

Resumen de algoritmos para maratones de programación

Diego Alejandro Martínez - Manuel Felipe Pineda

2 de octubre de 2012

Índice

1. Plantilla	2	4.7. Intersección de dos rectas	5
2. Grafos	2	4.8. Intersección de dos segmentos	5
2.1. Dijkstra	2	4.9. Determinar si dos segmentos se intersectan o no	5
2.2. Bellman-Ford	2	4.10. Centro del círculo que pasa por tres puntos	5
2.3. Floyd-Warshall	2	4.11. Par de puntos más cercanos	5
2.4. Johnson	2	4.12. Par de puntos más alejados	5
2.5. Minimum Spanning Tree: Kruskal	2	4.13. Área de un polígono	5
2.6. Minimum Spanning Tree: Prim	2	4.14. Convexhull	5
2.7. Breadth First Search	2	5. Strings	5
2.8. Depth First Search	2	5.1. Knuth-Morris-Pratt KMP	5
2.9. Strongly Connected Components	2	5.2. Aho-Corasick	5
2.10. Puntos de articulación	2	5.3. Suffix Array	5
2.11. 2-SAT	2	6. Teoría de Juegos	5
2.12. Maximum bipartite matching	2	7. Estructuras de Datos	5
2.13. Flujo Máximo	2	7.1. Prefix Tree - Trie	5
2.14. Lowest Common Ancestor: TarjanOLCA	2	7.2. Fenwick Tree	5
3. Matemáticas	2	7.3. Interval Tree	5
3.1. Aritmética Modular	2	8. Hashing	5
3.2. Potencia modular	5	8.1. FNV Hash	5
3.3. Criba de Eratóstenes	5	8.2. JSW Hash	5
4. Geometría	5	9. Misceláneo	5
4.1. Utilidades Geometría	5	9.1. Bitwise operations	5
4.2. Distancia mínima: Punto-Segmento	5		
4.3. Distancia mínima: Punto-Recta	5		
4.4. Determinar si un polígono es convexo	5		
4.5. Determinar si un punto está dentro de un polígono convexo	5		
4.6. Determinar si un punto está dentro de un polígono cualquiera	5		

1. Plantilla

```
#include <cstdio>
#include <cmath>
#include <iostream>
#include <string>
#include <vector>
#include <queue>
#include <stack>
#include <list>
#include <map>

using namespace std;

#define all(x) x.begin(),x.end()
#define rep(i,a,b) for(int i=a;i<b;i++)
#define REP(i,n) rep(i,0,n)
#define foreach(x, v) for (typeof (v).begin() x = (v).begin(); \
x != (v).end(); ++x)
#define D(x) cout << #x " = " << x << endl;

typedef long long int lld;
typedef pair<int,int> pii;
typedef vector<int> vi;
typedef vector<pii> vpii;

int main(){

    return 0;
}
```

.....

2. Grafos

- 2.1. Dijkstra
- 2.2. Bellman-Ford
- 2.3. Floyd-Warshall
- 2.4. Johnson
- 2.5. Minimum Spanning Tree: Kruskal
- 2.6. Minimum Spanning Tree: Prim
- 2.7. Breadth First Search
- 2.8. Depth First Search
- 2.9. Strongly Connected Components
- 2.10. Puntos de articulación
- 2.11. 2-SAT
- 2.12. Maximum bipartite matching
- 2.13. Flujo Máximo
- 2.14. Lowest Common Ancestor: TarjanOLCA

3. Matemáticas

3.1. Aritmética Modular

Colección de códigos útiles para aritmética modular

```
int gcd (int a, int b) {
    int tmp;
    while (b) {
        a %= b;
        tmp = a;
        a = b;
        b = tmp;
    }
}
```

```

    return a;
}

// a % b (valor positivo)
int mod (int a, int b) {
    return ((a % b) + b) % b;
}

// returns d = gcd(a,b); finds x,y such that d = ax + by
int extended_euclid (int a, int b, int &x, int &y) {
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a / b;
        int t = b;
        b = a % b;
        a = t;
        t = xx;
        xx = x - q * xx;
        x = t;
        t = yy;
        yy = y - q * yy;
        y = t;
    }
    return a;
}

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

// finds all solutions to ax = b (mod n)
vi modular_linear_equation_solver (int a, int b, int n) {
    int x, y;
    vi solutions;
    int d = extended_euclid (a, n, x, y);
    if (!(b % d)) {
        x = mod (x * (b / d), n);
        for (int i = 0; i < d; i++)
            solutions.push_back (mod (x + i * (n / d), n));
    }
}

```

```

    }
    return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse (int a, int n) {
    int x, y;
    int d = extended_euclid (a, n, x, y);
    if (d > 1)
        return -1;
    return mod (x, n);
}

// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b. Here, z is unique modulo M = lcm(x,y).
// Return (z,M). On failure, M = -1.
pii chinese_remainder_theorem (int x, int a, int y, int b) {
    int s, t;
    int d = extended_euclid (x, y, s, t);
    if (a % d != b % d)
        return make_pair (0, -1);
    return make_pair (mod (s * b * x + t * a * y, x * y) / d, x * y / d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i. Note that the solution is
// unique modulo M = lcm_i (x[i]). Return (z,M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.
pii chinese_remainder_theorem (const vi &x, const vi &a) {
    pii ret = make_pair (a[0], x[0]);
    for (int i = 1; i < x.size (); i++) {
        ret = chinese_remainder_theorem (ret.first, ret.second, x[i], a[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

// computes x and y such that ax + by = c; on failure, x = y = -1
void linear_diophantine (int a, int b, int c, int &x, int &y) {
    int d = gcd (a, b);
}

```

```
if (c % d) {  
    x = y = -1;  
}  
else {  
    x = c / d * mod_inverse (a / d, b / d);  
    y = (c - a * x) / b;  
}  
}
```

3.2. Potencia modular

3.3. Criba de Eratóstenes

4. Geometría

4.1. Utilidades Geometría

4.2. Distancia mínima: Punto-Segmento

4.3. Distancia mínima: Punto-Recta

4.4. Determinar si un polígono es convexo

4.5. Determinar si un punto está dentro de un polígono convexo

4.6. Determinar si un punto está dentro de un polígono cualquiera

4.7. Intersección de dos rectas

4.8. Intersección de dos segmentos

4.9. Determinar si dos segmentos se intersectan o no

4.10. Centro del círculo que pasa por tres puntos

4.11. Par de puntos más cercanos

4.12. Par de puntos más alejados

4.13. Área de un polígono

4.14. Convexhull

5. Strings

5.1. Knuth-Morris-Pratt KMP

5.2. Aho-Corasick

5.3. Suffix Array

6. Teoría de Juegos

7. Estructuras de Datos

7.1. Prefix Tree - Trie

7.2. Fenwick Tree