Decimal Array Expansion

locked

by mishraiiit

Editorial by mishraiiit

Let $Len[i][j]$ denote the length of the sequence formed after j times expanding a single digit i. Similarly, let $Sum[i][j]$ denote the sum of the sequence formed after j times expanding a single digit i.

We only need to compute these values these values for those i and j such that ($0 \le i \le 9$) and ($0 \le j \le 63$). Now, if replacement string of digit i is S_i. Then, we have the following reccurence for $Len[i][j]$ and $Sum[i][j]$:

$Len[i][j]$ = $\sum_{k=1}^{len(S_i)} Len[S_i[k]][j-1]$

$Sum[i][j]$ = $\sum_{k=1}^{len(S_i)} Sum[S_i[k]][j-1]$

And the base case:

$Len[i][0] = 1$

$Sum[i][0] = i$

Now, we compute how many times do we apply the expansion algorithm. Let's denote it by t. Now, we can iterate on t and find the minimum value of t, such that $\sum_{i=1}^{len(A)} Len[A_i][t] \ge m$.

Now, we compute a $prefixLen[i] and prefixSum[i]$, which would denote the length and sum of elements after t times expanding the subarray $[0, i]$ of A.

Now, we write a function $expansionSum(i)$ which would tell the sum till i-th position in the t times expanded sequence. Answer to a query $[l, r]$ would be $expansionSum(r) - expansionSum(l-1)$

For each $expansionSum(i)$, first we use binary search over prefix array in order to find the largest index i for which $prefixLen[i] \le r$ and add $prefixSum[i-1]$ to the result, next, we know that the remaining (if any) segment is produced from digit $A[i]$using t transformations. So we expand $A[i]$ using 1 transformation and solve the problem recursively for replacement rule for $A[i+1]$ and $k-1$ transformations. The replacement string has at most 9 elements, so during each recursive call we can iterate explicitly over it.

Overall complexity is O(n + q*(64*10 + logn)).

Statistics

Difficulty: Hard

Time Complexity:	O(n + q*(64*10 + logn))

Required Knowledge: Recursion, Dynamic Programming, Binary Search

Publish Date: Jan 23 2018

Set by mishraiiit

Problem Setter's code :

```
#include "bits/stdc++.h"
#define ll long long int
using namespace std;
struct ${$(){
    ios_base::sync_with_stdio(0);cin.tie(0);
    cout << fixed << setprecision(9);
}}$;

ll inf = 1000000000000000001;

ll add(ll u, ll v) {
    if(u == inf || v == inf)
        return inf;
    return min(inf, u + v);
}

const int N = 100001;
string A;
ll SZ[N];
ll SUM[N];
string S[10];
pair<ll, ll> M[10][64];
int expan = 0;
ll n, m, q;

void init() {
    n = m = q = 0;
    expan = 0;
    for(int i = 0; i < 10; i++)
        for(int j = 0; j < 64; j++)
            M[i][j] = make_pair(0, 0);
    for(int i = 0; i < 10; i++)
        S[i].clear();
    memset(SUM, 0, N * sizeof(ll));
    memset(SZ, 0, N * sizeof(ll));
    A = "";
}

ll getExpansion(ll digit, ll level, ll n) {
    if(n == 0) return 0;
    ll ans = 0;
    for(int i = 0; i < S[digit].size(); i++) {
        if(M[S[digit][i] - '0'][level - 1].first == n) {
            return ans + M[S[digit][i] - '0'][level - 1].second;
        } else if(M[S[digit][i] - '0'][level - 1].first > n) {
            return ans + getExpansion(S[digit][i] - '0', level - 1, n);
        } else {
            ans = add(ans, M[S[digit][i] - '0'][level - 1].second);
            n = n - M[S[digit][i] - '0'][level - 1].first;
        }
    }
    assert(0);
    return 0;
}

ll sum(ll id) {
    if(id == 0)
        return 0;
    int l = 0, r = n - 1;
    if(SZ[n - 1] == id)
        return SUM[n - 1];
    while(l != r) {
        int mid = (l + r) >> 1;
        if(SZ[mid] > id) {
            r = mid;
        } else {
            l = mid + 1;
        }
    }
    return SUM[l - 1] + getExpansion(A[l] - '0', expan, id - SZ[l - 1]);
}

int main() {
    init();
    cin >> n >> m >> q;

    cin >> A;

    for(int i = 0; i < 10; i++) {
        cin >> S[i];
    }

    for(int i = 0; i < 10; i++) {
        M[i][0] = {1, i};
    }


    for(int i = 1; i < 64; i++) {
        for(int j = 0; j < 10; j++) {
            for(int k = 0; k < S[j].size(); k++) {
                M[j][i].first = add(M[j][i].first, M[S[j][k] - '0'][i - 1].
first);
                M[j][i].second = add(M[j][i].second, M[S[j][k] - '0'][i - 1
].second);
            }
        }
    }

    while(1) {
        ll sz = 0;
        for(int i = 0; i < n; i++) {
            sz = add(sz, M[A[i] - '0'][expan].first);
        }
        if(sz >= m) break;
        else expan++;
    }

    for(int i = 0; i < n; i++) {
        SZ[i] = M[A[i] - '0'][expan].first;
        SUM[i] = M[A[i] - '0'][expan].second;
        if(i) {
            SZ[i] += SZ[i - 1];
            SUM[i] += SUM[i - 1];
        }
    }

    while(q--) {
        ll l, r;
        cin >> l >> r;
        cout << sum(r) - sum(l - 1) << endl;
    }

    return 0;
}
```

Tested by pkacprzak

Problem Tester's code :

```
//intended solution dp + recursion for each query, something like O(q * (64
*10 + log(n)))
#include <iostream>
#include <cstdio>
#include <string>
#include <sstream>
#include <vector>
#include <set>
#include <map>
#include <queue>
#include <stack>
#include <cmath>
#include <algorithm>
#include <cstring>
#include <ctime>
```

```
#include <cassert>
using namespace std;
#define pb push_back
#define mp make_pair
#define pii pair<int,int>
#define fi first
#define se second
#define vi vector<int>
#define vpii vector<pii>
#define SZ(x) ((int)(x.size()))
#define dbg cerr << "debug here" << endl;
#define info(vari) cerr << #vari<< " = "<< (vari) <<endl;

#define assertRange(x,a,b) assert((a) <= (x) and (x) <= (b))
typedef long long ll;
const int INF = 1e9;

const int N = 1e5;
const int Q = 1e5;
const ll M = 1e17;
const ll LEN = 9;

struct Block {
    vector<ll> c = vector<ll>(10,0);
    ll len = 0;
    ll sum = 0;
    void set(int idx, int val) {
        c[idx] = val;
        update();
    }
    Block applyRules(const vector<vector<int>>& rules) {
        Block res;
        for (int i = 0; i < 10; ++i) {
            for (auto e : rules[i]) {
                res.c[e] += c[i];
            }
        }
        res.update();
        return res;
    }
    void update() {
        len = 0;
        for (int i = 0; i < 10; ++i) {
            len += c[i];
        }
        sum = 0;
        for (int i = 0; i < 10; ++i) {
            sum += i*c[i];
        }
    }
};

Block f[10][70];
vector<ll> prefLen;
vector<ll> prefSum;

ll solve(const vector<int>& a, ll r, int k, const vector<vector<int>>& rule
s, int depth) {
    if (r == 0) return 0;
    ll res = 0;
    int i = 0;
    //cout << a.size() << " " << r << " " << k << endl;
    if (depth > 0) {
        while (f[a[i]][k].len <= r) {
            res += f[a[i]][k].sum;
            r -= f[a[i]][k].len;
            ++i;
            //cout << i << endl;
        }
    } else {

        /*
        while (prefLen[i] <= r) {
            ++i;
        }
        if (i > 0) {
            res += prefSum[i-1];
            r -= prefLen[i-1];
        }
        */
        auto it = upper_bound(prefLen.begin(), prefLen.end(), r);
        if (it == prefLen.begin()) {
            i = 0;
        } else {
            i = distance(prefLen.begin(), it);
            res += prefSum[i-1];
            r -= prefLen[i-1];
        }
    }
    if (k == 0) return res;
    res += solve(rules[a[i]], r, k-1, rules, depth+1);
    return res;
}
int main() {
    ios_base::sync_with_stdio(0);
    int n, q;
    ll m;
    cin >> n >> m >> q;
    assertRange(n, 1, N);
    assertRange(q, 1, Q);
    assertRange(m, 1, M);
    vector<int> a(n);
    for (int i = 0; i < n; ++i) {
        int d;
        cin >> d;
        assertRange(d, 0, 9);
        a[i] = d;
    }
    vector<vector<int>> rules(10);
    for (int i = 0; i < 10; ++i) {
        int len;
        cin >> len;
        assertRange(len, 2, LEN);
        for (int j = 0; j < len; ++j) {
            int x;
            cin >> x;
            assertRange(x, 0, 9);
            rules[i].pb(x);
        }
    }
    for (int i = 0; i < 10; ++i) {
        f[i][0].set(i, 1);
        for (int j = 1; f[i][j-1].len <= 1e17; ++j) {
            f[i][j] = f[i][j-1].applyRules(rules);
        }
    }

    int k = 0;
    {
        ll totalLen = n;
        vector<Block> blocks(n);
        for (int i = 0; i < n; ++i) {
            blocks[i].set(a[i], 1);
        }
        while (totalLen < m) {
            ++k;
            totalLen = 0;
            for (int i = 0; i < n; ++i) {
                blocks[i] = blocks[i].applyRules(rules);
                totalLen += blocks[i].len;
            }
        }
    }
    prefLen = vector<ll>(n);
    prefSum = vector<ll>(n);
    for (int i = 0; i < n; ++i) {
        if (i > 0) {
            prefLen[i] += prefLen[i-1];
            prefSum[i] += prefSum[i-1];
        }
        prefLen[i] += f[a[i]][k].len;
        prefSum[i] += f[a[i]][k].sum;
    }
    for (int qid = 0; qid < q; ++qid) {
        //info(qid);
        ll l, r;
        cin >> l >> r;
        assertRange(l, 1, r);
        assertRange(r, l, m);
        ll res = solve(a, r, k, rules, 0);
        if (l > 1) {
            res -= solve(a, l-1, k, rules, 0);
        }
        cout << res << endl;
    }
    return 0;
}
```