

CODE LIBRARY

MD. SHAHADAT HOSSAIN SHAHIN

UNIVERSITY OF DHAKA

<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 %I64d",&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.
25. #define CLR(a)          memset(a,0,sizeof(a))
26. #define SET(a)          memset(a,-1,sizeof(a))
27.
28. #define eps            1e-9
29. #define PI             acos(-1.0)
30. #define MAX            100010
31. #define MOD            1000000007
32. #define INF            2000000000
33.
34. int setBit(int n,int pos){ return n = n | (1 << pos); } //sets the pos'th bit to 1
35. int resetBit(int n,int pos){ return n = n & ~(1 << pos); } //sets the pos'th bit to 0
36. bool checkBit(int n,int pos){ return (bool)(n & (1 << pos)); } //returns the pos'th bit
```

Graph

MST (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.
7. int par[MAX];
8.
9. int findParent(int u){
10.     while(par[u]!=u) u = par[u];
11.     return u;
12. }
13.
14. /*
15. int findParent(int u){
16.     if(par[u]==u) return u;
17.     else return par[u] = findParent(par[u]);
18. }
19. */
20.
21. int kruskal(int n,int m){
22.     sort(ara+1,ara+m+1,cmp);
23.     int i,mst;
24.     mst = 0;
25.     for(i=1;i<=n;i++) par[i] = i;
26.     for(i=1;i<=m;i++){
27.         edge x = ara[i];
28.         par[x.u] = findParent(x.u);
29.         par[x.v] = findParent(x.v);
30.         if(par[x.u]!=par[x.v]){
31.             par[par[x.u]] = par[x.v];
32.             mst += x.c;
33.         }
34.     }
35.     return mst;
36. }
```

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.city = s;
19. a.cost = 0;
20. priority_queue <node> q;
21. q.push(a);
22. while(!q.empty()){
23.     a = q.top();
24.     q.pop();
25.     u = a.city;
26.     if(!vis[u]){
27.         vis[u] = true;
28.         for(i=0;i<ed[u].size();i++){
29.             v = ed[u][i];
30.             c = co[u][i];
31.             if(dis[v]>dis[u]+c){
32.                 dis[v] = dis[u]+c;
33.                 b.city = v;
34.                 b.cost = dis[v];
35.                 q.push(b);
36.             }
37.         }
38.     }
39. }
40. }

```

Floyd Warshall :

```

1. int dis[MAX][MAX];
2.
3. void warshall(int n){
4.     int i,j,k;
5.     for(i=0;i<n;i++)
6.         for(j=0;j<n;j++) dis[i][j] = INF;
7.
8.     for(k=0;k<n;k++){
9.         for(i=0;i<n;i++){
10.            for(j=0;j<n;j++){
11.                if(dis[i][k]!=INF && dis[k][j]!=INF && dis[i][k]+dis[k][j]<=dis[i][j])
12.                    dis[i][j] = dis[i][k]+dis[k][j];
13.            }
14.        }
15.    }
16. }

```

Bellman Ford :

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

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.     for(int i=1;i<=n;i++) if(!vis[i]) findArt(i,-1);
27. }
```

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.     for(int i=1;i<=n;i++) if(!vis[i]) findBridge(i,-1);
26. }
```

Strongly Connected Component (Kosaraju's Algorithm) :

```
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. int n;
13.
14. void topSort(int s){
15.     int i,x;
16.     vis[s] = 1;
17.     for(i=0;i<edges[s].size();i++){
18.         x = edges[s][i];
19.         if(!vis[x]) topSort(x);
20.     }
21.     topSortedNodes.push(s);
22. }
23.
24. void markComponent(int s){
```

```

25.     int i,x;
26.     vis[s] = 1;
27.     compNum[s] = cnum;
28.     for(i=0;i<trans[s].size();i++){
29.         x = trans[s][i];
30.         if(!vis[x]) markComponent(x);
31.     }
32. }
33.
34. void SCC(){
35.     int i,x;
36.     CLR(vis);
37.     for(int i=1;i<=n;i++){
38.         if(!vis[i]) topSort(i);
39.     }
40.     cnum = 0;
41.     CLR(vis);
42.     while(!topSortedNodes.empty()){
43.         x = topSortedNodes.top();
44.         topSortedNodes.pop();
45.         if(!vis[x]){
46.             cnum++;
47.             markComponent(x);
48.         }
49.     }
50. }

```

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.
8.  /*
9.  In the following code
10. bcc_counter --> Total number of biconnected components
11. bcc vector keeps the list of nodes in a single BCC.
12. */
13.
14.
15. vector <int> edges[MAX];
16. bool vis[MAX] , isArt[MAX];
17. int Time;
18. int low[MAX],st[MAX];
19. vector <int> bcc[MAX];
20. int bcc_counter , n;
21. stack <int> S;
22.
23. void findBCC(int s,int par){
24.     S.push(s);
25.     int i,x,child = 0;
26.     vis[s] = 1;
27.     Time++;
28.     st[s] = low[s] = Time;
29.     for(i=0;i< edges[s].size();i++){

```

```

30.     x = edges[s][i];
31.     if(!vis[x]){
32.         child++;
33.         findBCC(x,s);
34.         low[s] = min(low[s],low[x]);
35.         if(par!=-1 && low[x]>=st[s]){
36.             isArt[s] = 1;
37.             bcc[bcc_counter].pb(s);
38.             while(1){
39.                 bcc[bcc_counter].pb(S.top());
40.                 if(S.top()==x){
41.                     S.pop();
42.                     break;
43.                 }
44.                 S.pop();
45.             }
46.             bcc_counter++;
47.         }
48.         else if(par==-1){
49.             if(child>1){
50.                 isArt[s] = 1;
51.                 bcc[bcc_counter].pb(s);
52.                 while(1){
53.                     bcc[bcc_counter].pb(S.top());
54.                     if(S.top()==x){
55.                         S.pop();
56.                         break;
57.                     }
58.                     S.pop();
59.                 }
60.                 bcc_counter++;
61.             }
62.         }
63.     }
64.     else{
65.         if(par!=x) low[s] = min(low[s],st[x]);
66.     }
67. }
68. if(par==-1 && child>1) isArt[s] = 1;
69. }
70.
71. void processBCC(){
72.     for(int i=0;i<n;i++){
73.         if(!vis[i]){
74.             Time = 0;
75.             findBCC(i,-1);
76.             bool lala = false;
77.             while(!S.empty()){
78.                 lala = true;
79.                 bcc[bcc_counter].push_back(S.top());
80.                 S.pop();
81.             }
82.             if(lala) bcc_counter++;
83.         }
84.     }
85. }

```



```

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.
7.  vector <int> edges[MAX];
8.  bool vis[MAX] , isArt[MAX];
9.  int st[MAX] , low[MAX] , Time = 0;
10. stack <PII> S;
11. int n,bcc_counter;
12.
13. void findBCC(int s,int par)
14. {
15.     int i,x,child = 0;
16.     vis[s] = 1;
17.     Time++;
18.     st[s] = low[s] = Time;
19.     for(i=0;i<edges[s].size();i++){
20.         x = edges[s][i];
21.         if(!vis[x]){
22.             S.push(mp(s,x));
23.             child++;
24.             findBCC(x,s);
25.             low[s] = min(low[s],low[x]);
26.             if(/*par!=-1 &&*/ low[x]>=st[s]){
27.                 isArt[s] = 1;
28.                 PII cur,e = mp(s,x);
29.                 bcc_counter++;
30.                 cout << "Edges of Component " << bcc_counter << ":" << endl;
31.                 do{
32.                     cur = S.top();
33.                     S.pop();
34.                     cout << cur.xx << "--" << cur.yy << endl;
35.                 }while(cur!=e);
36.             }
37.         }
38.         else if(par!=x && st[x]<st[s]){
39.             S.push(mp(s,x));
40.             low[s] = min(low[s],st[x]);
41.         }
42.     }
43.     if(par==-1 && child>1) isArt[s] = 1;
44. }
45.
46. void processBCC(){
47.     bcc_counter = 0;
48.     for(int i=0;i<n;i++){
49.         if(!vis[i]){
50.             findBCC(i,-1);
51.         }
52.     }
53. }

```

Data Structures

Segment Tree

```
1.  /*
2.  Segment tree with point update and range query
3.  */
4.
5.  int ara[MAX];
6.
7.  struct node{
8.      int sum;
9.  }tree[4*MAX];
10.
11. node Merge(node a,node b){
12.     node ret;
13.     ret.sum = a.sum+b.sum;
14.     return ret;
15. }
16.
17. void build(int n,int st,int ed){
18.     if(st==ed){
19.         tree[n].sum = ara[st];
20.         return;
21.     }
22.     int mid = (st+ed)/2;
23.     build(2*n,st,mid);
24.     build(2*n+1,mid+1,ed);
25.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
26. }
27.
28. void update(int n,int st,int ed,int id,int v){
29.     if(id>ed || id<st) return;
30.     if(st==ed && ed==id){
31.         tree[n].sum = v;
32.         return;
33.     }
34.     int mid = (st+ed)/2;
35.     update(2*n,st,mid,id,v);
36.     update(2*n+1,mid+1,ed,id,v);
37.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
38. }
39.
40. node query(int n,int st,int ed,int i,int j){
41.     if(st>=i && ed<=j) return tree[n];
42.     int mid = (st+ed)/2;
43.     if(mid<i) return query(2*n+1,mid+1,ed,i,j);
44.     else if(mid>=j) return query(2*n,st,mid,i,j);
45.     else return Merge(query(2*n,st,mid,i,j),query(2*n+1,mid+1,ed,i,j));
46. }
```

```

1.  /*
2.  Segment tree with range update and range query
3.  */
4.  int ara[MAX];
5.
6.  struct node{
7.      int sum;
8.  }tree[4*MAX];
9.
10. int lazy[4*MAX];
11.
12. node Merge(node a,node b){
13.     node ret;
14.     ret.sum = a.sum+b.sum;
15.     return ret;
16. }
17.
18. void lazyUpdate(int n,int st,int ed){
19.     if(lazy[n]!=0){
20.         tree[n].sum += ((ed-st+1)*lazy[n]);
21.         if(st!=ed){
22.             lazy[2*n] += lazy[n];
23.             lazy[2*n+1] += lazy[n];
24.         }
25.         lazy[n] = 0;
26.     }
27. }
28.
29. void build(int n,int st,int ed){
30.     lazy[n] = 0;
31.     if(st==ed){
32.         tree[n].sum = ara[st];
33.         return;
34.     }
35.     int mid = (st+ed)/2;
36.     build(2*n,st,mid);
37.     build(2*n+1,mid+1,ed);
38.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
39. }
40. void update(int n,int st,int ed,int i,int j,int v){
41.     lazyUpdate(n,st,ed);
42.     if(st>j || ed<i) return;
43.     if(st>=i && ed<=j){
44.         lazy[n] += v;
45.         lazyUpdate(n,st,ed);
46.         return;
47.     }
48.     int mid = (st+ed)/2;
49.     update(2*n,st,mid,i,j,v);
50.     update(2*n+1,mid+1,ed,i,j,v);
51.     tree[n] = Merge(tree[2*n],tree[2*n+1]);
52. }
53.
54. node query(int n,int st,int ed,int i,int j){
55.     lazyUpdate(n,st,ed);
56.     if(st>=i && ed<=j) return tree[n];
57.     int mid = (st+ed)/2;
58.     if(mid<i) return query(2*n+1,mid+1,ed,i,j);
59.     else if(mid>=j) return query(2*n,st,mid,i,j);
60.     else return Merge(query(2*n,st,mid,i,j),query(2*n+1,mid+1,ed,i,j));
61. }

```

Implicit Segment Tree

```
1.  /*
2.  Point Update, Range Query
3.  */
4.
5.  struct node{
6.      int sum;
7.      node *left,*right;
8.      node(){ }
9.      node(int value){
10.         sum = value;
11.         left = right = NULL;
12.     }
13. };
14.
15. void update(node *cur,int st,int ed,int id,int v)
16. {
17.     if(id<st || id>ed) return;
18.     if(id==st && id==ed){
19.         cur->sum = v;
20.         return;
21.     }
22.     int mid = (st+ed)/2;
23.     if(cur->left==NULL) cur->left = new node(0);
24.     if(cur->right==NULL) cur->right = new node(0);
25.     update(cur->left,st,mid,id,v);
26.     update(cur->right,mid+1,ed,id,v);
27.     cur->sum = cur->left->sum + cur->right->sum;
28. }
29.
30. int query(node *cur,int st,int ed,int i,int j)
31. {
32.     if(st>=i && ed<=j) return cur->sum;
33.     int mid = (st+ed)/2;
34.     if(cur->left==NULL) cur->left = new node(0);
35.     if(cur->right==NULL) cur->right = new node(0);
36.     if(mid<i) return query(cur->right,mid+1,ed,i,j);
37.     else if(mid>=j) return query(cur->left,st,mid,i,j);
38.     else return query(cur->right,mid+1,ed,i,j)+query(cur->left,st,mid,i,j);
39. }
40.
41. int main()
42. {
43.     int n = 1000000000;
44.     node *root = new node(0);
45.     update(root,1,n,5,1);
46.     update(root,1,n,3,1);
47.     cout << query(root,1,n,1,5) << endl;
48.     return 0;
49. }
```

```
1.  /*
2.  Range Update, Range Query
3.  */
4.
5.  struct node{
```

```

6.     int sum,lazy;
7.     node *left,*right;
8.     node(){ }
9.     node(int value){
10.         sum = value;
11.         lazy = 0;
12.         left = right = NULL;
13.     }
14. };
15.
16. void lazyUpdate(node *cur,int st,int ed)
17. {
18.     if(cur->lazy!=0){
19.         cur->sum += ((ed-st+1)*cur->lazy);
20.         if(st!=ed){
21.             if(cur->left==NULL) cur->left = new node(0);
22.             if(cur->right==NULL) cur->right = new node(0);
23.             cur->left->lazy += cur->lazy;
24.             cur->right->lazy += cur->lazy;
25.         }
26.         cur->lazy = 0;
27.     }
28. }
29.
30. void update(node *cur,int st,int ed,int i,int j,int v){
31.     lazyUpdate(cur,st,ed);
32.     if(st>j || ed<i) return;
33.     if(st>=i && ed<=j){
34.         cur->lazy += v;
35.         lazyUpdate(cur,st,ed);
36.         return;
37.     }
38.     int mid = (st+ed)/2;
39.     if(cur->left==NULL) cur->left = new node(0);
40.     if(cur->right==NULL) cur->right = new node(0);
41.     update(cur->left,st,mid,i,j,v);
42.     update(cur->right,mid+1,ed,i,j,v);
43.     cur->sum = cur->left->sum + cur->right->sum;
44. }
45.
46. int query(node *cur,int st,int ed,int i,int j){
47.     lazyUpdate(cur,st,ed);
48.     if(st>=i && ed<=j) return cur->sum;
49.     int mid = (st+ed)/2;
50.     if(cur->left==NULL) cur->left = new node(0);
51.     if(cur->right==NULL) cur->right = new node(0);
52.     if(mid<i) return query(cur->right,mid+1,ed,i,j);
53.     else if(mid>=j) return query(cur->left,st,mid,i,j);
54.     else return query(cur->right,mid+1,ed,i,j)+query(cur->left,st,mid,i,j);
55. }
56.
57. int main()
58. {
59.     int n = 1000000000;
60.     node *root = new node(0);
61.     update(root,1,n,1,5,1);
62.     update(root,1,n,4,10,1);
63.     update(root,1,n,9,14,1);
64.     cout << query(root,1,n,1,20) << endl;
65.     return 0;
66. }

```

BIT

```
1.  /*
2.  Initially All the array elemets are zero
3.  Point Update(Adding xx to index i)
4.  Query returns sum from index 1 to index i
5.  */
6.
7.  int tree[MAX];
8.
9.  //n --> size
10. //x --> value to be added to index idx
11. void update(int idx, int x, int n){
12.     while(idx<=n){
13.         tree[idx]+=x;
14.         idx += idx & (-idx);
15.     }
16. }
17.
18. int query(int idx){
19.     int sum=0;
20.     while(idx>0){
21.         sum += tree[idx];
22.         idx -= idx & (-idx);
23.     }
24.     return sum;
25. }
```

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,b;
16.     //b--> block size
17. }quer[MAX_SZ];
18.
19. bool cmp(data a,data b){
20.     if(a.seg==b.seg) return a.r<b.r;
21.     return a.seg<b.seg;
22. }
23.
24. void Add(int id){
25.     cnt[ara[id]]++;
26.     ///update ans
```

```

27. }
28.
29. void Remove(int id){
30.     cnt[ara[id]]--;
31.     ///update ans
32. }
33.
34. void Mo(int q)
35. {
36.     int L = 0, R = 0,l,r;
37.     ans = 0;
38.     Add(0);
39.     for(int i=0;i<q;i++){
40.         l = quer[i].l;
41.         r = quer[i].r;
42.         while(L>l){
43.             Add(L-1); L--;
44.         }
45.         while(L<l){
46.             Remove(L); L++;
47.         }
48.         while(R>r){
49.             Remove(R); R--;
50.         }
51.         while(R<r){
52.             Add(R+1); R++;
53.         }
54.         res[quer[i].id] = ans;
55.     }
56. }
57.
58. int main()
59. {
60.     int q;
61.     sort(quer,quer+q,cmp);
62.     Mo(q);
63.     return 0;
64. }

```

Lowest Common Ancestor

```

1. #define lg 14
2.
3. int L[MAX]; // Depth of a node
4. int T[MAX]; // Immediate Parent of a node
5. int P[MAX][lg+2]; // P[i][j] denotes (2^j)th parent of node i
6.
7. void lca_build(int n){
8.     SET(P);
9.     int i,j;
10.    for(i=1;i<=n;i++) P[i][0] = T[i];
11.    for(j=1;(1<<j)<=n;j++)
12.        for(i=1;i<=n;i++)
13.            if(P[i][j-1]!=-1) P[i][j] = P[P[i][j-1]][j-1];
14. }
15.
16. int lca_query(int x,int y){

```

```

17.     if(L[x]<L[y]) swap(x,y);
18.
19.     int i,j;
20.     for(i=lg;i>=0;i--){
21.         if(L[x] - (1<<i) >= L[y]) x = P[x][i];
22.     }
23.     if(x==y) return x;
24.     for(i=lg;i>=0;i--){
25.         if(P[x][i]!=-1 && P[x][i]!=P[y][i]){
26.             x = P[x][i];
27.             y = P[y][i];
28.         }
29.     }
30.     return T[x];
31. }

```

Trie

```

1. #define TC 26
2.
3. struct node{
4.     bool endmark;
5.     node *next[TC];
6.     node(){
7.         endmark = false;
8.         for(int i=0;i<TC;i++) next[i] = NULL;
9.     }
10. }*root;
11.
12. void Insert(char *str,int len){
13.     node* cur = root;
14.     for(int i=0;i<len;i++){
15.         int id = str[i]-'a';
16.         if(cur->next[id]==NULL) cur->next[id] = new node();
17.         cur = cur->next[id];
18.     }
19.     cur->endmark = true;
20. }
21.
22. bool Search(char *str,int len){
23.     node *cur = root;
24.     for(int i=0;i<len;i++){
25.         int id = str[i]-'a';
26.         if(cur->next[id]==NULL) return false;
27.         cur = cur->next[id];
28.     }
29.     return cur->endmark;
30. }
31.
32. void Delete(node* cur){
33.     for(int i=0;i<TC;i++) if(cur->next[i]!=NULL) Delete(cur->next[i]);
34.     delete(cur);
35. }

```


Matrix Exponentiation

```
1. struct matrix{
2.     int mat[2][2];
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.    void print(){
46.        for(int i=0;i<dim;i++){
47.            for(int j=0;j<dim;j++){
48.                printf("%d ",mat[i][j]);
49.            }
50.            printf("\n");
51.        }
52.    }
53. };
```

LCS($n \cdot \log(n)$)

```
1.  /*
2.  The size of the vector after each iteration denotes the size of the LCS of the sub array
   starting at 1 and ending at i
3.  */
4.
5.  int ara[MAX];
6.  vector <int> v;
7.  int max_lcs = 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_lcs = max(max_lcs,(int)v.size());
17. }
18. cout << "The size of the lcs is : " << max_lcs << endl;
```

Max Flow(Edmonds Carp)

```
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. int cap[MAX][MAX];
7. int par[MAX]; //keeps track of the parent in a path from s to d
8. int mCap[MAX]; //mCap[i] keeps track edge that have minimum cost on the shortest path from s to i
9.
10. bool getPath(int s,int d,int n)
11. {
12.     SET(par);
13.     for(int i=1;i<=n;i++) mCap[i] = INF;
14.     queue<int> q;
15.     q.push(s);
16.     while(!q.empty()){
17.         int u = q.front();
18.         q.pop();
19.         for(int i=1;i<=n;i++){
20.             if(cap[u][i]!=0 && par[i]==-1){
21.                 par[i] = u;
22.                 mCap[i] = min(mCap[u],cap[u][i]);
23.                 if(i==d) return true;
24.                 q.push(i);
25.             }
26.         }
27.     }
28.     return false;
29. }
30.
31. int getFlow(int s,int d,int n)
32. {
33.     int F = 0;
34.     while(getPath(s,d,n)){
35.         int f = mCap[d];
36.         F += f;
37.         int u = d;
38.         while(u!=s){
39.             int v = par[u];
40.             cap[u][v] += f;
41.             cap[v][u] -= f;
42.             u = v;
43.         }
44.     }
45.     return F;
46. }
47.
48. int main()
49. {
50.     int maxFlow = getFlow(s,d,n);
51.     return 0;
52. }
```

Max Flow(Dinic)

```
1. //Complexity V*V*E
2.
3. #define MAX_NODES    5000
4.
5. int src,snk;
6. int dist[MAX_NODES],work[MAX_NODES];
7.
8. struct Edge{
9.     int to , rev_pos , c , f;
10. };
11.
12. vector <Edge> G[MAX_NODES];
13.
14. //This function is written for undirected graph
15. //The graph doesn't have multiple edge between two nodes
16. void addEdge(int u,int v,int c){
17.     Edge a = {v,(int)G[v].size(),c,0};
18.     Edge b = {u,(int)G[u].size(),c,0};
19.     G[u].pb(a);
20.     G[v].pb(b);
21. }
22.
23. bool dinic_bfs(){
24.     SET(dist);
25.     dist[src] = 0;
26.     queue <int> q;
27.     q.push(src);
28.     while(!q.empty()){
29.         int u = q.front();
30.         q.pop();
31.         for(int i=0;i<G[u].size();i++){
32.             Edge &e = G[u][i];
33.             int v = e.to;
34.             if(dist[v]==-1 && e.f<e.c){
35.                 dist[v] = dist[u]+1;
36.                 q.push(v);
37.             }
38.         }
39.     }
40.     return (dist[snk]>=0);
41. }
42.
43. int dinic_dfs(int u, int fl){
44.     if (u == snk) return fl;
45.     for (; work[u] < G[u].size(); work[u]++){
46.         Edge &e = G[u][work[u]];
47.         if (e.c <= e.f) continue;
48.         int v = e.to;
49.         if (dist[v] == dist[u] + 1){
50.             int df = dinic_dfs(v, min(fl, e.c - e.f));
51.             if (df > 0){
52.                 e.f += df;
53.                 G[v][e.rev_pos].f -= df;
54.                 return df;
55.             }
56.         }
57.     }
58.     return 0;
```

```
59. }
60.
61.
62. int maxFlow(int _src, int _snk){
63.     src = _src;
64.     snk = _snk;
65.     int result = 0;
66.     while (dinic_bfs()){
67.         CLR(work);
68.         while (int delta = dinic_dfs(src, INF)) result += delta;
69.     }
70.     return result;
71. }
72.
73. int main()
74. {
75.     cout << maxFlow(src,snk) << endl;
76.     return 0;
77. }
```

Number Theory

Sieve

```
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. }
```

Euler Phi

```
1. int phi[MAX+10];
2.
3. void calcPhi() {
4.     int i,p,k;
5.     for (i = 1; i <= MAX; i++) phi[i] = i;
6.     for (p = 2; p <=MAX; p++) {
7.         if (phi[p] == p) {
8.             for (k = p; k <= MAX; k += p) {
9.                 phi[k] /= p;
10.                phi[k] *= (p-1);
11.            }
12.        }
13.    }
14. }
```

Extended Euclid

```
1. // ax+by = gcd(a,b)
2. // returns (x,y)
3.
4. PLL extEuclid(ll a,ll b)
5. {
6.     ll s = 1,t = 0,st = 0,tt = 1;
7.     while(b){
8.         s = s - (a/b)*st;
9.         swap(s,st);
10.        t = t - (a/b)*tt;
11.        swap(t,tt);
12.        a = a%b;
13.        swap(a,b);
14.    }
15.    return mp(s,t);
16. }
```