York University Team Notebook C++ (2019-2020)

Contents

1 Template

1.1 Template

```cpp
using namespace std;
#include <bits/stdc++.h>
#define ll long long
#define FOR(i,n) for(int (i)=0;(i)<(n);++(i))
#define PRE(i, m, n, in) for(int (i)=(m);(i)<(n);i+=in)
#define forn(i,n) for(int (i)=0;(i)<(n);++(i))
#define for1(i,n) for((int) i=1;i<=int(n);++i)
#define fore(i,l,r) for((int) i=l;i<=int(r);++i)
#define srt(v) sort(v.begin(),v.end())
#define printv(a) printa(a,0,a.size())
#define debug(x) cout<<#x" = "<<(x)<<endl
#define printa(a,L,R) for(int i=L;i<R;i++) cout<<a[i]<<(i==R-1?"\n":" ")
#define printv(a) printa(a,0,a.size())
#define print2d(a,r,c) for(int i=0;i<r;i++) for(int j=0;j<c;j++) cout<<a[i
    ][j]<<(j==c-1?"\n":" ")

typedef vector<string>vs;
typedef pair<ll,ll> ii;
typedef vector<ll> vl;
typedef vector<vl> vvl;
typedef vector<ii> vii;
typedef map<ll,ll> ml;
typedef map<ll, string>mpls;

int main() {
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);

        return 0;
}
```

2 Data Structure

2.1 Fenwick Tree

```cpp
//fenwick trie with range update and range sum query
void add(ll p, ll x){
    for(int i = p; i <= n; i += i & -i)
        sum1[i] += x, sum2[i] += x * p;
}
void range_add(ll l, ll r, ll x){
    add(l, x), add(r + 1, -x);
}
ll ask(ll p){
    ll res = 0;
    for(int i = p; i; i -= i & -i)
        res += (p + 1) * sum1[i] - sum2[i];
    return res;
}
ll range_ask(ll l, ll r){
    return ask(r) - ask(l - 1);
}
// two dimensional, single update, range query
void add(int x, int y, int z){
    int memo_y = y;
    while(x <= n){
        y = memo_y;
        while(y <= n)
            tree[x][y] += z, y += y & -y;
        x += x & -x;
    }
}
void ask(int x, int y){
    int res = 0, memo_y = y;
    while(x){
        y = memo_y;
        while(y)
            res += tree[x][y], y -= y & -y;
        x -= x & -x;
    }
}
// 2D, range update, single query
void add(int x, int y, int z){
    int memo_y = y;
    while(x <= n){
        y = memo_y;
        while(y <= n)
            tree[x][y] += z, y += y & -y;
        x += x & -x;
    }
}
void range_add(int xa, int ya, int xb, int yb, int z){
    add(xa, ya, z);
    add(xa, yb + 1, -z);
    add(xb + 1, ya, -z);
    add(xb + 1, yb + 1, z);
}
void ask(int x, int y){
    int res = 0, memo_y = y;
    while(x){
        y = memo_y;
        while(y)
            res += tree[x][y], y -= y & -y;
        x -= x & -x;
    }
}
// 2D, range update, range query
ll t1[N][N], t2[N][N], t3[N][N], t4[N][N];
void add(ll x, ll y, ll z){
    for(int X = x; X <= n; X += X & -X)
        for(int Y = y; Y <= m; Y += Y & -Y){
            t1[X][Y] += z;
            t2[X][Y] += z * x;
            t3[X][Y] += z * y;
            t4[X][Y] += z * x * y;
        }
}
void range_add(ll xa, ll ya, ll xb, ll yb, ll z){ //(xa, ya)      (xb, yb)

    add(xa, ya, z);
    add(xa, yb + 1, -z);
    add(xb + 1, ya, -z);
    add(xb + 1, yb + 1, z);
}
ll ask(ll x, ll y){
    ll res = 0;
    for(int i = x; i; i -= i & -i)
        for(int j = y; j; j -= j & -j)
            res += (x + 1) * (y + 1) * t1[i][j]
                - (y + 1) * t2[i][j]
                - (x + 1) * t3[i][j]
                + t4[i][j];
    return res;
}
ll range_ask(ll xa, ll ya, ll xb, ll yb){
    return ask(xb, yb) - ask(xb, ya - 1) - ask(xa - 1, yb) + ask(xa - 1,
         ya - 1);
}
```

2.2 Segment Tree(single update)

```
#include<bits/stdc++.h>

#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
#define ms(a,x) memset(a,x,sizeof(a))
#define tr t[root]
using namespace std;

const int N=5e4+5;
struct segtree{
    int l,r,val;
}t[N<<2];
int a[N];
void build(int root,int l,int r){
    tr.l=l;
    tr.r=r;
    if(l==r){
        tr.val=a[l];
        return;
    }
    int mid=(l+r)>>1;
    build(root<<1,l,mid);
    build(root<<1|1,mid+1,r);
    tr.val=t[root<<1].val+t[root<<1|1].val;
}
void update(int root,int i,int x){
    int l=tr.l;
    int r=tr.r;
    if(tr.l==tr.r){
        tr.val+=x;
        return;
    }
    int mid=(l+r)/2;
    if(i<=mid) update(root<<1,i,x);
    else update(root<<1|1,i,x);
    tr.val=t[root<<1].val+t[root<<1|1].val;
}
int q(int root,int l,int r){
    if(l<=tr.l&&r>=tr.r){
        return tr.val;
    }
    int mid=(tr.l+tr.r)>>1;
    int ans=0;
    if(l<=mid) ans+=q(root<<1,l,r);
    if(r>mid) ans+=q(root<<1|1,l,r);
    return ans;
}
```

2.3 Segment Tree(range update)

```
#include<bits/stdc++.h>

#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
using namespace std;
typedef long long ll;

const int N=1e6+5;
ll a[N];
struct segt{
    int l,r;
    ll val,add;
}t[N<<2];
void build(int root,int l,int r){
    t[root].l=l;
    t[root].r=r;
    if(l==r){
        t[root].val=a[l];
        return ;
    }
    int mid=(l+r)>>1;
    build(root<<1,l,mid);
    build(root<<1|1,mid+1,r);
    t[root].val=t[root<<1].val+t[root<<1|1].val;
}
void spread(int p){
    if(t[p].add){
        t[p*2].val+=t[p].add*(t[p<<1].r-t[p<<1].l+1);
        t[p<<1|1].val+=t[p].add*(t[p<<1|1].r-t[p<<1|1].l+1);
        t[p<<1].add+=t[p].add;
        t[p<<1|1].add+=t[p].add;
        t[p].add=0;
    }
}
void update(int root,int l,int r,ll x){
    if(l<=t[root].l&&r>=t[root].r){
        t[root].val+=x*(t[root].r-t[root].l+1);
        t[root].add+=x;
        return;
    }
    spread(root);
    int mid=(t[root].l+t[root].r)/2;
    if(l<=mid) update(root<<1,l,r,x);
    if(r>mid) update(root<<1|1,l,r,x);
    t[root].val=t[root<<1].val+t[root<<1|1].val;
}
ll q(int root,int l,int r){
    if(l<=t[root].l&&r>=t[root].r) return t[root].val;
    spread(root);
    int mid=(t[root].l+t[root].r)>>1;
    ll ans=0;
    if(l<=mid) ans+=q(root<<1,l,r);
    if(r>mid) ans+=q(root<<1|1,l,r);
    return ans;
}
```

2.4 Persistent Segment Tree

```
//find the nth biggest in [l,r]
#include<bits/stdc++.h>

#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
using namespace std;

typedef long long ll;
#define lson tree[tree[root].l]
#define rson tree[tree[root].r]
#define tr tree[root]
#define tcur tree[cur]
const int maxn=1e5+5;
const int mxsz=(maxn*20)+10;
int tot=0;
int a[maxn],ver[maxn];
struct segtree{
        int l,r,data;
}tree[mxsz];
void build(int l,int r,int root){
        if(l==r){
                tr.data=0;
                return;
        }
        int mid=(l+r)>>1;
        tree[root].l=++tot;
        build(l,mid,tr.l);
        tree[root].r=++tot;
        build(mid+1,r,tr.r);
        tree[root].data=lson.data+rson.data;
}
void update(int root,int cur,int l,int r,int id){
        if(l==r) return;
        int mid=(l+r)>>1;
        if(id<=mid){
                tcur.r=tr.r;
                tcur.l=++tot;
                update(tr.l,tcur.l,l,mid,id);
        }else{
                tcur.l=tr.l;
                tcur.r=++tot;
                update(tr.r,tcur.r,mid+1,r,id);
        }
        tcur.data=tree[tcur.l].data+tree[tcur.r].data;
}
int query(int o,int v,int l,int r,int kth){
        if(l==r) return l;
        int mid=(l+r)>>1;
        int res=tree[tree[v].l].data-tree[tree[o].l].data;
        if(res>=kth)  return query(tree[o].l,tree[v].l,l,mid,kth);
        else return query(tree[o].r,tree[v].r,mid+1,r,kth-res);
}
int n,m,idx[maxn];
int main(){
        update(ver[i-1],ver[i],1,len,x);
        query(ver[l-1],ver[r],1,len,k)];
}
```

2.5 Sparse Table

```
#include <bits/stdc++.h>
using namespace std;
const int logn = 21;
const int maxn = 2000001;
int f[maxn][logn], Logn[maxn];
void pre() {
  Logn[1] = 0;
  Logn[2] = 1;
  for (int i = 3; i < maxn; i++) {
    Logn[i] = Logn[i / 2] + 1;
  }
}
int main() {
  int n,m;
  cin>>n>>m;
  for (int i = 1; i <= n; i++) f[i][0] = read();
  pre();
  for (int j = 1; j <= logn; j++)
    for (int i = 1; i + (1 << j) - 1 <= n; i++)
      f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
  for (int i = 1; i <= m; i++) {
    int x,y;
    cin>>x>>y;
    int s = Logn[y - x + 1];
    printf("%d\n", max(f[x][s], f[y - (1 << s) + 1][s]));
  }
  return 0;
}
```

3 Graph Theory

3.1 Lowest Common Ancestor

```
#include<bits/stdc++.h>
#define MXN 50007
using namespace std;
std::vector<int> v[MXN];
std::vector<int> w[MXN];

int fa[MXN][31], cost[MXN][31], dep[MXN];
int n, m;
int a, b, c;
void dfs(int root, int fno) {
  fa[root][0] = fno;
  dep[root] = dep[fa[root][0]] + 1;
  for (int i = 1; i < 31; ++i) {
    fa[root][i] = fa[fa[root][i - 1]][i - 1];
    cost[root][i] = cost[fa[root][i - 1]][i - 1] + cost[root][i - 1];
  }
  int sz = v[root].size();
  for (int i = 0; i < sz; ++i) {
    if (v[root][i] == fno) continue;
    cost[v[root][i]][0] = w[root][i];
```

```
      dfs(v[root][i], root);
    }
  }
  int lca(int x, int y) {
    if (dep[x] > dep[y]) swap(x, y);
    int tmp = dep[y] - dep[x], ans = 0;
    for (int j = 0; tmp; ++j, tmp >>= 1)
      if (tmp & 1) ans += cost[y][j], y = fa[y][j];
    if (y == x) return ans;
    for (int j = 30; j >= 0 && y != x; --j) {
      if (fa[x][j] != fa[y][j]) {
        ans += cost[x][j] + cost[y][j];
        x = fa[x][j];
        y = fa[y][j];
      }
    }
    ans += cost[x][0] + cost[y][0];
    return ans;
  }
  int main() {
    memset(fa, 0, sizeof(fa));
    memset(cost, 0, sizeof(cost));
    memset(dep, 0, sizeof(dep));
    scanf("%d", &n);
    for (int i = 1; i < n; ++i) {
      scanf("%d %d %d", &a, &b, &c);
      ++a, ++b;
      v[a].push_back(b);
      v[b].push_back(a);
      w[a].push_back(c);
      w[b].push_back(c);
    }
    dfs(1, 0);
    scanf("%d", &m);
    for (int i = 0; i < m; ++i) {
      scanf("%d %d", &a, &b);
      ++a, ++b;
      printf("%d\n", lca(a, b));
    }
    return 0;
  }
```

3.2 Strongly Connected Components

```
#include<bits/stdc++.h>

using namespace std;

const int INF = 0x3f3f3f3f;
typedef long long ll;

int n,m;
const int N=1e4+5;
vector<int> vec[N];//graph
int id=1,scc=0;
int ids[N],low[N];//vevrtices with same low value are in the same SCC
bool onstack[N];
int stk[N],top=-1;
int out[N];
void dfs(int x){
    stk[++top]=x;
    onstack[x]=1;
    ids[x]=low[x]=id++;
    forn(i,vec[x].size()){
        int to=vec[x][i];
        if(ids[to]==-1) dfs(to);
        if(onstack[to]) low[x]=min(low[to],low[x]);
    }
    if(ids[x]==low[x]){
        while(top>-1){
            int node=stk[top--];
            onstack[node]=0;
            low[node]=ids[x];
            if(node==x) break;
        }
        scc++;
    }
}
void tarjan(){
    for1(i,n) ids[i]=-1;
    for1(i,n){
        if(ids[i]==-1) dfs(i);
    }
}
```

3.3 Shortest Path(Dijkstra)

```
const int INF = 0x3f3f3f3f;
const int MAXN = 1000010;
struct qnode {
    int v;
    int c;
    qnode(int _v = 0, int _c = 0): v(_v), c(_c){}
    bool operator<(const qnode& r) const{ return c > r.c;}
};
struct Edge {
    int v, cost;
    Edge(int _v = 0, int _cost = 0): v(_v), cost(_cost){}
};
vector<Edge> E[MAXN];
bool vis[MAXN];
int dist[MAXN];
//              1
void Dijkstra(int n, int start){
    memset(vis, false, sizeof(vis));
    for (int i = 1; i <= n; i++)
        dist[i] = INF;
    priority_queue<qnode> que;
    while (!que.empty())
        que.pop();
    dist[start] = 0;
    que.push(qnode(start, 0));
    qnode tmp;
    while (!que.empty()) {
        tmp = que.top();
        que.pop();
        int u = tmp.v;
        if (vis[u])
            continue;
        vis[u] = true;
        for (int i = 0; i < E[u].size(); i++) {
            int v = E[tmp.v][i].v;
            int cost = E[u][i].cost;
            if (!vis[v] && dist[v] > dist[u] + cost) {
                dist[v] = dist[u] + cost;
                que.push(qnode(v, dist[v]));
            }
        }
    }
}
void addedge(int u, int v, int w){
    E[u].push_back(Edge(v, w));
}
```

3.4 Topological Sort

```
//in-degree
in_deg[N];
bool toposort() {
    queue<int> q;
    for (i = 0; i < n; i++)
        if (in_deg[i] == 0) q.push(i);
    vector<int> ans;
    while (!q.empty()) {
        u = q.pop();
        ans.push_back(u);
        for each edge(u, v) {
            if (--in_deg[v] == 0) q.push(v);
        }
    }
}
//dfs
bool dfs(int u) {
  c[u] = -1;
  for (int v = 0; v <= n; v++)
    if (G[u][v]) {
      if (c[v] < 0)
        return false;
      else if (!c[v])
        dfs(v);
    }
  c[u] = 1;
  topo.push_back(u);
  return true;
}
bool toposort() {
  topo.clear();
  memset(c, 0, sizeof(c));
  for (int u = 0; u <= n; u++)
    if (!c[u])
      if (!dfs(u)) return false;
  reverse(topo.begin(), topo.end());
  return true;
}
```

4 Math

4.1 Baby-step Giant-step

```
// solve a^x=b(mod n), 0<= x <n
#define MOD 76543
int hs[MOD], head[MOD], next[MOD], id[MOD], top;
void insert(int x, int y)
{
    int k = x % MOD;
    hs[top] = x, id[top] = y, next[top] = head[k], head[k] = top++;
}
int find(int x)
{
    int k = x % MOD;
    for (int i = head[k]; i !=   1 ; i = next[i])
        if (hs[i] == x) return id[i];
    return   1 ;
}
int BSGS(int a, int b, int n)
{
    memset(head,  1 , sizeof(head));
    top = 1;
    if (b == 1) return 0;
    int m = sqrt(n * 1.0), j;
    long long x = 1, p = 1;
    for (int i = 0; i < m; ++i, p = p * a % n)
        insert(p * b % n, i);
    for (long long i = m;; i += m) {
        if ((j = find(x = x * p % n)) !=   1 ) return i   j;
        if (i > n) break;
    }
    return   1 ;
}
```

4.2 Euler's totient function

```
int euler_phi(int n) {
  int m = int(sqrt(n + 0.5));
  int ans = n;
  for (int i = 2; i <= m; i++)
    if (n % i == 0) {
      ans = ans / i * (i - 1);
      while (n % i == 0) n /= i;
    }
  if (n > 1) ans = ans / n * (n - 1);
  return ans;
}
//linear sieve
void phi_table(int n, int* phi) {
  for (int i = 2; i <= n; i++) phi[i] = 0;
  phi[1] = 1;
  for (int i = 2; i <= n; i++)
    if (!phi[i])
      for (int j = i; j <= n; j += i) {
        if (!phi[j]) phi[j] = j;
        phi[j] = phi[j] / i * (i - 1);
      }
}
```

4.3 Extended Euclidean

```
int ex_gcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int d = ex_gcd(b, a % b, x, y);
  int temp = x;
  x = y;
  y = temp - a / b * y;
  return d;
}
//the smallest integer solution to ax+by=c;
bool liEu(int a, int b, int c, int& x, int& y) {
  int d = ex_gcd(a, b, x, y);
  if (c % d != 0) return 0;
  int k = c / d;
  x *= k;
  y *= k;
  return 1;
}
```

4.4 Gause Elimination

```
#define eps 1e  9
const int MAXN = 220;
```

```
double a[MAXN][MAXN], x[MAXN]; //left matrix and right numbers,x stores
    solution
int equ, var; //number of equations and variables
//0: no solution 1:solution
int Gauss(){
    int i, j, k, col, max_r;
    for (k = 0, col = 0; k < equ && col < var; k++, col++) {
        max_r = k;
        for (i = k + 1; i < equ; i++)
            if (fabs(a[i][col]) > fabs(a[max_r][col]))
                max_r = i;
        if (fabs(a[max_r][col]) < eps)
            return 0;
        if (k != max_r) {
            for (j = col; j < var; j++)
                swap(a[k][j], a[max_r][j]);
            swap(x[k], x[max_r]);
        }
        x[k] /= a[k][col];
        for (j = col + 1; j < var; j++)
            a[k][j] /= a[k][col];
        a[k][col] = 1;

        for (i = 0; i < equ; i++)
            if (i != k) {
                x[i]   = x[k] * a[i][col];
                for (j = col + 1; j < var; j++)
                    a[i][j]   = a[k][j] * a[i][col];
                a[i][col] = 0;
            }
    }
    return 1;
}
```

4.5 Inverse and Modulo of Combanation

```
// a*x=1(mod p) x=a^(p-2)
inline int qpow(long long a, int b) {
  int ans = 1;
  a = (a % p + p) % p;
  while(b) {
    if (b & 1) ans = (a * ans) % p;
    a = (a * a) % p;
    b>>=1;
  }
  return ans;
}
//linear time
inv[1] = 1;
for (int i = 2; i <= n; ++i) inv[i] = (long long)(p - p / i) * inv[p % i]
    % p;
// calculate n pick m mod p
//when n and m are not too big, ans=fact[n]*inv[m]*inv[n-m]%p
fact[0] = 1;
for (int i = 1; i < maxn; i++) {
    fact[i] = fact[i - 1] * i %mod;
}
inv[maxn - 1] = power(fact[maxn - 1], mod - 2);
for (int i = maxn - 2; i >= 0; i--) {
    inv[i] = inv[i + 1] * (i + 1) %mod;
}
// when n and m are big but p is small
long long Lucas(long long n, long long m, long long p) {
  if (m == 0) return 1;
  return (C(n % p, m % p, p) * Lucas(n / p, m / p, p)) % p;
}
```

4.6 Prime

```
//Euler sieve
int pri[MAXN],vis[MAXN],cnt=-1;
void init() {
    for (int i = 2; i < MAXN; ++i) {
        if (!vis[i]) pri[++cnt] = i;
        for (int j = 0; j < cnt; ++j) {
            if (1ll * i * pri[j] >= MAXN) break;
            vis[i * pri[j]] = 1;
            if (i % pri[j]==0) break;
        }
    }
}
```

5 String

5.1 KMP

```
vector<int> prefix_function(string s) {
  int n = (int)s.length();
  vector<int> pi(n);
  for (int i = 1; i < n; i++) {
    int j = pi[i - 1];
    while (j > 0 && s[i] != s[j]) j = pi[j - 1];
    if (s[i] == s[j]) j++;
    pi[i] = j;
  }
  return pi;
}
```

5.2 Trie

```
struct trie {
  int nex[100000][26], cnt;
  bool exist[100000];  // if this node is a end of a word

  void insert(char *s, int l) {
    int p = 0;
    for (int i = 0; i < l; i++) {
      int c = s[i] - 'a';
      if (!nex[p][c]) nex[p][c] = ++cnt;  // if the node DNE, add it
      p = nex[p][c];
    }
    exist[p] = 1;
  }
  bool find(char *s, int l) {
    int p = 0;
    for (int i = 0; i < l; i++) {
      int c = s[i] - 'a';
      if (!nex[p][c]) return 0;
      p = nex[p][c];
    }
    return exist[p];
  }
};
```

5.3 Z-function

```
// The Z-function for this string is an array of length n where the i-th
    element is equal to the greatest number of characters starting from
    the position i that coincide with the first characters of s.
```

```
// In other words, z[i] is the length of the longest common prefix between
     s and the suffix of s starting at i.

vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r) z[i] = min(r - i + 1, z[i - l]);
    while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

5.4 AC-Automaton

```
//given some words and a passage, find how many words appeared in the
    passage
#include<bits/stdc++.h>
#define forn(i, n) for (int i = 0; i < (int)(n); ++i)
#define ms(a, x) memset(a, x, sizeof(a))
#define endl '\n'
using namespace std;

const int N=1e6+5;
namespace AC{
  int tr[N][26],tot=0;
  int e[N],fail[N];
  queue<int> q;
  void ini(){
    ms(tr,0);
    ms(fail,0);
    ms(e,0);
    tot=0;
  }
  void insert(string s){
    int u=0;
    forn(i,s.length()){
      int c=s[i]-'a';
      if(tr[u][c]==0) tr[u][c]=++tot;
      u=tr[u][c];
    }
    e[u]++;
  }
  void build(){
    forn(i,26) if(tr[0][i]) q.push(tr[0][i]);
    while(!q.empty()){
      int u=q.front();
      q.pop();
      forn(i,26){
        if(tr[u][i]){
          fail[tr[u][i]]=tr[fail[u]][i];
          q.push(tr[u][i]);
        }
        else tr[u][i]=tr[fail[u]][i];
      }
    }
  }
  int query(string t){
    int u=0,res=0;
    forn(i,t.length()){
      u=tr[u][t[i]-'a'];
      for(int j=u;j&&e[j]!=-1;j=fail[j]){
        res+=e[j];
        e[j]=-1;
      }
    }
    return res;
  }
}
int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
    AC::ini();
    int n;
    cin>>n;
    string st;
    while(n--){
      cin>>st;
      AC::insert(st);
    }
    AC::build();
    cin>>st;
    cout<<AC::query(st)<<endl;
  return 0;
}
```

5.5 Manacher

```
vector<int> d1(n);
for (int i = 0, l = 0, r = -1; i < n; i++) {
  int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
  while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
    k++;
  }
  d1[i] = k--;
  if (i + k > r) {
    l = i - k;
    r = i + k;
  }
}
```