CP Algorithms Handbook

Stanisław Fiedler

February 6, 2025

Contents

1 Sorting and Searching

1.1 Quick Sort

```c
void qwik_sort(int p, int q, int tab[]){
    int a = p, b = q;
    int m = tab[(p+q)/2];

    while(a <= b){
        while(m > tab[a])
            a++;
        while(m < tab[b])
            b--;
        if(a <= b){
            swap(tab[a],tab[b]);
                a++;
                b--;
        }
    }
    if(b > p)
        qwik_sort(p, b, tab);
    if(a < q)
        qwik_sort(a, q, tab);
}
```

1.2 Merge Sort

```c
void merge(int left, int mid, int right, int t[]){
    int a = left, b = mid;
    int i = left;
    while(a < mid && b < right)
        tab[i++] = (t[a] < t[b] ? t[a++] : t[b++]);
    while(a < mid)
        tab[i++] = t[a++];
    while(b < right)
        tab[i++] = t[b++];
    for(int k = left; k < right; k++)
        t[k] = tab[k];
}

void mergesort(int left, int right, int t[]){
    if(left+1 != right){
        int m = (left+right)/2;
        mergesort(left, m, t);
        mergesort(m, right, t);
        merge(left, m, right, t);
    }
}
```

2 Dynamic Programming

2.1 State represenation

2.1.1 All subsets

This way you can generate all subsets in order ready for DP.

```c
    int sub = whole_set;
    while(sub > 0){

        sub = (sub-1)&whole_set;
    }
```

3 Graph Algorithms

3.1 Shortest Path

3.1.1 Djikstra

```
priority_queue<pair<long long int ,int>> q;
dist[1] = 0;
q.push({0, 1});
while(!q.empty()){
    int a = q.top().second;
    q.pop();
    if(vis[a]) continue;
    vis[a] = true;
    for(auto v : adj[a]){
        int b = v.first;
        int w = v.second;
        if(dist[b] > dist[a] + w){
            dist[b] = dist[a] + w;
            q.push({-dist[b], b});
        }
    }
}
```

3.1.2 Belman-Ford

```
dist[1] = 0;
for(int i = 1; i <= n-1; i++){
    for(auto e : edges){
        int a, b, w;
        tie(a, b ,w) = e;
        if(dist[b] > dist[a]+w){
            par[b] = a;
            dist[b] = dist[a]+w;
        }
    }
}
for(auto e : edges){
    int a, b, w;
    tie(a, b ,w) = e;
    if(dist[b] > dist[a]+w){
        found = true;
        node = b;
        par[b] = a;
        dist[b] = dist[a]+w;
    }
}
```

3.1.3 Floyd–Warshal

```
long long int dist[SIZE][SIZE] {};
for(int i = 0; i <= n; i++){
    for(int k = 0; k <= n; k++){
        dist[i][k] = __INT64_MAX__/2;
    }
}
for(int i = 1; i <= n; i++){
    dist[i][i] = 0;
    for(auto v : adj[i]){
        int b = v.first;
        long long int w = v.second;
        dist[i][b] = min(w, dist[i][b]);
        dist[b][i] = min(w, dist[i][b]);
    }
}

for(int k = 1; k <= n; k++){
    for(int i = 1; i <= n; i++){
        for(int j = 1; j <= n; j++){
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

3.2   Union Find

```cpp
int s_size[MX];
int link[MX];

int max_size = 0;
int num;

int find(int x){
    while(x != link[x])
        x = link[x];
    return x;
}

bool same(int a, int b){
    return find(a) == find(b);
}

void unite(int a, int b){
    a = find(a);
    b = find(b);
    if(s_size[a] < s_size[b])
        swap(a, b);
    s_size[a] += s_size[b];
    max_size = max(max_size, s_size[a]);
    link[b] = a;
    num--;
}

int main(){
    for(int i = 0; i < MX; i++){
        s_size[i] = 1;
        link[i] = i;
    }

    int n, m;
    cin >> n >> m;
    num = n;
    for(int i = 0; i < m; i++){
        int a, b;
        cin >> a >> b;
        if(!same(a, b))
            unite(a, b);
        cout << num << " " << max_size << endl;
    }
    return 0;
}
```

3.3   Path through all edges

```cpp
set <int> adj[MX];

bool vis[MX];

void zero(){
    for(int i = 0; i < MX; i++)
        vis[i] = false;
}

int dfs1(int a){
    int ret = 0;
    vis[a] = true;
    for(int b : adj[a]){
        ret += 1;
        if(!vis[b])
            ret += dfs1(b);
    }
    return ret;
}

int main(){
    int n, m;
    cin >> n >> m;
```

```cpp
    for(int i = 0; i < m; i++){
        int a, b;
        cin >> a >> b;
        adj[a].insert(b);
        adj[b].insert(a);
    }

    zero();
    if(dfs1(1)/2 != m){
        cout << "IMPOSSIBLE";
        return 0;
    }

    for(int i = 1; i <= n; i++){
        if(adj[i].size() & 1){
            cout << "IMPOSSIBLE";
            return 0;
        }
    }

    vector<int> ans;

    stack<int> s;
    s.push(1);
    while(!s.empty()){
        int a = s.top();
        if(adj[a].size() == 0){
            ans.push_back(a);
            s.pop();
        }
        else{
            int b = *adj[a].begin();
            adj[a].erase(adj[a].begin());
            adj[b].erase(a);
            s.push(b);
        }
    }

    for(int a : ans)
        cout << a << " ";


    return 0;
}
```

3.4 Topo Sort

```cpp
int n, m;
stack<int> res;
vector<int> g[100007];
int vis[100007];
bool imp = false;

void dfs(int n){
    vis[n] = 1;
    for(int v : g[n]){
        if(vis[v] == 0)
            dfs(v);
        else if(vis[v] == 1)
            imp = true;
    }
    vis[n] = 2;
    res.push(n);
}

int main(){
    cin >> n >> m;
    for(int i = 0 ; i < m; i++){
        int a, b;
        cin >> a >> b;
        g[a].push_back(b);
    }

```

```
27      for(int i = 1; i <= n; i++){
28          if(vis[i] == 0){
29              dfs(i);
30          }
31      }
32
33      if(!imp){
34          while(!res.empty()){
35              cout << res.top() << " ";
36              res.pop();
37          }
38      }
39      else{
40          cout << "IMPOSSIBLE";
41      }
42
43      return 0;
44 }
```

3.5 Flows and Cuts

All problems in this section can be solved using the same basic algorithm defined in 3.5.1.

3.5.1 Maximum Flow

This implementation of Ford-Fulkerson algorithm (known as Edmonds-Karp algorithm) uses BFS to check if it's possible to expand the flow through the graph and assign depths from the source. Later it uses DFS to expand te flow.

```
1  ll flow[MX][MX];
2  ll used[MX][MX];
3
4  vector<int> adj[MX];
5  vector<int> radj[MX];
6
7  bool vis[MX];
8
9  bool bfs(int s, int lvl[], const int n){
10     queue<int> q;
11     q.push(s);
12     lvl[s] = 1;
13     while(!q.empty()){
14         int p = q.front();
15         q.pop();
16         for(int v : adj[p]){
17             if(lvl[v] == 0 and flow[p][v] - used[p][v] > 0){
18                 lvl[v] = lvl[p]+1;
19                 q.push(v);
20             }
21         }
22         for(int v : radj[p]){
23             if(lvl[v] == 0 and used[v][p] > 0){
24                 lvl[v] = lvl[p]+1;
25                 q.push(v);
26             }
27         }
28     }
29     return lvl[n] > 0;
30 }
31
32 int dfs(int x, ll val, const int lvl[], const int n){
33     vis[x] = true;
34     if(x == n){
35         return val;
36     }
37     for(int v : adj[x]){
38         if(lvl[v] == lvl[x]+1 and flow[x][v] - used[x][v] > 0 and !vis[v]){
39             int r = dfs(v, min(val, flow[x][v]-used[x][v]), lvl, n);
40             if(r > 0){
41                 used[x][v] += r;
42                 return r;
43             }
44         }
```

```
45        }
46        for(int v : radj[x]){
47            if(lvl[v] == lvl[x]+1 and used[v][x] > 0 and !vis[v]){
48                int r = dfs(v, min(val, used[v][x]), lvl, n);
49                if(r > 0){
50                    used[v][x] -= r;
51                    return r;
52                }
53            }
54        }
55        return 0;
56  }
```

To find the flow:

```
1     long long int res = 0;
2
3     while(bfs(1, lvl, n)){
4         res += dfs(1, __INT_MAX__, lvl, n);
5         for(int i = 0 ; i < MX; i++){
6             vis[i] = false;
7             lvl[i] = 0;
8         }
9     }
```

3.5.2 Minimum Cut

To find Minimum Cut in a graph we need to find the Maximum Flow and check which edges connect two created disjont sets of nodes. This modification to finds them:

```
1     long long int res = 0;
2
3     while(bfs(1, lvl, n)){
4         res += dfs(1, __INT_MAX__, lvl, n);
5         for(int i = 0 ; i < MX; i++){
6             vis[i] = false;
7             lvl[i] = 0;
8         }
9     }
10    cout << res << endl;
11    for(int i = 0 ; i < MX; i++){
12        vis[i] = false;
13        lvl[i] = 0;
14    }
15
16    bfs(1, lvl, n);
17    for(int a = 1; a <= n; a++){
18        if(lvl[a] > 0){
19            for(int b : adj[a]){
20                if(lvl[b] == 0){
21                    cout << a << " " << b << endl;
22                }
23            }
24        }
25    }
```

3.5.3 Maximum Matching

3.6 2SAT

Given logical formula in the conjunctive normal form:

$$(a_1 \vee b_1) \wedge (a_2 \vee b_2) \wedge ... \wedge (a_n \vee b_n)$$

we can eliminate disjunction by repacing each $(a_i \vee b_i)$ element with pair:

$$\neg a_i \to b_i \wedge \neg b_i \to a_i$$

```cpp
vector<int> adj[MX];
vector<int> Tadj[MX];
stack<int> scc_stack;

int scc[MX];

bool vis[MX];
int scc_val[MX];

int not_node(int a){
    if(!(a&1))
        return a+1;
    else
        return a-1;
}

void dfs1(int a){
    vis[a] = true;
    for(int b : adj[a]){
        if(!vis[b])
            dfs1(b);
    }
    scc_stack.push(a);
}

void dfs2(int a, int scc_n){
    vis[a] = true;
    for(int b : Tadj[a]){
        if(!vis[b])
            dfs2(b, scc_n);
    }
    scc[a] = scc_n;
}

void zero(){
    for(int i = 0 ; i < MX; i++)
        vis[i] = false;
}

int main(){
    int n, m;
    cin >> n >> m;
    for(int i = 0 ; i < n; i++){
        char op;
        int a, b;
        cin >> op >> a;
        if(op == '+'){
            a *= 2;
        }
        else{
            a *= 2;
            a++;
        }
        cin >> op >> b;
        if(op == '+'){
            b *= 2;
        }
        else{
            b *= 2;
            b++;
        }
        Tadj[a].push_back(not_node(b));
        Tadj[b].push_back(not_node(a));
        adj[not_node(b)].push_back(a);
        adj[not_node(a)].push_back(b);
    }

    for(int i = 1 ; i <= 2*m+1; i++){
        if(!vis[i])
            dfs1(i);
    }

    zero();
```

```
74      int scc_n = 1;
75      while(!scc_stack.empty()){
76          int a = scc_stack.top();
77          scc_stack.pop();
78          if(scc[a] == 0){
79              dfs2(a, scc_n);
80              scc_n++;
81          }
82      }
83
84      for(int i = 2; i <= m*2; i+=2){
85          if(scc[i] == scc[i+1]){
86              cout << "IMPOSSIBLE" << endl;
87              return 0;
88          }
89      }
90      // 1 - false; 2 - true;
91      for(int i = 2; i <= 2*m; i+=2){
92          int a = scc[i];
93          int b = scc[i+1];
94          if(scc_val[a] != 0){
95              scc_val[b] = (scc_val[a] == 1 ? 2 : 1);
96          }
97          if(scc_val[b] != 0){
98              scc_val[a] = (scc_val[b] == 1 ? 2 : 1);
99          }
100         if(a < b){
101             scc_val[a] = 1;
102             scc_val[b] = 2;
103         }
104         else{
105             scc_val[a] = 2;
106             scc_val[b] = 1;
107         }
108     }
109
110     for(int i = 2; i <= m*2; i+=2)
111         cout << (scc_val[scc[i]] == 2 ? "+" : "-") << " ";
112
113     return 0;
114 }
```

4   Range Queries

4.1   Segment trees

BASE size table:

a	$2 \cdot 10^5$	$5 \cdot 10^5$	10^6
log_2a	18	19	20

4.1.1   Point-Range Trees

```
1  int sal[2*BASE];
2
3  void add(int p, int val){
4      p += BASE;
5      sal[p] += val;
6      while(p > 0){
7          p >>= 1;
8          sal[p] = sal[2*p] + sal[2*p+1];
9      }
10 }
11
12 int query(int p, int q){
13     int ret =0 ;
14     p += BASE -1;
15     q += BASE +1;
16     while(p>>1 != q>>1){
17         if(not(p&1)){
```

```
18          ret += sal[p+1];
19      }
20      if(q&1){
21          ret += sal[q-1];
22      }
23      p >>= 1;
24      q >>= 1;
25  }
26  return ret;
27 }
```

4.1.2 Range-Range Trees

1. range ADD insert, range MAX value query

```
1  void max_add(int a, int b, int val, int k=1, int x=0, int y=BASE-1){
2      if(a <= x and y <= b){
3          max_tree[k][1] += val;
4          return;
5      }
6      max_tree[2*k][1] += max_tree[k][1];
7      max_tree[2*k+1][1] += max_tree[k][1];
8      max_tree[k][0] += max_tree[k][1];
9      max_tree[k][1] = 0;
10
11     int d = (x+y)/2;
12     if(a <= d){
13         max_add(a, b, val, 2*k, x, d);
14         }
15     if(b > d){
16         max_add(a, b, val, 2*k+1, d+1, y);
17     }
18     max_tree[k][0] = max(max_tree[2*k+1][0] + max_tree[2*k+1][1], max_tree[2*k][0]
         + max_tree[2*k][1]);
19
20 }
```

```
1  ll max_query(int a, int b, int k=1, int x=0, int y=BASE-1){
2      if(a <= x and y <= b){
3          return max_tree[k][0] + max_tree[k][1];
4      }
5      max_tree[2*k][1] += max_tree[k][1];
6      max_tree[2*k+1][1] += max_tree[k][1];
7      max_tree[k][0] += max_tree[k][1];
8      max_tree[k][1] = 0;
9      int d = (x+y)/2;
10     ll ret = -INF;
11     if(a <= d){
12         ret = max(ret, max_query(a, b, 2*k, x, d));
13     }
14     if(b > d){
15         ret = max(ret, max_query(a, b, 2*k+1, d+1, y));
16     }
17     return ret;
18 }
```

2. range ADD insert, range SUM qyerry

```
1  void sum_add(int a, int b, int val, int k=1, int x=0, int y=BASE-1){
2      if(a <= x and y <= b){
3          sum_tree[k][1] += val;
4          return;
5      }
6      sum_tree[2*k][1] += sum_tree[k][1];
7      sum_tree[2*k+1][1] += sum_tree[k][1];
8      sum_tree[k][0] += sum_tree[k][1]*(y-x+1);
9      sum_tree[k][1] = 0;
10     int d = (x+y)/2;
11     if(a <= d){
12         int w = (d <= b ? d : b)+1;
13         w -= (x <= a ? a : x);
14         sum_tree[k][0] += val*(w);
15         sum_add(a, b, val, 2*k, x, d);
16     }
```

```cpp
        if(b > d){
            int w = (y <= b ? y : b)+1;
            w -= (d+1 <= a ? a : d+1);
            sum_tree[k][0] += val*(w);
            sum_add(a, b, val, 2*k+1, d+1, y);
        }
    }
```

```cpp
ll sum_query(int a, int b, int k=1, int x=0, int y=BASE-1){
    if(a <= x and y <= b){
        return sum_tree[k][0] + sum_tree[k][1]*(y-x+1);
    }
    sum_tree[2*k][1] += sum_tree[k][1];
    sum_tree[2*k+1][1] += sum_tree[k][1];
    sum_tree[k][0] += sum_tree[k][1]*(y-x+1);
    sum_tree[k][1] = 0;
    int d = (x+y)/2;
    ll ret = 0;
    if(a <= d){
        ret += sum_query(a, b, 2*k, x, d);
    }
    if(b > d){
        ret += sum_query(a, b, 2*k+1, d+1, y);
    }
    return ret;
}
int sal[2*BASE];
```

4.2   Max Subarray Sum

```cpp
ll ssum[2*BASE];
ll prfx[2*BASE];
ll sufx[2*BASE];
ll tsum[2*BASE];

void insert(int val, int p){
    p+=BASE;
    ssum[p] = val;
    prfx[p] = val;
    sufx[p] = val;
    tsum[p] = val;
    int l, r;
    while(p){
        p >>= 1;
        l = 2*p;
        r = 2*p+1;
        tsum[p] = tsum[l] + tsum[r];
        prfx[p] = max(prfx[l], tsum[l] + prfx[r]);
        sufx[p] = max(sufx[l] + tsum[r], sufx[r]);
        ssum[p] = max(sufx[l] + prfx[r], max(ssum[l], ssum[r]));
    }
}
```

5   Tree Algorithms

5.1   Binary Lifitng

```cpp
int main(){
    int n, q;
    cin >> n >> q;
    boss[1][0] = 0;
    for(int i = 2; i <= n; i++){
        cin >> boss[i][0];
    }
    for(int i = 1; i < LOG_MX; i++){
        for(int emp = 1; emp <= MX; emp++){
            boss[emp][i] = boss[boss[emp][i-1]][i-1];
        }
    }
```

5.2 Tree Traversal

```
1  vector <int> adj[MX];
2  int subtree[MX];
3  int flat[MX];
4  int vals[MX];
5
6  int idx = 1;
7  int dfs(int a, int p){
8      flat[a] = idx;
9      idx++;
10     for(int b : adj[a]){
11         if(b == p)
12             continue;
13         subtree[a] += dfs(b, a)+1;
14     }
15     return subtree[a];
16 }
```

5.3 LCA

LCA can be found using binary lifting or by building a segment tree build on pre- and postorder.

6 String Algorithms

6.1 Trie

```
1  int trie[TMX][30];
2  bool stop[TMX];
3  int next_node = 1;
4  int dp[MX];
5
6  void insert(string s){
7      int idx = 0;
8      for(char c : s){
9          if(trie[idx][c-'a'] == 0){
10             trie[idx][c-'a'] = next_node;
11             next_node ++;
12         }
13         idx = trie[idx][c-'a'];
14     }
15     stop[idx] = true;
16 }
17
18 int main(){
19     string text;
20     int n;
21     cin >> text >> n;
22     for(int i = 0; i < n; i++){
23         string s;
24         cin >> s;
25         insert(s);
26     }
```

6.2 Pattern Finding

6.2.1 KMP

```
1  int pi[MX];
2
3  int main(){
4      string text, pattern;
5      cin >> text >> pattern;
6      string s = pattern + "+" + text;
7      for(int i = 1; i < s.size(); i++){
8          int j = pi[i-1];
9          while(j > 0 and s[i] != s[j])
10             j = pi[j-1];
11         if(s[i] == s[j])
12             j++;
```

```
13          pi[i] = j;
14      }
15
16      int res = 0;
17      for(int i = 0; i < s.size(); i++){
18          if(pi[i] == pattern.size())
19              res++;
20      }
21
22      cout << res;
23      return 0;
```

6.2.2 Hashing

```
1  const int MOD = 1e9+9;
2  const int P = 9973;
3  const int MX = 1e6+7;
4
5  ll ppow[MX];
6  ll pfx_hash[MX];
7
8  int res = 0;
9
10 int main(){
11     string s, pattern;
12     cin >> s >> pattern;
13     ppow[0] = 1;
14     for(int i = 1 ; i < MX; i++){
15         ppow[i] = (ppow[i-1]*P)%MOD;
16     }
17
18     ll pattern_hash = 0;
19     for(int i = 0; i < pattern.size(); i++){
20         pattern_hash = (pattern_hash + (pattern[i]-'a'+1)*ppow[i])%MOD;
21     }
22
23     for(int i = 0; i < s.size(); i++){
24         pfx_hash[i+1] = (pfx_hash[i] + (s[i]-'a'+1)*ppow[i])%MOD;
25     }
26
27     for(int i = 0; i+pattern.size()-1 < s.size(); i++){
28         int a = i, b = i+pattern.size();
29         ll sub_hash = pfx_hash[b] - pfx_hash[a]+MOD;
30         sub_hash %= MOD;
31         if(sub_hash == (pattern_hash*ppow[a])%MOD){
32             res++;
33         }
34     }
35     cout << res << endl;
36     return 0;
```

6.3 Palindormes

Fiding longest palindromic substring.

```
int p[MX];

int main(){
    string txt1, txt = "#";
    cin >> txt1;
    for(char c : txt1){
        string s{c};
        txt += s + "#";
    }
    txt = "^" + txt + "$";
    int l = 1, r = 1;
    for(int i = 1; i < txt.size(); i++){
        p[i] = max(0, min(r-i, p[l+r-i]));
        while(txt[i-p[i]] == txt[i+p[i]])
            p[i]++;
        if(i+p[i] > r){
            l = i-p[i];
            r = i+p[i];
        }
    }
    int maxi = 0;
    int imaxi;
    for(int i = 0; i < txt.size(); i++){
        if(p[i] > maxi){
            maxi = p[i];
            imaxi = i;
        }
    }
    string res = txt.substr(imaxi-maxi+1,maxi*2-1);
    for(auto c : res){
        if(c != '#')
            cout << c;
    }
```

7 Mathematics

7.1 Fermat's Theorem

Modular inverse:

$$\text{If m is prime, then: } x^{-1} \bmod m = x^{m-2} \bmod m$$

$$x^{\varphi(m)} \bmod m = x^{k \bmod (m-1)} \bmod m$$

```
int exp(int a, int b, int MOD){
    if(b == 0)
        return 1;
    if(b & 1){
        return ((ll)a * exp(a, b-1, MOD))%MOD;
    }
    ll tmp = exp(a, b/2, MOD);
    return (tmp*tmp)%MOD;
}
```

7.2 Fast Fibonacci

```
void mul(ll a[][2], ll b[][2]){
    ll res[2][2];
    res[0][0] = a[0][0]*b[0][0] + a[0][1]*b[1][0];
    res[0][1] = a[0][0]*b[0][1] + a[0][1]*b[1][1];
    res[1][0] = a[1][0]*b[0][0] + a[1][1]*b[1][0];
    res[1][1] = a[1][0]*b[0][1] + a[1][1]*b[1][1];
```

```
 7
 8      a[0][0] = res[0][0] % MOD;
 9      a[0][1] = res[0][1] % MOD;
10      a[1][0] = res[1][0] % MOD;
11      a[1][1] = res[1][1] % MOD;
12 }
13
14 int main(){
15      ll n;
16      cin >> n;
17      ll m_pow[2][2] =
18              {{0, 1},
19               {1, 1}};
20      ll m[2][2] =
21              {{1, 0},
22               {0, 1}};
23
24      auto pow = bitset<64> (n);
25
26      for(int i = 0 ; i < 64; i++){
27          if(pow[i]){
28              mul(m,m_pow);
29          }
30          mul(m_pow, m_pow);
31      }
32
33      cout << m[0][1];
34
35      return 0;
36 }
```

8 Geometry

8.1 Nie wiem co to robi

```
 1 #include <iostream>
 2
 3 using namespace std;
 4
 5 struct P {
 6      int x, y;
 7      void read() {
 8          cin >> x >> y;
 9      }
10      P operator-(const P &b) const {
11          return P{x - b.x, y - b.y};
12      }
13      void operator-=(const P &b) {
14          x -= b.x;
15          y -= b.y;
16      }
17      long long operator *(const P& b) const {
18          return (long long)x * b.y - (long long) y * b.x;
19      }
20
21      bool operator <(const P& b) const;
22 };
23
24 P most_left;
25
26 bool P::operator<(const P& b) const {
27      long long res = (*this - most_left) * (b - most_left);
28      if (res == 0) {
29          return x < b.x;
30      }
31      return res > 0;
32 }
33 int main() {
34      int t;
35      cin >> t;
36      while (t--) {
37          P a, b, c;
```

```
38          a.read();
39          b.read();
40          c.read();
41
42          long long res = (b - a) * (c - a);
43          if (res > 0)
44              cout << "LEFT\n";
45          else if (res < 0) {
46              cout << "RIGHT\n";
47          } else {
48              cout << "TOUCH\n";
49          }
50      }
51
52      return 0;
53  }
```

9   Other Algorithms