P[i].x;
   return fabs(area)*.5;
}
//  intersecting point between  circle and line...
inline bool intersectioncl(circle C,line L,point &p1,point &p2) {
        if( distancepl( C.center, L ) > C.r + eps ) return false;
        double a, b, c, d, x = C.center.x, y = C.center.y;
        d = C.r*C.r - x*x - y*y;
        if( eq( L.a, 0) ) {
                p1.y = p2.y = -L.c / L.b;
                a = 1;
                b = 2 * x;
                c = p1.y * p1.y - 2 * p1.y * y - d;
                d = b * b - 4 * a * c;
                d = sqrt( fabs (d) );
                p1.x = ( b + d ) / ( 2 * a );
                p2.x = ( b - d ) / ( 2 * a );
        }
        else {
                a = L.a *L.a + L.b * L.b;
                b = 2 * ( L.a * L.a * y - L.b * L.c - L.a * L.b * x);
                c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
                d = b * b - 4 * a * c;
                d = sqrt( fabs(d) );
                p1.y = ( b + d ) / ( 2 * a );
                p2.y = ( b - d ) / ( 2 * a );
                p1.x = ( -L.b * p1.y -L.c ) / L.a;
                p2.x = ( -L.b * p2.y -L.c ) / L.a;
        }
```

```
        return true;
}

//Find Points that are r1 unit away from A, and r2 unit away from B:
inline bool findpointAr1Br2(point A,double r1,point B, double r2,point &p1,point &p2) {
        line L;
        circle C;
        L.a = 2 * (B.x - A.x );
        L.b = 2 * (B.y - A.y );
        L.c = A.x * A.x + A.y * A.y - B.x * B.x - B.y * B.y + r2 * r2 - r1 * r1;
        C.center = A;
        C.r = r1;
        return intersectioncl( C, L, p1, p2 );
}

//Intersection Area between Two Circles:
inline double intersectionArea2C( circle C1, circle C2 ) {
        C2.center.x = distancepp( C1.center, C2.center );
        C1.center.x = C1.center.y = C2.center.y = 0;
        if( C1.r < C2.center.x - C2.r + eps ) return 0;
        if( -C1.r + eps > C2.center.x - C2.r ) return pi * C1.r * C1.r;
        if( C1.r + eps > C2.center.x + C2.r ) return pi * C2.r * C2.r;
        double c, CAD, CBD, res;
        c = C2.center.x;
        CAD = 2 * acos( (C1.r * C1.r + c * c - C2.r * C2.r) / (2 * C1.r * c) );
        CBD = 2 * acos( (C2.r * C2.r + c * c - C1.r * C1.r) / (2 * C2.r * c) );
        res=C1.r * C1.r * ( CAD - sin( CAD ) ) + C2.r * C2.r * ( CBD - sin ( CBD ) );
        return .5 * res;
}

//Circle Through Thee Points:
circle CircleThrough3points( point A, point B, point C) {
        double den; circle c;
        den = 2.0 *((B.x-A.x)*(C.y-A.y) - (B.y-A.y)*(C.x-A.x));
        c.center.x =( (C.y-A.y)*(B.x*B.x+B.y*B.y-A.x*A.x-A.y*A.y)- (B.y-
A.y)*(C.x*C.x+C.y*C.y-A.x*A.x-A.y*A.y) );
        c.center.x /= den;
        c.center.y =( (B.x-A.x)*(C.x*C.x+C.y*C.y-A.x*A.x-A.y*A.y) - (C.x-
A.x)*(B.x*B.x+B.y*B.y-A.x*A.x-A.y*A.y) );
        c.center.y /= den;
        c.r = distancepp( c.center, A );
        return c;
}

// Rotating a Point anticlockwise by 'theta' radian w.r.t Origin:
```

```
inline point rotate2D( point P ,double theta) {
        point Q;
        Q.x = P.x * cos( theta ) - P.y * sin( theta );
        Q.y = P.x * sin( theta ) + P.y * cos( theta );
        return Q;
}


double ang(point a,point b,point c){    //returns angle <bac
   double absq = sq_distance(a , b);
   double bcsq = sq_distance(c , b), acsq = sq_distance(a , c);
   double cosp = (absq+acsq - bcsq)/(2.0*sqrt(absq * acsq) );
   return acos(cosp);
}
// radian to degree.
double convrd(double theta){
   double ret=180; ret/=pi; return ret*theta;
}
// degree to radian...
double convdr(double theta){
   double ret=pi; ret/=(double)180.0; return ret*theta;
}


// check whether a point lies inside a quad....
bool inside_quad(quad q,point p){

        double val=cross(q.p[0],q.p[1],p)*cross(q.p[3],q.p[2],p);
   if(val>0) return 0;

        val=cross(q.p[0],q.p[3],p)*cross(q.p[1],q.p[2],p);
   if(val>0) return 0;

        return 1;
}
// check whether a point lies inside a segment....
bool inside_segment(segment S,point P){
   if( eq ( distancepp( S.A, P ) + distancepp( S.B, P ), distancepp( S.A, S.B ) ) )   return
1;
   return 0;
}

// calculate slope.....
double inline cal_slope(point p1,point p2){
   double num,den;
```

```
    num=p2.y-p1.y;
    den=p2.x-p1.x;
    if(iseq(den,0)) return inf;
    return num/den;
}

bool sort_x(point a,point b){
    if(iseq(a.x,b.x)) return a.y<b.y;
    return a.x<b.x;
}
bool sort_y(point a,point b){
    if(iseq(a.y,b.y)) return a.x<b.x;
    return a.y<b.y;
}

// newly added ..  ( need  modification..... ) .....

bool is_square(segment l[5]){

    if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[1].A,l[1].B))) return 0;
    if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[2].A,l[2].B))) return 0;
    if(!iseq(sq_distance(l[2].A,l[2].B),sq_distance(l[3].A,l[3].B))) return 0;
    if(!iseq(sq_distance(l[3].A,l[3].B),sq_distance(l[0].A,l[0].B))) return 0;

            point p[5];
            p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

            int i,j,k;
            double com=pi;
            for(i=0,j=3;i<4;j=i++){
                    k=i+1; k%=4;
                    double val=ang(p[i],p[j],p[k]);
                    val*=2.0;
                    if(!iseq(val,pi)) return 0;
            }

    return 1;
}

bool is_rect(segment l[5]){

    if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[2].A,l[2].B))) return 0;
    if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[3].A,l[3].B))) return 0;

            point p[5];
```

```
        p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

        int i,j,k;
        double com=pi;
        for(i=0,j=3;i<4;j=i++){
                k=i+1; k%=4;
                double val=ang(p[i],p[j],p[k]);
                val*=2.0;
                if(!iseq(val,pi)) return 0;
        }
   return 1;
}

// check parallelogram.....
bool is_para(segment l[5]){

   if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[2].A,l[2].B))) return 0;
   if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[3].A,l[3].B))) return 0;

        point p[5];
        p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

        int i,j,k;
        double com=pi;
        for(i=0,j=3;i<4;j=i++){
                k=i+1; k%=4;
                double val=ang(p[i],p[j],p[k]);
                val*=2.0;
                if(iseq(val,pi)) return 0;
        }
   return 1;
}

bool is_rhombus(segment l[5]){

   if(!iseq(sq_distance(l[0].A,l[0].B),sq_distance(l[1].A,l[1].B))) return 0;
   if(!iseq(sq_distance(l[1].A,l[1].B),sq_distance(l[2].A,l[2].B))) return 0;
   if(!iseq(sq_distance(l[2].A,l[2].B),sq_distance(l[3].A,l[3].B))) return 0;
   if(!iseq(sq_distance(l[3].A,l[3].B),sq_distance(l[0].A,l[0].B))) return 0;

        point p[5];
        p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

        int i,j,k;
        double com=pi;
```

```
            for(i=0,j=3;i<4;j=i++){
                        k=i+1; k%=4;
                        double val=ang(p[i],p[j],p[k]);
                        val*=2.0;
                        if(iseq(val,pi)) return 0;
            }

   return 1;
}

// check trapezium.....
bool is_trap(segment l[5]){

            point p[5];
            p[0]=l[0].A; p[1]=l[0].B; p[2]=l[1].B; p[3]=l[2].B;

            int ans1=0,ans2=0;

            if(iseq(cross(point(p[1].x-p[0].x,p[1].y-p[0].y),point(p[2].x-p[3].x,p[2].y-
p[3].y)),0.0)) ans1=1;
   if(iseq(cross(point(p[1].x-p[2].x,p[1].y-p[2].y),point(p[0].x-p[3].x,p[0].y-p[3].y)),0.0))
ans2=1;
   return ans2^ans1;
}
// convert spherical to cartesian co-ordinate.....
void sph_to_cartesian(double R,double lat,double lng,point3D &p){

   lat=convdr(lat);
   lng=convdr(lng);

   p.x=R*sin(lat)*cos(lng);
   p.y=R*sin(lat)*sin(lng);
   p.z=R*cos(lat);
}
// convert longitude/latitude to cartesian co-ordinate.....
void earth_to_cartesian(double R,double lat,double lng,point3D &p){

   lat=convdr(lat);
   lng=convdr(lng);

   p.x=R*cos(lat)*cos(lng);
   p.y=R*cos(lat)*sin(lng);
   p.z=R*sin(lat);}
// geo template end>>>>>
```

## Recently Added:

### Meet in the middle + Ternary Mask:

```
struct ternary{
    ii pow3[maxm];

    void init(){
        init(maxm-1);
    }
    void init(int n){
        pow3[0]=1;
        for(int i=1;i<=n;i++){
            pow3[i]=pow3[i-1]*3;
        }
    }
    int get_bit(int mask,int k){
        ii tmp=mask; tmp/=pow3[k];
        return (tmp%3);
    }
    int set_bit(int mask,int k,int v){
        ii tmp=mask;
        tmp/=pow3[k];
        tmp%=3;
        mask-=(tmp*pow3[k]);
        mask+=(v*pow3[k]);
        return mask;
    }
};
ternary t_mask;

int n,req;
int a[maxm],b[maxm];

set<int>can_set;

int build(int mask,int a[],int n,int flag){

    int ret=0;

    for(int i=0;i<n;i++){
                int bit_val=t_mask.get_bit(mask,i);

                for(int j=1;j<=bit_val;j++){
                        ret+=a[i];
                        if(ret>req) return -1;
                }
    }

        if(flag) can_set.insert(ret);
    return ret;
}

int main(){

    int i,j,k,l,test,t=1;

    t_mask.init();
```

```
    scanf("%d",&test);

    while(test--){

                can_set.clear();

        scanf("%d %d",&n,&req);

        int n1=n/2,n2=n-n1;

        for(i=0;i<n1;i++){
            scanf("%d",&a[i]);
        }
                for(i=n1,j=0;i<n;i++,j++){
                        scanf("%d",&b[j]);
                }


        int tot=0;
        for(i=0;i<t_mask.pow3[n1];i++){
            build(i,a,n1,1);
        }

                bool soln_found=false;
        for(i=0;i<t_mask.pow3[n2];i++){
            int now=build(i,b,n2,0);
                        if(now==-1) continue;
                        now=req-now;
                        if(can_set.find(now)!=can_set.end()){
                                soln_found=true;
                                break;
                        }
        }

        printf("Case %d: ",t++);
        if(soln_found==true){
            printf("Yes\n");
        }
        else{
            puts("No");
        }
    }
    return 0;
}
```

## Treap :

```
/*
Algo        : Treap (Balanced BST).
Problem     : Spoj - Yet another range difference query.
*/

//  TREAP  >>>>>>>>>>>>>>>>>>>>>>>>>

typedef int treap_type;
```

```
struct node{

    treap_type value,min_val,max_val,diff;
    ii priority;
    int cnt;
    node *left,*right;

    node(){}
    node(treap_type _value){
        cnt=1;
        value=min_val=max_val=_value;
        diff=inf;
        priority=rand();
        left=right=NULL;
    }
};

// 1-based . . . . . .
struct treap{

    node *root;

    void fix(node * &t){
        if(t==NULL) return ;
        t->cnt=get_count(t->left)+get_count(t->right)+1;
                t->diff=inf;

        if(t->left){
            t->min_val=t->left->min_val;
            t->diff=mini(t->left->diff,t->value - t->left->max_val);
        }
                else t->min_val=t->value;

        if(t->right){
            t->max_val=t->right->max_val;
            t->diff=mini( t->diff,mini( t->right->diff , t->right->min_val-
t->value) );
        }
                else t->max_val=t->value;
    }

    inline int get_count(node* t){
        return t ? t->cnt : 0;
    }

    inline void left_rotate(node* &t){
        node* tmp = t->left;
        t->left = tmp->right;
        tmp->right = t;
        t = tmp;
    }

    inline void right_rotate(node* &t){
        node* tmp = t->right;
        t->right = tmp->left;
        tmp->left = t;
        t = tmp;
    }
```

```
bool insert(node * &t,treap_type value){

    if(t==NULL){
        t=new node(value);
        fix(t);
        return true;
    }

    if(t->value==value) return false;
    bool ret;

    if(value < t->value) ret=insert(t->left,value);
    else                 ret=insert(t->right,value);

    if(t->left && t->left->priority > t->priority){
        left_rotate(t);
    }
    else if(t->right && t->right->priority > t->priority){
        right_rotate(t);
    }

    if(t->left) fix(t->left);
    if(t->right) fix(t->right);
    fix(t);

    return ret;
}

bool insert(treap_type value){
    return insert(root,value);
}

inline ii get_priority(node* t){
    return t ? t->priority : -1;
}

bool erase(node* &t, treap_type val){

    if(!t) return false;

    bool ret;
    if(t->value != val){
        ret=erase(val < t->value ? t->left : t->right, val);
    }
    else{
        if(!t->left && !t->right){
            delete t;
            t = NULL;
        }else{
            if(get_priority(t->left) < get_priority(t->right))
                right_rotate(t);
            else
                left_rotate(t);

            ret=erase(t, val);
        }
    }
```

```
        if(t){
            if(t->left)  fix(t->left);
            if(t->right) fix(t->right);
            fix(t);
        }

        return ret;
    }

    bool erase(treap_type value){
                return erase(root,value);
    }

    node* find(node * t,int value){

        if(t==NULL) return NULL;

                node *ret;
        if(value > t->value){
            ret=find(t->right,value);
                        if(ret==NULL) return t;
            return ret;
        }
        else if(value==t->value){
            return t;
        }
        else{
            return find(t->left,value);
        }
    }

    node* find(int value){
        return find(root,value);
    }

    treap_type find_max(int beg,int end){
        if(beg>end) swap(beg,end);
        return 0;
    }

    treap_type find_min(int beg,int end){
        if(beg>end) swap(beg,end);
        return 0;
    }
};

// END >>>>>>>>>>>>>>>>>>

treap tree;

ii find_min(node *t,int left,int beg,int end){

    if(t==NULL) return inf;

    ii r_left=left;
    ii r_right=left+t->cnt-1;

    ii sz=tree.get_count(t->left)+1;
        ii curr_ind=left+tree.get_count(t->left);
```

```
    if(beg<=r_left && end>=r_right){
        return t->diff;
    }

    ii ret=inf;

    if(beg>curr_ind)              ret=mini(ret,find_min(t-
>right,curr_ind+1,beg,end));
    else if(end<curr_ind)         ret=mini(ret,find_min(t-
>left,left,beg,end));

    else                          ret=mini(mini(find_min(t->left,left,beg,end)
                                           ,find_min(t-
>right,curr_ind+1,beg,end)),ret);

    if(curr_ind>beg && curr_ind<=end && t->left){
        ret=mini(ret,t->value - t->left->max_val);
    }
    if(curr_ind<end && curr_ind>=beg && t->right){
        ret=mini(ret,t->right->min_val - t->value);
    }

    return ret;
}

ii find_min(int k,int l){

    if(l<=k) return -1;
    return find_min(tree.root,1,k,l);
}

ii find_kth(node *t,int kth){

    if(tree.get_count(t)<kth){
                return -1;
    }

    int sz=tree.get_count(t->left)+1;

    if(sz==kth){
        return t->value;
    }

    if(kth<sz){
        return find_kth(t->left,kth);
    }
    else{
        return find_kth(t->right,kth-sz);
    }
}

ii find_max(int k,int l){

    if(l<=k) return -1;
    return find_kth(tree.root,l)-find_kth(tree.root,k);
}
```

```
int find_count(node *t,int value){

        if(t==NULL) return 0;

        if(value==t->value)    return tree.get_count(t->left);
        else if(value > t->value)  return tree.get_count(t->left)

        +find_count(t->right,value)+1;
        else return find_count(t->left,value);
}

int n,m;
char type[5];

int main(){

    int i,j,k,l,test,t=1,val;

    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    scanf("%d",&test);

    while(test--){

        scanf("%s",type);

        //printf("%s\n",type);

        if(type[0]=='I'){
            scanf("%d",&val);
            tree.insert(val);
        }

        if(type[0]=='D'){
            scanf("%d",&val);
            tree.erase(val);
        }

        if(type[0]=='N'){
            scanf("%d %d",&k,&l);
            printf("%d\n",find_min(k+1,l+1));
        }

        if(type[0]=='X'){
            scanf("%d %d",&k,&l);
            printf("%d\n",find_max(k+1,l+1));
        }

    }

    return 0;
}
```

**Suffix Automation :**

```
/*
```

```
Algo        : Suffix-Automation.
Problem     : Codeforces 235c- Cyclical Quest.
*/

int n;
char s[maxm];

// Suffix-Automation
struct state {
        int len, link;
        bool suffix;
        ii count;
        map<char,int> next;
};

const int MAXLEN = maxm+2;
state st[MAXLEN*2];
int sz, last;
pair<int, int> sorter[MAXLEN * 2 + 10];

inline void sa_init() {
        sz = last = 0;
        st[0].len = 0;
        st[0].link = -1;
        st[0].count = 0;
        st[0].suffix=0;
        ++sz;
}

inline void sa_extend (char c) {

        int cur = sz++;
        st[cur].len = st[last].len + 1;
        st[cur].suffix=0;
        st[cur].count=1;

        int p;
        for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link)
                st[p].next[c] = cur;
        if (p == -1)
                st[cur].link = 0;
        else {
                int q = st[p].next[c];

                if (st[p].len + 1 == st[q].len)
                        st[cur].link = q;
                else {
                        int clone = sz++;
                        st[clone].len = st[p].len + 1;
                        st[clone].next = st[q].next;
                        st[clone].link = st[q].link;

                        for (; p!=-1 && st[p].next[c]==q; p=st[p].link)
                                st[p].next[c] = clone;
                        st[q].link = st[cur].link = clone;
                }
        }
        last = cur;
}
```

```
//  Suffix-Automation End. ..

void post_process(){

    int i;

    for(i=0;i<sz;i++){
        sorter[i]=mp(st[i].len,i);
    }
    sort(sorter,sorter+sz);

    for(i=sz-1;i>=0;i--){
        int ind=sorter[i].vv;
        st[st[ind].link].count+=st[ind].count;
    }
}

vector<pii>ans;
int pre[maxm];


void failure(char *p){

        int i,j,k,l;

        l=strlen(p);
        pre[1]=0;
        k=0;

        for(i=2;i<=l;i++){
                while(k>0 && p[k]!=p[i-1]) k=pre[k];
                if(p[k]==p[i-1]) k++;
                pre[i]=k;
        }

}


ii cal(char *s,int lim){

    int i;

    int curr_st=0,len=0;
    ii ret=0;

    failure(s);

    /*for(i=0;s[i];i++){
        printf("%d ",pre[i+1]);
    }
    puts("");
    */

    for(i=0;s[i];i++){

        while (curr_st && !st[curr_st].next.count(s[i])) {
            curr_st = st[curr_st].link;
            len=st[curr_st].len;
        }
```

```
        if (st[curr_st].next.count(s[i])) {
            curr_st = st[curr_st].next[s[i]];
            len++;
        }
        while(st[st[curr_st].link].len>=lim){
            curr_st=st[curr_st].link;
        }

        if(len>=lim && pre[i+1]<lim){
            ret+=st[curr_st].count;
        }
    }

    return ret;
}

char tmp[maxm];

int main(){

    int i,j,k,l;

    //freopen("in.txt","r",stdin);
    //freopen("out.txt","w",stdout);

    scanf("%s",s);

    sa_init();

    for(i=0;s[i];i++){
        sa_extend(s[i]);
    }

    post_process();

    int q;
    int len;

    scanf("%d",&q);

    for(i=1;i<=q;i++){
        scanf("%s",tmp);
        len=strlen(tmp);
        strcpy(s,tmp);
        for(j=len,k=0;k<len-1;k++,j++){
            s[j]=tmp[k];
        }
        s[j]=0;
        //puts(s);
        printf("%I64d\n",cal(s,len));
    }

    return 0;
}
```

**IDA*:**

```
int ida_star(int puzzle[maxm][maxm]){

    int i,j;
    node root;

    for(i=1;i<=4;i++){
        for(j=1;j<=4;j++){
            root.puzzle[i][j]=puzzle[i][j];
        }
    }

    curr_node=root;

    int bound=mini(50,heuristic());

    solution="";
    while(true){
        curr_node=root;
        pii zero=find_pos(root.puzzle,0);
        int next_bound=ida_search(zero,bound,0);
        if(next_bound<=bound) return 1;
        next_bound=mini(55,next_bound);
        //if(next_bound<=bound) break;
        bound=next_bound;
    }

    return 1;
}

int ida_search(pii pos_zero,int bound,int d){

    int f=heuristic();
    if(f+d>bound) return f+d;

    if(!f){
        if(solution.size()==0 || solution.size()>d+1){
            soln[d]=0;
            solution=soln;
        }
        return f;
    }

    int ret=-1,x,y;
    for(int i=0;i<4;i++){
        if(d && soln[d-1]==move[3-i]){
            continue;
        }
        x=dirx[i],y=diry[i];
        pii pos=pos_zero;
        int ret1=ida_search(new_pos,bound,d+1);
        if(!ret1) return ret1;
        // move
        if(ret==-1) ret=ret1;
        ret=mini(ret,ret1);
        // reverse move

    }
    return ret;
}
```

```
int heuristic(){
    int i,j,ret=0;

    return ret;
}
```

**Default Template :**

```
/*
Author      : Rashedul Hasan Rijul ( Silent_coder ).
Created on  : 2014-09-12
*/

#include<stdio.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
#include<ctype.h>
#include<iostream>
#include<algorithm>
#include<vector>
#include<string>
#include<queue>
#include<stack>
#include<map>
#include<set>
using namespace std;

#define maxm 2010
#define inf (1<<29)
#define ii int

#define pi  acos(-1.0)
#define eps 1e-9
#define iseq(a,b) (fabs(a-b)<eps)

#define pii pair<int,int>
#define mp  make_pair
#define uu first
#define vv second

ii on(ii n,ii k){ return (n|(1<<k)); }
ii off(ii n,ii k){ return (n-(n&(1<<k))); }
bool chck(ii n,ii k){ return (n&(1<<k)); }

ii mini(ii a,ii b){ if(a<b) return a;  return b;  }
ii maxi(ii a,ii b){ if(a>b) return a;  return b;  }

int n,m;

int main(){
```

```
int i,j,k,l,test,t=1;

//freopen("in.txt","r",stdin);
//freopen("out.txt","w",stdout);

scanf("%d",&test);

while(test--){
}
return 0;}
```