# SBT

#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;

}

# Compress Trie

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

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

for (int j=0,k=len[i],p;k;) {

//找公共部分

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

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

//分解该点

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;

}

//找子节点

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;

}

}

# Tree Chain

void BuildChain() {

int fi, la;

memset(visit, 0, sizeof(visit)); visit[1] = 1;

que[la = 1] = 1;

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

int u = que[fi];

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

int v = w[i].v;

if (!visit[v]) continue;

visit[v] = 1; father[v] = u;

que[++ la] = v;

}

}

for (int i = N; i; -- i) {

int u = que[i];

s[u] = 1;

for (int j = ww[u]; j; j = w[j].next) s[u] += s[w[j].v];

}

c[++C].head = 1; c[C].len = 1;

memset(clain, 0, sizeof(clain)); clain[1] = 1;

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

int u = que[i], maxs = 0, k = 0;

for (j = ww[u]; j; j = w[j].next) if (w[j].v != father[u]) {

int v = w[j].v;

if (s[v] > maxs) { maxs = s[v]; k = v; }

}

if (k) {

suc[u] = k;

clain[k] = clain[u];

++ c[clain[u]].len;

}

for (int j = ww[u]; j; j = w[j].next) if (w[j].v != father[u])

if (w[j].v != k) {

int v = w[j].v;

clain[v] = ++ C; c[C].head = v; c[C].len = 1;

}

}

}

# 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;

}