Codebook

Pitoni++

Žiga Gosar, Maks Kolman, Jure Slak

verzija: 30. december 2014

Kazalo

1 Grafi

1.1 Topološko sortiranje

Vhod: Število vozlišč n in število povezav m ter seznam povezav E oblike $u \to v$ dolžine m. Usmerjen graf G je tako sestavljen iz vozlišč z oznakami 0 do $n-1$ in povezavami iz E. G ne sme imeti zank, če pa jih ima, se jih lahko brez škode odstrani.

Izhod: Topološka ureditev usmerjenega grafa G, to je seznam vozlišč v takem vrstem redu, da nobena povezava ne kaže nazaj. Če je vrnjeni seznam krajši od n, potem ima G cikle.

Časovna zahtevnost: $O(V + E)$

Prostorska zahtevnost: $O(V)$

Testiranje na terenu: UVa 10305

```
vector<int> topological_sort(int n, int m, const int E[][2]) {
    vector<vector<int>> G(n);
    vector<int> ingoing(n, 0);

    for (int i = 0; i < m; ++i) {
        int a = E[i][0], b = E[i][1];
        G[a].push_back(b);
        ingoing[b]++;
    }

    queue<int> q;  // morda priority_queue, če je vrstni red pomemben
    for (int i = 0; i < n; ++i)
        if (ingoing[i] == 0)
            q.push(i);

    vector<int> res;
    while (!q.empty()) {
        int t = q.front();
        q.pop();

        res.push_back(t);

        for (int v : G[t])
            if (--ingoing[v] == 0)
                q.push(v);
    }

    return res;  // če res.size() != n, ima graf cikle.
}
```

1.2 Mostovi in prerezna vozlišča grafa

Vhod: Število vozlišč n in število povezav m ter seznam povezav E oblike $u \to v$ dolžine m. Neusmerjen graf G je tako sestavljen iz vozlišč z oznakami 0 do $n-1$ in povezavami iz E.

Izhod: Seznam prereznih vozlišč: točk, pri katerih, če jih odstranimo, graf razpade na dve komponenti in seznam mostov grafa G: povezav, pri katerih, če jih odstranimo, graf razpade na dve komponenti.

Časovna zahtevnost: $O(V)$

Prostorska zahtevnost: $O(V + E)$

Testiranje na terenu: UVa 315

```
namespace {
vector<int> low;
vector<int> dfs_num;
vector<int> parent;
}
```

```
6
7   void articulation_points_and_bridges_internal(int u, const vector<vector<int>>& G,
8           vector<bool>& articulation_points_map, vector<pair<int, int>>& bridges) {
9       static int dfs_num_counter = 0;
10      low[u] = dfs_num[u] = ++dfs_num_counter;
11      int children = 0;
12      for (int v : G[u]) {
13          if (dfs_num[v] == -1) {  // unvisited
14              parent[v] = u;
15              children++;
16
17              articulation_points_and_bridges_internal(v, G, articulation_points_map, bridges);
18              low[u] = min(low[u], low[v]);  // update low[u]
19
20              if (parent[u] == -1 && children > 1)  // special root case
21                  articulation_points_map[u] = true;
22              else if (parent[u] != -1 && low[v] >= dfs_num[u])  // articulation point
23                  articulation_points_map[u] = true;  // assigned more than once
24              if (low[v] > dfs_num[u])                // bridge
25                  bridges.push_back({u, v});
26          } else if (v != parent[u]) {
27              low[u] = min(low[u], dfs_num[v]);  // update low[u]
28          }
29      }
30  }
31
32  void articulation_points_and_bridges(int n, int m, const int E[][2],
33          vector<int>& articulation_points, vector<pair<int, int>>& bridges) {
34      vector<vector<int>> G(n);
35      for (int i = 0; i < m; ++i) {
36          int a = E[i][0], b = E[i][1];
37          G[a].push_back(b);
38          G[b].push_back(a);
39      }
40
41      low.assign(n, -1);
42      dfs_num.assign(n, -1);
43      parent.assign(n, -1);
44
45      vector<bool> articulation_points_map(n, false);
46      for (int i = 0; i < n; ++i)
47          if (dfs_num[i] == -1)
48              articulation_points_and_bridges_internal(i, G, articulation_points_map, bridges);
49
50      for (int i = 0; i < n; ++i)
51          if (articulation_points_map[i])
52              articulation_points.push_back(i);  // actually return only articulation points
53  }
```

2 Teorija števil

2.1 Evklidov algoritem

Vhod: $a, b \in \mathbb{Z}$

Izhod: Največji skupni delitelj a in b. Za pozitivna števila je pozitiven, če je eno število 0, je rezultat drugo število, pri negativnih je predznak odvisen od števila iteracij.

Časovna zahtevnost: $O(\log(a) + \log(b))$

Prostorska zahtevnost: $O(1)$

```
1   int gcd(int a, int b) {
2       int t;
3       while (b != 0) {
4           t = a % b;
5           a = b;
6           b = t;
7       }
8       return a;
9   }
```

2.2 Razširjen Evklidov algoritem

Vhod: $a, b \in \mathbb{Z},$. Števili $retx$, $rety$ sta parametra samo za vračanje vrednosti.

Izhod: Števila x, y, d, pri čemer $d = \gcd(a, b)$, ki rešijo Diofantsko enačbo $ax + by = d$. V posebnem primeru, da je b tuj a, je x inverz števila a v multiplikativni grupi Z_b^*.

Časovna zahtevnost: $O(\log(a) + \log(b))$

Prostorska zahtevnost: $O(1)$

Testiranje na terenu: UVa 756

```
int ext_gcd(int a, int b, int& retx, int& rety) {
    int x = 0, px = 1, y = 1, py = 0, r, q;
    while (b != 0) {
        r = a % b; q = a / b;  // quotient and reminder
        a = b; b = r;          // gcd swap
        r = px - q * x;        // x swap
        px = x; x = r;
        r = py - q * y;        // y swap
        py = y; y = r;
    }
    retx = px; rety = py;      // return
    return a;
}
```

2.3 Kitajski izrek o ostankih

Vhod: Sistem n kongruenc $x \equiv a_i \pmod{m_i}$, m_i so paroma tuji.

Izhod: Število x, ki reši ta sistem dobimo po formuli

$$x = \left[\sum_{i=1}^{n} a_i \frac{M}{m_i} \left[\left(\frac{M}{mi} \right)^{-1} \right]_{m_i} \right]_M, \qquad M = \prod_{i=1}^{n} m_i,$$

kjer $[x^{-1}]_m$ označuje inverz x po modulu m. Vrnjeni x je med 0 in M.

Časovna zahtevnost: $O(n \log(\max\{m_i, a_i\}))$

Prostorska zahtevnost: $O(n)$

Potrebuje: Evklidov algoritem (str. 4)

Testiranje na terenu: UVa 756

Opomba: Pogosto potrebujemo `unsigned long long` namesto `int`.

```
int mul_inverse(int a, int m) {
    int x, y;
    ext_gcd(a, m, x, y);
    return (x + m) % m;
}

int chinese_reminder_theorem(const vector<pair<int, int>>& cong) {
    int M = 1;
    for (size_t i = 0; i < cong.size(); ++i) {
        M *= cong[i].second;
    }
    int x = 0, a, m;
    for (const auto& p : cong) {
        tie(a, m) = p;
        x += a * M / m * mul_inverse(M/m, m);
        x %= M;
    }
    return (x + M) % M;
}
```

2.4 Hitro potenciranje

Vhod: Število g iz splošne grupe in $n \in \mathbb{N}_0$.

Izhod: Število g^n.
Časovna zahtevnost: $O(\log(n))$
Prostorska zahtevnost: $O(1)$
Testiranje na terenu: `http://putka.upm.si/tasks/2010/2010_3kolo/nicle`

```
int fast_power(int g, int n) {
    int r = 1;
    while (n > 0) {
        if (n & 1) {
            r *= g;
        }
        g *= g;
        n >>= 1;
    }
    return r;
}
```

2.5 Številski sestavi

Vhod: Število $n \in \mathbb{N}_0$ ali $\frac{p}{q} \in Q$ ter $b \in [2, \infty) \cap \mathbb{N}$.
Izhod: Število n ali $\frac{p}{q}$ predstavljeno v izbranem sestavu z izbranimi števkami in označeno periodo.
Časovna zahtevnost: $O(\log(n))$ ali $O(q \log(q))$.
Prostorska zahtevnost: $O(n)$ ali $O(q)$.
Testiranje na terenu: `http://putka.upm.si/tasks/2010/2010_finale/ulomki`
Opomba: Zgornja meja za bazo b je dolžina niza `STEVILSKI_SESTAVI_ZNAKI`.

```
char STEVILSKI_SESTAVI_ZNAKI[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";

string convert_int(int n, int baza) {
    if (n == 0) return "0";
    string result;
    while (n > 0) {
        result.push_back(STEVILSKI_SESTAVI_ZNAKI[n % baza]);
        n /= baza;
    }
    reverse(result.begin(), result.end());
    return result;
}

string convert_fraction(int stevec, int imenovalec, int base) {
    div_t d = div(stevec, imenovalec);
    string result = convert_int(d.quot, base);
    if (d.rem == 0) return result;

    string decimalke;  // decimalni del
    result.push_back('.');
    int mesto = 0;
    map<int, int> spomin;
    spomin[d.rem] = mesto;
    while (d.rem != 0) {  // pisno deljenje
        mesto++;
        d.rem *= base;
        decimalke += STEVILSKI_SESTAVI_ZNAKI[d.rem / imenovalec];
        d.rem %= imenovalec;
        if (spomin.count(d.rem) > 0) {  // periodicno
            result.append(decimalke.begin(), decimalke.begin() + spomin[d.rem]);
            result.push_back('(');
            result.append(decimalke.begin() + spomin[d.rem], decimalke.end());
            result.push_back(')');
            return result;
        }
        spomin[d.rem] = mesto;
    }
    result += decimalke;
    return result;  // koncno decimalno stevilo
}
```

3 Eulerjeva funkcija ϕ

Vhod: Število $n \in \mathbb{N}$.

Izhod: Število $\phi(n)$, to je število števil manjših ali enakih n in tujih n. Direktna formula:

$$\phi(n) = n \cdot \prod_{p|n} (1 - \frac{1}{p})$$

Časovna zahtevnost: $O(\sqrt{n})$.

Prostorska zahtevnost: $O(1)$.

Testiranje na terenu: `https://projecteuler.net/problem=69`

```
1  int euler_phi(int n) {
2      int res = n;
3      for (int i = 2; i*i <= n; ++i) {
4          if (n % i == 0) {
5              while (n % i == 0) {
6                  n /= i;
7              }
8              res -= res / i;
9          }
10     }
11     if (n > 1) res -= res / n;
12     return res;
13 }
```