# Size Balanced Tree

#define SBT a

struct SBTTree {

int l, r, s, key;

};

inline void leftrotate(int &x) {

int y = SBT[x].r; SBT[x].r = SBT[y].l; SBT[y].l = x;

SBT[y].s = SBT[x].s;

SBT[x].s = SBT[SBT[x].l].s + SBT[SBT[x].r].s + 1;

// renew(x); renew(y);

x = y;

}

inline void righrotate(int &x) {

int y = SBT[x].l; SBT[x].l = SBT[y].r; SBT[y].r = x;

SBT[y].s = SBT[x].s;

SBT[x].s = SBT[SBT[x].l].s + SBT[SBT[x].r].s + 1;

// renew(x); renew(y);

x = y;

}

inline void maintain(int &x, bool fl) {

if (!fl)

if (SBT[SBT[SBT[x].l].l].s > SBT[SBT[x].r].s)

righrotate(x);

else if (SBT[SBT[SBT[x].l].r].s > SBT[SBT[x].r].s)

leftrotate(SBT[x].l), righrotate(x);

else

return;

else

if (SBT[SBT[SBT[x].r].r].s > SBT[SBT[x].l].s)

leftrotate(x);

else if (SBT[SBT[SBT[x].r].l].s > SBT[SBT[x].l].s)

righrotate(SBT[x].r), leftrotate(x);

else

return;

maintain(SBT[x].l, 0);

maintain(SBT[x].r, 1);

maintain(x, 0);

maintain(x, 1);

}

inline void insert(int &x, int n) {

if (!x) {

SBT[n].l = SBT[n].r = 0; SBT[n].s = 1;

x = n;

return;

}

++ SBT[x].s;

if (SBT[n].key < SBT[x].key)

insert(SBT[x].l, n);

else

insert(SBT[x].r, n);

// renew(x);

maintain(x, SBT[n].key >= SBT[x].key);

}

inline void delett(int &x, int n) {

// if (!x) return;

if (x == n) {

if (!SBT[x].l || !SBT[x].r) {

x = SBT[x].l + SBT[x].r;

return;

}

righrotate(x); -- SBT[x].s;

delett(SBT[x].r, n);

// renew(x);

return;

}

-- SBT[x].s;

if (SBT[n].key < SBT[x].key)

delett(SBT[x].l, n);

else

delett(SBT[x].r, n);

// renew(x);

}

inline int findkth(int x, int k) {

if (k < 1 || k > SBT[x].s) return -1;

for (; ; )

if (k == SBT[SBT[x].l].s + 1)

return x;

else if (k <= SBT[SBT[x].l].s)

x = SBT[x].l;

else

k -= SBT[SBT[x].l].s + 1, x = SBT[x].r;

}

# Splay

#define SPL a

struct SPLTree {

int l, r, p, s, key;

};

inline void zig(int x) {

int y = SPL[x].p, z = SPL[y].p, w = SPL[x].l;

SPL[w].p = y; SPL[y].r = w;

SPL[y].p = x; SPL[x].l = y;

SPL[x].p = z;

if (y == SPL[z].l) SPL[z].l = x;

if (y == SPL[z].r) SPL[z].r = x;

// renew(y); renew(x);

}

inline void zag(int x) {

int y = SPL[x].p, z = SPL[y].p, w = SPL[x].r;

SPL[w].p = y; SPL[y].l = w;

SPL[y].p = x; SPL[x].r = y;

SPL[x].p = z;

if (y == SPL[z].l) SPL[z].l = x;

if (y == SPL[z].r) SPL[z].r = x;

// renew(y); renew(x);

}

inline void splay(int x) {

/\* int la = 0;

for (int i = x; ; i = SPL[i].p) {

que[++ la] = i;

if (!SPL[i].p) break;

}

for (int i = la; i; -- i) updata(que[i]); \*/

for (; SPL[x].p; ) {

int y = SPL[x].p, z = SPL[y].p;

if (!z)

if (x == SPL[y].l) zag(x); else zig(x);

else

if (x == SPL[y].l)

if (y == SPL[z].l) zag(y), zag(x); else zag(x), zig(x);

else

if (y == SPL[z].r) zig(y), zig(x); else zig(x), zag(x);

}

}

inline void cut(int x) {

if (!x) return;

int y = SPL[x].p;

if (x == SPL[y].l) SPL[y].l = 0; else SPL[y].r = 0;

SPL[x].p = 0;

// renew(y);

}

inline int join(int p, int q) {

if (!p) return q;

if (!q) return p;

int x = p;

for (; SPL[x].r; x = SPL[x].r);// updata(x);

splay(x);

SPL[q].p = x; SPL[x].r = q;

// renew(x);

return x;

}

# Compressed Trie

memset(T,0,sizeof(T)); T[0].l=1;

for (int i=1;i<=Number\_Of Name;++i)

for (int j=0,k=Length\_Of\_Name[i],p;k;) {

for (p=T[j].r;p>=T[j].l;--p,--k)

if (Name[T[j].a][p]!=Name[i][k]) break;

if (p>=T[j].l) {

T[++N]=T[j]; T[j].l=p+1; T[N].r=p;

memset(T[j].s,0,sizeof(T[j].s));

T[j].s[Name[T[j].a][p]-96]=N;

}

if (!k) break; else

if (T[j].s[Name[i][k]-96]) j=T[j].s[Name[i][k]-96]; else {

T[j].s[Name[i][k]-96]=++N;

T[N].a=i; T[N].l=1; T[N].r=k;

j=N;

}

}

# Dinic

for (off = t; ; )

if (build())

dinic(s);

else break;

inline void addedge(int u, int v, int c) {

w[++ W].v = v; w[W].c = c; w[W].next = ww[u]; ww[u] = W;

w[++ W].v = u; w[W].c = 0; w[W].next = ww[v]; ww[v] = W;

}

inline int build() {

int fi, la;

memset(dist, 0, sizeof(dist)); dist[que[la = 1] = s] = 1;

for (fi = 1; fi <= la; ++ fi) {

int u = que[fi];

for (int i = ww[u]; i; i = w[i].next) if (w[i].c) {

int v = w[i].v;

if (dist[v]) continue;

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

if (v == t) return 1;

}

}

return 0;

}

inline void dinic(int u) {

if (u == t) {

int flow = 1 << 30;

for (int i = t; i != s; i = w[stq[i] ^ 1].v) flow = min(flow, w[stq[i]].c);

for (int i = t; i != s; i = w[stq[i] ^ 1].v) {

w[stq[i]].c -= flow; w[stq[i] ^ 1].c += flow;

if (!w[stq[i]].c) off = w[stq[i] ^ 1].v;

}

maxflow += flow;

return;

}

for (int i = ww[u]; i; i = w[i].next) if (w[i].c) {

int v = w[i].v;

if (dist[v] != dist[u] + 1) continue;

stq[v] = i; dinic(v);

if (dist[u] > dist[off]) return;

off = t;

}

dist[u] = -1;

}

# Cost Flow

inline void addedge(int u, int v, int c, int q) {

w[++W].v = v; w[W].c = c; w[W].q = q; w[W].next = ww[u]; ww[u] = W;

w[++W].v = u; w[W].c = 0; w[W].q = -q; w[W].next = ww[v]; ww[v] = W;

}

inline int mcmf() {

int mincost = 0;

for (int la; ; ) {

memset(dist, 60, sizeof(dist));

// memset(dist, -60, sizeof(dist));

memset(visit, 0, sizeof(visit));

dist[s] = 0; visit[s] = 1; que[la = 1] = s;

for (int fi = 1; fi <= la; ++ fi) {

int u = que[fi];

for (int i = ww[u]; i; i = w[i].next) if (w[i].c) {

int v = w[i].v;

if (dist[v] <= dist[u] + w[i].q) continue;

// if (dist[v] >= dist[u] + w[i].q) continue;

dist[v] = dist[u] + w[i].q; stq[v] = i;

if (visit[v]) continue;

visit[v] = 1; que[++ la] = v;

}

visit[u] = 0;

}

if (dist[t] > 1 << 29) break;

// if (dist[t] <= 0) break;

// if (dist[t] < -1 << 29) break;

int flow = 1 << 30;

for (int i = t; i != s; i = w[stq[i] ^ 1].v)

flow = min(flow, w[stq[i]].c);

for (int i = t; i != s; i = w[stq[i] ^ 1].v) w[stq[i]].c -= flow, w[stq[i] ^ 1].c += flow;

mincost += dist[t] \* flow;

}

return mincost;

}

# KM

int hungary(int u) {

X[u] = 1;

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

if(!Y[i] && lx[u] + ly[i] == maq[u][i]) {

Y[i] = 1;

if(linky[i] == 0 || hungary(linky[i])) {

linky[i] = u;

return 1;

}

}

return 0;

}

void KM() {

memset(linky, 0, sizeof(linky));

memset(lx, 0, sizeof(lx));

// memset(lx, 60, sizeof(lx));

memset(ly, 0, sizeof(ly));

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

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

lx[i] = max(lx[i], maq[i][j]);

// lx[i] = min(lx[i], maq[i][j]);

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

for (; ; ) {

memset(X, 0, sizeof(X));

memset(Y, 0, sizeof(Y));

if (hungary(k)) break;

int d = 1 << 30;

// d = -1 << 30;

for (int i = 1; i <= N; ++ i) if (X[i])

for (int j = 1; j <= N; ++ j) if (!Y[j])

d = min(d, lx[i] + ly[j] - maq[i][j]);

// d = max(d, lx[i] + ly[j] - maq[i][j]);

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

if (X[i]) lx[i] -= d;

if (Y[i]) ly[i] += d;

}

}

}

# Tarjan

void tarjan(int u) {

s[u].low = s[u].dfn = ++ DFN;

stk[++ STK] = u; s[u].in = 1;

for (int i = ww[u]; i; i = w[i].next) {

int v = w[i].v;

if (!s[v].dfn) {

tarjan(v);

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

if (s[v].low >= s[u].dfn) s[u].cut = 1;

if (s[v].low > s[u].dfn) w[i].cut = 1;

} else

if (s[v].in) s[u].low = min(s[u].low, s[v].dfn);

}

if (s[u].low == s[u].dfn)

for (++ BLOCK; stk[STK + 1] != u; -- STK) {

int v = stk[STK];

s[v].in = 0;

s[v].block = BLOCK;

}

}

# Extend GCD

inline void exGCD(int a, int b, int c) {

if (b == 0) {

x = c / a; y = 0;

return;

}

exGCD(b, a % b, c);

int s = y, t = x - a / b \* y;

x = s; y = t;

}

inline int exGCD2(int N, int A, int B, int C) {

if (!N || !C) return 0;

int T = A / C \* N + B / C \* (N - 1) \* N / 2;

A %= C; B %= C;

return T + exGCD2((A + B \* N) / C, (A + B \* N) % C, C, B);

}

# Miller Rabin

const int prime[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47};

Int64 producMult(Int64 A,Int64 B,Int64 C) {

Int64 ANS = 0, P = A;

for (; B; B >>= 1) {

if (B & 1) if ((ANS += P) >= C) ANS -= C;

if ((P += P) >= C) P -= C;

}

return ANS;

}

Int64 producExp(Int64 A,Int64 B,Int64 C) {

Int64 ANS = 1, P = A;

for (; B; B >>= 1) {

if (B & 1) ANS = producMult(ANS, P, C);

P = producMult(P, P, C);

}

return ANS;

}

bool witness(Int64 A, Int64 N) {

if (N == 1) return 1;

Int64 TIMES = 0, NN = N - 1;

for (; !(NN & 1); NN >>= 1) ++ TIMES;

Int64 X, Y = producExp(A, NN, N), Z;

for (; TIMES--; )

{

Z = producMult(Y, Y, N); X = Y; Y = Z;

if (Y == 1 && X != 1 && X != N - 1) return 0;

}

if (Y != 1) return 0;

return 1;

}

bool millerRabin(Int64 N)

{

for (int i = 0; i < 15; ++ i) if (N == prime[i]) return 1;

for (int i = 0; i < 15; ++ i)

if (!witness(prime[i], N)) return 0;

return 1;

}

# Pollard Rho

Int64 gcd(Int64 A,Int64 B) {

if (!B) return A; else return gcd(B, A % B);

}

Int64 pollarRho(Int64 N) {

Int64 x = rand() % N, y = x;

for (int i = 2, k = 2; ; ++ i) {

x = producMult(x, x, N);

if (!x) x = N - 1; else -- x;

if (x == y) return N;

Int64 GCD = gcd(N + y - x, N);

if (GCD > 1 && GCD < N) return GCD;

if (i == k) y = x, k += k;

}

}

void findFactor(Int64 N) {

if (N == 1) return;

if (Miller\_Rabin(N)) { factor[++FACTOR] = N; return; }

Int64 Q = N;

for (; Q == N || Q == 1; ) Q = pollardRho(N);

findFactor(Q);

findFactor(N / Q);

}

# Minimum Expression

int work(char st[],int len) {

int i = 0, j = 1, k = 0;

while (k < len)

if (st[(i + k) % len] == st[(j + k) % len]) ++ k; else {

if (st[(i + k) % len] > st[(j + k) % len]) i += k + 1; else j += k + 1;

if (i == j) ++ j;

k = 0;

}

return (i % len < j % len ? i % len : j % len);

}

# O(N) Prime

memset(next, 0, sizeof(next));

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

{

if (!next[i]) prime[++ PRIME] = i, next[i] = i;

for (int j = 1; j <= PRIME && i \* prime[j] <= N; ++ j)

{

next[i \* prime[j]] = prime[j];

if (i % prime[j] == 0) break;

}

}

# KMP & Extend KMP

next[1] = 0;

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

for (j = next[i - 1]; j > 0 && str[j + 1] != str[i]; j = next[j]);

if (str[j + 1] == str[i]) next[i] = j + 1; else next[i] = 0;

}

for (A[k = 2] = 0; S[A[2] + 1] == S[A[2] + 2]; ++A[2]);

for (int i = 3; i <= N; ++i) {

int maxLen = k + A[k] - 1, j = i - k + 1;

if (i + A[j] - 1 < maxLen) A[i] = A[j]; else {

int p = max(0, maxLen - i + 1);

for (; S[i + p] == S[1 + p]; ++p);

A[i] = p; k = i;

}

}

for (A[k = 2] = 0; S[A[2] + 1] == S[A[2] + 2]; ++A[2]);

for (int i = 3; i <= N; ++i) {

int maxLen = k + A[k] - 1, j = i - k + 1;

if (i + A[j] - 1 < maxLen) A[i] = A[j]; else {

int p = max(0, maxLen - i + 1);

for (; S[i + p] == S[1 + p]; ++p);

A[i] = p; k = i;

}

}

# Suffix Array

void suffixSort() {

s[n++] = 'z' + 1;

int m = 27;

for (int i = 0; i < m; i++) cnt[i] = 0;

for (int i = 0; i < n; i++) cnt[rk[i] = s[i] - 'a']++;

for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1];

for (int i = n - 1; i >= 0 ; i--) sa[--cnt[rk[i]]] = i;

for (int l = 1, p = 0; p < n; m = p, l \*= 2) {

p = 0;

for (int i = n - l; i < n; i++) tmp[p++] = i;

for (int i = 0; i < n; i++) if (sa[i] >= l) tmp[p++] = sa[i] - l;

for (int i = 0; i < m; i++) cnt[i] = 0;

for (int i = 0; i < n; i++) cnt[rk[tmp[i]]] ++;

for (int i = 1; i < m; i++) cnt[i] += cnt[i - 1];

for (int i = n - 1; i >= 0; i--) sa[--cnt[rk[tmp[i]]]] = tmp[i];

swap(tmp, rk);

rk[sa[0]] = 0;

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

rk[sa[i]] = (tmp[sa[i]] == tmp[sa[i - 1]] && tmp[sa[i] + l] == tmp[sa[i - 1] + l]) ? p - 1 : p++;

}

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

for (int i = 0, k = 0; i < n; height[rk[i++]] = k) {

k = k ? k - 1 : 0;

if (!rk[i]) continue;

for (int j = sa[rk[i] - 1]; s[i + k] == s[j + k]; k++);

}

n--;

}