

CODE LIBRARY

MD. SHAHADAT HOSSAIN SHAHIN

UNIVERSITY OF DHAKA

Table of Contents

TEMPLATE.....	3
DISJOINT SET UNION.....	3
MINIMUM SPANNING TREE (KRUSKAL)	5
DIJKSTRA	5
BELLMAN FORD	6
FLOYD WARSHAL	6
STRONGLY CONNECTED COMPONENT	7
ARTICULATION POINT	8
BRIDGE	9
BRIDGE TREE.....	9
BICONNECTED COMPONENT.....	11
EULER TRAIL/CIRCUIT	13
DIRECTED GRAPH	13
UNDIRECTED GRAPH.....	14
CENTROID DECOMPOSITION.....	16
MAXIMUM FLOW	20
EDMONDS KARP	20
DINIC.....	21
MINIMUM COST MAXIMUM FLOW	22
MAXIMUM BIPARTITE MATCHING.....	24
SEGMENT TREE	26
POINT UPDATE, RANGE QUERY.....	26
RANGE UPDATE, RANGE QUERY.....	26
HOW MANY NON ZERO ELEMENTS IN THE TREE.....	27
IMPLICIT SEGMENT TREE	28
POINT UPDATE, RANGE QUERY.....	28
RANGE UPDATE, RANGE QUERY.....	29
BINARY INDEXED TREE.....	31
MO'S ALGORITHM	31
LOWEST COMMON ANCESTOR	32
TRIE	33
ARRAY IMPLEMENTATION	33

HEAVY LIGHT DECOMPOSITION	34
TREAP	36
NORMAL TREAP	36
IMPLICIT TREAP.....	38
MAXIMUM CONTIGUOUS SUM MERGING	40
MATRIX EXPONENTIATION	41
EXTENDED EUCLID (SOLVING LINEAR DIOPHANTINE EQUATION)	42
FRACTION CLASS.....	45
SIEVE OF ERATOSTHENES.....	46
GAUSSIAN ELIMINATION	47
GAUSS-JORDAN ELIMINATION	48
GAUSS RELATED PROBLEM	48
CHINESE REMAINDER THEOREM	51
LUCAS THEOREM & MODULO OPERATION WITH COMPOSITE NUMBER.....	51
FIBONACCI NUMBERS	53
KNUTH MORRIS PRATT	54
SUFFIX ARRAY.....	55
LONGEST INCREASING SUBSEQUENCE (NLOG(N))	56

<http://www.planetb.ca/syntax-highlight-word>

<http://remove-line-numbers.ruurtjan.com/>

Template

```
1. #include <bits/stdc++.h>
2.
3. using namespace std;
4.
5. typedef long long ll;
6. typedef unsigned long long ull;
7. typedef long double ld;
8.
9. #define si(a)          scanf("%d",&a)
10. #define sii(a,b)       scanf("%d %d",&a,&b)
11. #define sii(a,b,c)     scanf("%d %d %d",&a,&b,&c)
12.
13. #define sl(a)          scanf("%lld",&a)
14. #define sll(a,b)       scanf("%lld %lld",&a,&b)
15. #define slll(a,b,c)    scanf("%lld %lld %lld",&a,&b,&c)
16.
17. #define pb             push_back
18. #define PII            pair <int,int>
19. #define PLL            pair <ll,ll>
20. #define mp             make_pair
21. #define xx             first
22. #define yy             second
23. #define all(v)         v.begin(),v.end()
24. #define un(x)          x.erase(unique(all(x)), x.end())
25.
26. #define D(x)           cerr << #x " = " << (x) << '\n'
27. #define DBG            cerr << "In" << '\n'
28.
29. #define CLR(a)          memset(a,0,sizeof(a))
30. #define SET(a)          memset(a,-1,sizeof(a))
31.
32. #define eps            1e-9
33. #define PI             acos(-1.0)
34. #define MAX            300010
35. #define MOD            1000000007
36. #define INF            2000000000
37.
38. int setBit(int n,int pos){ return n = n | (1 << pos); } //sets the pos'th bit to 1
39. int resetBit(int n,int pos){ return n = n & ~(1 << pos); } //sets the pos'th bit to 0
40. bool checkBit(int n,int pos){ return (bool)(n & (1 << pos)); } //returns the pos'th bit
41.
42. //int dx[] = {+0, +0, +1, -1, -1, +1, -1, +1};
43. //int dy[] = {-1, +1, +0, +0, +1, +1, -1, -1}; //Four & Eight Direction
```

Disjoint Set Union

```

1. //Time complexity = 5*number of operations
2. struct DisjointSet{
3.     int *root,*rnk,n;
4.     DisjointSet(){}
5.     DisjointSet(int sz){
6.         root = new int[sz+1];
7.         rnk = new int[sz+1];
8.         n = sz;
9.     }
10.    ~DisjointSet(){
11.        delete[] root;
12.        delete[] rnk;
13.    }
14.    void init(){
15.        for(int i=1;i<=n;i++){
16.            root[i] = i;
17.            rnk[i] = 0;
18.        }
19.    }
20.    int findRoot(int u){
21.        if(u!=root[u]) root[u] = findRoot(root[u]);
22.        return root[u];
23.    }
24.    void Merge(int u,int v){
25.        int ru = findRoot(u); int rv = findRoot(v);
26.        if(rnk[ru]>rnk[rv]) root[rv] = ru;
27.        else root[ru] = rv;
28.        if(rnk[ru]==rnk[rv]) rnk[rv]++;
29.    }
30. };
31.
32. int main(){
33.     DisjointSet *S;
34.     S = new DisjointSet(n);
35.     S->init();
36.     int ru = S->findRoot(u);
37.     S->Merge(u,v);
38.     delete S;
39.
40.     //or
41.
42.     DisjointSet S(n);
43.     S.init();
44.     int ru = S.findRoot(u);
45.     S.Merge(u,v);
46.     return 0;
47. }

```

Minimum Spanning Tree (Kruskal)

```
1. struct edge{
2.     int u,v,c;
3. }ara[MAX];
4.
5. bool cmp(edge a,edge b) { return a.c<b.c;}
6. int par[MAX];
7.
8. int findParent(int u){
9.     while(par[u]!=u) u = par[u];
10.    return u;
11. }
12. /*int findParent(int u){
13.     if(par[u]==u) return u;
14.     else return par[u] = findParent(par[u]);
15. }*/
16.
17. int kruskal(int n,int m){
18.     sort(ara+1,ara+m+1,cmp);
19.     int i,mst;
20.     mst = 0;
21.     for(i=1;i<=n;i++) par[i] = i;
22.     for(i=1;i<=m;i++){
23.         edge x = ara[i];
24.         par[x.u] = findParent(x.u);
25.         par[x.v] = findParent(x.v);
26.         if(par[x.u]!=par[x.v]){
27.             par[par[x.u]] = par[x.v];
28.             mst += x.c;
29.         }
30.     }
31.     return mst;
32. }
```

Dijkstra

```
1. vector <int> ed[MAX],co[MAX];
2. int dis[MAX];
3. bool vis[MAX];
4.
5. struct node{
6.     int city,cost;
7. };
8.
9. bool operator < (node a,node b){return a.cost>b.cost;}
10.
11. void dijkstra(int s,int n)
12. {
13.     CLR(vis);
14.     int i,x,u,v,c;
15.     node a,b;
16.     for(i=1;i<=n;i++) dis[i] = INF;
17.     dis[s] = 0;
18.     a = {s,0};
```

```
19. priority_queue <node> q;
20. q.push(a);
21. while(!q.empty()){
22.     a = q.top();
23.     q.pop();
24.     u = a.city;
25.     if(!vis[u]){
26.         vis[u] = true;
27.         for(i=0;i<ed[u].size();i++){
28.             v = ed[u][i];
29.             c = co[u][i];
30.             if(dis[v]>dis[u]+c){
31.                 dis[v] = dis[u]+c;
32.                 b = {v,dis[v]};
33.                 q.push(b);
34.             }
35.         }
36.     }
37. }
38. }
```

Bellman Ford

```
1. int dis[MAX];
2. struct data{
3.     int u,v,c;
4. } edge[MAX];
5.
6. bool bellmanFord(int n,int e,int s){
7.     int i,j;
8.     for(i=1;i<=n;i++) dis[i] = INF;
9.     dis[s] = 0;
10.    for(j=1; j<=n-1; j++){
11.        for(i=1; i<=e; i++){
12.            if(dis[edge[i].u]!=INF && dis[edge[i].u]+edge[i].c<dis[edge[i].v])
13.                dis[edge[i].v] = dis[edge[i].u]+edge[i].c;
14.        }
15.    }
16.    bool negativeCycle = false;
17.    for(i=1; i<=e; i++){
18.        if(dis[edge[i].u]!=INF && dis[edge[i].u]+edge[i].c<dis[edge[i].v]){
19.            negativeCycle = true;
20.        }
21.    }
22.    return negativeCycle;
23. }
```

Floyd Warshal

```
1. int dis[MAX][MAX],P[MAX][MAX];
2. void warshall(int n){
3.     int i,j,k;
4.     for(i=0; i<n; i++)
5.         for(j=0; j<n; j++){
6.             if(dis[i][j]!=INF) P[i][j] = i;
```

```
7.         else P[i][j] = -1;
8.     }
9.     for(k=0; k<n; k++){
10.        for(i=0; i<n; i++){
11.            for(j=0; j<n; j++){
12.                if(dis[i][k]!=INF && dis[k][j]!=INF && dis[i][k]+dis[k][j]<=dis[i][j]){
13.                    dis[i][j] = dis[i][k]+dis[k][j]; P[i][j] = k;
14.                }
15.            }
16.        }
17.    }
18. }
19. void printPath(int s,int d)
20. {
21.     if(P[s][d]==-1) puts("No Path!");
22.     else if(P[s][d]==s) printf("%d\n",s);
23.     else{
24.         printPath(s,P[s][d]);
25.         printPath(P[s][d],d);
26.     }
27. }
28.
29. /* Print d when the function returns */
```

Strongly Connected Component

```
1.  /*
2.  Step 1: Topsort All the nodes
3.  Step 2: Run DFS from the unvisited nodes in topsorted order.
4.          This will mark the component related to the node.
5.  */
6.
7.  vector <int> edges[MAX],trans[MAX];
8.  int compNum[MAX];
9.  bool vis[MAX];
10. int cnum;
11. stack <int> topSortedNodes;
12.
13. void topSort(int s){
14.     int i,x;
15.     vis[s] = 1;
16.     for(i=0;i<edges[s].size();i++){
17.         x = edges[s][i];
18.         if(!vis[x]) topSort(x);
19.     }
20.     topSortedNodes.push(s);
21. }
22.
23. void markComponent(int s){
24.     int i,x;
25.     vis[s] = 1;
26.     compNum[s] = cnum;
27.     for(i=0;i<trans[s].size();i++){
28.         x = trans[s][i];
29.         if(!vis[x]) markComponent(x);
30.     }
31. }
```



```
32.
33. void SCC(int n){
34.     int i,x;
35.     CLR(vis);
36.     for(int i=1;i<=n;i++){
37.         if(!vis[i]) topSort(i);
38.     }
39.     cnum = 0;
40.     CLR(vis);
41.     while(!topSortedNodes.empty()){
42.         x = topSortedNodes.top();
43.         topSortedNodes.pop();
44.         if(!vis[x]){
45.             cnum++;
46.             markComponent(x);
47.         }
48.     }
49. }
```

Articulation Point

```
1. vector <int> edges[MAX];
2. bool vis[MAX] , isArt[MAX];
3. int st[MAX] , low[MAX] , Time = 0 , n;
4.
5. void findArt(int s,int par){
6.     int i,x,child = 0;
7.     vis[s] = 1;
8.     Time++;
9.     st[s] = low[s] = Time;
10.    for(i=0;i< edges[s].size();i++){
11.        x = edges[s][i];
12.        if(!vis[x]){
13.            child++;
14.            findArt(x,s);
15.            low[s] = min(low[s],low[x]);
16.            if(par!=-1 && low[x]>=st[s]) isArt[s] = 1;
17.        }
18.        else{
19.            if(par!=x) low[s] = min(low[s],st[x]);
20.        }
21.    }
22.    if(par==-1 && child>1) isArt[s] = 1;
23. }
24.
25. void processArticulation(){
26.     Time = 0;
27.     for(int i=1;i<=n;i++) if(!vis[i]) findArt(i,-1);
28. }
```

Bridge

```
1. vector <int> ed[MAX];
2. vector <PII> res;
3. bool vis[MAX];
4. int st[MAX] , low[MAX] , Time = 0 , n;
5.
6. void findBridge(int s,int par){
7.     int i,x;
8.     vis[s] = 1;
9.     Time++;
10.    st[s] = low[s] = Time;
11.    for(i=0;i<ed[s].size();i++){
12.        x = ed[s][i];
13.        if(!vis[x]){
14.            findBridge(x,s);
15.            low[s] = min(low[s],low[x]);
16.            if(low[x]>st[s]) res.pb(mp(s,x));
17.        }
18.        else{
19.            if(par!=x) low[s] = min(low[s],st[x]);
20.        }
21.    }
22. }
23.
24. void processBridge(){
25.     Time = 0;
26.     for(int i=1;i<=n;i++) if(!vis[i]) findBridge(i,-1);
27. }
```

Bridge Tree

```
1. struct edges{
2.     int u,v;
3. }ara[EDGES];
4.
5. vector <int> ed[NODES]; // actual graph
6. vector <int> isBridge[NODES]; // if the edge is a bridge, the entry will be 1
7. vector <int> brTree[NODES]; // edges of the bridge tree
8.
9. bool vis[MAX];
10. int st[MAX] , low[MAX] , Time = 0;
11. int n , m;
12. int cnum;
13. int comp[MAX];
14.
15. void findBridge(int s,int par)
16. {
17.     int i,x,child = 0,j;
18.     vis[s] = 1;
19.     Time++;
20.     st[s] = low[s] = Time;
21.     for(i=0;i<ed[s].size();i++){
22.         x = ed[s][i];
23.         if(!vis[x]){
```

```
24.         child++;
25.         findBridge(x,s);
26.         low[s] = min(low[s],low[x]);
27.         if(low[x]>st[s]){
28.             isBridge[s][i] = 1;
29.             j = lower_bound(ed[x].begin(),ed[x].end(),s)-ed[x].begin();
30.             isBridge[x][j] = 1;
31.         }
32.     }
33.     else{
34.         if(par!=x) low[s] = min(low[s],st[x]);
35.     }
36. }
37. }
38.
39. void processBridge(){
40.     CLR(vis);
41.     Time = 0;
42.     for(int i=1;i<=n;i++) if(!vis[i]) findBridge(i,-1);
43. }
44.
45. void dfs(int s){
46.     int i,x;
47.     vis[s] = 1;
48.     comp[s] = cnum;
49.     for(i=0;i<ed[s].size();i++){
50.         if(!isBridge[s][i]){
51.             x = ed[s][i];
52.             if(!vis[x]) dfs(x);
53.         }
54.     }
55. }
56.
57. int main(){
58.     int i,u,v,ans,k;
59.     scanf("%d %d",&n,&m);
60.     for(i=1;i<=m;i++){
61.         sii(u,v);
62.
63.         ed[u].pb(v);
64.         ed[v].pb(u);
65.
66.         isBridge[u].pb(0);
67.         isBridge[v].pb(0);
68.
69.         ara[i].u = u;
70.         ara[i].v = v;
71.     }
72.     for(i=1;i<=n;i++) sort(all(ed[i]));
73.     processBridge();
74.
75.     cnum = 0;
76.     CLR(vis);
77.     for(i=1;i<=n;i++){
78.         if(!vis[i]){
79.             cnum++;
80.             dfs(i);
81.         }
82.     }
83.
84.     n = cnum; //number of nodes in the bridge tree
```

```

85.
86.     for(i=1;i<=m;i++){
87.         if(comp[ara[i].u]!=comp[ara[i].v]){
88.             brTree[comp[ara[i].u]].pb(comp[ara[i].v]);
89.             brTree[comp[ara[i].v]].pb(comp[ara[i].u]);
90.         }
91.     }
92.     return 0;
93. }

```

Biconnected Component

```

1.  /*
2.   A graph is biconnected if every node is reachable from every other node even after
   removing a single node.
3.   Algorithm of checking Biconnectivity :
4.       1) The graph is connected.
5.       2) There is no articulation point in the graph.
6.
7.   In the following code
8.   bcc_counter --> Total number of biconnected components
9.   bcc vector keeps the list of nodes in a single BCC.
10. */
11. vector <int> edges[MAX];
12. bool vis[MAX], isArt[MAX];
13. int Time;
14. int low[MAX],st[MAX];
15. vector <int> bcc[MAX];
16. int bcc_counter, n;
17. stack <int> S;
18.
19. void popBCC(int s,int x){
20.     isArt[s] = 1;
21.     bcc[bcc_counter].pb(s);
22.     while(1){
23.         bcc[bcc_counter].pb(S.top());
24.         if(S.top()==x){
25.             S.pop();
26.             break;
27.         }
28.         S.pop();
29.     }
30.     bcc_counter++;
31. }
32.
33. void findBCC(int s,int par){
34.     S.push(s);
35.     int i,x,child = 0;
36.     vis[s] = 1;
37.     Time++;
38.     st[s] = low[s] = Time;

```

```

39.     for(i=0; i< edges[s].size(); i++){
40.         x = edges[s][i];
41.         if(!vis[x]){
42.             child++;
43.             findBCC(x,s);
44.             low[s] = min(low[s],low[x]);
45.             if(par!=-1 && low[x]>=st[s]) popBCC(s,x);
46.             else if(par==-1) if(child>1) popBCC(s,x);
47.         }
48.         else if(par!=x) low[s] = min(low[s],st[x]);
49.     }
50.     if(par==-1 && child>1) isArt[s] = 1;
51. }
52.
53. void processBCC(){
54.     // might need to clear S in every case
55.     for(int i=0; i<n; i++){
56.         if(!vis[i]){
57.             Time = 0;
58.             findBCC(i,-1);
59.             bool lala = false;
60.             while(!S.empty()){
61.                 lala = true;
62.                 bcc[bcc_counter].push_back(S.top());
63.                 S.pop();
64.             }
65.             if(lala) bcc_counter++;
66.         }
67.     }
68. }

1.  /*
2.      In the following code
3.      bcc_counter --> Number of BCCs
4.      The code prints the edges of a single bcc serially
5.  */
6.  vector <int> edges[MAX];
7.  bool vis[MAX], isArt[MAX];
8.  int st[MAX], low[MAX], Time = 0;
9.  stack <PII> S;
10. int n,bcc_counter;
11.
12. void findBCC(int s,int par){
13.     int i,x,child = 0;
14.     vis[s] = 1;
15.     Time++;
16.     st[s] = low[s] = Time;
17.     for(i=0; i<edges[s].size(); i++){
18.         x = edges[s][i];
19.         if(!vis[x]){
20.             S.push(mp(s,x));
21.             child++;
22.             findBCC(x,s);
23.             low[s] = min(low[s],low[x]);
24.             if(/*par!=-1 &&*/ low[x]>=st[s]){
25.                 isArt[s] = 1;
26.                 PII cur,e = mp(s,x);
27.                 bcc_counter++;
28.                 cout << "Edges of Component " << bcc_counter << ":" << endl;
29.                 do{
30.                     cur = S.top();

```

```

31.             S.pop();
32.             cout << cur.xx << "--" << cur.yy << endl;
33.         }
34.         while(cur!=e);
35.     }
36. }
37.     else if(par!=x && st[x]<st[s]){
38.         S.push(mp(s,x));
39.         low[s] = min(low[s],st[x]);
40.     }
41. }
42. if(par==-1 && child>1) isArt[s] = 1;
43. }
44.
45. void processBCC(){
46.     // might need to clear S in every case
47.     bcc_counter = 0;
48.     for(int i=0; i<n; i++){
49.         if(!vis[i]){
50.             Time = 0;
51.             findBCC(i,-1);
52.         }
53.     }
54. }

```

Euler Trail/Circuit

Directed Graph

```

1.  /*
2.   This code finds Euler path/circuit in a DIRECTED graph
3.   1. Choose any vertex v and push it onto a stack. Initially all edges are unmarked.
4.   2. While the stack is nonempty, look at the top vertex, u, on the stack. If u has a
      n unmarked incident edge,
5.   say, to a vertex w, then push w onto the stack and mark the edge uw. On the other h
      and, if u has no unmarked
6.   incident edge, then pop u off the stack and print it.
7.   When the stack is empty, you will have printed a sequence of vertices that correspo
      nd to an Eulerian circuit/Path
8.  */
9.
10. vector <int> ed[MAX];
11. int in[MAX],out[MAX],work[MAX];
12.
13. stack <int> tp,fp;
14. // return 0 if no euler path or circuit exists
15. // return 1 if a euler trail exists
16. // return 2 if a euler circuit exists
17. int findEulerTrail(int n){
18.     int start=-1,cnt = 0,ret;
19.     for(int i=0;i<n;i++){
20.         if(in[i]==out[i]);
21.         else if(in[i]+1==out[i]){start = i;cnt++;}
22.         else if(in[i]==out[i]+1) cnt++;
23.         else return 0;
24.     }
25.     if(start==-1){

```

```

26.         ret = 2;
27.         for(int i=0;i<n;i++) if(out[i]) start = i;
28.     }
29.
30.     if(cnt>2) return 0;
31.
32.     while(!tp.empty()) tp.pop();
33.     while(!fp.empty()) fp.pop();
34.
35.     tp.push(start);
36.     int u,v,i,j;
37.     while(!tp.empty()){
38.         u = tp.top();
39.         if(work[u]==ed[u].size()){
40.             fp.push(u);
41.             tp.pop();
42.         }
43.         else{
44.             v = ed[u][work[u]];
45.             tp.push(v);
46.             work[u]++;
47.         }
48.     }
49.     // Have to check if all the edges are visited
50.     // Counting num of edges in the trail is enough
51.     return ret;
52. }
53.
54. int main(){
55.     // need to clear ed, in , out , work per case
56.     int n,m,u,v;
57.     for(i=0;i<m;i++){
58.         sii(u,v);
59.         ed[u].pb(v);
60.         in[v]++;
61.         out[u]++;
62.     }
63.     int res = findEulerTrail(n);
64.     //printing trail/circuit
65.     while(fp.empty()){
66.         printf("%d\n",S.top());
67.         S.pop();
68.     }
69.     return 0;
70. }

```

Undirected Graph

```

1.  /*
2.      This code finds Euler path/circuit in a directed graph
3.      1. Choose any vertex v and push it onto a stack. Initially all edges are unmarked.
4.      2. While the stack is nonempty, look at the top vertex, u, on the stack. If u has a
        n unmarked incident edge,
5.          say, to a vertex w, then push w onto the stack and mark the edge uw. On the other h
        and, if u has no unmarked
6.          incident edge, then pop u off the stack and print it.
7.          When the stack is empty, you will have printed a sequence of vertices that correspo
        nd to an Eulerian circuit/Path
8.  */
9.

```

```
10. vector <int> ed[MAX];
11. int deg[MAX],work[MAX];
12. stack <int> tp,fp;
13. // return 0 if no euler path or circuit exists
14. // return 1 if a euler trail exists
15. // return 2 if a euler circuit exists
16. int findEulerTrail(int n){
17.     int start=-1,cnt = 0,ret;
18.     for(int i=0;i<n;i++) if(deg[i]%2){ start = i; cnt++;}
19.     if(start==-1){
20.         ret = 2;
21.         for(int i=0;i<n;i++) if(deg[i]) start = i;
22.     }
23.     if(cnt>2) return 0;
24.
25.     while(!tp.empty()) tp.pop();
26.     while(!fp.empty()) fp.pop();
27.
28.     tp.push(start);
29.     int u,v,i,j;
30.     while(!tp.empty()){
31.         u = tp.top();
32.         if(work[u]==ed[u].size()){
33.             fp.push(u);
34.             tp.pop();
35.         }
36.         else{
37.             // next two lines are not needed for directed graph
38.             while(work[u] < ed[u].size() && ed[u][work[u]]!=-1) work[u]++;
39.             if(work[u]==ed[u].size()) continue;
40.             v = ed[u][work[u]];
41.             // next two lines are not needed for directed graph
42.             j = lower_bound(ed[v].begin(),ed[v].end(),u)-ed[v].begin();
43.             ed[v][j] = -1;
44.             tp.push(v);
45.             work[u]++;
46.         }
47.     }
48.     // Have to check if all the edges are visited
49.     // Counting num of edges in the trail is enough
50.     return ret;
51. }
52.
53. int main(){
54.     // need to clear ed, in , out , work per case
55.     int n,m,u,v;
56.     for(i=0;i<m;i++){
57.         sii(u,v);
58.         ed[u].pb(v);
59.         ed[v].pb(u);
60.         deg[u]++;deg[v]++;
61.     }
62.     for(i=0;i<n;i++) sort(all(ed[i]));
63.     int res = findEulerTrail(n);
64.     //printing trail/circuit
65.     while(fp.empty()){
66.         printf("%d\n",S.top());
67.         S.pop();
68.     }
69.     return 0;
70. }
```


Centroid Decomposition

```

1.  /*
2.      Problem : Two kinds of operations in a tree(Initially node 1 is white, others are black)
3.      Update v : Change the color of node v to white
4.      Query v : Distance of closest white node from node v(can be node v)
5.  */
6.
7.  /* Using dis[i][j] array to calculate distance between two nodes in original tree */
8.
9.  vector <int> ed[MAX]; // adjacency list of the input tree
10. bool isCentroid[MAX]; // if the node is already a centroid of some part
11. int sub[MAX]; // subtree size of a node
12. int cpar[MAX]; // parent of a node in the centroid tree
13. int clevel[MAX]; // level of a node in centroid tree
14. int dis[20][MAX]; // dis[i][j] = distance of node j from the root of the i'th level of decomposition
15.
16. void calcSubTree(int s,int par){
17.     int i,x;
18.     sub[s] = 1;
19.     for(i=0;i<ed[s].size();i++){
20.         x = ed[s][i];
21.         if(x!=par && !isCentroid[x]){
22.             calcSubTree(x,s);
23.             sub[s] += sub[x];
24.         }
25.     }
26. }
27.
28. int nn;// number of nodes in the part
29.
30. int getCentroid(int s,int par){
31.     int i,x;
32.     for(i=0;i<ed[s].size();i++){
33.         x = ed[s][i];
34.         if(!isCentroid[x] && x!=par && sub[x]>(nn/2)) return getCentroid(x,s);
35.     }
36.     return s;
37. }
38.
39. void setDis(int s, int from, int par, int l) {
40.     dis[from][s] = l;
41.     int i,x;
42.     for(i=0;i<ed[s].size();i++) {
43.         x = ed[s][i];
44.         if(x!=par && !isCentroid[x]) {
45.             setDis(x, from, s, l+1);
46.         }
47.     }
48. }
49.
50. //complexity → O(nlog(n))
51. void decompose(int s,int p,int l){
52.     calcSubTree(s,p);
53.     nn = sub[s];
54.     int c = getCentroid(s,p);

```

```

55.
56. // setDis(c,...) calculates the distance of every node from c which
57. // are in the subtree of c in centroid tree
58. setDis(c,l,p,0);
59.
60. isCentroid[c] = true;
61. cpar[c] = p;
62. clevel[c] = l;
63.
64. int i,x;
65. for(i=0;i<ed[c].size();i++){
66.     x = ed[c][i];
67.     if(!isCentroid[x]) decompose(x,c,l+1);
68. }
69. }
70.
71. int ans[MAX];
72.
73. inline void update(int v) {
74.     int u = v;
75.     while(u!=-1) {
76.         ans[u] = min(ans[u], dis[clevel[u]][v]);
77.         u = cpar[u];
78.     }
79. }
80.
81. inline int query(int v) {
82.     int ret = INF;
83.     int u = v;
84.     while(u != -1) {
85.         ret = min(ret, dis[clevel[u]][v]+ans[u]);
86.         u = cpar[u];
87.     }
88.     return ret;
89. }
90.
91.
92. int main(){
93.     decompose(1,-1,0);
94.     for(i=1;i<=n;i++) ans[i] = INF;
95.     update(v);
96.     query(v));
97.     return 0;
98. }

```

```

1. /*Using Spares Table and LCA to find distance between two nodes in the original tree*/
2.
3. vector <int> ed[MAX]; // adjacency list of the input tree
4. bool isCentroid[MAX]; // if the node is already a centroid of some part
5. int sub[MAX]; // subtree size of a node
6. int cpar[MAX]; // parent of a node in the centroid tree
7. int L[MAX]; // Depth of a node in the original tree
8. int P[MAX][20]; // P[i][j] denotes (2^j)th parent of node i in the original tree
9.
10. /*****LCA*****/
11.
12. int lg;

```

```

13.
14. void dfs(int s,int par,int l){
15.     int i,x;
16.     L[s] = 1;
17.     for(i=0; i<ed[s].size(); i++){
18.         x = ed[s][i];
19.         if(x!=par){
20.             P[x][0] = s;
21.             dfs(x,s,l+1);
22.         }
23.     }
24. }
25.
26. void lca_build(int n){
27.     dfs(1,-1,0);
28.     lg = (log(n)/log(2.0))+2;
29.     int i,j;
30.     for(j=1; (1<<j)<=n; j++){
31.         for(i=1; i<=n; i++)
32.             if(P[i][j-1]!=-1) P[i][j] = P[P[i][j-1]][j-1];
33.     }
34.
35. int lca_query(int x,int y){
36.     if(L[x]<L[y]) swap(x,y);
37.     int i,j;
38.     for(i=lg; i>=0; i--){
39.         if(L[x] - (1<<i) >= L[y]) x = P[x][i];
40.
41.         if(x==y) return x;
42.         for(i=lg; i>=0; i--){
43.             if(P[x][i]!=-1 && P[x][i]!=P[y][i]){
44.                 x = P[x][i];
45.                 y = P[y][i];
46.             }
47.         }
48.         return P[x][0];
49.     }
50.
51. // returns distance between node u and node v in the original tree
52. int dist(int u,int v){
53.     return L[u] + L[v] - 2*L[lca_query(u,v)];
54. }
55.
56. /*****Decomposition*****/
57.
58. void calcSubTree(int s,int par){
59.     int i,x;
60.     sub[s] = 1;
61.     for(i=0;i<ed[s].size();i++){
62.         x = ed[s][i];
63.         if(x!=par && !isCentroid[x]){
64.             calcSubTree(x,s);
65.             sub[s] += sub[x];
66.         }
67.     }
68. }
69.
70. int nn;// number of nodes in the part
71.
72. int getCentroid(int s,int par){
73.     int i,x;

```

```
74.     for(i=0;i<ed[s].size();i++){
75.         x = ed[s][i];
76.         if(!isCentroid[x] && x!=par && sub[x]>(nn/2)) return getCentroid(x,s);
77.     }
78.     return s;
79. }
80.
81. //complexity --> O(n*log(n))
82. void decompose(int s,int p){
83.     calcSubTree(s,p);
84.     nn = sub[s];
85.     int c = getCentroid(s,p);
86.     isCentroid[c] = true;
87.     cpar[c] = p;
88.     int i,x;
89.     for(i=0;i<ed[c].size();i++){
90.         x = ed[c][i];
91.         if(!isCentroid[x]) decompose(x,c);
92.     }
93. }

1.  /*****Operation*****/
2.  int ans[MAX];
3.
4.  void update(int v){
5.      int u = v;
6.      while(u!=-1){
7.          ans[u] = min(ans[u],dist(u,v));
8.          u = cpar[u];
9.      }
10. }
11.
12. int query(int v){
13.     int u = v,ret = INF;
14.     while(u!=-1){
15.         ret = min(ret,ans[u]+dist(u,v));
16.         u = cpar[u];
17.     }
18.     return ret;
19. }
20.
21. int main(){
22.     decompose(1,-1);
23.     lca_build(n);
24.     for(i=1;i<=n;i++) ans[i] = INF;
25.     update(v);
26.     query(v);
27.     return 0;
28. }
```

Maximum Flow

Edmonds Karp

```
1. //Edmonds Carp Algorithm
2. //Finds Max Flow using ford fulkerson method
3. //Finds path from source to sink using bfs
4. //Complexity V*E*E
5.
6. vector <int> ed[MAX];
7. int cap[MAX][MAX];
8. int par[MAX]; //keeps track of the parent in a path from s to d
9. int mCap[MAX]; //mCap[i] keeps track edge that have minimum cost on the shortest path from s to i
10.
11. bool getPath(int s,int d,int n){
12.     for(int i=0; i<=n; i++) mCap[i] = INF;
13.     SET(par);
14.     queue <int> q;
15.     q.push(s);
16.     while(!q.empty()){
17.         int u = q.front();
18.         q.pop();
19.         for(int i=0; i<ed[u].size(); i++){
20.             if(cap[u][ed[u][i]]!=0 && par[ed[u][i]]==-1){
21.                 par[ed[u][i]] = u;
22.                 mCap[ed[u][i]] = min(mCap[u],cap[u][ed[u][i]]);
23.                 if(ed[u][i]==d) return true;
24.                 q.push(ed[u][i]);
25.             }
26.         }
27.     }
28.     return false;
29. }
30.
31. int getFlow(int s,int d,int n){
32.     int F = 0;
33.     while(getPath(s,d,n)){
34.         int f = mCap[d];
35.         F += f;
36.         int u = d;
37.         while(u!=s){
38.             int v = par[u];
39.             cap[u][v] += f;
40.             cap[v][u] -= f;
41.             u = v;
42.         }
43.     }
44.     return F;
45. }
46.
47. int main(){
48.     int maxFlow = getFlow(s,d,n);
49.     return 0;
50. }
```

Dinic

```

1. // Complexity V*V*E
2. #define MAXN 5010
3. int src,snk;
4. int dist[MAXN],work[MAXN];
5.
6. struct Edge{
7.     int to, rev_pos, c, f;
8. };
9.
10. vector <Edge> ed[MAXN];
11.
12. int M[MAXN][MAXN]; //Don't forget to SET the array
13.
14. /* if there are multiple edges between same pair of nodes
15.    v might appear in the edge list of u multiple times */
16. void addEdge(int u,int v,int c) {
17.     Edge a = {v,ed[v].size(),c,0};
18.     Edge b = {u,ed[u].size(),0,0}; // cap 0 should be replaced by c for directed graphs
19.     ed[u].pb(a);
20.     ed[v].pb(b);
21. }
22.
23. /* Makes sure u will appear in the node list of u only once
24.    but takes extra log(edges) time */
25. void addEdge(int u,int v,int c) {
26.     if(M.find(mp(u,v))==M.end()){
27.         M[mp(u,v)] = ed[u].size();
28.         M[mp(v,u)] = ed[v].size();
29.         Edge a = {v,ed[v].size(),c,0};
30.         Edge b = {u,ed[u].size(),c,0}; //c should be replaced by 0 for undirected graph
31.         ed[u].pb(a);
32.         ed[v].pb(b);
33.     }
34.     else{
35.         int x = M[mp(u,v)];
36.         int y = M[mp(v,u)];
37.         ed[u][x].c += c;
38.         ed[v][y].c += c; // If the graph is directed, comment it out
39.     }
40. }
41.
42. bool dinic_bfs(){
43.     SET(dist);
44.     dist[src] = 0;
45.     queue <int> q;
46.     q.push(src);
47.     while(!q.empty()){
48.         int u = q.front();
49.         q.pop();
50.         for(int i=0; i<ed[u].size(); i++){
51.             Edge &e = ed[u][i];
52.             int v = e.to;
53.             if(dist[v]==-1 && e.f<e.c){
54.                 dist[v] = dist[u]+1;
55.                 q.push(v);
56.             }
57.         }

```

```
58.     }
59.     return (dist[snk]>=0);
60. }
61.
62. int dinic_dfs(int u, int fl){
63.     if (u == snk) return fl;
64.     for (; work[u] < ed[u].size(); work[u]++){
65.         Edge &e = ed[u][work[u]];
66.         if (e.c <= e.f) continue;
67.         int v = e.to;
68.         if (dist[v] == dist[u] + 1){
69.             int df = dinic_dfs(v, min(fl, e.c - e.f));
70.             if (df > 0){
71.                 e.f += df;
72.                 ed[v][e.rev_pos].f -= df;
73.                 return df;
74.             }
75.         }
76.     }
77.     return 0;
78. }
79.
80.
81. int maxFlow(int _src, int _snk){
82.     src = _src;
83.     snk = _snk;
84.     int result = 0;
85.     while (dinic_bfs()){
86.         CLR(work);
87.         while (int delta = dinic_dfs(src, INF)) result += delta;
88.     }
89.     return result;
90. }
```

Minimum Cost Maximum Flow

```
1. // Maximizes the flow first, then minimizes the cost
2.
3. // Complexity -->
4.
5. #define MAXN 105 // maximum number of nodes
6.
7. int N;
8. int dis[MAXN] , par[MAX] , mCap[MAX] , pos[MAX];
9. bool vis[MAXN];
10.
11. struct Edge{
12.     int to, rev_pos, cap, cost, flow;
13. };
14.
15. vector <Edge> ed[MAXN];
16.
17. void addEdge(int u,int v,int cap,int cost){
18.     Edge a = {v,ed[v].size(),cap,cost,0};
19.     Edge b = {u,ed[u].size(),0,-cost,0};
20.     ed[u].pb(a);
21.     ed[v].pb(b);
22. }
```

```

23. // 1 based indexing
24. /*
25.     SPFA Technique :
26.     Insert a node in the queue if we get a better distance for
27.     the node and the node is not already in the queue
28.     A combination of Dijkstra and BFS
29. */
30.
31. bool SPFA(){
32.     int i,u,v;
33.     CLR(vis);
34.     for(i=0;i<=N;i++) mCap[i] = dis[i] = INF;
35.     queue<int> q;
36.     dis[src] = 0;
37.     vis[src] = true; // src is in the queue now
38.     q.push(src);
39.
40.     while(!q.empty()){
41.         u = q.front();
42.         q.pop();
43.         vis[u] = false; // u is not in the queue now
44.         for(i=0;i<ed[u].size();i++){
45.             Edge e = ed[u][i];
46.             v = e.to;
47.             if(e.cap>e.flow && dis[v]>dis[u]+e.cost){
48.                 dis[v] = dis[u]+e.cost;
49.                 par[v] = u;
50.                 pos[v] = i;
51.                 mCap[v] = min(mCap[u],e.cap-e.flow);
52.                 if(!vis[v]){
53.                     vis[v] = true;
54.                     q.push(v);
55.                 }
56.             }
57.         }
58.     }
59.     return (dis[snk]!=INF);
60. }
61.
62. pair<int,int> MCMF() {
63.     int F = 0, C = 0, f;
64.     while(SPFA()){
65.         int u = snk;
66.         f = mCap[u];
67.         F += f;
68.         while(u!=src){
69.             int v = par[u];
70.             ed[v][pos[u]].flow += f; // edge of v-->u increases
71.             ed[u][ed[v][pos[u]].rev_pos].flow -= f;
72.             u = v;
73.         }
74.         C += dis[snk] * f;
75.     }
76.     return mp(F,C);
77. }
78.
79. int main(){
80.     //dont forget to initialize src and snk
81.     MCMF();
82.     return 0;
83. }

```


Maximum Bipartite Matching

```

1. // worst case complexity V*E
2.
3. #define L 105
4. #define R 105
5.
6. vector<int> G[L]; //The adjacency list for the nodes in the left side
7. int matchL[L], matchR[R];
8. bool vis[L];
9.
10. bool dfs(int s)
11. {
12.     int i, x;
13.     vis[s] = true;
14.     for(i = 0; i < G[s].size(); i++){
15.         x = G[s][i];
16.         if(matchR[x] == -1 || (!vis[matchR[x]] && dfs(matchR[x]))){
17.             matchL[s] = x;
18.             matchR[x] = s;
19.             return 1;
20.         }
21.     }
22.     return 0;
23. }
24.
25. // n = number of nodes in the left side
26. int match(int n) {
27.     int i;
28.     bool done;
29.     SET(matchL); SET(matchR);
30.     while(true){
31.         CLR(vis);
32.         done = true;
33.
34.         for(i=1; i<=n; i++){
35.             if(matchL[i] == -1 && !vis[i] && dfs(i)) done = false;
36.
37.             if(done) break;
38.         }
39.
40.         int cnt = 0; // number of matches
41.         for(i=1; i<=n; i++){
42.             if(matchL[i] != -1) cnt++;
43.
44.             return cnt;
45. }

```

A minimum vertex cover of a graph is a minimum set of nodes such that each edge of the graph has at least one endpoint in the set.

Minimum Vertex Cover = Maximum Matching

Maximum independent set of a graph is a maximum set of nodes such that no two nodes of the set are connected by an edge.

Maximum Independent Set = Total Nodes – Minimum Vertex Cover

Finding Minimum Vertex Cover :

Run BPM. After that do a dfs from every **unmatched** node in the left side. Go through the edge u to v if and only if :

- u is on the left side and $\text{matchL}[u] \neq v$ or
- u is on the right side and $\text{matchR}[u] == v$

After this, all visited nodes in the left side and the unvisited nodes in the right side will form a maximum independent set. But why? (See the proof)

The rest of the nodes will form MVC then.

Proof :

Size of Maximum independent set = Total nodes - Maximum Matching

In our algorithm, It's easy to observe that the set will be an independent set. And it's size will be equal to **Total nodes - Maximum Matching**. Because, we are keeping exactly one node of every matched edge in the set. EASY!! :P

Dilworth's theorem :

An anti chain is a set of nodes of a graph such that there is no path from any node to another node using the edges of the graph. Dilworth's theorem states that in a directed acyclic graph, the size of a minimum **general** path cover equals the size of a maximum anti chain.

Segment Tree

Point Update, Range Query

```
1. int ara[MAX];
2.
3. struct node{
4.     int sum;
5. }tree[4*MAX];
6.
7. node Merge(node a,node b){
8.     node ret;
9.     ret.sum = a.sum+b.sum;
10.    return ret;
11. }
12.
13. void build(int n,int st,int ed){
14.     if(st==ed){
15.         tree[n].sum = ara[st];
16.         return;
17.     }
18.     int mid = (st+ed)/2;
19.     build(2*n,st,mid);
20.     build(2*n+1,mid+1,ed);
21.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
22. }
23.
24. void update(int n,int st,int ed,int id,int v){
25.     if(id>ed || id<st) return;
26.     if(st==ed && ed==id){
27.         tree[n].sum = v;
28.         return;
29.     }
30.     int mid = (st+ed)/2;
31.     update(2*n,st,mid,id,v);
32.     update(2*n+1,mid+1,ed,id,v);
33.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
34. }
35.
36. node query(int n,int st,int ed,int i,int j){
37.     if(st>=i && ed<=j) return tree[n];
38.     int mid = (st+ed)/2;
39.     if(mid<i) return query(2*n+1,mid+1,ed,i,j);
40.     else if(mid>=j) return query(2*n,st,mid,i,j);
41.     else return Merge(query(2*n,st,mid,i,j),query(2*n+1,mid+1,ed,i,j));
42. }
```

Range Update, Range Query

```
1. int ara[MAX];
2.
3. struct node{
4.     int sum;
5. }tree[4*MAX];
6.
7. int lazy[4*MAX];
```

```

8.
9. node Merge(node a,node b){
10.     node ret;
11.     ret.sum = a.sum+b.sum;
12.     return ret;
13. }
14.
15. void lazyUpdate(int n,int st,int ed){
16.     if(lazy[n]!=0){
17.         tree[n].sum += ((ed-st+1)*lazy[n]);
18.         if(st!=ed){
19.             lazy[2*n] += lazy[n];
20.             lazy[2*n+1] += lazy[n];
21.         }
22.         lazy[n] = 0;
23.     }
24. }
25.
26. void build(int n,int st,int ed){
27.     lazy[n] = 0;
28.     if(st==ed){
29.         tree[n].sum = ara[st];
30.         return;
31.     }
32.     int mid = (st+ed)/2;
33.     build(2*n,st,mid);
34.     build(2*n+1,mid+1,ed);
35.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
36. }
37. void update(int n,int st,int ed,int i,int j,int v){
38.     lazyUpdate(n,st,ed);
39.     if(st>j || ed<i) return;
40.     if(st>=i && ed<=j){
41.         lazy[n] += v;
42.         lazyUpdate(n,st,ed);
43.         return;
44.     }
45.     int mid = (st+ed)/2;
46.     update(2*n,st,mid,i,j,v);
47.     update(2*n+1,mid+1,ed,i,j,v);
48.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
49. }
50.
51. node query(int n,int st,int ed,int i,int j){
52.     lazyUpdate(n,st,ed);
53.     if(st>=i && ed<=j) return tree[n];
54.     int mid = (st+ed)/2;
55.     if(mid<i) return query(2*n+1,mid+1,ed,i,j);
56.     else if(mid>=j) return query(2*n,st,mid,i,j);
57.     else return Merge(query(2*n,st,mid,i,j),query(2*n+1,mid+1,ed,i,j));
58. }

```

How Many Non Zero Elements in the Tree

```

1. /*
2.     Given an array consisting of all zeroes
3.     Update -> Add some value to all the elements of a range
4.             (no element ever gets negative value)
5.     Query -> How many non zero element in a range
6. */

```

```
7.
8. int tree[MAX];    /// how many non zero elements in this segment
9. int lazy[MAX];    /// how many times a node is fully updated
10.
11. void lazyUpdate(int n,int st,int ed){
12.     if(st!=ed){
13.         if(lazy[n]) tree[n] = ed-st+1;
14.         else tree[n] = tree[2*n]+tree[2*n+1];
15.     }
16.     else{
17.         if(lazy[n]) tree[n] = ed-st+1;
18.         else tree[n] = 0;
19.     }
20. }
21.
22. void build(int n,int st,int ed){
23.     lazy[n] = tree[n] = 0;
24.     if(st==ed) return;
25.     int mid = (st+ed)/2;
26.     build(2*n,st,mid);
27.     build(2*n+1,mid+1,ed);
28. }
29.
30. void update(int n,int st,int ed,int i,int j,int v){
31.     if(st>j || ed<i) return;
32.     if(st>=i && ed<=j){
33.         lazy[n] += v;
34.         lazyUpdate(n,st,ed);
35.         return;
36.     }
37.     int mid = (st+ed)/2;
38.     update(2*n,st,mid,i,j,v);
39.     update(2*n+1,mid+1,ed,i,j,v);
40.     lazyUpdate(n,st,ed);
41. }
42.
43. int query(int n,int st,int ed,int i,int j){
44.     if(st>=i && ed<=j) return tree[n];
45.     int mid = (st+ed)/2;
46.     if(mid<i) return query(2*n+1,mid+1,ed,i,j);
47.     else if(mid>=j) return query(2*n,st,mid,i,j);
48.     else return query(2*n,st,mid,i,j) + query(2*n+1,mid+1,ed,i,j);
49. }
```

Implicit Segment Tree

Point Update, Range Query

```
1. struct node{
2.     int sum;
3.     node *left,*right;
4.     node(){ }
5.     node(int value){
6.         sum = value;
7.         left = right = NULL;
8.     }
9. };
```

```
10.
11. void update(node *cur,int st,int ed,int id,int v)
12. {
13.     if(id<st || id>ed) return;
14.     if(id==st && id==ed){
15.         cur->sum = v;
16.         return;
17.     }
18.     int mid = (st+ed)/2;
19.     if(cur->left==NULL) cur->left = new node(0);
20.     if(cur->right==NULL) cur->right = new node(0);
21.     update(cur->left,st,mid,id,v);
22.     update(cur->right,mid+1,ed,id,v);
23.     cur->sum = cur->left->sum + cur->right->sum;
24. }
25.
26. int query(node *cur,int st,int ed,int i,int j)
27. {
28.     if(st>=i && ed<=j) return cur->sum;
29.     int mid = (st+ed)/2;
30.     if(cur->left==NULL) cur->left = new node(0);
31.     if(cur->right==NULL) cur->right = new node(0);
32.     if(mid<i) return query(cur->right,mid+1,ed,i,j);
33.     else if(mid>=j) return query(cur->left,st,mid,i,j);
34.     else return query(cur->right,mid+1,ed,i,j)+query(cur->left,st,mid,i,j);
35. }
36.
37. int main()
38. {
39.     int n = 1000000000;
40.     node *root = new node(0);
41.     update(root,1,n,5,1);
42.     update(root,1,n,3,1);
43.     cout << query(root,1,n,1,5) << endl;
44.     return 0;
45. }
```

Range Update, Range Query

```
1. struct node{
2.     int sum,lazy;
3.     node *left,*right;
4.     node(){}
5.     node(int value){
6.         sum = value;
7.         lazy = 0;
8.         left = right = NULL;
9.     }
10. };
11.
12. void lazyUpdate(node *cur,int st,int ed)
13. {
14.     if(cur->lazy!=0){
15.         cur->sum += ((ed-st+1)*cur->lazy);
16.         if(st!=ed){
17.             if(cur->left==NULL) cur->left = new node(0);
18.             if(cur->right==NULL) cur->right = new node(0);
19.             cur->left->lazy += cur->lazy;
20.             cur->right->lazy += cur->lazy;
21.         }

```

```
22.         cur->lazy = 0;
23.     }
24. }
25.
26. void update(node *cur,int st,int ed,int i,int j,int v){
27.     lazyUpdate(cur,st,ed);
28.     if(st>j || ed<i) return;
29.     if(st>=i && ed<=j){
30.         cur->lazy += v;
31.         lazyUpdate(cur,st,ed);
32.         return;
33.     }
34.     int mid = (st+ed)/2;
35.     if(cur->left==NULL) cur->left = new node(0);
36.     if(cur->right==NULL) cur->right = new node(0);
37.     update(cur->left,st,mid,i,j,v);
38.     update(cur->right,mid+1,ed,i,j,v);
39.     cur->sum = cur->left->sum + cur->right->sum;
40. }
41.
42. int query(node *cur,int st,int ed,int i,int j){
43.     lazyUpdate(cur,st,ed);
44.     if(st>=i && ed<=j) return cur->sum;
45.     int mid = (st+ed)/2;
46.     if(cur->left==NULL) cur->left = new node(0);
47.     if(cur->right==NULL) cur->right = new node(0);
48.     if(mid<i) return query(cur->right,mid+1,ed,i,j);
49.     else if(mid>=j) return query(cur->left,st,mid,i,j);
50.     else return query(cur->right,mid+1,ed,i,j)+query(cur->left,st,mid,i,j);
51. }
52.
53. int main()
54. {
55.     int n = 1000000000;
56.     node *root = new node(0);
57.     update(root,1,n,1,5,1);
58.     update(root,1,n,4,10,1);
59.     update(root,1,n,9,14,1);
60.     cout << query(root,1,n,1,20) << endl;
61.     return 0;
62. }
```

Binary Indexed Tree

```

1.  /*
2.     Initially the tree array is set to zero
3.     Point Update(Adding v to index p)
4.     Query returns sum of the range [1,p]
5.  */
6.
7.  int tree[10];
8.
9.  //n --> size of the array
10. //v --> value to be added to index idx
11. void update(int n,int p,int v){
12.     while( p<=n ) tree[p] += v , p += p & (-p);
13. }
14.
15. //returns sum of range[1,p]
16. int query(int p){
17.     int sum = 0;
18.     while(p > 0) sum += tree[p], p -= p & (-p);
19.     return sum;
20. }
21.
22. //returns sum of range[l,r]
23. int range_query(int l,int r){
24.     if(l>r) return 0;
25.     return query(r)-query(l-1);
26. }

```

Mo's Algorithm

```

1.  /*
2.     Better to keep Query array 0 based
3.  */
4.
5.  #define MAX_SZ 100010
6.  #define MAX_VAL 100010
7.
8.  int bs;//block size
9.  int ara[MAX];
10. int cnt[MAX_VAL];
11. int res[SZ];
12. int ans;
13.
14. struct data{
15.     int l,r,id,bn; //bn --> block number
16. } quer[MAX_SZ];
17.
18. bool cmp(data a,data b){
19.     if(a.seg==b.seg) return a.r<b.r;
20.     return a.seg<b.seg;
21. }
22.
23. void Add(int id){
24.     cnt[ara[id]]++;

```



```
25.    ///update ans
26. }
27.
28. void Remove(int id){
29.     cnt[ara[id]]--;
30.     ///update ans
31. }
32.
33. void Mo(int q){
34.     int L = 0, R = 0,l,r;
35.     ans = 0;
36.     Add(0);
37.     for(int i=0; i<q; i++) {
38.         l = quer[i].l;
39.         r = quer[i].r;
40.
41.         while(L>l) Add(--L);
42.         while(L<l) Remove(L++);
43.         while(R>r) Remove(R--);
44.         while(R<r) Add(++R);
45.
46.         res[quer[i].id] = ans;
47.     }
48. }
49.
50. int main()
51. {
52.     int q;
53.     sort(quer,quer+q,cmp);
54.     Mo(q);
55.     return 0;
56. }
```

Lowest Common Ancestor

```
1. int lg;
2.
3. int L[MAX]; // Depth of a node
4. int P[MAX][20]; // P[i][j] denotes (2^j)th parent of node i
5. vector<int> ed[MAX];
6.
7. void dfs(int s,int par,int l){
8.     int i,x;
9.     L[s] = l;
10.    for(i=0; i<ed[s].size(); i++){
11.        x = ed[s][i];
12.        if(x!=par){
13.            P[x][0] = s;
14.            dfs(x,s,l+1);
15.        }
16.    }
17. }
18.
19. void lca_build(int n) {
20.     lg = (log(n)/log(2.0))+2;
21.     int i,j;
22.     for(j=1; (1<<j)<=n; j++)
23.         for(i=1; i<=n; i++)
```

```

24.         if(P[i][j-1]!=-1) P[i][j] = P[P[i][j-1]][j-1];
25.     }
26.
27. int lca_query(int x,int y){
28.     if(L[x]<L[y]) swap(x,y);
29.     int i,j;
30.     for(i=lg; i>=0; i--){
31.         if(L[x] - (1<<i) >= L[y]) x = P[x][i];
32.
33.         if(x==y) return x;
34.         for(i=lg; i>=0; i--){
35.             if(P[x][i]!=-1 && P[x][i]!=P[y][i]){
36.                 x = P[x][i];
37.                 y = P[y][i];
38.             }
39.         }
40.         return P[x][0];
41.     }
42.
43. int main(){
44.     SET(P);
45.     dfs(1,-1,0); // nodes start from 1
46.     lca_build(100); // total nodes = n
47. }

```

Trie

Array Implementation

```

1. #define MAX      200000
2. #define SZ       26
3.
4. int root,now;
5. int nxt[MAX][SZ] , cnt[MAX][SZ];
6.
7. void Clear(){
8.     root = now = 1;
9.     CLR(nxt),CLR(cnt);
10. }
11.
12. int scale(char ch) { return (ch - 'a');}
13.
14. void Insertu(string s){
15.     int cur = root, to;
16.     for(int i=0;i<s.size();i++){
17.         to = scale(s[i]) ;
18.         if( nxt[cur][to]==0) nxt[cur][to] = ++now;
19.         cnt[cur][to]++;
20.         cur = nxt[cur][to];
21.     }
22. }
23.
24. bool Find(string s){
25.     int cur = root, to;
26.     for(int i=0;i<s.size();i++){
27.         to = scale(s[i]) ;
28.         if( !nxt[cur][to] || !cnt[cur][to]) return false;

```

```

29.         cur = nxt[cur][to];
30.     }
31.     return true;
32. }
33.
34. /// If a string doesn't exist in the trie, Delete function for that string won't work
35. void Delete(string s){
36.     int cur = root, to;
37.     for(int i=0;i<s.size();i++){
38.         to = scale(s[i]) ;
39.         cnt[cur][to]--;
40.         cur = nxt[cur][to];
41.     }
42. }

```

Heavy Light Decomposition

```

1.  /*Code of Lightoj-1348 : Aladdin and the Return Journey*/
2.
3.  int n;
4.  int chainNo , chainHead[MAX] , chainId[MAX];
5.  int it , baseArray[MAX] , posInBaseArray[MAX] ;
6.  int subtreeSz[MAX];
7.  int ara[MAX];
8.  vector <int> ed[MAX];
9.
10. int lg;
11.
12. int L[MAX]; // Depth/level of a node
13. int P[MAX][20]; // P[i][j] denotes (2^j)th parent of node i
14.
15. void dfs(int s,int par){
16.     int i,x;
17.     subtreeSz[s] = 1;
18.     for(i=0;i<ed[s].size();i++){
19.         x = ed[s][i];
20.         if(x!=par){
21.             P[x][0] = s;
22.             L[x] = L[s]+1;
23.             dfs(x,s);
24.             subtreeSz[s] += subtreeSz[x];
25.         }
26.     }
27. }
28.
29. /*Exact code of segment tree here(lazy propagation added if needed)*/
30.
31. /*HLD starts*/
32. void HLD(int s,int p){
33.     it++;
34.     posInBaseArray[s] = it;
35.     baseArray[it] = ara[s];
36.     if(chainHead[chainNo]==-1) chainHead[chainNo] = s;
37.     chainId[s] = chainNo;
38.
39.     int heavyChild = -1, heavyChildSz = 0;
40.     int i,x;
41.     for(i=0;i<ed[s].size();i++){

```

```

42.     x = ed[s][i];
43.     if(x!=p && subtreeSz[x]>heavyChildSz){
44.         heavyChildSz = subtreeSz[x];
45.         heavyChild = x;
46.     }
47. }
48. if(heavyChild!=-1) HLD(heavyChild,s);
49. for(i=0;i<ed[s].size();i++){
50.     x = ed[s][i];
51.     if(x!=p && x!=heavyChild){
52.         chainNo++; HLD(x,s);
53.     }
54. }
55. }
56.
57. void HLDConstruct(){
58.     it = 0;
59.     chainNo = 1;
60.     SET(chainHead);
61.
62.     // for calculating sub tree size
63.     // also saves the level of every node for LCA function
64.     SET(P);
65.     L[1] = 0;
66.     dfs(1,-1);
67.     HLD(1,-1);
68.     build(1,1,n);
69. }
70. /*HLD ends*/

```

```

1.  /*LCA starts*/
2.
3.  void lca_build(){
4.      lg = (log(n)/log(2))+1;
5.      int i,j;
6.      //P is set to -1 in HLDConstruct
7.      //P[i][0] is updated in dfs
8.      for(j=1; (1<<j)<=n; j++){
9.          for(i=1; i<=n; i++){
10.             if(P[i][j-1]!=-1) P[i][j] = P[P[i][j-1]][j-1];
11.         }
12.     }
13. int lca_query(int x,int y){
14.     if(L[x]<L[y]) swap(x,y);
15.     int i,j;
16.     for(i=lg; i>=0; i--){
17.         if(L[x] - (1<<i) >= L[y]) x = P[x][i];
18.     }
19.     if(x==y) return x;
20.
21.     for(i=lg; i>=0; i--){
22.         if(P[x][i]!=-1 && P[x][i]!=P[y][i]){
23.             x = P[x][i]; y = P[y][i];
24.         }
25.     }
26.     return P[x][0];
27. }
28. /*LCA ends*/

```

```

29.
30. // path from u to v, L[u] >= L[v]
31. int call(int u,int v)
32. {
33.     int ret = 0,l,r,head;
34.     while(true){
35.         l = posInBaseArray[v];
36.         if(chainId[u]!=chainId[v]){
37.             head = chainHead[chainId[u]];
38.             l = posInBaseArray[head];
39.         }
40.         r = posInBaseArray[u];
41.         if(l<=r) ret += query(1,1,n,l,r).sum;
42.         if(chainId[u]==chainId[v]) return ret;
43.         u = P[head][0];
44.     }
45. }
46.
47. int getResult(int u,int v){
48.     int lca = lca_query(u,v);
49.     int ret = 0;
50.     return ret = call(u,lca) + call(v,lca) - baseArray[posInBaseArray[lca]];
51. }
52.
53. void updateNode(int id, int v){
54.     id = posInBaseArray[id];
55.     update(1,1,n,id,v);
56. }
57.
58. int main()    {
59.     HLDConstruct();
60.     lca_build();
61.     return 0;
62. }

```

Treap

Normal Treap

```

1. struct node{
2.     int prior; // Heap value
3.     int val; // BST value
4.     int sz; // Subtree Size
5.     int sum; // This bst maintains the sum of it's child nodes
6.     struct node *l,*r,*p;
7. };
8.
9. typedef node* pnode;
10.
11. pnode Treap;
12.
13. int get_sz(pnode t) { return t?t->sz:0; }
14. int get_sum(pnode t) { return t?t->sum:0; }
15.
16. void update(pnode t){
17.     if(!t) return;
18.     if(t->l) t->l->p = t;

```

```
19.     if(t->r) t->r->p = t;
20.     t->sz = get_sz(t->l) + 1 + get_sz(t->r);
21.     t->sum = get_sum(t->l) + 1 + get_sum(t->r);
22. }
23.
24. pnode init(int val){
25.     pnode ret = (pnode)malloc(sizeof(node));
26.     ret->val = val;
27.     ret->sz = 1;
28.     ret->prior=rand();
29.     ret->p = ret->l = ret->r = NULL;
30.     return ret;
31. }
32.
33. // l contains the nodes having BST value less than val, r contains the rest
34. void split(pnode t,pnode &l,pnode &r,int val){
35.     if(!t) l = r = NULL;
36.     else if(t->val<=val) split(t->r,t->r,r,val) , l = t ;
37.     else split(t->l,l,t->l,val) , r = t ;
38.     update(t);
39. }
40.
41. void Merge(pnode &t,pnode l,pnode r){
42.     if(!l || !r) t = l ? l : r;
43.     else if(l->prior > r->prior) Merge(l->r,l->r,r), t = l ;
44.     else Merge(r->l,l,r->l), t = r ;
45.     update(t);
46. }
47.
48. // Inserting a new node into BST
49. void Insert(pnode &t,pnode it){
50.     if(!t) t = it ;
51.     else if(it->prior>t->prior) split(t,it->l,it->r,it->val) , t = it ;
52.     else if(t->val<=it->val) Insert(t->r,it);
53.     else Insert(t->l,it);
54.     update(t);
55. }
56. //or
57. void Insert(pnode &t,pnode it){
58.     if(!t) t = it;
59.     pnode l,r;
60.     split(t,l,r,it->val);
61.     Merge(t,l,it);
62.     Merge(t,t,r);
63. }
64.
65. // Removing a node having BST value = val
66. void Remove(pnode &t,int val){
67.     if(!t)return;
68.     else if(t->val==val){
69.         pnode temp=t;
70.         Merge(t,t->l,t->r);
71.         free(temp);
72.     }
73.     else if(t->val<val) Remove(t->r,val);
74.     else Remove(t->l,val);
75.     update(t);
76. }
77.
78. void Delete(pnode &t){
79.     if(!t) return;
```

```

80.     if(t->l) Delete(t->l);
81.     if(t->r) Delete(t->r);
82.     delete(t);
83.     t = NULL;
84. }

```

Implicit Treap

```

1.  /**Treap as Interval Tree(1 based) With Insert and Remove Operation**/
2.
3.  typedef struct node{
4.      int prior,sz;
5.      int val; //value stored in the array
6.      int sum; //whatever info you want to maintain in segtree for each node
7.      int lazy; //whatever lazy update you want to do
8.      struct node *l,*r,*p;
9.  } node;
10.
11. typedef node* pnode;
12. pnode Treap;
13. int get_sz(pnode t){ return t?t->sz:0; }
14. int get_sum(pnode t){ return t?t->sum:0; }
15.
16. void lazyUpdate(pnode t){
17.     if(!t || !t->lazy)return;
18.     t->val += t->lazy;
19.     t->sum += t->lazy*get_sz(t);
20.     if(t->l) t->l->lazy += t->lazy;
21.     if(t->r) t->r->lazy += t->lazy;
22.     t->lazy=0;
23. }
24.
25. //operation of segtree and size,parent update
26. void operation(pnode t) {
27.     if(!t)return;
28.     lazyUpdate(t->l); lazyUpdate(t->r); //imp:propagate lazy before combining t->l,t->r;
29.     t->sz=get_sz(t->l)+1+get_sz(t->r);
30.     t->sum = get_sum(t->l) + t->val + get_sum(t->r); // updateing sum
31.     if(t->l) t->l->p = t;
32.     if(t->r) t->r->p = t;
33. }
34.
35. // The subarray[1:pos] is saved in node l, the rest in r
36. void split(pnode t,pnode &l,pnode &r,int pos,int add=0){
37.     if(!t) return void( l = r = NULL) ;
38.     lazyUpdate(t);
39.     int curr_pos = add + get_sz(t->l)+1;
40.     if(curr_pos<=pos) split(t->r,t->r,r,pos,curr_pos),l=t;
41.     else split(t->l,l,t->l,pos,add),r=t;
42.     operation(t);
43. }
44.
45. void Merge(pnode &t,pnode l,pnode r){
46.     lazyUpdate(l); lazyUpdate(r);
47.     if(!l || !r) t = l?l:r;
48.     else if(l->prior>r->prior) Merge(l->r,l->r,r) , t = l ;
49.     else Merge(r->l,l,r->l) , t = r ;
50.     operation(t);
51. }

```

```
52.
53. pnode init(int val){
54.     pnode ret = (pnode)malloc(sizeof(node));
55.     ret->prior = rand();
56.     ret->sz = 1;
57.     ret->val = ret->sum = val;
58.     ret->lazy = 0;
59.     ret->p = ret->l = ret->r = NULL;
60.     return ret;
61. }
```

```
1. //changes the value of the node at position id to val
2. void point_update(pnode &t,int id,int val){
3.     pnode L,mid,R;
4.     split(t,L,mid,id-1);
5.     split(mid,t,R,1);
6.     t->val = val;
7.     Merge(mid,L,t);
8.     Merge(t,mid,R);
9. }
10.
11. //deletes the node at position id
12. void Remove(pnode &t,int id){
13.     pnode L,mid,R,X;
14.     split(t,L,mid,id-1);
15.     split(mid,X,R,1);
16.     delete X;
17.     Merge(t,L,R);
18. }
19.
20. //inserts a node at position id having array value = val
21. void Insert(pnode &t,int id,int val){
22.     pnode L,R,mid;
23.     pnode it = init(val);
24.     split(t,L,R,id-1);
25.     Merge(mid,L,it);
26.     Merge(t,mid,R);
27. }
28.
29. int range_query(pnode t,int l,int r){
30.     pnode L,mid,R;
31.     split(t,L,mid,l-1);
32.     split(mid,t,R,r-l+1);
33.     int ans = t->sum;
34.     Merge(mid,L,t);
35.     Merge(t,mid,R);
36.     return ans;
37. }
38.
39. void range_update(pnode t,int l,int r,int val){
40.     pnode L,mid,R;
41.     split(t,L,mid,l-1);
42.     split(mid,t,R,r-l+1);
43.     t->lazy += val;
44.     Merge(mid,L,t);
45.     Merge(t,mid,R);
46. }
47.
```



```
48. void Delete(pnode &t){
49.     if(!t) return;
50.     if(t->l) Delete(t->l);
51.     if(t->r) Delete(t->r);
52.     delete(t);
53.     t = NULL;
54. }
55.
56. int ara[10];
57.
58. int main(){
59.     //creating a treap to use it as an interval tree of ara (1 based)
60.     int n = 10;
61.     for(int i=1; i<=n; i++){
62.         if(i==1) Treap = init(ara[i]);
63.         else Merge(Treap,Treap,init(ara[i]));
64.     }
65.     Delete(Treap); //Deleting when work done
66.     return 0;
67. }
```

Maximum Contiguous Sum Merging

```
1. // Maximum contiguous sum merging
2. void operation(pnode t){
3.     if(!t)return;
4.     t->sum = get_sum(t->l) + t->val + get_sum(t->r);
5.     t->res = max( max(get_res(t->l), get_res(t->r)), max(0, get_rsum(t->l)) + t->val + max(0, get_lsum(t->r)));
6.     t->lsum = max(max(0,get_lsum(t->r)) + t->val + get_sum(t->l),get_lsum(t->l));
7.     t->rsum = max(get_sum(t->r) + t->val + max(0,get_rsum(t->l)),get_rsum(t->r));
8. }
```

Matrix Exponentiation

```

1. struct matrix{
2.     ll mat[100][100];
3.     int dim;
4.     matrix(){};
5.     matrix(int d){
6.         dim = d;
7.         for(int i=0;i<dim;i++){
8.             for(int j=0;j<dim;j++){
9.                 mat[i][j] = 0;
10.            }
11.            // mat = mat * mul
12.        matrix operator *(const matrix &mul){
13.            matrix ret = matrix(dim);
14.            for(int i=0;i<dim;i++){
15.                for(int j=0;j<dim;j++){
16.                    for(int k=0;k<dim;k++){
17.                        ret.mat[i][j] += (mat[i][k])*(mul.mat[k][j]) ;
18.                        ret.mat[i][j] %= MOD ;
19.                    }
20.                }
21.            }
22.            return ret ;
23.        }
24.        matrix operator + (const matrix &add){
25.            matrix ret = matrix(dim);
26.            for(int i=0;i<dim;i++){
27.                for(int j=0;j<dim;j++){
28.                    ret.mat[i][j] = mat[i][j] + add.mat[i][j] ;
29.                    ret.mat[i][j] %= MOD ;
30.                }
31.            }
32.            return ret ;
33.        }
34.        matrix operator ^(int p){
35.            matrix ret = matrix(dim);
36.            matrix m = *this ;
37.            for(int i=0;i<dim;i++) ret.mat[i][i] = 1 ; //identity matrix
38.            while(p){
39.                if( p&1 ) ret = ret * m ;
40.                m = m * m ;
41.                p >>= 1 ;
42.            }
43.            return ret ;
44.        }
45. };

```

Extended Euclid (Solving Linear Diophantine Equation)

```

1.  /*
2.      c = gcd(a,b);
3.
4.      ax + by = c;
5.      (bq + r)x + by = c;
6.      bqx + rx + by = c;
7.      b(qx + y) + rx = c;
8.      bx' + ry' = c;          [r = a % b]
9.
10.     We get,
11.     x' = qx + y;
12.     y' = x
13.
14.     So,
15.     y = x' - qx;
16.     y = x' - qy';          [y' = x]
17.     and
18.     x = y'
19.
20.     If c is not the gcd then,
21.
22.     actual x = x * (c/gcd)
23.     actual y = y * (c/gcd)
24.     But if gcd doesn't divide c, there is no solution.
25. */
26.
27.
28. // returns (x,y) for ax + by = gcd(a,b)
29. // keep in mind that if a or b or both are negative, gcd(a,b) will be negative
30.
31. PLL extEuclid(ll a,ll b)
32. {
33.     if(b==0LL) return mp(1LL,0LL);
34.     PLL ret, got;
35.     got = extEuclid(b,a%b);
36.     ret = mp(got.yy, got.xx-(a/b)*got.yy);
37.     return ret;
38. }
39.
40. /*
41.     From one solution (x0,y0), we can obtain all the solutions of the given equation.
42.     Let g = gcd(a,b) and let x0,y0 be integers which satisfy the following:
43.     a*x0+b*y0 = c
44.     Now, we should see that adding b/g to x0 and at the same time subtracting a/g
45.     from y0 will not break the equality:
46.
47.     a*(x0 + b/g) + b*(y0 - a/g)
48.     = a*x0 + b*y0 + (a*b)/g - (b*a)/g
49.     = c
50.     Obviously, this process can be repeated again, so all the numbers of the form:
51.
52.     x = x0 + k * (b/g)
53.     y = y0 - k * (a/g)

```

```
54.     are solutions of the given Diophantine equation.
55.
56.     Solution with minimum (x+y):
57.      $x + y = x_0 + y_0 + k \cdot (b/g - a/g)$ 
58.      $x + y = x_0 + y_0 + k \cdot ((b-a)/g)$ 
59.
60.     If  $b > a$ , we need to find the  $k$  with the minimum value
61.     else we need to find the  $k$  with the maximum value
62. */
63.
64. // Iterative Implementation
65. PLL extEuclid(ll a, ll b){
66.     ll s = 1, t = 0, st = 0, tt = 1;
67.     while(b){
68.         s = s - (a/b)*st;
69.         swap(s, st);
70.         t = t - (a/b)*tt;
71.         swap(t, tt);
72.         a = a % b;
73.         swap(a, b);
74.     }
75.     return mp(s, t);
76. }
```

```
1. // returns number of solutions for the equation  $ax + by = c$ 
2. // where  $\text{minx} \leq x \leq \text{maxx}$  and  $\text{miny} \leq y \leq \text{maxy}$ 
3. ll numberOfSolutions(ll a, ll b, ll c, ll minx, ll maxx, ll miny, ll maxy)
4. {
5.     if(a==0 && b==0){
6.         if(c!=0) return 0;
7.         else return (long long)(maxx-minx+1)*(maxy-miny+1);
8.     }
9.
10.    ll gcd = __gcd(a, b);
11.    if(c%gcd!=0) return 0;
12.
13.    if(b==0){
14.        c /= a;
15.        if(c>=minx && c<=maxx) return maxy-miny+1;
16.        else return 0;
17.    }
18.
19.    if(a==0){
20.        c /= b;
21.        if(c>=miny && c<=maxy) return maxx-minx+1;
22.        else return 0;
23.    }
24.
25.    PLL sol = extEuclid(a, b);
26.
27.    a /= gcd;
28.    b /= gcd;
29.    c /= gcd;
30.
31.    ll x, y;
32.    x = sol.xx*c;
33.    y = sol.yy*c;
34. }
```

```
35.     ll lx,ly,rx,ry;
36.
37.     if(x<minx) lx = getCeil(minx-x,abs(b));
38.     else lx = -getFloor(x-minx,abs(b));
39.
40.     if(x<maxx) rx = getFloor(maxx-x,abs(b));
41.     else rx = -getCeil(x-maxx,abs(b));
42.
43.     if(b<0){
44.         lx *= -1;
45.         rx *= -1;
46.         swap(lx,rx);
47.     }
48.     if(lx>rx) return 0;
49.
50.
51.     if(y<miny) ly = getCeil(miny-y,abs(a));
52.     else ly = -getFloor(y-miny,abs(a));
53.
54.     if(y<maxy) ry = getFloor(maxy-y,abs(a));
55.     else ry = -getCeil(y-maxy,abs(a));
56.
57.     if(a<0){
58.         ly *= -1;
59.         ry *= -1;
60.         swap(ly,ry);
61.     }
62.     if(ly>ry) return 0;
63.
64.     ly *= -1;
65.     ry *= -1;
66.     swap(ly,ry);
67.
68.     lx = max(lx,ly);
69.     rx = min(rx,ry);
70.
71.     return max(rx-lx+1,0LL);
72. }
73.
74. // works for negative numbers as well
75. ll getFloor(ll a,ll b)
76. {
77.     double x = a/(double)b;
78.     ll f = floor(x);
79.     while(f>x) f--;
80.     while(f+1<x) f++;
81.     return f;
82. }
83.
84. ll getCeil(ll a,ll b)
85. {
86.     double x = a/(double)b;
87.     ll c = ceil(x);
88.     while(c<x) c++;
89.     while(c-1>x) c--;
90.     return c;
91. }
```

Fraction Class

```
1. //a --> numerator
2. //b --> denominator
3.
4. /*
5.     Every fraction must be reduced before passing
6.     Fraction f2(-3,7); // declaration process
7. */
8.
9. struct Fraction{
10.     ll a,b;
11.     Fraction(){}
12.     Fraction(ll _a,ll _b){
13.         a = _a;
14.         b = _b;
15.     }
16. };
17.
18. ll GCD(ll a,ll b){
19.     if(b==0) return a;
20.     else return GCD(b,a%b);
21. }
22.
23. ll LCM(ll a,ll b){
24.     ll ret;
25.     ll gcd = GCD(a,b);
26.     ret = a/gcd;
27.     ret *= b;
28.     return ret;
29. }
30.
31. Fraction Reduce(Fraction f){
32.     ll gcd = GCD(f.a,f.b);
33.     f.a /= gcd;
34.     f.b /= gcd;
35.     return f;
36. }
37.
38. Fraction Add(Fraction x,Fraction y){
39.     Fraction sum;
40.
41.     sum.b = LCM(x.b,y.b);
42.     sum.a = 0;
43.     sum.a += (sum.b/x.b)*x.a;
44.     sum.a += (sum.b/y.b)*y.a;
45.
46.     return Reduce(sum);
47. }
48.
49. Fraction Sub(Fraction x,Fraction y){
50.     y.a *= -1;
51.     return Add(x,y);
52. }
53.
54.
55.
56. Fraction Mul(Fraction x,Fraction y){
```

```
57.     Fraction prod;
58.
59.     ll gcd = GCD(x.a,y.b);
60.     x.a /= gcd;
61.     y.b /= gcd;
62.
63.     gcd = GCD(y.a,x.b);
64.     y.a /= gcd;
65.     x.b /= gcd;
66.
67.     prod.a = x.a*y.a;
68.     prod.b = x.b*y.b;
69.
70.     return Reduce(prod);
71. }
72.
73. Fraction Div(Fraction x,Fraction y){
74.     Fraction ret;
75.     swap(y.a,y.b);
76.     return Mul(x,y);
77. }
```

Sieve of Eratosthenes

```
1.  bool isComp[MAX+5];
2.  vector <int> primes;
3.
4.  void Sieve(){
5.      int i,j;
6.      for(i=4;i<=MAX;i+=2) isComp[i] = true;
7.      for(i=3;i<=sqrt(MAX);i+=2){
8.          if(!isComp[i]){
9.              for(j=i*i;j<=MAX;j+=i+i) isComp[j] = 1;
10.         }
11.     }
12.     for(i=2;i<=MAX;i++) if(!isComp[i]) primes.pb(i);
13. }
```

Gaussian Elimination

```

1. #define SZ                105
2. #define EPS                1e-8
3.
4. double mat[SZ][SZ]; // Augmented Matrix
5. int where[SZ]; // where[i] denotes the row index of the pivot element of column i
6. double ans[SZ];
7.
8. /// n for row, m for column
9. int Gauss(int n,int m)
10. {
11.     SET(where);
12.     for(int row=0,col=0;col<m && row<n;col++){
13.         int max_row = row;
14.         for(int i=row;i<n;i++){
15.             if( abs(mat[i][col]) > abs(mat[max_row][col]) ) max_row = i;
16.
17.             if( abs(mat[max_row][col]) < EPS ) continue;
18.
19.             for(int i=col;i<=m;i++) swap(mat[row][i],mat[max_row][i]);
20.
21.             where[col] = row;
22.
23.             double mul;
24.             for(int i=row+1;i<n;i++){
25.                 if(abs(mat[i][col])>EPS){
26.                     mul = mat[i][col]/mat[row][col];
27.                     for(int j=col;j<=m;j++) mat[i][j] -= mul*mat[row][j];
28.                 }
29.             }
30.             row++;
31.         }
32.
33.         // checking 0 row
34.         double sum;
35.         for(int i=0;i<n;i++){
36.             sum = 0;
37.             for(int j=0;j<m;j++) sum += abs(mat[i][j]);
38.             if( abs(sum) < EPS && abs(mat[i][m]) > EPS ) return 0; //no solution
39.         }
40.
41.         // back substitution
42.         double sltn;
43.         int cur;
44.         for(int i=m-1;i>=0;i--){
45.             //if(where[i] == -1) return INF; // infinitely many solutions
46.             sltn = mat[where[i]][m];
47.             cur = where[i];
48.             for(int j = i+1; j<m; j++)
49.                 sltn -= mat[cur][j]*ans[j];
50.             ans[i] = sltn/mat[cur][i];
51.         }
52.
53.         return 1; // unique solution
54.     }

```


Gauss-Jordan Elimination

```

1. #define SZ 105
2.
3. double mat[SZ][SZ]; // Augmented Matrix
4. int where[SZ]; // where[i] denotes the row index of the pivot element of column i
5. double ans[SZ];
6.
7. // n for row, m for column
8. int GaussJordan(int n,int m)
9. {
10.     SET(where);
11.     for(int row=0,col=0; col<m && row<n; col++){
12.         int max_row = row;
13.         for(int i=row; i<n; i++)
14.             if( abs(mat[i][col]) > abs(mat[max_row][col]) ) max_row = i;
15.
16.         if( abs(mat[max_row][col]) < EPS ) continue;
17.
18.         for(int i=col; i<=m; i++) swap(mat[row][i],mat[max_row][i]);
19.
20.         where[col] = row;
21.
22.         double mul;
23.         for(int i=0; i<n; i++)
24.             if( i!=row && abs(mat[i][col])>EPS){
25.                 mul = mat[i][col]/mat[row][col];
26.                 for(int j=col; j<=m; j++) mat[i][j] -= mul*mat[row][j];
27.             }
28.         row++;
29.     }
30.     for(int i=0; i<m; i++)
31.         if(where[i]!=-1)
32.             ans[i] = mat[where[i]][m]/mat[where[i]][i];
33.
34.     double sum;
35.     for(int i=0; i<n; i++){
36.         sum = 0;
37.         for(int j=0; j<m; j++) sum += ans[j] * mat[i][j];
38.         if( abs(sum - mat[i][m]) > EPS ) return 0; // no solution
39.     }
40.
41.     for(int i=0; i<m; i++)
42.         if (where[i]==-1) return INF; // Infinitely many solutions
43.
44.     return 1; // unique solution
45. }

```

Gauss Related Problem

```

1. /*
2.     Problem :   Given a set of numbers, Find a subset such that the xor of the elements
3.
4.               of the subset is as large as possible
5.     Idea      :   We will try to make the MSB of the result 1 first, then the next bit

```

```
5.          There will be an equation for every bit
6.          if the numbers are 1101,0010,1010 then the equation form MSB is
7.          1*ans[0] + 0*ans[1] + 1*ans[2] = 1
8.  */
9.
10. #define MAX          105
11. ll ara[MAX];
12. bitset <MAX> mat[70];
13.
14. int row,ans[MAX],where[MAX];
15.
16. // bn = bit number
17. // n = number of columns
18. // val = the target value of the bn'th bit of the result
19. void add(int bn,int val,int n){
20.     ++row;
21.     mat[row][MAX-1] = val;
22.     //Stores the bn'th bit of every number in the matrix
23.     for(int col=0; col<n; col++) mat[row][col]=( (ara[col]>>bn) & 1 );
24.     // If this column has a pivot entry, we will xor the row with the row containg pivota
    t entry
25.     for(int col=0; col<n; col++){
26.         if(mat[row][col]){
27.             if(where[col]) mat[row] ^= mat[where[col]];
28.             else break;
29.         }
30.     }
31.     // Setting the pivot
32.     for(int col=0; col<n; col++){
33.         if(mat[row][col]){
34.             where[col]=row;
35.             return;
36.         }
37.     }
38.     // If no pivot element in the row, the equation is not added
39.     --row;
40. }
41.
42. void solve(int n,int m){
43.     CLR(where);
44.     row = 0;
45.     for(int i=m;i>=0;i--){
46.         add(i,1,n); // Trying to keep the i'th bit 1 of theresult
47.     }
48.     // Back Substitution
49.     for(int i=n-1;i>=0;i--){
50.         if(mat[where[i]][MAX-1]){
51.             ans[i] = 1;
52.             for(int j=1;j<=row;j++){
53.                 if(mat[j][i]) mat[j].flip(MAX-1);
54.             }
55.         }
56.         else ans[i] = 0;
57.     }
58. }
59. }
60.
61. int main(){
62.     int n;
63.     ll tot;
64. }
```

~ 50 ~

```
65.     scanf("%d",&n);
66.     tot = 0;
67.     for(int i=0;i<n;i++){
68.         scanf("%lld",&ara[i]);
69.         tot ^= ara[i];
70.     }
71.     solve(n,63);
72.     ll res = 0;
73.     for(int i=0; i<n; i++) res ^= (ara[i]*ans[i]);
74.     printf("%lld\n",res);
75. }
```

Chinese Remainder Theorem

```

1.  /*
2.      X = a_1 % m_1
3.      X = a_2 % m_2
4.      X = a_3 % m_3
5.
6.      m_1,m_2,m_3 are pair wise co-prime
7.
8.      M = m_1*m_2*m_3
9.
10.     u_i = Modular inverse of (M/m_i) with respect m_i
11.
12.     X = ( a_1 * (M/m_1) * u_1 + a_2 * (M/m_2) * u_2 + a_3 * (M/m_3) * u_3 ) % M
13. */

```

Lucas Theorem & Modulo Operation with Composite Number

```

1.  /*
2.      If we need to find  $nCr \% P$  where  $P$  is a prime but  $P$  can be
3.      less than  $n$  or  $r$ , we can use Lucas Theorem.
4.
5.       $nCr = ((n_0 \text{ C } r_0) * (n_1 \text{ C } r_1) * (n_2 \text{ C } r_2) * \dots * (n_k \text{ C } r_k)) \% P$ 
6.
7.      Where  $n_i$  is the  $i$ 'th digit in  $P$  based representation of  $n$ 
8.      and  $r_i$  is the  $i$ 'th digit in  $P$  based representation of  $r$ 
9.
10.     ** What if  $P$  is a composite number? **
11.
12.      $P = (p_0^{a_0}) * (p_1^{a_1}) * \dots * (p_k^{a_k})$ 
13.     where all  $p_i$  are prime numbers.
14.
15.      $nCr = (n!) / ((r!) * (n-r)!)$ 
16.
17.     If all  $a_i$  are 1, then we can use lucas to find individual mods for
18.     each  $p_i$  and combine those using CRT
19.
20.     If any  $a_i$  is greater than 1,
21.
22.     Let's Suppose,  $n! = (p_i^u) * x$ 
23.                    $(n-r)! = (p_i^v) * y$ 
24.                    $(r)! = (p_i^w) * z$ 
25.                   (See the code for calculation of  $x,y,z$  when
26.                    $n$  or  $r$  has large value)
27.
28.     Let's suppose  $p_i^{a_i} = t$ ,
29.      $\gcd(t,x) = \gcd(t,y) = \gcd(t,z) = 1$ , so,  $x,y,z$  will have modular inverse
30.     with respect to  $t$  (see Note 1)
31.
32.     So, we will find  $(x / (y*z)) \% (p_i^{a_i})$  and then multiply the
33.     result by  $(p_i^s)$  where  $s = u - v - w$ ;
34.     If,  $s$  is not smaller than  $a_i$ , then the result is 0.

```

```
35.
36.     Then, we will use CRT to combine the result.
37.     Actually, we don't need Lucas theorem anymore. This technique
38.     will work for  $a_i = 1$  also.
39.
40.
41.     *****Note 1*****
42.      $\phi(p^a) = (p^a) - (p^{(a-1)})$  if  $p$  is prime
43.
44.      $a^{-1} \phi(p^x) = 1 \pmod{p^x}$  if  $\gcd(a,p) = 1$ 
45.     modular inverse of  $a$  with respect to  $p^a$  is
46.      $a^{-1} (\phi(p^x) - 1) \pmod{p^x}$ 
47.     *****
48. */
49.
50.
51.
52. // returns factorail(n) % (p^a) ignoring prime number p
53. // can be done using a loop if n is small
54.
55. ll fact[MAX]; // size at least p^a
56.
57. ll call(ll n,ll p,ll a)
58. {
59.     ll ret = 1LL;
60.     ll x,m = 1;
61.
62.     //m = p^a
63.     for(int i=1;i<=a;i++) m *= p;
64.
65.     fact[0] = 1;
66.     for(ll i=1;i<=m;i++){
67.         if(i%p==0) fact[i] = fact[i-1];
68.         else fact[i] = (fact[i-1]*i)%m;
69.     }
70.
71.     while(true){
72.         if(n==0) break;
73.
74.         y = n/m;
75.         ret *= bm(fact[m],y,m);
76.         ret %= m;
77.
78.         y = n%m;
79.
80.         ret *= fact[y];
81.         ret %= m;
82.         n /= p;
83.     }
84.     return ret;
85. }
```

Fibonacci Numbers

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n.$$

$$F_{n+k} = F_k F_{n+1} + F_{k-1} F_n.$$

$$F_{2n} = F_n(F_{n+1} + F_{n-1}).$$

F_{nk} is always a multiple of F_n

$$\gcd(F_m, F_n) = F_{\gcd(m, n)}.$$

$$F_n = \frac{\sqrt{5}}{5} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$

Knuth Morris Pratt

```

1.  /* Complexity = O(P+S) */
2.  /* Searches pat in str */
3.
4.  char str[MAX],pat[MAX];
5.  int pref[MAX];
6.  int match[MAX];
7.
8.  //pref[i] = length of the longest suffix which is also
9.             //a proper prefix of the original string
10. void prefixFunction(int P){
11.     int j=0;
12.     for(int i=1; i<P; i++){
13.         while(true){
14.             if(pat[i]==pat[j]) {
15.                 j = pref[i] = j+1;
16.                 break;
17.             }
18.             else {
19.                 if(j==0) {
20.                     pref[i] = 0;
21.                     break;
22.                 }
23.                 else j = pref[j-1];
24.             }
25.         }
26.     }
27. }
28.
29. void KMPMatcher(int S) {
30.     int j = 0;
31.     for(int i=0; i<S; i++) {
32.         while(true) {
33.             if(str[i]==pat[j]) {
34.                 j = match[i] = j+1;
35.                 break;
36.             }
37.             else {
38.                 if(j==0) {
39.                     match[i] = 0;
40.                     break;
41.                 }
42.                 else j = pref[j-1];
43.             }
44.         }
45.     }
46. }
47.
48. int main(){
49.     scanf("%s",str);
50.     scanf("%s",pat);
51.     int S = strlen(str);
52.     int P = strlen(pat);
53.
54.     prefixFunction(P);
55.     KMPMatcher(S);
56.     return 0;

```

```
57. }
58.
59. /*
60.     NOTES
61.
62.     We can avoid the matcher by calculating
63.     the prefix function for the string = P + "#" + S
64. */
```

Suffix Array

```
1.  /*
2.      Suffix Array contains the lexicographically sorted order of the
3.      suffixes of string(stores the starting index of the suffix only)
4.  */
5.
6.  #define MAXL 10100 // max length of the string
7.  #define MAXLG 15
8.
9.  char str[MAXL];
10.
11. struct entry{
12.     int pr[2]; // parameters for sorting
13.     int id; // starting index of the suffix
14. }suf[MAXL];
15.
16. int P[MAXLG+5][MAXL+5];
17. // P[i][j] = position of the suffix starting at character j after sorting
18. // on the basis of 2^i characters
19.
20. bool cmp(entry a,entry b){
21.     if(a.pr[0]==b.pr[0]) return a.pr[1]<b.pr[1];
22.     else return a.pr[0]<b.pr[0];
23. }
24.
25.
26. // complexity n * lg n * lg n
27. void generateSA(int L){
28.     // suf[i].id will contain the starting index of the i'th suffix in suffix array
29.     int stp,now,i;
30.
31.     for(i=0;i<L;i++) P[0][i] = str[i]-'a'+1;
32.
33.     for(now=1,stp=1 ; now<L ; stp++,now *= 2){
34.         for(i=0;i<L;i++){
35.             suf[i].pr[0] = P[stp-1][i];
36.             if(i+now<L) suf[i].pr[1] = P[stp-1][i+now];
37.             else suf[i].pr[1] = -1;
38.             suf[i].id = i;
39.         }
40.         sort(suf,suf+L,cmp);
41.         for(i=0;i<L;i++){
42.             if(i>0 && suf[i].pr[0]==suf[i-1].pr[0] && suf[i].pr[1]==suf[i-1].pr[1])
43.                 P[stp][suf[i].id] = P[stp][suf[i-1].id];
44.             else
45.                 P[stp][suf[i].id] = i+1;
46.         }
47.     }
```



```

48.
49. // complexity lg n
50. int getLCP(int x,int y,int L)
51. {
52.     int ret = 0,add,i;
53.     for(i=MAXLG;i>=0;i--){
54.         if(P[i][x]==P[i][y] && P[i][x]!=0){
55.             add = (1<<i);
56.
57.             ret += add;
58.
59.             x += add;
60.             y += add;
61.         }
62.     }
63.     return ret;
64. }
65.
66.
67. int main(){
68.     CLR(P); //don't forget
69.     gets(str);
70.     L = strlen(str);
71.     generateSA(L);
72.     return 0;
73. }

```

Longest Increasing Subsequence ($n \log(n)$)

```

1.  /*
2.  The size of the vector after each iteration denotes the size of the LIS of the sub array
   starting at 1 and ending at i
3.  */
4.
5.  int ara[MAX];
6.  vector <int> v;
7.  int max_lis = 0;
8.  for(i=1;i<=n;i++){
9.      x = lower_bound(all(v),ara[i])-v.begin();
10.     if(x==0){
11.         if(v.size()==0) v.pb(ara[i]);
12.         else v[0] = ara[i];
13.     }
14.     else if(x==v.size()) v.pb(ara[i]);
15.     else if(ara[i]<v[x]) v[x] = ara[i];
16.     max_lis = max(max_lis,(int)v.size());
17. }
18. cout << "The size of the lis is : " << max_lis << endl;

```