# Ayudamemoria

My room is random Sorted

October 17, 2024

## Contents

## 1 Template

```cpp
#include <bits/stdc++.h>
using namespace std;

#define forr(i, a, b) for (int i = int(a); i < int(b); i++)
#define forn(i, n) forr(i,0,n)
#define dforr(i, a, b) for (int i = int(b)-1; i >= int(a); i--)
#define dforn(i, n) dforr(i,0,n)
#define all(v) begin(v),end(v)
#define sz(v) (int(size(v)))
#define pb push_back
#define fst first
#define snd second
#define mp make_pair
#define endl '\n'
#define dprint(v) cout << #v " = " << v << endl

typedef long long ll;
typedef pair<int, int> pii;

int main() {
    ios::sync_with_stdio(0); cin.tie(0);
}
```

### 1.1 run.sh

```
1  clear
2  make -s $1 && ./$1 < $2
```

### 1.2 comp.sh

```
1  clear
2  make -s $1 2>&1 | head -$2
```

### 1.3 Makefile

```
1  CXXFLAGS = -std=gnu++2a -O2 -g -Wall -Wextra -Wshadow -Wconversion \
2  -fsanitize=address -fsanitize=undefined
```

## 2 Estructuras de datos

### 2.1 Sparse Table

```
1  #define oper min
2  int st[K][1<<K];int n; // K such that 2^K>n
3  void st_init(vector<int> a){
4      forn(i,n)st[0][i]=a[i];
5      forr(k,1,K)forn(i,n-(1<<k)+1)
6          st[k][i]=oper(st[k-1][i],st[k-1][i+(1<<(k-1))]);
7  }
8  int st_query(int s, int e){
9      int k=31-__builtin_clz(e-s);
10     return oper(st[k][s],st[k][e-(1<<k)]);
11 }
```

### 2.2 Segment Tree

```
1  // Dado un array y una operacion asociativa con neutro, get(i,j)
       opera en [i,j]
2  #define MAXN 100000
3  #define oper(x, y) max(x, y)
4  const int neutro=0;
5  struct RMQ{
6      int sz;
7      tipo t[4*MAXN];
8      tipo &operator[](int p){return t[sz+p];}
9      void init(int n){ // O(nlgn)
10         sz = 1 << (32-__builtin_clz(n));
11         forn(i, 2*sz) t[i]=neutro;
12     }
13     void updall(){dforn(i, sz) t[i]=oper(t[2*i], t[2*i+1]);} //
       O(N)
14     tipo get(int i, int j){return get(i,j,1,0,sz);}
15     tipo get(int i, int j, int n, int a, int b){ // O(lgn)
16         if(j<=a || i>=b) return neutro;
17         if(i<=a && b<=j) return t[n];
18         int c=(a+b)/2;
19         return oper(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
20     }
21     void set(int p, tipo val){ // O(lgn)
22         for(p+=sz; p>0 && t[p]!=val;){
23             t[p]=val;
24             p/=2;
25             val=oper(t[p*2], t[p*2+1]);
26         }
27     }
28 }rmq;
29 // Usage:
30 cin >> n; rmq.init(n); forn(i, n) cin >> rmq[i]; rmq.updall();
```

### 2.3 Segment Tree Lazy

```
1  //Dado un arreglo y una operacion asociativa con neutro, get(i, j)
       opera sobre el rango [i, j).
2  typedef int Elem;//Elem de los elementos del arreglo
3  typedef int Alt;//Elem de la alteracion
4  #define operacion(x,y) x+y
5  const Elem neutro=0; const Alt neutro2=0;
6  #define MAXN 100000
7  struct RMQ{
8      int sz;
9      Elem t[4*MAXN];
10     Alt dirty[4*MAXN];//las alteraciones pueden ser de distinto
       Elem
```

```
11      Elem &operator[](int p){return t[sz+p];}
12      void init(int n){//O(nlgn)
13          sz = 1 << (32-__builtin_clz(n));
14          forn(i, 2*sz) t[i]=neutro;
15          forn(i, 2*sz) dirty[i]=neutro2;
16      }
17      void push(int n, int a, int b){//propaga el dirty a sus hijos
18          if(dirty[n]!=0){
19              t[n]+=dirty[n]*(b-a);//altera el nodo
20              if(n<sz){
21                  dirty[2*n]+=dirty[n];
22                  dirty[2*n+1]+=dirty[n];
23              }
24              dirty[n]=0;
25          }
26      }
27      Elem get(int i, int j, int n, int a, int b){//O(lgn)
28          if(j<=a || i>=b) return neutro;
29          push(n, a, b);//corrige el valor antes de usarlo
30          if(i<=a && b<=j) return t[n];
31          int c=(a+b)/2;
32          return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c,
                  b));
33      }
34      Elem get(int i, int j){return get(i,j,1,0,sz);}
35      //altera los valores en [i, j) con una alteracion de val
36      void alterar(Alt val, int i, int j, int n, int a, int
            b){//O(lgn)
37          push(n, a, b);
38          if(j<=a || i>=b) return;
39          if(i<=a && b<=j){
40              dirty[n]+=val;
41              push(n, a, b);
42              return;
43          }
44          int c=(a+b)/2;
45          alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c,
                  b);
46          t[n]=operacion(t[2*n], t[2*n+1]);//por esto es el push de
                  arriba
47      }
48      void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
49  }rmq;
```

## 2.4   Fenwick Tree

```
1   struct Fenwick{
2       static const int sz=1<<K;
3       ll t[sz]={};
4       void adjust(int p, ll v){
5           for(int i=p+1;i<sz;i+=(i&-i)) t[i]+=v;
6       }
7       ll sum(int p){ // suma [0,p)
8           ll s = 0;
9           for(int i=p;i;i-=(i&-i)) s+=t[i];
10          return s;
11      }
12      ll sum(int a, int b){return sum(b)-sum(a);} // suma [a,b)
13
14      //funciona solo con valores no negativos en el fenwick
15      //longitud del minimo prefijo t.q. suma <= x
16      //para el maximo v+1 y restar 1 al resultado
17      int pref(ll v){
18          int x = 0;
19          for(int d = 1<<(K-1); d; d>>=1){
20              if( t[x|d] < v ) x |= d, v -= t[x];
21          }
22          return x+1;
23      }
24  };
25
26  struct RangeFT { // 0-indexed, query [0, i), update [l, r)
27      Fenwick rate, err;
28      void adjust(int l, int r, int x) { // range update
29          rate.adjust(l, x); rate.adjust(r, -x);
30          err.adjust(l, -x*l); err.adjust(r, x*r);
31      }
32      ll sum(int i) { return rate.sum(i) * i + err.sum(i); }
33  }; // prefix query
34
```

```
35
36  struct Fenwick2D{
37      ll t[N][M]={};
38      void adjust(int p, int q, ll v){
39          for(int i=p+1;i<N;i+=(i&-i))
40              for(int j= q+1; j<M; j+=(j&-j))
41                  t[i][j]+=v;
42      }
43      ll sum(int p,int q){ // suma [0,p)
44          ll s = 0;
45          for(int i=p;i;i-=(i&-i))
46              for(int j=q; j; j-=(j&-j))
47                  s+=t[i][j];
48          return s;
49      }
50      ll sum(int x1, int y1, int x2, int y2){
51          return sum(x2,y2)-sum(x1,y2)-sum(x2,y1)+sum(x1,y1);
52      } // suma [a,b)
53  };
```

### 2.5   Tabla Aditiva

```
1   // Tablita aditiva 2D
2   forn (dim, 2) {
3       forn (i, N) {
4           forn (j, M) {
5               int pi = i-(dim==0), pj = j-(dim==1);
6               if (pi >= 0 && pj >= 0) {
7                   dp[i][j] += dp[pi][pj];
8               }
9           }
10      }
11  }
12  // Generalizacion a 32 dimensiones para mascaras de bits
13  forn (i, 32) {
14      forn (mask, 1<<32) {
15          if ((mask>>i)&1) {
16              dp[mask] += dp[mask - (1<<i)];
17          }
18      }
```

```
19  }
```

### 2.6   Union Find

```
1   vector<int> uf(MAXN, -1);
2   int uf_find(int x) { return uf[x]<0 ? x : uf[x] = uf_find(uf[x]); }
3   bool uf_join(int x, int y){ // True sii x e y estan en !=
        componentes
4       x = uf_find(x); y = uf_find(y);
5       if(x == y) return false;
6       if(uf[x] > uf[y]) swap(x, y);
7       uf[x] += uf[y]; uf[y] = x; return true;
8   }
```

# 3   Matemática

### 3.1   Criba Lineal

```
1   const int N = 10'000'000;
2   vector<int> lp(N+1);
3   vector<int> pr;
4   for (int i=2; i <= N; ++i) {
5       if (lp[i] == 0) lp[i] = i, pr.push_back(i);
6       for (int j = 0; i * pr[j] <= N; ++j) {
7           lp[i * pr[j]] = pr[j];
8           if (pr[j] == lp[i]) break;
9       }
10  }
```

### 3.2   Phollard's Rho

```
1   ll mulmod(ll a, ll b, ll m) { return ll(__int128(a) * b % m); }
2
3   ll expmod(ll b, ll e, ll m) { // O(log b)
4       if (!e) return 1;
5       ll q=expmod(b,e/2,m); q=mulmod(q,q,m);
6       return e%2 ? mulmod(b,q,m) : q;
7   }
8
9   bool es_primo_prob(ll n, int a) {
10      if (n == a) return true;
```

```
11      ll s = 0, d = n-1;
12      while (d%2 == 0) s++, d/=2;
13      ll x = expmod(a,d,n);
14      if ((x == 1) || (x+1 == n)) return true;
15      forn(i,s-1){
16          x = mulmod(x,x,n);
17          if (x == 1) return false;
18          if (x+1 == n) return true;
19      }
20      return false;
21  }
22
23  bool rabin(ll n) { // devuelve true sii n es primo
24      if (n == 1) return false;
25      const int ar[] = {2,3,5,7,11,13,17,19,23};
26      forn(j,9) if (!es_primo_prob(n,ar[j])) return false;
27      return true;
28  }
29
30  ll rho(ll n) {
31      if ((n & 1) == 0) return 2;
32      ll x = 2, y = 2, d = 1;
33      ll c = rand() % n + 1;
34      while (d == 1) {
35          x = (mulmod(x,x,n)+c)%n;
36          y = (mulmod(y,y,n)+c)%n;
37          y = (mulmod(y,y,n)+c)%n;
38          d=gcd(x-y,n);
39      }
40      return d==n ? rho(n) : d;
41  }
42
43  void factRho(map<ll,ll>&prim, ll n){ //O (lg n)^3. un solo numero
44      if (n == 1) return;
45      if (rabin(n)) { prim[n]++; return; }
46      ll factor = rho(n);
47      factRho(factor); factRho(n/factor);
48  }
49  auto factRho(ll n){
50      map<ll,ll>prim;
51      factRho(prim,n);
52      return prim;
53  }
```

### 3.3  Divisores

```
1  // Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
2  void divisores(const map<ll,ll> &f, vector<ll> &divs, auto it, ll
       n=1){
3      if (it==f.begin()) divs.clear();
4      if (it==f.end()) { divs.pb(n); return; }
5      ll p=it->fst, k=it->snd; ++it;
6      forn(_, k+1) divisores(f,divs,it,n), n*=p;
7  }
```

### 3.4  Inversos Modulares

```
1  pair<ll,ll> extended_euclid(ll a, ll b) {
2      if (b == 0) return {1, 0};
3      auto [y, x] = extended_euclid(b, a%b);
4      y -= (a/b)*x;
5      if (a*x + b*y < 0) x = -x, y = -y;
6      return {x, y}; // a*x + b*y = gcd(a,b)
7  }
```

```
1  constexpr ll MOD = 1000000007; // tmb es comun 998'244'353
2  ll invmod[MAXN]; // inversos mdulo MOD hasta MAXN
3  void invmods() { // todo entero en [2,MAXN] debe ser coprimo con
       MOD
4      inv[1] = 1;
5      forr(i, 2, MAXN) inv[i] = MOD - MOD/i*inv[MOD%i]%MOD;
6  }
7
8  // si MAXN es demasiado grande o MOD no es fijo:
9  // versin corta, m debe ser primo. O(log(m))
10 ll invmod(ll a, ll m) { return expmod(a,m-2,m); }
11 // versin larga, a y m deben ser coprimos. O(log(a)), en general
       ms rpido
12 ll invmod(ll a, ll m) { return (extended_euclid(a,m).fst % m + m)
       % m; }
```

# 4 Geometria

```cpp
struct Point
{
    double x, y;
    double Point::operator*(const Point &o) const {
        return x * o.x + y * o.y; }
    double Point::operator^(const Point &o) const {
        return x * o.y - y * o.x; }
    Point Point::operator-(const Point &o) const {
        return {x - o.x, y - o.y}; }
    Point Point::operator+(const Point &o) const {
        return {x + o.x, y + o.y}; }
    Point Point::operator*(const double &u) const {
        return {x * u, y * u}; }
    Point Point::operator/(const double &u) const {
        return {x / u, y / u}; }
    double Point::norm_sq() const {
        return x * x + y * y; }
    double Point::norm() const {
        return sqrt(x * x + y * y); }
};

struct Comp {
    Vector o, v;
    Comp(Vector _o, Vector _v) : o(_o), v(_v) {}
    bool half(Vector p) {
        assert(!(p.x == 0 && p.y == 0));
        return (v ^ p) < 0 || ((v ^ p) == 0 && (v * p) < 0);
    }
    bool operator()(Vector a, Vector b) {
        return mp(half(a - o), 0ll) < mp(half(b - o), ((a - o) ^ (b
            - o)));
    }
};

struct Segment {
    Vector a, b;
    long double eval() const
    { // funcion auxiliar para ordenar segmentos
        assert(a.x != b.x || a.y != b.y);
        Vector a1 = a, b1 = b;
        if (a1.x > b1.x)
            swap(a1, b1);
        assert(x >= a1.x && x <= b1.x);
        if (x == a1.x)
            return a1.y;
        if (x == b1.x)
            return b1.y;
        Vector ab = b1 - a1;
        return a1.y + (x - a1.x) * (ab.y / ab.x);
    }
    bool operator<(Segment o) const
    { // orden de segmentos en un punto (x=cte)
        return (eval() - o.eval()) < -1e-13;
    }
};

bool ccw(const Point &a, const Point &m, const Point &b) {
    return ((a - m) ^ (b - m)) > EPS; }

bool collinear(const Point &a, const Point &b, const Point &c) {
    return fabs((b - a) ^ (c - a)) < EPS; }

double dist_sq(const Point &a, const Point &b) {
    return (a - b).norm_sq(); }

double dist(const Point &a, const Point &b) {
    return (a - b).norm(); }

bool in_segment(const Point &p, const Point &b, const Point &c) {
    return fabs(dist_sq(p, b) + dist_sq(p, c) - dist_sq(b, c)) <
        EPS; }

double angle(const Point &a, const Point &m, const Point &b) {
    Point ma = a - m, mb = b - m;
    return atan2(ma ^ mb, ma * mb);} //atan2l

void sweep_space() {
    vector<Event> eventos; // puntos, segmentos, ...
```

```
77      sort(eventos);        // sort por x, y, ...
78      set<Info> estado;     // mantener la informacion ordenada
79      forn(i, sz(eventos)) {
80          Event &e = eventos[i];
81          process(e, estado); // procesar un evento cambia el estado
82          ans = actualizar(ans);
83 } }
84
85 vector<pt> minkowski_sum(vector<pt> p, vector<pt> q){
86      int n=SZ(p),m=SZ(q),x=0,y=0;
87      fore(i,0,n) if(p[i]<p[x]) x=i;
88      fore(i,0,m) if(q[i]<q[y]) y=i;
89      vector<pt> ans={p[x]+q[y]};
90      fore(it,1,n+m){
91          pt a=p[(x+1)%n]+q[y];
92          pt b=p[x]+q[(y+1)%m];
93          if(b.left(ans.back(),a)) ans.pb(b), y=(y+1)%m;
94          else ans.pb(a), x=(x+1)%n;
95      }
96      return ans; }
```

## 4.1 Lower Envelope

```
1  const ll is_query = -(1LL<<62);
2  struct Line {
3      ll m, b;
4      mutable multiset<Line>::iterator it;
5      const Line *succ(multiset<Line>::iterator it) const;
6      bool operator<(const Line & rhs) const {
7          if (rhs.b != is_query) return m < rhs.m;
8          const Line *s = succ(it);
9          if (!s) return 0;
10         ll x = rhs.m;
11         return b - s->b > (s->m - m) * x;
12     }
13 };
14 struct HullDynamic : public multiset<Line> {
15     bool bad(iterator y) {
16         iterator z = next(y);
17         if (y == begin()) {
18             if (z == end()) return 0;
19             return y->m == z->m && y->b >= z->b;
20         }
21         iterator x = prev(y);
22         if (z == end()) return y->m == x->m && y->b >= x->b;
23         return (x->m-z->m)*(z->b-y->b) >= (z->b-x->b)*(y->m-z->m);
24     }
25     iterator next(iterator y) {return ++y;}
26     iterator prev(iterator y) {return --y;}
27     void insert_line(ll m, ll b) {
28         iterator y = insert((Line) {m, b});
29         y->it = y;
30         if (bad(y)) {erase(y); return;}
31         while (next(y) != end() && bad(next(y))) erase(next(y));
32         while (y != begin() && bad(prev(y))) erase(prev(y));
33     }
34     ll eval(ll x) {
35         Line l = *lower_bound((Line) {x, is_query});
36         return l.m * x + l.b;
37     }
38 } h;
39 const Line *Line::succ(multiset<Line>::iterator it) const {
40     return (++it==h.end() ? NULL : &*it); }
```

# 5 Strings

## 5.1 Hashing

```
1  struct StrHash { // Hash polinomial con exponentes decrecientes.
2      static constexpr ll ms[] = {1'000'000'007, 1'000'000'403};
3      static constexpr ll b = 500'000'000;
4      vector<ll> hs[2], bs[2];
5      StrHash(string const& s) {
6          int n = sz(s);
7          forn(k, 2) {
8              hs[k].resize(n+1), bs[k].resize(n+1, 1);
9              forn(i, n) {
10                 hs[k][i+1] = (hs[k][i] * b + s[i]) % ms[k];
11                 bs[k][i+1] = bs[k][i] * b      % ms[k];
12             }
```

7

```
13              }
14          }
15      ll get(int idx, int len) const { // Hashes en ‘s[idx,
            idx+len)‘.
16          ll h[2];
17          forn(k, 2) {
18              h[k] = hs[k][idx+len] - hs[k][idx] * bs[k][len] % ms[k];
19              if (h[k] < 0) h[k] += ms[k];
20          }
21          return (h[0] << 32) | h[1];
22      }
23  };
```

## 5.2   Suffix Array

```
1   #define RB(x) ((x) < n ? r[x] : 0)
2   void csort(vector<int>& sa, vector<int>& r, int k) {
3       int n = sz(sa);
4       vector<int> f(max(255, n)), t(n);
5       forn(i, n) ++f[RB(i+k)];
6       int sum = 0;
7       forn(i, max(255, n)) f[i] = (sum += f[i]) - f[i];
8       forn(i, n) t[f[RB(sa[i]+k)]++] = sa[i];
9       sa = t;
10  }
11  vector<int> compute_sa(string& s){ // O(n*log2(n))
12      int n = sz(s) + 1, rank;
13      vector<int> sa(n), r(n), t(n);
14      iota(all(sa), 0);
15      forn(i, n) r[i] = s[i];
16      for (int k = 1; k < n; k *= 2) {
17          csort(sa, r, k), csort(sa, r, 0);
18          t[sa[0]] = rank = 0;
19          forr(i, 1, n) {
20              if(r[sa[i]] != r[sa[i-1]] || RB(sa[i]+k) !=
                    RB(sa[i-1]+k)) ++rank;
21              t[sa[i]] = rank;
22          }
23          r = t;
24          if (r[sa[n-1]] == n-1) break;
```

```
25      }
26      return sa; // sa[i] = i-th suffix of s in lexicographical order
27  }
28  vector<int> compute_lcp(string& s, vector<int>& sa){
29      int n = sz(s) + 1, L = 0;
30      vector<int> lcp(n), plcp(n), phi(n);
31      phi[sa[0]] = -1;
32      forr(i, 1, n) phi[sa[i]] = sa[i-1];
33      forn(i,n) {
34          if (phi[i] < 0) { plcp[i] = 0; continue; }
35          while(s[i+L] == s[phi[i]+L]) ++L;
36          plcp[i] = L;
37          L = max(L - 1, 0);
38      }
39      forn(i, n) lcp[i] = plcp[sa[i]];
40      return lcp; // lcp[i] = longest common prefix between sa[i-1]
            and sa[i]
41  }
```

## 5.3   Kmp

```
1   template<class Char=char>struct Kmp {
2       using str = basic_string<Char>;
3       vector<int> pi; str pat;
4       Kmp(str const& _pat): pi(move(pfun(_pat))), pat(_pat) {}
5       vector<int> matches(str const& txt) const {
6           if (sz(pat) > sz(txt)) {return {};}
7           vector<int> occs; int m = sz(pat), n = sz(txt);
8           if (m == 0) {occs.push_back(0);}
9           int j = 0;
10          forn(i, n) {
11              while (j != 0 && txt[i] != pat[j]) {j = pi[j-1];}
12              if (txt[i] == pat[j]) {++j;}
13              if (j == m) {occs.push_back(i - j + 1);}
14          }
15          return occs;
16      }
17  };
```

## 5.4   Manacher

```
1  struct Manacher {
2      vector<int> p;
3      Manacher(string const& s) {
4          int n = sz(s), m = 2*n+1, l = -1, r = 1;
5          vector<char> t(m); forn(i, n) t[2*i+1] = s[i];
6          p.resize(m); forr(i, 1, m) {
7              if (i < r) p[i] = min(r-i, p[l+r-i]);
8              while (p[i] <= i && i < m-p[i] && t[i-p[i]] ==
                     t[i+p[i]]) ++p[i];
9              if (i+p[i] > r) l = i-p[i], r = i+p[i];
10         }
11     } // Retorna palindromos de la forma {comienzo, largo}.
12     pii at(int i) const {int k = p[i]-1; return pair{i/2-k/2, k};}
13     pii odd(int i) const {return at(2*i+1);} // Mayor centrado en
           s[i].
14     pii even(int i) const {return at(2*i);} // Mayor centrado en
           s[i-1,i].
15 };
```

### 5.5 String Functions

```
1  template<class Char=char>vector<int> pfun(basic_string<Char>const&
       w) {
2      int n = sz(w), j = 0; vector<int> pi(n);
3      forr(i, 1, n) {
4          while (j != 0 && w[i] != w[j]) {j = pi[j - 1];}
5          if (w[i] == w[j]) {++j;}
6          pi[i] = j;
7      } // pi[i] = lengh of longest proper suffix of w[0..i] that is
           also prefix
8      return pi;
9  }
10 template<class Char=char>vector<int> zfun(const
       basic_string<Char>& w) {
11     int n = sz(w), l = 0, r = 0; vector<int> z(n);
12     forr(i, 1, n) {
13         if (i <= r) {z[i] = min(r - i + 1, z[i - l]);}
14         while (i + z[i] < n && w[z[i]] == w[i + z[i]]) {++z[i];}
15         if (i + z[i] - 1 > r) {l = i, r = i + z[i] - 1;}
16     } // z[i] = lengh of longest prefix of w that also begins at
           index i
17     return z;
18 }
```

## 6 Flujo

### 6.1 Dinic

```
1  struct Dinic{
2      int nodes,src,dst;
3      vector<int> dist,q,work;
4      struct edge {int to,rev;ll f,cap;};
5      vector<vector<edge>> g;
6      Dinic(int x):nodes(x),g(x),dist(x),q(x),work(x){}
7      void add_edge(int s, int t, ll cap){
8          g[s].pb((edge){t,sz(g[t]),0,cap});
9          g[t].pb((edge){s,sz(g[s])-1,0,0});
10     }
11     bool dinic_bfs(){
12         fill(all(dist),-1);dist[src]=0;
13         int qt=0;q[qt++]=src;
14         for(int qh=0;qh<qt;qh++){
15             int u=q[qh];
16             forn(i,sz(g[u])){
17                 edge &e=g[u][i];int v=g[u][i].to;
18                 if(dist[v]<0&&e.f<e.cap)dist[v]=dist[u]+1,q[qt++]=v;
19             }
20         }
21         return dist[dst]>=0;
22     }
23     ll dinic_dfs(int u, ll f){
24         if(u==dst)return f;
25         for(int &i=work[u];i<sz(g[u]);i++){
26             edge &e=g[u][i];
27             if(e.cap<=e.f)continue;
28             int v=e.to;
29             if(dist[v]==dist[u]+1){
30                 ll df=dinic_dfs(v,min(f,e.cap-e.f));
31                 if(df>0){e.f+=df;g[v][e.rev].f-=df;return df;}
32             }
```

```
33          }
34          return 0;
35      }
36      ll max_flow(int _src, int _dst){
37          src=_src;dst=_dst;
38          ll result=0;
39          while(dinic_bfs()){
40              fill(all(work),0);
41              while(ll delta=dinic_dfs(src,INF))result+=delta;
42          }
43          return result;
44      }
45  };
```

## 6.2 Kuhn

```
1  vector<int> g[MAXN];
2  vector<bool> vis;
3  vector<int> match;
4
5  bool kuhn_dfs(int u){
6      if (vis[u]) return false;
7      vis[u] = true;
8      for (int v : g[u]) if (match[v] == -1 || kuhn_dfs(match[v])) {
9          match[v] = u;
10         return true;
11     } return false;
12 }
13
14 vector<int> kuhn(int n) {
15     match.resize(n, -1);
16     vis.resize(n);
17     forn(u, n) {
18         vis.assign(n, false);
19         kuhn_dfs(u);
20     }
21     return match;
22 } //n: cant de nodos  devuelve un vector con -1 si no matchea y
       sino su match
```

# 7   Optimización

## 7.1   Ternary Search

```
1  // mnimo  entero de f en (l,r)
2  ll ternary(auto f, ll l, ll r) {
3      for (ll d = r-l; d > 2; d = r-l) {
4          ll a = l+d/3, b = r-d/3;
5          if (f(a) > f(b)) l = a; else r = b;
6      }
7      return l+1; // retorna un punto, no un resultado de evaluar f
8  }
9
10 // mnimo  real de f en (l,r)
11 // para error < EPS, usar iters = log((r-l)/EPS)/log(1.618)
12 double golden(auto f, double l, double r, int iters) {
13     constexpr double ratio = (3-sqrt(5))/2;
14     double x1 = l+(r-l)*ratio, f1 = f(x1);
15     double x2 = r-(r-l)*ratio, f2 = f(x2);
16     while (iters--) {
17         if (f1 > f2) l=x1, x1=x2, f1=f2, x2=r-(r-l)*ratio, f2=f(x2);
18         else        r=x2, x2=x1, f2=f1, x1=l+(r-l)*ratio, f1=f(x1);
19     }
20     return (l+r)/2; // retorna un punto, no un resultado de
                evaluar f
21 }
```

## 7.2   Longest Increasing Subsequence

```
1  int lis(vector<int> const& a) {
2      int n = a.size();
3      const int INF = 1e9;
4      vector<int> d(n+1, INF);
5      d[0] = -INF;
6      forn(i,n){
7          int l = upper_bound(all(d), a[i]) - d.begin();
8          if (d[l-1] < a[i] && a[i] < d[l])
9              d[l] = a[i];
10     }
11     int ans = 0;
```

```
12      for (int l = 0; l <= n; l++) {
13          if (d[l] < INF)
14              ans = l;
15      }
16      return ans;
17 }
```

# 8   Otros

## 8.1   Fijar el numero de decimales

```
1 // antes de imprimir decimales, con una sola vez basta
2 cout << fixed << setprecision(DECIMAL_DIG);
```

## 8.2   Hash Table (Unordered Map/ Unordered Set)

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 template<class Key,class Val=null_type>using
      htable=gp_hash_table<Key,Val>;
4 // como unordered_map (o unordered_set si Val es vacio), pero sin
      metodo count
```

## 8.3   Indexed Set

```
1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 template<class Key, class Val=null_type>
4 using indexed_set = tree<Key, Val, less<Key>, rb_tree_tag,
5                          tree_order_statistics_node_update>;
6 // indexed_set<char> s;
7 // char val = *s.find_by_order(0); // acceso por indice
8 // int idx = s.order_of_key('a'); // busca indice del valor
```