TCR

git diff solution (Jens Heuseveldt, Ludo Pulles, Pim Spelier)

September 25, 2017

Contents

At the start of a contest, type this in a terminal:

```
1 printf "set nu sw=4 ts=4 noet ai hls shellcmdflag=-ic\nsyntax on\ncolor slate" > ~/.vimrc
2 echo "alias g11='g++ -Wall -Wshadow -DLOG -std=c++11'" >> ~/.bashrc
```

template.cpp

```cpp
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  // Order statistics tree (if supported by judge!):
5  #include <ext/pb_ds/assoc_container.hpp>
6  #include <ext/pb_ds/tree_policy.hpp>
7  using namespace __gnu_pbds;
8
9  template<class TK, class TM>
10 using order_tree = tree<TK, TM, less<TK>, rb_tree_tag, tree_order_statistics_node_update>;
11 // iterator find_by_order(int r) (zero based)
12 // int      order_of_key(TK v)
13 template<class TV> using order_set = order_tree<TV, null_type>;
14
15 #define x first
16 #define y second
17 #define pb push_back
18 #define mp make_pair
19 #define eb emplace_back
20 #define rep(i,a,b) for(auto i=(a);i!=(b); ++i)
21 #define all(v) (v).begin(), (v).end()
22
23 typedef long long ll;
24 typedef pair<int, int> pii;
25 typedef vector<int> vi;
26 typedef vector<vi> vvi;
27 template<class T> using min_queue = priority_queue<T, vector<T>, greater<T>>;
28
29 const int INF = 2147483647; // (1 << 30) - 1 + (1 << 30)
30 const ll LLINF = (1LL << 62) - 1 + (1LL << 62); // = 9.223.372.036.854.775.807
31 const double PI = acos(-1.0);
32
33 #ifdef LOG
34 #define DBG(x) cerr << __LINE__ << ": " << #x << " = " << (x) << endl
35 #else
36 #define DBG(x)
37 const bool LOG = false;
38 #endif
39
40 void Log() { if(LOG) cerr << "\n\n"; }
41 template<class T, class... S>
42 void Log(T t, S... s) { if(LOG) cerr << t << "\t", Log(s...); }
43
44 // lambda-expression: [] (args) -> retType { body }
45 int main() {
46     ios_base::sync_with_stdio(false); // fast IO
47     cin.tie(NULL); // fast IO
48     cerr << boolalpha; // print true/false
49     (cout << fixed).precision(10); // adjust precision
50
51     return 0;
52 }
```

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 **ALLE** opgaves **goed** lezen
- Test de kleine voorbeeldgevallen
- Houd na 2 uur een pauze en overleg waar iedereen mee bezig is
- Maak zelf wat test-cases
- Typ de dingen uit de TCR, die je zeker nodig hebt, alvast in
- 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

0.2   Wrong Answer

1. Print de oplossing om te debuggen! Kijk ook naar andere (mogelijk makkelijkere) problemen.
2. Bedenk zelf test-cases met **randgevallen**!
3. Controleer op **overflow** (gebruik **OVERAL** long long, long double).
   Kijk naar overflows in tussenantwoorden bij modulo.
4. Controleer de **precisie**.
5. Controleer op **typo's**.
6. Loop de voorbeeldinput accuraat langs.
7. Controller op off-by-one-errors (in indices of lus-grenzen)?

0.3   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:

```
bool isOverflown = __builtin_[add|mul|sub]_overflow(a, b, \&res);
```

0.4   Covering problems

$$\text{Minimum edge cover} \iff \text{Maximum independent set}$$

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  int abs(int x) { return x > 0 ? x : -x; }
2  int sign(int x) { return (x > 0) - (x < 0); }
3
4  // greatest common divisor
5  ll gcd(ll a, ll b) { while (b) a %= b, swap(a, b); return a; };
6  // least common multiple
7  ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }
8  ll mod(ll a, ll b) { return (a %= b) < 0 ? a + b : a; }
9
10 // safe multiplication (ab % m) for m <= 4e18 in O(log b)
11 ll mulmod(ll a, ll b, ll m) {
12     ll r = 0;
13     while (b) {
14         if (b & 1) r = (r + a) % m;
15         a = (a + a) % m;
16         b >>= 1;
17     }
18     return r;
19 }
20
21 // safe exponentation (a^b % m) for m <= 2e9 in O(log b)
22 ll powmod(ll a, ll b, ll m) {
23     ll r = 1;
24     while (b) {
25         if (b & 1) r = (r * a) % m; // r = mulmod(r, a, m);
26         a = (a * a) % m; // a = mulmod(a, a, m);
27         b >>= 1;
28     }
29     return r;
30 }
31
32 // returns x, y such that ax + by = gcd(a, b)
33 ll egcd(ll a, ll b, ll &x, ll &y) {
34     ll xx = y = 0, yy = x = 1;
35     while (b) {
36         x -= a / b * xx; swap(x, xx);
37         y -= a / b * yy; swap(y, yy);
38         a %= b; swap(a, b);
39     }
40     return a;
41 }
42
43 // Chinese remainder theorem
44 const pll NO_SOLUTION(0, -1);
45 // Returns (u, v) such that x = u % v <=> x = a % n and x = b % m
46 pll crt(ll a, ll n, ll b, ll m) {
47     ll s, t, d = egcd(n, m, s, t), nm = n * m;
48     if (mod(a - b, d)) return NO_SOLUTION;
49     return pll(mod(s * b * n + t * a * m, nm) / d, nm / d);
50     /* when n, m > 10^6, avoid overflow:
51     return pll(mod(mulmod(mulmod(s, b, nm), n, nm)
52                 + mulmod(mulmod(t, a, nm), m, nm), nm) / d, nm / d); */
53 }
54
55 // phi[i] = #{ 0 < j <= i | gcd(i, j) = 1 }
56 vi totient(int N) {
57     vi phi(N);
58     for (int i = 0; i < N; i++) phi[i] = i;
59     for (int i = 2; i < N; i++)
60         if (phi[i] == i)
61             for (int j = i; j < N; j += i) phi[j] -= phi[j] / i;
62     return phi;
63 }
64
65 // calculate nCk % p (p prime!)
```

```
66 ll lucas(ll n, ll k, ll p) {
67     ll ans = 1;
68     while (n) {
69         ll np = n % p, kp = k % p;
70         if (np < kp) return 0;
71         ans = mod(ans * binom(np, kp), p); // (np C kp)
72         n /= p; k /= p;
73     }
74     return ans;
75 }
76
77 // returns if n is prime for n < 3e24 ( > 2^64)
78 bool millerRabin(ll n)
79 {
80     if (n < 2 || n % 2 == 0) return n == 2;
81     ll d = n - 1, ad, s = 0, r;
82     for (; d % 2 == 0; d /= 2) s++;
83     for (int a : { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41 }) {
84         if (n == a) return true;
85         if ((ad = powmod(a, d, n)) == 1) continue;
86         for (r = 0; r < s && ad + 1 != n; r++)
87             ad = mulmod(ad, ad, n);
88         if (r == s) return false;
89     }
90     return true;
91 }
```

2 Datastructures

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

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

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

Use one-based indices $(i > 0)$!

```
1  int bit[MAXN + 1];
```

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

2.3   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 sizeL = 1e4, sizeR = 1e4;
2  bool vis[sizeR];
3  int par[sizeR]; // par : R -> L
4  vi adj[sizeL]; // adj : L -> (N -> R)
5
6  bool match(int u) {
7      for (int v : adj[u]) {
8          if (vis[v]) continue;
9          vis[v] = true;
10         if (par[v] == -1 || match(par[v])) {
11             par[v] = u;
12             return true;
13         }
14     }
15     return false;
16 }
17
18 // perfect matching iff ret == sizeL == sizeR
19 int maxmatch() {
20     fill_n(par, sizeR, -1);
21     int ret = 0;
22     for (int i = 0; i < sizeL; i++) {
23         fill_n(vis, sizeR, false);
24         ret += match(i);
25     }
26     return ret;
```

```
27  }
```

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

```
1  vvi adj, comps;
2  vi tidx, lnk, cnr, st;
3  vector<bool> vis;
4  int age, ncomps;
5
6  void tarjan(int v) {
7      tidx[v] = lnk[v] = ++age;
8      vis[v] = true;
9      st.pb(v);
10
11     for (int w : adj[v]) {
12         if (!tidx[w]) tarjan(w), lnk[v] = min(lnk[v], lnk[w]);
13         else if (vis[w]) lnk[v] = min(lnk[v], tidx[w]);
14     }
15
16     if (lnk[v] != tidx[v]) return;
17
18     comps.pb(vi());
19     int w;
20     do {
21         vis[w = st.back()] = false;
22         cnr[w] = ncomps;
23         comps.back().pb(w);
24         st.pop_back();
25     } while (w != v);
26     ncomps++;
27  }
28
29  void findSCC(int n) {
30      age = ncomps = 0;
31      vis.assign(n, false);
32      tidx.assign(n, 0);
33      lnk.resize(n);
34      cnr.resize(n);
35      comps.clear();
36
37      for (int i = 0; i < n; i++)
38          if (tidx[i] == 0) tarjan(i);
39  }
```

3.2.1   2-SAT $\mathcal{O}(V + E)$

Include `findSCC`.

```
1  void init2sat(int n) { adj.assign(2 * n, vi()); }
2
3  // vl, vr = true -> variable l, variable r should be negated.
4  void imply(int xl, bool vl, int xr, bool vr) {
5      adj[2 * xl + vl].pb(2 * xr + vr);
6      adj[2 * xr +!vr].pb(2 * xl +!vl);
7  }
8
9  void satOr(int xl, bool vl, int xr, bool vr) { imply(xl, !vl, xr, vr); }
10 void satConst(int x, bool v) { imply(x, !v, x, v); }
11 void satIff(int xl, bool vl, int xr, bool vr) {
12     imply(xl, vl, xr, vr);
13     imply(xr, vr, xl, vl);
14 }
15
16 bool solve2sat(int n, vector<bool> &sol) {
```

```
17      findSCC(2 * n);
18      for (int i = 0; i < n; i++)
19          if (cnr[2 * i] == cnr[2 * i + 1]) return false;
20      vector<bool> seen(n, false);
21      sol.assign(n, false);
22      for (vi &comp : comps) {
23          for (int v : comp) {
24              if (seen[v / 2]) continue;
25              seen[v / 2] = true;
26              sol[v / 2] = v & 1;
27          }
28      }
29      return true;
30  }
```

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 $w_{ij} < 0$ in the graph.

```
1  vector< pair<pii,int> > edges; // ((from, to), weight)
2  vi dist;
3
4  // when undirected, add back edges
5  bool bellman_ford(int V, int source) {
6      dist.assign(V, INF / 3);
7      dist[source] = 0;
8
9      bool updated = true;
10     int loops = 0;
11     while (updated && loops < n) {
12         updated = false;
13         for (auto e : edges) {
14             int alt = dist[e.x.x] + e.y;
15             if (alt < dist[e.x.y]) {
16                 dist[e.x.y] = alt;
17                 updated = true;
18             }
19         }
20     }
21     return loops < n; // loops >= n: negative cycles
22 }
```

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/KuoE0/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 void add_edge(int fr, int to, ll cap) {
12     g[fr].pb(edge(to, g[to].size(), cap));
13     g[to].pb(edge(fr, g[fr].size() - 1, 0));
14 }
15
16 bool dinic_bfs() {
17     fill_n(level, MAXN, 0);
18     level[s] = 1;
19
20     queue<int> q;
21     q.push(s);
22     while (!q.empty()) {
23         int cur = q.front();
24         q.pop();
25         for (edge e : g[cur]) {
26             if (level[e.to] == 0 && e.flow < e.cap) {
27                 level[e.to] = level[cur] + 1;
28                 q.push(e.to);
29             }
30         }
31     }
32     return level[t] != 0;
33 }
34
35 ll dinic_dfs(int cur, ll maxf) {
36     if (cur == t) return maxf;
37
38     ll f = 0;
39     bool isSat = true;
40     for (edge &e : g[cur]) {
41         if (level[e.to] != level[cur] + 1 || e.flow >= e.cap)
42             continue;
43         ll df = dinic_dfs(e.to, min(maxf - f, e.cap - e.flow));
44         f += df;
45         e.flow += df;
46         g[e.to][e.rev].flow -= df;
47         isSat &= e.flow == e.cap;
48         if (maxf == f) break;
49     }
50     if (isSat) level[cur] = 0;
51     return f;
52 }
53
54 ll dinic_maxflow() {
55     ll f = 0;
56     while (dinic_bfs()) f += dinic_dfs(s, LLINF);
57     return f;
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      fill_n(dist, n, LLINF);
14      fill_n(par, n, -1);
15
16      priority_queue< pii, vector<pii>, greater<pii> > q;
17      q.push(pii(dist[s] = 0, s));
18
19      while (!q.empty()) {
20          int d = q.top().x, v = q.top().y;
21          q.pop();
22          if (d > dist[v]) continue;
23
24          for (edge e : adj[v]) {
25              if (e.f < e.c && d + e.w < dist[e.t]) {
26                  q.push(pii(dist[e.t] = d + e.w, e.t));
27                  par[e.t] = e.r;
28              }
29          }
30      }
31      return dist[t] < INF;
32  }
33
34  pair<ll, ll> minCostMaxFlow(int s, int t) {
35      ll cost = 0, flow = 0;
36      while (findPath(s, t)) {
37          ll f = INF, c = 0;
38          int cur = t;
39          while (cur != s) {
40              const edge &rev = adj[cur][par[cur]], &e = adj[rev.t][rev.r];
41              f = min(f, e.c - e.f);
42              cur = rev.t;
43          }
44          cur = t;
45          while (cur != s) {
46              edge &rev = adj[cur][par[cur]], &e = adj[rev.t][rev.r];
47              c += e.w;
48              e.f += f;
49              rev.f -= f;
50              cur = rev.t;
51          }
52          cost += f * c;
53          flow += f;
54      }
55      return pair<ll, ll>(cost, flow);
56  }
57
58  inline void addEdge(int from, int to, ll cap, ll weight) {
59      adj[from].pb(edge(to, adj[to].size(), cap, weight));
60      adj[to].pb(edge(from, adj[from].size() - 1, 0, -weight));
61  }
```

3.6   Minimal Spanning Tree

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

```
1  struct edge { int x, y, w; };
2  edge edges[MAXM];
3
4  ll kruskal(int n, int m) { // n: #vertices, m: #edges
5      uf_init(n);
6      sort(edges, edges + m, [] (edge a, edge b) -> bool { return a.w < b.w; });
7      ll ret = 0;
```

```
 8      while (m--) {
 9          if (uf_find(edges[m].x) == uf_find(edges[m].y)) continue;
10          ret += edges[m].w;
11          uf_union(edges[m].x, edges[m].y);
12      }
13      return ret;
14 }
```

4 String algorithms

4.1 Trie

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

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

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

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

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

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

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

SUBSTRING: *consecutive characters* !!!

```
1  int dp[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++) {
6          for (int j = 0; j < n2; j++) {
7              if (i == 0 || j == 0) dp[i][j] = 0;
8              else if (w1[i - 1] == w2[j - 1]) dp[i][j] = dp[i - 1][j - 1] + 1;
9              else dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
10         }
11     }
12     return dp[n1][n2];
13 }
14
15 // backtrace
16 string getLCS(const string &w1, const string &w2) {
17     int i = w1.size(), j = w2.size();
18     string ret = "";
19     while (i > 0 && j > 0) {
20         if (w1[i - 1] == w2[j - 1]) ret += w1[--i], j--;
21         else if (dp[i][j - 1] > dp[i - 1][j]) j--;
22         else i--;
23     }
24     reverse(ret.begin(), ret.end());
25     return ret;
26 }
```

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

Also known as the 'Edit distance'.

```
1  int dp[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++) dp[i][0] = i; // removal
```

```
6        for (int j = 0; j <= n2; j++) dp[0][j] = j; // insertion
7        for (int i = 1; i <= n1; i++)
8            for (int j = 1; j <= n2; j++)
9                dp[i][j] = min(
10                    1 + min(dp[i - 1][j], dp[i][j - 1]),
11                    dp[i - 1][j - 1] + (w1[i - 1] != w2[j - 1])
12                );
13        return dp[n1][n2];
14   }
```

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

```
1  int kmp_search(const string &word, const string &text) {
2      int n = word.size();
3      vi T(n + 1, 0);
4      for (int i = 1, j = 0; i < n; ) {
5          if (word[i] == word[j]) T[++i] = ++j; // match
6          else if (j > 0) j = T[j]; // fallback
7          else i++; // no match, keep zero
8      }
9      int matches = 0;
10     for (int i = 0, j = 0; i < text.size(); ) {
11         if (text[i] == word[j]) {
12             i++;
13             if (++j == n) { // match at interval [i - n, i)
14                 matches++;
15                 j = T[j];
16             }
17         } else if (j > 0) j = T[j];
18         else i++;
19     }
20     return matches;
21 }
```

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

All given P must be unique!

```
1  const int MAXP = 100, MAXLEN = 200, SIGMA = 26, MAXTRIE = MAXP * MAXLEN;
2
3  int nP;
4  string P[MAXP], S;
5
6  int pnr[MAXTRIE], to[MAXTRIE][SIGMA], sLink[MAXTRIE], dLink[MAXTRIE], nnodes;
7
8  void ahoCorasick() {
9      fill_n(pnr, MAXTRIE, -1);
10     for (int i = 0; i < MAXTRIE; i++) fill_n(to[i], SIGMA, 0);
11     fill_n(sLink, MAXTRIE, 0);
12     fill_n(dLink, MAXTRIE, 0);
13     nnodes = 1;
14     // STEP 1: MAKE A TREE
15     for (int i = 0; i < nP; i++) {
16         int cur = 0;
17         for (char c : P[i]) {
18             int i = c - 'a';
19             if (to[cur][i] == 0) to[cur][i] = nnodes++;
20             cur = to[cur][i];
21         }
22         pnr[cur] = i;
23     }
24     // STEP 2: CREATE SUFFIX_LINKS AND DICT_LINKS
25     queue<int> q;
26     q.push(0);
```

```
27      while (!q.empty()) {
28          int cur = q.front();
29          q.pop();
30          for (int c = 0; c < SIGMA; c++) {
31              if (to[cur][c]) {
32                  int sl = sLink[to[cur][c]] = cur == 0 ? 0 : to[sLink[cur]][c];
33                  // if all strings have equal length, remove this
34                  dLink[to[cur][c]] = pnr[sl] >= 0 ? sl : dLink[sl];
35                  q.push(to[cur][c]);
36              } else to[cur][c] = to[sLink[cur]][c];
37          }
38      }
39      // STEP 3: TRAVERSE S
40      for (int cur = 0, i = 0, n = S.size(); i < n; i++) {
41          cur = to[cur][S[i] - 'a'];
42          for (int hit = pnr[cur] >= 0 ? cur : dLink[cur]; hit; hit = dLink[hit]) {
43              cerr << P[pnr[hit]] << " found at [" << (i + 1 - P[pnr[hit]].size()) << ", " << i
                      << "]" << endl;
44          }
45      }
46 }
```

5 Geometry

```
1 const double EPS = 1e-7, PI = acos(-1.0);
2
3 typedef long long NUM; // EITHER double OR long long
4 typedef pair<NUM, NUM> pt;
5 #define x first
6 #define y second
7
8 pt operator+(pt p, pt q) { return pt(p.x + q.x, p.y + q.y); }
9 pt operator-(pt p, pt q) { return pt(p.x - q.x, p.y - q.y); }
10
11 pt& operator+=(pt &p, pt q) { return p = p + q; }
12 pt& operator-=(pt &p, pt q) { return p = p - q; }
13
14 pt operator*(pt p, NUM l) { return pt(p.x * l, p.y * l); }
15 pt operator/(pt p, NUM l) { return pt(p.x / l, p.y / l); }
16
17 NUM operator*(pt p, pt q) { return p.x * q.x + p.y * q.y; }
18 NUM operator^(pt p, pt q) { return p.x * q.y - p.y * q.x; }
19
20 istream& operator>>(istream &in, pt &p) { return in >> p.x >> p.y; }
21 ostream& operator<<(ostream &out, pt p) { return out << '(' << p.x << ", " << p.y << ')'; }
22
23 NUM lenSq(pt p) { return p * p; }
24 NUM lenSq(pt p, pt q) { return lenSq(p - q); }
25 double len(pt p) { return hypot(p.x, p.y); } // more overflow safe
26 double len(pt p, pt q) { return len(p - q); }
27
28 typedef pt frac;
29 typedef pair<double, double> vec;
30 vec getvec(pt p, pt dp, frac t) { return vec(p.x + 1. * dp.x * t.x / t.y, p.y + 1. * dp.y * t.
       x / t.y); }
31
32 // square distance from pt a to line bc
33 frac distPtLineSq(pt a, pt b, pt c) {
34      a -= b, c -= b;
35      return frac((a ^ c) * (a ^ c), c * c);
36 }
37
38 // square distance from pt a to linesegment bc
39 frac distPtSegmentSq(pt a, pt b, pt c) {
40      a -= b; c -= b;
```

```
41      NUM dot = a * c, len = c * c;
42      if (dot <= 0) return frac(a * a, 1);
43      if (dot >= len) return frac((a - c) * (a - c), 1);
44      return frac(a * a * len - dot * dot, len);
45  }
46
47  // projects pt a onto linesegment bc
48  frac proj(pt a, pt b, pt c) { return frac((a - b) * (c - b), (c - b) * (c - b)); }
49  vec projv(pt a, pt b, pt c) { return getvec(b, c - b, proj(a, b, c)); }
50
51  bool collinear(pt a, pt b, pt c) { return ((a - b) ^ (a - c)) == 0; }
52
53  bool pointOnSegment(pt a, pt b, pt c) {
54      NUM dot = (a - b) * (c - b), len = (c - b) * (c - b);
55      return collinear(a, b, c) && 0 <= dot && dot <= len;
56  }
57
58  // true => 1 intersection, false => parallel, so 0 or \infty solutions
59  bool linesIntersect(pt a, pt b, pt c, pt d) { return ((a - b) ^ (c - d)) != 0; }
60  vec lineLineIntersection(pt a, pt b, pt c, pt d) {
61      double det = (a - b) ^ (c - d);
62      pt ret = (c - d) * (a ^ b) - (a - b) * (c ^ d);
63      return vec(ret.x / det, ret.y / det);
64  }
65
66  // dp, dq are directions from p, q
67  // intersection at p + t_i dp, for 0 <= i < return value
68  int segmentIntersection(pt p, pt dp, pt q, pt dq, frac &t0, frac &t1)
69  {
70      if (dp * dp == 0) swap(p, q), swap(dp, dq); // dq = 0
71      if (dp * dp == 0) { t0 = t1 = frac(0, 1); return p == q; } // dp = dq = 0
72
73      pt dpq = (q - p);
74      NUM c = dp ^ dq, c0 = dpq ^ dp, c1 = dpq ^ dq;
75      if (c == 0) { // parallel, dp > 0, dq >= 0
76          if (c0 != 0) return 0; // not collinear
77          NUM v0 = dpq * dp, v1 = v0 + dq * dp, dp2 = dp * dp;
78          if (v1 < v0) swap(v0, v1);
79          t0 = frac(v0 = max(v0, (NUM) 0), dp2);
80          t1 = frac(v1 = min(v1, dp2), dp2);
81          return (v0 <= v1) + (v0 < v1);
82      } else if (c < 0) c = -c, c0 = -c0, c1 = -c1;
83      t0 = t1 = frac(c1, c);
84      return 0 <= min(c0, c1) && max(c0, c1) <= c;
85  }
86
87  // Returns TWICE the area of a polygon to keep it an integer
88  NUM polygonTwiceArea(const vector<pt> &pts) {
89      NUM area = 0;
90      for (int N = pts.size(), i = 0, j = N - 1; i < N; j = i++)
91          area += pts[i] ^ pts[j];
92      return abs(area); // area < 0 <=> pts ccw
93  }
94
95  bool pointInPolygon(pt p, const vector<pt> &pts) {
96      double sum = 0;
97      for (int N = pts.size(), i = 0, j = N - 1; i < N; j = i++) {
98          if (pointOnSegment(p, pts[i], pts[j])) return true; // boundary
99          double angle = acos((pts[i] - p) * (pts[j] - p) / len(pts[i], p) / len(pts[j], p));
100         sum += ((pts[i] - p) ^ (pts[j] - p)) < 0 ? angle : -angle;
101     }
102     return abs(abs(sum) - 2 * PI) < EPS;
103 }
```

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

```cpp
1  // points are given by: pts[ret[0]], pts[ret[1]], ... pts[ret[ret.size()-1]]
2  vi convexHull(const vector<pt> &pts) {
3      if (pts.empty()) return vi();
4      vi ret;
5      // find one outer point:
6      int fsti = 0, n = pts.size();
7      pt fstpt = pts[0];
8      for(int i = n; i--; ) {
9          if (pts[i] < fstpt) fstpt = pts[fsti = i];
10     }
11     ret.pb(fsti);
12     pt refr = pts[fsti];
13
14     vi ord; // index into pts
15     for (int i = n; i--; ) {
16         if (pts[i] != refr) ord.pb(i);
17     }
18     sort(ord.begin(), ord.end(), [&pts, &refr] (int a, int b) -> bool {
19         NUM cross = (pts[a] - refr) ^ (pts[b] - refr);
20         return cross != 0 ? cross > 0 : lenSq(refr, pts[a]) < lenSq(refr, pts[b]);
21     });
22     for (int i : ord) {
23         // NOTE: > INCLUDES points on the hull-line, >= EXCLUDES
24         while (ret.size() > 1 &&
25                 ((pts[ret[ret.size()-2]]-pts[ret.back()]) ^ (pts[i]-pts[ret.back()])) >= 0)
26             ret.pop_back();
27         ret.pb(i);
28     }
29     return ret;
30 }
```

5.2 Rotating Calipers $\mathcal{O}(n)$

Finds the longest distance between two points in a convex hull.

```cpp
1  NUM rotatingCalipers(vector<pt> &hull) {
2      int n = hull.size(), a = 0, b = 1;
3      if (n <= 1) return 0.0;
4      while (((hull[1] - hull[0]) ^ (hull[(b + 1) % n] - hull[b])) > 0) b++;
5      NUM ret = 0.0;
6      while (a < n) {
7          ret = max(ret, lenSq(hull[a], hull[b]));
8          if (((hull[(a + 1) % n] - hull[a % n]) ^ (hull[(b + 1) % n] - hull[b])) <= 0) a++;
9          else if (++b == n) b = 0;
10     }
11     return ret;
12 }
```

5.3 Closest points $\mathcal{O}(n \log n)$

```cpp
1  int n;
2  pt pts[maxn];
3
4  struct byY {
5      bool operator()(int a, int b) const { return pts[a].y < pts[b].y; }
6  };
7
8  inline NUM dist(pii p) {
9      return hypot(pts[p.x].x - pts[p.y].x, pts[p.x].y - pts[p.y].y);
10 }
11
12 pii minpt(pii p1, pii p2) {
13     return (dist(p1) < dist(p2)) ? p1 : p2;
14 }
```

```
15
16  // closest pts (by index) inside pts[l ... r], with sorted y values in ys
17  pii closest(int l, int r, vi &ys) {
18      if (r - l == 2) { // don't assume 1 here.
19          ys = { l, l + 1 };
20          return pii(l, l + 1);
21      } else if (r - l == 3) { // brute-force
22          ys = { l, l + 1, l + 2 };
23          sort(ys.begin(), ys.end(), byY());
24          return minpt(pii(l, l + 1), minpt(pii(l, l + 2), pii(l + 1, l + 2)));
25      }
26      int m = (l + r) / 2;
27      vi yl, yr;
28      pii delta = minpt(closest(l, m, yl), closest(m, r, yr));
29      NUM ddelta = dist(delta), xm = .5 * (pts[m-1].x + pts[m].x);
30      merge(yl.begin(), yl.end(), yr.begin(), yr.end(), back_inserter(ys), byY());
31      deque<int> q;
32      for (int i : ys) {
33          if (abs(pts[i].x - xm) <= ddelta) {
34              for (int j : q) delta = minpt(delta, pii(i, j));
35              q.pb(i);
36              if (q.size() > 8) q.pop_front(); // magic from Introduction to Algorithms.
37          }
38      }
39      return delta;
40  }
```

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 m = (lo + hi) / 2;
7          (test(m) ? lo : hi) = m;
8      }
9      // assert(test(lo) && !test(hi));
10     return lo;
11 }
```

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

Given two polynomials $A(x) = a_0 + a_1 x + \ldots + a_{n/2} x^{n/2}$ and $B(x) = b_0 + b_1 x + \ldots + b_{n/2} x^{n/2}$, FFT calculates all coefficients of $C(x) = A(x) \cdot B(x) = c_0 + c_1 x + \ldots c_n x^n$, with $c_i = \sum_{j=0}^{i} a_j b_{i-j}$.

```
1  typedef complex<double> cpx;
2  const int logmaxn = 20, maxn = 1 << logmaxn;
3
4  cpx a[maxn] = {}, b[maxn] = {}, c[maxn];
5
6  void fft(cpx *src, cpx *dest) {
7      for (int i = 0, rep = 0; i < maxn; i++, rep = 0) {
8          for (int j = i, k = logmaxn; k--; j >>= 1) rep = (rep << 1) | (j & 1);
9          dest[rep] = src[i];
10     }
11     for (int s = 1, m = 1; m <= maxn; s++, m *= 2) {
12         cpx r = exp(cpx(0, 2.0 * PI / m));
13         for (int k = 0; k < maxn; k += m) {
14             cpx cr(1.0, 0.0);
15             for (int j = 0; j < m / 2; j++) {
```

```
16                  cpx t = cr * dest[k + j + m / 2];
17                  dest[k + j + m / 2] = dest[k + j] - t;
18                  dest[k + j] += t;
19                  cr *= r;
20              }
21          }
22      }
23  }
24
25  void multiply() {
26      fft(a, c);
27      fft(b, a);
28      for (int i = 0; i < maxn; i++) b[i] = conj(a[i] * c[i]);
29      fft(b, c);
30      for (int i = 0; i < maxn; i++) c[i] = conj(c[i]) / (1.0 * maxn);
31  }
```

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      vi 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          vi 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  }
```

6.4   Partial linear equation solver $\mathcal{O}(N^3)$

```
1  typedef double NUM;
2
3  #define MAXN 110
4  #define EPS 1e-5
5
6  NUM mat[MAXN][MAXN + 1], vals[MAXN];
```

```
 7  bool hasval[MAXN];
 8
 9  bool is_zero(NUM a) { return -EPS < a && a < EPS; }
10  bool eq(NUM a, NUM b) { return is_zero(a - b); }
11
12  int solvemat(int n)
13  {
14      for(int i = 0; i < n; i++)
15          for (int j = 0; j < n; j++) cin >> mat[i][j];
16      for (int i = 0; i < n; i++) cin >> mat[i][n];
17
18      int pivrow = 0, pivcol = 0;
19      while (pivcol < n) {
20          int r = pivrow, c;
21          while (r < n && is_zero(mat[r][pivcol])) r++;
22          if (r == n) { pivcol++; continue; }
23
24          for (c = 0; c <= n; c++) swap(mat[pivrow][c], mat[r][c]);
25
26          r = pivrow++; c = pivcol++;
27          NUM div = mat[r][c];
28          for (int col = c; col <= n; col++) mat[r][col] /= div;
29          for (int row = 0; row < n; row++) {
30              if (row == r) continue;
31              NUM times = -mat[row][c];
32              for (int col = c; col <= n; col++) mat[row][col] += times * mat[r][col];
33          }
34      }
35      // now mat is in RREF
36      for (int r = pivrow; r < n; r++)
37          if (!is_zero(mat[r][n])) return 0;
38
39      fill_n(hasval, n, false);
40      for (int col = 0, row; col < n; col++) {
41          hasval[col] = !is_zero(mat[row][col]);
42          if (!hasval[col]) continue;
43          for (int c = col + 1; c < n; c++) {
44              if (!is_zero(mat[row][c])) hasval[col] = false;
45          }
46          if (hasval[col]) vals[col] = mat[row][n];
47          row++;
48      }
49
50      for (int i = 0; i < n; i++)
51          if (!hasval[i]) return 2;
52      return 1;
53  }
```