­

1. Kata – Kata Bijak
2. Fast Stuffs : BIT Range, Fast Read, Fast Mod
3. Dynamic Convex-Hull trick
4. AVL
5. KD Tree
6. Gomory-Hu Tree
7. Dinic Maxflow
8. Maximum Bipartite Matching
9. Minimum Cost Maximum Flow
10. Hopcroft karp
11. Stable Marriage
12. 2-SAT
13. Flow-shop Scheduling (Johnson’s Problem)
14. Konig's Theorem
15. Sprague Grundy Theorem
16. Hungarian
17. Edmond’s Blossom
18. Eulerian Path/Cycle
19. Online Bridge
20. FFT
21. Smallest String rotation
22. Z-Algorithm
23. KMP
24. Manacher’s Algorithm (palindrome count)
25. Suffix Array + LCP
26. Biconnected Component
27. Aho Corasick
28. Tarjan's SCC
29. Tarjan’s Articulation Point and Bridge
30. Lowest Common Ancestor
31. Josephus problem
32. Lucas Theorem
33. Extended Euclid
34. Pollard-Rho + Miller Rabin
35. Brent’s Cycle Finding
36. Erdos-Gallai
37. Gaussian Elimination
38. Matrix Operations
39. Johnson’s Algorithm
40. Geometry
41. Rotating Calipher
42. Great Circle Distance
43. Combinatorics: sifat2 gitu biar jago
44. Next Permutation
45. DP Optimization
46. Prime Numbers :’( + Mobius
47. Chinese Remainder Theorem
48. Tiles Solvability
49. Formulas + Theorems
50. Heavy Light Decomposition
51. PDBS Template
52. Kata – Kata Bijak

* Jangan takut *gambling* – *hash* dan *random* adalah teman
* Soal *Math* – Ingat kata guru SMA, cari pola, atau bikin rumus *magic*
* Soal *String –* Semoga langgeng (?)
* Soal Geometri – Minta doa ibu
* Jangan biarkan trauma menghantuimu :’ (ICPC Jakarta K)
* Pola bisa konvergen setelah beberapa lama
* Pastikan gak ada yang salah ngerti soal
* Pusing? Ngantuk? Gak ada ide? Sana ke WC
* Kalau N <= 100 ada kemungkinan max-flow
* Pastiin sketsa solusi udah bener sebelum ngoding :’ (ICPC Jakarta C)
* Kalo nganggur, udah nekat aja
* Pastikan gak ada yang telat
* OP TI MIS dan SE MA NGAT

1. Fast Stuffs : BIT Range, Fast Read, Fast Mod

ll bit[2][N];

void update(int a, int b, ll val) {

update(bit[0], a, val);

update(bit[0], b + 1, -val);

update(bit[1], a, val \* (a - 1));

update(bit[1], b + 1, -val \* b);

}

ll query(int x) {

return query(bit[0], x) \* x - query(bit[1], x);

}

Barret 61: inline int rem(long long a){return a-mod\*((a>>29)\*MM>>32);}

inline void fastRead\_int(int &x) {

register int c = getchar\_unlocked();

x = 0;

for(; ((c<48 || c>57) && c != '-'); c = getchar\_unlocked());

for(; c>47 && c<58 ; c = getchar\_unlocked()) {

x = (x<<1) + (x<<3) + c - 48;

}

}

ios::sync\_with \_stdio(false); cin.tie(NULL);

clock\_t first\_attempt = clock();

inline void cek\_time(){

clock\_t cur = clock()- first\_attempt;

cerr<<"TIME : "<<(double) cur/CLOCKS\_PER\_SEC<<endl;

}

1. Dynamic Convex-Hull trick

const ll is\_query = -(1LL<<62);

struct Line {

ll m, b;

mutable function<const Line\*()> succ;

bool operator<(const Line& rhs) const {

if (rhs.b != is\_query) return m < rhs.m;

const Line\* s = succ();

if (!s) return 0;

ll x = rhs.m;

return b - s->b < (s->m - m) \* x;

}

};

struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum

bool bad(iterator y) {

auto z = next(y);

if (y == begin()) {

if (z == end()) return 0;

return y->m == z->m && y->b <= z->b;

}

auto x = prev(y);

if (z == end()) return y->m == x->m && y->b <= x->b;

return (x->b - y->b)\*(z->m - y->m) >= (y->b - z->b)\*(y->m - x->m);// beware overflow!

}

void insert\_line(ll m, ll b) {

auto y = insert({ m, b });

y->succ = [=] { return next(y) == end() ? 0 : &\*next(y); };

if (bad(y)) { erase(y); return; }

while (next(y) != end() && bad(next(y))) erase(next(y));

while (y != begin() && bad(prev(y))) erase(prev(y));

}

ll eval(ll x) {

auto l = \*lower\_bound((Line) { x, is\_query });

return l.m \* x + l.b;

}

};

// minimum : change m,c to negative. Then, the result of the query change to -result.

1. AVL

struct node{

int key, height, size;

struct node\* left;

struct node\* right;

};

inline int getH(struct node \*cur){

if (cur == NULL) return 0;

return cur -> height;

}

inline int getB(struct node\* cur) {

if (cur == NULL) return 0;

return getH(cur->left) - getH(cur->right);

}

inline int getS(struct node\* cur) {

if (cur == NULL) return 0;

return cur -> size;

}

inline void updateH(struct node\* cur) {

cur->height = max(getH(cur->left), getH(cur->right))+1;

}

inline void updateS(struct node\* cur) {

cur->size = 1 + getS(cur->left) + getS(cur->right);

}

struct node\* newnode(int key) {

struct node\* cur = new node();

cur -> key = key;

cur -> left = NULL;

cur -> right = NULL;

cur -> height = 1;

cur -> size = 1;

return cur;

}

struct node\* RotateRight(struct node\* y) {

struct node\* x = y->left;

struct node\* xR = x->right;

x->right = y;

y->left = xR;

updateH(y); updateH(x);

updateS(y); updateS(x);

return x;

}

struct node\* RotateLeft(struct node\* y) {

struct node\* x = y->right;

struct node\* xL = x->left;

x->left = y;

y->right = xL;

updateH(y); updateH(x);

updateS(y); updateS(x);

return x;

}

struct node\* minval(struct node\* cur) {

struct node\* proc = cur;

while(proc->left != NULL) proc=proc->left;

return proc;

}

struct node\* insert(struct node\* cur, int key) {

if (cur == NULL) return newnode(key);

if (key == cur -> key) return cur;

if (key < cur -> key) cur -> left = insert(cur->left, key);

else cur -> right = insert(cur->right, key);

updateH(cur); updateS(cur);

int balance = getB(cur);

if (balance > 1 && getB(cur->left) >= 0) return RotateRight(cur);

if (balance > 1 && getB(cur->left) < 0) {cur->left = RotateLeft(cur->left); return RotateRight(cur);}

if (balance <-1 && getB(cur->right)<= 0) return RotateLeft(cur);

if (balance <-1 && getB(cur->right) > 0) {cur->right = RotateRight(cur->right); return RotateLeft(cur);}

return cur;

}

struct node\* Delete(struct node\* cur, int key) {

if (cur == NULL) return cur;

if (key < cur -> key) cur->left = Delete(cur->left, key);

else if (key > cur -> key) cur->right = Delete(cur->right, key);

else {

if ((cur->left==NULL)||(cur->right==NULL)) {

struct node\* temp = cur -> left? cur->left : cur -> right;

if (temp == NULL) {

temp = cur;

cur = NULL;

} else \*cur = \*temp;

free(temp);

} else {

struct node\* temp = minval(cur->right);

cur -> key = temp -> key;

cur -> right = Delete(cur->right, temp->key);

}

}

if (cur == NULL) return cur;

updateH(cur); updateS(cur);

int balance = getB(cur);

if (balance > 1 && getB(cur->left) >= 0) return RotateRight(cur);

if (balance > 1 && getB(cur->left) < 0) {cur->left = RotateLeft(cur->left); return RotateRight(cur);}

if (balance <-1 && getB(cur->right) <= 0) return RotateLeft(cur);

if (balance <-1 && getB(cur->right) > 0) {cur->right = RotateRight(cur->right); return RotateLeft(cur);}

return cur;

}

1. KD Tree

#include <bits/stdc++.h>

using namespace std;

const int D = 2;

const int N = 1<<17;

const long long INF = 2000000000000000007;

struct Point {

int p[D];

bool operator !=(const Point &a) const {

for (int i = 0; i < D; i++)

if (p[i] != a.p[i])

return true;

return false;

}

};

long long distance(Point a, Point b) {

long long ans=0;

for(int i=0;i<D;i++) ans+=(a.p[i]-b.p[i])\*1ll\*(a.p[i]-b.p[i]);

return ans;

}

// Usage: insert all Point first, then call build()

struct NearestNeighbor {

struct kd\_node {

int axis, value;

Point p;

kd\_node \*left, \*right;

};

struct cmp\_Points {

int axis;

cmp\_Points(){}

cmp\_Points(int x): axis(x) {}

bool operator () (const Point &a, const Point &b) const {

return a.p[axis]<b.p[axis];

}

};

typedef kd\_node\* node\_ptr;

int n;

vector<Point> arr;

node\_ptr root;

void build\_tree(node\_ptr &node, int from, int to, int axis) {

if(from>to) {

node=NULL;

return;

}

node=new kd\_node();

if(from==to) {

node->p=arr[from];

node->left=NULL;

node->right=NULL;

return;

}

int mid=(from+to)/2;

nth\_element(arr.begin()+from,arr.begin()+mid,arr.begin()+to+1,cmp\_Points(axis));

node->value=arr[mid].p[axis];

node->axis=axis;

int naxis = (axis == D - 1) ? 0 : axis + 1;

build\_tree(node->left,from,mid,naxis);

build\_tree(node->right,mid+1,to,naxis);

}

void nearest\_neighbor(node\_ptr node, Point q, Point &ans) {

#define dist distance(ans, q)

if(node==NULL) return;

if(node->left==NULL && node->right==NULL) {

if(q!=node->p) {

if (distance(ans, q) > distance(node->p,q)){

ans = node->p;

}

}

return;

}

if(q.p[node->axis]<=node->value) {

nearest\_neighbor(node->left, q, ans);

if(q.p[node->axis]+sqrt(dist)>=node->value)

nearest\_neighbor(node->right, q, ans);

}

else {

nearest\_neighbor(node->right, q, ans);

if(q.p[node->axis]-sqrt(dist)<=node->value)

nearest\_neighbor(node->left, q, ans);

}

#undef dist

}

void insert(Point p) {

arr.push\_back(p);

}

void build() {

n = arr.size();

build\_tree(root,0,n - 1,0);

}

void clear() {

arr.clear();

}

Point nearest\_neighbor(Point x) {

Point p = x;

p.p[0] += 1e9;

nearest\_neighbor(root, x, p);

return p;

}

} KD;

int tests, n;

Point arr[N];

int main() {

int i;

scanf("%d", &tests);

while(tests--) {

scanf("%d", &n);

KD.clear();

for(i=0;i<n;i++) {

scanf("%d %d", &arr[i].p[0], &arr[i].p[1]);

KD.insert(arr[i]);

}

KD.build();

for(i=0;i<n;i++) {

Point p = KD.nearest\_neighbor(arr[i]);

long long ans = distance(p, arr[i]);

printf("%lld\n", ans);

}

}

return 0;

}

1. Bellman Ford

struct edge{ int to, w; };

vector<edge> adjList[NMAX+5];

int dist[NMAX+5];

int calc(int s, int t){

for (int i = 0; i <= n; i++) dist[i] = INF;

dist[s] = 0;

edge tmp;

for (int i = 0; i < n - 1; i++){

for (int j = 0; j < n; j++){

for (edge k: adjList[j]) {

tmp = k;

dist[tmp.to] = min(dist[tmp.to], dist[j] + tmp.w);

} } }

// any edge from u to v with weight w

// if dist[u] + w < dist[v] => negative cycle

return dist[t];

}

1. Dinic Maxflow

struct edge{

int to, rev;

int flow, cap;

};

vector<edge> G[MAXE];

inline void add(int s, int t, int cap) {

edge a = {t, G[t].size(), 0, cap};

edge b = {s, G[s].size(), 0, 0};

G[s].push\_back(a);

G[t].push\_back(b);

}

inline bool search() {

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

dist[source] = 0;

int tail = 0;

q[tail] = source;

for(int head = 0; head <= tail; head++) {

int u = q[head];

int sz = G[u].size();

for(int i = 0; i < sz; i++) {

int v = G[u][i].to;

if (dist[v] < 0 && G[u][i].flow < G[u][i].cap) {

dist[v] = dist[u] + 1;

q[++tail] = v;

}

}

}

return dist[sink] >= 0;

}

int dinic(int now, int flo) {

if (now == sink)

return flo;

int size = G[now].size();

for(int &i = work[now]; i < size; i++) {

int to = G[now][i].to;

int flow = G[now][i].flow;

int cap = G[now][i].cap;

int rev = G[now][i].rev;

if (flow >= cap) continue;

if (dist[to] == dist[now] + 1) {

int fflow = dinic(to, min(flo, cap - flow));

if (fflow) {

G[now][i].flow += fflow;

G[to][rev].flow -= fflow;

return fflow;

}

}

}

return 0;

}

inline int maxflow() {

int ans = 0;

while(search()) {

for(int i = 0; i <= n + 1; i++) work[i] = 0;

while(true) {

int res = dinic(source, INF);

if (res == 0) break;

ans += res;

}

}

return ans;

}

1. Maximum Bipartite Matching

// The code below finds a augmenting path:

bool dfs(int v){// v is in X, it returns true if and only if there is an augmenting path starting from v

if(mark[v])

return false;

mark[v] = true;

for(auto &u : adj[v])

if(match[u] == -1 or dfs(match[u])) // match[i] = the vertex i is matched with in the current matching, initially -1

return matched[v] = u, match[u] = v, true;

return false;

}

An easy way to solve the problem is:

for(int i = 0;i < n;i ++)if(matched[i] == -1){

memset(mark, false, sizeof mark);

dfs(i);

}

But there is a faster way:

while(true){

memset(mark, false, sizeof mark);

bool fnd = false;

for(int i = 0;i < n;i ++) if(matched[i] == -1 && !mark[i])

fnd |= dfs(i);

if(!fnd)

break;

}

1. Minimum Cost Maximum Flow (Negative cost, not negative cycle)

struct edge{

int to, rev;

int flow, cap;

int cost;

};

vector<edge> G[500];

inline void add(int s, int t, int capa, int costs) {

edge a = {t, G[t].size(), 0, capa, costs};

edge b = {s, G[s].size(), 0, 0, -costs};

G[s].push\_back(a);

G[t].push\_back(b);

}

inline bool SPFA() {

for(int i = 0; i <= sink; i++) dist[i] = INF, flag[i] = false, bt[i] = -1, idx[i] = -1;

dist[source] = 0;

queue<int> q;

q.push(source);

flag[source] = true;

while(!q.empty()) {

int now = q.front();

q.pop();

flag[now] = false;

int size = G[now].size();

for(int i = 0; i < size; i++) {

int to = G[now][i].to;

int cost = G[now][i].cost;

int capa = G[now][i].cap;

if (capa > 0 && dist[to] > dist[now] + cost) {

dist[to] = dist[now] + cost;

bt[to] = now;

idx[to] = i;

if (!flag[to]) {

flag[to] = true;

q.push(to);

}

}

}

}

return bt[sink] != -1;

}

pair<int,int> MCMF() {

pair<int,int> res; res.first = 0, res.second = 0;

while(true) {

if (!SPFA()) break;

int mins = INF;

int ptr = sink;

int total = 0;

while(ptr != source) {

int from = bt[ptr];

int id = idx[ptr];

if (G[from][id].cap < mins)

mins = G[from][id].cap;

total += G[from][id].cost;

ptr = from;

}

res.first += mins;

res.second += total \* mins;

ptr = sink;

while(ptr != source) {

int from = bt[ptr];

int id = idx[ptr];

int rev = G[from][id].rev;

G[from][id].cap -= mins;

G[ptr][rev].cap += mins;

ptr = from;

}

}

return res;

}

// \*\* MINIMUM CUT STOER \*\*/

#include <cmath>

#include <vector>

#include <iostream>

using namespace std;

typedef vector<int> VI;

typedef vector<VI> VVI;

const int INF =1000000000;

pair<int, VI> GetMinCut(VVI &weights){

int N = weights.size();

VI used(N), cut, best\_cut;

int best\_weight =-1;

for(int phase = N-1; phase >=0; phase--){

VI w = weights[0];

VI added = used;

int prev, last =0;

for(int i =0; i < phase; i++){

prev = last;

last =-1;

for(int j =1; j < N; j++)

if(!added[j]&&(last ==-1|| w[j]> w[last])) last = j;

if(i == phase-1){

for(int j =0; j < N; j++) weights[prev][j]+= weights[last][j];

for(int j =0; j < N; j++) weights[j][prev]= weights[prev][j];

used[last]=true;

cut.push\_back(last);

if(best\_weight ==-1|| w[last]< best\_weight){

best\_cut = cut;

best\_weight = w[last];

}

}else{

for(int j =0; j < N; j++)

w[j]+= weights[last][j];

added[last]=true;

}

}

}

return make\_pair(best\_weight, best\_cut);

}

1. Hopcroft karp

// node from 1..n

// 0 is NIL

// left side 1..n

// right side n+1..n+m

// G = {0} U {1..n} U {n+1..n+m}

bool bfs() {

queue<int> q;

for(int i = 1 ; i <= n ; i++)

if(match[i] == 0) {

dist[i] = 0;

q.push(i);

}

else

dist[i] = INF;

dist[0] = INF;

while(!q.empty()) {

int cur = q.front();

q.pop();

if(cur) {

for(int nex : adj[cur]) {

if(dist[match[nex]] == INF) {

dist[match[nex]] = dist[cur] + 1;

q.push(match[nex]);

}

}

}

}

return dist[0] != INF;

}

int dfs(int now) {

if(now == 0) return 1; // found 1 augmenting path

for(int nex : adj[now]) {

if(dist[match[nex]] == dist[now] + 1 && dfs(match[nex])) {

match[nex] = now;

match[now] = nex;

return 1;

}

}

dist[now] = INF;

return 0;

}

int hopcroftKarp() {

int ret = 0;  
 memset(match, 0, sizeof match);

while(bfs()) {

for(int i = 1 ; i <= n ; i++)

if(match[i] == 0)

ret += dfs(i);

}

return ret;

}

1. Stable Marriage

While there is a free man m: let w be the most preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

1. 2-SAT

Build an implication graph with 2 vertices for each variable (the variable itself and its inverse). For each clause x V y, add edges (x', y) and (y', x). The formula is satisfiable iff x and x' are in different SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (Kosaraju's last step). Assign true to all variables of the current SCC (if it hasn't been previously assigned false), and false to all inverses.

1. Flow-shop Scheduling

Schedule N jobs on 2 machines to minimize completion time. i-th job takes ai and bi time to execute on 1st and 2nd machine, respectively. Each job must be first executed on the first machine, then on second. Both machines execute all jobs in the same order. Solution: sort jobs by key ai < bi ? ai : (oo-bi), i.e. first execute all jobs with ai < bi in order of increasing ai, then all other jobs in order of decreasing bi.

1. Konig's Theorem

Consider a bipartite graph where the vertices are partitioned into left (L) and right (R) sets. Suppose there is a maximum matching which partitions the edges into those used in the matching (Em) and those not (E0). Let T consist of all unmatched vertices from L, as well as all vertices reachable from those (starting from vertices of T ) by going left-to-right along edges from E0 and right-to-left along edges from Em . This essentially means that for each unmatched vertex in L, we add into T all vertices that occur in a path alternating between edges from E0 and Em. Then is a minimum vertex cover. Intuitively, vertices in T are added if they are in R and subtracted if they are in L to obtain the minimum vertex cover.

1. Sprague Grundy Theorem

Two positions G and H can be added to make a new position G+H in a combined game where the current player can choose either to move in G or in H. Explicit computation of the set G+H is by repeated application of the rule G+H={G+h | h ϵ H} U {g+H | g ϵ G}, which incidentally indicates that position addition is both commutative and associative as expected.

+ operator here are defined as nim sum (xor)

1. Hungarian

int a[MAXN + 5][MAXN + 5],u[MAXN + 5],v[MAXN + 5];

bool used[MAXN + 5];

int p[MAXN + 5],way[MAXN + 5];

int minv[MAXN + 5];

int ans[MAXN + 5];

int Assign(){

for(int i = 1 ; i <= k ; i++){

p[0] = i;

int j0 = 0;

for(int j = 1 ; j <= k ; j++)

minv[j] = 1000000000,used[j] = 0;

do{

used[j0] = 1;

int i0 = p[j0],delta = 1000000000, j1;

for(int j = 1 ; j <= k ; j++)

if(!used[j]){

int cur = val[i0][j] - u[i0] - v[j];

if(cur < minv[j])

minv[j] = cur,way[j] = j0;

if(minv[j] < delta)

delta = minv[j], j1 = j;

}

for(int j = 0 ; j <= k ; j++)

if(used[j])

u[p[j]] += delta, v[j] -= delta;

else

minv[j] -= delta;

j0 = j1;

}while(p[j0] != 0);

do{

int j1 = way[j0];

p[j0] = p[j1];

j0 = j1;

}while(j0);

}

for(int i = 1 ; i <= k ; i++)

ans[p[i]] = i;

int ret = 0;

for(int i = 1 ; i <= k ; i++)

ret += val[i][ans[i]];// person i is matched with job ans[i]

return ret;

}

1. Edmond’s Blossom

/\* Blossom | O(V^3) \*/

int lca(vector<int>&match, vector<int>&base, vector<int>&p,int a,int b){

vector<bool> used(SZ(match));

while(true){

a = base[a];

used[a]=true;

if(match[a]==-1)break;

a = p[match[a]];

}

while(true){

b = base[b];

if(used[b])return b;

b = p[match[b]];

}

return-1;

}

void markPath(vector<int>&match, vector<int>&base, vector<bool>&blossom, vector<int>&p,int v,int b,int children){

for(; base[v]!= b; v = p[match[v]]){

blossom[base[v]]= blossom[base[match[v]]]=true;

p[v]= children;

children = match[v];

}

}

int findPath(vector<vector<int>>&graph, vector<int>&match, vector<int>&p,int root){

int n = SZ(graph);

vector<bool> used(n);

FORIT(it, p)\*it =-1;

vector<int> base(n);

for(int i =0; i < n;++i) base[i]= i;

used[root]=true;

int qh =0;

int qt =0;

vector<int> q(n);

q[qt++]= root;

while(qh < qt){

int v = q[qh++];

FORIT(it, graph[v]){

int to =\*it;

if(base[v]== base[to]|| match[v]== to)continue;

if(to == root || match[to]!=-1&& p[match[to]]!=-1){

int curbase = lca(match, base, p, v, to);

vector<bool> blossom(n);

markPath(match, base, blossom, p, v, curbase, to);

markPath(match, base, blossom, p, to, curbase, v);

for(int i =0; i < n;++i){

if(blossom[base[i]]){

base[i]= curbase;

if(!used[i]){

used[i]=true;

q[qt++]= i;

}

}

}

}elseif(p[to]==-1){

p[to]= v;

if(match[to]==-1)return to;

to = match[to];

used[to]=true;

q[qt++]= to;

}

}

}

return-1;

}

int maxMatching(vector<vector<int>> graph){

int n = SZ(graph);

vector<int> match(n,-1);

vector<int> p(n);

for(int i =0; i < n;++i){

if(match[i]==-1){

int v = findPath(graph, match, p, i);

while(v !=-1){

int pv = p[v];

int ppv = match[pv];

match[v]= pv;

match[pv]= v;

v = ppv;

}

}

}

int matches = 0;

for(int i = 0; i < n;++i){

if(match[i]!=-1){

++matches;

}

}

return matches/2;

}

1. Eulerian Path/Cycle

void eulerian\_path(int cur){

stack<int> st;

vector<int> ans;

st.push(cur);

//V is multiset

while(!st.empty()){

int cur = st.top();

if(V[cur].size()){

auto it = V[cur].begin();

st.push(\*it);

V[cur].erase(it);

//use this for bidirectional graph

//if(V[\*it].count(cur)){

// V[\*it].erase(V[\*it].find(cur));

//}

}else{

ans.pb(cur);

st.pop();

}

}

}

1. Online Bridge

// O(N + M)

int psetBridge[N], psetComp[N]; // UF super vertice, UF tree

int par[N];

bool seen[N];

int size[N];

int bridge;

int n;

void reset() {

for(int i = 0 ; i <= n ; i++) {

psetBridge[i] = psetComp[i] = i;

par[i] = -1;

seen[i] = 0;

size[i] = 1;

}

bridge = 0;

}

int findsB(int x) {

if(x == -1) return -1;

return x == psetBridge[x] ? x : psetBridge[x] = findsB(psetBridge[x]);

}

int findsC(int x) {

x = findsB(x);

return x == psetComp[x] ? x : psetComp[x] = findsC(psetComp[x]);

}

void makeRoot(int x) { // make super vertice which contains x root of its tree

x = findsB(x);

int root = x;

int chld = -1;

while(x != -1) {

int papa = findsB(par[x]);

par[x] = chld;

psetComp[x] = root;

chld = x;

x = papa;

}

size[root] = size[chld];

}

void mergePath(int u,int v) { // remove bridge between u and v in tree

vector<int> vu,vv;

int lca = -1;

while(1) {

if(u != -1) {

u = findsB(u);

vu.push\_back(u);

if(seen[u]) {

lca = u;

break;

}

seen[u] = 1;

u = par[u];

}

if(v != -1) {

v = findsB(v);

vv.push\_back(v);

if(seen[v]) {

lca = v;

break;

}

seen[v] = 1;

v = par[v];

}

}

for(int i = 0 ; i < 2 ; i++) {

vector<int> &proc = i ? vv : vu;

for(int x : proc) {

psetBridge[x] = lca;

if(x == lca) {

break;

}

bridge--;

}

for(int x : proc)

seen[x] = 0;

}

}

void addEdge(int u,int v) {

u = findsB(u);

v = findsB(v);

if(u != v) {

int uu = findsC(u);

int vv = findsC(v);

if(uu != vv) {

bridge++;

if(size[uu] > size[vv]) {

swap(u,v);

swap(vv,uu);

}

makeRoot(u);

par[u] = psetComp[u] = v;

size[vv] += size[u];

}

else

mergePath(u,v);

}

}

1. FFT

const double PI = acos(-1);

typedef complex<double> base;

void fft (vector<base> & a, bool invert) {

int n = (int) 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);

base wlen (cos(ang), sin(ang));

for (int i=0; i<n; i+=len) {

base w (1);

for (int j=0; j<len/2; ++j) {

base u = a[i+j], v = a[i+j+len/2] \* w;

a[i+j] = u + v;

a[i+j+len/2] = u - v;

w \*= wlen;

}

}

}

if (invert)

for (int i=0; i<n; ++i)

a[i] /= n;

}

void multiply (const vector<int> &a, const vector<int> &b, vector<int> &res){

vector<base> 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] \*= fb[i];

fft (fa, true);

res.resize (n);

for (size\_t i=0; i<n; ++i)

res[i] = int (fa[i].real() + 0.5); // <- beware res[i] negative, floor function retard

}

// just check if g^((p-1)/r) % p != 1 for every possible r (r = prime factor of p-1) instead of find g such that g^1, g^2, g^3, ... g^p-1 are the permutation of {1, 2, 3, ..., p-1}

bool check\_generator\_version\_langrage(ll g) {

for(int i = 1; i <= top; i++) {

ll value = power(g, (p-1) / prime\_factors[i]);

if (value == 1) return false;

}

return true;

}

/\*\* FFT Modular Arithmetic \*\*/

const int mod = 7340033; // c \* 2^k + 1

const ll root = 5; // root = g ^ c % mod

const ll root\_1 = 4404020; root\_l = (root)^-1 % mod

const ll root\_pw = 1<<20; root\_pw = (1 << k)

int rev\_element[7340033];

ll getmod(ll a, ll tmod) {return ((a%tmod)+tmod)%tmod;}

void fft (vector<ll> & a, bool invert) {

int n = (int) 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) {

ll wlen = invert ? root\_1 : root;

for (int i=len; i<root\_pw; i<<=1)

wlen = ll (wlen \* 1ll \* wlen % mod);

for (int i=0; i<n; i+=len) {

ll w = 1;

for (int j=0; j<len/2; ++j) {

ll u = a[i+j], v = ll (a[i+j+len/2] \* 1ll \* w % mod);

a[i+j] = getmod(u+v,mod);

a[i+j+len/2] = getmod(u-v,mod);

w = ll (w \* 1ll \* wlen % mod);

}

}

}

if (invert) {

ll nrev = rev\_element[n];

for (int i=0; i<n; ++i)

a[i] = int (a[i] \* 1ll \* nrev % mod);

}

}

void precalc(){

rev\_element[1] = 1;

for (int i=2; i<mod; i++)

rev\_element[i] = (mod - (mod/i) \* rev\_element[mod%i] % mod) % mod;

}

void multiply (const vector<ll> & a, const vector<ll> & b, vector<ll> & res) {

vector <ll> 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);

forn(i,n)

fa[i] \*= fb[i];

fft (fa, true);

res.resize (n);

forn(i,n) // for(i=0;i<n;)

res[i] = fa[i] % mod;

}

void FWHT(vi &P,int bits ,bool inverse = false) { //FWHT xor on vector P

int x = 1<<bits ,u,v;

for (int len = 1; 2 \* len <= x ; len <<= 1)

for (int i = 0; i < x; i += 2 \* len)

for (int j = 0; j < len; j++){

u = P[i + j] ;

v = P[i + len + j] ;

P[i + j] = (u + v)%mod ;

P[i + len + j] = (u - v + mod)%mod ;

}

if (inverse){

int xinv = pmod(x,mod-2) ;

for (int i = 0; i < x ; i++) P[i] = ((ll)P[i]\*(ll)xinv)%mod ;

}

}

FWHT(P,16);

FN(i,1<<16) P[i] = pmod(P[i],N) ; //pmod(x,y)=x^y

FWHT(P,16,true);

void to\_transform(ll dim, ll \*data) { // and transform

ll len, i, j, u, v;

for (len = 1; 2 \* len <= dim; len <<= 1) {

for (i = 0; i < dim; i += 2 \* len) {

for (j = 0; j < len; j++) {

u = data[i + j];

v = data[i + len + j];

data[i + j] = v;

data[i + len + j] = (u + v);

moddo(data[i + len + j]);

}

}

}

}

void inv\_transform(ll dim, ll \*data) {

ll len, i, j, u, v;

for (len = 1; 2 \* len <= dim; len <<= 1) {

for (i = 0; i < dim; i += 2 \* len) {

for (j = 0; j < len; j++) {

u = data[i + j];

v = data[i + len + j];

data[i + j] = mod - u + v;

data[i + len + j] = u;

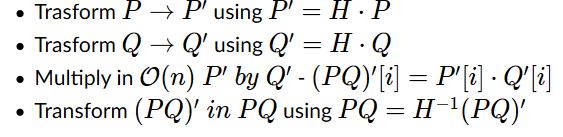
moddo(data[i + j]);

}

}

}

}



And matrices : 0 1 Or matrices : 1 1 inv and : -1 1 inv or : 0 1  
 1 1 1 0 1 0 1 -1

1. Smallest String Rotation

int minimumExpression(string s) {

s = s + s;

int len = s.size(), i = 0, j = 1, k = 0;

while(i + k < len && j + k < len) {

if(s[i+k] == s[j+k]) k++;

else if(s[i+k] > s[j+k]) { i = i+k+1; if(i <= j) i = j+1; k = 0; }

else if(s[i+k] < s[j+k]) { j = j+k+1; if(j <= i) j = i+1; k = 0; }

}

return min(i, j);

}

1. Z-Algorithm

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-L] == s[R]) R++;

z[i] = R-L; R--;

}

}

}

1. KMP

const int MAXN = 1000005;

#define MAX\_N 100010

char T[MAX\_N], P[MAX\_N]; // T = text, P = pattern

int b[MAX\_N], n, m; // b = back table, n = length of T, m = length of P

void kmpPreprocess() { // call this before calling kmpSearch()

int i = 0, j = -1; b[0] = -1; // starting values

while (i < m) { // pre-process the pattern string P

while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b

i++; j++; // if same, advance both pointers

b[i] = j;

} }

void kmpSearch() { // this is similar as kmpPreprocess(), but on string T

int i = 0, j = 0; // starting values

while (i < n) { // search through string T

while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b

i++; j++; // if same, advance both pointers

if (j == m) { // a match found when j == m

printf("P is found at index %d in T\n", i - j);

j = b[j]; // prepare j for the next possible match

} } }

1. Manacher’s Algorithm (palindrome count)

const int MAXN = 1000005;

char s[MAXN];

int odd[MAXN], even[MAXN];

int len;

void manacher() {

// construct odd length

for(int i = 0, l = 0, r = -1 ; i < len ; i++) {

int k = (i > r ? 1 : min(odd[l + r - i],r - i));

while(i + k < len && i - k >= 0 && s[i + k] == s[i - k]) k++;

odd[i] = k--;

if(i + k > r){

l = i - k;

r = i + k;

}

}

//construct even length

for(int i = 0, l = 0, r = -1 ; i < len ; i++) {

int k = (i > r ? 0 : min(even[l + r - i + 1],r - i + 1)) + 1;

while(i + k - 1 < len && i - k >= 0 && s[i + k - 1] == s[i - k]) k++;

even[i] = --k;

if(i + k - 1 > r) {

l = i - k;

r = i + k - 1;

}

}

}

1. Suffix Array + LCP

class Element\_suffix{

public:

int rank\_now, rank\_pref, pos;

};

class Suffix{

private:

inline bool same\_rank(Element\_suffix a, Element\_suffix b) {

return a.rank\_now == b.rank\_now && a.rank\_pref == b.rank\_pref;

}

inline void reset\_freq(bool is\_sort\_now) {

for(int i = 0; i <= end; i++) freq[i] = 0;

for(int i = 0; i < n; i++) freq[ is\_sort\_now ? suf[i].rank\_now+1 : suf[i].rank\_pref+1 ]++;

start[0] = 0;

for(int i = 1; i <= end; i++) {

start[i] = freq[i-1];

freq[i] += freq[i-1];

}

}

public:

int sorted[20][MAX], freq[MAX], start[MAX], SA[MAX], end, n;

Element\_suffix suf[MAX], tmp[MAX];

void build\_suffix() {

n = strlen(s);

if (n == 1) {

SA[0] = 0;

return;

}

end = max(n, 1 << 8);

for(int i = 0; i < n; i++) sorted[0][i] = (int)s[i];

int step = 1;

for(int cnt = 1; cnt < n; step++, cnt \*= 2) {

for(int i = 0; i < n; i++) {

suf[i].rank\_pref = sorted[step-1][i];

suf[i].rank\_now = (i + cnt < n) ? sorted[step-1][i+cnt] : -1;

suf[i].pos = i;

}

reset\_freq(1);

for(int i = 0; i < n; i++) tmp[start[suf[i].rank\_now+1]++] = suf[i];

reset\_freq(0);

for(int i = 0; i < n; i++) suf[start[tmp[i].rank\_pref+1]++] = tmp[i];

for(int i = 0; i < n; i++) {

sorted[step][suf[i].pos] = (i && same\_rank(suf[i], suf[i-1])) ? sorted[step][suf[i-1].pos] : i;

}

} step--;

for(int i = 0; i < n; i++) SA[sorted[step][i]] = i;

}

};

int compute\_lcp(int x, int y) {

int ans = 0;

for(int k = 20; k >= 0; k--) {

int s = (1 << k);

if (x + s - 1 < n && y + s - 1 < n && sorted[k][x] == sorted[k][y]) {

ans += s;

x += s;

y += s;

}

}

return ans;

}

void buildLCP(){

phi[SA[0]] = -1;

for(int i = 1 ; i < len ; i++)

phi[SA[i]] = SA[i - 1];

for(int i = 0, l = 0 ; i < len ; i++){

if(phi[i] == -1)

PLCP[i] = 0;

else{

while(s[i + l] == s[phi[i] + l]) l++;

PLCP[i] = l;

l = max(0,l - 1);

}

}

for(int i = 0 ; i < len ; i++)

LCP[i] = PLCP[SA[i]];

}

1. Biconnected Component

void dfs(int v, int bef = -1) {

num[v] = low[v] = counter++;

for (int u : adj[v]) {

if (num[u] == -1) {

edge.emplace\_back(v, u);

if (v == root)

childroot++;

dfs(u, v);

if (childroot > 1 && v == root) {

artp[v] = 1;

while (edge.size() > 0) {

auto it = edge.back(); edge.pop\_back();

block[nblock].push\_back(it);

if (it == make\_pair(v, u))

break;

}

nblock++;

}

if (low[u] >= num[v] && v != root) {

artp[v] = 1;

while (edge.size() > 0) {

auto it = edge.back(); edge.pop\_back();

block[nblock].push\_back(it);

if (it == make\_pair(v, u))

break;

}

nblock++;

}

if (low[u] > num[v])

bridge.emplace\_back(u, v);

low[v] = min(low[v], low[u]);

}

else if (bef != u && num[v] > num[u]) {

low[v] = min(low[v], num[u]);

edge.emplace\_back(v, u);

}

}

}

int main() {

for (int i = 0; i < gr.n; i++) if (gr.num[i] == -1) {

root = i;

childroot = 0;

edge.clear();

dfs(i);

artp[i] = childroot > 1;

if (edge.size() > 0) {

while (edge.size() > 0) {

auto it = edge.back(); edge.pop\_back();

block[nblock].push\_back(it);

}

nblock++;

}

}

}

1. Aho Corasick

#include<bits/stdc++.h> /\*\* Aho-Corasick Dictionary Matching \*\*/

const int NALPHABET = 26;

struct Node {

Node\*\* children, go;

bool leaf;

char charToParent;

Node\* parent, suffLink, dictSuffLink;

int count, value;

Node(){

children = new Node\*[NALPHABET];

go = new Node\*[NALPHABET];

for(int i = 0; i < NALPHABET;++i){

children[i] = go[i] = NULL;

}

parent = suffLink = dictSuffLink = NULL;

leaf = false;

count = 0;

}

};

Node\* createRoot() {

Node\* node = new Node();

node->suffLink = node;

return node;

}

void addString(Node\* node, const string& s, int value =-1) {

for(int i = 0; i < s.length(); ++i){

int c = s[i] - 'a';

if(node->children[c] == NULL){

Node\* n = new Node();

n->parent = node;

n->charToParent = s[i];

node->children[c] = n;

}

node = node->children[c];

}

node->leaf = true;

node->count++;

node->value = value;

}

Node\* suffLink(Node\* node);

Node\* dictSuffLink(Node\* node);

Node\* go(Node\* node, char ch);

int calc(Node\* node);

Node\* suffLink(Node\* node) {

if (node->suffLink == NULL){

if (node->parent->parent == NULL){

node->suffLink = node->parent;

} else {

node->suffLink = go(suffLink(node->parent),node->charToParent);

}

}

return node->suffLink;

}

Node\* dictSuffLink(Node\* node) {

if(node->dictSuffLink == NULL){

Node\* n = suffLink(node);

if (node == n){

node->dictSuffLink = node;

} else {

while (!n->leaf && n->parent != NULL){

n = dictSuffLink(n);

}

node->dictSuffLink = n;

}

}

return node->dictSuffLink;

}

Node\* go(Node\* node, char ch) {

int c = ch -'a';

if (node->go[c] == NULL){

if (node->children[c] != NULL) {

node->go[c]= node->children[c];

} else {

node->go[c]= node->parent == NULL? node : go(suffLink(node), ch);

}

}

return node->go[c];

}

int calc(Node\* node) {

if (node->parent == NULL) {

return 0;

} else {

return node->count + calc(dictSuffLink(node));

}

}

int main() {

Node\* root = createRoot();

addString(root,"a",0);

addString(root,"aa",1);

addString(root,"abc",2);

string s("abcaadc");

Node\* node = root;

for (int i = 0; i < s.length(); ++i){

node = go(node, s[i]);

Node\* temp = node;

while (temp != root) {

if (temp->leaf) {

printf("string (%d) occurs at position %d\n", temp->value, i);

}

temp = dictSuffLink(temp);

}

}

return 0;

}

1. Tarjan’s Articulation Point and Bridge

void tarjanAPB(int u){

dlow[u] = dnum[u] = nxt++;

for ( int i = 0; i < adlis[u].size(); i++ ){

int v = adlis[u][i];

if ( dnum[v] == -1 ) {

dpar[v] = u;

if ( u == dfs\_root ) child\_root++;

tarjanAPB(v);

if ( dlow[v] >= dnum[u] ) {

isAP[u] = true;

}

if ( dlow[v] > dnum[u] ) {

is\_bridge[u][v] = true;

}

dlow[u] = min(dlow[u], dlow[v]);

}

else if ( v != dpar[u] ) {

dlow[u] = min(dlow[u], dnum[v]);

}

}

}...

nxt=0;

RESET(dnum,-1);

RESET(dlow,-1);

RESET(dpar,-1);

RESET(isAP,0);

RESET(is\_bridge,0);

for ( int i=0; i < nvert; i++ ){

if ( dnum[i] == -1 ){

dfs\_root = i;

child\_root = 0;

tarjanAPB(i);

is\_AP[dfs\_root] = (child\_root > 1);

}

}

1. Tarjan’s SCC

int low[MAXN], num[MAXN], vis[MAXN];

vector<int> stek, adj[MAXN];

int cnt;

void tarjan(int v) {

low[v] = num[v] = cnt++;

vis[v] = 1; // mark as inside stack

stek.pb(v);

for(int nex : adj[v]) {

if(vis[nex] == 0) tarjan(nex);

if(vis[nex] == 1) low[v] = min(low[v],low[nex]);

}

if(low[v] == num[v]) {

while(1) {

int u = stek.back(); stek.pop\_back();

vis[u] = 2; // mark as out of stack and has been processed

if(u == v) break;

}

}

}

1. Lowest Common Ancestor

void compute\_depth(int x, int current\_depth){

depth[x]=current\_depth;

for (int i=0; i<adlis[x].size(); i++){

compute\_depth(adlis[x][i], current\_depth+1);

}

}

int lca(int a, int b){

if (depth[a] > depth[b]) swap(a,b);

int delta\_depth=depth[b]-depth[a];

for (int i=0; i<=lg; i++){

if (delta\_depth & (1<<i)){

b=pparent[b][i];

}

}

if (a==b) return a;

for (int i=lg; i>=0; i--){

if ((pparent[a][i] != -1) && (pparent[a][i]!=pparent[b][i])){

a=pparent[a][i];

b=pparent[b][i];

}

}

return pparent[a][0];

}

...

for (int j=1; j<=lg; j++){

for (int i=0; i<n; i++){

if (pparent[i][j-1] != -1){

pparent[i][j]=pparent[pparent[i][j-1]][j-1];

}

}

}

1. Josephus problem

int x = 0;

for (int i = 2; i <= n; ++i)

x = (x + i) % i;

x = x + 1;

1. Lucas Theorem

// in case MOD is small yet a and b can be larger than it

int comb (int a,int b) {

int res = 1;

while(a > 0 && b > 0) { // beware of a%mod < b%mod, should return 0

res = (res \* C[a % MOD][b % MOD]) % MOD;

a /= MOD; b /= MOD;

}

return res;

}

1. Extended Euclid

int extended\_euclid(int a, int b, int & x, int & y) {

if (a == 0) { x = 0; y = 1; return b; }

int x1, y1;

int d = extended\_euclid(b%a, a, x1, y1);­

x = y1 - (b / a) \* x1; y = x1;

return d;

} // to obtain solution = c, x0 \*= c / gcd, y0 \*= c /gcd

![A screenshot of a cell phone

Description generated with very high confidence](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAeAB4AAD/4RDuRXhpZgAATU0AKgAAAAgABAE7AAIAAAAMAAAISodpAAQAAAABAAAIVpydAAEAAAAYAAAQzuocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHl1c3Vmc2hvbGVoAAAFkAMAAgAAABQAABCkkAQAAgAAABQAABC4kpEAAgAAAAMxNgAAkpIAAgAAAAMxNgAA6hwABwAACAwAAAiYAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAxNzoxMToyNSAxMzo1OToxMQAyMDE3OjExOjI1IDEzOjU5OjExAAAAeQB1AHMAdQBmAHMAaABvAGwAZQBoAAAA/+ELHmh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMTctMTEtMjVUMTM6NTk6MTEuMTYyPC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPnl1c3Vmc2hvbGVoPC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAF8CBwMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APpCiiigAorkPF3i/V/DMGpXsGgQ3WnafBHK9xNfGEyliQVRRE2SOMkkdaq3vinxTaapYx3mk2FjbS39tbORK9yLhJt3zRyYj2su3BDKeoPTGRasHodzRRRQAUUVR1C7v7e6sY7DTftkU82y5l89Y/syYJ34PL84GB60AXqKwNJ1a/1nXNQMD28Om6fdNZtE8DNNM6opLB94CjLYxtOduc88aN5rmk6dcLb6hqdnazONyxz3CIzD1AJzR28wL1FUbzXNJ0+ZIb/U7O1ldd6pPcIjMvPIBPI4P5VYtLy2v7VbmxuIbmB87ZYXDq2Dg4I4PNAE1FFU21OFdbj0sq/nyW73AbA27VZVIznOcsO3rR1sHmXKKjnYpbyMrrGwQkO65VeOpGRkfiK5LwL44h8ReFf7S1bUNLjuFMkkiQSBPKhDlVZ1Z2K5Azycc0AdjRVSLVdPnS2eC+tpFu8/ZmSZSJsDJ2YPzYAJ49Kt0AFFFFABRUN3e2un2zXF/cw20C43SzSBFGfUniov7QS50qS90gx6iPLZoRDMu2ZhnCh+gyRjPajoBboqtp011c6bbzahaCyupI1aW2Eok8piOV3DhsdMirNGwBRRRQAUUUUAFFQXouTZSiwkiiudp8t5ozIgPuoZSR+IrM8H6xc6/wCD9M1W+SJLi7gEkiwghAfYEk4/GgDaoqre6naadJaR3kvlteTi3gG0nfIVLAcDjhTyeOKjs7u/m1S/gutN+zWkBQW10Z1f7Tlcsdg5TaeOevWgC9RRRQAUVDeXcNhYz3d2/lwW8bSyPgnaqjJOByeBWBda9ep4s8PwWklrLpOsRStzC4mUrHvDBt2MHI4K5HrzwAdLRRRQAUUVHOJWt5BbOkcxUiN5ELqrY4JUEEjPbI+ooAkorgdH8X69eN4ZkvP7OMWqXdzaXSRW8isrRCUhkYyHAPljgg9+fTvqA62Ciiuet9W1L/hYN1oty1q9kLBbyAxwssikyFCrEsQ3TOQB19uTrYOlzoaKKxjrsl7pmn3/AIbsv7Vt7u4VHfzRB5UWSGlw4yduPu4yaANmiiigAooooAKKKKACiisjxNrL6Ho32iGMSTzTxW0Ct90SSuEUtyOAWyRkZx1FAGvRWbJqB0TRRc+I72BmV1SSe3tnjQl3CoAm5yOWUdT68CqFvq2pf8LButFuWtXshYLeQGOFlkUmQoVYliG6ZyAOvtydbf13Dpc6GiiqOt6pHomg32qTo0kdnbvOyL1bapOB9cUm7K40m3ZF6is7RG1OXTkm1ie1llmAkRbaBohGpAO07nbcR/e4z6Vo1TVnZkp3V0FFFFIYUUUUAFFFFABRRRQBx3xVEs/w51SytbW7u7m7jEcMVrayTMzbgedinaMA8nAqp42li1bQvDyDT9SubeXU4JJo00+43JEhO8uoTco56EDPbIrvKKP81+APX8fxPHb7QlbSPE9rpWg3UcFprVte2UA02SNQimISNApQAn5ZOE6j2IzNLawXN14yWDS9Y0+z1SSzkglg0SU7yOHdoSg3ruPzoRuZS3BzmvXKKOlv66f5IfW/9df8zgPC+qr4csBFc6E1tb3+rC3gl07TJbWI7kUCV4JDuhG4bfQkZ75NvxzbQPr3ha5/s24uZ7fUgzzwWMkxihMbhtzIp2qWKcE89exx1s9lb3NxbzTpve3YtFljhWIxnb0Jx0J6ZOOtT0+qfa34WJ2TRwvw403TrK68QPaaI2nSPqMjRO+mPal7c42BSyLlchvlHT0GRXHaxp2qyaNr9lqGkald32q+IFa6ljspJFWyR0KFWC4YbRgBckZOQMGva6KXbyS/C3+Q+/rf8/8AM8k1+a61nVLG1uPCOqWMA1S1vNSnWCS4WRUiyFG1TkBsLhR2JwOa6Ce4ufDej+GotFsYtGtL3VY4JrJk3uEkZm+8T8pIHIwSCcZ4ru6zNX8P6drrWzalHNIbWUTQ+XcyxbHHRvkYZI9TQtLeqf5f5Ct+X+f+ZfnjaaB445ngZhgSRhSy+43Aj8wa5GXTLv8A4WJaJ/bl/uOlTHzNkG4DzYuP9VjH4Z4rsQNqgDPA7nNGBuzgZxjNHW/9bND6Nf1umYniSz1FvA2q2mmzS3eoPZypC8gUM7FTgfKAM84HFednw5Pcx2l5a6fJa2Hh7QTDareRNbme9IDD5W2nCuqnJ4LetewVS1jR7DX9Ll07V7Zbm0mxvjYkZwQRyCCOQOlJrf8Aruv1H2/rs/0PK/Blo1z4m0TTbFJrZvDWhl5TeWrrtu7gjdlG2k8bue/Y16D4E1a713wPpmp6k6yXVzEXkZVCgncRwB9KvWegabYWtxBZwPELlt08izP5sh6ZMmd5OOM54HFP0XRLDw/piafpMckNrH9yN5nk2+wLkkD2q27/ANebb/MhKxR1PTb24v3kgt96EDDf21c2/b+4ilR+HWjTdNvbe/SSe22IAct/bVzcY4/uOoU/j0rdoqVoVucR4jtrn/hZeiX9/b3U+i2dpM8Yt7d5wl0SACyoCfuE4OOufWr8em6Vb/D67s00KW209o5T9gkgMzvkkglBuYkn5sdR7YrqKKTV48o07S5jxa505rrwtoKzaVdz3lnoaQfYNT8PXFzbvJjaVVlAaCQFMF+AVZDyBkdz44jlm8H2UE2jm8EtxALiHy5bpLcD5izxRENOqkAbeh4J4FdhRVSd/vv+N/6/AlK1vSx4o+mlvAl1pt7omoNJD4gjuLSO30a4jEUDSIxaFVDGNdvmcBsjODycVpnwdY65c+MdCttIexs3aC50zzNNeG3SdI9rMm5AvLBQ2PvDPUZr1iqep6Xa6vai2vvOMW7cVhuJId3BGGKMNynPKnIPcUulv66f5XGt7/11/wAzB8E2sd9bnxRdaVDYajqkMasojUMkaKABkdQSC302j+EVx1tpd6dfupdB0+SS8uTeut5f6PNaXdmzRsEzdcJOm7gKckAqc/Lx60qqihUAVVGAAMAClolq/wAAjojyzRbFLTUtAn0fQ9Q025sbeVtenexlQ3H7rBVm2/6S5kwwK7+hI6809H0/7Pp3gfUdR0K+KWVvc2OoK2lyvKoaMhVaMIXZCSecFfm969fopvW/9d/8xJWPGLTQbc+DPDv9qeF7iQadr04uLeTSHkeK3dpmwECEtH80Z+UFcgdxV6902eS58YxeGdJvrWK7aylKRWMtr9pjQ/6QqMyqCxUkYBy2T1r1mil/X5f5D63/AK6/5nj+u6NbXWn+JP8AhH/Dt3baJLpIhSzXSpYvPvt2Ukjg2BgVGAZNoHTnjI1NFFv4f8aSXWl6HqcVk/h9XnWDTJlNxcIxfDFlAaUqSMscknBOeK9MqK6to7y1kt593lyqVbY7IcH3Ugj8DRqtv63/AM/wQaPf+tv8vxZz+tG28XfDS7mhsWuI77Tmmgt7iEF9xQsnynPzA4xjv0rjYrDSbqXwXYjw3ex2MCSm+gOizxxLI0IX94PLAOXHJOQcZPFeqxxpDEkUKLHGihVRRgKB0AHYU6jS7t1Dor9DxiPTdQk8L6Zbw6bcppFnq9359le6NcToImZjATbZRpEGQcDO0kEj5TixF4etzqHhaDU9LvNQt4ru6zIdFmijS2dW2RlMuViDnASQggDlQvJ9fopg9f69f8zjPhoJLfwzeWrWV3apBqFz9nguLWS32wtIzRhA6j5cHt06cVyHheGex8ZaTqf9g3WmxSC6W9SDSLrzEaQhlWedixuDkZDgBQfqK9iopdb+Vv0Df77nkuiTTxQeEFl0rWEa11i8ecNpdwPLRxMFY5ThT5qc+59Djf8AiHZRXt9pvmQSzNBHK6xz6HLqdnKSANsiR/Mj55VxjA3DnOK7uik1dB1v/W9/1PJm0wK0r+L/AAvdXcb6Pbx6XbW9rJeLaOsZEkSkBjE+7b87beg+b5alttL1S6v20rX7K6vbyXwsLOaea1eSB7kEvtaYrsLcg7s4yOua9Uopy96/z/X/ADBafh+n+R5J4dsIbnxH4Z+0+HrxEGgvp+oSXOkyorS7YwI5CyYIwrAE/L2B7VSsdKsLLwX4ejj8MXkGrWOrQy3LR6HMJFCyks29Y/mGwDkE8YFe0UVXNrfzv+N/1J5dLeVvwseR3uj3bxajC+kXsni+bVTJZaqLVyI4TICjC5xtSMR/KY9wPUbeap6/odo3/CbXNj4evmvmvIJdNkj0mfIkATfJD8mAS6tll64GSRivaKKlaf16f5Fbu/8AX9anlt9a/bPHA1Kz0ue+nlu7UiLU9DnR4VXaTJBeAARKByY37q4wN2DtfEO3+2XejwS6Mt/AHlkM8+nzX8MLBcBWt42AYtuOHfhcH147iiiytb+v6/rbQFvc8Ys9Ht9Q0/wRZ+INC1OeS0E9rfGbTLglIyjKis4XG3JXHJUdegzU2peF7628QahapDJCY3t/7FuYtGmvJYoUVQkcVyJQkOGVgwk2g7iSSG49hop315hW0sch45s5rt9Fe5s5r/RYbstqdpDEZTIuwhC0S5Miq+0lQD2OOK4rU9Eik0id38O3T6S+vW8umWTaVJK9vbho/tBEQQmKNtrnYQuR25xXslFJaO/9dP8AIfS39df8zjPiHo1rf/D8Q22jrdray2zwW0dnvaNBKm4JHjI+TcMAdMiuf1HT9E1PxXOJPDl7JpMOg+VbxNotwsKyb2YKqeWBuAPHGRnjnNep0Umr3+f4qw07f153PF30vU7+w0A6zaTNYroUdri+0G5vmt7pTiQ+UrK6ORtxIQQQvBHfrvE+nS3HwZmstUtLnVL5dPEcayWfmzPOE2q+xN+GzzkE4zya7qiqk+ZNd/8Ag/5ij7rT7f1+h5c+k6ZDqVtLq3hq4u/D39kqljZxaTJIsFxvYy5twmY3bK4cqOh5rPtfCkd/rHhqy8ZaPeXkjaXPBeytbTSAbpFMCSToMFlUHkt8pGcg4New0UX1v/XX/MSVlb+un+R4nq3h6y8vxrJbeHL77V9tgbTJItKn3KQEDvCwTg7lYll5PBJPFdh4WtbPT/iLqq6PpV3Yafc6fbsCdPmgiklUuWJLKBv2suc8k57g13lFCdrf10sN9f663CiiikAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFY/izXW8M+Fr7WVt47n7HH5hieYx7x6Bgrc+nH5Vo2M8l1p1vcXEPkSyxK7xbt3lkjJXOBnHTOKAJ6KKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiuS8deI9Y8NaTdalYW9qba1jjb/AEhS5uZHkCiJQrAr1+8c9RgHBoA62isbw5P4gmtbn/hJ7azgmWciA2hOHjwMMQScHOe9WJtSuo5nRNEv5lU4EiPAFb3GZQfzAo2BamjRVOzvZ7mUrNpl1aADIeZoiD7fI7H9K5zxx4k1vw3Yy3un21o0EbQRRJcAs13LJJt2LtYbMDnJBznpxyAdfRXOWOvXml6W0vjuTT9Pna6MNv8AZ2OJkJAQhcscknpk4roidqknOAM8DNAC0Vl/8JBZ/wDPHUf/AAWXP/xurDXzzaVNdadbyzSrGxihmjaFncDhcOARk8ZNHQC5RXA+E/E3jDxDcJO9ppL6bFqE9pcTRh0ZkjyPMQFz1bjHPQ9K3/8AhM9M/wCfXW//AAQ3v/xmn0uLqb9FYH/CZ6Z/z663/wCCG9/+M0f8Jnpn/Prrf/ghvf8A4zSGY/xVi+1eGLGyuCE0+71S2iv5WbascG/LFj2GQoz7034eWrjVfEl/ZQLa6HdXaLpsMa7Y2VE2NKi+jEdR1xmtr/hM9M/59db/APBDe/8Axmj/AITPTP8An11v/wAEN7/8Zojpfz/4H+QPW39d/wDP8DforA/4TPTP+fXW/wDwQ3v/AMZo/wCEz0z/AJ9db/8ABDe//GaAN+isD/hM9M/59db/APBDe/8Axmt+gAooooAKKKKACiuR8T+JtX0fxPoun2FpbPBqN1HB+9yXkB3GQrhvlCKoOSDksB2q74S8Q3HiNNWuJIo47W21GW0tWTOZEjwCxye7bumKFr/Xp/mD0/r1/wAjoaKzDqt4CQNA1E+4kt+f/ItWbO6mug/n2FxZ7cY89ozu+mx2/XFAFqiuGufG2p2HjHUrG9tLZNPsNNmv5MEmSNFOIyWDFTvwx24BAxznIrW8LeKo9Y0/TYdRmt4taurIXktnDn92p2nnrj769Tzmhaq6/rf/ACYPR2f9bf5o6Oiq95exWMavMs7BjgCC3eU/kgJH1rmtF8SRvrXiBZ/7SkjS9QQqbC4by1+zxEjGz5fmJOOOue9C1dgOtormPHviHUfDXhObVtJggkMILyNcqSqrtOBtDKSWbavtuz2xS6Rq/iGC2vLzxZZQRWqpC9v/AGbDLNKSw+dTGu5jtJAyB2JwB0FrcHodNRWB/wAJnpn/AD663/4Ib3/4zR/wmemf8+ut/wDghvf/AIzQBv14x441ONtT8aya95ck1jZxW2i2cgyV8xCWnRfUN/GPuheor0j/AITPTP8An11v/wAEN7/8Zo/4TPTP+fXW/wDwQ3v/AMZpNXGnYk8HSQyeDtMFqZHiigEKyScmXZ8m8HJyG25B7gg1t1gf8Jnpn/Prrf8A4Ib3/wCM0f8ACZ6Z/wA+ut/+CG9/+M1Und3IirKxv0Vgf8Jnpn/Prrf/AIIb3/4zVzTfEFnqty0FrDqMbqhcm60y4t1xkDhpI1BPPTOevoaRRp0UUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVyviH4fad4k1c6jeX2owybI1EcMy+WDG25W2sp5B59DjkV1VFHW4dLFLS9Li0q3kjjllnkmlM0085BeVz3OAB0AGAAAAOKjm8P6NcTPLPpFhLI53M72yFmPqSRzWjRQBTs9I03T5TJYafa2sjDaWhgVCR6ZArA1r4e6drmvHVrjUNShn3wyKkUqeWjxHKMFZSM8njoc9M11dFHW4eRxXizw5dL4PTTNEtLnUp3vobqWRpYw7lZlkdmLMoyQDgAY4AwBXZxsXiVmjaNmAJRsZX2OCR+Rp1FHS39f1oAUhGVI6ZHaloo3AztB0Oz8OaLBpemhxbwbtpkbcxLMWJJ7kkmn3et6Vp90ltf6nZ208gBSKa4VGbJwMAnJ5q9XjviuEXPi3xfpwa5+zalDZpcXMFlNdm12DJwIlYKcYO1ivXdyKG9Q6HqVz4g0azjiku9XsYEnUtE0tyiiQDupJ5H0qj4n8SWujaNcyQ6rptrfC3ae3S8cESAc8JvUsD0yD1I69K4rTLHToPEuqXraXJqvhzWdKgTT2s7JpY9ig74iqL8hZju5AGSe9ENld6Nc+MlvtGvSNVsoxp6W1o06rGISggygIUqxxg4B6ilK9n/AF8vUcd18v0/4Y3LfxbqOqeFfDBt/Jg1bxCqnfGmUt0Cb5ZArE5wOADnkjOa6m81nTNLeKLU9TtLWSQfILidI2ftkAkZ59K838LJLa3fw6F7b3Ns0Wn3Vk8V1bvEyTCNGxhwDyEbkcHHWqHiSw1d7fxvFPpeo3WoateRW1vJFZySoliNvzKwBHALZUfNnnHernbmaXn+dl/n6ER1ir+X9fp6nsaOskavGwdGAKspyCPUVyNx4yaz+J7aBeX2mQWAsFuC037uXzWfasYYvhs8nhe4FW9L8RGO+0jRp9FvLH7XZGWF2AKR7P8Alm3QhguCRjjOK4nxBoOrahf+MrGLTpn1DXp7e2trowN5MdoqrljJjaMYbK5znHFT9rTz/wAv+D8ilrHXy/zO8sdWuoPGV5oWouJRJB9tsZdoUmPdteM46lWK4PcMM8jJLnSLuXx9ZarDJcx20NnJFOPtH7qQk/Kojz94cktjso55xl3EBHxW0C3iZnNlo9wZnPXazRqufqVP5Guzp6WTXn+bQurXp+SYUUUUhhRRRQBxPjTS9a1HXrCbStOlJsreV7W/t7qJGincbNrrICNm3kkAtxx77Xg7w9/wi3hGx0hpRPLAhM0oH+skYlnb8ya3KKForA9XczD4a0JiSdF04k8km0Tn9Ks2em2OnBxp9lb2ofG/yIlTdjpnA5q1RQB5h4n8Ia74l1zV44rWXTI9QMNpLeRXUTRT2iHcSyEGQPyygLtHqfXoW02+h+Jen3lvpcx0y20t7L7QskQVWLxsPlL7sAKR0rrqKFpb+ulgev8AX9dkFZ2naWbDUtVujKHGoXKzhduNmIkjx7/cz+NaNFAGbruhWfiLTfsGpCRrfzY5WVG27ijBgD7ZA4rSoqO52fZZfNfy02Hc+7G0Y5Oe1JuyY1qylH4h0Wa6NtFq9g86tsMS3KFg2cYxnOc8Y9aemt6VJqJ0+PU7Nr0MVNstwhkBAyRtznOOa8c8KRWNza+FE11ntYNOilgtGn02ZorqWZhsLSsnlcEAqQx3HBBHSt7wzo80Hgu00K+0S5/4SOzkuVS9ltm2QyPvP2hZiNpzlehLZ7cU3ov6/r/hxbs3vFPjgaXq2i22ialpN0brVI7G7tSfMmQMTlhtcbSMYwVPJH461tq11qfjS8sLSQR2GkxoLkhQTNNIu4Jk9Aq4JxySw5wCD5xNDdW/gnwRpjaFqovNI1W3kvlj02ZxEE3b33KpDgk5ypOc10umi/T/AIWFb6UGGqtcme2yNrHfbJ5fX3Uge4ofup9bX+e3+bDdrpe3y3/yR2UeuaTLqDWMWqWT3itta3W4QyA+m3Oc1eOdpwQDjgkV49pNrd2+seB4Lfw/qckGlWjmaSazkiBupdiyFmZeNoZ3LHgkYBJr0CDXbjXdJ16Gwsbiy1CxeW1SO5AG59m5GBBIwdymiWkW1rv+AR1a+RmeC/Fs3i7SNQim1TT4tQFxPFbfYwBIIkO1ZTE7MevPpgitvwrrUuuaMZLyNYr61nktLuNPuiWNsMR7Hhh7EVwvg/w7ePqHg9YrG5sLPw9YSNdvcQNC0t1KoDIAwBYA5JYcdOTXS+AgXufFFyn+pn1yYxnsdqojEf8AAlaqtrby/JpfjuTfS/n+abt8jr6KKKkoKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACsm48MaXc3lxdGKeGa5AE7Wt3LB5uBgFhGwDHHGTzjitaigCCysrbTrKKzsII7e3hUJHFGuFUegFT0UUAZ+saPb61axxTs8UkMqzwTxEB4ZFPDLnI9Rg8EEg9a0B055NFFAFZtPtm1NNQaLddJEYUkLE7VJBIA6DJAyepwPSrNFFAGfZaPBZ6pfajuea7vSoeSTHyoo+WNcdFGSfqSTWhRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAU2SNJY2jlRXRwVZWGQwPUEU6igDEh8H6JCLdVtpWhtnEkFvJdSvDGw+6VjZigx2447YrboooAKz20eD/hIF1iJniuTD5EwXG2dAcruHqpJwR6kd60KKACq1np9tp6zC0j2efK00pLFi7t1JJJPoPYAAcCrNFAEdxE01tLFHM0DuhVZUALISOozkZHvVfStLtdF0uDT7BCkEK4Xcclj1LE9ySSSe5NXKKACiiigD//Z)

For any integer r. Note: becareful X0 and Y0 may overflow after x0 \*= c/gcd.

To prevent overflow, change r and make x close to 0. Need modulo b/gcd(a,b).

1. Pollard-Rho + Miller Rabin

vector<long long> A({2, 3, 5, 7, 11, 13, 17, 19, 23});

// if n < 3,825,123,056,546,413,051, it is enough to test a = 2, 3, 5, 7, 11, 13, 17, 19, and 23.

long long largemul(long long a, long long b, long long n) {

// assert(0 <= a && a < n && 0 <= b && b < n);

long long r = 0;

for (; b; b >>= 1, a <<= 1) {

if (a >= n) a -= n;

if (b & 1) {

r += a;

if (r >= n) r -= n;

}

}

return r;

}

long long fastexp(long long a, long long b, long long n) {

// assert(0 <= a && a < n && b >= 0);

long long ret = 1;

for (; b; b >>= 1, a = largemul(a, a, n))

if (b & 1) ret = largemul(ret, a, n);

return ret;

}

bool mrtest(long long n) {

if (n == 1) return false;

long long d = n-1;

int s = 0;

while ((d & 1) == 0) {

s++;

d >>= 1;

}

s--;

if (s < 0) s = 0;

for (int j = 0; j < (int)A.size(); j++) {

if (A[j] >= n) continue;

long long ad = fastexp(A[j], d, n);

if (ad == 1) continue;

bool notcomp = false;

long long a2rd = ad;

for (int r = 0; r <= s; r++) {

if(a2rd == n-1) {notcomp = true; break;}

a2rd = largemul(a2rd, a2rd, n);

}

if (!notcomp) {

return false;

}

}

return true;

}

long long gcd(long long a, long long b) { return a ? gcd(b % a, a) : b; }

long long pollard\_rho(long long n) {

int i = 0, k = 2;

long long x = 3, y = 3; // random seed = 3, other values possible

while (1) {

i++;

x = largemul(x, x, n)-1; // generating function

if (x < 0) x += n;

long long d = gcd(llabs(y - x), n); // the key insight

if (d != 1 && d != n) return d;

if (i == k) y = x, k <<= 1;

}

}

1. Brent’s Cycle Finding

int f(int x) {

// state transition from x

return // something;

}

pair<int,int> brent(int x0) {

int lambda = 0, mu = 0, power;

int tortoise = x0;

int hare = f(x0);

// finding succesive power of two

// find lambda

lambda = power = 1;

while (tortoise != hare){

if (power == lambda) {

power <<= 1;

tortoise = hare;

lambda = 0;

}

hare = f(hare);

++lambda;

}

// find mu

tortoise = hare = x0;

for (int i = 0; i < lambda; ++i)

hare = f(hare);

while (tortoise != hare)

{

tortoise = f(tortoise);

hare = f(hare);

++mu;

}

return make\_pair(lambda, mu);

}

1. Erdos-Gallai (Graph Checker)

sort(d+1, d+n+1, greater<int>);

for (i=1;i<=n;i++)

x[i] = x[i-1] + d[i];

if (x[n]&1) {

printf("Not possible\n");

continue;

}

can = true;

for (k=1;k<=n;k++) {

sum = x[k];

tmp = k\*(k-1);

for (i=k+1;i<=n;i++)

tmp += min(d[i], k);

if (sum > tmp) {

can = false;

break;

}

}

if (can) printf("Possible\n");

else printf("Not possible\n");

1. Gaussian Elimination

struct Matrix {

int r,c;

vector<vector<long double> > vc;

Matrix(){}

Matrix(int r,int c) {

(\*this).r = r;

(\*this).c = c;

vc.assign(r,vector<long double>(c,0.0));

return ;

}

void swapRow(int x,int y) {

FOR (i,0,c)

swap(vc[x][i],vc[y][i]);

return ;

}

};

Matrix M;

void gaussianElimination(Matrix &M,unsigned long long result[]) {

RESET(result,0);

int r = M.r, c = M.c;

FOR (i,0,r) {

int mx = i;

FOR (j,i+1,r)

if (M.vc[j][i] > M.vc[mx][i])

mx = j;

M.swapRow(mx,i);

FOR (j,i+1,r)

FORD (k,c-1,i)

M.vc[j][k] -= M.vc[i][k] \* M.vc[j][i] / M.vc[i][i];

}

FORD (i,r-1,0) {

FORN (k,i+1,c-2) {

M.vc[i][c-1] -= M.vc[i][k] \* result[k];

M.vc[i][k] = 0;

}

M.vc[i][c-1] /= M.vc[i][i];

result[i] = M.vc[i][c-1];

M.vc[i][i] = 1.00;

}

return ;

}

1. Matrix Operations

typedef double mat[MAX][MAX];

**LU-Decomposition**

Decomposes an n x n matrix A into LU.

void LU\_decompose(int n, mat A, mat LU) {

REP(i, n) REP(j, N) LU[i][j] = A[i][j];

for (int i = 0; i < n; i++) {

for (int j = i+1; j < n; j++) {

double temp = LU[j][i]/LU[i][i];

for (int k = i; k < n; k++)

LU[j][k] -= LU[i][k] \* temp;

LU[j][i] = temp;

}

}

}

**Back Substitution**

Solves matrix equation Ux = b.

void back\_substitute(int n, mat U, double x[], double b[]){

for (int i = n-1; i >= 0; i--){

x[i] = b[i];

for (int j = i+1; j < n; j++)

x[i] -= x[j]\*U[i][j];

x[i] /= U[i][i];

}

}

**Forward Substitution**

Solves matrix equation Lx = b.

void forward\_substitute(int n, mat L, double x[], double b[]){

for (int i = 0; i < n; i++){

x[i] = b[i];

for (int j = 0; j < i; j++)

x[i] -= x[j]\*L[i][j];

**Linear System**

Solves matrix equation Ax = b.

void solve(int n, mat A, double x[], double b[]){

mat LU;

REP(i, n) REP(j, n) LU[i][j] = A[i][j];

for (int i = 0; i < n; i++)

LU[i][n] = b[i];

for (int i = 0; i < n; i++){

int pivot = i;

for (int j = i+1; j < n; j++)

if (fabs(LU[j][i]) > fabs(LU[pivot][i]))

pivot = j;

for (int j = i; j <= n; j++)

swap(LU[i][j], LU[pivot][j]);

for (int j = i+1; j < n; j++){

double temp = LU[j][i]/LU[i][i];

for (int k = i; k <= n; k++)

LU[j][k] -= LU[i][k] \* temp;

LU[j][i] = temp;

}

}

for (int i = 0; i < n; i++)

b[i] = LU[i][n];

back\_substitute(n, LU, x, b);

}

**Matrix Inverse**

Compute the inverse Ac of matrix A.

void inverse(int n, mat A, mat Ac) {

mat LU;

LU\_decompose(n, A, LU);

double x[MAX], e[MAX], y[MAX];

REP(j, n) {

REP(i, n) e[i] = 0;

e[j] = 1;

forward\_substitute(n, LU, y, e);

back\_substitute(n, LU, x, y);

REP(i, n) Ac[i][j] = x[i];

}

}

**Matrix Determinant**

Returns the determinant of matrix A.

double det(int n, mat A) {

mat LU;

LU\_decompose(n, A, LU);

double res = 1;

REP(i, n) res \*= LU[i][i];

return res \* (n % 2 ? -1 : 1);

}

1. Johnson’s Algorithm

all-pair shortest path in directed graph with negative edge

algorithm :

add a dummy source vertice, make dummy edge from this source

to every node in original graph with weight 0

run Bellman-Ford

if negative cycle exists, terminate, continue otherwise

transform each edge (u,v)'s weight to w(u,v) + dist(u) - dist(v)

run Dijkstra's V times

total complexity : O(VE log V)

in sparse graph, better than Floyd-Warshall

// don't forget to add (dist(v) - dist(u)) to normalize

1. Geometry

Proyeksi segitiga: BC^2 = AC^2 + AB^2 – 2AD.AC

#define EPS 1E-9

#define PI acos(-1)

// >>>> Constructor of point

struct point {

double x,y;

point() { x = y = 0.0; }

point(double \_x, double \_y) : x(\_x), y(\_y) {}

bool operator == (point other) const {

return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS));

}

};

// >>>> Constructor of vector

struct vec {

double x, y;

vec(double \_x, double \_y) : x(\_x), y(\_y) {}

};

// >>>> Constructor of line (ax + by = c)

struct line {

double a,b,c;

};

// Distance of two points

double dist(point p1, point p2) {

return hypot(p1.x - p2.x, p1.y - p2.y);

}

double DEG\_to\_RAD(double theta) {

return theta \* PI / 180.0;

}

// Rotate a point THETA degrees

point rotate(point p, double theta) {

double rad = DEG\_to\_RAD(theta);

return point(p.x \* cos(rad) - p.y \* sin(rad),

p.x \* sin(rad) + p.y \* cos(rad));

}

// Make a line l from 2 given points

void pointsToLine(point p1, point p2, line &l) {

if (fabs(p1.x - p2.x) < EPS) {

l.a = 1.0 ; l.b = 0.0 ; l.c = -p1.x;

} else {

l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);

l.b = 1.0;

l.c = -(double)(l.a \* p1.x) - p1.y;

}

}

// Check if two lines are parallel

bool areParallel(line l1, line l2) {

return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS);

}

// Check if two lines are same

bool areSame(line l1, line l2) {

return areParallel(l1, l2) && (fabs(l1.c - l2.c) < EPS);

}

// Check if two lines are intersect (at point P)

bool areIntersect(line l1, line l2, point &p) {

if (areParallel(l1, l2)) return false;

p.x = (l2.b \* l1.c - l1.b \* l2.c) / (l2.a \* l1.b - l1.a \* l2.b);

if (fabs(l1.b) > EPS) p.y = -(l1.a \* p.x + l1.c);

else p.y = -(l2.a \* p.x + l2.c); return true;

}

// Convert 2 points to vector A -> B

vec toVec(point a, point b) {

return vec(b.x - a.x, b.y - a.y);

}

// Scale a vector

vec scale(vec v, double s) {

return vec(v.x \* s, v.y \* s);

}

// Translate P according to v

point translate(point p, vec v) {

return point(p.x + v.x, p.y + v.y);

}

// Dot product of two vectors

double dot(vec a, vec b) {

return a.x \* b.x + a.y \* b.y;

}

// Cross product of two vectors

double cross(vec a, vec b) {

return a.x \* b.y - a.y \* b.x;

}

double norm\_sq(vec v) {

return v.x \* v.x + v.y \* v.y;

}

// Get the minimum distance of point P and line AB

// Line PC is the minimum distance

double distToLine(point p, point a, point b, point &c) {

vec ap = toVec(a, p), ab = toVec(a,b);

double u = dot(ap, ab) / norm\_sq(ab);

c = translate(a, scale(ab, u));

return dist(p,c);

}

// Get the minimum distance of point P and line segment AB

// Line PC is the minimum distance

double distToLineSegment(point p, point a, point b, point &c) {

vec ap = toVec(a, p), ab = toVec(a,b);

double u = dot(ap, ab) / norm\_sq(ab);

if (u < 0.0) {

c = point(a.x, a.y);

return dist(p,a);

}

if (u > 1.0) {

c = point(b.x, b.y);

return dist(p, b);

}

return distToLine(p, a, b, c);

}

// Returns angle AOB in RADIANS

double angle(point a, point o, point b) {

vec oa = toVec(o, a), ob = toVec(o, b);

return acos(dot(oa,ob) / sqrt(norm\_sq(oa) \* norm\_sq(ob)));

}

// Heron's Formula : Find the area of triangle double

heronsFormula(double a, double b, double c) {

double s = perimeter(a, b, c) \* 0.5;

return sqrt(s \* (s - a) \* (s - b) \* (s - c));

}

// Find the radius incircle of triangle ABC (lengths)

double rInCircle(double ab, double bc, double ca) {

return heronsFormula(ab,bc,ca) / (0.5 \* perimeter(ab, bc, ca));

}

// Find the radius incircle of triangle ABC (points)

double rInCircle(point a, point b, point c) {

return rInCircle(dist(a, b), dist(b, c), dist (c, a));

}

// Returns 1 if there is an incircle center, return 0 otherwise

// ctr will be the incircle center

// r is the same as rInCircle

int inCircle(point p1, point p2, point p3, point &ctr, double &r) {

r = rInCircle(p1, p2, p3);

if (fabs(r) < EPS) return 0;

line l1, l2;

double ratio = dist(p1, p2) / dist(p1, p3);

point p = translate(p2, scale(toVec(p2, p3), ratio / (1 + ratio)));

pointsToLine(p1, p, l1);

ratio = dist(p2, p1) / dist(p2 , p3);

p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)));

pointsToLine(p2, p, l2);

areIntersect(l1, l2, ctr);

return 1;

}

// Find the radius circumcircle of triangle ABC (lengths)

double rCircumCircle(double ab, double bc, double ca) {

return ab \* bc \* ca / (4.0 \* heronsFormula(ab, bc, ca));

}

// Find the radius circumcircle of triangle ABC (points)

double rCircumCircle(point a, point b, point c) {

return rCircumCircle(dist(a, b), dist(b, c), dist(c , a));

}

A screenshot of a social media post

Description generated with very high confidence

// Line segment PQ intersect with line AB at this point

point lineIntersectSeg(point p, point q, point A, point B) {

double a = B.y - A.y;

double b = A.x - B.x;

double c = B.x \* A.y - A.x \* B.y;

double u = fabs(a \* p.x + b \* p.y + c);

double v = fabs(a \* q.x + b \* q.y + c);

return point((p.x \* v + q.x \* u) / (u + v),

(p.y \* v + q.y \* u) / (u + v));

}

// Cuts polygon Q along the line AB

vector<point> cutPolygon(point a, point b, const vector<point> &Q) {

vector<point> P;

for (int i = 0; i < (int)Q.size(); i++) {

double left1 = cross(toVec(a,b), toVec(a, Q[i])), left2 = 0;

if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q[i+1]));

// Q[i] is on the left of AB

// edge(Q[i], Q[i+1]) crosses line AB

if (left1 > -EPS) P.push\_back(Q[i]);

if (left1 \* left2 < -EPS)

P.push\_back(lineIntersectSeg(Q[i], Q[i+1], a, b));

}

if (!P.empty() && !(P.back() == P.front()))

P.push\_back(P.front());

return P;

}

//-- Line Segment Intersection

int pyt(PII a, PII b){

int dx=a.x-b.x;

int dy=a.y-b.y;

return (dx\*dx + dy\*dy);

}

int det(PII a, PII b, PII c){

return ((a.x\*b.y)+(b.x\*c.y)+(c.x\*a.y)

-(a.x\*c.y)-(b.x\*a.y)-(c.x\*b.y));

}

bool insec(pair<PII,PII> t1, pair<PII,PII> t2){

bool hsl;

h1=det(t1.F,t1.S, t2.F);

h2=det(t1.F,t1.S, t2.S);

h3=det(t2.F,t2.S, t1.F);

h4=det(t2.F,t2.S, t1.S);

hsl=false;

if ((h1\*h2<=0) && (h3\*h4<=0) && !((h1==0) && (h2==0) && (h3==0) && (h4==0))){

hsl=true;

}

return hasil;

}

...

//sg1 dan sg2 adalah pair<PII,PII>

if (insec(sg1,sg2)){

le=sqrt((double)pyt(sg2.x, sg2.y));

r1=fabs(crosp(MP(sg2.x, sg1.x),sg2)/le);

r2=fabs(crosp(MP(sg2.x, sg1.y),sg2)/le);

r2=r1+r2;

dix=sg1.x.x + (r1/r2)\*(sg1.y.x - sg1.x.x);

diy=sg1.x.y + (r1/r2)\*(sg1.y.y - sg1.x.y);

//intersect here

return MP(dix,diy);

}

// returns the area, which is half the determinant

// works for both convex and concave polygons

double area(vector<point> P) {

double result = 0.0, x1, y1, x2, y2;

for (int i = 0; i < P.size() - 1; i++) {

x1 = P[i].x;

x2 = P[i + 1].x;

y1 = P[i].y;

y2 = P[i + 1].y;

result += (x1 \* y2 - x2 \* y1);

}

return fabs(result) / 2.0;

}

// returns true if point p is in either convex/concave polygon P

bool inPolygon(point p, const vector<point> &P) {

if ((int) P.size() == 0) return false;

double sum = 0; // assume first vertex = last vertex

for (int i = 0; i < (int) P.size() - 1; i++) {

if (ccw(p, P[i], P[i + 1]))

sum += angle(P[i], p, P[i + 1]); // left turn/ccw

else

sum -= angle(P[i], p, P[i + 1]);

} // right turn/cw

return fabs(fabs(sum) - 2 \* PI) < EPS;

}

// check if point p inside (CONVEX/CONCAVE) polygon vp

int inPolygon(point p, const vector< point >& vp) {

int wn = 0, n = (int)vp.size() - 1;

for(int i = 0; i<n; i++) {

long long cs = cross(vp[i+1], vp[i], p);

if(cs == 0 && 1LL \* (vp[i].x - p.x) \* (vp[i+1].x - p.x ) <= 0 && 1LL \* (vp[i].y - p.y) \* (vp[i+1].y - p.y ) <= 0)

return 1;

if(vp[i].y <= p.y) {

if(vp[i+1].y > p.y && cs > 0)

wn++;

}

else {

if(vp[i+1].y <= p.y && cs < 0)

wn--;

}

}

return wn;

}

PT ComputeCentroid(const vector<PT> &p) {

PT c(0,0);

double scale = 6.0 \* ComputeSignedArea(p);

for (int i = 0; i < p.size(); i++){

int j = (i+1) % p.size();

c = c + (p[i]+p[j])\*(p[i].x\*p[j].y - p[j].x\*p[i].y);

}

return c / scale;

} // compute distance between point (x,y,z) and plane ax+by+cz=d

double DistancePointPlane(double x, double y, double z,

double a, double b, double c, double d)

{

return fabs(a\*x+b\*y+c\*z-d)/sqrt(a\*a+b\*b+c\*c);

}

//circle-circle intersect

for(int i = 1; i < n; i++) {

for(int j = i + 1; j <= n; j++) {

double d = dist(P[i], P[j]);

double r0 = P[i].r, x0 = P[i].x, y0 = P[i].y;

double r1 = P[j].r, x1 = P[j].x, y1 = P[j].y;

point center;

if (d > r0 + r1) continue;

if (d < fabs(r0 - r1) || fabs(d) < EPS) {

if (r0 < r1) center = P[i];

else center = P[j];

} else {

double a = (r0\*r0 - r1\*r1 + d\*d)/(2\*d);

double h = sqrt(r0\*r0 - a\*a);

double x2 = x0 + a\*(x1 - x0)/d;

double y2 = y0 + a\*(y1 - y0)/d;

double translationY = h\*(y1 - y0)/d;

double translationX = h\*(x1 - x0)/d;

center.x = x2 + translationY;

center.y = y2 - translationX;

ans = max(ans, go(center));

center.x = x2 - translationY;

center.y = y2 + translationX;

}

ans = max(ans, go(center));

}

}

// line segment with circle intersect

private int FindLineCircleIntersections(

float cx, float cy, float radius, // (x-cx)^2+(y-cy)^2=r^2

PointF point1, PointF point2, // segment from point1 to point 2

out PointF intersection1, out PointF intersection2)

{

float dx, dy, A, B, C, det, t;

dx = point2.X - point1.X;

dy = point2.Y - point1.Y;

A = dx \* dx + dy \* dy;

B = 2 \* (dx \* (point1.X - cx) + dy \* (point1.Y - cy));

C = (point1.X - cx) \* (point1.X - cx) +

(point1.Y - cy) \* (point1.Y - cy) -

radius \* radius;

det = B \* B - 4 \* A \* C;

if ((A <= 0.0000001) || (det < 0)) {

// No real solutions.

intersection1 = new PointF(float.NaN, float.NaN);

intersection2 = new PointF(float.NaN, float.NaN);

return 0;

} else if (det == 0) {

// One solution.

t = -B / (2 \* A);

intersection1 =

new PointF(point1.X + t \* dx, point1.Y + t \* dy);

intersection2 = new PointF(float.NaN, float.NaN);

return 1;

} else {

// Two solutions.

t = (float)((-B + Math.Sqrt(det)) / (2 \* A));

intersection1 =

new PointF(point1.X + t \* dx, point1.Y + t \* dy);

t = (float)((-B - Math.Sqrt(det)) / (2 \* A));

intersection2 =

new PointF(point1.X + t \* dx, point1.Y + t \* dy);

return 2;

}

}

point perpotongan(point p1, point p2, point p3, point p4) {

point has; has.x=-1; has.y=-1;

double d1=det(p1.x,p1.y,p2.x,p2.y);[

double d2=det(p1.x,1,p2.x,1);

double d3=det(p3.x,p3.y,p4.x,p4.y);

double d4=det(p3.x,1,p4.x,1);

double d5=det(p1.x,1,p2.x,1);

double d6=det(p1.y,1,p2.y,1);

double d7=det(p3.x,1,p4.x,1);

double d8=det(p3.y,1,p4.y,1);

double top=det(d1,d2,d3,d4);

double bot=det(d5,d6,d7,d8);

if (bot==0) return(has); //if line p1,p2 sejajar p3,p4 bot=0

has.x=top/bot;

d2=det(p1.y,1,p2.y,1);

d4=det(p3.y,1,p4.y,1);

top=det(d1,d2,d3,d4);

has.y=top/bot;

return(has);

}

1. Rotating Calipher

// Cited from Stanford Team Notebook

inline pii FindAllPodal\_And\_GetMaxPair() {

LL ans = 0;

int P = 0, Q = 0;

int sizeU = U.size();

int sizeL = L.size();

int i = 0;

int j = sizeL-1;

while(i < sizeU-1 || j > 0) {

LL tmpdist = dist(U[i], L[j]);

if (tmpdist > ans) {

P = i, Q = j;

ans = tmpdist;

}

if (i == sizeU-1) j--;

else if (j == 0) i++;

else if ((U[i+1].y - U[i].y)\*(L[j].x - L[j-1].x) > (U[i+1].x - U[i].x)\*(L[j].y - L[j-1].y)) i++;

else j--;

}

return make\_pair(U[P].pos, L[Q].pos);

}

1. Great Circle Distance

double dist3d(double lat1, double lon1, double lat2, double lon2){

double dlat = lat2 - lat1;

double dlon = lon2 - lon1;

double a = pow(sin(dlat/2),2) + cos(lat1) \* cos(lat2)\* pow(sin(dlon/2),2);

return (R\*2\*atan2(sqrt(a), sqrt(1-a)));

}

1. Combinatoric

  
. . .. . . .   
. 

  
where a, b, and c are non-negative integers

1. Next Permutation

bool nextPermutation(int x[], int n) {

int k = -1;

for (int i = n - 2; k == -1 && i >= 0; --i)

if (x[i] < x[i + 1]) k = i;

if (k == -1) return false;

int l = -1;

for (int i = n - 1; l == -1 && i > k; --i)

if (x[k] < x[i]) l = i;

swap(x[k], x[l]);

reverse(x + k + 1, x + n);

return true;

}

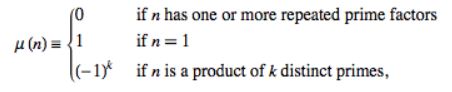
1. DP Optimization

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CH 1 | dpi=minj<i{dpj+bj\*ai} | b[j]≥b[j+1] or a[i]≤a[i+1] | n2 | n |
| CH 2 | dpi,j=mink<j{dpi-1,k+bk\*aj} | b[k]≥b[k+1] or a[j]≤a[j+1] | kn2 | kn |
| D&C | dpi,j=mink<j{dpi-1,k+Ck,j} | A[i][j]≤A[i][j+1] | kn2 | knlgn |
| Knuth | dpi,j=mini<k<j{dpi,k+dpk,j}+Ci,j | A[i,j-1]≤A[i,j]≤A[i+1,j] | n3 | n2 |

* *A*[*i*][*j*] — the smallest k that gives optimal answer, for example in *dp*[*i*][*j*] = *dp*[*i* - 1][*k*] + *C*[*k*][*j*]
* *C*[*i*][*j*] — some given cost function
* We can generalize a bit in the following way: *dp*[*i*] = *minj*<*i*{*F*[*j*] + *b*[*j*] \* *a*[*i*]}, where *F*[*j*] is computed from *dp*[*j*] in constant time.
* It is claimed (in the references) that **Knuth Optimization** is applicable if *C*[*i*][*j*] satisfies the following 2 conditions:
* **quadrangle inequality**:  http://codeforces.com/predownloaded/8c/b3/8cb3a915043ebfe9d8f715de2bca16160d20f1ed.png
* **monotonicity**: http://codeforces.com/predownloaded/20/17/20178ffeb11422f591b70a9bad1185fbf9b8f67d.png
* It is claimed that the recurrence *dp*[*j*] = *mini*<*j*{*dp*[*i*] + *C*[*i*][*j*]} can be solved in *O*(*nlogn*) (and even *O*(*n*)) if*C*[*i*][*j*] satisfies **quadrangle inequality**.

1. Prime Numbers + Mobius

1e9+7,9 1e4+7,9, MAX\_INT, 2243431237, 12433237, 1243537, 9997777, 1234567891, 109026347, 14951, 16249



![A screenshot of a cell phone

Description generated with very high confidence](data:image/jpeg;base64,/9j/4AAQSkZJRgABAQEAeAB4AAD/4RDuRXhpZgAATU0AKgAAAAgABAE7AAIAAAAMAAAISodpAAQAAAABAAAIVpydAAEAAAAYAAAQzuocAAcAAAgMAAAAPgAAAAAc6gAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAHl1c3Vmc2hvbGVoAAAFkAMAAgAAABQAABCkkAQAAgAAABQAABC4kpEAAgAAAAMyMQAAkpIAAgAAAAMyMQAA6hwABwAACAwAAAiYAAAAABzqAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMjAxNzoxMjoxOSAyMDozNzo1NAAyMDE3OjEyOjE5IDIwOjM3OjU0AAAAeQB1AHMAdQBmAHMAaABvAGwAZQBoAAAA/+ELHmh0dHA6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8APD94cGFja2V0IGJlZ2luPSfvu78nIGlkPSdXNU0wTXBDZWhpSHpyZVN6TlRjemtjOWQnPz4NCjx4OnhtcG1ldGEgeG1sbnM6eD0iYWRvYmU6bnM6bWV0YS8iPjxyZGY6UkRGIHhtbG5zOnJkZj0iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyI+PHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9InV1aWQ6ZmFmNWJkZDUtYmEzZC0xMWRhLWFkMzEtZDMzZDc1MTgyZjFiIiB4bWxuczpkYz0iaHR0cDovL3B1cmwub3JnL2RjL2VsZW1lbnRzLzEuMS8iLz48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOnhtcD0iaHR0cDovL25zLmFkb2JlLmNvbS94YXAvMS4wLyI+PHhtcDpDcmVhdGVEYXRlPjIwMTctMTItMTlUMjA6Mzc6NTQuMjA4PC94bXA6Q3JlYXRlRGF0ZT48L3JkZjpEZXNjcmlwdGlvbj48cmRmOkRlc2NyaXB0aW9uIHJkZjphYm91dD0idXVpZDpmYWY1YmRkNS1iYTNkLTExZGEtYWQzMS1kMzNkNzUxODJmMWIiIHhtbG5zOmRjPSJodHRwOi8vcHVybC5vcmcvZGMvZWxlbWVudHMvMS4xLyI+PGRjOmNyZWF0b3I+PHJkZjpTZXEgeG1sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZi1zeW50YXgtbnMjIj48cmRmOmxpPnl1c3Vmc2hvbGVoPC9yZGY6bGk+PC9yZGY6U2VxPg0KCQkJPC9kYzpjcmVhdG9yPjwvcmRmOkRlc2NyaXB0aW9uPjwvcmRmOlJERj48L3g6eG1wbWV0YT4NCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgCiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIAogICAgICAgICAgICAgICAgICAgICAgICAgICAgPD94cGFja2V0IGVuZD0ndyc/Pv/bAEMABwUFBgUEBwYFBggHBwgKEQsKCQkKFQ8QDBEYFRoZGBUYFxseJyEbHSUdFxgiLiIlKCkrLCsaIC8zLyoyJyorKv/bAEMBBwgICgkKFAsLFCocGBwqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKioqKv/AABEIAL8CSQMBIgACEQEDEQH/xAAfAAABBQEBAQEBAQAAAAAAAAAAAQIDBAUGBwgJCgv/xAC1EAACAQMDAgQDBQUEBAAAAX0BAgMABBEFEiExQQYTUWEHInEUMoGRoQgjQrHBFVLR8CQzYnKCCQoWFxgZGiUmJygpKjQ1Njc4OTpDREVGR0hJSlNUVVZXWFlaY2RlZmdoaWpzdHV2d3h5eoOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4eLj5OXm5+jp6vHy8/T19vf4+fr/xAAfAQADAQEBAQEBAQEBAAAAAAAAAQIDBAUGBwgJCgv/xAC1EQACAQIEBAMEBwUEBAABAncAAQIDEQQFITEGEkFRB2FxEyIygQgUQpGhscEJIzNS8BVictEKFiQ04SXxFxgZGiYnKCkqNTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXqCg4SFhoeIiYqSk5SVlpeYmZqio6Slpqeoqaqys7S1tre4ubrCw8TFxsfIycrS09TV1tfY2dri4+Tl5ufo6ery8/T19vf4+fr/2gAMAwEAAhEDEQA/APojUZru302eXTbRb26RcxW7SiISH03kHH1xXCxePfGE3iC40WPwLbm9t4EuJE/txcBHJAOfKx1U16HXE6d/yW7W/wDsDWv/AKMkoXxJev5Nj+y3/W6RZ0/x/YtqWp2XiFrPRJNPkhhJub5MSu8QcqCQvTOOM568V0V7qun6bYi81G/tbS1OMTzzKic9PmJxzXA6Loumaj46+IM1/YW9zJ5kMO6aMOQhtwSBnpn/AAritGW6vP8AhA0u9UsrC3/saVbObUrQXERnEmCoBdQH2BcHPQEDrQne3y/FN/p+INWv8/zX+Z7xZ3trqFql1YXMN1byDKTQyB0YexHBqtqWu6RozRrq+q2VgZf9WLq4SLf9NxGa5T4b2FtY3fiD7Fr1rqyyXaGZLGwNtBBLs+YJ8zKxI2k7TwevJrPszoQ+IvjJvF/2HzwkHkfb9uPsflc7N38O7dux360P9L/1/W2ol/X9f1rodjH4hWXxlJoSwAomnJfC6EmQwaRk24x/s5znv0qax8SaHqd41nputafeXKAloLe6SR1x1yoOa8sXU9BtJL+/ttMvBoS+EEMdlJIyytCbiQBd24kAg9c8A1Bc3Wzxd4HCW3hmxX7agt7fSZzPOsRjYHc4VRtwQOhye9C3Sf8AWrX6A9Lv+tk/1PXE8Q6NJqx0uPV7BtQU4NotyhlB/wBzOf0om8Q6Nbaoum3Gr2EV+5AW1kuUWVs9MITk5+leOqbHQ9Ns7m1m0LxLoi6oskK7XtdUjmabjBHLspPIYAkDnipfEEdjp0PiW9tbrw/rulm+kn1DT9RV7a9ilBGY45R8xOR8uR7A0Lpf+tv8/wCrja1sv63/AMv6ses2Wstc61q9lNDBDFpxixMLtHZw6biXQcxY/wBrqORxU2m67pGsmQaPqtlfmL/WC1uEl2fXaTivIPEUzXDeOZjHLHZyXWjveoc7ltzGhkB/Dr7ZrvrKbwcnjbTodDtLeTU2sZCk+nbfLitwV4k2EDBONuQehximl3/rS/8AXzJvpf8Arf8Ar8DqVvbV757JbmFrqNBI8AkHmKhOAxXqASDz7VDJrWlw21zcS6lZpDaOY7iVp1Cwvx8rnOFPI4PrXIePL2Pwl4g0rxjID9mijlsL7b3jdS8Z/CRQP+B1xmo6PeWEHgePVL20sBfT3N7eXF9bCeAXsqh1DqWUZwWVSTwRUrX8vn0/D8bIp6f1/XX8D2CLXdIm0z+0odVspLHcF+1JcIYskgAb84zkgdepqBvFnh1LE3j6/pa2okMJnN5GEDjqu7ON3t1rx/xJpdtbeHPGHl+IbHVZJhYreWun6cbeGN/tC4YsHZS5BIIBzwM11/i/R9OtvFGkf2LfaPpmqW9rL9msNTs82twhKhsEYCuNqjIJbHbFPz/ra4v6/r+vzOuvvECQzaP/AGeLW9t9TuPKE4vY0CrtLbkB/wBaePurz37VZk8QaNDqq6ZLq1imoMQFtGuUEpz0wmc/pXkum3lvdHwrFaada2H2XxRPFILGZpbaWTyXLPEW6ISeg4HNMt/+Ea/4Ufqv9sfY/wC3f9J+0+bt+1fbt7bf9rdnbjHb2ovZN/1sv8/y76O12l/W7/yPW77xJoemSSJqWs6faPEyrItxdJGULDKg5PBI5HrWhHIk0SywuskbgMrqchgehB715j4O0iDUvHviOTxBYwXV8mn6fHKbiIOQWg/eDB9SOfpW78JGJ+F+lAkkIZkX2UTOAPwAxTtbQm99f62udEfEOijVv7LOr2A1DOPsn2lPNz1+5nP6VnaN410nW/EGr6Ta3Vv52luFbFwjGQbQWYKDnapO0nseOK8uvvsGjaZd3tpPoXiPRV1Rp3t51e11OOczZ2o4+ZmDdMgEgY6VsXl7o2lH4kTarYpdRieHzLRJPKeRHhjGCw5VSx5PTrUp6X8v8v8AMq2tvP8AzPR7PxRoGoSTJYa5pt08CF5Vhu43MajqWweAPU1R8NeN9H8S6FPqlvd20MNtJIswa5Q+UquyqzkHChgu4Z7GvOXuD/wszwZG9v4ZttzyolvpEpmkEJhYYkbao2+gxyc4rMmWI+AtAiS5srOzHiO4XUJZ4FmiiPnTeV5yblyucfeIA4NP/hvxS/Un+vwf+R7dYazpeq2j3Wl6lZ3tvGSHmt51kRccnJBIFN03XtI1lpBo+q2V+Yv9YLW4SXZ9dpOK8ovtG0pbzxB/avjSxPnaP5d8ulaQY0jQyDZLIUdwSDxg8lSe3NaXhm4+wfELSrS+g8P6jPcWUsdpqWhs0TJCoVj50QJXBwMHOAelNav+vP8Ay/qwPRf15f5/1c7TxH4z0vwxqWlWepXEET6jOYw0s6xiJQpO9snpkBfqRVbS/H2k3dxqEOqXVlpUlrqMtjCtxeIDcbNvzKGx13dBn61Q8eraDxL4NlvVhEf9qMrPKBjBgkwCT74rm9P0TS7vwx8Srq5sLeadtQvlMskYZsLGCoBPTBORjvUc1k2+if4cv+ZdrtJdbf8At3+R6dqet6VosaSaxqdnp6SHCNdXCRBj7FiM1ka9470TQYdJmnv7SSHVLlYYpftSKgQglpd2cFRgDI7ketcJZ6vb3FxpEMOm6PPq8Ph22km1DXb0xxrEy9ETa245BLMMehNY3h57SXwzo0lx9gNtH42kRXhXbAEKucJu6ISeB9Ku3vcvn/7db+v6ZF/dv5f+23PZfEGrto3hu61S2hhumgjDpHLdJbo+SBzK/wAqjnqawZfiFA/xAh8LabFY3MoRWuZZNRSMxkk5REwTI4AJK8YFHxZx/wAKl13bjb9nGMf761T0e2g/4XVqz+THuGjWrg7BkMXkyfr70L40v62b/Qb+G/8AW6X6nc3V1b2Nq9zezxW8EYy8srhFUepJ4FRadqmn6va/adJvra+g3FfNtplkXI7ZUkZriPiwsMcfhy8uLq0VbbVUYWl/J5dvcttPDvghNoBIJBHsTxV7wt4VB8Tah4v1f+zbnUNQCLb/AGICWO2jVSvySkAsWB5IAz0xSjrf+u3+YPQ6bU9Z0vRYUl1jUrPT43OFe6nWIMfQFiM0h1zSRZwXZ1SyFtckiCb7QmyUgE/K2cHgE8ehrifFWpWp+IS2drpWkTatBpvmG91q8MUMcTOflRNrbjkEkgDjAJrgtEitdV0vSrS5jsbmzbxvIvl2qH7MymMnCKc/JnoPSiPvf15pBLT+vJs9007WdM1iB5tI1K0v4ozteS1nWVVPoSpOKx7nxxpf9q6VZaTc2mqfb7w2kr212r/ZyI3fJC55+TGDivP/ABvaNY+IPGVvokAt1m8OwSzR2qBd4EzB2wO/l7h9K0tSPhL/AITDwCPDR0/zjcOYxZ7cmDyH+9t7ZxjPfPvRHVr+urX6f1YHpf0f5J/r/Vz0XV9Z07QdOe/1m9hsrZOskzhQTjOBnqTjgDk1ynhDWNe8W/YvElvqVqukXE0qvpyhG8uIAiPLAFvNzgsNwABxtzVr4o3ksHgC8tLQ/wCl6o8enwAdS0zBD/46WP4V0Ijt9G8P+WFb7NZWu3bH97YidBjvgUr2Tk/67/oO12or+u36/cRJ4l0KTVP7Mj1rTnv9237It2hlz6bM5z+FPPiDRhqC2B1exF40hiW3+0p5hcAErtznIBBx15FeG61qNtJ4G0W40zTvDWlWL3lvLZQx3bXF/nzVOc4GG67slj1B5rrbLTy6/E29061jfVlupVtpggMisLcbQp6g5Jxjuab0Tb6X/C3+YLVpLrb8b/5Hotvr+j3epSada6tYz30ed9rHco0q465QHIq9LLHDE0szrHGg3M7nAUepNeHaVpVhP4a8MTL4w0S2UT27WcdpooN0JsjMeVlLZJyHJHqTXpvxAvbCy8JyHVbEX8Es8UQgecwxsxcbd79lB5OeK0jTcpKPnb8v63I5rK/lc17DXtI1TzP7M1WxvPKGZPs9ykmwepwTis6/8X6dFBGdJvNO1KZpYlaJdRijKo7Ab8k89QQP4uAOtcIlxJ/ws+wSRNDhlOm3SvFpDl9qbAVEjYGeRwMetRQWNpbfBbwzNb20Ucst7ZtJIqAM5M4JyeprujhYJxb6uP4t/wCX4nO60rO3RS/BJ/qdZffErTbez1+SzEN3NozqvkrdKGnB2gsMZIUF8ZweRjiums9a0vULiS3sdRtLm4hGZYYZ1d4/94A5H415b4iS1XQ/iNEqwrN9rhwoADBCsP44z+tdLqmmWOk+PPBo0y0htAVuoT5KBdyCHIU46jPNROjT5FbR2/KCkP2k1J+Tf/pVjqYvEWiXFzFbwaxp8s8y7o4kukLOPUAHJHB/Kp7DV9N1QSHTNQtbwRHbIbeZZNh9DgnFePw6NpzfBbQ5jZQedcapF5suwb3zclTluv3ePpWj4/sv7P1bW00K3W2aTw4DItsm3Ki4UMcDHRNw+lU8LDm5E3u191v8w9rK17dE/vdj0u117SL6SaOx1WxuXgBaZYblHMYHUsAePxrBsPH9nq1lHcaUtrOTqQspI2v40ZELlRJz1ztJVRyw6VzNjpdi2teG5o/FOjyMJM2kGnaQEeaPYdyEpI21CuclhgGqVuLZfDenJCIhKvjECRUxuA+0PtB/DpTWHpp99u6628iXVnyt+v5NnqVzrmk2d8lleapZW91JjZBLcIsjZ6YUnJo1rV7TQdGudS1CVI4beMudzhdxxwoz3J4A7k151pb+GIvDnieTxhFazXi6hcG+jl2+e43fu1TJDcrt24I56V2XiVYpvhxqZMDqn9lyMsc4y6fuiQDnPzD6nmuadKMGlr/XY3hJydvX/ITwl4sj8TadHcSixtpplMkVtBfrcOUAXJOAMEFgCOcHHrWs+r6bFv8AM1C1TZMIG3TqNspGQh54bBBx1rgdFns9L8SeEHuGitYrjw4yK7EIrSfuWIz64BNYt5JZa3DelTHdWVz40gQ91kXy0B+oP6iuh4aMqmmi/wDtrGCrSjTu9X/9rzf8A9Y0/V9N1ZZG0rULW9WM4c206ybT6HaTioV8SaHJJPHHrOnvJbKXmRbpCYgOpYZ4A964HxbaPYeK9Yj8P26W003heVttugXcyyAA4HcDIFaFg3ggWXheOzs7a6uZdq2YswvmIfLO9pMEHaBndnPJ5BNZOhHk51fX/g7/AHf1Y09o1LlfT/gf5mrpPjq21y00i402O2kW/uHhmRr6NXtsBiPl6uxAB2jkA56Vuya5pMOpLp8uqWSXrEBbZrhBIc9MLnNeXaAYP7H+Hn2fy94v5hJsxkNsk647/Wkt/wDhHf8AhT+qf2v9k/tnNx9p83b9p+1722f7W7O3GO1b1MNBSdr2vbz3a/T8iY1JWV+3yPULzxBo2nO6ahq1javGVV1nuUQqWGQDk8ZAyKfd6zpmn2kd1f6jaWtvLjy5pp1RHzyMMTg1wvhLS4dQ8aa5Jr1nDc3qadYJIZ4wxDNCd4wfUjn6VgaHbQz6H4akt9V0+21S2ivEtrXVYDJbzR+eQQG42sMDoc47YrP6tC9r7Wv803+n9WBVpNXt/Wn+f9XPYre5gu7dJ7SaOeGQZSSJgysPUEcGqVx4h0az1JNOu9XsIL6TGy1luUWVs9MKTk5rC+HN3BcaDeR29hbWX2e/likFnM0lvK/BZ4ieiknoOBg1wXiOGxsf+EpvbS78P63pzXckup6dqiPb3cUgAykUo+Y9Bt4x6GuOrH2dRxf9bG8Hzxuetan4h0bRZI01jV7DT3kGUW6uUiLfTcRms3WfHGj6LrWj6ddXlqG1UsUle5RVRApKscnkMcKPU1w9zrdjqOtazLp+j6DaTR21ut/d+I7tt21ogwVYsE7QDg8qCc9+azPB72smkfDKa9EGzzb6MPIABgCTaPm/QVKXfyG3pf8ArY9RsPFFtJpd5f6zJY6XBbXklt5j38UkZCtgEuDhSf7h5HQ1T8T+P9H8O+EjrkF3aahHIwjtUiu0C3DlgMB+RgZyTzgA153aWlpdaenmavZ6dfxeKNQksV1C3MttO+SCj9ADg8HIPpk1Bqc8Fx4L8SWkul6ZBeWur2TXNxpUrSWs8jyR5KBvuNgAMB+NTH3kvl+Nv8/+AU9G/n+F/wDI9d8N69F4g0pblZbBp1OJorG+W6SInkDeoGcrg9B1qCLxVaTa5LbxTWL6dHYG8bUF1CIgYkKMCmchRtOX+7kEdRXF+NLubwR4mvbnSYgn/CR6cLW2SNcAXqMEj4HTKyf+OVnatpEWgalrGlW/KWfgJ4s/3iHfJ/E5P40pS05l5/gn/kvvQKOtvT81/n+DPUrXxDot7qBsbLV7C4vAu428VyjyAYznaDnGKS58SaHZXr2d5rOn291GhkeCW6RXVQMlipOQMc5rzm70nT9L8N/Da506ygtrj+0bJTLHGA5EkLFwT1O49fWsO4/4Rk/A/Xv7d+x/295tz9q88r9p+1+Y2zr82emP9nPbNVP3ebyv+n+f9XFD3redvxv/AJHuaOskavGwdGAKspyCPUGlqlo3/IBsP+vaP/0EVdqpLlk0TF3imFQLZWqXz3qW0K3UiCN5xGA7KDkKW6kDJ496noqSiCKxtIJriaG1hjluiDO6RgNMQMAsf4uOOe1VpdA0efSk0yfSbGTT4/uWj2yGJfohGB19K0KKAILKxtNNtEtdOtYbS3T7kMEYRF+gHAqvqOhaTrDxNq2l2V80JzGbm3SQp9NwOKv0UAV20+ya4edrOBpnh8h5DEu5o852E4yVyTx05qla+FvD9kFFloWm24WVZl8qzjTEg6OMD7wycHrzWrRQBmr4d0RNVOqJo+nrqBJY3YtUEufXfjP60T+HdEutTXUbrR9PmvkIK3UlqjSqR0w5GePrWlRQBXSws457mZLSBZbvH2hxGA02Bgbz/Fgcc9qh03RNK0YSDR9Ms7ASnMgtbdIt59TtAzV6igCC8sbTUbVrbULWG6t3ILRTxh0bByMg8cEA0XljaajaPa6haw3Vu/3oZ4w6N9VPBqeigDPi8P6NDpbabDpFjHYMwZrVLZBESCCDsxjOQD07VJqWj6ZrMCw6xp1pfxKdyx3UCyqD6gMDVyigCmmj6bHFaRJp1osdk261QQKBAcEZQY+U4JHHrUcnh/RpdVXU5dJsX1BcFbtrZDKPo+M/rWhRQBBHZWsN3PdQ20MdxcbRNMsYDy7Rhdx6nA6Z6UtnZWun2q21hbQ2sCZKxQxhFXJycAcdSTU1FAGafDmiNqw1RtH086gDuF2bVPNz678Z/WpJdE0qa+kvZtMs5LqaIwyTvboZHjP8BbGSvt0q9RQBl2nhjQbDy/sOiabbeVJ50fk2kabHxjcMDhsEjPXmpU0HR0F4E0qxUXxzd4tkH2g/7fHzdT1z1q/RQBRsNE0nSrWS20vTLOygl/1kVvbpGr9uQoANN03QNH0Z5H0fSbGwaX/WNa2yRF/rtAzWhRQBV1HS9P1e2+zatY219BuDeVcwrIuR0OGBGaI9LsIoLmCKxtkiumZriNYVCzFhhiwx8xI4OetWqKAMy68NaFfJbJe6Lp1ytqoS3Wa0RxCo6BMj5Rx0FOl8P6NPZzWk+kWEltPL500L2yFJH/vMpGC3A5PPFaNFAFe4sLO8sWsru0gntGUI1vLGGjKjoCp4xxSpY2kd695HawpdSII3nWMB2QchS3UgZOB71PRQByWraDrL+KZtTtYdN1eyuLVbf7DqUrRCDBJJRhG4IbIyCuflHPGKseEfCw8OyajceTZ2Z1CRHNjp6kW8G1cfLnGSe5CrnA4456WihaA9ShqWg6RrLRNq+lWV+0P+rN1bpKU+m4HFKuh6SkgdNLsldZ/tAYW6AiXG3zOn3scbuuKvUUAQCytRfNei2hF08YiacRjzGQHIUt1xk5xVK28MaDZXIuLPRNNt5xJ5olitI1YPgjdkDOcEjPua1KKAOVsdA1m9v7KfxZc2tyumXEs9r9nz+9diwR3G1QuxGwFG7nksa6qiijpYOtzIXwn4cQzlNA0tTcczEWUY83nPzcc8gHnvWhb2VraSTyWttDA9w/mTNHGFMrYxuYjqcADJ9KnooAzoPDui2upvqNro9hDfOSWuo7VFlbPXLgZOfrVy5toLy2e3u4Y54ZBh4pUDKw9CDwaloo2Az7bw/o1n5f2PSLGDyixj8q2Rdm4YbGBxkcH1qX+ydO+wxWX2C1+ywlWig8lfLjKnIKrjAweRirdFVzyfUXKuxQuNC0i7uJbi60qynmmQRyySW6M0iZB2sSMkZA4PoKsyWdtNcQzzW8Uk1vnyZGQFo8jB2nqMjjipqKXM+4WRTGkaaLGOyGn2otYnDxwCBdiMDuBC4wDnnPrUxtLY3f2o28RuPL8rztg37M527uuM84qaijmbCyKFloWkabcvcadpdlaTyDDywW6Izd+SBk0f2DpAuHuBpVl50kqzPJ9nTc0i52uTjJYZOD1Gav0U+eV73DlW1ilPoul3WoR31zptnNeRY8u4kt1aRMdMMRkVamhiuIHhuI0likUo8bqGVlPBBB6in0UrthZblK60XS76zitL3TbO4toQBFBNAromBgYUjAwOOKBo2mKoVdNtAolWYAQLgSKAFfp94AAA9QAKu0U+aXcOVdiH7JbG9F4beL7UI/KE+wbwmc7d3XGecVBaaLpdhdy3VjptnbXE3+smht1R378sBk1dopczWlwsihDoOj28wmt9KsYpRKZg6WyKwkIwXyB97BIz1pZND0mXUl1GXS7J71SCty1uhkH0bGavUU+eW9w5UQx2ltFdTXMVvEk84USyqgDSbeBuPU4zxmqs3h/RrixSyn0ixltI2LJbvbIY1JOSQpGASSfzrQooUmtmFkRW1rb2VslvZwR28EYwkUSBVUegA4FUrnw7ol7qSaheaPp9xeoQUuZbVGkXHTDEZGK0qKm93cfSxn3fh/Rr/UI76+0mxubyLHl3E1sjyJjphiMikl8O6LPZR2c2j2ElrHIZUge1QojkklguMA5JOevNaNFAGfLoOjz2MtlPpVjJazSmaSB7ZCjuTkuVIwWJ79aVNB0iLTBp0WlWKWIYOLVbZBEGByDsxjOQDnHWr9FAHL6p4f1TW/GenXOotYjRNLl+1W0aBjPJPsKgvkbQq5JGDk8VvT6Xp9zNLNc2NtNLNAbaV5IVZniJyYySOV5PynirVFHSwdblZ9NsZIbaKSyt3jtGV7dGiUiFlGFKDHykDgEdK891jwF4h1qS8tL5fDkqXjNG+ttabb9bdj9zAQLuC/Lu3dO2a9Loo63AZDClvbxwxDCRqEUegAwKfRRQ3fUSVlYKKKKBhRRRQAUVDeXtrp9q9zf3MNrbxjLyzSBEX6k8CsLVPHOh2PhHU9estRtNQg0+1a4ZbadXzgHaOCcZYY+ufSgDo6K5XwsE8PeC4tU8UajHHeXireajeXUixqJXA+XJwFVeEA9B6mtdfE2hSX8Fims6e11cIHhgW5QvIpGQVGckEHI9RQBp0Vxfiq6bwt4s0XXopGWz1K7j0vUot3ynfkQzc8Aq+FJ7q/P3RXaUAFFFFAFfUJ7i20y5nsrX7ZcxQs8Vvv2ecwBITcc4yeM+9eV+DPjVqnju9vbLRPCMKXtj/r7W71bypFGcE48o9DwfQ49a9crwD4zeEdQ8EeKrf4q+CVEctvIDqduo+Vs/KXIHVWztb3IPqQAeonXPHA6eC7E/TXB/8ZrG17x/448P2T3dx8M5bu3jBLvZaukxUDvtEe7HfgV1Xgzxdpvjjwra65o75hnGHjYjdC4+8je4/UEHvW7QBj+E/Edt4u8J6drtipSG9hEmwnJjbOGQn1VgR+FbFUNK0Sx0T7YNMh8hLy5a6lQH5fMYAMQO2cZwOMknvXO/E3xvb+CfBOpXsV7aRaqLdjZW80qh5HJCgqhOWAJyfpQB2NFeWadrWp+D/hZq2uXXieHxVPPKo09lkV1SVwsax7wSD+8OSBwOcYrvtB0tfD3h+O3urpp5UUy3d3M/Msh5eQk9BnPsBgdBQBq0V5b4h8V/238TvBun+DfFHnw3E8z6hBZTxyQmKFQ53YBILZA64I7d63vC1xNonjbV/B8zO9nHAmpaWznJSF2KyRZ9EcfL6KwGeBQB2lFFFABRXK/ErxZ/whfw/wBT1iGa3jvI4itotwwAklPQAE/McZOB6GsvwbdxaVpd3r2qePYdY0i6SHa1xNGVtZwn7weYGx8zEEIANvQDuQDvqKyY/FWgS6I2sJren/2ajbXuzcoIkbptLZwDkgYPOadpfifQtbnMGjazYX8yxiYx21ykjBDjDEA5xyOfegDUoqrqOqWGj2TXeq3kFnbKQDLPIEUE9Bk96or4v8ONpMWqf2/pgsJn8uO6a7jWNnxnbuJxu4PHXigDYorIfxb4cjFsX8QaWouxm3JvYx5wztynPzcgjjvxTrvxToNjqi6be6zYW96xAFvLcIr5P3Rgnqew79qANWisW78ZeGLC9ns77xDpdvc267popbyNWiHHLAn5eo6+tSf8JV4e/tCOx/t7TPtkpQR2/wBsj8xy4BXC5ycggj1yKANais/T9f0fVrmW30vVrG9mh/1sVtcpI0fOOQpJHPHNYurarcR/EPRLG28T6VaQMkv2rR5ipurolSUKc7gBgnjHQ9ewB1VFYLeO/CSCct4m0gC3YJMft0eI2OcA88H5W49j6VnePviBp3g3wXJrKXllJNMo+wxvKCtyxIGVwfmADZJHQUAdfRWfb+INGu9Kk1O01axnsIsiS6iuUaJMdcuDgYyKNJ1/SNehkl0XU7W+SFtsht5lfyz6Njp6jPUc0AaFFc/J4+8IxIzv4m0nasgiYi8jOG9Dg1b1HxToGkX0dnqms2FncyEBYp7hUbnpkE8Z6DPU8CgDVorHvvF3hzTL6Wy1HXtNtbqFN8kE12iOi4zkqTkcEfmK1Le5gvLaO4tJo54JF3JLEwZXHqCOCKAJKKyJfFvhyCO5ebXtNRbSTyrgtdoPKfn5W54PB4PofStK1ure9tUubOeO4gkGUlicMrD1BHBoAlorJ1xtfh8q48PJY3Ijz51ndlozKOMbJRkKeCMFSDkcjHNPT/GunXGoR6Zqsc2i6rJwlnqACGU9P3bglJOf7rE8dBQB0VFFed+OfE/inw94u8N29rJpyaRq2sQWXEbNcFWGXBJO0cgjgZwR0oA9EooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAPMPH91Cfix4RtfEU8Vt4dhhuL1muXCQzXKD5FYnglQdwH864a8ji1/wCGnxW8XWUaSQ6pfqkBWPaZILZk+bGB1+Y+vc161q2i+L7jxHfT6brdgNKurZI4ba7tS5s5AD+9UDG85OeSB+Qq54d8F6b4f8CReFebyz8mSKdpQAbjzCTIWx6ljQBk/FK5sIvhdfazIvmrZQG4s1BwrSuhjiJHcAyggeuK4DX9L0/wZ4Y+HPhp0+xRi6XVdQuANsjNbRh2APGXZmCgdegFereFdJvNP8OJoeuRx3cWnsILW4bDC5gTBidl7OBhTnum4Hmt2WCGfZ58SSeW29N6g7W6ZGeh5PPvQB558Xpmvvh3puyCSK4vdV08RQTLiRHMyttIGfmAByAexr0euZ1TQr3XPGmm3N8ETR9Hzcwx78tcXRBVWIHRUUnGeSx9BXTUAebX3hpZNQuH/wCFbxXG6Vj539qqvmcn5sZ4z1xU+jeHhba1aTf8K9jsNkoP2oamknlf7W3POK9CooAKgvUtpLC4TUBGbRomE4lxsKY+bdnjGM5qevNNU1nT/iFrsujnU7aDwxp0wF/IblUOpSqc+Spz/qlI+Y9zgDvQBi/A7wZNoGteItW0e+m/4RS/mK6ZbSrgygN/rv8AdHKqerDkgcV7LWdHrOiwwqkWpWEcaKFVVnQBQOgHPSsnXPiT4O8PWclxqfiLT1CD/VxTrJIx9Aikk9KAOnrzX4xwxaoPCWgsgMup69Au/aCUiTLyEf8AfIro/AHiS78X+GTr1zB9mtb6d3sIiMOtuMKpfk/MSrN9GFZ2n+B9Vf4gHxB4j1s6jb2U9zLpVoIwBbCYKuCcc7UXA68sTx0oAp/GGzSHwLZPBAiWtprNjNOiKAAnnqDx9WFQ/GRpLmDwxpFxN9n0fUtZii1OYnapiALbGboAxGOetdzr+iWniTw9faNqK7ra9gaGTHUZH3h7g4I9wKr6NZ31x4bjsPFkFvdXEY8qZiA8dyFPyybT03YBIPQ564BoA8y0PWdG1H9oi/vrFYbiK2sYNIsfsqqNxbfI8gHGUUIylhkdAK6e5SSX9oiwaE/LB4blM3PZrhQo/NT+VdytnbLOk628QlSPy0kCDcqf3Qew9q5/wvoF1barqniLW0jTVtWZFMSMGFrAgIjiDdzyWY9NzHsAaAOmooooA88+OlzZQ/B7XI7yaCOWWHFusrAM77hwgPU4z07Zrn7+9sF8b/DzWWubNvDCW8yG4jZDbxXpjAUuw4DHoCehBr164tbe7jCXcEU6A5CyoGAP4037DaC0NqLWH7OesPljYe/TpQB45cXtlN46+IepxXFofDkmjRwzXJkX7PLehTtAJ4LgHHGecd63fghpvh2X4daBqenQ2MuqW9h9mnniKtLHucuyMRyDnnB5r0VdPs1sxZraQC2AwIBENg/4DjFOt7O2s1K2lvFArHJESBQfyoA8s+La3tt488D6lPe/YNEtrmZJrx4hJHa3DoBE8gPGCeATwDk1d0Dwfo2sp4ys5NWXXItZZDdSpbItvFOY2UtGVODJjaWweDjnJNekywxzwtFPGskbjDI6ghh7g0kFvDawrDbRJDEowqRqFUfQCgDyH4UwT+IprKDX7CMSeC4JNLcvGMSXW8qXHHOI0U59ZCfeuZ0jTVuvD/jjQfGXiSPSLmTVJ5L61mtkee5QkNFLGzEMwOAFAzyPevoWOGKEuYo0QyNvfaoG5vU+p4HNRyWNpNdR3M1rDJcRf6uVowXT6HqOp/OgDxa9i8L3fx20Wx1e6sZ0/wCEae1uheOiySTMfL2Sc8SlGIx15rc+KmkWvhW38PeNdIsET/hGbqNLiKCNcvZMBGy++0bdvp+o9KfS9PkuPPextmmzu8wwqWz65xmua1TwrrWreJLr7VrYk8OXbW8kmntF80Zi5KK3912CliecAgdc0AWvA2jDTdAF7c28cWparI19eFUAIeU7thIA+6CF/CuP+Iuo6TbfGD4fedeWUVzDd3Pnl5UV41aHCbu4BJ4z1PSvVarzadZXEwluLO3lkGMO8SseOnJFAHkPhzUfCi/H7xveT3+kLD9itVSR54whJT97gk4zwN361ywmtdR/Y8iEEkV1Pp8qmUIQ72yi+zyByo2c/T2r6DOj6aWYnTrTLfePkLz9eKki0+zgikigtII45Bh0SIAOPQgDnqaAPIPirqDat4LSbw7DEuh6X4htvt81qgkSaBUR2lAGAyq7gEcglM5wDU95othqFz4o1O08Utq9/e+GZoJn0yBERUHMbMyEjzDyFHUqD6CvXIreGC3WCCGOKFRtWNFAUD0AHFY3iDwpZa34P1Dw/b402C+iMZe1jVdpPOdvQ+47jIoA8k8E3Og+J/E/hBbjV/DajQtLezW0trgu+oGSNUwUdVwB97Ydx3E9etRWlmZfEnxF0fxj4kt9Ge/vHkZLu3R/tNmy4iaNnOcKOMLnB967jUfh9qev2ltp2rr4ftbSGaKR7nT7ArPJ5bBhsycRZKgHBbgnBrv57K1uZYpbm2hmkhJMTyRhjGT3BPTp2oA8O1BPDkHxV+H1jrF7bSJa6JLFc/2k0aSlvJUR+cpPDkcgHnPSvbdP0yz0nS4dO0yBLS0gTy4oohgRr7UTaZYXMxluLG2lkPV5IVYn8SKtUAeKeGrPTNE1rx/4vi00yQeHI/7O04upkYfZ4iZT6klzy3ufeuq+CskUPw3sNLiLTPYxgzzrgxGWUmZkRgedm8KR2PFd9FDFApWGNIwzFyEUDLE5J+pJJJoggitoVhtokiiQYVI1Cqv0AoAzNb1uXSvJhstKvdUvLjd5UNumEGMZLyNhEHI6nJ7A4NYd74S1LxfCI/G13Cmn7gw0nTx8hI6eZMw3t/wAIOvWt7WNN1LUHhOm67caUEBDiGCKTzM4xnzFOMc9PWs3/hHPEX/Q73//AIA2v/xugDooII7a2jggXbHEgRFyTgAYAya8p+MOt6VD4m8CwTanZxzWniO3luY2uFDQptJ3OM5UYI5PHNerxI0cKJJI0rqoDSMACx9SBxz7VBPpen3MxlubG2mkbq8kKsT+JFAEtrd299ax3NlPFcW8o3RywuHRx6gjg1LTY4o4Y1jhRY0UYVVGAPwp1ABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAQ3lpDf2M9pdKXguI2jkUMVJVhgjIwRweorzfUfhP8JtKubKDUdAtYZb+byLZWmmJlkP8Iw3WvSLy5WzsZ7l0d1hjaQpGMswAzgDueK8p0zWL3xH4v8ABVxr18uJrR9aNlGqiKBpFEVuikfMxxKxO4nlSRgUAar/AAY+GEepQ2LeF4PPmikmQebLgqhQNzv9ZFqx/wAKM+G3/QrW/wD3+l/+Lrprr/ketL/7Bt5/6Ntq5zxFq+uQ/ECLT4NXSw0pNNmv7to4Y3eKKMoA2XBwzMXHddq9M9ADtLCxttL063sbCFYLW2jWKGJOiKowB+VWK5vwBrlz4h8E6ffakyG/aJftSpj5HKhsHHAO1lJA6EkV0lABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAVnW3h3RLK6W6s9HsLe4QsVlitUV1JGCQQM8gAGtGigCjNYNJ4gtNRDgJb2s8BTHJMjxMD+HlH86sNZWr3DzvbQtNJF5LyGMFmjyTsJ6lck8dOTU1FAENpZ21hapbWNvFbQJ9yKFAir9AOBU1FFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUVneIbm8s/DeoXOmxLLdRW7tErSbBkDrnB6denavL7LxR4oH/CvJntJ7q6vbO43Wsd/wDLe/uYyssrEAL1LEENjtkmhau39df8g7f1sew0VxEfxIS20rV5dd0qSx1HSrmO2ksY51l815ceVsfCghs9SBjnNFn8QpxqH2DXNIhsbia2luLX7PqKXKSmNdzIxUAo2OehB5weKV9L/wBbX/Idjt6K8+0/4l6jcWOjarf+F3stG1aWKBLs3yu8bycKTHtHybuN2c45wOleg1TTRN0wooopDCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKAK2pW73elXdtHgPNC8a56ZKkV594d0DxAt94GfUdHeyXQbS5tLpmuInBzCiI42sSQxB46jHNelUULR3/AK6/5g9bf1/Wx5h4i8BarrmoeLJUtrci5vLC8sBcsrRXJgjwyOvJAPK8jv6VPp3h67uLqSSD4eaJ4bRLOVTLsge4klZCoWJouEXk5LdQeg5r0iilbTl/raw76pnnM/hXWX+FPhjR1s839hcWL3MPmp8gjcFznODgDsTntXo1FFW5N383f8v8iUrW9LBRRRUjCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiuNX4t+BnupbaPxBDJPCSskccUjspBwcgL60AdlRXJf8LR8H/9BY/+Ak3/AMRVa4+MXgK1IF14ihhJ6CSGVf5rQB21FZmgeItJ8UaUupaBfR31mzlBLHnG4dRzzWnQAUUUUAFFFFABRRRQAUVT1bVbTQ9HutT1KUQ2lpE0srnsoH6n2rO8JXWr6npA1XW1Fu1/ia3sQo/0SIj5VZurORy3YE4A4oA3aK8zh8aeMdQ+Ik3hqysdMt1ks3vFe5jcyWMYmMaGVQ43s4G4KNuNwySAa6H4e+KbrxZol9c3ywF7PUrixWe3RkjuVjbAkVSSVz6ZPTrQB1dFctNr95ofjqDStZZZNN1kn+zbraFMM6rlrd8cHcAWQ9Thl54rqaACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACuF8OeP8AUfE3jK4s9P0Jn8PxS3FuNVEnHmRFQcjGMMxIA68Z9h0Xi0pH4Q1SeS6ubRLe2e4aa1l8uRRGN/Dds7cH2JrhvhHoE/h34RW2qC5vmvbywe8NvPMWiV33SBlTsTlc0Add4b1+48S6rqd1a7F0azmayt2C5a5lQ4kkDZ+6DlQO5BPpXR15z8M7a+uPgDo0OjXS299cWLGO5kX/AFbu7Fnx3IJJHqcZrg11XWW+EPiPxVP4n1pUi1CeHQIVuSHx5ojj3N96bJ4w2QMH8AD17xjr114WtrXW8LLpNvII9Sj2/OkbkKJlP+weo7qxPaujVgyhlIIIyCO9ct4g3ah8JdXi1R4pbg6PNHeeWQQsohO8cccNnirfgKaW4+HPhua4BEsmlWzPn1MS5oA36K4fxL4hGn69Nbnx3puj7VU/Y7iyWR0yo5Lbx169O9ZP/CXKP+ao6P8A+Cxf/jlAHp1fP/xk8Kal4D8WwfFTwSPLaOQf2pbKPlfJALkDqr9G98Hqcj39TlQchsjqOhqO6tYL6zmtbyJJredDHLG4yrqRggj0IoAyPB3izTvG3hW01zSHzDcL80ZILQuPvI2OhB/TB71rXVpbX1s9ve28VxC4w0UyB1b6g8GvBfBGha78KvjtN4V0SJ9S8N6xGbop5gJtEyQHbPQqRt/2hjuOPoCgDlfCvge08Gazqh8PhLbR9R2TfYBnEFwMhmT0Vl25HYrxxwOqorg/GWr37/ErwZ4c0q9kthcTzX18I2K74Ik4U+oZjjHtQB3lc5o+u3Wu+LNVis/LXSNJf7G7lctcXXDPg9lQEL05JPpWL4Y1+61j4reK45NQYaZp5h0+0tXICvOELzMo7kZA+lVPhPeQaV8FxrFzuYLJf3t0eNxZZ5S344WgDsfFWsT+HvCuoava20N09lCZzFNP5Ksq8t8+1sHGcccnA4zmub03xxqJt/C+ra1YpZ6d4jijh8sPuNncuC0eWwNyyDA6cNj14818Tza54p+D9jrOo6vfJqfizUIrW0063l228cMrlRH5fR/kBJY85x07+gfFZLS2+Cl1JpbI0dh9kkspI2BCmOePYVI+mKAPRaK4Pxjruo3Hj7w/4L0md7JdQSS8v7uPiRYI/wCBD/CWbgnsOlV/hpd6hdeKPGUUmp3t7pGnX62Fgl3J5pQoCZfnI3N8z4ySTgAEnFADfilIdW13wf4PBbydZ1Iz3aqfvwW6+Yyn2J2/lXo1eYeIiZP2lPB0Z5EWlXki+xIIP8q9JvJZYLKea2t2upo42aOBGVTKwHCgsQBk8ZJxQBzmleEJbLxl4m8QXF6Hn1lIYYPLQqbaKNMAckgksSfyqt8O/BF54K0mOyvNV+2JDCIYookZI/vs7ysCxzI7NyewAA9TL/wlHin/AKJ/qH/gxtP/AI5VrTfEHiC71GGC+8G3thbuSHuZL62dY+OpVXLH04HegDN+LunS33wx1W4smaO+0tBqVpKhw0UsJ3hh74DD8a6Tw9qya/4Z0zV4toS+tIrgBTwN6hsfhnFM8UIJPCGsI3AaxnB/79tXN/Bl3k+DPhkyAg/YwOfQMQP0FAHcUUUUAFcL4a8d6v4o1izn0/w/nw3eSXUSaj5/zx+S20OyY4DsCAOvGa7K+tPt1hNamee3EqlfNt32SJ7q3Y+9fP8A4c1PUPCXwCtNS0jUL0Xmqao2nIZJt8doGupFMiIRgNgHn1OaAPoiivMtfmuvBHjbwlYadqWpXVnr0stjeQ3d487Z2grMjOSUYE87cKR2HWsjwDo+u6x4o8Qeb4y1sW2g+IjFDFLcGUTQqoLRPnqDlee2DgcmgD2SiuH+MHijUPCPw2vdR0c+XdvJHbpOQD5G9gpfnjjPGe5FZGkaV4tHie2tpf7XstCu7J4bxr3VknmEyjKSwsHZlJ5BA456egB6fRXhem3euMdZ8D33iHWf+Ejj16CG2uxdEP8AY2BkEoHp5Sybh0zt71pajqWt3nxTufCFq2sXtho2lxyKlpqS2080j4/fSSllL7c4x0zyQaAPYqK8N1+Pxta+DfCJ1TXNQsddbX4dJka2vF2zwu77WkC5UyYCg59Oc5rf8a+FfEeh/DXULrR/FmuXmoaa73cJa42tJBuUtExHLEIGw3XNAHqdY/inXLrw9obX9jo15rUqyIn2WyXMhDHBbHoOv+c1yHg6b/hLvFH9vaXruqyaJZ2VrHHAbrdHPcNHvcuMclUaMMP7xOeRU3xulvbH4Wajqml6nfadd2JjeOSznMW7dKiENjqMMaAPQR09KK8u8fQaovxZ8G2mmeINUsYNaN3FdwQ3GI9kUIbKqeAxyfm5IJBHIFM0R766v/iP4Ym1fUnstKeBrOZrpmuIRJB5jKJT8xGRjkk4J570Aei6LrNpr+kRalpzM9tMXCMwxnaxUn6ZU1frxH4cahH4S+BEOu299eXeojRbm7h06a4Z4QInILKnYBiufqaseFX8bX0fhLX7Q6jKl20MmrXF3qcT211DKOWSHf8Au2BYbQoB4wRnigD2aivAtT1XxTqlnrSQanrKeKjr7WdlZWcki2ptFkCkjAC7Qu/c+dwI5I6Vt63f6tYfErR/AWntrOoafbaVJfSY1Lybq7YyMqgz7lYqg7AgnHOQKAPYqK8L8WN4z0j4Vvdalq+padqdhrYgs3S+V3ktZZI9gnMfDsAxHPPGe9er+F/D974ejvYr3X77WY55/Ng+2kM1uuANgbqwzzz+XqAbtFeV+K28RP8AGHQ9J0/xHc2aakZ5XhtzlIbOOFRypGDI0hfDdvl9K0Ph7q8s/jLxdpx1O4udLs76O0sBe3BlkMqR5uArMcsASvHOPagDuNV1A6Xp0l2LK6vfLIzDZxh5CCcZCkjOOuBz7GodH8Q6T4gheTSL6K58s4ljGVkiPo6HDIfZgDWg8iRRtJKyoigszMcAAdSTXAa7daP4svF/4RvR5dY1SDdHHq1pM1olqeh/0sckdPlQOD3BoA9BrnvGPjG18F6Q+o39hqN3CkbyMbK3MgQKBne3AXOeMnsfSrfhqy1jT9Dit/EWppql8pO64SER5HYHHUj+9gZ9BWJ8W/8AkkHif/sHS/yoA6TRdSXWdA0/VEjMS3ttHcCMnJQOobGfbNXawvA3/JO/Dn/YKtf/AEUtbtABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAHnXxTm8R6rayeEdD0Ka4ttatPJk1NWOy2ZpAGDDGABHuPJGcgCu806wi0zSbXT4OYbWBIEz/dVQo/QVZooA4bwHp76JpeoeB78yJ9heU2UoGPOs5WJRlbpuUsUI7EDsRm9J8N9BfwNYeFFW4j0/T3jktnSX94kiNuD5IwTkk4II56V1dFAHC+LtOWw8Er4N8OJJ9s1kPaxsWLMqOc3Fw7d8ByST1ZgO9dlp9jFpmmWthbAiG1hSGMH+6qgD9BViigAooooAK53xf4oPh6zgt9Ptvt+tag5h0+yBx5j92Y9kXqx9Pciuiryu/+GXjibxhfeItP+IsVldXS+UijQ45BDCDkRqXdiB3OOp5NAHZ+EfC3/CO21xcX1z9v1nUJPOv75lwZX7Ko7Io4Vew+tdDXmA8E/FJcA/FiMknjOgQf41Q1H4WfEbVYXjvPi7dqsgIZbfTFhBHp8kgoA9JtPEdjfeJLzRLMvPcWMKSXMiAGOIsSFjJz9/Ck4x0+orhH8I3vij42azrN/LqemW2kWdvZ6bdW58oyFwXlKllIYfNtPXn6cbnwv+HcXw38Nz6d9u/tG6ubhp57wxGMyEgADBZugHr612lAHK+Hvh3onhnWZ9S003bSzPJLsuJzIqySBRJIM8lm2LkknvjGTVDwdpieH7/XPCF9APsdxczX2n7l/dzW0xBkjHujsykejKe9dzRQBwbfCHQms7C2/tDWAmluraaftn/HjjOBGMY6HGWDHgc8U3xbpcernRPAmkRFbSGW3ub/AG52wWkLAohOMbnZAAPRWPQV31FAHnHjEC88Zf6V4W1p7jT7XOm6rpUrhrkyD54GZRtjXgAl2HqMc1rfCrwpc+Dvh7Z6dqQA1CR5Lm7AffiSRixG7vgYGec4rsaKAPMvH8J0v4wfD7xCq/u5LifSp27fvUPlj/vrdXptYHjXwyvi3wpc6Ysot7rKzWdzj/UXCHdG/wCDAZ9iR3qTwpr76/owe9g+yanbN5GoWZPMEw6j3U8Mp7qQaANuuK8C+KtZ8Q+IPFFjqyWHkaNeJaQzWaOvmPtLODuY/dyo4x3rnbm+Hiz4ieLE8QyTr4a8J28aizSVoluJmjLtI+CC2AMAE45Bq78CEih+G8O+5abUL2R9SuxIxaQecxKFmPUlFBzQBv8AxQ1b+w/hZ4ivuNy2MkaZz95xsX9WFWvAGktoXw70DTZARJb2EKyAjGH2gt+pNYfiaBvHHi6y8OW5D6Npc6XmsSA/LJIvMVt7nOHb0AHrXe0AFFFFAFe/tPt9hLa/aJ7bzFx5tu+yRPdW7GuV0r4XaBpfhi78Ol7690m6XDWt5ceYsZ3FyycAq245z6gV2VFAHM2XgXT7XVItSu77UtTvbaJ4bWe+ufMa2VvvFOAAx4+Y5PHWmeGfAOn+FNTu77TtQ1SV72RprlLm68xJZGwC5GPvcDmupooAp6tpNhruk3Gmavax3dlcpsmhkHDD+nPORyDWTofguy0O4t5hfajqD2kRhtft9x5v2dCACE4HOABuOTjjOK6KigDE/wCER0j/AITb/hK/s/8AxNfsn2PzM8bN2c4/vds+nFRa54M03XNYttXMt3YapbRmJL2xnMUhiJyY26hlzzgg4PTFdBRQBx2p/DHRdUt7CB7vVII7C4F3EILsqWuAxbzmJBLPlj8xqxrur6h4aTSrC10G/wDENnNDJDcXCv5kisqDyw42/NvOQWOAOp9K6migDnPAHhaLwb4H07Ro40jkij3zhOR5rHc2D3AJwD6AVJ4v8H2HjbRzper3F7HZsf3kVrP5Yl5BAbjnBAIrfooA468+Gthf6tpupXWta495pgYWsxveY9w2sfu9WHB9am0L4fab4f8AEd7rdpqGqzXWoY+1rc3XmJOQu1SwxzgHA9K6uigDl/Dvw+0Twz5iWAuZYDDJbw29zN5kdvC7b3jRT0UnGc5PA5pmj/DrSNDmthZ3GoNY2crTWumzXRe2t3JyCqkZ4JyoJIU8jB5rq6KAPDbXwlPZrfz6qfHFjr9zeXE7R6DKwtHdnYq6bf3YBGCd+DknIrurHwXPr/hvw9e+MJbi28U6fbjdqFhN5U0bMPmXcPlYHuCCCc+tdxRQBx2r/DHRda0aPTLy61MQi4N1M8d2Q9zNlSHkOPmI2jHYYwK6jTrL+z9PitftNxdeWCPOuX3yNznlsDPpVmigDK/4Ryw/4S8+JWEjah9iFipZvlSPfvOB2JPU+1Z+heAtF8O6zc6lp4uDJPLNKscsu6OBpWDSGNccbiBnrwAOldLRQBleJbRb3Q5YW0aHWvmRhZTOqq5DA5Jbjjr74xWTHrXiqKNY4vBKIijCquqRAAegG2urooApaTdX15Y+bqmnf2dPuI8nz1l47HcvFZ/ivwnaeL9NOn6leX8Fo6MksVpP5YmVsZD8HI4/U1u0UAY/hvw3D4Y08WVpfX91boqJEt5P5vlKowFXgYGK2KKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigArzD4ieKW1jw/qmiaJa3hZ9Qg0hdQgm8rZdO65VcEMQqnkjjPGCM49PridR+GFjfzuV1bUbaBtQfUhbwtGFW4kBDtnbuPDHGT8pOR2FAGr4miWG30OJMlU1S2UbiScDPc9aPFvi0eFI7F20y7v/tdwkAFsBkMzqigZIBYlxgZHCsf4ak8Txs66RsGduqQE89smrWr6HBrN1pc1zLKo027F2kaEbZHCMq7sjOBuyMY5AoAr+G/Ef9v/ANpRvZyWdxpt4bSeNnDqWCK+VYcEYcD2INbdYXh7wrb+HbzUp7a8vJ/7QuXuWjnkBWNnYs2AAM8tjJycBRnjndoAKKKKACiiigAooooAKQIocsFAZgATjk4paKAOf1LwN4c1fUbi91HThNNdqi3K+dIsdyEGE8yMMFk2jpuBxVzQfDeleGLE2eiWv2eEkEhpXkY4UKo3OScBQABnAAAGK1KKAGqipnYqruOTgYyfWnUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFABRRRQAUUUUAFFFFAH/9k=)

Sum(i=1 to N) Sum(j=i+1 to N) gcd(i,j):

G = sum(g=1 to n) h(g) \* cnt(g)

In this example H(g) = g. present H(g) as mobious : double loop, j+=i. f[j]+=h[i]\*miu[j/i]

G = sum(g=1 to n) (sum(d|g) f(d)) \* cnt(g)

G = sum(d=1 to n) f(d) \* cnt2(d). here cnt2(d) = how many d, d|gcd(i,j).

G = sum(d=1 to n) f(d) \* (n/d) \* (n/d-1)/2 (f(d) is totient here).

Loop through different values of n/d.

For(int i = 1; la; i <= n; i = la+1) {

la = n / (n / i); // n/x yields the same value for i <= x <= la

}

Next : triplet (i, j, k) : such that gcd(a[i], a[j], a[k]) = 1.

H(g) = 1 if (g = 0) else h(g) = 0.

Cnt2(d) = gcd(a[i], a[j], a[k]) is multiple of d.

If dp[x] is the number of i such that x | a[i], cnt2[d] = C(dp[x], 3)

For pair of LCM : excersice.

F(d) = sum(g|d)miu(g)dg.

G=sum(l=1 to n) ((1+(n/l)(n/l)/2)^2 g(l).

1. Chinese Remainder Theorem

// Chinese remainder theorem (special case): find z such that

// z % x = a, z % y = b. Here, z is unique modulo M = lcm(x,y).

// Return (z,M). On failure, M = -1.

PII chinese\_remainder\_theorem(int x, int a, int y, int b) {

int s, t;

int d = extended\_euclid(x, y, s, t);

if (a%d != b%d) return make\_pair(0, -1);

return make\_pair(mod(s\*b\*x+t\*a\*y,x\*y)/d, x\*y/d);

}

// Chinese remainder theorem: find z such that

// z % x[i] = a[i] for all i. Note that the solution is

// unique modulo M = lcm\_i (x[i]). Return (z,M). On

// failure, M = -1. Note that we do not require the a[i]'s

// to be relatively prime.

PII chinese\_remainder\_theorem(const VI &x, const VI &a) {

PII ret = make\_pair(a[0], x[0]);

for (int i = 1; i < x.size(); i++) {

ret = chinese\_remainder\_theorem(ret.second, ret.first, x[i], a[i]);

if (ret.second == -1) break;

}

return ret;

}

1. Tiles Solvability

If the grid width is odd, then the number of inversions in a solvable situation is even.

If the grid width is even, and the blank is on an even row counting from the bottom (second-last, fourth-last etc), then the number of inversions in a solvable situation is odd.

If the grid width is even, and the blank is on an odd row counting from the bottom (last, third-last, fifth-last etc) then the number of inversions in a solvable situation is even.

1. Formulas + Theorem

**Cayley’s Formula**: There are n^(n-2) spanning trees of a complete graph with n labeled vertices.

**Derangement**: A permutation of the elements of a set such that none of the elements appear in their original positions. F(n) = (n – 1) \* (F(n-1) + F(n-2)). F(0) = 1. F(1) = 0.

**Euler’s Formula for Planar Graph**:

V – E + F = 2 [V: vertices E: edges F: faces]

**Pick’s Theorem**:

A = i + b/2 – 1

A: Area I: internal points B: Border points

**Spanning Tree of Complete Bipartite Graph**: N^(M-1) \* M^(N-1)

N: row M: column

**Pythagorean Triples**

Integer solutions of x^2 + y^2 = z^2. All relatively prime triples are given by: x = 2mn, y = m^2 – n^2, z = m^2 + n^2, where m > n, gcd(m, n) = 1, and m != n (mod 2).

**Moser’s Circle**: Determine the number of pieces into which a circle is divided if n points on its circumference are joined by chords with no three internally concurrent. Solution: g(n) = nC4 + nC2 + 1.

**Matrix-Tree Theorem**: Let matrix T = [t\_ij], where t\_ij is the number of multiedges between i and j, for i != j, and t\_ii = -deg[i]. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and column from T.

A graph G is **bipartite** if and only if there is no cycle in G of odd length.

**Euler's Theorem**: a^ φ(m) = 1 (mod n), if gcd(a, n) = 1.

**Wilson's Theorem**: p is prime iff (p-1)! = -1 (mod p).

**Burnside’s lemma**: state that the number of combinations is sum(k=1 to n) c(k)/n where there are n ways to change the position of a combination, and there are c(k) combinations that remain unchanged when kth way is applied

**Catalan Number**: Cat(n) = C(2n,n) – C(2n,n+1) = 1/(n+1) C(2n, n).

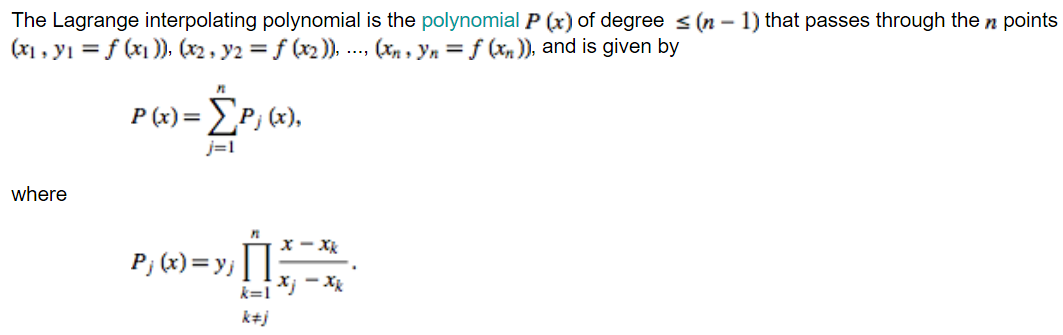
Cat(n+1) = sum(i=0 to n) Cat(i) Cat(n-i), n >= 0.

Cat(n+1) = ((2(2n+1))/(n+2)) Cat(n).

Count distinctc binary tree. Count n pairs of parentheses matched.

**Prime under**: π(10^3) = 168, π(10^4) = 1.229, π(10^5) = 9.592, π(10^6) = 78.498 π(10^7) = 664.579, π(10^8) = 5.761.455, π(10^9) = 50.847.534

**Chicken McNugget Theorem** states that for any two [relatively prime](https://artofproblemsolving.com/wiki/index.php?title=Relatively_prime) [positive integers](https://artofproblemsolving.com/wiki/index.php?title=Positive_integer) **m,n**, the greatest integer that cannot be written in the form **am+bn** for [nonnegative](https://artofproblemsolving.com/wiki/index.php?title=Nonnegative) integers **a,b** is **mn – m - n**.



Menelaus: then D, E and F are collinear (biimplikasi)



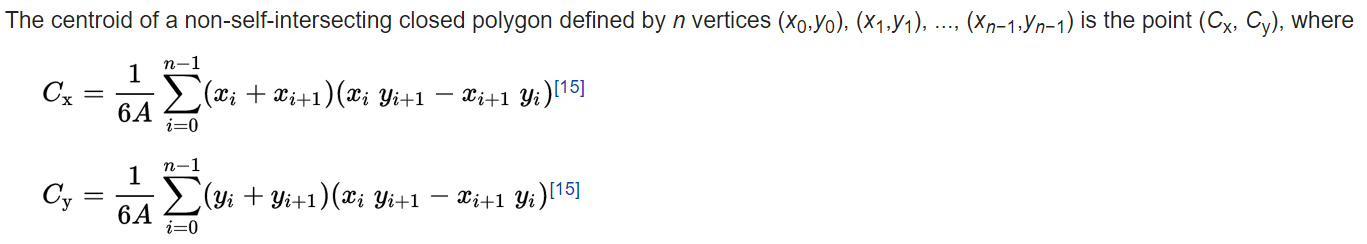
Ceva” then AD, BE and CF are concurrent at O (biimplikasi)

d2.a = b2.m + c2.n – a.n.m

Inequality QM-AM-GM-HM



Equality (x1=x2=..=xn)

  
A = signed Area

1. Heavy Light Decomposition

void HLD(int now) {

if (ChainHead[cnt] == -1) ChainHead[cnt] = now;

ChainNo[now] = cnt;

indx[now] = ptr;

int sc = -1, size = child[now].size(), next;

if (size == 0) ChainTail[cnt] = now;

ptr += 1;

for(int i = 0; i < size; i++) {

next = child[now][i];

if (sc == -1 || sub[sc] < sub[next]) sc = next;

}

if (sc != -1) HLD(sc); //sub = size\_subtree

for(int i = 0; i < size; i++) {

next = child[now][i];

if (sc != next) {

cnt += 1;

HLD(next);

}

}

}

inline int query\_up(int u, int v, int col) {

int uchain = ChainNo[u];

int vchain = ChainNo[v];

int res = 0;

while(true) {

ca = indx[ChainHead[uchain]];

cb = indx[ChainTail[uchain]];

if (uchain == vchain) {

if (u != v) {

res += update(1, 1, ptr - 1, indx[v] + 1, indx[u], col);

}

break;

}

res += update(1, 1, ptr - 1, indx[ChainHead[uchain]], indx[u], col);

u = ChainHead[uchain];

u = P[u][0];

uchain = ChainNo[u];

}

return res;

} // update tinggal ganti querynya

1. PBDS Template

#include <ext/pb\_ds/assoc\_container.hpp>

#include <ext/pb\_ds/tree\_policy.hpp>

#include <ext/rope>

using namespace \_\_gnu\_pbds;

using namespace \_\_gnu\_cxx;

typedef tree<int, null\_type, less<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> ordered\_set;

ordered\_set S;

// S.find\_by\_order(x) -> return pointer to the x-th element

// (int)S.order\_of\_key(x) -> return the position of lower\_bound(x)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |