Bamboo Team Notes

Contents

1 Number theory

1.1 Extended Euclide

```cpp
int bezout(int a, int b) {
    // return x such that ax + by == gcd(a, b)
    int xa = 1, xb = 0;
    while (b) {
        int q = a / b;
        int r = a - q * b, xr = xa - q * xb;
        a = b; xa = xb;
        b = r; xb = xr;
    }
    return xa;
}

pair<int, int> solve(int a, int b, int c) {
    // solve ax + by == c
    int d = __gcd(a, b);
    int x = bezout(a, b);
    int y = (d - a * x) / b;
    c /= d;
    return make_pair(x * c, y * c);
}

int main() {
    int a = 100, b = 128;
    int c = __gcd(a, b);
    int x = bezout(a, b);
    int y = (c - a * x) / b;
    cout << x << ' ' << y << endl;
    pair<int, int> xy = solve(100, 128, 40);
    cout << xy.first << ' ' << xy.second << endl;
    return 0;
}
```

1.2 System of linear equations

```cpp
// extended version, uses diophantine equation solver to solve system of congruent equations
pair<int, int> solve(int a, int b, int c) {
    // solve ax + by == c
    int d = __gcd(a, b);
    int x = bezout(a / d, b / d);
    int y = (d - a * x) / b;
    c /= d;
    return make_pair(x * c, y * c);
}

int lcm(int a, int b) {
    return a / __gcd(a, b) * b;
}

int solveSystem(vector<int> a, vector<int> b) {
    // xi mod bi = ai
    int A = a[0], B = b[0];
    // x mod B = A
    for (int i = 1; i < a.size(); ++i) {
        int curB = b[i], curA = a[i];
        // x = Bi + A = curB * j + curA
        pair<int, int> ij = solve(B, -curB, curA - A);
        assert(B * ij.first + A == curB * ij.second + curA);
        int newA = (B * ij.first + A);
        B = lcm(B, curB);
        A = newA % B;
        if (i + 1 == a.size()) return A;
    }
}

int main() {
    vector<int> a = {0, 3, 3};
    vector<int> b = {3, 6, 9};
    cout << solveSystem(a, b) << endl;
    return 0;
}
```

2 String

2.1 Suffix Array

```cpp
#include <bits/stdc++.h>

using namespace std;

struct SuffixArray {
    static const int N = 100010;

    int n;
    char *s;
    int sa[N], tmp[N], pos[N];
    int len, cnt[N], lcp[N];

    SuffixArray(char *t) {
        s = t;
        n = strlen(s + 1);
        buildSA();
    }

    bool cmp(int u, int v) {
        if (pos[u] != pos[v]) {
            return pos[u] < pos[v];
        }
        return (u + len <= n && v + len <= n) ? pos[u + len] < pos[v + len] : u > v;
    }

    void radix(int delta) {
        memset(cnt, 0, sizeof cnt);
        for (int i = 1; i <= n; i++) {
            cnt[i + delta <= n ? pos[i + delta] : 0]++;
        }
        for (int i = 1; i < N; i++) {
            cnt[i] += cnt[i - 1];
        }
        for (int i = n; i > 0; i--) {
            int id = sa[i];
            tmp[cnt[id + delta <= n ? pos[id + delta] : 0]--] = id;
        }
        for (int i = 1; i <= n; i++) {
            sa[i] = tmp[i];
        }
    }

    void buildSA() {
        for (int i = 1; i <= n; i++) {
            sa[i] = i;
            pos[i] = s[i];
        }
        len = 1;
```

```
        while (1) {
            radix(len);
            radix(0);
            tmp[1] = 1;
            for (int i = 2; i <= n; i++) {
                tmp[i] = tmp[i - 1] + cmp(sa[i - 1], sa[i]);
            }
            for (int i = 1; i <= n; i++) {
                pos[sa[i]] = tmp[i];
            }
            if (tmp[n] == n) {
                break;
            }
            len <<= 1;
        }

        len = 0;
        for (int i = 1; i <= n; i++) {
            if (pos[i] == n) {
                continue;
            }
            int j = sa[pos[i] + 1];
            while (s[i + len] == s[j + len]) {
                len++;
            }
            lcp[pos[i]] = len;
            if (len) {
                len--;
            }
        }
    }
};
```

2.2 Aho Corasick

```
struct AhoCorasick {
    static const int ALPHABET_SIZE = 26;

    struct Node {
        Node* to[ALPHABET_SIZE];
        Node* fail;
        int ending_length; // 0 if is not ending

        Node() {
            for (int i = 0; i < ALPHABET_SIZE; ++i) to[i] = nullptr;
            fail = nullptr;
            ending_length = false;
        }
    };

    Node* root;

    void add(const string &s) {
        Node* cur_node = root;
        for (char c : s) {
            c -= 'a';
            if (!cur_node->to[c]) {
                cur_node->to[c] = new Node();
            }
            cur_node = cur_node->to[c];
        }
        cur_node->ending_length = s.size();
    }

    AhoCorasick(const vector<string> &a) {
        root = new Node();
        for (const string &s : a) add(s);

        queue<Node*> Q;
        root->fail = root;
        Q.push(root);

        while (!Q.empty()) {
            Node *par = Q.front(); Q.pop();
            for (int c = 0; c < ALPHABET_SIZE; ++c) {
                if (par->to[c]) {
                    par->to[c]->fail = par == root ? root : par->fail->to[c];
                    Q.push(par->to[c]);
                } else {
                    par->to[c] = par == root ? root : par->fail->to[c];
                }
            }
        }
    }
};
```

2.3 Z algorithm

```
vector<int> calcZ(const string &s) {
    int L = 0, R = 0;
    int n = s.size();
    vector<int> Z(n);
    Z[0] = n;
    for (int i = 1; i < n; i++) {
        if (i > R)
        {
            L = R = i;
            while (R < n && s[R] == s[R - L]) R++;
            Z[i] = R - L; R--;
        }
        else
        {
            int k = i - L;
            if (Z[k] < R - i + 1) Z[i] = Z[k];
            else
            {
                L = i;
                while (R < n && s[R] == s[R - L]) R++;
                Z[i] = R - L; R--;
            }
        }
    }
    return Z;
}
```

3 Combinatorial optimization

4 Geometry

5 Numerical algorithms

5.1 Simplex Algorithm

```
/**
 * minimize c^T * x
 * subject to Ax <= b
 * and x >= 0
 * The input matrix a will have the following form
 * 0 c c c c c
 * b A A A A A
 * b A A A A A
 * b A A A A A
 * Result vector will be: val x x x x x
 **/

typedef long double ld;
const ld EPS = 1e-8;
struct LPSolver {
    static vector<ld> simplex(vector<vector<ld>> a) {
        int n = (int) a.size() - 1;
        int m = (int) a[0].size() - 1;
        vector<int> left(n + 1);
        vector<int> up(m + 1);
        iota(left.begin(), left.end(), m);
        iota(up.begin(), up.end(), 0);
        auto pivot = [&](int x, int y) {
            swap(left[x], up[y]);
            ld k = a[x][y];
            a[x][y] = 1;
            vector<int> pos;
            for (int j = 0; j <= m; j++) {
                a[x][j] /= k;
                if (fabs(a[x][j]) > EPS) pos.push_back(j);
            }
            for (int i = 0; i <= n; i++) {
                if (fabs(a[i][y]) < EPS || i == x) continue;
                k = a[i][y];
                a[i][y] = 0;
                for (int j : pos) a[i][j] -= k * a[x][j];
            }
        };
        while (1) {
```

```
    int x = -1;
    for (int i = 1; i <= n; i++) {
        if (a[i][0] < -EPS && (x == -1 || a[i][0] < a[x][0])) {
            x = i;
        }
    }
    if (x == -1) break;
    int y = -1;
    for (int j = 1; j <= m; j++) {
        if (a[x][j] < -EPS && (y == -1 || a[x][j] < a[x][y])) {
        y = j;
        }
    }
    if (y == -1) return vector<ld>(); // infeasible
    pivot(x, y);
}
while (1) {
    int y = -1;
    for (int j = 1; j <= m; j++) {
        if (a[0][j] > EPS && (y == -1 || a[0][j] > a[0][y])) {
            y = j;
        }
    }
    if (y == -1) break;
    int x = -1;
    for (int i = 1; i <= n; i++) {
        if (a[i][y] > EPS && (x == -1 || a[i][0] / a[i][y] < a[x][0] / a[x][y])) {
            x = i;
        }
    }
    if (x == -1) return vector<ld>(); // unbounded
    pivot(x, y);
}
vector<ld> ans(m + 1);
for (int i = 1; i <= n; i++) {
    if (left[i] <= m) ans[left[i]] = a[i][0];
}
ans[0] = -a[0][0];
return ans;
}
};
```

6 Graph algorithms

7 Data structures

8 Miscellaneous