Standard Code Library

Your TeamName

Your School

March 18, 2021

Contents

一切的开始

宏定义

```
1   #include<bits/stdc++.h>
2   using namespace std;
3   typedef long long LL;
4   typedef unsigned u32;
5   typedef unsigned long long u64;
6   typedef long double LD;
7   #define il inline
8   #define pln putchar('\n')
9   #define For(i,a,b)  for(int i=(a),(i##i)=(b);i<=(i##i);++i)
10  #define Rep(i,n)    for(int i=0,(i##i)=(n);i<(i##i);++i)
11  #define Fodn(i,a,b) for(int i=(a),(i##i)=(b);i>=(i##i);--i)
12  const int M=1000000007,INF=0x3f3f3f3f;
13  const long long INFLL=0x3f3f3f3f3f3f3f3fLL;
14  const int N=1000010;
15  // -----------------------------------------------------------------------
```

数学

快速幂

- 注意乘法溢出

```
1   inline LLL fp(LLL a, LLL b, LLL Mod) {
2       LLL res = (Mod != 1);
3       for (a %= Mod; b; b >>= 1, a = a * a % Mod)
4           if (b & 1) res = res * a % Mod;
5       return res;
6   }
```

GCD

```
1   template <typename T>
2   inline T gcd(T a, T b) {
3       while (b){
4           T t = b;
5           b = a % b;
6           a = t;
7       }
8       return a;
9   }
10
11  template <typename T>
12  inline T lcm(T a, T b) { return a / gcd(a, b) * b; }
13
14  template <typename T>
15  inline T exgcd(T a, T b, T &x, T &y) {
16      T m = 0, n = 1, t;
17      x = 1, y = 0;
18      while (b){
19          t = m, m = x - a / b * m, x = t;
20          t = n, n = y - a / b * n, y = t;
21          t = b, b = a % b, a = t;
22      }
23      return a;
24  }
```

CRT

- 同余方程合并
- 返回最小正数解或最小非负解无解则返回-1

```
1   inline LL Crt(LL a1, LL a2, LL mod1, LL mod2) {
2       LL u, v;
3       LL g = exgcd(mod1, mod2, u, v);
```

```
4        if ((a2 - a1) % g)
5            return -1;
6        LL m12 = abs(lcm(mod1, mod2));
7        LLL res = (((LLL)mod1 * ((LLL)u * ((a2 - a1) / g) % m12) % m12) + a1) % m12;
8        return res <= 0 ? res + m12 : res; /* 求最小正数解还是非负解 */
9    }
```

线性筛

```
1    struct primenumberlist{
2    #define MAXN (100000000)
3        int cnt, pri[10000000];
4        bool np[MAXN + 10];
5        primenumberlist(){
6            np[1] = 1; cnt = 0;
7            for (int i = 2; i <= MAXN; ++i) {
8                if (!np[i]) pri[++cnt] = i;
9                for (int j = 1; j <= cnt; ++j) {
10                   LL t = pri[j] * i;
11                   if (t > MAXN) break;
12                   np[t] = 1;
13                   if (!(i % pri[j])) break;
14               }
15           }
16       }
17   } prime;
```

φ 欧拉函数

```
1    template <typename T>
2    inline T phi(T x) {
3        T res = x;
4        for (T i = 2; i * i <= x; ++i)
5            if ((x % i) == 0) {
6                res = res / i * (i - 1);
7                while ((x % i) == 0) x /= i;
8            }
9        if (x > 1) res = res / x * (x - 1);
10       return res;
11   }
```

Miller-Rabin 素性测试

- $n <= 10^{18}$

```
1    namespace MillerRabin {
2        const LLL test[]={2ll,325ll,9375ll,28178ll,450775ll,9780504ll,1795265022ll};
3
4        inline bool isprime(LLL n) {
5            if (n==13||n==19||n==73||n==193||n==407521||n==299210837ll)return 1;
6            if (n <= 3) return n > 1;
7            if (n <= 6) return n == 5;
8            if (!(n & 1) || !(n % 3) || !(n % 5)) return 0;
9
10           LLL d = n - 1; int t = 0;
11           while (!(d & 1)) d >>= 1, ++t;
12           for (LLL ai = 0, a = test[0]; ai < 7; ++ai, a = test[ai]) {
13               if (a % n == 0) return 0;
14               LLL v = fp(a, d, n); if (v == 1 || v == n - 1) continue;
15               LLL pre = v;
16               for (int i = 1; i <= t; ++i) {
17                   v = v * v % n;
18                   if (v == 1)
19                       if (pre != 1 && pre != (n - 1)) return 0; else break;
20                   pre = v;
21               }
22               if (v != 1) return 0;
23           }
24           return 1;
```

```
25            }
26    }
```

Pollard-Rho 分解质因数

- 求 n 的一个非平凡因子

- 调用 `pollard_rho()` 前先判断 n 的素性

```cpp
1  namespace PollardRho{
2      mt19937 mt(20011224);//19491001
3
4      inline LLL pollard_rho(LLL n, LLL c) {
5          LLL x = uniform_int_distribution<LL>(1, n - 1)(mt), y = x;
6          LLL val = 1;
7          for (int dep = 1;; dep <<= 1, x = y, val = 1) {
8              for (int stp = 1; stp <= dep; ++stp) {
9                  y = (y * y + c) % n;
10                 val = val * abs(x - y) % n;
11                 if ((stp & 127) == 0) {
12                     LLL d = gcd(val, n);
13                     if (d > 1) return d;
14                 }
15             }
16             LLL d = gcd(val, n);
17             if (d > 1) return d;
18         }
19     }
20
21     //接口根据题意重写
22     vector<LLL> factor;
23     void getfactor(LLL x, LLL c = 19260817) {
24         if (MillerRabin::isprime(x)) {factor.emplace_back(x); return;}
25         LLL p = x;
26         while (p == x) p = pollard_rho(x, c--);
27         getfactor(p); getfactor(x / p);
28     }
29     inline LLL ask(LLL x) {
30         factor.clear();
31         while (!(x & 1)) x >>= 1, factor.emplace_back(2);
32         if (x > 1) getfactor(x);
33         return factor.size();
34     }
35 }
```

组合数

- 数较小模数为较大质数求逆元

-   – 如果模数固定可以 O(n) 预处理阶乘的逆元

- 数较大模数为较小质数用 $Lucas$ 定理

-   –

$$C_n^m \equiv C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{m}{p} \rfloor} * C_{n\ mod\ p}^{m\ mod\ p} (mod\ p)$$

- 数较大模数较小用 $exLucas$ 定理求 $C_n^m mod\ P$

exLucas

- O(P log P)

- 不要求 P 为质数

```cpp
1  namespace EXLUCAS {
2      inline LL idxp(LL n, LL p) {
3          LL nn = n;
4          while (n > 0) nn -= (n % p), n /= p;
```

```
5          return nn / (p - 1);
6      }
7
8      LL facp(LL n, LL p, LL pk) {
9          if (n == 0) return 1;
10         LL res = 1;
11         if (n >= pk) {
12             LL t = n / pk, k = 1, els = n - t * pk;
13             for (LL i = 1; i <= els; ++i) if (i % p) k = k * i % pk;
14             res = k;
15             for (LL i = els + 1; i < pk; ++i) if (i % p) k = k * i % pk;
16             res = res * fp(k, n / pk, pk) % pk;
17         }
18         else for (LL i = 1; i <= n; ++i) if (i % p) res = res * i % pk;
19         return res * facp(n / p, p, pk) % pk;
20     }
21
22     inline LL exlucas(LL n, LL m, LL p, LL pk, LL k) {
23         LL a = facp(n, p, pk) * fp(facp(n - m, p, pk) * facp(m, p, pk) % pk, pk / p * (p - 1) - 1, pk) % pk;
24         LL b = idxp(n, p) - idxp(m, p) - idxp(n - m, p);
25         if (b >= k) return 0;
26         while (b--) a *= p;
27         return a % pk;
28     }
29
30     /* 接口 */ inline LL exlucas(LL n, LL m, LL p) {
31         LL a = 0, b = 1;
32         for (LL i = 2; i * i <= p; ++i) {
33             if (p % i) continue;
34             LL t = 0, pk = 1;
35             while (p % i == 0) ++t, p /= i, pk *= i;
36             a = Crt(a, exlucas(n, m, i, pk, t), b, pk);
37             b *= pk;
38         }
39         return (p > 1) ? Crt(a, exlucas(n, m, p, p, 1), b, p) : a;
40     }
41 }
```

二维计算几何

- Point 直接支持整型和浮点型
- 部分函数可以对整型改写
- 多边形 (凸包) 按逆时针存在下标 `1..n`

点向量基本运算

```
1  template <typename T>
2  struct Point {
3      T x, y;
4      Point() {}
5      Point(T u, T v) : x(u), y(v) {}
6      Point operator+(const Point &a) const { return Point(x + a.x, y + a.y); }
7      Point operator-(const Point &a) const { return Point(x - a.x, y - a.y); }
8      Point operator*(const T &a) const { return Point(x * a, y * a); }
9      T operator*(const Point &a) const { return x * a.x + y * a.y; }
10     T operator%(const Point &a) const { return x * a.y - y * a.x; }
11     double len() const { return hypot(x, y); }
12     double operator^(const Point &a) const { return (a - (*this)).len(); }
13     double angle() const { return atan2(y, x); }
14     bool id() const { return y < 0 || (y == 0 && x < 0); }
15     bool operator<(const Point &a) const { return id() == a.id() ? (*this) % a > 0 : id() < a.id(); }
16 };
17 typedef Point<double> point;
18
19 #define sqr(x) ((x) * (x))
20 const point O(0, 0);
21 const double PI(acos(-1.0)), EPS(1e-8);
22 inline bool dcmp(const double &x, const double &y) { return fabs(x - y) < EPS; }
```

```
23  inline int sgn(const double &x) { return fabs(x) < EPS ? 0 : ((x < 0) ? -1 : 1); }
24  inline double mul(point p1, point p2, point p0) { return (p1 - p0) % (p2 - p0); }
```

位置关系

```
1   inline bool in_same_seg(point p, point a, point b) {
2       if (fabs(mul(p, a, b)) < EPS) {
3           if (a.x > b.x) swap(a, b);
4           return (a.x <= p.x && p.x <= b.x && ((a.y <= p.y && p.y <= b.y) || (a.y >= p.y && p.y >= b.y)));
5       } else return 0;
6   }
7
8   inline bool is_right(point st, point ed, point a) {
9       return ((ed - st) % (a - st)) < 0;
10  }
11
12  inline point intersection(point s1, point t1, point s2, point t2) {
13      return s1 + (t1 - s1) * (((s1 - s2) % (t2 - s2)) / ((t2 - s2) % (t1 - s1)));
14  }
15
16  inline bool parallel(point a, point b, point c, point d) {
17      return dcmp((b - a) % (d - c), 0);
18  }
19
20  inline double point2line(point p, point s, point t) {
21      return fabs(mul(p, s, t) / (t - s).len());
22  }
23
24  inline double point2seg(point p, point s, point t) {
25      return sgn((t - s) * (p - s)) * sgn((s - t) * (p - t)) > 0 ? point2line(p, s, t) : min((p ^ s), (p ^ t));
26  }
```

多边形

求多边形面积

```
1   inline double area(int n, point s[]) {
2       double res = 0;
3       s[n + 1] = s[1];
4       for (int i = 1; i <= n; ++i)
5           res += s[i] % s[i + 1];
6       return fabs(res / 2);
7   }
```

判断点是否在多边形内

- 特判边上的点
- 使用了 a[1]...a[n+1] 的数组

```
1   inline bool in_the_area(point p, int n, point area[]) {
2       bool ans = 0; double x;
3       area[n + 1] = area[1];
4       for (int i = 1; i <= n; ++i) {
5           point p1 = area[i], p2 = area[i + 1];
6           if (in_same_seg(p, p1, p2)) return 1; //特判边上的点
7           if (p1.y == p2.y) continue;
8           if (p.y < min(p1.y, p2.y)) continue;
9           if (p.y >= max(p1.y, p2.y)) continue;
10          ans ^= (((p.y - p1.y) * (p2.x - p1.x) / (p2.y - p1.y) + p1.x) > p.x);
11      }
12      return ans;
13  }
```

凸包

- *Andrew* 算法

- O(n log n)

- 注意是否应该统计凸包边上的点

```cpp
inline bool pcmp1(const point &a, const point &b) { return a.x == b.x ? a.y < b.y : a.x < b.x; }

inline int Andrew(int n, point p[], point ans[]) { //ans[] 逆时针存凸包
    sort(p + 1, p + 1 + n, pcmp1);
    int m = 0;
    for (int i = 1; i <= n; ++i) {
        while (m > 1 && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    int k = m;
    for (int i = n - 1; i >= 1; --i) {
        while (m > k && mul(ans[m - 1], ans[m], p[i]) < 0) --m; //特判凸包边上的点
        ans[++m] = p[i];
    }
    return m - (n > 1); //返回凸包有多少个点
}
```

凸包直径·平面最远点对

- 旋转卡壳算法
- O(n)
- 凸包的边上只能有端点，否则不满足严格单峰
- 使用了 a[1]...a[n+1] 的数组

```cpp
inline double Rotating_Caliper(int n, point a[]) {
    a[n + 1] = a[1];
    double ans = 0;
    int j = 2;
    for (int i = 1; i <= n; ++i) {
        while (fabs(mul(a[i], a[i + 1], a[j])) < fabs(mul(a[i], a[i + 1], a[j + 1]))) j = (j % n + 1);
        ans = max(ans, max((a[j] ^ a[i]), (a[j] ^ a[i + 1])));
    }
    return ans;
}
```

平面最近点对

- 分治 + 归并
- O(n log n)

```cpp
namespace find_the_closest_pair_of_points {
    const int N = 200010; //maxn
    inline bool cmp1(const point &a, const point &b) { return a.x < b.x || (a.x == b.x && a.y < b.y); }
    inline bool operator>(const point &a, const point &b) { return a.y > b.y || (a.y == b.y && a.x > b.x); }

    point a[N], b[N];
    double ans;
    inline void upd(const point &i, const point &j) { ans = min(ans, i ^ j); }

    void find(int l, int r) {
        if (l == r) return;
        if (l + 1 == r) {
            if (a[l] > a[r]) swap(a[l], a[r]);
            upd(a[l], a[r]); return;
        }
        int mid = (l + r) >> 1;
        double mx = (a[mid + 1].x + a[mid].x) / 2;
        find(l, mid); find(mid + 1, r);
        int i = l, j = mid + 1;
        for (int k = l; k <= r; ++k) b[k] = a[((j > r) || (i <= mid && a[j] > a[i])) ? (i++) : (j++)];
        for (int k = l; k <= r; ++k) a[k] = b[k];
        int tot = 0;
        for (int k = l; k <= r; ++k) if (fabs(a[k].x - mx) <= ans) {
            for (int j = tot; j >= 1 && (a[k].y - b[j].y <= ans); --j) upd(a[k], b[j]);
            b[++tot] = a[k];
        }
    }
```

```
29      //接口
30      inline double solve(int n, point ipt[]){
31          ans = 0x3f3f3f3f3f3f3fll; //max distance
32          for (int i = 1; i <= n; ++i) a[i] = ipt[i];
33          sort(a + 1, a + 1 + n, cmp1);
34          find(1, n);
35          return ans;
36      }
37  }
```

圆

三点垂心

```
1   inline point geto(point p1, point p2, point p3) {
2       double a = p2.x - p1.x;
3       double b = p2.y - p1.y;
4       double c = p3.x - p2.x;
5       double d = p3.y - p2.y;
6       double e = sqr(p2.x) + sqr(p2.y) - sqr(p1.x) - sqr(p1.y);
7       double f = sqr(p3.x) + sqr(p3.y) - sqr(p2.x) - sqr(p2.y);
8       return {(f * b - e * d) / (c * b - a * d) / 2, (a * f - e * c) / (a * d - b * c) / 2};
9   }
```
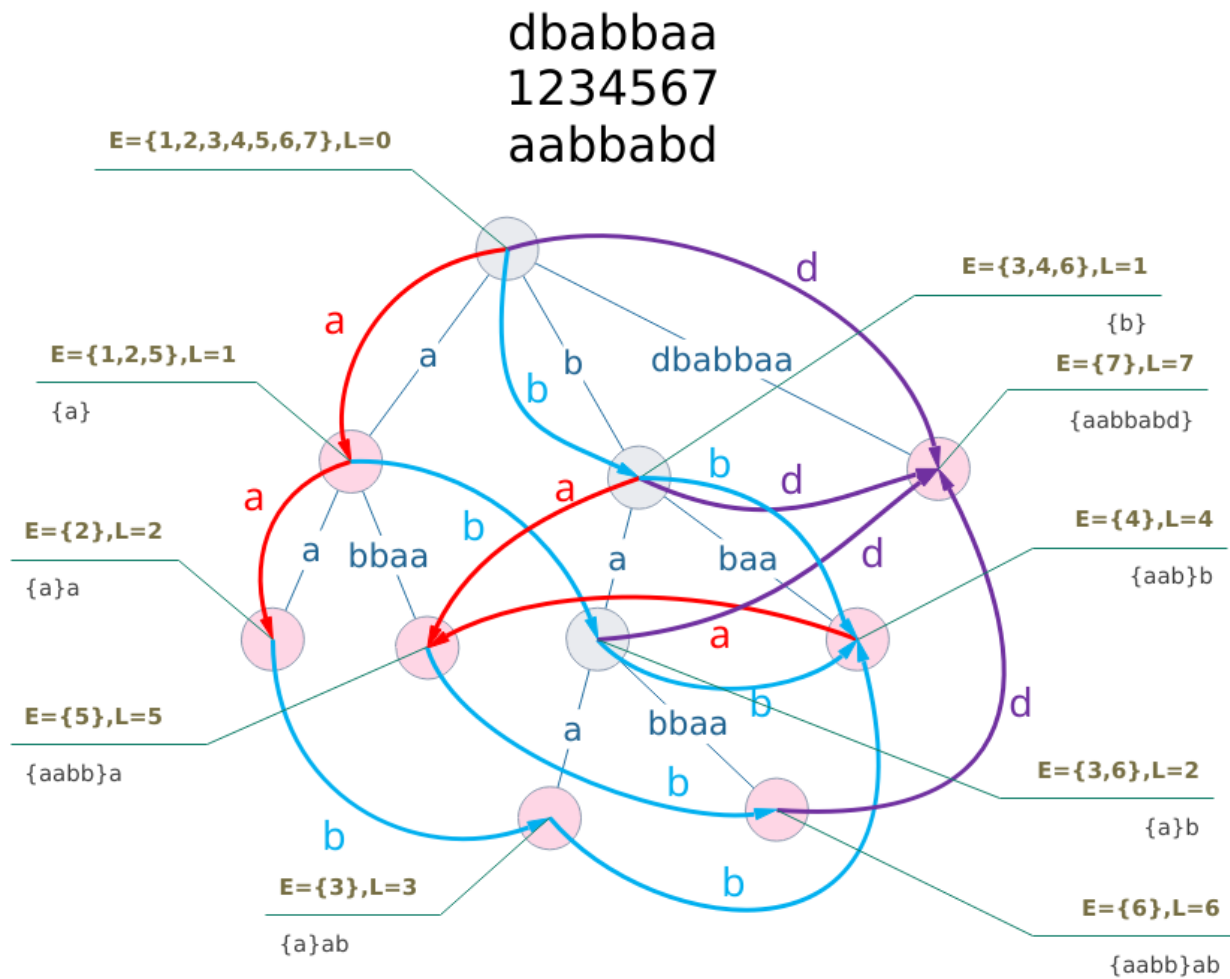
最小覆盖圆

- 随机增量 O(n)

```
1   inline void min_circlefill(point &o, double &r, int n, point a[]) {
2       mt19937 myrand(20011224); shuffle(a + 1, a + 1 + n, myrand); //越随机越难 hack
3       o = a[1];
4       r = 0;
5       for (int i = 1; i <= n; ++i) if ((a[i] ^ o) > r + EPS) {
6           o = a[i];
7           r = 0;
8           for (int j = 1; j < i; ++j) if ((o ^ a[j]) > r + EPS) {
9               o = (a[i] + a[j]) * 0.5;
10              r = (a[i] ^ a[j]) * 0.5;
11              for (int k = 1; k < j; ++k) if ((o ^ a[k]) > r + EPS) {
12                  o = geto(a[i], a[j], a[k]);
13                  r = (o ^ a[i]);
14              }
15          }
16      }
17  }
```

字符串

后缀自动机

杂项

STL

- copy

```
template <class InputIterator, class OutputIterator>
  OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);
```