

Bugged Coders

1 Data sctrucutres

1.1 Segment tree

```

1 int nums[]={1,3,4,5,7};
2 struct segmentTree{
3     int l, r,sum;
4     segmentTree *nodeLeft,*nodeRight;
5     segmentTree(int a, int b){
6         l=a;
7         r=b;
8         int m=(l+r)/2;
9         if(l!=r){
10             nodeLeft=new segmentTree(l,m);
11             nodeRight=new segmentTree(m+1,r);
12             sum=nodeLeft->sum+nodeRight->sum;
13         }
14         else sum=nums[l];
15     }
16     int query(int a, int b){
17         if(b<l || a>r) return 0;
18         if(a<=l && r<=b) return sum;
19         return nodeLeft->query(a,b)+nodeRight->query(a,b);
20     }
21     void update(int pos, int v){
22         if(l!=r){
23             int m=(l+r)/2;
24             if(pos<=m) nodeLeft->update(pos,v);
25             else nodeRight->update(pos,v);
26             sum=nodeLeft->sum+nodeRight->sum;
27         }
28         else sum=v;
29     }
30 };

```

1.2 Segment tree- Lazy Propagation

```

1 int nums[]={1,3,5,7,9,11};
2 struct segmentTree{

```

```

3     int l, r,sum,lazy;
4     segmentTree *nodeLeft,*nodeRight;
5     segmentTree(int a, int b){
6         l=a;
7         r=b;
8         int m=(l+r)/2;
9         lazy=0;
10        if(l!=r){
11            nodeLeft=new segmentTree(l,m);
12            nodeRight=new segmentTree(m+1,r);
13            sum=nodeLeft->sum+nodeRight->sum;
14        }
15        else sum=nums[l];
16    }
17    int query(int a, int b){
18        if(nodeLeft!=nullptr && lazy!=0) nodeLeft->lazy=lazy;
19        if(nodeRight!=nullptr && lazy!=0) nodeRight->lazy=lazy;
20        sum+=(r-l+1)*lazy;lazy=0;
21        if(b<l || a>r) return 0;
22        if(a<=l && r<=b) return sum;
23        return nodeLeft->query(a,b)+nodeRight->query(a,b);
24    }
25    int update(int a, int b, int v){
26        int increment=0;
27        if(b<l || a>r) return 0;
28        if(a<=l && r<=b){
29            if(nodeLeft!=nullptr) nodeLeft->lazy+=lazy;
30            if(nodeRight!=nullptr) nodeRight->lazy+=lazy;
31            increment=(r-l+1)*v;
32            sum+=increment;
33            return increment;
34        }
35        increment=nodeLeft->update(a,b,v)+nodeRight->update(a,b,v);
36        sum+=increment;
37        return increment;
38    }
39 };

```

1.3 Disjoin Set

```

1 //Se usa para detectar ciclos en un grafo no dirigido convexo & en el
  algoritmo de Krustal.
2 vector<pair<int,int>>ds;

```

```

3 void init(int n){
4     ds.assign(n+1,{-1,0});
5 }
6 int find(int x){
7     if(-1==ds[x].first) return x;
8     return ds[x].first=find(ds[x].first);
9 }
10 bool unionDs(int x, int y){
11     int px=find(x),py=find(y);
12     int &rx=ds[px].second,&ry=ds[py].second;
13     if(px==py) return false;
14     else{
15         if(rx>ry) ds[py].first=px;
16         else{
17             ds[px].first=py;
18             if(rx==ry) ry+=1;
19         }
20     }
21     return true;
22 }

```

1.4 Sparse Table

```

1 //Se usa para RMQ porque se puede hacer en O(1), no acepta updates
2 vector<int>lg;
3 vector<vector<int>>>st;
4 int *nums;
5 void init(int n){
6     int logn=(int) log2(n)+1;
7     lg.assign(n+1,0);
8     st.assign(logn,vector<int>(n+1));
9     for(int i=0;i<n;i++) st[0][i]=nums[i];
10    lg[1]=0;
11    for(int i=2;i<=n;i++) lg[i]=lg[i/2]+1;
12    for(int i=1;i<logn;i++)
13        for(int j=0;j+(1<<i)<n;j++)st[i][j]=min(st[i-1][j],st[i-1][j
14        +(1<<(i-1))]);
15 }
16 int query(int a,int b){
17     int logn=lg[(b-a+1)];
18     cout<<st[logn][a]<<endl;
19     return min(st[logn][a],st[logn][b-(1<<logn)+1]);
20 }

```

1.5 Treap

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef struct Node *pitem;
4 struct Node{
5     int value,key;
6     pitem l,r;
7     Node(int v) value(v),key(rand()),l(nullptr),r(nullptr);
8 };
9 struct treap
10 {
11 void split(pitem t, int value, pitem& left,pitem& right){
12     if(!t) void(left=right=nullptr);
13     if(t->value<=x) split(t->r,value,t->r,right),left=t;
14     else split(t->l,value,left,t->l),right=t;
15 }
16 void marge(pitem t,pitem left,pitem right){
17     if(!left || ! right){t=left?left:right;return;}
18     if(left->key>right->key) marge(left->r,left->r,right), t=left;
19     else marge(right->l,left,right->l), t=right;
20 }
21 void insert(pitem &t,pitem x){
22     if(!t)t=x;
23     else if(x->key>t->key){
24         split(t,x->value,x->l,x->r), t=x;
25     }
26     else insert(x->value<t->value? t->l:t->r,x);
27 }
28 };

```

1.6 Trie

```

1 struct trie{
2     bool isFinal;
3     trie *children[26];
4     trie(){
5         isFinal=false;
6         for(int i=0;i<26;i++)children[i]=nullptr;
7     }
8 };
9
10 void inserString(string str,trie *root){

```

```

11     trie *aux=root;
12     for(int i=0;i<str.size();i++){
13         int index=str[i]-'a';
14         if(aux->children[index]==nullptr){
15             aux->children[index]=new trie();
16         }
17         aux=aux->children[index];
18     }
19     aux->isFinal=true;
20 }
21 bool existInTrie(string str,trie *root){
22     trie *aux=root;
23     for(int i=0;i<str.size();i++){
24         int index=str[i]-'a';
25         if(aux->children[index]==nullptr) return false;
26         aux=aux->children[index];
27     }
28     return aux->isFinal;
29 }

```

1.7 Cartesian Tree

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 typedef long long ll;
4 struct node {
5     int idx, val, par, ch[2];
6     friend bool operator<(node a, node b) { return a.idx < b.idx; }
7     void init(int _idx, int _val, int _par) {
8         idx = _idx, val = _val, par = _par, ch[0] = ch[1] = 0;
9     }
10 } tree[N];
11 int root, top, stk[N];
12 int cartesian_build(int n) {
13     for (int i = 1; i <= n; i++) {
14         int k = i - 1;
15         while (tree[k].val > tree[i].val) k = tree[k].par;
16         tree[i].ch[0] = tree[k].ch[1];
17         tree[k].ch[1] = i;
18         tree[i].par = k;
19         tree[tree[i].ch[0]].par = i;
20     }
21     return tree[0].ch[1];

```

```

22 }
23 int dfs(int x) {
24     if (!x) return 0;
25     int sz = dfs(tree[x].ch[0]);
26     sz += dfs(tree[x].ch[1]);
27     ans = max(ans, (ll)(sz + 1) * tree[x].val);
28     return sz + 1;
29 }

```

2 DP

2.1 Digit DP

```

1 ll dp[20][20][3];
2 ll n,k,d;
3 vector<int>num;
4 ll bk(int i, int len, int t){
5     if(len>k) return 0;
6     if(i==n){
7         if(len==k) return 1;
8         return 0;
9     }
10    ll &res=dp[i][len][t];
11    if(res!=-1) return res;
12    res=0;
13    int tope;
14    if(t==0) tope=num[i];
15    else tope=9;
16    for(int j=0;j<=tope;j++){
17        int newt=t;
18        int newlen=len;
19        if(t==0 && j<tope) newt=1;
20        if(d==j) newlen++;
21        if(newlen<=k)res+=bk(i+1,newlen,newt);
22    }
23    return res;
24 }
25 ll rep(int a){
26     num.clear();
27     while(a>0){
28         num.push_back(a%10);
29         a/=10;
30     }

```

```

31 reverse(num.begin(),num.end());
32 n=num.size();
33 memset(dp,-1,sizeof(dp));
34 return bk(0,0,0);
35 }

```

3 Graph

3.1 Krustal

```

1 // Este algoritmo sirve para buscar MST de un grafo convexo no dirigido
2 vector<tuple<int,int,int>>edges;
3 int n,m;
4 //Insertar Disjoin set
5 int kruskal(){
6     sort(edges.begin(),edges.end());
7     int res=0;
8     for(int i=0;i<m;i++){
9         int c,a,b;
10        tie(c,a,b)=edges[i];
11        if(unionDs(a,b)==false) continue;
12        else res+=c;
13    }
14    return res;
15 }

```

3.2 Kosaraju's (SCC)

```

1 //Sirve para encontrar los SCC
2 struct Kosaraju{
3     int s;
4     vector<vector<int>> g,gr;
5     vector<int> visited,ids,topologic_sort;
6     Kosaraju(int n){
7         s=n;
8         g.assign(n+1,vector<int>());
9         gr.assign(n+1,vector<int>());
10        visited.assign(n+1,0);
11        ids.assign(n+1,0);
12    }
13    void addEdge(int a,int b){
14        g[a].push_back(b);
15        gr[b].push_back(a);

```

```

16    }
17    void dfs(int u){
18        if(visited[u]!=0) return;
19        visited[u]=1;
20        for(int node:g[u])dfs(node);
21        topologic_sort.push_back(u);
22    }
23    void dfsr(int u,int id){
24        if(visited[u]!=0) return;
25        visited[u]=1;
26        ids[u]=id;
27        for(int node:gr[u])dfsr(node,id);
28    }
29    void algo(){
30        for(int i=1;i<=s;i++) if(visited[i]==0) dfs(i);
31        fill(visited.begin(),visited.end(),0);
32        reverse(topologic_sort.begin(),topologic_sort.end());
33        int id=0;
34        for(int i=0;i<topologic_sort.size();i++){
35            if(visited[topologic_sort[i]]==0)dfsr(topologic_sort[i],id
36                ++);
37        }
38        int search(int node){
39            return ids[node];
40        }
41    };

```

3.3 2 Sat

```

1 //Se usa para los problems en los cuales tengamos dos dosible variables
2 struct twoSat{
3     int s;
4     vector<vector<int>> g,gr;
5     vector<int> visited,ids,topologic_sort,val;
6     twoSat(int n){
7         s=n;
8         g.assign(n*2+1,vector<int>());
9         gr.assign(n*2+1,vector<int>());
10        visited.assign(n*2+1,0);
11        ids.assign(n*2+1,0);
12        val.assign(n+1,0);
13    }

```

```

14 void addEdge(int a,int b){
15     g[a].push_back(b);
16     gr[b].push_back(a);
17 }
18 void addOr(int a,bool ba,int b,bool bb){
19     addEdge(a+(ba?s:0),b+(bb?0:s));
20     addEdge(b+(bb?s:0),a+(ba?0:s));
21 }
22 void addXor(int a,bool ba,int b,bool bb){
23     addOr(a,ba,b,bb);
24     addOr(a,!ba,b,!bb);
25 }
26 void addAnd(int a,bool ba,int b,bool bb){
27     addXor(a,!ba,b,bb);
28 }
29 void dfs(int u){
30     if(visited[u]!=0) return;
31     visited[u]=1;
32     for(int node:g[u])dfs(node);
33     topologic_sort.push_back(u);
34 }
35 void dfsr(int u,int id){
36     if(visited[u]!=0) return;
37     visited[u]=1;
38     ids[u]=id;
39     for(int node:gr[u])dfr(node,id);
40 }
41 bool algo(){
42     for(int i=0;i<s*2;i++) if(visited[i]==0) dfs(i);
43     fill(visited.begin(),visited.end(),0);
44     reverse(topologic_sort.begin(),topologic_sort.end());
45     int id=0;
46     for(int i=0;i<topologic_sort.size();i++){
47         if(visited[topologic_sort[i]]==0)dfr(topologic_sort[i],id
48             ++);
49     }
50     for(int i=0;i<s;i++){
51         if(ids[i]==ids[i+s]) return false;
52         val[i]=(ids[i]>ids[i+s]?0:1);
53     }
54     return true;
55 }
};

```

4 Strings

4.1 KMP

```

1 vector<int> kmp(string s){
2     int n=s.size();
3     vector<int>pi(n);
4     for(int i=1;i<n;i++){
5         int j=pi[i-1];
6         while(j>0 && s[i]!=s[j])j=pi[j-1];
7         if(s[i]==s[j]) j++;
8         pi[i]=j;
9     }
10    return pi;
11 }

```

4.2 Hashing

```

1 struct Hash{
2     const int mod=1e9+123;
3     const int p=257;
4     vector<int> prefix;
5     static vector<int>pow;
6     Hash(string str){
7         int n=str.size();
8         while(pow.size()<=n){
9             pow.push_back(1LL*pow.back()*p%mod);
10        }
11        vector<int> aux(n+1);
12        prefix=aux;
13        for(int i=0;i<n;i++){
14            prefix[i+1]=(prefix[i]+1LL*str[i]*pow[i])%mod;
15        }
16    }
17    inline int getHashInInterval(int i,int len,int MxPow){
18        int hashing=prefix[i+len]-prefix[i];
19        if(hashing<0) hashing+=mod;
20        hashing=1LL*hashing*pow[MxPow-(len+i-1)]%mod;
21        return hashing;
22    }
23 };
24 vector<int> Hash::pow{1};

```

5 Math

5.1 Linear Sieve

```

1 //O(N) for find all the primes in the given range
2 bool is_compositive[10000000+1];
3 vector<int>primes;
4 void sieve(int n){
5     primes.clear();
6     fill(is_compositive,is_compositive+n,false);
7     for(int i=2;i<=n;i++){
8         if(!is_compositive[i]) primes.push_back(i);
9         for(int j=0;j<primes.size() && primes[j]*i<=n;j++){
10             is_compositive[i*primes[j]]=true;
11             if(!(i%primes[j])) break;
12         }
13     }
14 }
15
16 int n;cin>>n;
17 sieve(n);
18 cout<<primes.size()<<endl;
19 for(int i=0;i<primes.size();i++){
20     cout<<primes[i]<<endl;
21 }
22 }
```

5.2 Euler Sieve

```

1 //this is a sieve for a euler function that given the number of coprime
  numbers of x but in a range
2 vector<int>sieve;
3 void eulerSieve(int n){
4     sieve.clear();
5     sieve.push_back(0);
6     for(int i=1;i<=n;i++){
7         sieve.push_back(i);
8     }
9     for(int i=2;i<=n;i++){
10         if(sieve[i]==i)
11             for(int j=i;j<=n;j+=i)sieve[j]-=(sieve[j]/i);
12     }
13 }
```

5.3 Euler Sieve Gauss Reduction

```

1 // sum(pi(n)) of the divisors of n is equal to n
2 vector<int>sieve;
3 void eulerSieve(int n){
4     sieve.clear();
5     sieve.push_back(0);
6     sieve.push_back(1);
7     for(int i=2;i<=n;i++){
8         sieve.push_back(i-1);
9     }
10    for(int i=2;i<=n;i++){
11        for(int j=i*2;j<=n;j+=i)sieve[j]-=sieve[i];
12    }
13 }
```

5.4 Mobius Sieve

```

1 /* f(x)=0 if has square prime factor
2    f(x)=1 if if is square-free and even
3    f(x)=-1 if is square-free and odd
4    properti the sum of function of divisors of x is equal to 0 if x>1*/
5 vector<int>sieve;
6 void ms(int n){
7     sieve.assign(n+1,-1);
8     sieve[1]=1;
9     for(int i=2;i<=n;i++){
10         for(int j=i*2;j<=n;j+=i)sieve[j]-=sieve[i];
11     }
12 }
```

5.5 Binary Exponentiation

```

1 long long binPow(long long a, long long b) {
2     long long res = 1;
3     while (b > 0) {
4         if (b & 1)
5             res = res * a;
6         a = a * a;
7         b >>= 1;
8     }
9     return res;
10 }
```

6 Flows

6.1 Dinics

```

1 struct dinics{
2     int m,n;
3     ll mF=1e18;
4     vector<tuple<int,ll,ll>>edge;
5     vector<vector<int>>adj;
6     vector<int>level,id;
7     void init(int _n){
8         m=0;
9         n=_n;
10        level.resize(n+1);
11        id.resize(n+1);
12        adj.resize(n+1);
13    }
14    void addEdge(int u,int v,ll f,bool directed=true){
15        edge.push_back({v,f,0});
16        adj[u].push_back(m);
17        edge.push_back({u,(directed?0:f),0});
18        adj[v].push_back(m+1);
19        m+=2;
20    }
21    bool bfs(int s, int t){
22        fill(level.begin(),level.end(),-1);
23        queue<int>aux;
24        aux.push(s);
25        level[s]=0;
26        while(!aux.empty()){
27            int v=aux.front();aux.pop();
28            for(auto idx:adj[v]){
29                auto &[u,c,f]=edge[idx];
30                if(c-f<1 || level[u]==-1) continue;
31                aux.push(u);
32                level[u]=level[v]+1;
33            }
34        }
35        return level[t]!=-1?1:0;
36    }
37    ll dfs(int u,int t, ll f){
38        if(u==t || f==0) return f;
39        for(auto &cdx=id[u];cdx<adj[u].size();cdx++){

```

```

40            int idx=adj[u][cdx];
41            auto &[v,c,fv]=edge[idx];
42            if(level[v]!=level[u]+1 || c-fv<1) continue;
43            ll res=dfs(v,t,min(f,c-fv));
44            if(!res) continue;
45            auto &fr=get<2>(edge[idx^1]);
46            fv+=res;
47            fr-=res;
48            return res;
49        }
50        return 0;
51    }
52    ll maxFlow(int s,int t){
53        ll mf=0;
54        while(bfs(s,t)){
55            fill(id.begin(),id.end(),0);
56            while(ll f=dfs(s,t,mF)) mf+=f;
57        }
58        return mf;
59    }
60 };

```

7 Geometry

8 Others