

TCR

December 18, 2016

git diff solution (Jens Heuseveldt, Ludo Pulles, Peter Ypma)

vim ~/.vimrc

```

1 set nu sw=4 ts=4 noet ai hls
2 syntax on
3 colorscheme slate

```

template.cpp

```

1 #include<bits/stdc++.h>
2
3 #define x first
4 #define y second
5
6 using namespace std;
7
8 typedef long long ll;
9 typedef pair<int, int> pii;
10 typedef pair<ll, ll> pll;
11 typedef vector<int> vi;
12
13 const int INF = 2147483647; // (1 << 30) - 1 + (1 << 30)
14 const ll LLINF = 9223372036854775807LL; // (1LL << 62) - 1 + (1LL << 62)
15 const double pi = acos(-1.0);
16
17 // lambda-expression: [] (args) -> retType { body }
18
19 const bool LOG = false;
20 void Log() { if(LOG) cerr << "\n\n"; }
21 template<class T, class... S>
22 void Log(T t, S... s) {
23     if(LOG) cerr << t << "\t", Log(s...);
24 }
25
26 template<class T1, class T2>
27 ostream& operator<<(ostream& out, const pair<T1,T2> &p) {
28     return out << '(' << p.first << ", " << p.second << ')';
29 }
30
31
32 template<typename T1, typename T2>
33 ostream& operator<<(ostream &out, pair<T1, T2> p) {
34     return out << "(" << p.first << ", " << p.second << ")";
35 }
36
37 template<class T>
38 using min_queue = priority_queue<T, vector<T>, greater<T>>;
39
40 // Order Statistics Tree (if this is supported by the judge software)
41 #include <ext/pb_ds/assoc_container.hpp>
42 #include <ext/pb_ds/tree_policy.hpp>
43 using namespace __gnu_pbds;
44 template<class TIn, class TOut> // key, value types. TOut can be null_type
45 using order_tree = tree<TIn, TOut, less<TIn>,
46     rb_tree_tag, tree_order_statistics_node_update>;
47 // find_by_order(int r) (0-based)
48 // order_of_key(TIn v)
49 // use key pair<TIn,int> {value, counter} for multiset/multimap
50
51 int main() {
52     ios_base::sync_with_stdio(false); // faster IO
53     cin.tie(NULL); // faster IO

```

```

54     cerr << boolalpha;                // (print true/false)
55     (cout << fixed).precision(10);    // set floating point precision
56     // TODO: code
57     return 0;
58 }

```

Prime numbers: 982451653 , 81253449 , $10^3 + \{-9, -3, 9, 13\}$, $10^6 + \{-17, 3, 33\}$, $10^9 + \{7, 9, 21, 33, 87\}$

0.1 De winnende aanpak

- Goed slapen & een vroeg ritme hebben
- Genoeg drinken & eten tijdens de wedstrijd
- Een lijst van alle problemen met info waar het over gaat, en wie het goed kan oplossen
- Ludo moet de opgave goed lezen
- Test de kleine voorbeeldgevallen
- Houd na 2 uur een pauze en overleg waar iedereen mee bezig is
- Maak zelf wat test-cases
- Na een WA, print het probleem, en probeer het ook weg te leggen
- Typ de dingen uit de TCR, die je zeker nodig hebt, alvast in
- Peter moet meer papier gebruiken om fouten te voorkomen
- Als iemand niks te doen heeft, kan hij nodige dingen uit de TCR typen.
- We moeten ook een voorbeeld test-case voor TCR algoritmes hebben om te testen of het goed overgetypt is
- Bij geometrie moeten we om kunnen gaan met meerdere input manieren (voor bv. lijnen)
- Gebruik veel long long's
- Bij een verkeerd antwoord, kijk naar genoeg debug output

0.2 Detecting overflow

These are GNU builtins, detect both over- and underflow. Returns a boolean upon failure, otherwise the result is present in ref. Follow the template:

```
_builtin_[u|s][add|mul|sub](ll)?_overflow(in, out, &ref)
```

0.3 Wrong Answer

- Edge cases: $n \in \{-1, 0, 1, 2\}$. Empty list/graph?
- Beware of typos
- Test sample input; make custom testcases
- Read carefully
- Check bounds (use long long or long double)
- Off by one error (in indices or loop bounds)
- Not enough precision
- Assertions
- Missing modulo operators
- Cases that need a (completely) different approach

0.4 Covering problems

Minimum edge cover \iff Maximum independent set

Matching A set of edges without common vertices (*Maximum is the **largest** such set, maximal is a set which you cannot add more edges to without breaking the property*).

Minimum Vertex Cover A set vertices (cover) such that each edge in the graph is incident to at least one vertex of the set.

Minimum Edge Cover A set of edges (cover) such that every vertex is incident to at least one edge of the set.

Maximum Independent Set A set of vertices in a graph such that no two of them are adjacent.

König's theorem In any bipartite graph, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover

A useful identity: $\bigoplus_{x=0}^{a-1} x = \{0, a-1, 1, a\}[a \bmod 4]$.

1 Mathematics

```

1
2 int abs(int x) { return x > 0 ? x : -x; }
3 int sign(int x) { return (x > 0) - (x < 0); }
4
5 // greatest common divisor
6 ll gcd(ll a, ll b) { while (b) { a %= b; swap(a, b); } return a };
7 // least common multiple
8 ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
9 ll mod(ll a, ll m) { return ((a % b) + b) % b; }
10
11 // safe multiplication (ab % m) for m <= 4e18 in O(log b)
12 ll modmul(ll a, ll b, ll m) {
13     ll r = 0;
14     while (b) {
15         if (b & 1) r = mod(r + a, m);
16         a = mod(a + a, m);
17         b >>= 1;
18     }
19     return r;
20 }
21
22 // safe exponentiation (a^b % m) for m <= 2e9 in O(log b)
23 ll modpow(ll a, ll b, ll m) {
24     ll r = 1;
25     while (b) {
26         if (b & 1) r = (r * a) % m;
27         a = (a * a) % m;
28         b >>= 1;
29     }
30     return r;
31 }
32
33 // returns x, y such that ax + by = gcd(a, b)
34 ll egcd(ll a, ll b, ll &x, ll &y)
35 {
36     ll xx = y = 0, yy = x = 1;
37     while (b) {
38         x -= a / b * xx; swap(x, xx);
39         y -= a / b * yy; swap(y, yy);
40         a %= b; swap(a, b);
41     }
42     return a;
43 }
44
45 // Chinese remainder theorem

```

```

46 const pll NO_SOLUTION(0, -1);
47 // Returns (u, v) such that x = u % v <=> x = a % n and x = b % m
48 pll crt(ll a, ll n, ll b, ll m)
49 {
50     ll s, t, d = egcd(n, m, s, t), nm = n * m;
51     if (mod(a - b, d)) return NO_SOLUTION;
52     return pll(mod(s * b * n + t * a * m, nm) / d, nm / d);
53     /* when n, m > 10^6, avoid overflow:
54     return pll(mod(modmul(modmul(s, b, nm), n, nm) +
55                     modmul(modmul(t, a, nm), m, nm), nm) / d, nm / d); */
56 }
57
58 int phi[N]; // phi[i] = #{ j | gcd(i, j) = 1 }
59
60 void sievePhi() {
61     for (int i = 0; i < N; i++) phi[i] = i;
62     for (int i = 2; i < N; i++)
63         if (phi[i] == i)
64             for (int j = i; j < N; j += i)
65                 phi[j] -= phi[j] / i * (i - 1);
66 }
67
68 // calculate nCk % p (p prime!)
69 ll lucas(ll n, ll k, ll p) {
70     ll ans = 1;
71     while (n) {
72         ll np = n % p, kp = k % p;
73         if (np < kp) return 0;
74         ans = mod(ans * binom(np, kp), p); // (np C kp)
75         n /= p; k /= p;
76     }
77     return ans;
78 }

```

2 Datastructures

2.1 Segment tree $\mathcal{O}(\log n)$

```

1 typedef /* Tree element */ S;
2 const int n = 1 << 20;
3 S t[2 * n];
4
5 // sum segment tree
6 S combine(S l, S r) { return l + r; }
7 // max segment tree
8 S combine(S l, S r) { return max(l, r); }
9
10 void build() {
11     for (int i = n; --i > 0; )
12         t[i] = combine(t[2 * i], t[2 * i + 1]);
13 }
14
15 // set value v on position p
16 void update(int p, int v) {
17     for (t[p += n] = v; p /= 2; )
18         t[p] = combine(t[2 * p], t[2 * p + 1]);
19 }
20
21 // sum on interval [l, r)
22 S query(int l, int r) {
23     S resL, resR;
24     for (l += n, r += n; l < r; l /= 2, r /= 2) {
25         if (l & 1) resL = combine(resL, t[l++]);
26         if (r & 1) resR = combine(t[--r], resR);
27     }

```

```

28     return combine(resL, resR);
29 }

```

2.2 Binary Indexed Tree $\mathcal{O}(\log n)$

Use one-based indices!

```

1 int bit[MAXN];
2
3 // arr[idx] += val
4 void update(int idx, int val) {
5     while (idx < MAXN) bit[idx] += val, idx += idx & -idx;
6 }
7
8 // returns sum of arr[i], where i: [1, idx]
9 int query(int idx) {
10     int ret = 0;
11     while (idx) ret += bit[idx], idx -= idx & -idx;
12     return ret;
13 }

```

2.3 Trie

```

1 const int SIGMA = 26;
2
3 struct trie {
4     bool word;
5     trie **child;
6
7     trie() : word(false), child(new trie*[SIGMA]) {
8         for (int i = 0; i < SIGMA; i++) child[i] = NULL;
9     }
10
11     void addWord(const string &str)
12     {
13         trie *cur = this;
14         for (char ch : str) {
15             int idx = ch - 'a';
16             if (!cur->child[idx]) cur->child[idx] = new trie();
17             cur = cur->child[idx];
18         }
19         cur->word = true;
20     }
21
22     bool isWord(const string &str)
23     {
24         trie *cur = this;
25         for (char ch : str) {
26             int idx = ch - 'a';
27             if (!cur->child[idx]) return false;
28             cur = cur->child[idx];
29         }
30         return cur->word;
31     }
32 };

```

2.4 Disjoint-Set / Union-Find $\mathcal{O}(\alpha(n))$

```

1 int par[MAXN], rnk[MAXN];
2
3 void uf_init(int n) {
4     fill_n(par, n, -1);

```

```

5     fill_n(rnk, n, 0);
6 }
7
8 int uf_find(int v) {
9     return par[v] < 0 ? v : par[v] = uf_find(par[v]);
10 }
11
12 void uf_union(int a, int b) {
13     if ((a = uf_find(a)) == (b = uf_find(b))) return;
14     if (rnk[a] < rnk[b]) swap(a, b);
15     if (rnk[a] == rnk[b]) rnk[a]++;
16     par[b] = a;
17 }

```

3 Graph Algorithms

3.1 Maximum matching $\mathcal{O}(nm)$

This problem could be solved with a flow algorithm like Dinic's algorithm which runs in $\mathcal{O}(\sqrt{V}E)$, too.

```

1 const int nodesLeft = 1e4, nodesRight = 1e4;
2 bool vis[nodesRight]; // vis[rightnodes]
3 int par[nodesRight]; // par[rightnode] = leftnode
4 vector<int> adj[nodesLeft]; // adj[leftnode][i] = rightnode
5
6 bool match(int cur) {
7     for (int nxt : adj[cur]) {
8         if (vis[nxt]) continue;
9         vis[nxt] = true;
10        if (par[nxt] == -1 || match(par[nxt])) {
11            par[nxt] = cur;
12            return true;
13        }
14    }
15    return false;
16 }
17
18 // perfect matching iff matches == nodesLeft && matches == nodesRight
19 int maxmatch() {
20     int matches = 0;
21     for (int i = 0; i < nodesLeft; i++) {
22         fill_n(vis, nodesRight, false);
23         if (match(i)) matches++;
24     }
25     return matches;
26 }

```

3.2 Strongly Connected Components $\mathcal{O}(V + E)$

```

1 vector<vi> adj; // adjacency matrix
2 vi index, lowlink; // lowest index reachable
3 stack<int> tarjanStack;
4 vector<bool> inStack; // true iff in tarjanStack
5 int newId; // ordering in DFS
6 vector<vi> scc; // Output: collection of vertex sets
7
8 void tarjan(int v) {
9     index[v] = lowlink[v] = newId++;
10    tarjanStack.push(v);
11    inStack[v] = true;
12    for (int w : adj[v]) {
13        if (index[w] == 0) {
14            tarjan(w);

```

```

15         lowlink[v] = min(lowlink[v], lowlink[w]);
16     } else if (inStack[w]) {
17         lowlink[v] = min(lowlink[v], index[w]);
18     }
19 }
20
21 if (lowlink[v] == index[v]) {
22     scc.push_back(vi());
23     int w;
24     do {
25         w = tarjanStack.top();
26         scc.back().push_back(w);
27         inStack[w] = false;
28         tarjanStack.pop();
29     } while (w != v);
30 }
31 }
32
33 int findSCC() {
34     newId = 1;
35     index.clear(); index.resize(n + 1, 0);
36     lowlink.clear(); lowlink.resize(n + 1, 0);
37     inStack.clear(); inStack.resize(n + 1, false);
38     while (!tarjanStack.empty()) tarjanStack.pop();
39     scc.clear();
40
41     for (int i = 0; i < n; i++) {
42         if (index[i] == 0) tarjan(i);
43     }
44     return scc.size();
45 }

```

3.3 Shortest path

3.3.1 Floyd-Warshall $\mathcal{O}(V^3)$

```

1 int n = 100, d[MAXN][MAXN];
2 for (int i = 0; i < n; i++) fill_n(d[i], n, INF / 3);
3 // set direct distances from i to j in d[i][j] (and d[j][i])
4 for (int i = 0; i < n; i++)
5     for (int j = 0; j < n; j++)
6         for (int k = 0; k < n; k++)
7             d[j][k] = min(d[j][k], d[j][i] + d[i][k]);

```

3.3.2 Bellman Ford $\mathcal{O}(VE)$

This is only useful if there are edges with weight $w_{i,j} < 0$ in the graph.

```

1 vector< pair<pii,int> > edges; // ((from, to), weight)
2 vector<int> dist(MAXN);
3
4 bool bellman_ford(int source) {
5     for (int i = 0; i < MAXN; i++) dist[i] = INF / 3;
6     dist[source] = 0;
7
8     bool updated;
9     int loops = 0;
10    do {
11        updated = false;
12        for (auto e : edges) {
13            int alt = dist[e.first.first] + e.second;
14            if (alt < dist[e.first.second]) {
15                dist[e.first.second] = alt;
16                updated = true;
17            }
18        }
19    } while (updated);
20 }

```

```

18         // if undirected graph:
19         alt = dist[e.first.second] + e.second;
20         if (UNDIRECTED && alt < dist[e.first.first]) {
21             dist[e.first.first] = alt;
22             updated = true;
23         }
24     }
25 } while(updated && loops < n);
26 return loops < n; // loops >= n: negative cycles
27 }

```

3.4 Max-flow min-cut

3.4.1 Dinic's Algorithm $\mathcal{O}(V^2E)$

Let's hope this algorithm works correctly! ...

```

1 // http://www.slideshare.net/KuoEO/acmicpc-dinics-algorithm
2 struct edge {
3     int to, rev;
4     ll cap, flow;
5     edge(int t, int r, ll c) : to(t), rev(r), cap(c), flow(0) {}
6 };
7
8 int s, t, level[MAXN]; // s = source, t = sink
9 vector<edge> g[MAXN];
10
11 bool dinic_bfs() {
12     fill_n(level, MAXN, 0);
13     level[s] = 1;
14
15     queue<int> q;
16     q.push(s);
17     while (!q.empty()) {
18         int cur = q.front();
19         q.pop();
20         for (edge e : g[cur]) {
21             if (level[e.to] == 0 && e.flow < e.cap) {
22                 level[e.to] = level[cur] + 1;
23                 q.push(e.to);
24             }
25         }
26     }
27     return level[t] != 0;
28 }
29
30 ll dinic_dfs(int cur, ll maxf) {
31     if (cur == t) return maxf;
32
33     ll f = 0;
34     bool isSat = true;
35     for (edge &e : g[cur]) {
36         if (level[e.to] != level[cur] + 1 || e.flow >= e.cap)
37             continue;
38         ll df = dinic_dfs(e.to, min(maxf - f, e.cap - e.flow));
39         f += df;
40         e.flow += df;
41         g[e.to][e.rev].flow -= df;
42         isSat &= e.flow == e.cap;
43         if (maxf == f) break;
44     }
45     if (isSat) level[cur] = 0;
46     return f;
47 }
48
49 ll dinic_maxflow() {
50     ll f = 0;

```

```

51     while (dinic_bfs()) f += dinic_dfs(s, LLINF);
52     return f;
53 }
54
55 void add_edge(int fr, int to, ll cap) {
56     g[fr].push_back(edge(to, g[to].size(), cap));
57     g[to].push_back(edge(fr, g[fr].size() - 1, 0));
58 }

```

3.5 Min-cost max-flow

Find the cheapest possible way of sending a certain amount of flow through a flow network.

```

1  struct edge {
2      // to, rev, flow, capacity, weight
3      int t, r;
4      ll f, c, w;
5      edge(int _t, int _r, ll _c, ll _w) : t(_t), r(_r), f(0), c(_c), w(_w) {}
6  };
7
8  int n, par[MAXN];
9  vector<edge> adj[MAXN];
10 ll dist[MAXN];
11
12 bool findPath(int s, int t)
13 {
14     fill_n(dist, n, LLINF);
15     fill_n(par, n, -1);
16
17     priority_queue<pii, vector<pii>, greater<pii> > q;
18     q.push(pii(dist[s] = 0, s));
19
20     while (!q.empty()) {
21         int d = q.top().first, v = q.top().second;
22         q.pop();
23         if (d > dist[v]) continue;
24
25         for (edge e : adj[v]) {
26             if (e.f < e.c && d + e.w < dist[e.t]) {
27                 q.push(pii(dist[e.t] = d + e.w, e.t));
28                 par[e.t] = e.r;
29             }
30         }
31     }
32     return dist[t] < INF;
33 }
34
35 pair<ll, ll> minCostMaxFlow(int s, int t)
36 {
37     ll cost = 0, flow = 0;
38     while (findPath(s, t)) {
39         ll f = INF, c = 0;
40         int cur = t;
41         while (cur != s) {
42             const edge &rev = adj[cur][par[cur]], &e = adj[rev.t][rev.r];
43             f = min(f, e.c - e.f);
44             cur = rev.t;
45         }
46         cur = t;
47         while (cur != s) {
48             edge &rev = adj[cur][par[cur]], &e = adj[rev.t][rev.r];
49             c += e.w;
50             e.f += f;
51             rev.f -= f;
52             cur = rev.t;
53         }
54         cost += f * c;

```

```

55     flow += f;
56 }
57 return pair<ll, ll>(cost, flow);
58 }
59
60 inline void addEdge(int from, int to, ll cap, ll weight)
61 {
62     adj[from].push_back(edge(to, adj[to].size(), cap, weight));
63     adj[to].push_back(edge(from, adj[from].size() - 1, 0, -weight));
64 }

```

3.6 Minimal Spanning Tree

3.6.1 Kruskal $\mathcal{O}(E \log V)$

```

1 struct edge {
2     int x, y, s;
3     void read() { cin >> x >> y >> s; }
4 };
5
6 edge edges[MAXM];
7
8 int kruskal(int n, int m) {
9     uf_init(n);
10    sort(edges, edges + m, [] (const edge &a, const edge &b)
11        -> bool { return a.s > b.s; });
12    ll ret = 0;
13    while (m--) {
14        if (uf_find(edges[m].x) != uf_find(edges[m].y)) {
15            ret += edges[m].s;
16            uf_union(edges[m].x, edges[m].y);
17        }
18    }
19    return ret;
20 }

```

4 String algorithms

4.1 Z-algorithm $\mathcal{O}(n)$

```

1 // z[i] = length of longest substring starting from s[i],
2 // which is also a prefix of s.
3 vector<int> z_function(const string &s) {
4     int n = (int) s.length();
5     vector<int> z(n);
6     for (int i = 1, l = 0, r = 0; i < n; ++i) {
7         if (i <= r)
8             z[i] = min(r - i + 1, z[i - l]);
9         while (i + z[i] < n && s[z[i]] == s[i + z[i]])
10             ++z[i];
11         if (i + z[i] - 1 > r)
12             l = i, r = i + z[i] - 1;
13     }
14     return z;
15 }

```

4.2 Suffix array $\mathcal{O}(n \log^2 n)$

This creates an array $P[0], P[1], \dots, P[n-1]$ such that the suffix $S[i \dots n]$ is the $P[i]^{th}$ suffix of S when lexicographically sorted.

```

1 #define fst first.first
2 #define snd first.second
3
4 typedef pair<int, int> pii;
5 typedef pair<pii, int> tii;
6
7 const int maxlogn = 17, int maxn = 1 << maxlogn;
8
9 tii make_triple(int a, int b, int c) { return tii(pii(a, b), c); }
10
11 int p[maxlogn + 1][maxn];
12 tii L[maxn];
13
14 void suffixArray(string S)
15 {
16     int N = S.size(), stp = 1, cnt = 1;
17     for (int i = 0; i < N; i++) p[0][i] = S[i];
18     for (; cnt < N; stp++, cnt <= 1) {
19         for (int i = 0; i < N; i++) {
20             L[i] = make_triple(p[stp-1][i], i + cnt < N ? p[stp-1][i + cnt] : -1, i);
21         }
22         sort(L, L + N);
23         for (int i = 0; i < N; i++) {
24             p[stp][L[i].second] = i > 0 && L[i].first == L[i-1].first
25                 ? p[stp][L[i-1].second] : i;
26         }
27     }
28     // result is in p[stp - 1][0 .. (N - 1)]
29 }

```

4.3 Longest Common Subsequence $\mathcal{O}(n^2)$

SUBSTRING: *consecutive characters*!!!

```

1 int table[STR_SIZE][STR_SIZE]; // DP problem
2
3 int lcs(const string &w1, const string &w2) {
4     int n1 = w1.size(), n2 = w2.size();
5     for (int i = 0; i <= n1; i++) table[i][0] = 0;
6     for (int j = 0; j <= n2; j++) table[0][j] = 0;
7
8     for (int i = 1; i < n1; i++) {
9         for (int j = 1; j < n2; j++) {
10             table[i][j] = w1[i - 1] == w2[j - 1] ?
11                 (table[i - 1][j - 1] + 1) :
12                 max(table[i - 1][j], table[i][j - 1]);
13         }
14     }
15     return table[n1][n2];
16 }
17
18 // backtrace
19 string getLCS(const string &w1, const string &w2) {
20     int i = w1.size(), j = w2.size();
21     string ret = "";
22     while (i > 0 && j > 0) {
23         if (w1[i - 1] == w2[j - 1]) ret += w1[--i], j--;
24         else if (table[i][j - 1] > table[i - 1][j]) j--;
25         else i--;
26     }
27     reverse(ret.begin(), ret.end());
28     return ret;
29 }

```

4.4 Levenshtein Distance $\mathcal{O}(n^2)$

```

1 int costs[MAX_SIZE][MAX_SIZE]; // DP problem
2
3 int levDist(const string &w1, const string &w2) {
4     int n1 = w1.size(), n2 = w2.size();
5     for (int i = 0; i <= n1; i++) costs[i][0] = i; // removal
6     for (int j = 0; j <= n2; j++) costs[0][j] = j; // insertion
7     for (int i = 1; i <= n1; i++) {
8         for (int j = 1; j <= n2; j++) {
9             costs[i][j] = min(
10                 min(costs[i - 1][j] + 1, costs[i][j - 1] + 1),
11                 costs[i - 1][j - 1] + (w1[i - 1] != w2[j - 1])
12             );
13         }
14     }
15     return costs[n1][n2];
16 }

```

4.5 Knuth-Morris-Pratt algorithm $\mathcal{O}(N + M)$

```

1 int kmp_search(const string &word, const string &text) {
2     int n = word.size();
3     vector<int> table(n + 1, 0);
4     for (int i = 1, j = 0; i < n; ) {
5         if (word[i] == word[j]) {
6             table[++i] = ++j;
7         } else if (j > 0) {
8             j = table[j];
9         } else i++;
10    }
11    int matches = 0;
12    for (int i = 0, j = 0; i < text.size(); ) {
13        if (text[i] == word[j]) {
14            i++;
15            if (++j == n) {
16                matches++;
17                // match at interval [i - j, i)
18                j = table[j];
19            }
20        } else if (j > 0) j = table[j];
21        else i++;
22    }
23    return matches;
24 }

```

4.6 Aho-Corasick Algorithm $\mathcal{O}(N + \sum_{i=1}^m |S_i|)$

All given patterns must be unique!

```

1
2 const int MAXP = 100, MAXLEN = 200, SIGMA = 26, MAXTRIE = MAXP * MAXLEN;
3
4 int npatterns;
5 string patterns[MAXP], S;
6
7 int wordIdx[MAXTRIE], to[MAXTRIE][SIGMA], sLink[MAXTRIE], dLink[MAXTRIE], nnodes;
8
9 void ahoCorasick()
10 {
11     // 1. Make a tree, 2. create sLinks and dLinks, 3. Walk through S
12
13     fill_n(wordIdx, MAXTRIE, -1);
14     for (int i = 0; i < MAXTRIE; i++) fill_n(to[i], SIGMA, 0);

```

```

15  fill_n(sLink, MAXTRIE, 0);
16  fill_n(dLink, MAXTRIE, 0);
17  nnodes = 1;
18
19  for(int i = 0; i < npatterns; i++) {
20      int cur = 0;
21      for (char c : patterns[i]) {
22          int idx = c - 'a';
23          if(to[cur][idx] == 0) to[cur][idx] = nnodes++;
24          cur = to[cur][idx];
25      }
26      wordIdx[cur] = i;
27  }
28
29  queue<int> q;
30  q.push(0);
31  while(!q.empty()) {
32      int cur = q.front(); q.pop();
33      for(int c = 0; c < SIGMA; c++) {
34          if(to[cur][c]) {
35              int sl = sLink[to[cur][c]] = cur == 0 ? 0 : to[sLink[cur]][c];
36              // if all strings have equal length, remove this:
37              dLink[to[cur][c]] = wordIdx[sl] >= 0 ? sl : dLink[sl];
38              q.push(to[cur][c]);
39          } else to[cur][c] = to[sLink[cur]][c];
40      }
41  }
42
43  for (int cur = 0, i = 0, n = S.size(); i < n; i++) {
44      int idx = S[i] - 'a';
45      cur = to[cur][idx];
46      // we have a match! (if g[i][j] >= 0)
47      for (int hit = wordIdx[cur] >= 0 ? cur : dLink[cur]; hit; hit = dLink[hit]) {
48          cerr << "Match for " << patterns[wordIdx[hit]] << " at " << (i + 1 - patterns[
              wordIdx[hit]].size()) << endl;
49      }
50  }
51 }

```

5 Geometry

```

1  typedef double NUM; // either double or long long
2
3  struct pt {
4      NUM x, y;
5
6      pt() : x(0), y(0) {}
7      pt(NUM _x, NUM _y) : x(_x), y(_y) {}
8      pt(const pt &p) : x(p.x), y(p.y) {}
9
10     pt operator*(NUM scalar) const {
11         return pt(scalar * x, scalar * y); // scalar
12     }
13     NUM operator*(const pt &rhs) const {
14         return x * rhs.x + y * rhs.y; // dot product
15     }
16     NUM operator^(const pt &rhs) const {
17         return x * rhs.y - y * rhs.x; // cross product
18     }
19     pt operator+(const pt &rhs) const {
20         return pt(x + rhs.x, y + rhs.y); // addition
21     }
22     pt operator-(const pt &rhs) const {
23         return pt(x - rhs.x, y - rhs.y); // subtraction
24     }

```

```

25     bool operator==(const pt &rhs) const {
26         return x == rhs.x && y == rhs.y;
27     }
28     bool operator!=(const pt &rhs) const {
29         return x != rhs.x || y != rhs.y;
30     }
31 };
32
33 // distance SQUARED from pt a to pt b
34 NUM sqDist(const pt &a, const pt &b) {
35     return (a - b) * (a - b);
36 }
37
38 // distance SQUARED from pt a to line bc
39 double sqDistPointLine(pt a, pt b, pt c) {
40     a = a - b;
41     c = c - b;
42     return (a ^ c) * (a ^ c) / (double)(c * c);
43 }
44
45 // distance SQUARED from pt a to line segment c
46 double sqDistPointSegment(pt a, pt b, pt c) {
47     a = a - b;
48     c = c - b;
49     NUM dot = a * c, len = c * c;
50     if (dot <= 0) return a * a;
51     if (dot >= len) return (a - c) * (a - c);
52     return a * a - dot * dot / ((double) len);
53     // pt proj = c * dot / ((double) len);
54 }
55
56 bool between(NUM a, NUM b, NUM n) {
57     return min(a, b) <= n && n <= max(a, b);
58 }
59 bool collinear(pt a, pt b, pt c) {
60     return (a - b) ^ (a - c) == 0;
61 }
62
63 // point a on segment bc
64 bool pointOnSegment(pt a, pt b, pt c)
65 {
66     return collinear(a, b, c) &&
67         between(b.x, c.x, a.x) && between(b.y, c.y, a.y);
68 }
69
70 pt lineLineIntersection(pt a, pt b, pt c, pt d, bool &cross)
71 {
72     pt res = (c - d) * (a ^ b) - (a - b) * (c ^ d);
73     NUM det = (a.x - b.x) * (c.y - d.y) - (a.y - b.y) * (c.x - d.x);
74     cross = det != 0;
75     if (cross) res = res / det;
76     return res;
77 }
78
79 // Line segment a1 -- a2 intersects with b1 -- b2?
80 // returns 0: no, 1: yes at i1, 2: yes at i1 -- i2
81 int segmentsIntersect(pt a1, pt a2, pt b1, pt b2, pt &i1, pt &i2) {
82     if (((a2 - a1) ^ (b2 - b1)) < 0) swap(a1, a2);
83     // assert(a1 != a2 && b1 != b2);
84     pt q = a2 - a1, r = b2 - b1, s = b1 - a1;
85     NUM cross = q ^ r, c1 = s ^ r, c2 = s ^ q;
86     if (cross == 0) {
87         // line segments are parallel
88         if ((q ^ s) != 0) return 0; // no intersection
89         NUM v1 = s * q, v2 = (b2 - a1) * q, v3 = q * q;
90         if (v2 < v1) swap(v1, v2), swap(b1, b2);
91
92         if (v1 > v3 || v2 < 0) return 0; // intersection empty

```

```

93     i1 = v2 > v3 ? a2 : b2;
94     i2 = v1 < 0 ? a1 : b1;
95     return i1 == i2 ? 1 : 2; // one point or overlapping
96 } else { // cross > 0
97     i1 = pt(a1) + pt(q) * (1.0 * c1 / cross); // needs double
98     return 0 <= c1 && c1 <= cross && 0 <= c2 && c2 <= cross;
99     // intersection inside segments
100 }
101 }
102
103 // complete intersection check
104 int segmentsIntersect2(pt a1, pt a2, pt b1, pt b2, pt &i1, pt &i2) {
105     if (a1 == a2 && b1 == b2) {
106         i1 = a1;
107         return a1 == b1;
108     } else if (a1 == a2) {
109         i1 = a1;
110         return pointOnSegment(a1, b1, b2);
111     } else if (b1 == b2) {
112         i1 = b1;
113         return pointOnSegment(b1, a1, a2);
114     } else return segmentsIntersect(a1, a2, b1, b2, i1, i2);
115 }
116
117 // Returns TWICE the area of a polygon to keep it an integer
118 NUM polygonTwiceArea(const vector<pt> &polygon) {
119     NUM area = 0;
120     for (int i = 0, N = polygon.size(), j = N - 1; i < N; j = i++)
121         area += polygon[i] ^ polygon[j];
122     return abs(area);
123 }
124
125 // returns 0 outside, 1 inside, 2 on boundary
126 int pointInPolygon(pt p, const vector<pt> &polygon) {
127     // Check crossings with horizontal semi-line through p to +x
128     int crosscount = 0, N = polygon.size();
129     for (int i = 0, j = N - 1; i < N; j = i++) {
130         if (pointOnSegment(p, polygon[i], polygon[j])) return 2;
131
132         // check if it crosses the vertical y = p.y line
133         NUM l = (p.x - polygon[i].x) * (polygon[j].y - polygon[i].y);
134         NUM r = (p.y - polygon[i].y) * (polygon[j].x - polygon[i].x);
135         if (polygon[j].y > p.y) {
136             if (polygon[i].y <= p.y && l < r) crosscount++;
137         } else {
138             if (polygon[i].y <= p.y && l > r) crosscount++;
139         }
140     }
141     return crosscount & 1;
142 }
143
144 // Assumption: polygon has unique points
145 int pointInConvex(pt p, const vector<pt> &polygon) {
146     // the cross product should always have the same sign,
147     // when the point is inside the convex
148     int N = polygon.size(), sgn = 0;
149     bool onBoundary = false;
150     for (int i = 0, j = N - 1; i < N; j = i++) {
151         NUM cross = (polygon[j] - p) ^ (polygon[i] - p);
152         if (cross == 0) onBoundary = true;
153         else if (sgn == 0) sgn = sign(cross);
154         else if (sgn != sign(cross)) return 0;
155     }
156     return onBoundary ? 2 : 1;
157 }

```

5.1 Convex Hull $\mathcal{O}(n \log n)$

```

1 // output contains indices of the points on the hull
2 void convex_hull(const vector<pt> &pts, vector<int> &output) {
3     output.clear();
4     if (pts.size() < 3) {
5         if (pts.size() >= 1) output.push_back(0);
6         if (pts.size() >= 2) output.push_back(1);
7         return;
8     }
9
10    unsigned int bestIndex = 0;
11    NUM minX = pts[0].x, minY = pts[0].y;
12    for(unsigned int i = 1; i < pts.size(); ++i) {
13        if (pts[i].x < minX || (pts[i].x == minX && pts[i].y < minY)) {
14            bestIndex = i;
15            minX = pts[i].x;
16            minY = pts[i].y;
17        }
18    }
19    vector<int> ordered; //index into pts
20    for(unsigned int i = 0; i < pts.size(); ++i) {
21        if (i != bestIndex) ordered.push_back(i);
22    }
23
24    pt refr = pts[bestIndex];
25    sort(ordered.begin(), ordered.end(), [&pts,&refr] (int a, int b) -> bool {
26        NUM cross = (pts[a] - refr) ^ (pts[b] - refr);
27        return cross != 0 ? cross > 0 : sqDist(refr, pts[a]) < sqDist(refr, pts[b]);
28    });
29
30    output.push_back(bestIndex);
31    output.push_back(ordered[0]);
32    output.push_back(ordered[1]);
33    for(unsigned int i = 2; i < ordered.size(); ++i) {
34        //NOTE: > INCLUDES and >= EXCLUDES points on the hull-line
35        while (output.size() > 1 && ((pts[output[output.size() - 2]] - pts[output.back()]) ^ (
36            pts[ordered[i]] - pts[output.back()])) > 0) {
37            output.pop_back();
38        }
39        output.push_back(ordered[i]);
40    }
41    return;
42 }
```

6 Miscellaneous

6.1 Binary search $\mathcal{O}(\log(hi - lo))$

```

1 bool test(int n);
2
3 int search(int lo, int hi) {
4     // assert(test(lo) && !test(hi));
5     while (hi - lo > 1) {
6         int c = (lo + hi) / 2;
7         if (test(c)) lo = c;
8         else hi = c;
9     }
10    // assert(test(lo) && !test(hi));
11    return lo;
12 }
```

6.2 Fast Fourier Transform $\mathcal{O}(n \log n)$

Given two polynomials $A(x) = a_0 + a_1x + \dots + a_{n/2}x^{n/2}$ and $B(x) = b_0 + b_1x + \dots + b_{n/2}x^{n/2}$, FFT calculates all coefficients of $C(x) = A(x) \cdot B(x) = c_0 + c_1x + \dots + c_nx^n$.

```

1
2 typedef complex<double> cpx;
3 const int logmaxn = 20, maxn = 1 << logmaxn;
4
5 cpx a[maxn] = {}, b[maxn] = {}, c[maxn];
6
7 void fft(cpx *src, cpx *dest)
8 {
9     for (int i = 0, rep = 0; i < maxn; i++, rep = 0) {
10         for (int j = i, k = logmaxn; k-- >= 1; j >= 1) rep = (rep << 1) | (j & 1);
11         dest[rep] = src[i];
12     }
13     for (int s = 1, m = 1; m <= maxn; s++, m *= 2) {
14         cpx r = exp(cpx(0, 2.0 * pi / m));
15         for (int k = 0; k < maxn; k += m) {
16             cpx cr(1.0, 0.0);
17             for (int j = 0; j < m / 2; j++) {
18                 NUM t = cr * dest[k + j + m / 2];
19                 dest[k + j + m / 2] = dest[k + j] - t;
20                 dest[k + j] += t;
21                 cr *= r;
22             }
23         }
24     }
25 }
26
27 void multiply()
28 {
29     fft(a, c);
30     fft(b, a);
31     for (int i = 0; i < maxn; i++) b[i] = conj(a[i] * c[i]);
32     fft(b, c);
33     for (int i = 0; i < maxn; i++) c[i] = conj(c[i]) / (1.0 * maxn);
34 }

```

6.3 Minimum Assignment (Hungarian Algorithm) $\mathcal{O}(n^3)$

```

1 int a[MAXN + 1][MAXM + 1]; // matrix, 1-based
2
3 int minimum_assignment(int n, int m) { // n rows, m columns
4     vector<int> u(n + 1), v(m + 1), p(m + 1), way(m + 1);
5
6     for (int i = 1; i <= n; i++) {
7         p[0] = i;
8         int j0 = 0;
9         vector<int> minv(m + 1, INF);
10        vector<char> used(m + 1, false);
11        do {
12            used[j0] = true;
13            int i0 = p[j0], delta = INF, j1;
14            for (int j = 1; j <= m; j++)
15                if (!used[j]) {
16                    int cur = a[i0][j] - u[i0] - v[j];
17                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
18                    if (minv[j] < delta) delta = minv[j], j1 = j;
19                }
20            for (int j = 0; j <= m; j++) {
21                if (used[j]) u[p[j]] += delta, v[j] -= delta;
22                else minv[j] -= delta;
23            }
24            j0 = j1;

```

```
25         } while (p[j0] != 0);
26     do {
27         int j1 = way[j0];
28         p[j0] = p[j1];
29         j0 = j1;
30     } while (j0);
31 }
32
33 // column j is assigned to row p[j]
34 // for (int j = 1; j <= m; ++ j) ans[p[j]] = j;
35 return -v[0];
36 }
```
