Team notebook

L9, International Institute of Information Technology, Hyderabad

December 27, 2018

Contents

1 DP

1.1 SOS Dp

```
// iterate over all the masks Complexity - O(3^n)
for (int mask = 0; mask < (1<<n); mask++){
       F[mask] = A[0];
    // iterate over all the subsets of the mask
    for(int i = mask; i > 0; i = (i-1) & mask){
       F[mask] += A[i];
    }
}
//iterative version
```

```
for(int mask = 0; mask < (1<<N); ++mask){
	dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
	for(int i = 0;i < N; ++i){
		if(mask & (1<<i))
			dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
		else
			dp[mask][i] = dp[mask][i-1];
	}
	F[mask] = dp[mask][N-1];
}
//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
	F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
	if(mask & (1<<i))
		F[mask] += F[mask^(1<<i)];
} // Complexity - O(N * 2^n);
```

2 Data Strutures

2.1 BIT

```
ordered_set bit[N];
void insert(int x, int y)
{
	for(int i = x; i < N; i += i & -i)
		bit[i].insert(mp(y, x));
}

void remove(int x, int y)
{
	for(int i = x; i < N; i += i & -i)
		bit[i].erase(mp(y, x));
}

int query(int x, int y)
{
	int ans = 0;
	for(int i = x; i > 0; i -= i & -i)
		ans += bit[i].order_of_key(mp(y+1, 0));
	return ans;
}
```

2.2 Lazy Propagation

```
void build(int l,int h,int p)
{
	if(l==h)
	{
		tr[p]=ar[l];
		return;
	}
	int m=l+(h-l)/2;
	build(l,m,2*p+1);
	build(m+1,h,2*p+2);
	tr[p]=min(tr[2*p+1],tr[2*p+2]);
}
void puke(int p)
{
	tr[2*p+1]+=lz[p];
	lz[2*p+1]+=lz[p];
	tr[2*p+2]+=lz[p];
	lz[2*p+2]+=lz[p];
	lz[p]=0;
}
void update(int l,int h,int ql,int qh,int v,int p)
{
	if(l>qh || h<ql)
		return;
	if(l>=ql && h<=qh)
	{
		tr[p]+=v;
		lz[p]+=v;
		return;
	}
	int m=l+(h-l)/2;
	puke(p);
	update(l,m,ql,qh,v,2*p+1);
	update(m+1,h,ql,qh,v,2*p+2);
	tr[p]=min(tr[2*p+1],tr[2*p+2]);
}
ll query(int l,int h,int ql,int qh,int p)
{
	if(l>qh || h<ql)
		return LONG_MAX;
	if(l>=ql && h<=qh)
		return tr[p];
	int m=l+(h-l)/2;
```

```
	puke(p);
	return min(query(l,m,ql,qh,2*p+1),query(m+1,h,ql,qh,2*p+2));
}
```

2.3 Mergesort Tree

```
void build(ll node, ll curr_l, ll curr_r)
{
	// pair in node of tree stores element and its frequency in the
	    subarray
	if(curr_l == curr_r)
		arr[node].insert(mp(a[curr_r], 1));
	else
	{
		ll curr_mid = (curr_l + curr_r) >> 1;
		build(2 * node + 1, curr_l, curr_mid);
		build(2 * node + 2, curr_mid + 1, curr_r);

		for(auto it: arr[2 * node + 1])
			arr[node].insert(it);
		for(auto it:arr[2 * node + 2])
		{
			ll f = it.first;
			auto it1 = arr[node].lower_bound(mp(f, NINF));
			ll f1 = (*it1).second;
			if(it1 != arr[node].end() && (*it1).first == f)
			{
				arr[node].erase(arr[node].find(*it1));
				arr[node].insert(mp(f, it.second + f1));
			}
			else
				arr[node].insert(it);
		}
	}
}
ll query(ll node, ll curr_l, ll curr_r, ll query_l, ll query_r, ll x)
{
	if(query_l > query_r || curr_l > curr_r || curr_l > query_r ||
	    curr_r < query_l) return 0;
	if(curr_l >= query_l && curr_r <= query_r)
	{
		return sz(arr[node]) - arr[node].order_of_key(mp(x, NINF));
	}
	else
	{
		ll curr_mid = (curr_l + curr_r) >> 1;
		ll left_value = query(2 * node + 1, curr_l, curr_mid,
		    query_l, min(query_r, curr_r), x);
		ll right_value = query(2 * node + 2, curr_mid + 1, curr_r,
		    max(query_l, curr_l), query_r, x);
		return left_value + right_value;
	}
}
void point_update(ll node, ll curr_l, ll curr_r, ll pos, ll new_val, ll
    prev_val)
{
	if(new_val == prev_val) return;
	if(curr_l == curr_r)
	{
		arr[node].erase(arr[node].find(mp(prev_val, 1)));
		arr[node].insert(mp(new_val, 1));
	}
	else
	{
		ll curr_mid = (curr_l + curr_r) >> 1;
		if(curr_mid >= pos)
			point_update(2 * node + 1, curr_l, curr_mid, pos,
			    new_val, prev_val);
		else
			point_update(2 * node + 2, curr_mid + 1, curr_r,
			    pos, new_val, prev_val);

		auto it = arr[node].lower_bound(mp(prev_val, NINF));
		ll fi = (*it).first;
		ll se = (*it).second;
		arr[node].erase(it);
		auto it1 = arr[node].lower_bound(mp(new_val, NINF));
		if(it1 != arr[node].end() && (*it1).first == new_val)
		{
			ll f1 = (*it1).first;
			ll f2 = (*it1).second;
			arr[node].erase(arr[node].find(mp(f1, f2)));
			arr[node].insert(mp(f1, f2 + 1));
		}
		else
			arr[node].insert(mp(new_val, 1));
		if(se != 1)
			arr[node].insert(mp(prev_val, se - 1));
```

```
	}
}
```

2.4 Sqrt Decomposition

```
#define MAXN 10000
#define SQRSIZE 100
int arr[MAXN], block[SQRSIZE], blk_sz; // original array, decomposed
    array, block size
void update(int idx, int val)
{
	int blockNumber = idx / blk_sz;
	block[blockNumber] += val - arr[idx];
	arr[idx] = val;
}
int query(int l, int r)
{
	int sum = 0;
	while (l<r and l%blk_sz!=0 and l!=0)
	{
		// traversing first block in range
		sum += arr[l];
		l++;
	}
	while (l+blk_sz <= r)
	{
		// traversing completely overlapped blocks in range
		sum += block[l/blk_sz];
		l += blk_sz;
	}
	while (l<=r)
	{
		// traversing last block in range
		sum += arr[l];
		l++;
	}
	return sum;
}
// Fills values in input[]
void preprocess(int input[], int n)
{
	// initiating block pointer
	int blk_idx = -1;

	// calculating size of block
	blk_sz = sqrt(n);

	// building the decomposed array
	for (int i=0; i<n; i++)
	{
		arr[i] = input[i];
		if (i%blk_sz == 0)
		{
			// entering next block
			// incementing block pointer
			blk_idx++;
		}
		block[blk_idx] += arr[i];
	}
}
```

2.5 Stack for Range Min

```
hd[h++]=pi(ar[1],1);
lt[1]=1;
for(i=2;i<=n;i++)
{
	while(h!=0 && hd[h-1].ff>=ar[i])
		h--;
	if(!h) lt[i]=1;
	else lt[i]=hd[h-1].ss+1;
	hd[h++]=pi(ar[i],i);
}
tl[g++]=pi(ar[n],n);
rt[n]=n;
for(i=n-1;i>=1;i--)
{
	while(g!=0 && tl[g-1].ff>=ar[i])
		g--;
	if(!g) rt[i]=n;
	else rt[i]=tl[g-1].ss-1;
	tl[g++]=pi(ar[i],i);
}
```

2.6 Ternary Search

```
//for integers (for maxima)
int lo = -1, hi = n;
while (hi - lo > 1)
{
    int mid = (hi + lo)>>1;
    if (f(mid) > f(mid + 1))
        hi = mid;
    else
        lo = mid;
 }
 //for reals (for minima)
for(int i = 0; i < LOG; i++)
{
    ld m1 = (A * 2 + B) / 3.0;
    ld m2 = (A + 2 * B) / 3.0;

    if(f(m1) > f(m2))
       A = m1;
    else
       B = m2;
}
ans = f(A);
```

2.7 Trie

```
typedef struct node{
       node* nxt[2];
}node;
node* nnode()
{
       node* tmp = new node;
       tmp->nxt[0]=tmp->nxt[1]=NULL;
       return tmp;
}
node* insert(node* root,char ar[])
{
       node* cur=root;
       for(int i=0;i<=31;++i)
       {
              if(cur->nxt[ar[i]-'0']==NULL)
              {
                     node* tmp=nnode();
                     cur->nxt[ar[i]-'0']=tmp;
                     cur=cur->nxt[ar[i]-'0'];
              }
              else
                     cur=cur->nxt[ar[i]-'0'];
       }
       return root;
}
ll search(node* root,char ar[])
{
       node* cur=root;
       ll s=0;
       for(int i=0;i<=31;++i)
       {
              ll v=ar[i]-'0';
              if(cur->nxt[v^1]==NULL)
              {
                     s+=(1ll<<i);
                     cur=cur->nxt[v^1];
              }
              else
                     cur=cur->nxt[v];
       }
       return s;
}
```

3 Geometry

3.1 Closest Points

```
ll closepair(int l,int h,int p)
{
       if(l==h)
       {
              vec[p].pb(ar[l]);
              return inf;
       }
       if(l==h-1)
       {
              if(ar[l].y>=ar[h].y)
              {
```

```
                vec[p].pb(ar[l]);
                vec[p].pb(ar[h]);
        }
        else
        {
                vec[p].pb(ar[h]);
                vec[p].pb(ar[l]);
        }
        return dist(ar[l],ar[h]);
}
int i,j,c=0;
int mid = l+(h-l)/2;
ll d1=closepair(l,mid,2*p+1);
ll d2=closepair(mid+1,h,2*p+2);
ll d=min(d1,d2);
for(i=0,j=0;i<vec[2*p+1].size() && j<vec[2*p+2].size();)
{
        if(vec[2*p+1][i].y>=vec[2*p+2][j].y)
        {
                if(abs(vec[2*p+1][i].x-ar[mid].x)<=d)
                        vec[p].pb(vec[2*p+1][i]);
                i++;
        }
        else
        {
                if(abs(vec[2*p+2][j].x-ar[mid].x)<=d)
                        vec[p].pb(vec[2*p+2][j]);
                j++;
        }
}
while(i<vec[2*p+1].size())
{
        if(abs(vec[2*p+1][i].x-ar[mid].x)<=d)
                vec[p].pb(vec[2*p+1][i]);
        i++;
}
while(j<vec[2*p+2].size())
{
        if(abs(vec[2*p+2][j].x-ar[mid].x)<=d)
                vec[p].pb(vec[2*p+2][j]);
        j++;
}
for(i=0;i<vec[p].size();++i)
{
        c=0;
                for(j=i+1;j<vec[p].size() && c<=16;++j,c++)
                        d=min(d,dist(vec[p][i],vec[p][j]));
        }
        return d;
}
```

3.2 Convex Hull Trick

```
struct line
{
        ld m;
        ld c;
        ll ind;
};
line ar[M];
vector<line> st[2*M];
ld meet(line a,line b)
{
        return (b.c-a.c)/(a.m-b.m);
}
bool check(line a,line b,line k)
{
        // to check if meeting pt. of l1,l3 is to the left of l1,l3
        ld x13=meet(a,k);
        ld x12=meet(a,b);
        if(x13<=x12) return 1;
        else return 0;
}
bool cmp(line a,line b)
{
        // compare function to sort lines as per slope and y-intercept
        if(a.m>b.m) return 1;
        else if(a.m==b.m && a.c>b.c) return 1;
        return 0;
}
void build(ll l,ll h,ll p)
{
        // each segtree node has a stack st associated for the lines with
            indices l to r
        // Now ub array stores for each line the maximum x co-ordinate
            till which it gives
        //            minimum value of mx+c
        if(l==h)
```

```
{
        st[p].pb(ar[l+1]);
        ub[p].pb(1e9);
        return;
}
ll mid=l+(h-l)/2;
build(l,mid,2*p+1);
build(mid+1,h,2*p+2);
for(int i=0;i<st[2*p+1].size();i++)
{
        // push the same stack as that of left child 2*p+1
        // later check for right child 2*p+2
        // adding them later causes no issue as their slopes will
            be all less
        // than the slopes of lines in left child
        st[p].pb(st[2*p+1][i]);
        ub[p].pb(ub[2*p+1][i]);
}
ll lg;
ld mel;
for(int i=0;i<st[2*p+2].size();i++)
{
        lg=st[p].size()-1;
        while(lg>=1 && check(st[p][lg-1],st[p][lg],st[2*p+2][i]))
        {
                // pop the line l2 at top of stack if meeting point
                    of l1,the second line
                // next to top and l3,the line to be added is at
                    left of intersection of l1,l2
                st[p].pop_back();
                ub[p].pop_back();
                lg--;
        }
        mel=meet(st[p].back(),st[2*p+2][i]);
        if(lg==0 && mel<=0)
        {
                // check if there is only one line and the next line
                // even carries greater priority than the one
                    present
                st[p].pop_back();
                ub[p].pop_back();
        }
        else
                ub[p][lg]=mel;
        st[p].pb(st[2*p+2][i]);
                ub[p].pb(1e9);
        }
}
ld query(ll l,ll h,ll p,ll ind)
{
        if(l>=ql && h<=qh)
        {
                /* log N search (if all queries aren't available online)
                   auto it =
                       lower_bound(ub[p].begin(),ub[p].end(),pos[ind])-ub[p].begin
                   return pos[ind]*(st[p][it].m)+st[p][it].c; */
                // below is way to tackle offline queries
                for(int i=ls[p];i<ub[p].size();i++)
                {
                        if(pos[ind]>ub[p][i])
                                continue;
                        ls[p]=i;
                        return pos[ind]*(st[p][i].m)+st[p][i].c;
                }
        }
        else if(l>qh || h<ql)
                return 3e18;
        ll mid=l+(h-l)/2;
        return min(query(l,mid,2*p+1,ind),query(mid+1,h,2*p+2,ind));
}
```

3.3 Geometry

```
struct pt
{
    ll x, y;
};
double dist(pt p1, pt p2)
{
    return (p1.x - p2.x)*(p1.x - p2.x) +
          (p1.y - p2.y)*(p1.y - p2.y);
}
// orientation
ll orientation(pt p,pt q,pt r)
{
        ll v = (q.y-p.y)*(r.x-q.x)-(r.y-q.y)*(q.x-p.x);
        if(v==0) return 0;
        else if(v<0) return 2;
```

```
        else return 1;
}
double distpointtoline(pt a,pt b,pt c)
{
        ll ma = a.y-b.y;
        ll mb = b.x-a.x;
        ll mc = 0;
        mc -= ma*a.x + mb*a.y;
        return abs(ma*c.x+mb*c.y+mc)/sqrt(ma*ma+mb*mb);
}
// area of polygon
ll cmp(pt p1,pt p2)
{
        ll v = orientation(p0,p1,p2);
        if(v==0)
        {
                if(dist(p0,p1)>dist(p0,p2))
                        return 0;
                else return 1;
        }
        else
        {
                if(v==2) return 1;
                else return 0;
        }
}
double area_of_polygon()
{
        if(n < 3) return 0;
        ll i,j;
        ll miny = ar[1].y;
        ll mn = 1;
        for(i=1;i<=n;++i)
        {
                if(ar[i].y<miny || (ar[i].y==miny && ar[i].x<ar[mn].x))
                {
                        mn=i;
                        miny=ar[i].y;
                }
        }
        swap(ar[1],ar[mn]);
        p0=ar[1];
        sort(ar+2,ar+1+n,cmp);
        ll o;
        double sum = 0;
        for(i=1;i<=n;++i)
        {
                if(i == n) o = 1;
                else o = i+1;
                sum = (sum + ar[o].x*ar[i].y - ar[o].y*ar[i].x);
        }
        return abs(sum/2);
}
// no. of points on straight line between p and q
int getBoundaryCount(pt p,pt q)
{
    // Check if line parallel to axes
    if (p.x==q.x)
        return abs(p.y - q.y) - 1;
    if (p.y == q.y)
        return abs(p.x-q.x) - 1;
    return __gcd(abs(p.x-q.x),abs(p.y-q.y))-1;
}
// lines intersect check
bool onSegment(pt p, pt q, pt r)
{
    if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
       return 1;
    return 0;
}
ll is_intersect(pt p1,pt q1,pt p2,pt q2)
{
        ll p = orientation(p1,q1,p2);
        ll q = orientation(p1,q1,q2);
        ll r = orientation(p2,q2,p1);
        ll s = orientation(p2,q2,q1);
        if(p!=q && r!=s) return 1;
        else if(p == 0 && onSegment(p1,p2,q1))
                return 1;
        else if(q == 0 && onSegment(p1,q2,q1))
                return 1;
        else if(r == 0 && onSegment(p2,p1,q2))
                return 1;
        else if(s == 0 && onSegment(p2,q1,q2))
                return 1;
        else return 0;
}
// st stores the convexhull points
void convexhull()
```

```
{
	if(n < 3) return;
	ll i,j,miny = ar[1].y,mn = 1;
	for(i=1;i<=n;++i)
	{
		if(ar[i].y<miny || (ar[i].y==miny && ar[i].x<ar[mn].x))
		{
			mn=i;
			miny=ar[i].y;
		}
	}
	ll m = 1;
	st[h++] = ar[mn];
	ar[0] = ar[mn];
	p0 = ar[mn];
	swap(ar[mn],ar[1]);
	sort(ar+2,ar+1+n,cmp);
	for(i=2;i<=n;++i)
	{
		while(i < n && orientation(p0,ar[i],ar[i+1])==0)
			++i;
		ar[m] = ar[i];
		m++;
	}
	if(m < 3) return;
	st[h++] = ar[1];
	st[h++] = ar[2];
	for(i=3;i<m;++i)
	{
		while(h>=2 && orientation(st[h-2],st[h-1],ar[i])!=2)
			h--;
		st[h++] = ar[i];
	}
}
```

4 Graph Algorithms

4.1 Bellman Ford

```
ll bellmanford()
{
	ll g,i,w,u,v,j;
	for(i=1;i<=n;i++)
		val[i]=1e18;
	g = 1;
	val[p]=0;
	for(i=1;i<=n && g;i++)
	{
		g = 0;
		for(j=0;j<m;j++)
		{
			w = edges[j].ff;
			u = edges[j].ss.ff;
			v = edges[j].ss.ss;
			if(val[v]>val[u]+w)
			{
				g++;
				val[v]=val[u]+w;
			}
		}
		if(i == n && g)
		{
			cout << "negative cycle exists" << endl;
			return 0;
		}
	}
	for(i=1;i<=n;i++)
		cout << val[i] << " ";
	return 1;
}
```

4.2 Binary Lifting

```
int mat[1000005][21],mrk[1000005],lv[1000005],lg[1000005];
vector<int> adj[1000005];
void dfs(int u,int p,int l)
{
	mat[u][0]=p;
	lv[u]=l;
	for(int i=0;i<adj[u].size();i++)
	{
		if(adj[u][i]!=p)
			dfs(adj[u][i],u,l+1);
	}
}
```

```
int getpr(int u,int k)
{
	if(lv[u]<k)
		return 0;
	for(int i=20;i>=0 && k>0;i--)
	{
		int s = (1<<i);
		if(k>=s)
		{
			u = mat[u][i];
			k -= s;
		}
	}
	return u;
}
void setparents()
{
	ll tg=1;
	ll j=0,i;
	for(i=1;i<=1000000;i++)
	{
		if(tg==i)
		{
			lg[i]=j;
			tg*=2;
			j++;
		}
		else
			lg[i]=lg[i-1];
	}
	for(j=1;(1<<j)<n;j++)
	{
		for(i=1;i<=n;i++)
			mat[i][j]=mat[mat[i][j-1]][j-1];
	}
}
```

4.3 Bridge Tree

```
const int M=3e5+6;
int lv[M],prs[M],bridge[M],vist[M],vist2[M],cmpno;
vector<int> adj[M], tree[M], ind[M];
queue<int> q[M];
```

```
int bridge_marker(int u,int ed,int p,int lev)
{
	vist[u]=1;
	lv[u]=lev;
	int v,j,min_levl=lev;
	for(int i=0;i<adj[u].size();i++)
	{
		v=adj[u][i];
		j=ind[u][i];
		if(!vist[v])
			min_levl=min(min_levl,bridge_marker(v,j,u,lev+1));
		else if(v!=p)
			min_levl=min(min_levl,lv[v]);
	}
	if(min_levl==lev)
		bridge[ed]=1;
	return min_levl;
}
void bridging(int u,int curr_comp)
{
	prs[u]=curr_comp;
	q[curr_comp].push(u);
	int i,v,j;
	while(!q[curr_comp].empty())
	{
		u=q[curr_comp].front();
		q[curr_comp].pop();
		for(i=0;i<adj[u].size();i++)
		{
			v=adj[u][i];
			j=ind[u][i];
			if(vist2[v]) continue;
			if(bridge[j])
			{
				cmpno++;
				tree[curr_comp].pb(cmpno);
				tree[cmpno].pb(curr_comp);
				bridging(v,cmpno);
			}
			else
			{
				prs[v]=curr_comp;
				q[curr_comp].push(v);
				vist2[v]=1;
			}
```

```
		}
	}
}
```

4.4 Centroid Decomposition

```
ll lv[XV],sz[XV],pr[XV];
vector<ll> adj[XV];
vector<ll> tmp[XV];
ll sizespecify(ll u,ll p)
{
	ll c=1;
	sz[u]=1;
	for(ll i=0;i<tmp[u].size();i++)
	{
		ll j=tmp[u][i];
		if(j!=p)
		{
			c+=sizespecify(j,u);
			sz[u]+=sz[j];
		}
	}
	return c;
}
ll getcentroid(ll u,ll p,ll s)
{
	for(ll i=0;i<tmp[u].size();i++)
	{
		ll j=tmp[u][i];
		if(j!=p && sz[j]>s/2)
			return getcentroid(j,u,s);
	}
	return u;
}
void decompose(ll u, ll p)
{
	ll cs = sizespecify(u,p);
	ll cnt = getcentroid(u,p,cs);
	pr[cnt]=p;
	lv[cnt]=lv[p]+1;
	for(ll i=0;i<tmp[cnt].size();i++)
	{
		ll j=tmp[cnt][i];
		for(ll k=0;k<tmp[j].size();k++)
		{
			if(tmp[j][k]==cnt)
				tmp[j].erase(tmp[j].begin()+k);
		}
		decompose(j,cnt);
	}
	tmp[cnt].clear();
}
```

4.5 DSU on Trees

```
// query : find the number of distinct colors at a particular distance
//        t from u in subtree of u
void dfs(int u,int p,int l)
{
	sz[u]=1;
	lv[u]=l;
	for(int i=0;i<adj[u].size();i++)
	{
		if(adj[u][i]!=p)
		{
			dfs(adj[u][i],u,l+1);
			sz[u]+=sz[adj[u][i]];
		}
	}
}
void add(int u,int p,int r)
{
	if(r>0) st[lv[u]].insert(col[u]);
	else st[lv[u]].erase(col[u]);
	for(int i=0;i<adj[u].size();i++)
		if(adj[u][i]!=p && !big[adj[u][i]]) add(adj[u][i],u,r);
}
int dsu(int u,int p,int r)
{
	int bg=0,i;
	for(i=0;i<adj[u].size();i++)
		if(adj[u][i]!=p && sz[adj[u][i]]>sz[bg]) bg=adj[u][i];
	big[bg]=1;
	for(i=0;i<adj[u].size();i++)
		if(adj[u][i]!=p && adj[u][i]!=bg) dsu(adj[u][i],u,-1);
	if(bg) dsu(bg,u,1);
```

```
        add(u,p,1);
        for(i=0;i<que[u].size();i++)
                ans[que[u][i].ff]=st[lv[u]+que[u][i].ss].size();
        big[bg]=0;
        if(r==-1) add(u,p,-1);
}
```

4.6 Dinics

```
// struct edge{int a,b;ll c,f;
//   edge(int u,int v,ll cap):a(u),b(v),c(cap),f(0){}};
// struct flow{
//   const static ll inf = 1e18;//set s=source,t=sink,nodeCt
//   int Dlevel[L],Dptr[L],s,t,nodeCt;
//   queue<int> Q;vector<edge> E;vi ad[L];
//   void addEdge(int a,int b,int c=1,bool directed=1){
//     if(a==b)return ;//1 based index
//     ad[a].pb(sz(E)),E.pb(edge(a,b,c));
//     ad[b].pb(sz(E)),E.pb(edge(b,a,directed?0:c)); }
//   bool Dbfs(void){FEN(i,nodeCt)Dlevel[i]=0;Q.push(s),Dlevel[s]=1;
//     while(!Q.empty()){int levelsz=sz(Q),v;
//       while(levelsz--){v=Q.front();Q.pop();
//      for(auto &e:ad[v]) if(!Dlevel[E[e].b]&&E[e].f<E[e].c){
//         Dlevel[E[e].b]=Dlevel[v]+1;
//         Q.push(E[e].b);}}}
//     return Dlevel[t]>0;}//by default edges are undirected
//   ll Ddfs(int x,ll flow){if(!flow) return 0;if(x==t) return flow;
//     for(int &pt=Dptr[x];pt<sz(ad[x]);++pt){
//       int e=ad[x][pt];//call dinic()
//       if(Dlevel[E[e].b]==Dlevel[x]+1){
//      if(ll pushed=Ddfs(E[e].b,min(flow,E[e].c-E[e].f))){
//        E[e].f+=pushed;E[e^1].f-=pushed;return pushed;}}}
//     return 0 ;}
//   ll dinic(void){ll flow=0;while(Dbfs()){FEN(i,nodeCt)Dptr[i]=0;
//       while(ll pushed=Ddfs(s,inf)) flow += pushed ;} return flow ;}
struct edge
{
  ll a, b, capacity, flow;
};
const int MAXN = 5000;
ll INF = 1e18, dinics_level[MAXN], dinics_ptr[MAXN], N, source, sink, M;
vector<edge>E;
vector<ll>dinics_adj[MAXN];
void add_edge(ll a, ll b, ll capacity)
{
  edge e1 = {a, b, capacity, 0};
  edge e2 = {b, a, 0, 0};
  dinics_adj[a].pb(sz(E));
  E.pb(e1);
  dinics_adj[b].pb(sz(E));
  E.pb(e2);
}
bool dinics_bfs()
{
  for(int i=0;i<N;i++) dinics_level[i] = -1;
  dinics_level[source] = 0;
  queue<ll>q;
  q.push(source);
  while(!q.empty())
  {
    ll u = q.front();
    q.pop();
    for(auto it : dinics_adj[u])
    {
      if(dinics_level[E[it].b] < 0 && E[it].flow < E[it].capacity)
      {
        dinics_level[E[it].b] = dinics_level[u] + 1;
        q.push(E[it].b);
      }
    }
  }
  return dinics_level[sink] > 0;
}
ll dinics_dfs(ll curr_node, ll flow)
{
  if(!flow) return 0;
  if(curr_node == sink) return flow;
  for(int pt = dinics_ptr[curr_node]; pt < sz(dinics_adj[curr_node]);
      ++pt)
  {
    ll e = dinics_adj[curr_node][pt];
    if(dinics_level[E[e].b] == dinics_level[curr_node] + 1)
    {
      if(ll pushed = dinics_dfs(E[e].b, min(flow, E[e].capacity -
          E[e].flow)))
      {
        E[e].flow += pushed;
        E[e^1].flow -= pushed;
```

```
      return pushed;
     }
    }
  }
  return 0;
}
ll dinics()
{
  ll flow = 0;
  while(dinics_bfs())
  {
    for(int i=0;i<N;i++) dinics_ptr[i] = 0;
    while(ll pushed = dinics_dfs(source, INF)) flow += pushed;
  }
  return flow;
}
```

4.7 Djikstra

```
ll n;
ll djikstra(ll s,ll t)
{
       ll i,j,k,u;
       for(i=1;i<=n;i++)
       {
              mrk[i]=0;
              pr[i]=0;
              far[i]=1e18;
              sto[i]=1e18;
       }
       far[s]=0;
       priority_queue< pi,vector<pi>,greater<pi> > pq;
       pq.push(pi(0,s));
       while(!pq.empty())
       {
              u=pq.top().second;
              pq.pop();
              if(!mrk[u])
              {
                     mrk[u]=1;
                     for(i=0;i<adj[u].size();i++)
                     {
                            j=adj[u][i].second;
                            k=adj[u][i].first;
                            if(!mrk[j])
                            {
                                   if(far[j]>far[u]+k ||
                                       (far[j]==far[u]+k && sto[j]>k))
                                   {
                                          far[j]=far[u]+k;
                                          sto[j]=k;
                                          pr[j]=ind[u][i];
                                          pq.push(pi(far[j],j));
                                   }
                            }
                     }
              }
       }
       return far[t];
}
```

4.8 Floyd Warshall

```
//if(i == j) dist[i][j] = 0 else dist[i][j] = 1e18; if edge(a, b) = c
    exists then dist[a][b] = dist[b][a] = c
for(int k=0;k<n;k++)
       for(int i=0;i<n;i++)
              for(int j=0;j<n;j++)
                     dist[i][j] = min(dist[i][j], dist[i][k] +
                         dist[k][j]); // dist[i][j] is shortest dist
                         between i and j
```

4.9 HLD

```
vector <int> adj[N], costs[N], indexx[N];
int baseArray[N], ptr;
int chainNo, chainInd[N], chainHead[N], posInBase[N];
int depth[N], pa[LN][N], otherEnd[N], subsize[N];
int st[N*6], qt[N*6];
/*
 * make_tree:
 * Used to construct the segment tree. It uses the baseArray for
     construction
 */
```

```
void make_tree(int cur, int s, int e) {
	if(s == e-1) {
		st[cur] = baseArray[s];
		return;
	}
	int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
	make_tree(c1, s, m);
	make_tree(c2, m, e);
	st[cur] = st[c1] > st[c2] ? st[c1] : st[c2];
}
/*
 * update_tree:
 * Point update. Update a single element of the segment tree.
 */
void update_tree(int cur, int s, int e, int x, int val) {
	if(s > x || e <= x) return;
	if(s == x && s == e-1) {
		st[cur] = val;
		return;
	}
	int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
	update_tree(c1, s, m, x, val);
	update_tree(c2, m, e, x, val);
	st[cur] = st[c1] > st[c2] ? st[c1] : st[c2];
}
/*
 * query_tree:
 * Given S and E, it will return the maximum value in the range [S,E)
 */
void query_tree(int cur, int s, int e, int S, int E) {
	if(s >= E || e <= S) {
		qt[cur] = -1;
		return;
	}
	if(s >= S && e <= E) {
		qt[cur] = st[cur];
		return;
	}
	int c1 = (cur<<1), c2 = c1 | 1, m = (s+e)>>1;
	query_tree(c1, s, m, S, E);
	query_tree(c2, m, e, S, E);
	qt[cur] = qt[c1] > qt[c2] ? qt[c1] : qt[c2];
}
/*
 * query_up:
 * It takes two nodes u and v, condition is that v is an ancestor of u
 * We query the chain in which u is present till chain head, then move to
     next chain up
 * We do that way till u and v are in the same chain, we query for that
     part of chain and break
 */
int query_up(int u, int v) {
	if(u == v) return 0; // Trivial
	int uchain, vchain = chainInd[v], ans = -1;
	// uchain and vchain are chain numbers of u and v
	while(1) {
		uchain = chainInd[u];
		if(uchain == vchain) {
			// Both u and v are in the same chain, so we need
			    to query from u to v, update answer and break.
			// We break because we came from u up till v, we
			    are done
			if(u==v) break;
			query_tree(1, 0, ptr, posInBase[v]+1,
			    posInBase[u]+1);
			// Above is call to segment tree query function
			if(qt[1] > ans) ans = qt[1]; // Update answer
			break;
		}
		query_tree(1, 0, ptr, posInBase[chainHead[uchain]],
		    posInBase[u]+1);
		// Above is call to segment tree query function. We do
		    from chainHead of u till u. That is the whole chain
		    from
		// start till head. We then update the answer
		if(qt[1] > ans) ans = qt[1];
		u = chainHead[uchain]; // move u to u's chainHead
		u = pa[0][u]; //Then move to its parent, that means we
		    changed chains
	}
	return ans;
}
/*
 * LCA:
 * Takes two nodes u, v and returns Lowest Common Ancestor of u, v
 */
int LCA(int u, int v) {
	if(depth[u] < depth[v]) swap(u,v);
	int diff = depth[u] - depth[v];
	for(int i=0; i<LN; i++) if( (diff>>i)&1 ) u = pa[i][u];
```

```
        if(u == v) return u;
        for(int i=LN-1; i>=0; i--) if(pa[i][u] != pa[i][v]) {
                u = pa[i][u];
                v = pa[i][v];
        }
        return pa[0][u];
}
void query(int u, int v) {
        /*
         * We have a query from u to v, we break it into two queries, u to
             LCA(u,v) and LCA(u,v) to v
         */
        int lca = LCA(u, v);
        int ans = query_up(u, lca); // One part of path
        int temp = query_up(v, lca); // another part of path
        if(temp > ans) ans = temp; // take the maximum of both paths
        printf("%d\n", ans);
}
/*
 * change:
 * We just need to find its position in segment tree and update it
 */
void change(int i, int val) {
        int u = otherEnd[i];
        update_tree(1, 0, ptr, posInBase[u], val);
}
/*
 * Actual HL-Decomposition part
 * Initially all entries of chainHead[] are set to -1.
 * So when ever a new chain is started, chain head is correctly assigned.
 * As we add a new node to chain, we will note its position in the
     baseArray.
 * In the first for loop we find the child node which has maximum
     sub-tree size.
 * The following if condition is failed for leaf nodes.
 * When the if condition passes, we expand the chain to special child.
 * In the second for loop we recursively call the function on all normal
     nodes.
 * chainNo++ ensures that we are creating a new chain for each normal
     child.
 */
void HLD(int curNode, int cost, int prev) {
        if(chainHead[chainNo] == -1) {
                chainHead[chainNo] = curNode; // Assign chain head
        }
        chainInd[curNode] = chainNo;
        posInBase[curNode] = ptr; // Position of this node in baseArray
            which we will use in Segtree
        baseArray[ptr++] = cost;
        int sc = -1, ncost;
        // Loop to find special child
        for(int i=0; i<adj[curNode].size(); i++) if(adj[curNode][i] !=
            prev) {
                if(sc == -1 || subsize[sc] < subsize[adj[curNode][i]]) {
                        sc = adj[curNode][i];
                        ncost = costs[curNode][i];
                }
        }
        if(sc != -1) {
                // Expand the chain
                HLD(sc, ncost, curNode);
        }
        for(int i=0; i<adj[curNode].size(); i++) if(adj[curNode][i] !=
            prev) {
                if(sc != adj[curNode][i]) {
                        // New chains at each normal node
                        chainNo++;
                        HLD(adj[curNode][i], costs[curNode][i], curNode);
                }
        }
}

/*
 * dfs used to set parent of a node, depth of a node, subtree size of a
     node
 */
void dfs(int cur, int prev, int _depth=0) {
        pa[0][cur] = prev;
        _depthpth[cur] = _depth;
        subsize[cur] = 1;
        for(int i=0; i<adj[cur].size(); i++)
                if(adj[cur][i] != prev) {
                        otherEnd[indexx[cur][i]] = adj[cur][i];
                        dfs(adj[cur][i], cur, _depth+1);
                        subsize[cur] += subsize[adj[cur][i]];
                }
}

main()
{
```

```
	chainNo = 0;
	dfs(root, -1); // We set up subsize, depth and parent for each node
	HLD(root, -1, -1); // We decomposed the tree and created baseArray
	make_tree(1, 0, ptr); // We use baseArray and construct the needed
	    segment tree

	// Below Dynamic programming code is for LCA.
	for(int i=1; i<LN; i++)
		for(int j=0; j<n; j++)
			if(pa[i-1][j] != -1)
				pa[i][j] = pa[i-1][pa[i-1][j]];
}
```

4.10 LCA

```
ll srt[100009],n,t[200009][18],s[200009][18],lv[100009];
ll lg[1000009],tc,h;
vector<ll> adj[100009];
ll dfs(ll u,ll p,ll l)
{
	srt[u] = tc++;
	t[h][0] = l;
	s[h++][0] = u;
	lv[u] = l;
	ll c = 1;
	for(ll i = 0; i < adj[u].size() ; i++)
	{
		ll j = adj[u][i];
		if(j != p)
		{
			dfs(j,u,l+1);
			t[h][0] = l;
			s[h++][0] = u;
		}
	}
	tc++;
	return h;
}
ll lca(ll u,ll v)
{
	ll l = min(srt[u],srt[v]);
	ll r = max(srt[u],srt[v]);
	ll j = lg[r-l+1];
	if(t[l][j] <= t[r-(1<<j)+1][j])
		return s[l][j];
	else
		return s[r-(1<<j)+1][j];
}
ll dist(ll u,ll v)
{
	return lv[u]+lv[v]-2*lv[lca(u,v)];
}
void pre_lca_computation()
{
	ll tg = 1,j = 0, i;
	for(i = 1; i <= 1000000 ; i++)
	{
		if(tg == i)
		{
			lg[i] = j;
			tg *= 2;
			j++;
		}
		else
			lg[i] = lg[i-1];
	}
	ll sz = dfs(1,0,0);
	for(j = 1; (1<<j) <= sz ; j++)
	{
		for(i = 0; (i+(1<<j)-1) < sz ; i++)
		{
			if(t[i][j-1] < t[i+(1<<(j-1))][j-1])
			{
				t[i][j] = t[i][j-1];
				s[i][j] = s[i][j-1];
			}
			else
			{
				t[i][j] = t[i+(1<<(j-1))][j-1];
				s[i][j] = s[i+(1<<(j-1))][j-1];
			}
		}
	}
}
```

4.11 MST

```
ll parent(ll i) // return root parent of an element
{
    while(par[i] != i)
    {
        par[i] = par[par[i]];
        i = par[i];
    }
    return i;
}
void connect(ll a,ll b) // connects node a and b in same subset
{
    ll pa = parent(a);
    ll pb = parent(b);
    if(size[pa] < size[pb])
    {
        par[pa] = par[pb];
        size[pb] += size[pa];
    }
    else
    {
        par[pb] = par[pa];
        size[pa] += size[pb];
    }
}
ll kruskal()
{
    ll cost = 0;       // m is no of edges
    for(int i=0;i<m;i++)
    {
        ll x1 = e[i].second.first; // e contains first element as weight
             of edge and second as a pair of nodes connecting them
        ll x2 = e[i].second.second;
        if(parent(x1) != parent(x2))
        {
            cost += e[i].first;
            connect(x1, x2);
        }
    }
    return cost;
}
```

4.12 Mo's on trees

```
// query : find in the subtree of u, the no. of various colors
//               that occurs k times
bool cmp(pii a,pii b)
{
       pi f=a.ss;
       pi s=b.ss;
       return (f.ff==s.ff) ? (f.ss<s.ss) : (f.ff<s.ff);
}
void add(int p)
{
       if(p>=1 && p<=n)
       {
              cnt[ar[p]]++;
              if(cnt[ar[p]]<=n) qnt[cnt[ar[p]]]++;
       }
}
void remove(int p)
{
       if(p>=1 && p<=n)
       {
              cnt[ar[p]]--;
              if(cnt[ar[p]]<n) qnt[cnt[ar[p]]+1]--;
       }
}
void dfs(int u,int p)
{
       srt[u]=tc++;
       ar[h++]=col[u];
       for(int i=0;i<adj[u].size();i++)
       {
              int j=adj[u][i];
              if(j!=p) dfs(j,u);
       }
       fin[u]=tc-1;
}
main()
{
       // general graph input, n nodes and colors
       tc=h=1;
       dfs(1,0);
       s=ceil(sqrt(n));
       for(i=1;i<=q;i++)
       {
              cin >> uv[i] >> k[i];
```

```
            xdj.pb(pii(i,pi(srt[uv[i]]/s+1,fin[uv[i]]/s+1)));
        }
        sort(xdj.begin(),xdj.end(),cmp);
        cl=cr=0;
        for(i=0;i<xdj.size();i++)
        {
            j=xdj[i].ff;
            l=srt[uv[j]];
            r=fin[uv[j]];
            while(cl>l) add(--cl);
            while(cl<l) remove(cl++);
            while(cr>r) remove(cr--);
            while(cr<r) add(++cr);
            otp[j]=qnt[k[j]];
        }
}
```

4.13 Strongly Connected Comp

```
const int Nnodes = 100005;
vector<ll>adj[Nnodes], adj2[Nnodes]; //normal & reverse adjaceny list
vector<ll>arr1, temp;
bool vis[Nnodes], vis2[Nnodes];
void dfs1(ll node)
{
        vis[node] = true;
        for(int i=0;i<adj[node].size();i++)
        {
            if(!vis[adj[node][i]])
            {
                dfs1(adj[node][i]);
            }
        }
        arr1.pb(node);
}
void dfs2(ll node)
{
        vis2[node] = true;
        temp.pb(node);
        for(int i=0;i<adj2[node].size();i++)
        {
            if(!vis2[adj2[node][i]])
            {
                dfs2(adj2[node][i]);
            }
        }
}
for(int i=0;i<n;i++) if(!vis[i]) dfs1(i);
reverse(arr1.begin(), arr1.end());
for(int i=0;i<arr1.size();i++)
{
        if(!vis2[arr1[i]])
        {
            dfs2(arr1[i]);
            sort(temp.begin(), temp.end());
            for(int i=0;i<temp.size();i++)
            {
                cout << temp[i] + 1 << " ";
            }
            cout << endl; // scc are printed with a space between them
                and different components are separated by a newline
            temp.clear();
        }
}
```

4.14 Top Sort

```
vector<ll>adj[MAXN], adj2[MAXN], tsort; //adj2 is reverse
void top_sort_kahn()
{
    queue<ll>q;
    for(int i=0;i<26;i++)
    {
        indeg[i] = adj2[i].size();
        if(adj2[i].size() == 0 && adj[i].size() != 0)
        {
            q.push(i);
        }
    }
    while(!q.empty())
    {
        ll f = q.front();
        tsort.pb(f);
        val1++;
        q.pop();
        for(auto it:adj[f])
```

```
        {
            indeg[it]--;
            if(indeg[it] == 0)
            {
                q.push(it);
            }
        }
    }
}
```

5 Maths

5.1 FFT

```
#pragma GCC optimize("Ofast")
#pragma GCC
    target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
#pragma GCC optimize("unroll-loops")
#include<stdio.h>
#include<stdlib.h>
#define lint long long
#define swap(a, b) a^=b,b^=a,a^=b

const int m = 663224321;
#define MAX_N 262144

int omega[20][524288];
int temp[MAX_N];

typedef struct Polynomial {
        int len;
        int *arr;
} polynomial;

int power( int x, int y ) {
        int ans = 1;
        for( ; y; y>>=1, x = 1ll*x*x %m )
                if( y&1 )
                        ans = 1ll*ans*x %m;
        return ans;
}
```

```
void disp( int *a, int n )
{
        for( int i=0; i<n; ++i )
                printf("%d ", a[i]);
        printf("\n");
}
void ntt( int *a, int n, int inv ) {
        register int i,li,j,k,w,e,o;
        for( i=1, j=n>>1; i<n-1; ++i )
        {
                if ( i<j )
                        swap(a[i], a[j]);
                k = n>>1;
                for( ; j>=k; j-=k, k>>=1);
                j += k;
        }
        for( li=1,i=1; i<n; i<<=1,++li )
                for( j=0; j<n; j+=(i<<1) )
                        for( k=j; k<j+i; ++k )
                                o = 1ll*omega[li][k-j]*a[k+i] %m, e=a[k],
                                    a[k] = (e+o)%m, a[k+i]=(e-o+m)%m;
        if( inv == -1 )
        {
                for( i=1; i<(n>>1); ++i )
                        swap(a[i],a[n-i]);
                int invn = power(n,m-2);
                for( i=0; i<n; ++i )
                        a[i] = 1ll*a[i]*invn %m;
        }
}

polynomial *invert( polynomial *a, int n ) {
        polynomial *b = (polynomial*)malloc(sizeof(polynomial));
        for( b->len=1; b->len<n; b->len<<=1 );
        b->len <<= 1;
        b->arr = (int *)malloc(sizeof(int)*(b->len));
        b->arr[0] = power(a->arr[0], m-2);

        for( int j=1; j<n; j<<=1 ) {
                j <<= 1;
                for( int i=(j>>1); i<j; ++i ) b->arr[i] = 0;
                for( int i=0; i<j; ++i ) b->arr[i+j] = b->arr[i];

                for( int i=0; i<j && i<a->len; ++i ) temp[i] = a->arr[i];
                for( int i=a->len; i<j; ++i ) temp[i] = 0;
```

```
                ntt(b->arr+j, j, 1);
                ntt(temp, j, 1);
                for( int i=0; i<j; ++i )
                        temp[i] = 1ll*temp[i]*b->arr[i+j] %m;
                ntt(temp, j, -1);

                j >>= 1;
                for( int i=0; i<j; ++i )
                        temp[i] = temp[i+j], temp[i+j] = 0;

                j <<= 1;
                ntt(temp, j, 1);
                for( int i=0; i<j; ++i )
                        temp[i] = 1ll*temp[i]*b->arr[i+j] %m;
                ntt(temp, j, -1);

                j >>= 1;
                for( int i=0; i<j; ++i )
                        b->arr[j+i] = (temp[i] == 0)?(0):(m-temp[i]);
        }
        b->arr = realloc(b->arr, sizeof(int)*(b->len=n));
        return b;
}

polynomial *create_polynomial( int n ) {
        polynomial *c = (polynomial*)malloc(sizeof(polynomial));
        c->arr = (int *)malloc(sizeof(int)*(c->len = n));
        return c;
}

void Prepare()
{
        for (int i = 0; i < 2; i ++) ntt[i].set(magic[i], 3);
        for (int i = 0; i < 3; i ++)
                for (int j = i + 1; j < 3; j ++)
                        Inv[i][j] = mypow(magic[i], magic[j] - 2, magic[j]);
}

int CRT(int *a)
{
        int x[3];
        for (int i = 0; i < 2; i ++)
        {
                x[i] = a[i];
                for (int j = 0; j < i; j ++)
                {
                        int t = (x[i] - x[j] + magic[i]) % magic[i];
                        if (t < 0) t += magic[i];
                        x[i] = 1LL * t * Inv[j][i] % magic[i];
                }
        }
        int sum = 1, ret = x[0] % Mod;
        for (int i = 1; i < 2; i ++)
        {
                sum = 1LL * sum * magic[i - 1] % Mod;
                ret += 1LL * x[i] * sum % Mod;
                if(ret >= Mod) ret -= Mod;
        }
        return ret;
}

void linearsieve() {
        vector<int> primes;
        // cnt, func
        for( int i=2; i<n; ++i )
        {
                if( cnt[i] == 0 )
                {
                        cnt[i] = 1;
                        func[i] = ;//
                        primes.push_back(i)
                }
                for( int p : primes )
                {
                        int ip = i*p;
                        if( ip >= n ) break;
                        if( i%p == 0 )
                        {
                                cnt[ip] = cnt[i]+1;
                                func[ip] = func[i]* //
                                break;
                        }
                        cnt[ip] = 1;
                        func[ip] = func[i]*func[p];
                }
        }
}

// f is known. a[n] = sum_{i=1}^{n-1} a[i]*f[n-i]
```

```
	f[0] = 1;
	for( int i=1; i<n; ++i )
		f[i] = 1ll*i*f[i-1] %m;

	a[0] = 0;
	a[1] = 1;
	a[2] = 1;
	for( int i=3; i<n; ++i )
	{
		a[i] = a[i-1];
		for( int j=1; ; )
		{
			multiply( f + j + 1, a + i - j - 1, j );
			j <<= 1;
			for( int k = i; k < i+j-1; ++k )
				a[k] = (a[k] + c1[k-i]) %m;
			if( (j>>1)&(i-2) )
				break;
		}
		// a[i] = (f[i] + m - a[i]) %m;
	}

int main() {
	for( lint i=0; i<20; ++i )
		for( lint j=0, x=1, w = power(3, (m-1)/(1<<i)); j<(1<<i);
		    ++j, x = 1ll*x*w %m )
			omega[i][j] = x;
	polynomial *c = create_polynomial(100001);


	for( int i=n-1; i>0; --i )
		seg[i] = seg[i<<1] + seg[i<<1 + 1];

	for( l+=n, r+=n; l<r; l>>=1, r>>=1 )
	{
		if( l&1 ) res += seg[l++];
		if( r&1 ) res += seg[--r];
	}

	for( i += n, seg[i] = x; i>0; i>>=1 )
		seg[i>>1] = seg[i] + seg[i^1];
}
```

5.2 Game Theory

```
Green HackenBush Game If two players are playing a game where move
allowed is to cut an edge of a (rooted) tree then for each node,set value
    of
the node to xor of (1 + value of its child) for all children of the node
    in
the tree. For leaves, value is 0. See value of root to determine the
    winner.
For a (rooted) graph, where each player can remove an edge from the
component connected to the root and one unable to remove loses. Create
bridge-tree and do same as tree except val[node] = orignalVal[node]
(numEdges[node] & 1)
StairCase Nim Stair Case from {1...N}. Ans = xor of aeven
k-Nim. One Player can reduce the size of {1...k} piles. Starting position
is loosing iff for all j belongs{0...LOGMAX}sum(i = 1 to n) ai & 2^j = 0
    mod(k+1)
```

5.3 Rabbin Miller

```
ll mulmod(ll a, ll b, ll mod)
{
    ll x = 0,y = a % mod;
    while(b)
    {
        if(b % 2 == 1) x = (x+y) % mod;
        y = (y * 2) % mod;
        b /= 2;
    }
    return x;
}
ll power(ll a, ll b, ll mod)
{
    ll ans = 1;
    while(b)
    {
        if(b % 2) ans = mulmod(ans, a, mod);
        a = mulmod(a, a, mod);
        b = b >> 1;
    }
    return ans;
}
// bool Miller(ll p, ll iteration)
```

```
// {
//     ll s, i;
//     if (p < 2) return 0;
//     int arr[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
//     for(int i=0;i<9;i++) if(p == arr[i]) return 1;
//     for(int i=0;i<9;i++) if(p % arr[i] == 0) return 0;
//     if ( (p % 6 != 1 && p % 6 != 5) ) return 0;
//     s = p - 1;
//     while(s % 2 == 0) s /= 2;
//     for(i=0;i<iteration;i++)
//     {
//         ll a = rand() % (p-1) + 1,temp = s;
//         ll mod = power(a, temp, p);
//         while(temp != p-1 && mod != 1 && mod != p-1)
//         {
//                 mod = mulmod(mod, mod, p);
//             temp *= 2;
//         }
//         if (mod != p - 1 && temp %2 == 0) return 0;
//     }
//     return 1;
// }
bool isPrime(ll n)
{
  ll d = n - 1;
  int s = 0;
  while (d % 2 == 0)
  {
    s++;
    d >>= 1;
  }
  int a[9] = { 2, 3, 5, 7, 11, 13, 17, 19, 23};
  for(int i=0;i<9;i++) if(n == a[i]) return true;
  for(int i = 0; i < 9; i++) {
    bool comp = power(a[i], d, n) != 1;
    if(comp) for(int j = 0; j < s; j++) {
      ll fp = power(a[i], (1LL << (ll)j)*d, n);
      if (fp == n - 1) {
        comp = false;
        break;
      }
    }
    if(comp) return false;
  }
  return true;
}
```

5.4 nCrLarge

```
ll invert_mod(ll k,ll m){
if(m==0)return(k==1||k==-1)?k:0;
if(m<0)m=-m;k%=m;
if(k<0)k+=m;int neg=1;
ll p1=1,p2=0,k1=k,m1=m,q,r,temp;
while(k1>0){
q=m1/k1;r=m1%k1;
temp=q*p1+p2;p2=p1;p1=temp;
m1=k1;k1=r;neg=!neg;
}return neg?m-p2:p2;
}
// Preconditions:0<=k<=n;p>1 prime
ll choose_mod_one(ll n,ll k,ll p){
if(k<p)return choose_mod_two(n,k,p);
ll q_n,r_n,q_k,r_k,choose;
q_n=n/p;r_n=n%p;q_k=k/p;r_k=k%p;
choose=choose_mod_two(r_n,r_k,p);
choose*=choose_mod_one(q_n,q_k,p);
return choose%p;
}
// Preconditions:0<=k<=min(n,p-1);p>1 prime
ll choose_mod_two(ll n,ll k,ll p){
n%=p;if(n<k)return 0;
if(k==0||k==n)return 1;
if(k>n/2)k=n-k;ll num=n,den=1;
for(n=n-1;k>1;--n,--k)num=(num*n)%p,den=(den*k)%p;
den=invert_mod(den,p);
return (num*den)%p;
}
ll fact_exp(ll n, ll p){
ll ex=0;do{n/=p;ex+=n;
}while(n>0);return ex;
}
//returns nCk % p in O(p).n and k can be large.
ll choose_mod(ll n, ll k, ll p){
if(k<0||n<k)return 0;if(k==0||k==n)return 1;
if(fact_exp(n)>fact_exp(k)+fact_exp(n-k))return 0;
return choose_mod_one(n,k,p);
}
```

6 Strings

6.1 KMP

```
// void precompute(string s)
// {
//     f.pb(0);        // f is lps array
//     ll n = s.length();
//     for(int i=1;i<n;i++)
//     {
//         ll j = f[i - 1];
//         while(j > 0 && s[i] != s[j]) j = f[j - 1];
//         if(s[i] == s[j]) j++;
//         f.pb(j);
//     }
// }
ll kmp()
{
        ll j=0,c=0,i;
        for(i=1;pt[i]!='\0';i++)
        {
                while(pt[j]!=pt[i])
                {
                        if(j==0)
                                break;
                        j=sto[j-1];
                }
                if(pt[j]==pt[i])
                        j++;
                sto[i]=j;
        }
        ll l=i;
        j=0;
        for(i=0;str[i]!='\0';i++)
        {
                while(pt[j]!=str[i])
                {
                        if(j==0)
                                break;
                        j=sto[j-1];
                }
                if(pt[j]==str[i])
                        j++;
                if(j==l)
                {
                        c++;
                        ans.pb(i-l+2);
                        j=sto[j-1];
                }
        }
        return c;
}
```

6.2 String Hashing

```
const int MAXN = 100005;
ll mod1 = 1e9 + 7, mod2 = 1073676287;
ll hashe1[MAXN], hashe2[MAXN];
pair<ll,ll> h(ll l, ll r) // 0 based indexing
{
        if(l == 0)
        {
                return mp(hashe1[r], hashe2[r]);
        }
        else
        {
                ll var1 = (((hashe1[r] - hashe1[l - 1] + mod1) % mod1) *
                    power(41, mod1 - 1 - l, mod1) ) % mod1;
                ll var2 = (((hashe2[r] - hashe2[l - 1] + mod2) % mod2) *
                    power(53, mod2 - 1 - l, mod2) ) % mod2;
                return mp(var1, var2);
        }
}
string a; cin >> a;
hashe1[0] = hashe2[0] = a[0];
ll val1 = 41, val2 = 53;;
for(int i=1;i<a.length();i++)
{

        hashe1[i] = ((a[i] * val1) % mod1 + hashe1[i - 1]) % mod1;
        hashe2[i] = ((a[i] * val2) % mod2 + hashe2[i - 1]) % mod2;
        val1 = (val1 * 41) % mod1;
        val2 = (val2 * 53) % mod2;
}
cout << h(l, r).first << " " << h(l,r).second << endl;
```

6.3 String Trie

```
typedef struct node{
	node* nxt[26];
	bool leaf;
}node;
node* newnode()
{
	node* tmp = new node;
	for(ll i=0;i<=25;++i)
		tmp->nxt[i] = NULL;
	tmp-> leaf = 0;
	return tmp;
}
node* insert(node* root,string str)
{
	node* cur = root;
	for(int i=0;i<str.size();++i)
	{
		if(cur->nxt[str[i]-'A'] == NULL)
		{
			node* tmp = newnode();
			cur->nxt[str[i]-'A'] = tmp;
		}
		cur = cur->nxt[str[i]-'A'];
	}
	cur->leaf = 1;
	return root;
}
bool search_string(node* root,string str)
{
	node* cur = root;
	ll l = str.size();
	for(int i=0;i<l;++i)
	{
		if(cur->nxt[str[i]-'A'] == NULL)
			return 0;
		else
			cur = cur->nxt[str[i]-'A'];
	}
	if(cur->leaf == 1)
		return 1;
	else
		return 0;
}
```

```
void cleartrie(node* root)
{
	node* cur = root;
	for(int i=0;i<26;++i)
	{
		if(cur->nxt[i] != NULL)
			cleartrie(cur->nxt[i]);
	}
	root = NULL;
}
```

6.4 Z Alogrithm

```
void getZarr(string str, int Z[])
{
    int n = str.length(), L = 0, R = 0, k;
    for (int i = 1; i < n; ++i)
    {
        if (i > R)
        {
            L = R = i;
            while (R<n && str[R-L] == str[R]) R++;
            Z[i] = R-L;
            R--;
        }
        else
        {
            k = i-L;
            if (Z[k] < R-i+1) Z[i] = Z[k];
            else
            {
                L = i;
                while (R<n && str[R-L] == str[R]) R++;
                Z[i] = R-L;
                R--;
            }
        }
    }
}
```

7 Template

```
#pragma GCC optimize("Ofast")
#pragma GCC
    target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,mmx,avx,tune=native")
#pragma GCC optimize("unroll-loops")
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<pair<ll,ll> ,null_type,less<pair<ll,ll>
    >,rb_tree_tag,tree_order_statistics_node_update> ordered_set;
//cout << k << "kth smallest: " << *A.find_by_order(k-1) << endl;
//cout << "No of elements less than " << X << " are " <<
    A.order_of_key(X) << endl;
struct custom_hash {
   static uint64_t splitmix64(uint64_t x) {
       x += 0x9e3779b97f4a7c15;
       x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
       x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
       return x ^ (x >> 31);
   }

   size_t operator()(uint64_t x) const {
       static const uint64_t FIXED_RANDOM =
           chrono::steady_clock::now().time_since_epoch().count();
       return splitmix64(x + FIXED_RANDOM);
   }
};
std::ios::sync_with_stdio(false); cin.tie(0); cout.tie(0);
unordered_map<ll,ll,custom_hash> m;
Bitset - To find number of 1s set between l and r both inclusive in a
    Bitset Let the bitset be temp;
temp &= (par<<l);
temp & = (par>>(MAXN - r - 1));
cout << temp.count() << endl;
where par is bitset with all bits set and MAXN is size of bitset
```