Team notebook

Nicolas Alba

December 16, 2023

Contents

1 KMP$_A$*utomaton*

```
const int MAXN = 1e5 + 5, alpha = 26;
const char L = 'A';
int aut[MAXN][alpha]; //aut[i][j] = a donde vuelvo si estoy en i y pongo
    una j

// Match es cuando llegas a s.size()
// O(MAXN * alpha)
void build(string &s){
       // do not forget to clean if reuse use:
       // memset(aut, 0, sizeof aut);
       int lps = 0;
       aut[0][s[0]-L] = 1;
       int n = s.size();
       for(int i = 1; i < n+1; i++){
              for(int j = 0; j < alpha; j++) aut[i][j] = aut[lps][j];
              if(i < n){
                     aut[i][s[i]-L] = i + 1;
                     lps = aut[lps][s[i]-L];
              }
       }
}
```

2 Line Container Double

```
// Same as Line Container but for double, and special case with the index.
struct Line {
       mutable ld k, m, p;
       ll idx; // for having index aditionally
       bool operator<(const Line& o) const { return k < o.k; }
       bool operator<(ld x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
       // (for doubles, use inf = 1/.0, div(a,b) = a/b)
       const ld inf = 1/.0;

       ld div(ld a, ld b) { // floored division
              return a / b; }
       bool isect(iterator x, iterator y) {
              if (y == end()) { x->p = inf; return false; }
              if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
              else x->p = div(y->m - x->m, x->k - y->k);
              return x->p >= y->p;
```

```
        }
        void add(ld k, ld m, ll idx) {
                auto z = insert({k, m, 0, idx}), y = z++, x = y;
                while (isect(y, z)) z = erase(z);
                if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
                while ((y = x) != begin() && (--x)->p >= y->p)
                        isect(x, erase(y));
        }
        ld query(ld x, ll idx) {
                assert(!empty());
                auto it = lower_bound(x);
                if (idx == it->idx) {
                    // if you need especial case with the index
                    //return -1;
                }
                auto l = *it;
                return l.k * x + l.m;
        }
};
```

3 Max K Freq Subarray

```
// Find the max sub array that has same freq
// Of elements from 1 to K.
// nums <= 4*10^5
// 1 <= nums[i] <= k, 4*10^5

/*
n:6 k:2
nums:2 2 1 1 2 2
ans: 4
*/

#define bint __int128
ll findMaxFreqSubarray(ll n, ll k, vl nums) {
    bint MOD=212345678987654321LL; // prime
    bint  PI=1e9 + 7; // prime
    vector<bint> pows(k+1, 1);
    for (int i = 0; i < k; i++) {
        pows[i+1] = (pows[i] * PI) % MOD;
    }
    bint oneHash = 0;
    for (int i = 0; i < k; i++) {
        oneHash += pows[i];
        oneHash %= MOD;
    }
    vector<bint> hashes = {0}; // hashes with same freq
    for (int i = 0; i <= n/k; i++) {
        hashes.pb((hashes.back() + oneHash) % MOD);
    }
    map<bint,ll> prefixes;
    prefixes[0] = -1;
    bint actual = 0;

    set<pair<ll,ll>> freqs;
    for (int i = 1; i <= k; i++) {
        freqs.insert({0, i});
    }
    vl cnt(k+1);
    ll ans = 0;
    for (int i =0; i < n; i++) { // n
        freqs.erase({cnt[nums[i]], nums[i]});
        cnt[nums[i]]++;
        freqs.insert({cnt[nums[i]], nums[i]});
        ll mn = freqs.begin()->F;
        actual = (actual + pows[nums[i]-1]) % MOD;
        bint needed = (actual - hashes[mn] + MOD) % MOD;
        if (prefixes.count(needed)) {
            ans = max(ans, i - prefixes[needed]);
        }
        if (prefixes.count(needed)) continue;
        prefixes[needed] = i;
    }
    return ans;
}
```

4 Point in Convex Polygon

```
ll IN = 0;
ll ON = 1;
ll OUT = 2;
vector<string> ANS = {"IN", "ON", "OUT"};

#define pt pair<ll,ll>
```

```
#define x first
#define y second

pt sub(pt a, pt b) { return {a.x - b.x, a.y - b.y}; }
ll cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x = 180 -> sin = 0
ll orient(pt a, pt b, pt c) { return cross(sub(b,a),sub(c,a)); }//
    clockwise = -

// poly is in clock wise order
// Returns if the query point is IN, ON, or OUT the convex polygon
// O(log(poly.size()))
ll insidePoly(vector<pt> &poly, pt query) {
    ll n = poly.size();
    ll left = 1;
    ll right = n - 2;
    ll ans = -1;
    if (!(orient(poly[0], poly[1], query) <= 0
         && orient(poly[0], poly[n-1], query) >= 0)) {
        return OUT;
    }
    while (left <= right) {
        ll mid = (left + right) / 2;
        if (orient(poly[0], poly[mid], query) <= 0) {
            left = mid + 1;
            ans = mid;
        } else {
            right = mid - 1;
        }
    }
    left = ans;
    right = ans + 1;
    if (orient(poly[left], query, poly[right]) < 0) {
        return OUT;
    }
    if (orient(poly[left], poly[right], query) == 0
       || (left == 1 && orient(poly[0], poly[left], query) == 0)
       || (right == n-1 && orient(poly[0], poly[right], query) == 0)) {
        return ON;
    }
    return IN;
}
```

5 Sum*subStringOccurrencesFrom1ToN*

```
/*
You are given a string S consisting of digits and positive integers L and
    R for each of T test cases. Solve the following problem.

For a positive integer x, let us define f(x) as the number of contiguous
    substrings of the decimal representation of
x (without leading zeros) that equal S.
Find L <= K <= R
 f (k).
*/

// Copy kmp automaton
ll dp[20][20][20][2][2];

ll ff(ll pos, ll posAut, ll matches, bool free, bool anyNumber) {
    if (!anyNumber) {
        matches = 0;
        posAut = 0;
    }
    ll match = anyNumber ? p.size() == posAut : 0;
    if (pos == t.size()) {
        if (anyNumber) {
            return matches + match;
        } else {
            return 0;
        }
    }
    if (dp[pos][posAut][matches][free][anyNumber] != -1) {
        return dp[pos][posAut][matches][free][anyNumber];
    }
    ll ans = 0;
    for (char c = '0'; c <= '9'; c++) {
        if (!free && c > t[pos]) break;
        ans += ff(pos + 1, aut[posAut][c - L], matches + match, free || c
            < t[pos], anyNumber || c != '0');
    }
    return dp[pos][posAut][matches][free][anyNumber] = ans;
}

ll f(string s) {
    t = s;
    memset(dp, -1, sizeof dp);
```

```
    return ff(0, 0, 0, 0, 0);
}
```

6 Xor Trie

```
// Find given an array of inserted nums into the trie
// It could find the max (someNum ^ someInsertedNum)
// Take into account the maximum position of the bit
const int start = 30;
struct Node {
    Node *left = nullptr;
    Node *right = nullptr;

    void insert(ll num, ll bit=start) {
        if (bit == -1) return;
        Node *&next = (num >> bit) & 1 ? right : left;
        if (next == nullptr) next = new Node();
        next->insert(num, bit-1);
    }

    ll max(ll num, ll bit=start) {
        if (bit == -1) return 0;
        Node *&next = (num >> bit) & 1 ? left : right;
        Node *&other = (next == left) ? right : left;
        if (next != nullptr) return (1 << bit) + next->max(num, bit-1);
        return other->max(num, bit-1);
    }
};
```

7 gcc

```
g++ -std=c++11 -O2 -Wall test.cpp -o test

-Wall -Wextra -O2

-pedantic warns about non-standard C++ language extensions.
-Wshadow    warns if a declared name shadows the same name in some outer
    level. For example, this will cause a warning:
int n;
void solve()
{
    // Solve the problem
}
int main()
{
    int n; cin >> n;
    solve();
}
-Wformat=2    warns if an argument type in printf()/scanf() does not
    correspond to the format string. This is partially enabled by -Wall,
    but -Wformat=2 is more strict.

-Wfloat-equal    warns if two floating point values are compared
    directly: a == b. Usually the correct way is: fabs(a - b) < eps.

-Wconversion    warns if data can be lost in an implicit conversion. Most
    often it is accidental assignment of a long long value to an int
    variable. I have this warning enabled since I failed a problem by
    writing pair<int, int> instead of pair<int, long long> :)

   An explicit cast (for example, (double)my_long_long_var) will not
    trigger this warning.

-Wlogical-op    warns about logical operators in places where GCC expects
    bitwise operators.

-Wshift-overflow=2    warns about left shift overflows (GCC 6+).

-Wduplicated-cond    warns about repeated conditions in if () else if ()
    (GCC 6+).

There are also -Wcast-qual  -Wcast-align, but they are less useful
    (though don't hurt). You can read more about GCC warnings here:
    https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html
```