

June 2, 2022

Md. Mehrab Hossain Opi, Arnob Sarker,
Sharif Minhazul Islam

Contents

1 Data Structure	1
1.1 MO with Update [43 lines]	1
1.2 MO [28 lines]	1
1.3 Persistent Segment Tree [58 lines]	1
1.4 SQRT Decomposition [106 lines]	2
1.5 Segment Tree [73 lines]	2
1.6 Trie Bit [61 lines]	3
2 Dynamic Programming	3
2.1 Convex Hull Trick [91 lines]	3
2.2 Divide and Conquer DP [26 lines]	4
2.3 Knuth Optimization [32 lines]	4
2.4 LIS O(nlogn) with full path [17 lines]	4
2.5 SOS DP [18 lines]	4
3 Flow	5
3.1 Blossom [58 lines]	5
3.2 Dinic [72 lines]	5
3.3 Flow [6 lines]	5
3.4 HopCroftKarp [67 lines]	6
3.5 Hungarian [116 lines]	6
3.6 MCMF [116 lines]	7
4 Game Theory	7
4.1 Points to be noted [14 lines]	7
5 Geometry	8
5.1 Geometry [384 lines]	8
5.2 Rotation Matrix [39 lines]	9
6 Graph	10
6.1 2SAT [92 lines]	10
6.2 BellmanFord [57 lines]	11
6.3 BridgeTree [66 lines]	11
6.4 Centroid Decomposition [49 lines]	12
6.5 DSU on Tree [69 lines]	12
6.6 Dijkstra [33 lines]	13
6.7 Heavy Light Decomposition [73 lines]	13
6.8 K'th Shortest path [40 lines]	13
6.9 LCA [46 lines]	14
6.10 SACK [50 lines]	14
6.11 SCC [43 lines]	14
7 Math	15
7.1 Big Sum [13 lines]	15
7.2 CRT [52 lines]	15
7.3 FFT [85 lines]	15
7.4 GaussElimination [39 lines]	16
7.5 GaussMod2 [44 lines]	16

```

7.6 Karatsuba Idea [5 lines] . . . . . 16
7.7 Linear Diophantine [12 lines] . . . . . 16
7.8 Matrix [100 lines] . . . . . 16
7.9 Miller-Rabin-Pollard-Rho [68 lines] . . . . . 17
7.10 Mod Inverse [5 lines] . . . . . 17
7.11 NTT [96 lines] . . . . . 17
7.12 No of Digits in n! in base B [7 lines] . . . . . 18
7.13 SOD Upto N [16 lines] . . . . . 18
7.14 Sieve-Phi-Mobius [26 lines] . . . . . 18

8 Misc 18
8.1 Bit hacks [12 lines] . . . . . 18
8.2 Bitset C++ [13 lines] . . . . . 18
8.3 Template [34 lines] . . . . . 18
8.4 Vimrc [26 lines] . . . . . 19
8.5 build [3 lines] . . . . . 19
8.6 debug [6 lines] . . . . . 19
8.7 stress-test [16 lines] . . . . . 19

9 String 19
9.1 Aho-Corasick [124 lines] . . . . . 19
9.2 Double Hasing [50 lines] . . . . . 20
9.3 KMP [23 lines] . . . . . 20
9.4 Palindromic Tree [30 lines] . . . . . 20
9.5 Suffix Array [78 lines] . . . . . 21
9.6 Suffix Automata [118 lines] . . . . . 21
9.7 Trie [28 lines] . . . . . 22
9.8 Z-Algorithm [19 lines] . . . . . 22

1 Data Structure
1.1 MO with Update [43 lines]


---



```

//1 indexed
//Complexity:O(S × Q + Q × $\frac{N^2}{S^2}$)
//S = (2*n^2)^(1/3)
const int block_size = 2720; // 4310 for 2e5
const int mx = 1e5 + 5;
struct Query {
 int L, R, T, id;
 Query() {}
 Query(int _L, int _R, int _T, int _id) : L(_L),
 R(_R), T(_T), id(_id) {}
 bool operator<(const Query &x) const {
 if (L / block_size == x.L / block_size) {
 if (R / block_size == x.R / block_size)
 return T < x.T;
 return R / block_size < x.R / block_size;
 }
 return L / block_size < x.L / block_size;
 }
} Q[mx];
struct Update {
 int pos;
 int old, cur;
 Update() {};
 Update(int _p, int _o, int _c) : pos(_p),
 old(_o), cur(_c){};
} U[mx];
int ans[mx];

```



---


1.2 MO [28 lines]

```

const int N = 2e5 + 5;
const int Q = 2e5 + 5;
const int SZ = sqrt(N) + 1;
struct qry {
 int l, r, id, blk;
 bool operator<(const qry& p) const {
 return blk == p.blk ? r < p.r : blk < p.blk;
 }
};
qry query[Q];
ll ans[Q];
void add(int id) {}
void remove(int id) {}
ll get() {}
int n, q;
void MO() {
 sort(query, query + q);
 int cur_l = 0, cur_r = -1;
 for (int i = 0; i < q; i++) {
 qry q = query[i];
 while (cur_l > q.l) add(--cur_l);
 while (cur_r < q.r) add(++cur_r);
 while (cur_l < q.l) remove(cur_l++);
 while (cur_r > q.r) remove(cur_r--);
 ans[q.id] = get();
 }
}
/* 0 indexed. */

```



---


1.3 Persistent Segment Tree [58 lines]

```

const int mxn = 1e5+5; //CHECK here for every
problem
int root[mxn], leftchild[25*mxn],
rightchild[25*mxn], value[25*mxn], a[mxn];
int now = 0, n;

int build(int L, int R){
 int node = ++now;

```


```

```

if(L == R){
    return node;
}
int mid = (L+R)>>1;
leftchild[node] = build(L, mid);
rightchild[node] = build(mid+1, R);
//initialize value[node]
return node;
}

int update(int nownode, int L, int R, int val){
int node = ++now;
if(L == R){
    //value[node] = value[nownode]+1;
    //update value[node]
    return node;
}
int mid = (L+R)>>1;
leftchild[node] = leftchild[nownode];
rightchild[node] = rightchild[nownode];

if(mid <= val){//change condition as required
    leftchild[node] = update(leftchild[nownode],
    L, mid, val);
}
else{
    rightchild[node] =
    update(rightchild[nownode], mid+1, R, val);
}
//value[node] = value[nownode]+1;
//update value[node]
return node;
}

int query(int leftnode, int rightnode, int L, int
R, int k){
if(L==R) return L;
//int leftcnt = value[leftchild[rightnode]]-
//value[leftchild[leftnode]]; a
//change as required
int mid = (L+R)>>1;
if(leftcnt >= k){//change condition as required
    return query(leftchild[leftnode],
    leftchild[rightnode], L, mid, k);
}
else{
    return query(rightchild[leftnode],
    rightchild[rightnode], mid+1, R, k-leftcnt);
}

void persistentsegtree(){
root[0] = build(0, mxn);
for(int i=1; i<=n; i++){
    root[i] = update(root[i-1], 0, mxn, a[i]);
}
}

```

1.4 SQRT Decomposition [106 lines]

```

struct sqrtDecomposition
{
    static const int sz = 320; //sz = sqrt(N);
    int numberofblocks;

    struct node
    {
        int L, R;
        bool islazy = false;
        ll lazyval=0;
        //extra data needed for different problems

        void ini(int l, int r)
        {
            for(int i=l; i<=r; i++){
                //...initialize as need
            }
            L=l, R=r;
        }

        void semiupdate(int l, int r, ll val){
            if(l>r) return;
            if(islazy){
                for(int i=L; i<=R; i++){
                    //...distribute lazy to everyone
                }
                islazy = 0;
                lazyval = 0;
            }
            for(int i=l; i<=r; i++){
                //...do it manually
            }
        }

        void fullupdate(ll val){
            if(islazy){
                //...only update lazyval
            }
            else{
                for(int i=L; i<=R; i++){
                    //...everyone are not equal, make them
                    equal
                }
                islazy = 1;
                //update lazyval
            }
        }

        void update(int l, int r, ll val){
            if(l<=L && r>=R) fullupdate(val);
            else semiupdate(max(l, L), min(r, R), val);
        }

        ll semiquery(int l, int r){
            if(l>r) return 0;
            if(islazy){
                for(int i=L; i<=R; i++){
                    //...distribute lazy to everyone
                }
                islazy = 0;
                lazyval = 0;
            }
        }
    }
}

```

```

    }
    ll ret = 0;
    for(int i=l; i<=r; i++){
        //...take one by one
    }
    return ret;
}

ll fullquery(){
    //return stored value;
}

ll query(int l, int r){
    if(l<=L && r>=R) return fullquery();
    else return semiquery(max(l, L), min(r, R));
}

vector<node> blocks;

void init(int n){
    numberofblocks = (n+sz-1)/sz;
    int curL = 1, curR = sz;
    blocks.resize(numberofblocks+5);
    for(int i=1; i<=numberofblocks; i++){
        curR = min(n, curR);
        blocks[i].ini(curL, curR);
        curL += sz;
        curR += sz;
    }
}

void update(int l, int r, ll val){
    int left = (l-1)/sz+1;
    int right = (r-1)/sz+1;
    for(int i=left; i<=right; i++){
        blocks[i].update(l, r, val);
    }
}

ll query(int l, int r){
    int left = (l-1)/sz+1;
    int right = (r-1)/sz+1;
    ll ret = 0;
    for(int i=left; i<=right; i++){
        ret += blocks[i].query(l, r);
    }
    return ret;
}
}

```

1.5 Segment Tree [73 lines]

```

/*edit: data, combine, build check datatype*/
template<typename T>
struct SegmentTree {
#define lc (C << 1)
}
```

```

#define rc (C << 1 / 1)
#define M ((L+R)>>1)
struct data {
    T sum;
    data() :sum(0) {};
};
vector<data>st;
vector<bool>isLazy;
vector<T>lazy;
int N;
SegmentTree(int _N) :N(_N) {
    st.resize(4 * N);
    isLazy.resize(4 * N);
    lazy.resize(4 * N);
}
void combine(data& cur, data& l, data& r) {
    cur.sum = l.sum + r.sum;
}
void push(int C, int L, int R) {
    if (!isLazy[C]) return;
    if (L != R) {
        isLazy[lc] = 1;
        isLazy[rc] = 1;
        lazy[lc] += lazy[C];
        lazy[rc] += lazy[C];
    }
    st[C].sum = (R - L + 1) * lazy[C];
    lazy[C] = 0;
    isLazy[C] = false;
}
void build(int C, int L, int R) {
    if (L == R) {
        st[C].sum = 0;
        return;
    }
    build(lc, L, M);
    build(rc, M + 1, R);
    combine(st[C], st[lc], st[rc]);
}
data Query(int i, int j, int C, int L, int R) {
    push(C, L, R);
    if (j < L || i > R || L > R) return data();
    // default val 0/INF
    if (i <= L && R <= j) return st[C];
    data ret;
    data d1 = Query(i, j, lc, L, M);
    data d2 = Query(i, j, rc, M + 1, R);
    combine(ret, d1, d2);
    return ret;
}
void Update(int i, int j, T val, int C, int L,
           int R) {
    push(C, L, R);
    if (j < L || i > R || L > R) return;
    if (i <= L && R <= j) {
        isLazy[C] = 1;
        lazy[C] = val;
        push(C, L, R);
    }
    else {
        if (i < j) {
            Update(i, j, val, lc, L, M);
            Update(i, j, val, rc, M + 1, R);
            combine(st[C], st[lc], st[rc]);
        }
        void Update(int i, int j, T val) {
            Update(i, j, val, 1, 1, N);
        }
        T Query(int i, int j) {
            return Query(i, j, 1, 1, N).sum;
        }
    }
}

```

1.6 Trie Bit [61 lines]

```

struct Trie {
    struct node {
        int next[2];
        int cnt, fin;
        node() :cnt(0), fin(0) {
            for (int i = 0; i < 2; i++) next[i] = -1;
        }
    };
    vector<node>data;
    Trie() {
        data.push_back(node());
    }
    void key_add(int val) {
        int cur = 0;
        for (int i = 30; i >= 0; i--) {
            int id = (val >> i) & 1;
            if (data[cur].next[id] == -1) {
                data[cur].next[id] = data.size();
                data.push_back(node());
            }
            cur = data[cur].next[id];
            data[cur].cnt++;
        }
        data[cur].fin++;
    }
    int key_search(int val) {
        int cur = 0;
        for (int i = 30; ~i; i--) {
            int id = (val >> i) & 1;
            if (data[cur].next[id] == -1) return 0;
            cur = data[cur].next[id];
        }
        return data[cur].fin;
    }
    void key_delete(int val) {
        int cur = 0;
        for (int i = 30; ~i; i--) {
            int id = (val >> i) & 1;
            cur = data[cur].next[id];
            data[cur].cnt--;
        }
        data[cur].fin--;
    }
    bool key_remove(int val) {

```

```

        if (key_search(val)) return key_delete(val),
        1;
        return 0;
    }
    int maxXor(int x) {
        int cur = 0;
        int ans = 0;
        for (int i = 30; ~i; i--) {
            int b = (x >> i) & 1;
            if (data[cur].next[!b] + 1 &&
                data[data[cur].next[!b]].cnt > 0) {
                ans += (1LL << i);
                cur = data[cur].next[!b];
            }
            else cur = data[cur].next[b];
        }
        return ans;
    }
}



---



## 2 Dynamic Programming



### 2.1 Convex Hull Trick [91 lines]



```

struct Hull_Static{
/*all m need to be decreasing order
if m is in increasing order then negate the
m(like,add_line(-m,c)),
remember in query you have to negate the x also*/
const ll inf=10000000000000000000;
int min_or_max;//if min then 0 otherwise 1
int pointer; /*keep track for the best line for
previous query, requires all insert first*/
vector<ll>M,C; //y=m*x+c;
inline void clear(){
 min_or_max=0;//initially with minimum trick
 pointer=0;
 M.clear();
 C.clear();
}
Hull_Static(){clear();}
Hull_Static(int _min_or_max){
 clear();
 this->min_or_max=_min_or_max;
}
bool bad_min(int idx1,int idx2,int idx3){
 return(C[idx3]-C[idx1])*(M[idx1]-M[idx2]) <
 (C[idx2]-C[idx1])*(M[idx1]-M[idx3]);
 return 1.0*(C[idx3]-C[idx1])*(M[idx1]-M[idx2]) <
 1.0*(C[idx2]-C[idx1])*(M[idx1]-M[idx3]); //for
 overflow
}
bool bad_max(int idx1,int idx2,int idx3){
 return(C[idx3]-C[idx1])*(M[idx1]-M[idx2]) >
 (C[idx2]-C[idx1])*(M[idx1]-M[idx3]);
}
```


```

```

return 1.0*(C[idx3]-C[idx1])*(M[idx1]-M[idx2])
> 1.0*(C[idx2]-C[idx1])*(M[idx1]-M[idx3]); //for
overflow
}
bool bad(int idx1,int idx2,int idx3){
    if(!min_or_max) return bad_min(idx1,idx2,idx3);
    else return bad_max(idx1,idx2,idx3);
}
void add_line(ll m,ll c){/*add line where m is
given in decreasing order
//if(M.size()>0 and M.back()==m) return; //same
gradient, no need to add
above line added from tarango khan, this line cost
me sevaral wa, but some code got ac with this*/
M.push_back(m);
C.push_back(c);
while(M.size()>=3 and bad((int)M.size()-3,
(int)M.size()-2,(int)M.size()-1)){
    M.erase(M.end()-2);
    C.erase(C.end()-2);
}
}
ll getval(ll idx,ll x){
    return M[idx]*x+C[idx];
}
ll getminval(ll x){/*if queries are
non-decreasing order*/
while(pointer<(int)M.size()-1 and
getval(pointer+
1,x)<getval(pointer,x))pointer++;
return M[pointer]*x+C[pointer];
}
ll getmaxval(ll x){
    while(pointer<(int)M.size()-1 and
getval(pointer+
1,x)>getval(pointer,x))pointer++;
return M[pointer]*x+C[pointer];
}
ll getminvalternary(ll x){
    ll lo=0,hi=(ll)M.size()-1;
    ll ans=inf; /*change with problem*/
    while(lo<=hi){
        ll mid1=lo+(hi-lo)/3;
        ll mid2=hi-(hi-lo)/3;
        ll val1=getval(mid1,x);
        ll val2=getval(mid2,x);
        if(val1<val2){
            ans=min(ans,val2);
            hi=mid2-1;
        }
        else{
            ans=min(ans,val1);
            lo=mid1+1;
        }
    }
    return ans;
}

```

```

11 getmaxvalternary(ll x){
    11 lo=0,hi=(ll)M.size()-1;
    11 ans=-inf; /*change with problem*/
    while(lo<=hi){
        11 mid1=lo+(hi-lo)/3;
        11 mid2=hi-(hi-lo)/3;
        11 val1=getval(mid1,x);
        11 val2=getval(mid2,x);
        if(val1<val2){
            ans=max(ans,val2);
            lo=mid1+1;
        }
        else{
            ans=max(ans,val1);
            hi=mid2-1;
        }
    }
    return ans;
}

```

2.2 Divide and Conquer DP [26 lines]

```

11 G,L; ///total group,cell size
11 dp[8001][801],cum[8001];
11 C[8001]; ///value of each cell
inline ll cost(ll l,ll r){
    return(cum[r]-cum[l-1])*(r-l+1);
}
void fn(ll g,ll st,ll ed,ll r1,ll r2){
    if(st>ed) return;
    ll mid=(st+ed)/2, pos=-1;
    dp[mid][g]=inf;
    for(int i=r1;i<=r2;i++){
        ll tcost=cost(i,mid)+dp[i-1][g-1];
        if(tcost<dp[mid][g]){
            dp[mid][g]=tcost, pos=i;
        }
    }
    fn(g,st,mid-1,r1,pos);
    fn(g,mid+1,ed, pos,r2);
}
int main(){
    for(int i=1;i<=L;i++)
        cum[i]=cum[i-1]+C[i];
    for(int i=1;i<=L;i++)
        dp[i][1]=cost(1,i);
    for(int i=2;i<=G;i++)fn(i,1,L,1,L);
}

```

2.3 Knuth Optimization [32 lines]

/*It is applicable where recurrence is in the form

$$dp[i][j] = \min_{i < k < j} \{ dp[i][k] + dp[k][j] \} + C[i][j]$$

condition for applicability is:

$$A[i, j-1] \leq A[i, j] \leq A[i+1, j]$$

Where,

$A[i][j]$ - the smallest k that gives optimal answer, like-

$$dp[i][j] = dp[i-1][k] + C[k][j]$$

$C[i][j]$ -given cost function
also applicable if: $C[i][j]$ satisfies the following 2 conditions:
 $C[a][c]+C[b][d] \leq C[a][d]+C[b][c]$, $a \leq b \leq c \leq d$
 $C[b][c] \leq C[a][d]$, $a \leq b \leq c \leq d$
reduces time complexity from $O(n^3)$ to $O(n^2)$ */
for(int s=0;s<=k;s++) //s-length(size) of substring
for(int l=0;l+s<=k;l++) { //l-left point
 int r=l+s; //r-right point
 if(s<2){
 res[l][r]=0; //DP base-nothing to break
 mid[l][r]=1; /*mid is equal to left border*/
 continue;
 }
 int mleft=mid[l][r-1]; /*Knuth's trick: getting
bounds on m*/
 int mright=mid[l+1][r];
 res[l][r]=inf;
 for(int m=mleft;m<=mright;m++){ /*iterating for
m in the bounds only*/
 int64 tres=res[l][m]+res[m][r]+(x[r]-x[l]);
 if(res[l][r]>tres){ //relax current solution
 res[l][r]=tres;
 mid[l][r]=m;
 }
 }
}
int64 answer=res[0][k];

2.4 LIS O(nlogn) with full path [17 lines]

```

int num[MX],mem[MX],prev[MX],array[MX],res[MX],_
maxlen;
void LIS(int SZ,int num[]){
    CLR(mem),CLR(prev),CLR(array),CLR(res);
    int i,k;
    maxlen=1;
    array[0]=-inf;
    RFOR(i,1,SZ+1) array[i]=inf;
    prev[0]=-1,mem[0]=num[0];
    FOR(i,SZ){
        k=lower_bound(array,array+maxlen+1,num[i])-_
        array;
        if(k==1) array[k]=num[i],mem[k]=i,prev[i]=-1;
        else
            array[k]=num[i],mem[k]=i,prev[i]=mem[k-1];
            if(k>maxlen) maxlen=k;
    }
    k=0;
    for(i=mem[maxlen];i!= -1;i=prev[i])res[k++]=_
    num[i];
}

```

2.5 SOS DP [18 lines]

//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)];
}

3 Flow
3.1 Blossom [58 lines]


---


// Finds Maximum matching in General Graph
// Complexity O(NM)
// mate[i] = j means i is paired with j
// source: #comment-810242
vector<int> Blossom(vector<vector<int>>& graph) {
    //mate contains matched edge.
    int n = graph.size(), timer = -1;
    vector<int> mate(n, -1), label(n), parent(n),
        orig(n), aux(n, -1), q;
    auto lca = [&](int x, int y) {
        for (timer++; ; swap(x, y)) {
            if (x == -1) continue;
            if (aux[x] == timer) return x;
            aux[x] = timer;
            x = (mate[x] == -1 ? -1 :
                orig[parent[mate[x]]]);
        }
    };
    auto blossom = [&](int v, int w, int a) {
        while (orig[v] != a) {
            parent[v] = w; w = mate[w];
            if (label[w] == 1) label[w] = 0,
                q.push_back(w);
            orig[v] = orig[w] = a; v = parent[w];
        }
    };
    auto augment = [&](int v) {
        while (v != -1) {
            int pv = parent[v], nv = mate[pv];
            mate[v] = pv; mate[pv] = v; v = nv;
        }
    };
    auto bfs = [&](int root) {
        fill(label.begin(), label.end(), -1);
        iota(orig.begin(), orig.end(), 0);
        q.clear();
        label[root] = 0; q.push_back(root);
    };
}

```

```

for (int i = 0; i < (int)q.size(); ++i) {
    int v = q[i];
    for (auto x : graph[v]) {
        if (label[x] == -1) {
            label[x] = 1; parent[x] = v;
            if (mate[x] == -1)
                return augment(x), 1;
            label[mate[x]] = 0;
            q.push_back(mate[x]);
        }
        else if (label[x] == 0 && orig[v] != orig[x]) {
            int a = lca(orig[v], orig[x]);
            blossom(x, v, a); blossom(v, x, a);
        }
    }
    return 0;
}
// Time halves if you start with (any) maximal matching.
for (int i = 0; i < n; i++)
    if (mate[i] == -1)
        bfs(i);
    return mate;
}

3.2 Dinic [72 lines]


---


/*Complexity: O(V^2 E)
.Call Dinic with total number of nodes.
.Nodes start from 0.
.Capacity is long long data.
.make graph with create edge(u,v,capacity).
.Get max flow with maxFlow(src,des).*/
#define eb emplace_back
struct Dinic {
    struct Edge {
        int u, v;
        ll cap, flow = 0;
        Edge() {}
        Edge(int u, int v, ll cap) :u(u), v(v),
            cap(cap) {}
    };
    int N;
    vector<Edge>edge;
    vector<vector<int>>adj;
    vector<int>d, pt;
    Dinic(int N) :N(N), edge(0), adj(N), d(N), pt(N)
        {}
    void addEdge(int u, int v, ll cap) {
        if (u == v) return;
        edge.eb(u, v, cap);
        adj[u].eb(edge.size() - 1);
        edge.eb(v, u, 0);
        adj[v].eb(edge.size() - 1);
    }
    bool bfs(int s, int t) {
        queue<int>q({s});
        fill(d.begin(), d.end(), N + 1);

```

```

d[s] = 0;
while (!q.empty()) {
    int u = q.front(); q.pop();
    if (u == t) break;
    for (int k : adj[u]) {
        Edge& e = edge[k];
        if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
            d[e.v] = d[e.u] + 1;
            q.emplace(e.v);
        }
    }
}
return d[t] != N + 1;
}
ll dfs(int u, int T, ll flow = -1) {
    if (u == T || flow == 0) return flow;
    for (int& i = pt[u]; i < adj[u].size(); i++) {
        Edge& e = edge[adj[u][i]];
        Edge& oe = edge[adj[u][i] ^ 1];
        if (d[e.v] == d[e.u] + 1) {
            ll amt = e.cap - e.flow;
            if (flow != -1 && amt > flow) amt = flow;
            if (ll pushed = dfs(e.v, T, amt)) {
                e.flow += pushed;
                oe.flow -= pushed;
                return pushed;
            }
        }
    }
    return 0;
}
ll maxFlow(int s, int t) {
    ll total = 0;
    while (bfs(s, t)) {
        fill(pt.begin(), pt.end(), 0);
        while (ll flow = dfs(s, t)) {
            total += flow;
        }
    }
    return total;
}
};

3.3 Flow [6 lines]


---


Covering Problems:

```

- > Maximum Independent Set(Bipartite): Largest set of nodes which do not have any edge between them. sol: V-(MaxMatching)
- > Minimum Vertex Cover(Bipartite): -Smallest set of nodes to cover all the edges -sol: MaxMatching
- > Minimum Edge Cover(General graph): -Smallest set of edges to cover all the nodes -sol: V-(MaxMatching) (if edge cover exists, does not exit for isolated nodes)

> Minimum Path Cover(Vertex disjoint) DAG:
 -Minimum number of vertex disjoint paths that visit all the nodes -sol: make a bipartite graph using same nodes in two sides, one side is "from" other is "to", add edges from "from" to "to", then ans is $V - (\text{MaxMatching})$

> Minimum Path Cover(Vertex Not Disjoint) General graph: -Minimum number of paths that visit all the nodes -sol: consider cycles as nodes then it will become a path cover problem with vertex disjoint on DAG

3.4 HopcroftKarp [67 lines]

```
/* Finds Maximum Matching In a bipartite graph
   Complexity O(EV1/2)
   .1-indexed
   .No default constructor
   .add single edge for (u, v)*/

struct HK {
    static const int inf = 1e9;
    int n;
    vector<int> matchL, matchR, dist;
    //matchL contains value of matched node for L part.
    vector<vector<int>> adj;
    HK(int n) : n(n), matchL(n + 1), matchR(n + 1), dist(n + 1), adj(n + 1) {}

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    bool bfs() {
        queue<int> q;
        for (int u = 1; u <= n; u++) {
            if (!matchL[u]) {
                dist[u] = 0;
                q.push(u);
            } else dist[u] = inf;
        }
        dist[0] = inf; // unmatched node matches with 0.
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto v : adj[u]) {
                if (dist[matchR[v]] == inf) {
                    dist[matchR[v]] = dist[u] + 1;
                    q.push(matchR[v]);
                }
            }
        }
        return dist[0] != inf;
    }

    bool dfs(int u) {
        if (!u) return true;
    }
```

```
for (auto v : adj[u]) {
    if (dist[matchR[v]] == dist[u] + 1
        && dfs(matchR[v])) {
        matchL[u] = v;
        matchR[v] = u;
        return true;
    }
}
dist[u] = inf;
return false;
}

int max_match() {
    int matching = 0;
    while (bfs()) {
        for (int u = 1; u <= n; u++) {
            if (!matchL[u])
                if (dfs(u))
                    matching++;
        }
    }
    return matching;
}
```

3.5 Hungarian [116 lines]

```
/* Complexity: O(n^3) but optimized
   It finds minimum cost maximum matching.
   For finding maximum cost maximum matching
   add -cost and return -matching()
   1-indexed */

struct Hungarian {
    long long c[N][N], fx[N], fy[N], d[N];
    int l[N], r[N], arg[N], trace[N];
    queue<int> q;
    int start, finish, n;
    const long long inf = 1e18;
    Hungarian() {}
    Hungarian(int n1, int n2) : n(max(n1, n2)) {
        for (int i = 1; i <= n; ++i) {
            fy[i] = l[i] = r[i] = 0;
            for (int j = 1; j <= n; ++j) c[i][j] = inf;
        }
    }

    void add_edge(int u, int v, long long cost) {
        c[u][v] = min(c[u][v], cost);
    }

    inline long long getC(int u, int v) {
        return c[u][v] - fx[u] - fy[v];
    }

    void initBFS() {
        while (!q.empty()) q.pop();
        q.push(start);
        for (int i = 0; i <= n; ++i) trace[i] = 0;
        for (int v = 1; v <= n; ++v) {
            d[v] = getC(start, v);
            arg[v] = start;
        }
        finish = 0;
    }
```

```
}
```

```
void findAugPath() {
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v = 1; v <= n; ++v) if (!trace[v]) {
            long long w = getC(u, v);
            if (!w) {
                trace[v] = u;
                if (!r[v]) {
                    finish = v;
                    return;
                }
                q.push(r[v]);
            }
            if (d[v] > w) {
                d[v] = w;
                arg[v] = u;
            }
        }
    }
}

void subX_addY() {
    long long delta = inf;
    for (int v = 1; v <= n; ++v) if (trace[v] == 0
        && d[v] < delta) {
        delta = d[v];
    }
    // Rotate
    fx[start] += delta;
    for (int v = 1; v <= n; ++v) if (trace[v]) {
        int u = r[v];
        fy[v] -= delta;
        fx[u] += delta;
    }
    else d[v] -= delta;
    for (int v = 1; v <= n; ++v) if (!trace[v] &&
        !d[v]) {
        trace[v] = arg[v];
        if (!r[v]) {
            finish = v;
            return;
        }
        q.push(r[v]);
    }
}

void Enlarge() {
    do {
        int u = trace[finish];
        int nxt = l[u];
        l[u] = finish;
        r[finish] = u;
        finish = nxt;
    } while (finish);
```

```

long long maximum_matching() {
    for (int u = 1; u <= n; ++u) {
        fx[u] = c[u][1];
        for (int v = 1; v <= n; ++v) {
            fx[u] = min(fx[u], c[u][v]);
        }
    }
    for (int v = 1; v <= n; ++v) {
        fy[v] = c[1][v] - fx[1];
        for (int u = 1; u <= n; ++u) {
            fy[v] = min(fy[v], c[u][v] - fx[u]);
        }
    }
    for (int u = 1; u <= n; ++u) {
        start = u;
        initBFS();
        while (!finish) {
            findAugPath();
            if (!finish) subX_addY();
        }
        Enlarge();
    }
    long long ans = 0;
    for (int i = 1; i <= n; ++i) {
        if (c[i][l[i]] != inf) ans += c[i][l[i]];
        else l[i] = 0;
    }
    return ans;
}

3.6 MCMF [116 lines]
/*Credit: ShahjalalShohag
.Works for both directed, undirected and with
negative cost too
.doesn't work for negative cycles
.for undirected edges just make the directed
flag false
.Complexity: O(min(E^2 *V log V, E logV *
flow))*/
using T = long long;
const T inf = 1LL << 61;
struct MCMF {
    struct edge {
        int u, v;
        T cap, cost;
        int id;
        edge(int _u, int _v, T _cap, T _cost, int _id)
        {
            u = _u;
            v = _v;
            cap = _cap;
            cost = _cost;
            id = _id;
        }
    };
    int n, s, t, mxid;
    T flow, cost;
    vector<vector<int>> g;
}

```

```

vector<edge> e;
vector<T> d, potential, flow_through;
vector<int> par;
bool neg;
MCMF() {}
MCMF(int _n) { // 0-based indexing
    n = _n + 10;
    g.assign(n, vector<int>());
    neg = false;
    mxid = 0;
}
void add_edge(int u, int v, T cap, T cost, int id = -1, bool directed = true) {
    if (cost < 0) neg = true;
    g[u].push_back(e.size());
    e.push_back(edge(u, v, cap, cost, id));
    g[v].push_back(e.size());
    e.push_back(edge(v, u, 0, -cost, -1));
    mxid = max(mxid, id);
    if (!directed) add_edge(v, u, cap, cost, -1, true);
}
bool dijkstra() {
    par.assign(n, -1);
    d.assign(n, inf);
    priority_queue<pair<T, T>, vector<pair<T, T>>, greater<pair<T, T>> q;
    d[s] = 0;
    q.push(pair<T, T>(0, s));
    while (!q.empty()) {
        int u = q.top().second;
        T nw = q.top().first;
        q.pop();
        if (nw != d[u]) continue;
        for (int i = 0; i < (int)g[u].size(); i++) {
            int id = g[u][i];
            int v = e[id].v;
            T cap = e[id].cap;
            T w = e[id].cost + potential[u] -
                potential[v];
            if (d[u] + w < d[v] && cap > 0) {
                d[v] = d[u] + w;
                par[v] = id;
                q.push(pair<T, T>(d[v], v));
            }
        }
        for (int i = 0; i < n; i++) { // update
            potential
                if (d[i] < inf) potential[i] += d[i];
        }
        return d[t] != inf;
    }
}
T send_flow(int v, T cur) {
    if (par[v] == -1) return cur;
    int id = par[v];
    int u = e[id].u;
    T w = e[id].cost;

```

```

T f = send_flow(u, min(cur, e[id].cap));
cost += f * w;
e[id].cap -= f;
e[id ^ 1].cap += f;
return f;
}
//returns {maxflow, mincost}
pair<T, T> solve(int _s, int _t, T goal = inf) {
    s = _s;
    t = _t;
    flow = 0, cost = 0;
    potential.assign(n, 0);
    if (neg) {
        // run Bellman-Ford to find starting
        potential
        d.assign(n, inf);
        for (int i = 0, relax = true; i < n &&
relax; i++) {
            for (int u = 0; u < n; u++) {
                for (int k = 0; k < (int)g[u].size(); k++) {
                    int id = g[u][k];
                    int v = e[id].v;
                    T cap = e[id].cap, w = e[id].cost;
                    if (d[v] > d[u] + w && cap > 0) {
                        d[v] = d[u] + w;
                        relax = true;
                    }
                }
            }
        }
        for (int i = 0; i < n; i++) if (d[i] < inf)
potential[i] = d[i];
    }
    while (flow < goal && dijkstra()) flow +=
send_flow(t, goal - flow);
flow_through.assign(mxid + 10, 0);
for (int u = 0; u < n; u++) {
    for (auto v : g[u]) {
        if (e[v].id >= 0) flow_through[e[v].id] =
e[v ^ 1].cap;
    }
}
return make_pair(flow, cost);
}

```

4 Game Theory

4.1 Points to be noted [14 lines]

```

>First Write a Brute Force solution
>Nim = all xor
>Misere Nim = Nim + corner case: if all piles are
1, reverse(nim)
>Bogus Nim = Nim

```

>Staircase Nim = Odd indexed pile Nim (Even indexed pile doesn't matter, as one player can give bogus moves to drop all even piles to ground)

>Sprague Grundy: [Every impartial game under the normal play convention is equivalent to a one-heap game of nim]

Every tree = one nim pile = tree root value; tree leaf value = 0; tree node value = mex of all child nodes.

[Careful: one tree node can become multiple new tree roots(multiple elements in one node), then the value of that node = xor of all those root values]

>Hackenbush(Given a rooted tree; cut an edge in one move; subtree under that edge gets removed; last player to cut wins):

Colon: // $G(u) = (G(v_1) + 1) \oplus (G(v_2) + 1) \oplus \dots [v_1, v_2, \dots \text{ are children of } u]$

For multiple trees ans is their xor

>Hackenbush on graph (instead of tree given an rooted graph):

fusion: All edges in a cycle can be fused to get a tree structure; build a super node, connect some single nodes with that super node, number of single nodes is the number of edges in the cycle.

Sol: [Bridge component tree] mark all bridges, a group of edges that are not bridges, becomes one component and contributes number of edges to the hackenbush. (even number of edges contributes 0, odd number of edges contributes 1)

5 Geometry

5.1 Geometry [384 lines]

```
namespace Geometry
{
#define M_PIacos(-1.0)
double eps=1e-8;
typedef double T; //coordinate point type
struct pt //Point
{
    T x,y;
    pt(){}
    pt(T _x,T _y):x(_x),y(_y){}
    pt operator+(pt p){
        return{x+p.x,y+p.y};
    }
    pt operator-(pt p){
        return{x-p.x,y-p.y};
    }
    pt operator*(T d){
        return{x*d,y*d};
    }
    pt operator*(pt d){/*I added for General
    linear transfo, not sure about that
    function*/}
}
```

```

        return{x*d.x,y*d.y};
    }
    pt operator/(T d){
        return{x/d,y/d}; /*only for floating point*/
    }
    pt operator/(pt d){/*I added for General
    linear transfo, not sure about that
    function*/
        return{x/d.x,y/d.y};
    }
    bool operator<(const pt& p) const {
        if(x!=p.x)
            return x<p.x;
        return y<p.y;
    }
    bool operator==(pt b){
        return x==b.x && y==b.y;
    }
    bool operator!=(pt b){
        return!(*(this)==b);
    }
    friend ostream& operator<<(ostream& os,const
    pt p){
        return os<<"("<<p.x<<","<<p.y<<")";
    }
    friend istream& operator>>(istream& is,pt &p){
        is>>p.x>>p.y;
        return is;
    }
};

T sq(pt p){
    return p.x*p.x+p.y*p.y;
}
double Abs(pt p){
    return sqrtl(sq(p));
}
pt translate(pt v,pt p){ /*To translate an
object by a vector v*/
    return p+v;
}
pt scale(pt c,double factor,pt p){/*To scale an
object by a certain ratio factor around a
center*/
    return c+(p-c)*factor;
}
pt rot(pt p,double a){/*To rotate a point by
angle a*/
    return{p.x*cos(a)-p.y*sin(a),p.x*sin(a)+p.y*
cos(a)};
}
pt perp(pt p){/*To rotate a point 90 degree*/
    return{-p.y,p.x};
}
pt linearTransfo(pt p,pt q,pt r,pt fp,pt
fq){/*so far don't know about that function*/
    return fp+(r-p)*(fq-fp)/(q-p);
}
T dot(pt v,pt w){

    return v.x*w.x+v.y*w.y;
}
bool isPerp(pt v,pt w){
    return dot(v,w)==0;
}
double angle(pt v,pt w){/*Find the smallest
angle of two vector*/
    double cosTheta=dot(v,w)/Abs(v)/Abs(w);
    return acos(max(-1.0,min(1.0,cosTheta)));
}
T cross(pt v,pt w){
    return v.x*w.y-v.y*w.x;
}
T orient(pt a,pt b,pt c){
    return cross(b-a,c-a); /*if c is left
side+ve, c is right side-ve, on line 0*/
}
bool inAngle(pt a,pt b,pt c,pt p){/*if p is in
the angle*/
    assert(orient(a,b,c)!=0);
    if(orient(a,b,c)<0)
        swap(b,c);
    return orient(a,b,p)>=0 && orient(a,c,p)<=0;
}
double orientedAngle(pt a,pt b,pt c){/*the
actual angle from ab to ac*/
    if(orient(a,b,c)>=0)
        return angle(b-a,c-a);
    else
        return 2*M_PI-angle(b-a,c-a);
}
///line
struct line{
    pt v;
    T c;
    line(){}
    line(pt p,pt q){/*From points P and Q*/
        v=(q-p),this->c=cross(v,p);
    }
    line(T a,T b,T c){/*From equation ax+by=c*/
        v=pt(b,-a),this->c=c;
    }
    line(pt v,T c){/*From direction vector v and
offset c*/
        this->v=v,this->c=c;
    }
    double getY(double x){/*self made, not sure if
it is okay*/
        assert(v.x!=0);
        double ret=(double)(c+v.y*x)/v.x;
        return ret;
    }
    double getX(double y){/*self made, not sure if
it is okay*/
        assert(v.y!=0);
        double ret=(double)(c-v.x*y)/-v.y;
        return ret;
    }
}
```

```

}
T side(pt p){/*which side a point is*/
    return cross(v,p)-c;
}
double dist(pt p){/*point to line dist*/
    return abs(side(p))/Abs(v);
}
double sqDist(pt p){/*square dist*/
    return side(p)*side(p)/(double)sq(v);
}
line perpThrough(pt p){/*perpendicular line
with point p*/
    return line(p,p+perp(v));
}
bool cmpProj(pt p,pt q){/*compare function to
sort points on a line*/
    return dot(v,p)<dot(v,q);
}
line translate(pt t){/*translate with vector
t*/
    return line(v,c+cross(v,t));
}
line shiftLeft(double dist){/*translate with
distance dist*/
    return line(v,c+dist*Abs(v));
}
pt proj(pt p){
    return p-perp(v)*side(p)/sq(v);
}
pt refl(pt p){
    return p-perp(v)*2*side(p)/sq(v);
}
bool areParallel(line l1,line l2){
    return(l1.v.x*l2.v.y==l1.v.y*l2.v.x);
}
bool areSame(line l1,line l2){
    return
    areParallel(l1,l2)and(l1.v.x*l2.c==l2.v.x*
    l1.c)and(l1.v.y*l2.c==l2.v.y*l1.c);
}
bool inter(line l1,line l2,pt& out){
    T d=cross(l1.v,l2.v);
    if(d==0) return false;
    out=(l2.v*l1.c-l1.v*l2.c)/d;
    return true;
}
line intBisector(line l1,line l2,bool
    interior){/*if change sign then returns the
other one*/
    assert(cross(l1.v,l2.v)!=0);
    double sign=interior?1:-1;
    return
    line(l2.v/Abs(l2.v)+l1.v*sign/Abs(l1.v),
        l2.c/Abs(l2.v)+l1.c*sign/Abs(l1.v));
}
//segment

```

```

bool inDisk(pt a,pt b,pt p){/*check whether
point p is in diameter AB*/
    return dot(a-p,b-p)<=0;
}
bool onSegment(pt a,pt b,pt p){/*check whether
point p is in segment AB*/
    return orient(a,b,p)==0 and inDisk(a,b,p);
}
bool properInter(pt a,pt b,pt c,pt d,pt& i){
    double oa=orient(c,d,a),
        ob=orient(c,d,b),
        oc=orient(a,b,c),
        od=orient(a,b,d);
    //Proper intersection exists iff opposite signs
    if(oa*ob<0 and oc*od<0){
        i=(a*ob-b*oa)/(ob-oa);
        return 1;
    }
    return 0;
}
/*To create sets of points we need a comparison
function*/
struct cmpX{
    bool operator()(pt a,pt b){
        return
        make_pair(a.x,a.y)<make_pair(b.x,b.y);
    }
    set<pt,cmpX>inters(pt a,pt b,pt c,pt d){
        pt out;
        if(properInter(a,b,c,d,out))
            return{out};
        set<pt,cmpX>s;
        if(onSegment(c,d,a))s.insert(a);
        if(onSegment(c,d,b))s.insert(b);
        if(onSegment(a,b,c))s.insert(c);
        if(onSegment(a,b,d))s.insert(d);
        return s;
    }
    bool LineSegInter(line l,pt a,pt b,pt& out){
        if(l.side(a)*l.side(b)>eps) return 0;
        return inter(l,line(a,b),out);
    }
    double segPoint(pt a,pt b,pt p){/*returns
    distance from a point p to segment AB*/
        if(a!=b){
            line l(a,b);
            if(l.getCmpProj(a,p)and l.getCmpProj(p,b))
                return l.dist(p);
        }
        return min(Abs(p-a),Abs(p-b));
    }
    double segSeg(pt a,pt b,pt c,pt d){/*returns
    distance from a segment AB to segment CD*/
        pt dummy;
        if(properInter(a,b,c,d,dummy))return 0;

```

```

        return
        min(min(min(segPoint(a,b,c),segPoint(a,b,
        d)),segPoint(c,d,a)),segPoint(c,d,b));
    }
    /*int latticePoints(pt a,pt b){
        // requires int representation
        return __gcd(abs(a.x-b.x),abs(a.y-b.y))+1;
    } // A = i+(b/2)-1; here
    A=area, i=pointsinside, b=points on line
    Polygon*/
    bool isConvex(vector<pt>&p){
        bool hasPos=0,hasNeg=0;
        for(int i=0,n=p.size();i<n;i++){
            int o=orient(p[i],p[(i+1)%n],p[(i+2)%n]);
            if(o>0)hasPos=1;
            if(o<0)hasNeg=true;
        }
        return!(hasPos and hasNeg);
    }
    double areaTriangle(pt a,pt b,pt c){
        return abs(cross(b-a,c-a))/2.0;
    }
    double areaPolygon(const vector<pt>&p){
        double area=0.0;
        for(int i=0,n=p.size();i<n;i++){
            area+=cross(p[i],p[(i+1)%n]);
        }
        return fabs(area)/2.0;
    }
    bool pointInPolygon(const vector<pt>&p,pt
    q){/*returns true if pt q is in polygon p*/
        bool c=false;
        for(int i=0,n=p.size();i<n;i++){
            int j=(i+1)%p.size();
            if((p[i].y<=q.y and q.y<p[j].y or
            p[j].y<=q.y and q.y<p[i].y)and
            q.x<p[i].x+(p[j].x-p[i].x)*(q.y-p[i].y)/
            (p[j].y-p[i].y))
                c=!c;
        }
        return c;
    }
    ll is_point_in_convex(vector<pt>& p, pt &x) { // O(log n)
        ll n = p.size(); /*this function from
        YouKnowWho*/
        if (n < 3) return 1;
        ll a =orient(p[0], p[1], x), b =
        orient(p[0], p[n - 1], x);
        if (a < 0 || b > 0) return 1;
        ll l = 1, r = n - 1;
        while (l + 1 < r) {
            int mid = l + r >> 1;
            if (orient(p[0], p[mid], x) >= 0) l =
            mid;
            else r = mid;
        }
        ll k = orient(p[l], p[r], x);
    }

```

```

if (k <= 0) return -k;
if (l == 1 && a == 0) return 0;
if (r == n - 1 && b == 0) return 0;
return -1;
}
pt centroidPolygon(vector<pt>&p){/*from rezaul,i
don't know about that*/
pt c(0,0);
double scale=6.0*areaPolygon(p);
// if(scale<eps) return c;
for(int i=0,n=p.size();i<n;i++){
    int j=(i+1)%n;
    c=c+(p[i]+p[j])*cross(p[i],p[j]);
}
return c/scale;
}
//Circle
pt circumCenter(pt a,pt b,pt c){/*return the
center of the circle go through point a,b,c*/
b=b-a,c=c-a;
assert(cross(b,c)!=0);
return a+perp(b*sq(c)-c*sq(b))/cross(b,c)/2;
}
bool circle2PtsRad(pt p1,pt p2,double r,pt& c){
double d2=sq(p1-p2);
double det=r*r/d2-0.25;
if(det<0.0) return false;
double h=sqrt(det);
c.x=(p1.x+p2.x)*0.5+(p1.y-p2.y)*h;
c.y=(p1.y+p2.y)*0.5+(p2.x-p1.x)*h;
return true;
}
int circleLine(pt c,double r,line l,pair<pt,pt>&
out){/*circle line intersection*/
double h2=r*r-l.sqDist(c);
if(h2<0) return 0; /*the line doesn't touch the
circle;*/
pt p=l.proj(c);
pt h=l.v*sqrt(h2)/Abs(l.v);
out=make_pair(p-h,p+h);
return 1+(h2>0);
}
int circleCircle(pt c1,double r1,pt c2,double
r2,pair<pt,pt>& out){/*circle circle
intersection*/
pt d=c2-c1;
double d2=sq(d);
if(d2==0){//concentric circles
    assert(r1!=r2);
    return 0;
}
double pd=(d2+r1*r1-r2*r2)/2;
double h2=r1*r1-pd*pd/d2;//h ^ 2
if(h2<0) return 0;
pt p=c1+d*pd/d2,h=perp(d)*sqrt(h2/d2);
out=make_pair(p-h,p+h);
return 1+h2>0;
}

```

```

int tangents(pt c1,double r1,pt c2,double
r2,bool inner,vector<pair<pt,pt>>&out){
if(inner)r2=-r2; /*returns tangent(the line
which touch a circle in one point)of two
circles*/
pt d=c2-c1; /*the same code can be used to find
the tangent to a circle passing through a
point by setting r2 to 0*/
double dr=r1-r2,d2=sq(d),h2=d2-dr*dr;
if(d2==0 or h2<0){
    assert(h2!=0);
    return 0;
}
for(int sign :{-1,1}){
    pt v=pt(d*dr+perp(d)*sqrt(h2)*sign)/d2;
    out.push_back(make_pair(c1+v*r1,c2+v*r2));
}
return 1+(h2>0);
}
//Convex Hull-Monotone Chain
pt H[100000+5];
vector<pt>monotoneChain(vector<pt>&points){
sort(points.begin(),points.end());
vector<pt>ret;
ret.clear();
int st=0;
for(int i=0,sz=points.size();i<sz;i++){
    while(st>=2 and
orient(H[st-2],H[st-1],points[i])<0)st--;
    H[st++]=points[i];
}
int taken=st-1;
for(int i=points.size()-2;i>=0;i--){
    while(st>=taken+2 and
orient(H[st-2],H[st-1],points[i])<0)st--;
    H[st++]=points[i];
}
for(int i=0;i<st;i++)ret.push_back(H[i]);
return ret;
}
//Convex Hull-Monotone Chain from you_know_who
int sz;
vector<pt> monotoneChain(vector<pt> &v) {
    if(v.size()==1) return v;
    sort(v.begin(), v.end());
    vector<pt> up(2*v.size()+2),
    down(2*v.size()+2);
    int szup=0, szdw=0;
    for(int i=0;i<v.size();i++) {
        while(szup>1 && orient(up[szup-2],
        up[szup-1], v[i])>0)
            szup--;
        while(szdw>1 && orient(down[szdw-2],
        down[szdw-1], v[i])<0)
            szdw--;
        up[szup++]=v[i];
        down[szdw++]=v[i];
    }
}

```

```

if(szdw>1) szdw--;
reverse(up.begin(), up.begin()+szup);
for(int i=0;i<szup-1;i++) down[szdw++]=
up[i];
if(szdw==2 && down[0].x==down[1].x &&
down[0].y==down[1].y)
    szdw--;
sz = szdw;
return down;
}
double cosA(double a,double b,double c){
    double val=b*b+c*c-a*a;
    val/= (2*b*c);
    return acos(val);
}
double triangle(double a,double b,double c){
    double s=(a+b+c)/2;
    return sqrtl(s*(s-a)*(s-b)*(s-c));
}
}
using namespace Geometry;

```

5.2 Rotation Matrix [39 lines]

```

struct { double x; double y; double z; } Point;
double rMat[4][4];
double inMat[4][1] = {0.0, 0.0, 0.0, 0.0};
double outMat[4][1] = {0.0, 0.0, 0.0, 0.0};
void mulMat() {
    for(int i = 0; i < 4; i++) {
        for(int j = 0; j < 1; j++) {
            outMat[i][j] = 0;
            for(int k = 0; k < 4; k++)
                outMat[i][j] += rMat[i][k] * inMat[k][j];
        }
    }
}
void setMat(double ang, double u, double v, double
w){
    double L = (u * u + v * v + w * w);
    ang = ang * PI / 180.0; /*converting to radian
    value*/
    double u2 = u*u; double v2 = v*v; double w2 =
    w*w;
    rMat[0][0]=(u2+(v2+w2)*cos(ang))/L;
    rMat[0][1]=(u*v*(1-cos(ang))-w*sqrt(L)*_
    sin(ang))/L;
    rMat[0][2]=(u*w*(1-cos(ang))+v*sqrt(L)*_
    sin(ang))/L;
    rMat[0][3]=0.0;
    rMat[1][0]=(u*v*(1-cos(ang))+w*sqrt(L)*_
    sin(ang))/L;
    rMat[1][1]=(v2+(u2+w2)*cos(ang))/L;
    rMat[1][2]=(v*w*(1-cos(ang))-u*sqrt(L)*_
    sin(ang))/L;
    rMat[1][3]=0.0;
    rMat[2][0]=(u*w*(1-cos(ang))-v*sqrt(L)*_
    sin(ang))/L;
    rMat[2][1]=(u*v*(1-cos(ang))-w*sqrt(L)*_
    sin(ang))/L;
    rMat[2][2]=(u*u*(1-cos(ang))+v*v+_
    w*w)*cos(ang))/L;
    rMat[2][3]=0.0;
}

```

```

rMat[2][1]=(v*w*(1-cos(ang))+u*sqrt(L)*_
sin(ang))/L;
rMat[2][2]=(w2 + (u2 + v2) * cos(ang)) / L;
rMat[2][3]=0.0; rMat[3][0]=0.0; rMat[3][1]=0.0;
rMat[3][2]=0.0; rMat[3][3]=1.0;
}
/*double ang;
double u, v, w; //points = the point to be
rotated
Point point, rotated; //u,v,w=unit vector of
line
inMat[0][0] = points.x; inMat[1][0] = points.y;
inMat[2][0] = points.z; inMat[3][0] = 1.0;
setMat(ang, u, v, w); mulMat();
rotated.x = outMat[0][0]; rotated.y =
outMat[1][0];
rotated.z = outMat[2][0];*/

```

6 Graph

6.1 2SAT [92 lines]

```

struct TwoSat {
    vector<bool> vis;
    vector<vector<int>> adj, radj;
    vector<int> dfs_t, ord, par;
    int n, intime; //For n node there will be 2*n
    node in SAT.
    void init(int N) {
        n = N;
        intime = 0;
        vis.assign(N * 2 + 1, false);
        adj.assign(N * 2 + 1, vector<int>());
        radj.assign(N * 2 + 1, vector<int>());
        dfs_t.resize(N * 2 + 1);
        ord.resize(N * 2 + 1);
        par.resize(N * 2 + 1);
    }
    inline int neg(int x) {
        return x <= n ? x + n : x - n;
    }
    inline void add_implication(int a, int b) {
        if (a < 0) a = n - a;
        if (b < 0) b = n - b;
        adj[a].push_back(b);
        radj[b].push_back(a);
    }
    inline void add_or(int a, int b) {
        add_implication(-a, b);
        add_implication(-b, a);
    }
    inline void add_xor(int a, int b) {
        add_or(a, b);
        add_or(-a, -b);
    }
    inline void add_and(int a, int b) {
        add_or(a, b);
        add_or(a, -b);
        add_or(-a, b);
    }
}

```

```

inline void force_true(int x) {
    if (x < 0) x = n - x;
    add_implication(neg(x), x);
}
inline void add_xnor(int a, int b) {
    add_or(a, -b);
    add_or(-a, b);
}
inline void add_nand(int a, int b) {
    add_or(-a, -b);
}
inline void add_nor(int a, int b) {
    add_and(-a, -b);
}
inline void force_false(int x) {
    if (x < 0) x = n - x;
    add_implication(x, neg(x));
}
inline void topsort(int u) {
    vis[u] = 1;
    for (int v : radj[u]) if (!vis[v]) topsort(v);
    dfs_t[u] = ++intime;
}
inline void dfs(int u, int p) {
    par[u] = p, vis[u] = 1;
    for (int v : adj[u]) if (!vis[v]) dfs(v, p);
}
void build() {
    int i, x;
    for (i = n * 2, intime = 0; i >= 1; i--) {
        if (!vis[i]) topsort(i);
        ord[dfs_t[i]] = i;
    }
    vis.assign(n * 2 + 1, 0);
    for (i = n * 2; i > 0; i--) {
        x = ord[i];
        if (!vis[x]) dfs(x, x);
    }
}
bool satisfy(vector<int>& ret) //ret contains the
value that are true if the graph is
satisfiable.
{
    build();
    vis.assign(n * 2 + 1, 0);
    for (int i = 1; i <= n * 2; i++) {
        int x = ord[i];
        if (par[x] == par[neg(x)]) return 0;
        if (!vis[par[x]]) {
            vis[par[x]] = 1;
            vis[par[neg(x)]] = 0;
        }
    }
    for (int i = 1; i <= n; i++) if (vis[par[i]])
        ret.push_back(i);
    return 1;
}

```

6.2 BellmanFord [57 lines]

```

#include<bits/stdc++.h>
using namespace std;
const int mx = 1e5+6;
const int INF = 0x3f3f3f3f;
struct edge{
    int u,v;
    int cost;
};
vector<edge> e;
vector<int> path(mx);
int dist[mx];
/*
Time-complexity: O(|V|*|E|)
Space-complexity: O(|V|)
To find any negative cycle assign dist[i] = 0
for all.
We can use floyd-warshall algorithm to find
negative cycle too.
*Handle LL carefully.
*/
void bellmanford(int s, int n){
    int m = e.size();
    memset(dist, 0x3f3f3f3f, sizeof dist);
    dist[s] = 0;
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (int j = 0; j < m; ++j)
            if (dist[e[j].u] < INF)
                if (dist[e[j].v] > dist[e[j].u] +
                    e[j].cost) {
                    dist[e[j].v] = max(-INF, dist[e[j].u] +
                        e[j].cost);
                    path[e[j].v] = e[j].u;
                    x = e[j].v;
                }
    }
    if (x == -1)
        cout << "No negative cycle from " << s;
    else {
        int y = x;
        for (int i = 0; i < n; ++i)
            y = path[y];
        vector<int> path;
        for (int cur = y; ; cur = path[cur]) {
            path.push_back(cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse(path.begin(), path.end());
        cout << "Negative cycle: ";
        for (size_t i = 0; i < path.size(); ++i)

```

```

        cout << path[i] << ' ';
    }
}

int main(){
}

6.3 BridgeTree [66 lines]
int N, M, timer, compid;
vector<pair<int, int>> g[mx];
bool used[mx], isBridge[mx];
int comp[mx], tin[mx], minAncestor[mx];
vector<int> Tree[mx]; // Store 2-edge-connected
// component tree. (Bridge tree).
void markBridge(int v, int p) {
    tin[v] = minAncestor[v] = ++timer;
    used[v] = 1;
    for (auto& e : g[v]) {
        int to, id;
        tie(to, id) = e;
        if (to == p) continue;
        if (used[to]) minAncestor[v] =
            min(minAncestor[v], tin[to]);
        else {
            markBridge(to, v);
            minAncestor[v] = min(minAncestor[v],
            minAncestor[to]);
            if (minAncestor[to] > tin[v]) isBridge[id] =
                true;
            // if (tin[u] <= minAncestor[v]) ap[u] = 1;
        }
    }
}
void markComp(int v, int p) {
    used[v] = 1;
    comp[v] = compid;
    for (auto& e : g[v]) {
        int to, id;
        tie(to, id) = e;
        if (isBridge[id]) continue;
        if (used[to]) continue;
        markComp(to, v);
    }
}
vector<pair<int, int>> edges;
void addEdge(int from, int to, int id) {
    g[from].push_back({ to, id });
    g[to].push_back({ from, id });
    edges[id] = { from, to };
}
void initB() {
    for (int i = 0; i <= compid; ++i)
        Tree[i].clear();
    for (int i = 1; i <= N; ++i) used[i] = false;
    for (int i = 1; i <= M; ++i) isBridge[i] =
        false;
    timer = compid = 0;
}
void bridge_tree() {

```

```

    initB();
    markBridge(1, -1); //Assuming graph is
    //connected.
    for (int i = 1; i <= N; ++i) used[i] = 0;
    for (int i = 1; i <= N; ++i) {
        if (!used[i]) {
            markComp(i, -1);
            ++compid;
        }
    }
    for (int i = 1; i <= M; ++i) {
        if (isBridge[i]) {
            int u, v;
            tie(u, v) = edges[i];
            // connect two components using edge.
            Tree[comp[u]].push_back(comp[v]);
            Tree[comp[v]].push_back(comp[u]);
            int x = comp[u];
            int y = comp[v];
        }
    }
}

6.4 Centroid Decomposition [49 lines]
ll n,subsize[mx];
vector<ll>adj[mx];
char ans[mx];
bool brk[mx];
void calculatesize(ll u,ll par){
    subsize[u]=1;
    for(ll i=0;i<(ll)adj[u].size();i++){
        ll v=adj[u][i];
        if(v==par or brk[v]==true)continue;
        calculatesize(v,u);
        subsize[u]+=subsize[v];
    }
}
ll getcentroid(ll u,ll par,ll n){
    ll ret=u;
    for(ll i=0;i<(ll)adj[u].size();i++){
        ll v=adj[u][i];
        if(v==par or brk[v]==true)continue;
        if(subsize[v]>(n/2)){
            ret=getcentroid(v,u,n);
            break;
        }
    }
    return ret;
}
void decompose(ll u,char rank){
    calculatesize(u,-1);
    ll c=getcentroid(u,-1,subsize[u]);
    brk[c]=true;
    ans[c]=rank;
    for(ll i=0;i<(ll)adj[c].size();i++){
        ll v=adj[c][i];
        if(brk[v]==true)continue;
        decompose(v,rank+1);
    }
}
```

```

}

int main(){
    scanf("%lld",&n);
    for(ll i=0;i<n-1;i++){
        ll a,b;
        scanf("%lld %lld",&a,&b);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    decompose(1,'A');
    for(ll i=1;i<=n;i++){
        printf("%c",ans[i]);
    }
}

6.5 DSU on Tree [69 lines]
int n;
//extra data you need
vector<int> adj[mxn];
vector<int> *dsu[mxn];
void call(int u, int p=-1){
    sz[u] = 1;
    for(auto v: adj[u]){
        if(v != p){
            dep[v] = dep[u]+1;
            call(v, u);
            sz[u] += sz[v];
        }
    }
}
void dfs(int u, int p = -1, int isb = 1){
    int mx=-1, big=-1;
    for(auto v: adj[u]){
        if(v != p && sz[v]>mx){
            mx = sz[v];
            big = v;
        }
    }
    for(auto v: adj[u]){
        if(v != p && v != big){
            dfs(v, u, 0);
        }
    }
    if(big != -1){
        dfs(big, u, 1);
        dsu[u] = dsu[big];
    }
    else{
        dsu[u] = new vector<int>();
    }
    dsu[u]->push_back(u);
    //calculation
    for(auto v: adj[u]){
        if(v == p || v == big) continue;

```

```

    for(auto x: *dsu[v]){
        dsu[u]->push_back(x);
        //calculation
    }
}

//calculate ans for node u

if(isb == 0){
    for(auto x: *dsu[u]){
        //reverse calculation
    }
}

int main()
{
    scanf("%d", &n);
    for(int i=1; i<n; i++){
        int u, v;
        scanf("%d %d", &u, &v);
        adj[u].pb(v);
        adj[v].pb(u);
    }

    dep[1] = 1;
    call(1);
    dfs(1);
}

```

6.6 Dijkstra [33 lines]

```

#include<bits/stdc++.h>
#define ff first
#define ss second
using namespace std;
const int mx = 1e5 + 5;
using ll = long long;
using pll = pair<ll, ll>;
vector<pll>adj[mx];
int dis[mx];
bool vis[mx];
//Complexity O(V+ElogV)
void Dijkstra(int src) {
    priority_queue<pll, vector<pll>, greater<pll>>
        >pq;
    pq.push({ 0,src });
    memset(dis, 0x3f3f3f3f, sizeof dis);
    memset(vis, 0, sizeof vis);
    dis[src] = 0;
    while (!pq.empty()) {
        int u = pq.top().ss;
        pq.pop();
        if (vis[u]) continue;
        vis[u] = true;
        for (auto v : adj[u]) {
            if (dis[v.ss] > dis[u] + v.ff) {
                dis[v.ss] = dis[u] + v.ff;
                pq.push({ dis[v.ss],v.ss });
            }
        }
    }
}

```

```

    }
}

int main() {
}



---



### 6.7 Heavy Light Decomposition [73 lines]



/*Heavy Light Decomposition  
Build Complexity O(n)  
Query Complexity O(lg^2 n)  
Call init() with number of nodes  
It's probably for the best to not do "using  
namespace hld"*/



```

namespace hld {
 //N is the maximum number of nodes
 /*par,lev,size corresponds to
 parent,depth,subtree-size*/
 //head[u] is the starting node of the chain u is
 //in
 //in[u] to out[u] keeps the subtree indices
 const int N=100000+7;
 vector<int>g[N];
 int par[N],lev[N],head[N],size[N],in[N],out[N];
 int cur_pos,n;
 //returns the size of subtree rooted at u
 /*maintains the child with the largest subtree
 at the front of g[u]*/
 //WARNING: Don't change anything here specially
 //with size[] if Jon Snow
 int dfs(int u,int p){
 size[u]=1,par[u]=p;
 lev[u]=lev[p]+1;
 for(auto &v : g[u]){
 if(v==p)continue;
 size[u]+=dfs(v,u);
 if(size[v]>size[g[u].front()]){
 swap(v,g[u].front());
 }
 }
 return size[u];
 }
 //decomposed the tree in an array
 //note that there is no physical array here
 void decompose(int u,int p){
 in[u]=++cur_pos;
 for(auto &v : g[u]){
 if(v==p)continue;
 head[v]=(v==g[u].front()? head[u]: v);
 decompose(v,u);
 }
 out[u]=cur_pos;
 }
 //initializes the structure with _n nodes
 void init(int _n,int root=1){
 n=_n;
 cur_pos=0;
 dfs(root,0);
 head[root]=root;
 }
}

```


```

```

    decompose(root,0);
}

//checks whether p is an ancestor of u
bool isances(int p,int u){
    return in[p]<=in[u] and out[u]<=out[p];
}

//Returns the maximum node value in the path u-v
ll query(int u,int v){
    ll ret=-INF;
    while(!isances(head[u],v)){
        ret=max(ret,seg.query(1,1,n,in[head[u]],in[u]));
        u=par[head[u]];
    }
    swap(u,v);
    while(!isances(head[u],v)){
        ret=max(ret,seg.query(1,1,n,in[head[u]],in[u]));
        u=par[head[u]];
    }
    if(in[v]<in[u])swap(u,v);
    ret=max(ret,seg.query(1,1,n,in[u],in[v]));
    return ret;
}

//Adds val to subtree of u
void update(int u,ll val){
    seg.update(1,1,n,in[u],out[u],val);
}

```

6.8 K'th Shortest path [40 lines]

```

int m,n,deg[MM],source,sink,K,val[MM][12];
struct edge{
    int v,w;
}adj[MM][500];
struct info{
    int v,w,k;
    bool operator<(const info &b) const{
        return w>b.w;
    }
};
priority_queue<info,vector<info>>Q;
void kthBestShortestPath(){
    int i,j;
    info u,v;
    for(i=0;i<n;i++)
        for(j=0;j<K;j++)val[i][j]=inf;
    u.v=source,u.k=0,u.w=0;
    Q.push(u);
    while(!Q.empty()){
        u=Q.top();
        Q.pop();
        for(i=0;i<deg[u.v];i++){
            v.v=adj[u.v][i].v;
            int cost=adj[u.v][i].w+u.w;
            for(v.k=u.k;v.k<K;v.k++){

```

```

if(cost==inf)break;
if(val[v.v][v.k]>cost){
    swap(cost,val[v.v][v.k]);
    v.w=val[v.v][v.k];
    Q.push(v);
    break;
}
}
for(v.k++;v.k<K;v.k++){
    if(cost==inf)break;
    if(val[v.v][v.k]>cost)swap(cost,
    val[v.v][v.k]);
}
}

6.9 LCA [46 lines]
const int Lg = 22;
vector<int>adj[mx];
int level[mx];
int dp[Lg][mx];
void dfs(int u) {
    for (int i = 1;i < Lg;i++)
        dp[i][u] = dp[i - 1][dp[i - 1][u]];
    for (int v : adj[u]) {
        if (dp[0][u] == v)continue;
        level[v] = level[u] + 1;
        dp[0][v] = u;
        dfs(v);
    }
}
int lca(int u, int v) {
    if (level[v] < level[u])swap(u, v);
    int diff = level[v] - level[u];
    for (int i = 0;i < Lg;i++)
        if (diff & (1 << i))
            v = dp[i][v];
    for (int i = Lg - 1;i >= 0;i--)
        if (dp[i][u] != dp[i][v])
            u = dp[i][u], v = dp[i][v];
    return u == v ? u : dp[0][u];
}
int kth(int u, int k) {
    for (int i = Lg - 1;i >= 0;i--)
        if (k & (1 << i))
            u = dp[i][u];
    return u;
}
//kth node from u to v. 0th is u.
int go(int u, int v, int k) {
    int l = lca(u, v);
    int d = level[u] + level[v] - (level[l] << 1);
    assert(k <= d);
    if (level[l] + k <= level[u]) return kth(u, k);
    k -= level[u] - level[l];
    return kth(v, level[v] - level[l] - k);
}
/*

```

```

LCA( $u, v$ ) with root  $r$ :  

lca( $u, v$ ) ^ lca( $u, r$ ) ^ lca( $v, r$ )  

Distance between  $u, v$ :  

level( $u$ ) + level( $v$ ) - 2 * level(lca( $u, v$ ))  

*/  

6.10 SACK [50 lines]  

int sz[maxn];
void getsz(int v, int p){
    sz[v] = 1;
    for(auto u : g[v])
        if(u!=p){
            getsz(u, v);
            sz[v] += sz[u];
        }
}
//SACK O(nlog^2n)
map<int,int>*cnt[maxn];
void dfs(int v, int p){
    int mx=-1,bigChild=-1;
    for(auto u : g[v])
        if(u!=p){
            dfs(u, v);
            if(sz[u]>mx)mx=sz[u],bigChild=u;
        }
    if(bigChild!=-1)cnt[v]=cnt[bigChild];
    else cnt[v]=new map<int,int>();
    (*cnt[v])[col[v]]++;
    for(auto u : g[v])
        if(u!=p && u!=bigChild){
            for(auto x : *cnt[u])
                (*cnt[v])[x.first]+=x.second;
        }
}
//SACK-O(nlogn)
vector<int>*vec[maxn];
int cnt[maxn];
void dfs(int v, int p, bool keep){
    int mx=-1,bigChild=-1;
    for(auto u : g[v])
        if(u!=p&&sz[u]>mx)mx=sz[u],bigChild=u;
    for(auto u : g[v])
        if(u!=p && u!=bigChild)dfs(u, v, 0);
    if(bigChild!=-1)
        dfs(bigChild, v, 1), vec[v]=vec[bigChild];
    else vec[v]=new vector<int>();
    vec[v]->push_back(v);cnt[col[v]]++;
    for(auto u : g[v])
        if(u!=p && u!=bigChild)
            for(auto x : *vec[u]){
                cnt[col[x]]++;
                vec[v]->push_back(x);
            }
}
/*in this step*vec[v]contains all of the subtree
of vertex v.*/
if(keep==0)
    for(auto u : *vec[v])cnt[col[u]]--;
}

```

6.11 SCC [43 lines]

/*components: number of SCC.
sz: size of each SCC.
comp: component number of each node.
Create reverse graph.
Run find_scc() to find SCC.
Might need to create condensation graph by
create_condensed().
Think about indeg/outdeg
for multiple test cases- clear
adj/radj/comp/vis/sz/topo/condensed.*/
vector<int>adj[mx], radj[mx];

```

int comp[mx], vis[mx], sz[mx], components;
vector<int>topo;
void dfs(int u) {
    vis[u] = 1;
    for (int v : adj[u])
        if (!vis[v]) dfs(v);
    topo.push_back(u);
}
void dfs2(int u, int val) {
    comp[u] = val;
    sz[val]++;
    for (int v : radj[u])
        if (comp[v] == -1)
            dfs2(v, val);
}
void find_scc(int n) {
    memset(vis, 0, sizeof vis);
    memset(comp, -1, sizeof comp);
    for (int i = 1;i <= n;i++)
        if (!vis[i])
            dfs(i);
    reverse(topo.begin(), topo.end());
    for (int u : topo)
        if (comp[u] == -1)
            dfs2(u, ++components);
}
vector<int>condensed[mx];
void create_condensed(int n) {
    for (int i = 1;i <= n;i++)
        for (int v : adj[i])
            if (comp[i] != comp[v])
                condensed[comp[i]].push_back(comp[v]);
}

```

7 Math

7.1 Big Sum [13 lines]

```

11 bigsum(11 a, 11 b, 11 m) {
    if (b == 0) return 0;
    11 sum; a %≡ m;
    if (b & 1) {
        sum = bigsum((a * a) % m, (b - 1) / 2, m);
        sum = (sum + (a * sum) % m) % m;
        sum = (1 + (a * sum) % m) % m;
    }
}

```

```

} else {
    sum = bigsum((a * a) % m, b / 2, m);
    sum = (sum + (a * sum) % m) % m;
}
return sum;
}

7.2 CRT [52 lines]
ll ext_gcd(ll A, ll B, ll* X, ll* Y) {
    ll x2, y2, x1, y1, x, y, r2, r1, q, r;
    x2 = 1; y2 = 0;
    x1 = 0; y1 = 1;
    for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r,
        x2 = x1, y2 = y1, x1 = x, y1 = y) {
        q = r2 / r1;
        r = r2 % r1;
        x = x2 - (q * x1);
        y = y2 - (q * y1);
    }
    *X = x2; *Y = y2;
    return r2;
}
/*-----BlackBox-----*/

```

```

class ChineseRemainderTheorem {
    typedef long long vlong;
    typedef pair<vlong, vlong> pll;
    /** CRT Equations stored as pairs of vector. See
     * addEquation()*/
    vector<pll> equations;
    public:
    void clear() {
        equations.clear();
    }
    /** Add equation of the form  $x = r \pmod{m}$  */
    void addEquation(vlong r, vlong m) {
        equations.push_back({r, m});
    }
    pll solve() {
        if (equations.size() == 0) return { -1,-1 };
        // No equations to solve
        vlong a1 = equations[0].first;
        vlong m1 = equations[0].second;
        a1 %= m1;
        /** Initially  $x = a_0 \pmod{m_0}$  */
        /** Merge the solution with remaining
         * equations */
        for (int i = 1; i < equations.size(); i++) {
            vlong a2 = equations[i].first;
            vlong m2 = equations[i].second;
            vlong g = __gcd(m1, m2);
            if (a1 % g != a2 % g) return { -1,-1 }; /**
             * Conflict in equations
             * Merge the two equations*/
            vlong p, q;
            ext_gcd(m1 / g, m2 / g, &p, &q);
            vlong mod = m1 / g * m2;
            vlong x = ((__int128)a1 * (m2 / g) % mod * q
            % mod + (__int128)a2 * (m1 / g) % mod * p %
            mod) % mod;
        }
    }
}

```

```

** Merged equation*/
a1 = x;
if (a1 < 0) a1 += mod;
m1 = mod;
}
return { a1, m1 };
}
7.3 FFT [85 lines]
template<typename float_t>
struct mycomplex {
    float_t x, y;
    mycomplex<float_t>(float_t _x = 0, float_t _y = 0) : x(_x), y(_y) {}
    float_t real() const { return x; }
    float_t imag() const { return y; }
    void real(float_t _x) { x = _x; }
    void imag(float_t _y) { y = _y; }
    mycomplex<float_t>& operator+=(const
        mycomplex<float_t> &other) { x += other.x; y
        += other.y; return *this; }
    mycomplex<float_t>& operator-=(const
        mycomplex<float_t> &other) { x -= other.x; y
        -= other.y; return *this; }
    mycomplex<float_t> operator+(const
        mycomplex<float_t> &other) const { return
        mycomplex<float_t>(*this) += other; }
    mycomplex<float_t> operator-(const
        mycomplex<float_t> &other) const { return
        mycomplex<float_t>(*this) -= other; }
    mycomplex<float_t> operator*(const
        mycomplex<float_t> &other) const {
        return {x * other.x - y * other.y, x * other.y
        + other.x * y}; }
    mycomplex<float_t> operator*(float_t mult) const
    {
        return {x * mult, y * mult};
    }
    friend mycomplex<float_t> conj(const
        mycomplex<float_t> &c) {
        return {c.x, -c.y};
    }
    friend ostream& operator<<(ostream &stream,
        const mycomplex<float_t> &c) {
        return stream << '(' << c.x << ", " << c.y <<
        ')';
    }
};

using cd = mycomplex<double>;
void fft(vector<cd> &a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1)
            j ^= bit;
        j ^= bit;
        if (i < j)

```

```

            swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <= 1) {
        double ang = 2 * PI / len * (invert ? -1 : 1);
        cd wlen(cos(ang), sin(ang));
        for (int i = 0; i < n; i += len) {
            cd w(1);
            for (int j = 0; j < len / 2; j++) {
                cd u = a[i+j], v = a[i+j+len/2] * w;
                a[i+j] = u + v;
                a[i+j+len/2] = u - v;
                w = w*wlen;
            }
        }
        if (invert) {
            for (cd & x : a){
                double z = n;
                z=1/z;
                x = x*z;
            }
            // x /= n;
        }
    }
    void multiply (const vector<bool> & a, const
        vector<bool> & b, vector<bool> & res)
    { //change all the bool to your type needed
        vector<cd> fa (a.begin(), a.end()), fb
        (b.begin(), b.end());
        size_t n = 1;
        while (n < max (a.size(), b.size())) n <= 1;
        n <= 1;
        fa.resize (n), fb.resize (n);
        fft (fa, false), fft (fb, false);
        for (size_t i=0; i<n; ++i)
            fa[i] = fa[i] * fb[i];
        fft (fa, true);
        res.resize (n);
        for (size_t i=0; i<n; ++i)
            res[i] = round(fa[i].real());
        while(res.back()==0) res.pop_back();
    }
    void pow(const vector<bool> &a, vector<bool>
        &res, long long int k){
        vector<bool> po=a;
        res.resize(1);
        res[0] = 1;
        while(k){
            if(k&1){
                multiply(po, res, res);
            }
            multiply(po, po, po);
            k/=2;
        }
    }
}

```

7.4 GaussElimination [39 lines]

```
template<typename ld>
int gauss(vector<vector<ld>>& a, vector<ld>& ans)
{
    const ld EPS = 1e-9;
    int n = a.size(); //number of equations
    int m = a[0].size() - 1; //number of variables
    vector<int> where(m, -1); //indicates which row contains the solution
    int row, col;
    for (col = 0, row = 0; col < m && row < n; ++col) {
        int sel = row; //which row contains the maximum value/
        for (int i = row + 1; i < n; i++)
            if (abs(a[i][col]) > abs(a[sel][col]))
                sel = i;
        if (abs(a[sel][col]) < EPS) continue; //it's basically 0.
        a[sel].swap(a[row]); //taking the max row up
        where[col] = row;
        ld t = a[row][col];
        for (int i = col; i <= m; i++) a[row][i] /= t;
        for (int i = 0; i < n; i++) {
            if (i != row) {
                ld c = a[i][col];
                for (int j = col; j <= m; j++)
                    a[i][j] -= a[row][j] * c;
            }
        }
        row++;
    }
    ans.assign(m, 0);
    for (int i = 0; i < m; i++)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; i++) {
        ld sum = 0;
        for (int j = 0; j < m; j++)
            sum += ans[j] * a[i][j];
        if (abs(sum - a[i][m]) > EPS) //L.H.S!=R.H.S
            ans.clear(); //No solution
    }
    return row;
}
```

7.5 GaussMod2 [44 lines]

```
template<typename T>
struct Gauss {
    int bits = 60;
    vector<T> table;
    Gauss() {
        table = vector<T>(bits, 0);
    }
    //call with constructor to define bit size.
    Gauss(int _bits) {
        bits = _bits;
        table = vector<T>(bits, 0);
    }
}
```

```
}
int basis() //return rank/size of basis
{
    int ans = 0;
    for (int i = 0; i < bits; i++)
        if (table[i])
            ans++;
    return ans;
}
bool can(T x) //can x be obtained from the basis
{
    for (int i = bits - 1; i >= 0; i--)
        x = min(x, x ^ table[i]);
    return x == 0;
}
void add(T x) {
    for (int i = bits - 1; i >= 0 && x; i--) {
        if (table[i] == 0) {
            table[i] = x;
            x = 0;
        }
        else x = min(x, x ^ table[i]);
    }
}
T getBest() {
    T x = 0;
    for (int i = bits - 1; i >= 0; i--)
        x = max(x, x ^ table[i]);
    return x;
}
void Merge(Gauss& other) {
    for (int i = bits - 1; i >= 0; i--)
        add(other.table[i]);
}

```

7.6 Karatsuba Idea [5 lines]

Three subproblems:

$$\begin{aligned} a &= xH \quad yH \\ d &= xL \quad yL \\ e &= (xH + xL)(yH + yL) - a - d \end{aligned}$$

Then $xy = a \cdot rn + e \cdot rn/2 + d$

7.7 Linear Diophantine [12 lines]

```
/**x'=x+(k*B/g), y'=y-(k*A/g); infinite soln
if A=B=0, C must equal 0 and any x,y is solution;
if A/B=0, (x,y)=(C/A,k)/(k,C/B) */
bool LDE(int A, int B, int C, int*x, int*y){
    int g=gcd(A,B);
    if(C%g!=0) return false;
    int a=A/g, b=B/g, c=C/g;
    extended_gcd(a,b,x,y); //ax+by=1
    if(g<0)a=-1; b=-1; c=-1; //Ensure gcd(a,b)=1
    *x*=c; *y*=c; //ax+by=c
    return true; //Solution Exists
}
```

7.8 Matrix [100 lines]

```
template<typename T>
struct Matrix {
```

```
T MOD = 1e9 + 7; //change if necessary
T add(T a, T b) const {
    T res = a + b;
    if (res >= MOD) return res - MOD;
    return res;
}
T sub(T a, T b) const {
    T res = a - b;
    if (res < 0) return res + MOD;
    return res;
}
T mul(T a, T b) const {
    T res = a * b;
    if (res >= MOD) return res % MOD;
    return res;
}
int R, C;
vector<vector<T>> mat;
Matrix(int _R = 0, int _C = 0) {
    R = _R, C = _C;
    mat.resize(R);
    for (auto& v : mat) v.assign(C, 0);
}
void print() {
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++)
            cout << mat[i][j] << " \n"[j == C - 1];
}
void createIdentity() {
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++)
            mat[i][j] = (i == j);
}
Matrix operator+(const Matrix& o) const {
    Matrix res(R, C);
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++)
            res[i][j] = add(mat[i][j] + o.mat[i][j]);
}
Matrix operator-(const Matrix& o) const {
    Matrix res(R, C);
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < C; j++)
            res[i][j] = sub(mat[i][j] + o.mat[i][j]);
}
Matrix operator*(const Matrix& o) const {
    Matrix res(R, o.C);
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < o.C; j++) {
            for (int k = 0; k < C; k++)
                res.mat[i][j] = add(res.mat[i][j], mul(mat[i][k], o.mat[k][j]));
        }
    }
    return res;
}
Matrix pow(long long x) {
    Matrix res(R, C);
```

```

res.createIdentity();
Matrix<T> o = *this;
while (x) {
    if (x & 1) res = res * o;
    o = o * o;
    x >>= 1;
}
return res;
}

Matrix inverse() ///Only square matrix && non-zero determinant
{
    Matrix res(R, R + R);
    for (int i = 0; i < R; i++) {
        for (int j = 0; j < R; j++) {
            res.mat[i][j] = mat[i][j];
            res.mat[i][R + i] = 1;
        }
        for (int i = 0; i < R; i++) {
            ///find row 'r' with highest value at [r][i]
            int tr = i;
            for (int j = i + 1; j < R; j++)
                if (abs(res.mat[j][i]) > abs(res.mat[tr][i]))
                    tr = j;
            ///swap the row
            res.mat[tr].swap(res.mat[i]);
            ///make 1 at [i][i]
            T val = res.mat[i][i];
            for (int j = 0; j < R + R; j++) res.mat[i][j] /= val;
            ///eliminate [r][i] from every row except i.
            for (int j = 0; j < R; j++) {
                if (j == i) continue;
                for (int k = R + R - 1; k >= i; k--) {
                    res.mat[j][k] -= res.mat[i][k] * res.mat[j][i] / res.mat[i][i];
                }
            }
        }
        Matrix ans(R, R);
        for (int i = 0; i < R; i++) {
            for (int j = 0; j < R; j++)
                ans.mat[i][j] = res.mat[i][R + j];
        }
    }
}

```

7.9 Miller-Rabin-Pollard-Rho [68 lines]

```

ll powmod(ll a, ll p, ll m) { ////(a^p % m)
    ll result = 1;
    a %= m;
    while (p) {
        if (p & 1)
            result = (vll)result * a % m;
        a = (vll)a * a % m;
        p >>= 1;
    }
    return result;
}

```

```

    }
    bool check_composite(ll n, ll a, ll d, int s) {
        ll x = powmod(a, d, n);
        if (x == 1 || x == n - 1)
            return false;
        for (int r = 1; r < s; r++) {
            x = (vll)x * x % n;
            if (x == n - 1)
                return false;
        }
        return true;
    }
    bool MillerRabin(ll n) {
        if (n < 2) return false;
        int r = 0;
        ll d = n - 1;
        while ((d & 1) == 0) {
            d >>= 1;
            r++;
        }
        for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23,
                      29, 31, 37}) {
            if (n == a) return true;
            if (check_composite(n, a, d, r))
                return false;
        }
        return true;
    }
    ll mult(ll a, ll b, ll mod) {
        return (vll)a * b % mod;
    }
    ll f(ll x, ll c, ll mod) {
        return (mult(x, x, mod) + c) % mod;
    }
    ll rho(ll n) {
        if (n % 2 == 0) return 2;
        ll x = myrand() % n + 1, y = x, c = myrand() % n
            + 1, g = 1;
        while (g == 1) {
            x = f(x, c, n);
            y = f(y, c, n);
            y = f(y, c, n);
            g = __gcd(abs(x - y), n);
        }
        return g;
    }
    set<ll>prime;
    void prime_factorization(ll n) {
        if (n == 1) return;
        if (MillerRabin(n)) {
            prime.insert(n);
            return;
        }
        ll x = n;
        while (x == n) x = rho(n);
        prime_factorization(x);
        prime_factorization(n / x);
    }
}

```

//call prime_factorization(n) for prime factors.
//call MillerRabin(n) to check if prime.

7.10 Mod Inverse [5 lines]

```

int modInv(int a, int m) {
    int x, y; //if g==1 Inverse doesn't exist
    int g = gcdExt(a, m, x, y);
    return (x % m + m) % m;
}

```

7.11 NTT [96 lines]

```

ll power(ll a, ll p, ll mod) {
    if (p==0) return 1;
    ll ans = power(a, p/2, mod);
    ans = (ans * ans)%mod;
    if(p%2) ans = (ans * a)%mod;
    return ans;
}
int primitive_root(int p) {
    vector<int> factor;
    int phi = p-1, n = phi;
    for (int i=2; i*i<=n; i++) {
        if (n%i) continue;
        factor.push_back(i);
        while (n%i==0) n/=i;
    }
    if (n>1) factor.push_back(n);
    for (int res = 2; res<=p; res++) {
        bool ok = true;
        for (int i=0; i<factor.size() && ok; i++)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}
int nttdata(int mod, int &root, int &inv, int &pw)
{
    int c = 0, n = mod-1;
    while (n%2==0) c++, n/=2;
    pw = (mod-1)/n;
    int g = primitive_root(mod);
    root = power(g, n, mod);
    inv = power(root, mod-2, mod);
    return c;
}
const int M = 786433;
struct NTT {
    int N;
    vector<int> perm;
    int mod, root, inv, pw;
    NTT(){}
    NTT(int mod, int root, int inv, int pw) :
        mod(mod), root(root), inv(inv), pw(pw) {}
    void precalculate() {
        perm.resize(N);
        perm[0] = 0;
        for (int k=1; k<N; k<<=1) {

```

```

for (int i=0; i<k; i++) {
    perm[i] <= 1;
    perm[i+k] = 1 + perm[i];
}
}

void fft(vector<ll> &v, bool invert = false) {
    if (v.size() != perm.size()) {
        N = v.size();
        assert(N && (N&(N-1)) == 0);
        precalculate();
    }

    for (int i=0; i<N; i++)
        if (i < perm[i])
            swap(v[i], v[perm[i]]);

    for (int len = 2; len <= N; len <= 1) {
        ll factor = invert ? inv : root;
        for (int i=len; i<pw; i<=1)
            factor = (factor * factor) % mod;
        for (int i=0; i<N; i+=len) {
            ll w = 1;
            for (int j=0; j<len/2; j++) {
                ll x = v[i+j], y = (w*v[i+j+len/2])%mod;
                v[i+j] = (x+y)%mod;
                v[i+j+len/2] = (x-y+mod)%mod;
                w = (w*factor)%mod;
            }
        }

        if (invert) {
            ll n1 = power(N, mod-2, mod);
            for (ll &x: v) x = (x*n1)%mod;
        }
    }

    vector<ll> multiply(vector<ll> a, vector<ll> &b) {
        while (a.size() && a.back() == 0)
            a.pop_back();
        while (b.size() && b.back() == 0)
            b.pop_back();
        int n = 1;
        while (n < a.size() + b.size()) n<=1;
        a.resize(n);
        b.resize(n);
        fft(a);
        fft(b);
        for (int i=0; i<n; i++) a[i] = (a[i] * b[i])%M;
        fft(a, true);
        while (a.size() && a.back() == 0)
            a.pop_back();
        return a;
    }

    //      int mod=786433, root, inv, pw;
    //      nttdata(mod, root, inv, pw);
    //      NTT nn = NTT(mod, root, inv, pw);
}

```

7.12 No of Digits in $n!$ in base B [7 lines]

```

ll NoOfDigitInNFactInBaseB(ll N,ll B){
    ll i;
    double ans=0;
    for(i=1;i<=N;i++)ans+=log(i);
    ans=ans/log(B),ans=ans+1;
    return(ll)ans;
}

```

7.13 SOD Upto N [16 lines]

```

ll SOD_Upto_N(ll N){
    ll i,j,ans=0;///upto N in Sqrt(N)
    for(i=1;i*i<=N;i++){
        j=N/i;
        ans+=(j*(j+1))/2-(((i-1)*i)/2);
        ans+=(j-i)*i;
    }
    return ans;
}

ll SODUptoN(ll N){
    ll res=0,u=sqrt(N);
    for(ll i=1;i<=u;i++)
        res+=(N/i)-i;
    res*=2,res+=u;
    return res;
}

```

7.14 Sieve-Phi-Mobius [26 lines]

```

const int N = 1e7;
vector<int>pr;
int mu[N + 1], phi[N + 1], lp[N + 1];
void sieve() {
    phi[1] = 1, mu[1] = 1;
    for (int i = 2; i <= N; i++) {
        if (lp[i] == 0) {
            lp[i] = i;
            phi[i] = i - 1;
            pr.push_back(i);
        }
        for (int j = 0; j < pr.size() && i * pr[j]
        <= N; j++) {
            lp[i * pr[j]] = pr[j];
            if (i % pr[j] == 0) {
                phi[i * pr[j]] = phi[i] * pr[j];
                break;
            }
            else
                phi[i * pr[j]] = phi[i] *
                phi[pr[j]];
        }
    }
    for (int i = 2; i <= N; i++) {
        if (lp[i / lp[i]] == lp[i]) mu[i] = 0;
        else mu[i] = -1 * mu[i / lp[i]];
    }
}

```

8 Misc

8.1 Bit hacks [12 lines]

```

# x & -x is the least bit in x.
# iterate over all the subsets of the mask
for (int s=m; ; s=(s-1)&m) {
    ... you can use s ...
    if (s==0) break;
}

# c = x&-x, r = x+c; (((r^x) >> 2)/c) | r is the
# next number after x with the same number of bits
# set.
# __builtin_popcount(x) //number of ones in binary
# __builtin_popcountll(x) // for long long
# __builtin_clz(x) // number of leading zeros
# __builtin_ctz(x) // number of trailing zeros,
# they also have long long version

```

8.2 Bitset C++ [13 lines]

```

bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_first()<<endl; // prints 1
bs._Find_next(idx). This function returns first
set bit after index idx. for example:

```

```

bitset<17>BS;
BS[1] = BS[7] = 1;
cout<<BS._Find_next(1)<<' , '<<BS._Find_next(3) << endl; // prints
7,7

```

So this code will print all of the set bits of BS:

```

for(int i=BS._Find_first();i< BS.size();i =
BS._Find_next(i))
    cout<<i<<endl;
//Note that there isn't any set bit after idx,
BS._Find_next(idx) will return BS.size(); same
as calling BS._Find_first() when bitset is
clear;

```

8.3 Template [34 lines]

```

// #pragma GCC optimize("O3,unroll-loops")
// #pragma GCC
target("avx2,bmi,bmi2,lzcnt,popcnt")
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
using ll = long long;

template <typename A, typename B> ostream&
operator<<(ostream& os, const pair<A, B>& p) {
    return os << '(' << p.first << ", " <<
    p.second << ')';
}

```

```

template <typename T_container, typename T =
    typename enable_if<!is_same<T_container,
    string>::value, typename
    T_container::value_type>::type> ostream&
operator<<(ostream& os, const T_container& v)
{ os << '{'; string sep; for (const T& x : v)
os << sep << x, sep = ", "; return os << '}';
}

void dbg_out() { cerr << endl; }
template <typename Head, typename... Tail> void
dbg_out(Head H, Tail... T) { cerr << " " <<
H; dbg_out(T...); }

#ifndef SMIE
#define debug(args...) cerr << "(" << #args <<
"):", dbg_out(args)
#else
#define debug(args...)
#endif

template <typename T> inline T gcd(T a, T b) { T
c;while (b) { c = b;b = a % b;a = c; }return
a; } // better than __gcd

ll powmod(ll a, ll b, ll MOD) { ll res = 1;a %=
MOD;assert(b >= 0);for ( ; b; b >>= 1) { if (b
& 1)res = res * a % MOD;a = a * a % MOD;
}return res; }

template <typename T>using orderedSet = tree<T,
    null_type, less_equal<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
//order_of_key(k) - number of element strictly
less than k
//find_by_order(k) - k'th element in set.(0
    indexed)(iterator)

mt19937
rng(chrono::steady_clock::now().time_since_epoch()
    .count());
//uniform_int_distribution<int>(0, i)(rng)
int main(int argc, char* argv[]) {
    ios_base::sync_with_stdio(false); //DON'T CC++
    cin.tie(NULL); //DON'T use for interactive
    int seed = atoi(argv[1]);
}

8.4 Vimrc [26 lines]
filetype plugin indent on
syntax on
set hls is ar ai cul wrap lbr nu rnu et magic sc
    aw sb spr gd so=2 tm=400 mouse=a sw=4 sts=4
    ts=4 ls=2 bs=indent,eol,start
au TerminalOpen * setlocal nonu noru
au BufNewFile *.cpp -r ./template.cpp | 14
set tgc
let mapleader = " "
imap jk <esc>
imap <c-h> <left>
noremap <c-c> "+y

```

```

nnoremap <leader>f :let @+ = expand('%:p')<cr>
noremap <leader>d "_d
nnoremap <leader>ca :%y+<cr>
nnoremap <leader>cd :cd %:h<cr>
nnoremap <leader>b :ls<cr>:b
nnoremap <leader>vim :vs ~/.vimrc<cr>
nnoremap <leader>r %x<c-o>x

inoremap {<cr> {<cr>}<esc>0
inoremap ( <c-[>()<left>
inoremap (( (
vnoremap ( <esc>`>a)<esc>`<i(<esc>

inorea aLL <esc>yT(f.abegin(), <c-r>.end())
inorea raLL <esc>yT(f.arbegin(), <c-r>.rend())

8.5 build [3 lines]
#!/bin/bash
>&2 echo -e "Compiling $1.cpp with -std=gnu++17
$2" &&
g++ -std=gnu++17 -Wshadow -Wall -Wextra
-Wno-unused-result -O2 -g -fsanitize=undefined
-fsanitize=address $2 "$1.cpp" -o "$1"

8.6 debug [6 lines]
#!/bin/bash
bar="-----"
build $1 -DSMIE && >&2 echo -e "\tRunning
$1\n$bar" && ts=$(date +%s%N) &&
\time -f "$bar\nMemory Usage : %M KB" "./$1" &&
>&2 echo -e "Execution Time : $(((date +%s%N) -
$ts)/1000000)) ms"

8.7 stress-test [16 lines]
#!/bin/bash
build $1 $2 && build $1_gen $2 && build $1_brute
$2 &&
for((i = 1; ; ++i)); do
    echo -e "\nTest Case $i
    ./$1_gen $i > inp
    echo -e "===== \nINPUT\n-----"
    cat inp
    ./$1 < inp > out1
    echo -e "\nOUTPUT\n-----"
    cat out1
    ./$1_brute < inp > out2
    echo -e "\nEXPECTED\n-----"
    cat out2
    echo ""
    diff -w out1 out2 || break
done

```

9 String

9.1 Aho-Corasick [124 lines]

```

const int NODE=3000500; //Maximum Nodes
const int LGN=30; //Maximum Number of Tries
const int MXCHR=53; //Maximum Characters
const int MXP=5005; //

```

```

struct node {
    int val;
    int child[MXCHR];
    vector<int>graph;
    void clear(){
        CLR(child,0);
        val=0;
        graph.clear();
    }
}Trie[NODE+10];
int maxNodeId,fail[NODE+10],par[NODE+10];
int nodeSt[NODE+10],nodeEd[NODE+10];
vlong csum[NODE+10],pLoc[MXP];
void resetTrie(){
    maxNodeId=0;
}
int getNode(){
    int curNodeId=++maxNodeId;
    Trie[curNodeId].clear();
    return curNodeId;
}
inline void upd(vlong pos){
    csum[pos]++;
}
inline vlong qry(vlong pos){
    vlong res=csum[pos];
    return res;
}
struct AhoCorasick {
    int root,size,euler;
    void clear(){
        root=getNode();
        size=euler=0;
    }
    inline int getname(char ch){
        if(ch=='-')return 52;
        else if(ch>='A' && ch<='Z')return 26+(ch-'A');
        else return(ch-'a');
    }
    void addToTrie(string &s,int id){
        //Add string s to the Trie in general way
        int len=SZ(s),cur=root;
        FOR(i,0,len-1){
            int c=getname(s[i]);
            if(Trie[cur].child[c]==0){
                int curNodeId=getNode();
                Trie[curNodeId].val=c;
                Trie[cur].child[c]=curNodeId;
            }
            cur=Trie[cur].child[c];
        }
        pLoc[id]=cur;
        size++;
    }
    void calcFailFunction(){
        queue<int>Q;

```

```

Q.push(root);
while(!Q.empty()){
    int s=Q.front();
    Q.pop();
//Add all the children to the queue:
FOR(i,0,MXCHR-1){
    int t=Trie[s].child[i];
    if(t!=0){
        Q.push(t);
        par[t]=s;
    }
}
if(s==root){/*Handle special case when s is root*/
    fail[s]=par[s]=root;
    continue;
}
//Find fall back of s:
int p=par[s],f=fail[p];
int val=Trie[s].val;
/*Fall back till you found a node who has got val as a child*/
while(f!=root && Trie[f].child[val]==0){
    f=fail[f];
}
fail[s]=(Trie[f].child[val]==0)? root :
Trie[f].child[val];
//Self fall back not allowed
if(s==fail[s]){
    fail[s]=root;
}
Trie[fail[s]].graph.push_back(s);
}
void dfs(int pos){
++euler;
nodeSt[pos]=euler;
for(auto x: Trie[pos].graph){
    dfs(x);
}
nodeEd[pos]=euler;
}
//Returns the next state
int goTo(int state,int c){
if(Trie[state].child[c]!=0){/*No need to fall back*/
    return Trie[state].child[c];
}
//Fall back now:
int f=fail[state];
while(f!=root && Trie[f].child[c]==0){
    f=fail[f];
}
int res=(Trie[f].child[c]==0)?
root:Trie[f].child[c];
return res;
}

```

```

/*Iterate through the whole text and find all the matchings*/
void findmatching(string &s){
    int cur=root, idx=0;
    int len=SZ(s);
    while(idx<len){
        int c=getname(s[idx]);
        cur=goTo(cur,c);
        upd(nodeSt[cur]);
        idx++;
    }
}
}acorasick;


---



## 9.2 Double Hasing [50 lines]



---



```

struct SimpleHash {
 int len;
 long long base, mod;
 vector<int> P, H, R;
 SimpleHash() {}
 SimpleHash(const char* str, long long b, long long m) {
 base = b, mod = m, len = strlen(str);
 P.resize(len + 4, 1), H.resize(len + 3, 0),
 R.resize(len + 3, 0);
 for (int i = 1; i <= len + 3; i++)
 P[i] = (P[i - 1] * base) % mod;
 for (int i = 1; i <= len; i++)
 H[i] = (H[i - 1] * base + str[i - 1] + 1007) %
 mod;
 for (int i = len; i >= 1; i--)
 R[i] = (R[i + 1] * base + str[i - 1] + 1007) %
 mod;
 }
 inline int range_hash(int l, int r) {
 int hashval = H[r + 1] - ((long long)P[r - 1] +
 1) * H[1] % mod;
 return (hashval < 0 ? hashval + mod : hashval);
 }
 inline int reverse_hash(int l, int r) {
 int hashval = R[l + 1] - ((long long)P[r - 1] +
 1) * R[r + 2] % mod;
 return (hashval < 0 ? hashval + mod : hashval);
 }
};
struct DoubleHash {
 SimpleHash sh1, sh2;
 DoubleHash() {}
 DoubleHash(const char* str) {
 sh1 = SimpleHash(str, 1949313259, 2091573227);
 sh2 = SimpleHash(str, 1997293877, 2117566807);
 }
 long long concate(DoubleHash& B, int l1, int r1,
 int l2, int r2) {
 int len1 = r1 - l1 + 1, len2 = r2 - l2 + 1;
 long long x1 = sh1.range_hash(l1, r1) ,
 x2 = B.sh1.range_hash(l2, r2);

```


```

```

x1 = (x1 * B.sh1.P[len2]) % 2091573227;
long long newx1 = (x1 + x2) % 2091573227;
x1 = sh2.range_hash(l1, r1);
x2 = B.sh2.range_hash(l2, r2);
x1 = (x1 * B.sh2.P[len2]) % 2117566807;
long long newx2 = (x1 + x2) % 2117566807;
return (newx1 << 32) ^ newx2;
}
inline long long range_hash(int l, int r) {
    return ((long long)sh1.range_hash(l, r) << 32)
    ^ sh2.range_hash(l, r);
}
inline long long reverse_hash(int l, int r) {
    return ((long long)sh1.reverse_hash(l, r) <<
    32) ^ sh2.reverse_hash(l, r);
}
};


```

9.3 KMP [23 lines]

```

char P[maxn],T[maxn];
int b[maxn],n,m;
void kmpPreprocess(){
    int i=0,j=-1;
    b[0]=-1;
    while(i<m){
        while(j>=0 and P[i]!=P[j])
            j=b[j];
        i++;j++;
        b[i]=j;
    }
}
void kmpSearch(){
    int i=0,j=0;
    while(i<n){
        while(j>=0 and T[i]!=P[j])
            j=b[j];
        i++;j++;
        if(j==m){
            //pattern found at index i-j
        }
    }
}


```

9.4 Palindromic Tree [30 lines]

```

struct PalindromicTree{
    int n, idx, t;
    vector<vector<int>> tree;
    vector<int> len, link;
    string s; // 1-indexed
    PalindromicTree(string str){
        s="$"+str;
        n=s.size();
        len.assign(n+5,0);
        link.assign(n+5,0);
        tree.assign(n+5, vector<int>(26,0));
    }
    void extend(int p){
        while(s[p-len[t]-1]!=s[p]) t=link[t];
    }
}

```

```

int x=link[t],c=s[p]-'a';
while(s[p-len[x]-1]!=s[p]) x=link[x];
if(!tree[t][c]){
    tree[t][c]=++idx;
    len[idx]=len[t]+2;
    link[idx]=len[idx]==1?2:tree[x][c];
}
t=tree[t][c];
}
void build(){
    len[1]=-1,link[1]=1;
    len[2]=0,link[2]=1;
    idx=t=2;
    for(int i=1;i<n;i++) extend(i);
}
};

9.5 Suffix Array [78 lines]

```

```

struct SuffixArray {
vector<int> p, c, rank, lcp;
vector<vector<int>> st;
SuffixArray(string const& s) {
    build_suffix(s + char(1));
    p.erase(p.begin());
    build_rank(p.size());
    build_lcp(s);
    build_sparse_table(lcp.size());
}
void build_suffix(string const& s) {
    int n = s.size();
    const int MX_ASCII = 256;
    vector<int> cnt(max(MX_ASCII, n), 0);
    p.resize(n); c.resize(n);
    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i=1; i<MX_ASCII; i++) cnt[i]+=cnt[i-1];
    for (int i = 0; i < n; i++) p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]]) classes++;
        c[p[i]] = classes - 1;
    }
    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0) pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++) cnt[c[pn[i]]]++;
        for (int i=1; i<classes; i++)
            cnt[i]+=cnt[i-1];
        for (int i=n-1;i>=0;i--)
            p[--cnt[c[pn[i]]]]=pn[i];
        cn[p[0]] = 0; classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h)) % n]};
        }
    }
};

9.6 Suffix Automata [118 lines]

```

```

const int MXCHR = 26;
/*
take an object of suffixAutomata
call extend(c) for each character c in string
call Process() to initiate the important values
*/
struct suffixAutomata {
/*
len -> largest string length of the corresponding
endpos-equivalent class
link -> longest suffix that is another
endpos-equivalent class
*/
pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 << h)) % n]};
if (cur != prev) ++classes;
cn[p[i]] = classes - 1;
}
c.swap(cn);
}
void build_rank(int n) {
    rank.resize(n, 0);
    for (int i = 0; i < n; i++) rank[p[i]] = i;
}
void build_lcp(string const& s) {
    int n = s.size(), k = 0;
    lcp.resize(n - 1, 0);
    for (int i = 0; i < n; i++) {
        if (rank[i] == n - 1) {
            k = 0;
            continue;
        }
        int j = p[rank[i] + 1];
        while (i + k < n && j + k < n && s[i+k] == s[j+k])
            k++;
        lcp[rank[i]] = k;
        if (k) k--;
    }
}
void build_sparse_table(int n) {
    int lim = __lg(n);
    st.resize(lim + 1, vector<int>(n)); st[0] = lcp;
    for (int k = 1; k <= lim; k++)
        for (int i = 0; i + (1 << k) <= n; i++)
            st[k][i] = min(st[k - 1][i], st[k - 1][i + (1 << (k - 1))]);
}
int get_lcp(int i) { return lcp[i]; }
int get_lcp(int i, int j) {
    if (j < i) swap(i, j);
    j--; /*for lcp from i to j we don't need last
    lcp*/
    int K = __lg(j - i + 1);
    return min(st[K][i], st[K][j - (1 << K) + 1]);
}
};

9.6 Suffix Automata [118 lines]

```

firstpos -> end position of the first occurrence of the largest string of that node

```

*/
struct state {
    int link, len;
    int next[MXCHR];
    state() {}
    state(int l) {
        len = l;
        link = -1;
        for (int i = 0; i < MXCHR; i++) next[i] = -1;
    }
};
vector<state> node;
int sz, last;
vector<int> cnt, distinct, firstPos, occur, SA;
vector<vector<int>> adj; // suffix links tree
// cnt and SA for counting sort the nodes.
int L;

suffixAutomata() {
    node.push_back(state(0));
    firstPos.push_back(-1);
    occur.push_back(0);
    last = 0;
    sz = 0;
    L = 0;
}
int getID(char c) {
    return c - 'a'; // change according to problem
}
void extend(char c) {
    int idx = ++sz, p = last, id = getID(c);
    L++;
    node.push_back(state(node[last].len + 1));
    firstPos.push_back(node[idx].len - 1);
    occur.push_back(1);
}
while (p != -1 && node[p].next[id] == -1) {
    node[p].next[id] = idx;
    p = node[p].link;
}
if (p == -1)
    node[idx].link = 0;
else {
    int q = node[p].next[id];
    if (node[p].len + 1 == node[q].len)
        node[idx].link = q;
    else {
        int clone = ++sz;
        state x = node[q];
        x.len = node[p].len + 1;
        node.push_back(x);
        firstPos.push_back(firstPos[q]);
    }
}

```

```

        occur.push_back(0);
    while (p != -1 && node[p].next[id] == q) {
        node[p].next[id] = clone;
        p = node[p].link;
    }
    node[idx].link = node[q].link = clone;
}
last = idx;
}
void Process() {
    cnt.resize(sz + 1);
    distinct.resize(sz + 1);
    SA.resize(sz + 1);
    adj.resize(sz + 1);
    for (int i = 0; i <= sz; i++)
        cnt[node[i].len]++;
    for (int i = 1; i <= L; i++) cnt[i] += cnt[i - 1];
    for (int i = 0; i <= sz; i++)
        SA[--cnt[node[i].len]] = i;
    for (int i = sz; i > 0; i--) {
        int idx = SA[i];
        occur[node[idx].link] += occur[idx];
        adj[node[idx].link].push_back(idx);
        distinct[idx] = 1;
        for (int j = 0; j < MXCHR; j++) {
            if (node[idx].next[j] != -1)
                distinct[idx] += distinct[node[idx].next[j]];
        }
    } // counts distinct substrings and occurrence
    // of each state
    for (int i = 0; i < MXCHR; i++)
        if (node[0].next[i] != -1) distinct[0] +=
            distinct[node[0].next[i]];
}
pair<int, int> lcs(string &str) {
    int mxlen = 0, bestpos = -1, pos = 0, len = 0;
    int u = 0; // LCS of two string. returns
    start position and length
    for (char c : str) {
        int v = getID(c);
        while (u && node[u].next[v] == -1) {
            u = node[u].link;
            len = node[u].len;
        }
        if (node[u].next[v] != -1) {
            len++;
            u = node[u].next[v];
        }
        if (len > mxlen) {
            mxlen = len;
            bestpos = pos;
        }
        pos++;
    }
    return {bestpos - mxlen + 1, mxlen};
}

```

```

    }
    state &operator[](int index) { return
        node[index]; }
};


```

9.7 Trie [28 lines]

```

const int maxn=100005;
struct Trie{
    int next[27][maxn];
    int endmark[maxn],sz;
    bool created[maxn];
    void insertTrie(string& s){
        int v=0;
        for(int i=0;i<(int)s.size();i++){
            int c=s[i]-'a';
            if(!created[next[c][v]]){
                next[c][v]=++sz;
                created[sz]=true;
            }
            v=next[c][v];
        }
        endmark[v]++;
    }
    bool searchTrie(string& s){
        int v=0;
        for(int i=0;i<(int)s.size();i++){
            int c=s[i]-'a';
            if(!created[next[c][v]])
                return false;
            v=next[c][v];
        }
        return(endmark[v]>0);
    }
};


```

9.8 Z-Algorithm [19 lines]

```

void compute_z_function(const char*S,int N){
    int L=0,R=0;
    for(int i=1;i<N;++i){
        if(i>R){
            L=R=i;
            while(R<N && S[R-L]==S[R])++R;
            Z[i]=R-L,--R;
        }
        else{
            int k=i-L;
            if(Z[k]<R-i+1)Z[i]=Z[k];
            else{
                L=i;
                while(R<N && S[R-k]==S[R])++R;
                Z[i]=R-L,--R;
            }
        }
    }
}


```


<ul style="list-style-type: none"> Any three consecutive Fibonacci numbers are pairwise coprime, which means that, for every n, $\gcd(F_n, F_{n+1}) = \gcd(F_n, F_{n+2}) = \gcd(F_{n+1}, F_{n+2}) = 1$. If p is a prime, $\begin{cases} p = 5 & \Rightarrow p \mid F_p, \\ p \equiv \pm 1 \pmod{5} & \Rightarrow p \mid F_{p-1} \\ p \equiv \pm 2 \pmod{5} & \Rightarrow p \mid F_{p+1} \end{cases}$ The only nontrivial square Fibonacci number is 144. Attila Pethő proved in 2001 that there is only a finite number of perfect power Fibonacci numbers. In 2006, Y. Bugeaud, M. Mignotte, and S. Siksek proved that 8 and 144 are the only such non-trivial perfect powers. If the members of the Fibonacci sequence are taken mod n, the resulting sequence is periodic with period at most n. 	<pre>///n / x yields the same value for i <= x <= la. } return 0; }</pre>
<p>Sum of floors</p> $\sum_{i=1}^n \left\lfloor \frac{n}{i} \right\rfloor =$ <pre>int32_t main() { BeatMeScanf; int i,j,k,n,m; cin>>n; //complexity O(sqrt(n)) for (int i = 1, last; i <= n; i = last + 1) { last = n / (n / i); debug(i,last,n/i); } }</pre>	<p>Mobius Function and Inversion</p> <p>Notes</p> <ul style="list-style-type: none"> For any positive integer n, define $\mu(n)$ as the sum of the primitive nth roots of unity. It has values in $\{-1, 0, 1\}$ depending on the factorization of n into prime factors: <ul style="list-style-type: none"> $\mu(n) = 1$ if n is a square-free positive integer with an even number of prime factors. $\mu(n) = -1$ if n is a square-free positive integer with an odd number of prime factors. $\mu(n) = 0$ if n has a squared prime factor. Here, a root of unity, occasionally called a de Moivre number, is any complex number that gives 1 when raised to some positive integer power n. An nth root of unity, where n is a positive integer (i.e. $n = 1, 2, 3, \dots$), is a number z (maybe complex) satisfying the equation $z^n = 1$. An nth root of unity is said to be primitive if it is not a kth root of unity for some smaller k, that is if $z^n = 1 \text{ and } z^k \neq 1 \text{ for } k = 1, 2, 3, \dots, n-1.$

<ul style="list-style-type: none"> $\sum_{d n} (-1)^{n+d} \cdot d^3$ <p>Gauss Circle Theorem</p> <ul style="list-style-type: none"> The Gauss circle problem is the problem of determining how many integer lattice points there are in a circle centered at the origin and with radius r. Since the equation of this circle is given in Cartesian coordinates by $x^2 + y^2 = r^2$, the question is equivalently asking how many pairs of integers m and n there are such that $m^2 + n^2 < r^2$ If the answer for a given r is denoted by $N(r)$ then $N(r) = 1 + 4 \sum_{i=0}^{\infty} \left(\left\lfloor \frac{r^2}{4i+1} \right\rfloor - \left\lfloor \frac{r^2}{4i+3} \right\rfloor \right)$ A much simpler sum appears if the sum of squares function $r_2(n)$ is defined as the number of ways of writing the number n as the sum of two squares. Then $N(r) = \sum_{n=0}^{r^2} r_2(n).$	<ul style="list-style-type: none"> $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$ $\sum_{i=0}^k \binom{n}{i} = 2^n$ $\sum_{i=0}^k \binom{n}{2i} = 2^{n-1}$ $\sum_{i=0}^k \binom{n}{2i+1} = 2^{n-1}$ $\sum_{i=0}^k (-1)^i \binom{n}{i} = (-1)^k \binom{n-1}{k}$ $\sum_{i=0}^k \binom{n+i}{i} = \binom{n+k+1}{k}$ $\sum_{i=0}^k \binom{n}{n-i} = \binom{n+k+1}{k}$ $\sum_{i=0}^k \binom{i}{n} = \binom{k+1}{n+1}$ $\sum_{i=0}^k \binom{i}{k} = \binom{k+1}{k+1}$ $1 \cdot \binom{n}{1} + 2 \cdot \binom{n}{2} + 3 \cdot \binom{n}{3} + \dots + n \cdot \binom{n}{n} = n \cdot 2^{n-1}$ $1^2 \cdot \binom{n}{1} + 2^2 \cdot \binom{n}{2} + 3^2 \cdot \binom{n}{3} + \dots + n^2 \cdot \binom{n}{n} = (n+n^2) \cdot 2^{n-2}$ Vandermonde's Identity: $\sum_{k=0}^r \binom{m}{k} \binom{n}{r-k} = \binom{m+n}{r}$ Hockey-Stick: $n, r \in \mathbb{N}, n > r, \sum_{i=r}^n \binom{i}{r} = \binom{n+1}{r+1}$ <p>Identity:</p> $\sum_{i=0}^k \binom{k}{i}^2 = \binom{2k}{k}$ <ul style="list-style-type: none"> $\sum_{k=0}^n \binom{n}{k} \binom{n}{n-k} = \binom{2n}{n}$ $\sum_{k=0}^n \binom{n}{k} \binom{k}{q} = 2^{n-q} \binom{n}{q}$ $\sum_{i=0}^n 3^i \binom{n}{i} = 4^n$ $\sum_{i=0}^n \binom{2n}{i} = 2^{2n-1} + \frac{1}{2} \binom{2n}{n}$ $\sum_{i=1}^n \binom{n}{i} \binom{n-1}{i-1} = \binom{2n-1}{n-1}$
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<p>3. Combinatorics</p> <p>Notes</p> <ul style="list-style-type: none"> $\sum_{0 \leq k \leq n} \binom{n-k}{k} = \text{Fib}_{n+1}$ $\binom{n}{k} = \binom{n}{n-k}$ $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$ $k \binom{n}{k} = n \binom{n-1}{k-1}$ 	$\text{Identity: } \sum_{i=0}^k \binom{k}{i}^2 = \binom{2k}{k}$ <ul style="list-style-type: none"> $\sum_{k=0}^n \binom{n}{k} \binom{n}{n-k} = \binom{2n}{n}$ $\sum_{k=0}^n \binom{n}{k} \binom{k}{q} = 2^{n-q} \binom{n}{q}$ $\sum_{i=0}^n 3^i \binom{n}{i} = 4^n$ $\sum_{i=0}^n \binom{2n}{i} = 2^{2n-1} + \frac{1}{2} \binom{2n}{n}$ $\sum_{i=1}^n \binom{n}{i} \binom{n-1}{i-1} = \binom{2n-1}{n-1}$
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

- $\sum_{k=0}^n \binom{2n}{k} = \frac{1}{2} \left(\binom{4n}{2n} + \binom{2n}{n}^2 \right)$
- An integer $n \geq 2$ is prime if and only if all the intermediate binomial coefficients $\binom{n}{1}, \binom{n}{2}, \dots, \binom{n}{n-1}$ are divisible by n.
- $\binom{n+k}{k}$ divides $\frac{\text{lcm}(n,n+1,\dots,n+k)}{n}$
- Kummer's theorem states that for given integers $n \geq m \geq 0$ and a prime number p, the largest power of p dividing $\binom{n}{m}$ is equal to the number of carries when m is added to $n - m$ in base p.
- Number of different binary sequences of length n such that no two 0's are adjacent = Fib_{n+1}
- Combination with repetition: Let's say we choose k elements from an n-element set, the order doesn't matter and each element can be chosen more than once. In that case, the number of different combinations is: $\binom{n+k-1}{k}$
- Number of ways to divide n different persons in n/k equal groups i.e. each having size k is $\binom{n-1}{k-1}$
- The number non-negative solution of the equation $x_1+x_2+x_3+\dots+x_k=n$ is $\binom{n+k-1}{n}$
- Number of binary sequence of length n and with k '1' is $\binom{n}{k}$
- The number of ordered pairs (a, b) of binary sequences of length n, such that the distance between them is k, can be

calculated as follows: $\binom{n}{k} \cdot 2^n$.
The distance between a and b is the number of components that differs in a and b — for example, the distance between (0, 0, 1, 0) and (1, 0, 1, 1) is 2).

Catalan numbers

- $C_n = \frac{1}{n+1} \binom{2n}{n}$
- $C_0 = 1, C_1 = 1$ and $C_n = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786
- Number of correct bracket sequence consisting of n opening and n closing brackets.
- The number of ways to completely parenthesize $n+1$ factors.
- The number of triangulations of a convex polygon with $+2$ sides (i.e. the number of partitions of polygon into disjoint triangles by using the diagonals).
- The number of ways to connect the $2n$ points on a circle to form n disjoint i.e. non-intersecting chords.
- The number of monotonic lattice paths from point (0,0) to point (n,n) in a square lattice of size $n \times n$, which do not pass above the main diagonal (i.e. connecting (0,0) to (n,n)).
- Number of permutations of length n that can be stack sorted (i.e. it can be shown that the

- rearrangement is stack sorted if and only if there is no index $i < j < k$, such that $a_i < a_j < a_k$.
- The number of non-crossing partitions of a set of n elements.
 - The number of rooted full binary trees with $n+1$ leaves (vertices are not numbered). A rooted binary tree is full if every vertex has either two children or no children.
 - The number of Dyck words of length $2n$. A Dyck word is a string consisting of n X's and n Y's such that no initial segment of the string has more Y's than X's. For example, the following are the Dyck words of length 6: XXXYYY XYXXYY XYXYXY XXYYXY XXXYYY.
 - The number of different ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines (a form of Polygon triangulation)
 - Number of permutations of $\{1, \dots, n\}$ that avoid the pattern 123 (or any of the other patterns of length 3); that is, the number of permutations with no three-term increasing subsequence. For $n = 3$, these permutations are 132, 213, 231, 312 and 321. For $n = 4$, they are 1432, 2143, 2413, 2431, 3142, 3214, 3241, 3421, 4132, 4213, 4231, 4312 and 4321
 - Number of ways to tile a staircase shape of height n with n rectangles.

- $N(n,k) = \frac{1}{n} \binom{n}{k} \binom{n}{k-1}$
- The number of expressions containing n pairs of parentheses, which are correctly matched and which contain k distinct nestings. For instance, $N(4, 2) = 6$ as with four pairs of parentheses six sequences can be created which each contain two times the sub-pattern '|':

$(\text{()}) (\text{()}) (\text{()}) (\text{()}) (\text{()}) (\text{()}) (\text{()}) (\text{()})$

- The number of paths from (0, 0) to $(2n, 0)$, with steps only northeast and southeast, not straying below the x-axis, with k peaks. And sum of all number of peaks is Catalan number.

Stirling numbers of the first kind

- The Stirling numbers of the first kind count permutations according to their number of cycles (counting fixed points as cycles of length one).
- $S(n,k)$ counts the number of permutations of n elements with k disjoint cycles.
- $S(n,k) = (n-1) * S(n-1,k) + S(n-1,k-1)$, where $S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $\sum_{k=0}^n S(n,k) = n!$

Stirling numbers of the second kind

- Stirling number of the second kind is the number of ways to partition a set of n objects into k non-empty subsets.
- $S(n,k) = k * S(n-1,k) + S(n-1,k-1)$, where $S(0,0) = 1, S(n,0) = S(0,n) = 0$
- $S(n,2) = 2^{n-1}$
- $S(n,k) * k! =$ number of ways to color n nodes using colors from 1 to k such that each color is used at least once.

Bell number

- Counts the number of partitions of a set.
- $B_{n+1} = \sum_{k=0}^n \binom{n}{k} * B_k$
- $B_n = \sum_{k=0}^n S(n,k)$, where $S(n,k)$ is stirling number of second kind.
- The number of multiplicative partitions of a squarefree number with i prime factors is the i -th Bell number, B_i .
- If a deck of n cards is shuffled by repeatedly removing the top card and reinserting it anywhere in the deck (including its original position at the top of the deck), with exactly n^0 repetitions of this operation, then there are n^n different shuffles that can be performed. Of these, the number that return the deck to its original sorted order is exactly B_n . Thus, the probability that the deck is in its original order after shuffling it in this way is B_n/n^n .

Lucas Theorem

- If p is prime then $\binom{p^k}{k} \equiv 0 \pmod{p}$
 - For non-negative integers m and n and a prime p , the following congruence relation holds:
- $$\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p},$$
- where,
 $m = m_k p^k + m_{k-1} p^{k-1} + \dots + m_1 p + m_0$, and
 $n = n_k p^k + n_{k-1} p^{k-1} + \dots + n_1 p + n_0$
- are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$, when $m < n$.

Derangement

- A derangement is a permutation of the elements of a set, such that no element appears in its original position.
- $d(n) = (n-1) * (d(n-1) + d(n-2))$, where $d(0) = 1, d(1) = 0$
- $d(n) = \lfloor \frac{n!}{e} \rfloor, n \geq 1$

4. Burnside Lemma

The task is to count the number of different necklaces from n beads, each of which can be painted in one of the k colors. When comparing two necklaces, they can be rotated, but not reversed (i.e. a cyclic shift is permitted).

Solution: