ESTRUTURAS DE DADOS

Union Find

```
struct UnionFind {
    int *rank, *parent, size;
    UnionFind(int msize) { size = msize; rank = new int[size]; parent = new int[size]; }
    ~UnionFind() { delete[] rank; delete[] parent; }
    void clear (int msize=-1) {
        if (msize >= 0) size = msize;
        for (int i = 0; i < size; i++)
            parent[i] = i, rank[i] = 1;
    }
    int find (int node) {
        if (node == parent[node]) return node;
        return parent[node] = find(parent[node]);
    }
    void union_ (int a, int b) {
        a = find(a), b = find(b);
        if (rank[a] <= rank[b])
            parent[a] = b, rank[b] += rank[a];
        else
            parent[b] = a, rank[a] += rank[b];
    }
}; // int main() { UnionFind uf(MAXN); uf.clear(n); }
```

Segment Tree

```
#define st_left(idx) (2*(idx)+1)
#define st_right(idx) (2*(idx)+2)
#define st_middle(left,right) (((left)+(right))/2)
template<class T> // class T must implement operator+
class segtree {
    void from_array (T* v, int idx, int left, int right) {
        if (left != right) {
            from_array(v, st_left(idx), left, st_middle(left,right));
            from_array(v, st_right(idx), st_middle(left,right)+1, right);
            tree[idx] = tree[st_left(idx)] + tree[st_right(idx)];
        } else
            tree[idx] = v[left]; // para clear(), mudar v[left] para 0
    }
    T read (int i, int j, int idx, int left, int right) {
        if (i <= left && right <= j) return tree[idx];
        if (j < left || right < i) return 0;
        return read(i, j, st_left(idx), left, st_middle(left,right)) + read(i, j,
st_right(idx), st_middle(left,right)+1, right);
    }
    void set (int x, T& v, int idx, int left, int right) {
        if (x < left || right < x) return;
        if (left != right) {
            set(x, v, st_left(idx), left, st_middle(left,right));
            set(x, v, st_right(idx), st_middle(left,right)+1, right);
            tree[idx] = tree[st_left(idx)] + tree[st_right(idx)];
        } else
            tree[idx] = v;
    }
public:
    T* tree; int size;
```

```
    segtree() {} ~segtree() { delete[] tree; }
    segtree(int size) { this->size = size; tree = new T[4*size]; }
    inline void from_array(T array[]) { from_array(array, 0, 0, size-1); }
    inline T read(int i, int j) { return read(i, j, 0, 0, size-1); }
    inline void set(int x, T v) { set(x, v, 0, 0, size-1); }
}; // int main () { segtree<T> tree(MAXN); tree.size = N; }
```

Lazy Propagation

```
// REFAZER!! (SEM STRUCT!)
#define st_left(idx) (2*(idx)+1)
#define st_right(idx) (2*(idx)+2)
#define st_middle(left,right) (((left)+(right))/2)
template<class T, class Q> // implement +(T,T)
class lsegtree {
    void from_array (T* v, int idx, int left, int right) {
        refreshr[idx] = false;
        if (left != right) {
            from_array(v, st_left(idx), left, st_middle(left,right));
            from_array(v, st_right(idx), st_middle(left,right)+1, right);
            tree[idx] = tree[st_left(idx)] + tree[st_right(idx)];
        } else
            tree[idx] = v[left];
    }
    T read (int i, int j, int idx, int left, int right) {
        update(this, idx, left, right);
        if (i <= left && right <= j) return tree[idx];
        if (j < left || right < i) return nil;
        return read(i, j, st_left(idx), left, st_middle(left,right)) + read(i, j,
st_right(idx), st_middle(left,right)+1, right);
    }
    void set (int i, int j, Q v, int idx, int left, int right) {
        update(this, idx, left, right);
        if (j < left || right < i) return;
        if (i <= left && right <= j) {
            refresh[idx] = v;
            refreshr[idx] = true;
            update(this, idx, left, right);
        } else {
            set(i, j, v, st_left(idx), left, st_middle(left,right));
            set(i, j, v, st_right(idx), st_middle(left,right)+1, right);
            tree[idx] = tree[st_left(idx)] + tree[st_right(idx)];
        }
    }
public:
    T *tree; Q *refresh, nil; bool *refreshr; int size;
    void (*update)(lsegtree<T,Q>*,int,int,int);
    lsegtree(int size, T nil, void (*f)(lsegtree<T,Q>*,int,int,int)) {
        this->size = size;
        this->nil = nil;
        tree = new T[4*size];
        refresh = new Q[4*size];
        refreshr = new bool[4*size];
        update = f;
    }
    lsegtree() {}
    ~lsegtree() { delete[] tree; delete[] refresh; delete[] refreshr; }
    inline void from_array(T array[]) { from_array(array, 0, 0, size-1); }
    inline T read(int i, int j) { return read(min(i,j), max(i,j), 0, 0, size-1); }
    inline void set(int i, int j, Q v) { set(min(i,j), max(i,j), v, 0, 0, size-1); }
```

```
};
void segt_update (lsegtree<int,int>* tree, int idx, int left, int right) {
    if (tree->refreshr[idx]) {
        if (left != right) {
            if (!tree->refreshr[st_left(idx)]) tree->refresh[st_left(idx)] = 0;
            if (!tree->refreshr[st_right(idx)]) tree->refresh[st_right(idx)] = 0;
            tree->refresh[st_left(idx)] += tree->refresh[idx]; //edit here
            tree->refresh[st_right(idx)] += tree->refresh[idx]; //edit here
            tree->refreshr[st_left(idx)] = tree->refreshr[st_right(idx)] = true;
        }
        tree->tree[idx] += (right-left+1)*tree->refresh[idx]; //edit here
        tree->refreshr[idx] = false;
    }
} // int main (){ lsegtree<int> t(MAXN, 0, segt_update); t.size = N;}
```

Hashstring

```
#define HBASE 101
#define HMAXLEN 10000000
#define hstring_new() hstring(new char[HMAXLEN], new hash_t[HMAXLEN])
typedef unsigned long long hash_t;
struct hstring { // CAREFUL! BASE 2^64!
    static hash_t base[]; int interval_pos, length; hash_t *h; char *s;
    inline hash_t valueOf(char c) { return c; }
    hstring():h(0),s(0),length(0),interval_pos(0) {} // CAREFUL! DOESN'T INIT.!
    hstring(char *s_, hash_t *h_, int length_=0, int interval_pos_=0) {
        if (!hstring::base[0]) {
            hstring::base[0] = 1;
            for (int i = 1; i <= HMAXLEN; i++)
                hstring::base[i] = hstring::base[i-1] * HBASE;
        }
        s = s_, h = h_, length = length_, interval_pos = interval_pos_;
    }
    void refresh() { length = strlen(s); refreshHash(); }
    void refreshHash(int pos=0) {
        if (!pos && length)
            h[0] = (interval_pos>0 ? h[-1]*HBASE : 0) + valueOf(s[0]);
        for (int i = pos ? pos : 1; i < length; i++)
            h[i] = h[i-1]*HBASE + valueOf(s[i]);
    }
    inline hash_t hash(int pos=0, int length=-1) const {
        if (length < 0) length = this->length-pos;
        return h[pos+length-1] - (pos||interval_pos>0 ? h[pos-1]*hstring::base[length] :
0);
    }
    inline hstring interval(int pos=0, int length=-1) {
        if (length < 0) length = this->length-pos;
        return hstring(s+pos, h+pos, length, interval_pos+pos);
    }
    bool operator==(const hstring& hs) const { return length == hs.length && hash() ==
hs.hash(); }
    inline int lcp(hstring& cmp) const {
        int start = 1, end = min(length, cmp.length), max_length = 0;
        if (end <= 0) return 0;
        while (start <= end) {
            int mid = (start+end)>>1;
            if (hash(0, mid) == cmp.hash(0, mid))
                start = (max_length=mid)+1;
            else
                end = mid-1;
```

```
        }
        return max_length;
    }
    bool operator<(const hstring& hs) const {
        int lcp = this->lcp(const_cast<hstring&>(hs));
        return (lcp==length && lcp<hs.length) || (lcp==hs.length && lcp<length ? false :
s[lcp]<hs.s[lcp]);
    } // suffix_array: (0 <= i < length)v.push_back(interval(i)), sort(v)
}; hash_t hstring::base[HMAXLEN+1] = {0};
// int main() { hstring hs = hstring_new(); scanf("%s",hs.s); hs.refresh(); }
```

Bigint

```
#include <sstream>
const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 2048;
const double EPS = 1e-10;
inline int cmp (double x, double y = 0, double tol = EPS) {
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}
struct  bigint {
    int v[TAM], n;
    bigint(int x = 0): n(1) { memset(v, 0, sizeof(v)); v[n++] = x; fix(); }
    bigint(char  *s):  n(1) {
        memset(v, 0, sizeof(v));
        int sign = 1;
        while (*s && !isdigit(*s))
            if (*s++ == '-')
                sign *= -1;
        char *t = strdup(s), *p = t + strlen(t);
        while (p > t) {
            *p = 0;
            p = max(t, p - DIG);
            sscanf (p, "%d", &v[n]);
            v[n++] *= sign;
        }
        free(t), fix();
    }
    bigint& fix(int m=0) {
        n = max(m, n);
        int sign = 0;
        for (int i=1, e=0; i <= n || e && (n=i); i++) {
            v[i]  +=  e;
            e = v[i] / BASE;
            v[i] %= BASE;
            if (v[i])
                sign = (v[i] > 0) ? 1 : -1;
        }
        for (int i = n-1; i > 0; i--)
            if (v[i] * sign < 0)
                v[i] += sign * BASE, v[i+1] -= sign;
        while (n && !v[n]) n--;
        return *this;
    }

    int cmp(const bigint& x=0) const {
        int i = max(n, x.n), t=0;
        while (true)
            if ((t = ::cmp(v[i], x.v[i])) || !i--)
```

```
            return t;
    }
    bool operator <(const bigint& x) const { return cmp(x) < 0; }
    bool operator ==(const bigint& x) const { return cmp(x) == 0; }
    bool operator !=(const bigint& x) const { return cmp(x) != 0; }

    operator string() const {
        ostringstream s;
        s << v[n];
        for (int i = n-1; i>0; i--) {
            s.width(DIG);
            s.fill('0');
            s << abs(v[i]);
        }
        return s.str();
    }
    friend ostream& operator <<(ostream& o, const bigint& x) { return o << (string) x; }

    bigint& operator +=(const bigint& x) {
        for (int i = 1; i <= x.n; i++)
            v[i] += x.v[i];
        return fix(x.n);
    }
    bigint operator +(const bigint& x) { return bigint(*this) += x; }
    bigint& operator -=(const bigint& x) {
        for (int i = 1; i <= x.n; i++)
            v[i] -= x.v[i];
        return fix(x.n);
    }
    bigint operator -(const bigint& x) { return bigint(*this) -= x; }
    bigint operator -() { bigint r = 0; return r -= *this; }

    void ams(const bigint& x, int m, int b) { //  *this  +=  (x  *  m)  <<  b;
        for (int i=1, e=0; (i <= x.n || e) && (n = i + b); i++) {
            v[i+b] += x.v[i] * m + e;
            e = v[i+b] / BASE;
            v[i+b]  %=  BASE;
        }
    }
    bigint operator *(const bigint& x) const {
        bigint  r;
        for (int i = 1; i <= n; i++)
            r.ams(x, v[i], i-1);
        return r;
    }
    bigint& operator *=(const bigint& x) { return *this = *this * x; }
    //  cmp(x  /  y)  ==  cmp(x)  *  cmp(y);  cmp(x  %  y)  ==  cmp(x);
    bigint div(const bigint& x) {
        if (x == 0) return 0;
        bigint q;
        q.n  =  max(n  -  x.n  +  1,  0);
        int d = x.v[x.n] * BASE + x.v[x.n-1];
        for (int i = q.n; i > 0; i--) {
            int  j  =  x.n  +  i  -  1;
            q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
            ams(x, -q.v[i], i-1);
            if (i == 1 || j == 1)
                break;
            v[j-1] += BASE * v[j];
            v[j]  =  0;
        }
```

```
        fix(x.n);
        return q.fix();
    }
    bigint& operator /=(const bigint& x) { return *this = div(x); }
    bigint& operator %=(const bigint& x) { div(x); return *this; }
    bigint operator /(const bigint& x) { return bigint(*this).div(x); }
    bigint operator %(const bigint& x) { return bigint(*this) %= x; }

    bigint pow(int x) {
        if (x < 0)
            return (*this == 1 || *this == -1) ? pow(-x) : 0;
        bigint r = 1;
        for (int i = 0; i < x; i++)
            r *= *this;
        return r;
    }
    bigint root(int  x) {
        if (cmp() == 0 || cmp() < 0 && x % 2 == 0)
            return 0;
        if (*this == 1 || x == 1)
            return *this;
        if (cmp() < 0)
            return -(-*this).root(x);
        bigint a = 1, d = *this;
        while (d  !=  1) {
            bigint b = a + (d /= 2);
            if (cmp(b.pow(x)) >= 0) {
                d += 1;
                a = b;
            }
        }
        return  a;
    }
};
```

ALGORITMOS

Josephus

$O(n) \mid O(k \lg n)$

```
def josephus(n, k): # 1..n
    r, i = 0, 2
    while i <= n:
        r, i = (r + k) % i, i + 1
    return r + 1

def josephus(n, k): # 1..n
    if n == 1: return 1
    return ((josephus(n - 1, k) + k - 1) % n) + 1

def josephus(n,k): # 0..n-1
    if n == 1: return 0
    if k == 1: return n-1
    if k > n: return (josephus(n - 1, k) + k) % n
    r = josephus(n - n/k, k) - n%k
    return r + (n if r < 0 else r/(k-1))

def josephus2(n): # 1..n, k=2
    from math import log
    return 2*(n - 2**(int(log(n,2))))+1
```

ALGORITMOS DE STRING

Z-Algorithm $O(n)$

Monta o array z[], onde z[i] indica o tamanho da maior substring começando em s[i] que é também prefixo de s.

```
// note: MAXN > maxLen(T)+maxLen(S)
int z[MAXN]; // s[:z[i]] == s[i:i+z[i]]
void z_algorithm(string& s) {
    int n = s.length();
    int L = 0, R = 0;
    for (int i = 1; i < n; i++) {
        if (i > R) {
            L = R = i;
            while (R < n && s[R-L] == s[R])
                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-L] == s[R])
                    R++;
                z[i] = R-L;
                R--;
            }
        }
    }
}
// finds the indexes of all occurences of T in S
void indexesOf(string T, string& S, vector<int>& v) {
    int m = T.length();
    T += "$" + S;
    z_algorithm(T);
    for (int i = m+1; i < T.length(); i++)
        if (z[i] == m)
            v.push_back(i-m-1);
}
```

Knuth-Morris-Pratt (KMP) $O(n+m)$

```
void failureFunction(char P[], int F[], int m /*strlen(P)*/) {
    int i=1, j=0;
    F[0] = 0;
    while (i < m)
        if(P[i] == P[j])
            F[i++] = ++j;
        else if (j > 0)
            j = F[j-1];
        else
            F[i++] = 0;
}
#define MAXP 10000 // maxLen(P)
int kmp(char T[] /*Text*/, char P[] /*Pattern*/) {
    static int F[MAXP];
```

```
    int i=0, j=0;
    int m = strlen(P);
    int n = strlen(T);
    failureFunction(P, F, m);
    while (i < n) {
        if (T[i] == P[j]) {
            if (j == m-1)
                return i-j;
            else
                i++, j++;
        }
        else if (j > 0)
            j = F[j-1];
        else
            i++;
    }
    return -1;
}
```

Manacher (longest palindrome) $O(n)$

```
// Transform S into T. Example: S = "abba", T = "^#a#b#b#a#$"
// ^ and $ signs are sentinels to avoid bounds checking
string preProcess(string s) {
    int n = s.length();
    if (n == 0) return "^$";
    string ret = "^";
    for (int i = 0; i < n; i++)
        ret += "#" + s.substr(i, 1);
    ret += "#$";
    return ret;
}
string longestPalindrome(string s) {
    string T = preProcess(s);
    int n = T.length();
    int *P = new int[n];
    int C = 0, R = 0;
    for (int i = 1; i < n-1; i++) {
        int i_mirror = (C<<1)-i; // i' = C-(i-C)
        P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;
        // Attempt to expand palindrome centered at i
        while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
            P[i]++;
        // If palindrome centered at i expand past R,
        // adjust center based on expanded palindrome.
        if (i + P[i] > R) {
            C = i;
            R = i + P[i];
        }
    }
    // Find the maximum element in P.
    int maxLen = 0;
    int centerIndex = 0;
    for (int i = 1; i < n-1; i++)
        if (P[i] > maxLen) {
            maxLen = P[i];
            centerIndex = i;
        }
    delete[] P;
    return s.substr((centerIndex - 1 - maxLen)/2, maxLen);
```

```
}
```

Minimum Lex Rotation (Lyndon word) $O(n)$

```
// # of rotations in str to find the lexicographically smaller string
int lexRot(string str) {
    int n = str.size();
    str += str;
    int ini, fim, rot;
    rot = ini = 0;
    fim = 1;
    while(fim < n && rot+ini+1 < n) {
        if (str[ini+rot] == str[ini+fim])
            ini++;
        else if (str[ini+rot] < str[ini+fim]) {
            fim += ini+1;
            ini = 0;
        } else {
            rot = max(rot+ini+1, fim);
            fim = rot+1;
            ini = 0;
        }
    }
    return rot;
}
```

Suffix Array $O(n \lg n)$

```
int sa[MAXN], invsa[MAXN], n, sz; char s[MAXN];
inline bool cmp(int a, int b) { return invsa[a+sz] < invsa[b+sz]; }
void sort_sa(int a, int b) {
    if (a == b) return;
    int pivot = sa[a + rand()%(b-a)], c = a, d = b;
    for (int i = c; i < b; i++) if (cmp(sa[i], pivot)) swap(sa[i], sa[c++]);
    for (int i = d-1; i >= a; i--) if (cmp(pivot, sa[i])) swap(sa[i], sa[--d]);
    sort_sa(a, c);
    for (int i = c; i < d; i++) invsa[sa[i]] = d-1;
    if (d-c == 1) sa[c] = -1;
    sort_sa(d, b);
}
void suffix_array(char* s) {
    n = strlen(s), invsa[n] = -1;
    for (int i = 0; i < n; i++) sa[i] = i, invsa[i] = s[i];
    sz = 0; sort_sa(0, n);
    for (sz = 1; sz < n; sz *= 2)
        for (int i = 0, j = i; i < n; i = j)
            if (sa[i] < 0) {
                while (sa[j] < 0) j += (-sa[j]);
                sa[i] = -(j-i);
            } else sort_sa(i, j=invsa[sa[i]]+1);
    for (int i = 0; i < n; i++) sa[invsa[i]] = i;
}
int main() { scanf(" %s", s); suffix_array(s); for (int i = 0; i < n; i++)
printf("%d\n", sa[i]); }
int lcp[MAXN];
void calc_lcp() {
    for (int i = 0, l = 0; i < n; i++, l = max(0, l-1)) {
        if (invsa[i] == 0) continue;
        int j = sa[invsa[i]-1];
```

```
        while (max(i+l, j+l) < n && s[i+l] == s[j+l]) l++;
        lcp[invsa[i]] = l;
    }//for(int i=0;i+1<n;i++)lcp[i]=lcp[i+1];lcp[n-1]=0;
}
```

ALGORITMOS DE GRAFOS

Tarjan (strongly connected components) $O(V+E)$

```
int idx[MAXN], idx_count, scc[MAXN], scc_count, sk[MAXN], sk_size;
bool stacked[MAXN], vis[MAXN];
void tarjan(int v) {
    int idxv;
    idx[v] = idxv = ++idx_count;
    sk[sk_size++] = v, stacked[v] = true;
    for (int i = 0; i < G[v].size(); i++) {
        int u = G[v][i];
        if (!vis[u]) {
            vis[u] = true;
            tarjan(u);
        }
        if (stacked[u])
            idx[v] = min(idx[v], idx[u]);
    }
    if (idx[v] == idxv) {
        int u;
        scc_count++;
        do {
            u = sk[--sk_size];
            stacked[u] = false;
            scc[u] = scc_count;
        }
        while (u != v);
    }
}
void find_scc(int N, int st=0) {
    for (int i = st; i < N; i++)
        stacked[i] = vis[i] = scc[i] = 0;
    idx_count = scc_count = sk_size = 0;
    for (int i = st; i < N; i++)
        if (!vis[i])
            tarjan(i);
}
```

2-Sat $O(V+E)$

```
#define NOT(v) ((v)^1)
//_2sat_edge(v_not ? NOT(v) : v, u_not ? NOT(u) : u);
inline bool _2sat_edge(int v, int u) {
    G[NOT(v)].push_back(u);
    G[NOT(u)].push_back(v);
}
bool _2sat(int N, int st=0) {
    find_scc(N, st);
    for (int i = st; i < N; i += 2)
        if (scc[i] == scc[NOT(i)])
            return false;
    return true;
```

```
}
```

Vértices de articulação $O(V + E)$

```
int n, time_s, visit[MAX];
vector<int> ADJ[MAX];
int dfs(int u, set<int>& ans){
    int menor = visit[u] = time_s++;
    int filhos = 0;
    for(int i = 0; i<ADJ[u].size(); i++){
       if(visit[ADJ[u][i]]==0){
          filhos++;
          int m = dfs(ADJ[u][i], ans);
          menor = min(menor,m);
          if(visit[u]<=m && (u!=0 || filhos>=2)){
              ans.insert(u);
          }
       }else{
          menor = min(menor, visit[ADJ[u][i]]);
       }
    }
    return menor;
}
set<int> get_articulacoes(){
    set<int> ans;
    time_s = 1;
    memset(visit, 0, n*sizeof(int));
    dfs(0,ans);
    return ans;
}
```

Tree-distance-sum $O(V + E)$

```
int distsum, n;
int dfs(int v, int p=-1, int w=0) {
    int k = 1;
    for (int i = 0; i < G[v].size(); i++) {
        int u = G[v][i].first, w = G[v][i].second;
        if (u != p) k += dfs(u, v, w);
    }
    distsum += w*(n-k)*k;
    return k;
}
```

Bellman-Ford $O(VE)$

```
#define MAXN 1000
#define $w first
#define $u second.first
#define $v second.second
vector< pair< int, pair<int, int> > > edges; // (w, (u,v))
int dist[MAXN], n;
int bellman_ford(int start) {
    for (int i = 0; i < n; i++)
        dist[i] = INF;
    dist[start] = 0;
    for (int i = 0; i < n; i++) {
        bool edit = false;
```

```
        for (int j = 0; j < m; j++)
            if (dist[edges[j].$u] + edges[j].$w < dist[edges[j].$v]) {
                dist[edges[j].$v] = dist[edges[j].$u] + edges[j].$w;
                edit = true;
            }
        if (!edit) break;
        if (i+1 == n) return true;
    }
    return false;
}
```

Dinic (max-flow) $O(V^2E)$

```
const int MAXN = 5005, MAXE = 30005;
typedef long long lint;
struct Graph {
    int n, m;
    vector<int> adj[MAXN];
    pair<int, int> edges[2*MAXE];
    // usar add_edge(v,u,c);add_edge(u,v,d); (mesmo que d=0!!)
    inline void add_edge(int v, int u, int c) {
        edges[m] = make_pair(u, c);
        adj[v].push_back(m++);
    }
};
struct DinicGraph : Graph {
    int dis[MAXN], pos[MAXN];
    int fluxo[2*MAXE];
    int src, dst;
} graph;

bool dinic_bfs(DinicGraph& g) {
    queue<int> qu;
    qu.push(g.src);

    for (int i = 0; i < g.n; i++) g.dis[i] = -1;
    g.dis[g.src] = 0;

    while (!qu.empty()) {
        int v = qu.front(); qu.pop();
        for (int i = 0; i < g.adj[v].size(); i++) {
            int e = g.adj[v][i];
            int u = g.edges[e].first;
            int c = g.edges[e].second;
            if (c > 0 && g.dis[u] == -1) {
                g.dis[u] = g.dis[v] + 1;
                qu.push(u);
            }
        }
    }
    return g.dis[g.dst] != -1;
}
int dinic_dfs(int v, int flow, DinicGraph& g) {
    if (v == g.dst) return flow;

    for (int& i = g.pos[v]; i < g.adj[v].size(); i++) {
        int e = g.adj[v][i];
        int u = g.edges[e].first;
        int c = g.edges[e].second;
        if (c > 0 && g.dis[u] == g.dis[v] + 1) {
```

```
            int flow_ = dinic_dfs(u, min(flow, c), g);
            if (flow_ > 0) {
                g.edges[e].second -= flow_;
                g.edges[e^1].second += flow_;
                return flow_;
            }
        }
    }
    return 0;
}
lint dinic(DinicGraph& g) {
    lint max_flow = 0;
    while (dinic_bfs(g)) {
        for (int i = 0; i < g.n; i++) g.pos[i] = 0;
        while (int flow = dinic_dfs(g.src, INT_MAX, g))
            max_flow += flow;
    }
    return max_flow;
}
```

Min-cut ST $O(V + E)$

```
// REFAZER! (OU DESCARTAR!)
vector< pair<int, int> > mincut(int n, int s) { //numVertices, S
    vector< pair<int, int> > mct;
    static int q[MAXN], vis[MAXN];
    int qi=0, qf=0;
    for (int i=0; i<MAXN; i++) vis[i]=0;
    q[qf++] = s;
    while (qf > qi) {
        int a = q[qi++];
        for (int i=0; i<deg[a]; i++) {
            int b = adj[a][i];
            if (!vis[b] && cap[a][b])
                vis[b] = 1, q[qf++] = b;
        }
    }
    for (int i=0; i<n; i++)
        for (int j=0; j<deg[i]; j++) {
            int k = adj[i][j];
            if (i < k && vis[i] != vis[k])
                mct.push_back(make_pair(i, k));
        }
    return mct;
}
```

Stoer-Wagner (graph min-cut) $O(V^3)$

```
#define MAXN 200
#define INF (1<<30)
int weight[MAXN][MAXN], isOnSet[MAXN];
int minimumCutPhase(int n, int v_initial, int lenV) {
    int visited[MAXN], sumWeight[MAXN];
    int penuV = v_initial, lastV = v_initial, minimumCut = 0;
    int cur = v_initial;
    for (int i = 0; i < n; i++)
        visited[i] = sumWeight[i] = 0;
    for (int i = 0; i < lenV; i++) {
        int next_v = -1, sum = -1;
        visited[cur] = 1;
```

```
        penuV = lastV;
        lastV = cur;
        for (int j = 0; j < n; j++) {
            if (visited[j] || !isOnSet[j]) continue;
            sumWeight[j] += weight[j][cur];
            if (sumWeight[j] > sum)
                sum = sumWeight[j], next_v = j;
        }
        cur = next_v;
    }
    for (int i = 0; i < n; i++) {
        if (penuV == i || lastV == i || !isOnSet[i]) continue;
        weight[i][penuV] = weight[penuV][i] = weight[i][penuV] + weight[i][lastV];
    }
    isOnSet[lastV] = 0;
    return sumWeight[lastV];
}
int stoer_wagner(int n) { // n = numOfVertices
    int minimumCut = INF, v_initial = 0;
    for (int i = 0; i < n ; i++)
        isOnSet[i] = 1;
    for (int i = 0; n-i > 1; i++) {
        int answer = minimumCutPhase(n, v_initial, n-i);
        if (answer < minimumCut) minimumCut = answer;
    }
    return minimumCut;
}
#define addedge(a,b,c) weight[a][b] = weight[b][a] = (c)
#define clear() memset(weight, 0, sizeof(weight))
```

SSP with Dijkstra (max-flow min-cost) $O(V^3)$

```
#define MAXN 210
#define INF (1<<28)

int n_elements, cap[MAXN][MAXN], cost[MAXN][MAXN], r_cost[MAXN][MAXN];
int vis[MAXN], pai[MAXN], dist[MAXN];
int dijkstra() {
    for (int i = 0; i < n_elements;  i++)
        vis[i] = 0, pai[i] = -1, dist[i] = INF;
    dist[0] = 0;
    int cur = 0, last_cur = -1;
    while (1) {
        if (last_cur == cur) break;
        vis[cur] = 1;
        int min_dist = INF, new_cur = cur;
        for (int i = 0 ; i < n_elements ; i++) {
            if (vis[i] == 1) continue;
            if (cap[cur][i] > 0 && dist[i] > cost[cur][i] + dist[cur])
                dist[i] = cost[cur][i] + dist[cur],  pai[i] = cur;
            if (dist[i] < min_dist) {
                min_dist = dist[i];
                new_cur = i;
            }
        }
        last_cur = cur;
        cur = new_cur;
    }
    if (pai[n_elements - 1] == -1) return 0;
    return 1;
```

```
}
void update_potentials() {
    for (int i = 0 ; i < n_elements ; i++)
        for (int j = 0 ; j < n_elements ; j++)
            if (dist[i] != INF && dist[j] != INF)
                cost[i][j] += dist[i] - dist[j];
}
// 0 is source, n_elements-1 is dest
pair<int, int> ssp() {
    update_potentials();
    int tot = 0;
    int m_flow = 0;
    while (dijkstra()) {
        int x = n_elements - 1;
        int r_max = INF;
        while (pai[x] != -1) {
            int p = pai[x];
            r_max = min(r_max, cap[p][x]);
            x = p;
        }
        x = n_elements - 1;
        while (pai[x] != -1) {
            int p = pai[x];
            cap[p][x] -= r_max, cap[x][p] += r_max;
            tot += r_cost[p][x]*r_max, x = p;
        }
        m_flow += r_max;
        update_potentials();
    }
    return make_pair(m_flow, tot);
}
// a->b with cost c and capacity cap_
void addedge(int a, int b, int c, int cap_) {
    cap[a][b] = cap_;
    cost[a][b] = c;
    r_cost[a][b] = c;
    cost[b][a] = -c;
    r_cost[b][a] = -c;
}
void ssp_clear(int n) {
    n_elements = n;
    memset(cost, 0, sizeof(cost));
    memset(cap, 0, sizeof(cap));
    memset(r_cost, 0, sizeof(r_cost));
}
```

Hopcroft (bipartite matching) $O(E\sqrt{V})$

```
const int MAXN = 50*50*4;
const int INF = MAXN+5;
const int NIL = MAXN-1;
// [g1: left side | g2: right side] of the bipartite graph
vector<int> g1, g2, adjs[MAXN];
int pair_g1[MAXN], pair_g2[MAXN], dist[MAXN], visited[MAXN];
bool hop_bfs() {
    queue<int> q;
    for (int i = 0; i < g1.size(); i++) {
        int v = g1[i];
        if (pair_g1[v] == NIL) {
            dist[v] = 0;
```

```
            q.push(v);
        } else {
            dist[v] = INF;
        }
    }
    dist[NIL] = INF;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        if (dist[v] < dist[NIL]) {
            for (int i = 0; i < adjs[v].size(); i++) {
                int u = adjs[v][i];
                if (dist[pair_g2[u]] == INF) {
                    dist[pair_g2[u]] = dist[v] + 1;
                    q.push( pair_g2[u] );
                }
            }
        }
    }
    return dist[NIL] != INF;
}
bool hop_dfs(int v) {
    if (v == NIL) return true;
    for (int i = 0; i < adjs[v].size(); i++) {
        int u = adjs[v][i];
        if (dist[pair_g2[u]] == dist[v] + 1) {
            pair_g2[u] = v;
            pair_g1[v] = u;
            return true;
        }
    }
    dist[v] = INF;
    return false;
}
void dfs_bipartite(int v, int color = 1) {
    visited[v] = color;
    if (color == 1) g1.push_back(v);
    else g2.push_back(v);
    int new_color = color == 1 ? 2 : 1;
    for (int i = 0; i < adjs[v].size(); i++) {
        int u = adjs[v][i];
        if (visited[u] == 0) dfs_bipartite(u, new_color);
    }
}
void bipartite(int n_nodes) {
    for (int i = 0; i < n_nodes; i++)
        visited[i] = 0;
    g1.clear(), g2.clear();
    for (int i = 0; i < n_nodes; i++)
        if (!visited[i])
            dfs_bipartite(i);
}
// USAGE: hopcroft(n), returns the number of EDGES
int hopcroft(int n_nodes) {
    bipartite(n_nodes);
    for (int i = 0; i < g1.size(); i++) pair_g1[g1[i]] = NIL;
    for (int i = 0; i < g2.size(); i++) pair_g2[g2[i]] = NIL;
    int matching = 0;
    while (hop_bfs()) {
        for (int i = 0; i < g1.size(); i++) {
            int v = g1[i];
```

```
            if (pair_g1[v] == NIL && hop_dfs(v))
                matching++;
        }
    }
    return matching;
}
void hopcroft_clear(int n_nodes) {
    for(int i = 0; i < n_nodes; i++)
        adjs[i].clear();
}
void addedge(int u, int v) {
    adjs[u].push_back(v);
    adjs[v].push_back(u);
}
```

ALGORITMOS DE MATEMÁTICA

Crivo de Eratóstenes $O(n \lg \lg n)$

```
#define MAXP 1000000
vector<int> primes;
bool is_prime[MAXP+1]={0,0,1};
void crivo(int N=MAXP) {
    for (int i = 3; i <= N; i += 2)
        is_prime[i] = true;
    for (int i = 3; i*i <= N; i += 2)
        if (is_prime[i])
            for (int j = i*i; j <= N; j += 2*i)
                is_prime[j] = false;
    primes.push_back(2);
    for (int i = 3; i <= N; i += 2)
        if (is_prime[i])
            primes.push_back(i);
}
```

Algoritmo de Euclides $O(\lg n)$

```
// ax+by=gcd(a,b)
int egcd(int a, int b, int& x, int& y) {
    if (a == 0) {
        x = 0; y = 1;
        return b;
    }
    int x1, y1;
    int d = egcd(b%a, a, x1, y1);
    x = y1 - (b / a) * x1;
    y = x1;
    return d;
}

int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a%b);
}
#define lcm(a,b) ((a)*(b)/gcd(a,b))
```

Equações de 3º grau $O(\lg n)$

```
#define cprint(n) printf("%.6lf + %.6lfi\n", (n).real(), (n).imag());
const complex<double> cbrt2 = pow(2, 1.0/3);
const complex<double> isqrt3 = complex<double>(0.0,sqrt(3.0));
complex<double> a, b, c, d, alfa, beta, delta, root1, root2, root3;
void calc_dga() { // solves ax^3+bx^2+cx+d=0 (only for a!=0 && (b!=0 || c!=0))
    alfa = (-27.0*a*a*d+9.0*a*b*c-2.0*b*b*b), beta = (3.0*a*c-b*b);
    delta = pow(sqrt(alfa*alfa+4.0*beta*beta*beta) + alfa, 1.0/3);
    root1 = delta/(3.0*cbrt2*a) - cbrt2*beta/(3.0*a*delta) - b/(3.0*a);
    root2 = -1.0/(6.0*cbrt2*a)*(1.0-isqrt3)*delta+
(1.0+isqrt3)*beta/(3.0*cbrt2*cbrt2*a*delta)-b/(3.0*a);
    root3 = -1.0/(6.0*cbrt2*a)*(1.0+isqrt3)*delta+(1.0-
isqrt3)*beta/(3.0*cbrt2*cbrt2*a*delta)-b/(3.0*a);
}
```

Fibonacci

$O(\_builtin\_popcount(n-1) = \lg n)$

```
// Prova (por mim) de que funciona (e como funciona):
// http://www.spoj.com/forum/viewtopic.php?f=41&t=13037
typedef unsigned long long bigint;
#define LOGMAXN 31
bigint f0, f1, faux;
bigint fib2[LOGMAXN+1][2]={{0,1}};
void generate_fib2() { // fib2[i]={Fib(n-1),Fib(n)}, n=2^i
    for (int i = 1; i <= LOGMAXN; i++) {
        fib2[i][1] = fib2[i-1][1]*((fib2[i-1][0]<<1)+fib2[i-1][1]);
        fib2[i][0] = fib2[i][1] - fib2[i-1][0]*((fib2[i-1][1]<<1)-fib2[i-1][0]);
    }
} // se a questão apenas exigir fib(2^n) ou fib(2^n-1), O(1) em fib2[]
inline bigint fib(int n) { // {0,1,1,2,...}
    if (!n--) return 0;
    f0 = 0, f1 = 1;
    while (n) {
        int i = __builtin_ctz(n); //__builtin_ctzll(n)
        faux = f1*fib2[i][1] + f0*fib2[i][0];
        f1 = f1*(fib2[i][0]+fib2[i][1]) + f0*fib2[i][1];
        f0 = faux;
        n -= 1<<i; // 1ULL<<i
    }
    return f1;
}
```

Log

$O(\log_{10} a = |a| = n)$

```
double log10f(char* a) {
    int n = strlen(a); double p = 0;
    for (int i = n-1; i >= 0; i--) // se souber pos_ponto, O(1)
        if (a[i] != '.') p = p/10 + (a[i] - '0'); else n = i;
    return (n-1) + log10(p);
}
```

GEOMETRIA

Área de Polígono

$O(n)$

```
struct polygon {
    int n, p[MAXN][2]; //[x,y]
};
```

```
double area(polygon *p) {
    double total = 0.0;
    int i, j;
    for (i=0; i<p->n; i++) {
        j = (i+1) % p->n;
        total += (p->p[i][0]*p->p[j][1]) - (p->p[j][0]*p->p[i][1]);
    }
    return(total / 2.0);
}
```

Convex Hull $O(n \lg n)$

```
#define EPS 1e-5

inline int cmp(double x, double y = 0, double tol = EPS) {
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

struct point {
    double x, y;
    point(double x = 0, double y = 0): x(x), y(y) {}
    point operator +(point q) { return point(x + q.x, y + q.y); }
    point operator -(point q) { return point(x - q.x, y - q.y); }
    //point operator *(double t) { return point(x * t, y * t); }
    //point operator /(double t) { return point(x / t, y / t); }
    double operator *(point q) { return x * q.x + y * q.y; }
    double operator %(point q) { return x * q.y - y * q.x; }
    int cmp(point q) const {
        if (int t = ::cmp(x, q.x)) return t;
        return ::cmp(y, q.y);
    }
    bool operator ==(point q) const { return cmp(q) == 0; }
    //bool operator !=(point q) const { return cmp(q) != 0; }
    bool operator < (point q) const { return cmp(q) < 0; }

    static point pivot;
}; point point::pivot;

typedef vector<point> polygon;

inline int ccw(point p, point q, point r) {
    return cmp((p - r) % (q - r));
}

bool radial_lt(point p, point q) {
    point P = p - point::pivot, Q = q - point::pivot;
    double R = P % Q;
    if (cmp(R)) return R > 0;
    return cmp(P * P, Q * Q) < 0;
}

polygon convex_hull(polygon& T) {
    int j = 0, k, n = T.size(); polygon U(n);
    point::pivot = *min_element(T.begin(), T.end());
    sort(T.begin(), T.end(), radial_lt);
    for (k = n-2; k >= 0 && ccw(T[0], T[n-1], T[k]) == 0; k--);
    reverse((k+1) + T.begin(), T.end());
    for (int i = 0; i < n; i++) {
        // troque o >= por > para manter pontos colineares
        while (j > 1 && ccw(U[j-1], U[j-2], T[i]) >= 0) j--;
```

```
        U[j++] = T[i];
    }
    U.erase(j + U.begin(), U.end());
    return U;
}
```

Even-odd rule (is point inside polygon) $O(n)$

```
# x, y -- x and y coordinates of point a list of tuples [(x, y), (x, y), ...]
def isPointInPath(x, y, poly):
    num = len(poly)
    i, j, c = 0, num-1, False
    for i in range(num):
        if  ((poly[i][1] > y) != (poly[j][1] > y)):
            if (x < (poly[j][0] - poly[i][0]) * (y - poly[i][1]) / (poly[j][1] - poly[i]
[1]) + poly[i][0]):
                c = not c
        j = i
    return c
```

MATEMÁTICA

Fibonacci

Propriedades

- $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$
- $F_{2n-1} = F_n^2 + F_{n-1}^2$
- $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$
- $\sum_{i=0}^{n} F_i = F_{n+2} - 1$
- $\sum_{i=0}^{n} F_i^2 = F_n F_{n+1}$
- $\sum_{i=0}^{n} F_i^3 = \frac{F_n F_{n+1}^2 - (-1)^n F_{n-1} + 1}{2}$
- $\gcd(F_m, F_n) = F_{\gcd(m,n)}$
- $\sqrt{5N^2 \pm 4} \in \mathbb{N} \leftrightarrow \exists k \in \mathbb{N} \mid F_k = N$

Exponenciação

- $\begin{bmatrix} F_0 & F_1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n = \begin{bmatrix} F_n & F_{n+1} \end{bmatrix}$

Fórmula de Binet

- $\varphi = \frac{1+\sqrt{5}}{2}$
- $F_n = \frac{\varphi^n}{\sqrt{5}}$
- $n(F) = \left\lfloor \log_\varphi \left(F \cdot \sqrt{5} + \frac{1}{2} \right) \right\rfloor$

Areas

Elipse

πab

Interesecção de 3 cilindros

$8(2-\sqrt{2})R^3$

Pentágono regular

$$\frac{5l^2}{4\tan(\frac{\pi}{5})}$$

Hexágono regular

$$\frac{3l^2\sqrt{3}}{2}$$

Cone

Lateral = $2\pi r^2$

Total = $3\pi r^2$

Polígono regular

$$\frac{Perimetro * Apotema}{2}$$

Toro (Rosquinha)

$4\pi^2 Rr = (2\pi r)(2\pi R)$

Volumes

Pirâmide

$$\frac{A_b * h}{3}$$

Cone

$$\frac{\pi r^3\sqrt{3}}{3}$$

Elipsoide

$$\frac{4}{3}\pi abc$$

Toro (Rosquinha)

$$4\pi^2 Rr^2 = (\pi r^2)(2\pi R)$$

Fórmulas de figuras geométricas

Elipse

$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1$$

a = distância do centro ao ponto mais extremo no eixo X

b = distância do centro ao ponto mais extremo no eixo Y

Ponto interno: $\frac{y_p^2}{b^2} - \frac{x_p^2}{a^2} < 1$

Hipérbole

$$\frac{(x-x_0)^2}{a^2} - \frac{(y-y_0)^2}{b^2} = 1$$

Uma reta paralela ao eixo Y separa as duas partes.

Apotema de polígono regular

$$a = \frac{l}{2\tan(\frac{\pi}{n})} = r\cos(\frac{\pi}{n})$$

r = raio do circulo circunscrito

Elipsoide

$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} + \frac{(z-z_0)^2}{c^2} = 1$$

Parábola

$$y = kx^2$$

Ponto interno: $y_p - kx_p^2 < 0$

Heron

Area do triângulo só com os lados.

$$s = \frac{a+b+c}{2}$$

$$\sqrt{s(s-a)(s-b)(s-c)}$$

Fatorial

Para n muito grande, usa-se a aproximação de Stirling

$$n! \tilde{=} \sqrt{2\pi n}(\frac{n}{e})^n$$

Euler's totient function (phi)

$$\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1) = p^k(1-\frac{1}{p})$$

$$\varphi(n) = \varphi(p_1^{k_1})\varphi(p_2^{k_2})...\varphi(p_r^{k_r})$$

BUILTIN FUNCTIONS

Para utilizar em long long, inserir "ll" no fim do nome de cada função.

```
int __builtin_ffs(unsigned int x)
        Retorna o 1-índice do 1-bit de x menos significante.

int __builtin_clz(unsigned int x)
        Retorna o número de 0-bits restantes em x, começando do bit mais significante.

int __builtin_ctz(unsigned int x)
        Retorna o número de 0-bits restantes em x, começando do bit menos significante.

int __builtin_popcount(unsigned int x)
        Retorna o número de 1-bits em x.

int __builtin_parity(unsigned int x)
        Retorna o número de 1-bits em x modulo 2.
```