sudo make a manual

sudo make a sandwich

19 de agosto de 2009

Sumário

1 Utilidades

1.1 .vimrc

```
set nocp bs=2 ts=4 sw=4 noet ru ai si cin hls is ic scs sc nu
syn on
set ww=<,>,b,s,[,]
set mouse=a bg=dark
hi Normal guibg=black guifg=white
set fen fdm=marker
```

1.2 Makefile

```
SRC = $(wildcard *.cpp)
TRG = $(patsubst %.cpp,%,$(SRC))

%: $<.cpp
	g++ -Wall -lm $< -o $@

all: $(TRG)

clean:
	rm $(TRG)
```

1.3 Template

```
#define foreach(i, c) for(typeof(c.begin()) i = c.begin(); i != c.end(); i++)

inline int min(int a, int b) { return a < b ? a : b; }
inline int max(int a, int b) { return a > b ? a : b; }

int main() {
    /* ... */

    return 0;
}
```

2 Estruturas de Dados

2.1 Heap Binário

Tempo: $O(\log n)$ em todas as operações.

```
#define pai(i) (((i)-1)>>1)
#define esq(i) (((i)<<1)+1)
#define dir(i) (((i)<<1)+2)

template <class T> struct heap {
    T h[MAXHEAP];           /* heap (de maximo) */
    int n;                  /* numero de elementos */

    void clear() { n = 0; }

    void sobe(int i) {
```

```
        if(i > 0 && h[i] > h[pai(i)])
            swap(h[pai(i)], h[i]), sobe(pai(i));
    }

    void desce(int i) {
        int fih = dir(i) < n && h[dir(i)] > h[esq(i)] ? dir(i) : esq(i);
        if(fih < n && h[fih] > h[i])
            swap(h[fih], h[i]), desce(fih);
    }

    T pop() {
        T r = h[0];
        h[0] = h[--n];
        desce(0);
        return r;
    }

    void push(T v) {
        h[n] = v;
        sobe(n++);
    }
};
```

2.2 Union-Find

Tempo: $O(\alpha(n))$ em todas as operações.

```
#define MAX 100000
int p[MAX], rank[MAX], len[MAX];

void makeset(int x) {
    p[x] = x;
    len[x] = 1;
    rank[x] = 0;
}

int findset(int x) {
    if(x != p[x])
        p[x] = findset(p[x]);
    return p[x];
}

void linkset(int x, int y) {
    if(rank[x] > rank[y]) {
        p[y] = x;
        len[x] += len[y];
    } else {
        p[x] = y;
        len[y] += len[x];
        if(rank[x] == rank[y])
            rank[y]++;
    }
}

void unionset(int x, int y) {
    linkset(findset(x), findset(y));
}

int sizeset(int x) {
    return len[findset(x)];
```

```
}
```

2.3 Fenwick Tree

Tempo: $O(\log n)$ em todas as operações

```
/* Incrementa val o numero de ocorrencias do valor idx
 * atualizando todos os seus responsaveis <= que maxy
 */
void update(int idx ,int val){
    while (idx <= maxy) tree[idx] += val, idx += (idx & -idx);
}
/* Le a frequencia acumulada de todos os numeros
 * ate idx, i.e 1 2 2 2 2 3 4, read(2) retorna 5
 */
int read(int idx){
    int sum = 0;
    if(idx > maxy) idx = maxy;
    while(idx > 0) sum += tree[idx], idx -= (idx & -idx);
    return sum;
}
```

3 Ordenação

3.1 Merge Sort

```
void merge(int a[], int aux[], int esq, int meio, int dir) {
    int i, esq_fim, n, aux_pos;

    esq_fim = meio - 1;
    aux_pos = esq;
    n = dir - esq + 1;

    while(esq <= esq_fim && meio <= dir)
        if(a[esq] <= a[meio])
            aux[aux_pos++] = a[esq++];
        else {
            aux[aux_pos++] = a[meio++];
            c += esq_fim - esq + 1; /* conta swaps */
        }

    while(esq <= esq_fim)
        aux[aux_pos++] = a[esq++];

    while(meio <= dir)
        aux[aux_pos++] = a[meio++];

    for(i = 0; i < n; i++) {
        a[dir] = aux[dir];
        dir--;
    }
}

void mergesort(int a[], int aux[], int esq, int dir) {
    int meio;

    if(dir > esq) {
```

```
        meio = (dir + esq) / 2;
        mergesort(a, aux, esq, meio);
        mergesort(a, aux, meio + 1, dir);
        merge(a, aux, esq, meio + 1, dir);
    }
}
```

3.2 Quicksort

```
void quicksort(int d[], int n) {
    if(n > 1) {
        int piv = d[0], l = 1, r = n;
        int tmp;

        while(l < r) {
            if(d[l] <= piv) {
                l++;
            } else {
                tmp = d[l];
                d[l] = d[--r];
                d[r] = tmp;
            }
        }

        tmp = d[--l];
        d[l] = d[0];
        d[0] = tmp;

        quicksort(d, l);
        quicksort(d + r, n - r);
    }
}
```

4 Matemática

4.1 Bigmod

Propriedade:

$$(A \cdot B \cdot C) \bmod N = ((A \bmod N) \cdot (B \bmod N) \cdot (C \bmod N)) \bmod N$$

$$r = b^p \bmod m$$

```
long long bigmod(long long b, long long p, long long m) {
    if(p == 0)
        return 1;
    else if(p % 2 == 0)
        return square(bigmod(b, p/2, m)) % m;
    else
        return ((b % m) * bigmod(b, p-1, m)) % m;
}
```

4.2 Long Integer

```
void simplify(char v[]) {
    int i;
    i=strlen(v)-1;
    while(i>0 && v[i]=='0') {v[i]='\0';i--;}
}
void add(char v[], int q) {
    int c=q;
    int i,d;
    for(i=0;v[i];i++)
    {
        d=((v[i]-'0')+c);
        c=d/10;d%=10;
        v[i]='0'+d;
    }
    while(c)
    {
        v[i]='0'+(c%10);
        c/=10;i++;
    }
    v[i]='\0';
}
void multi(char v[], int q) {
    int c=0;
    int i,d;
    for(i=0;v[i];i++)
    {
        d=((v[i]-'0')*q+c);
        c=d/10;d%=10;
        v[i]='0'+d;
    }
    while(c)
    {
        v[i]='0'+(c%10);
        c/=10;i++;
    }
    v[i]='\0';
}
int divi(char v[], int q)
// returns the reminder
{
    int i,l=strlen(v);
    int c=0,d;
    for(i=l-1;i>=0;i--)
    {
        d=c*10+(v[i]-'0');
        c=d%q; d/=q; v[i]='0'+d;
    }
    i=l-1;
    while(i>0 && v[i]=='0') i--;
    v[i+1]='\0';
    return c;
}
void add(char v1[], char v2[])
// v1 = v1+v2;
{
        int i,d,c=0;
        int l1=strlen(v1);
        int l2=strlen(v2);
```

```
        for(i=l1;i<l2;i++) v1[i]='0';
        for(i=l2;i<l1;i++) v2[i]='0';
        for(i=0;i<l1||i<l2;i++)
        {
                d=(v1[i]-'0')+(v2[i]-'0')+c;
                c=d/10;d%=10;
                v1[i]='0'+d;
        }
        while(c)
        {
                v1[i]='0'+(c%10);
                c/=10;i++;
        }
        v1[i]='\0';
        v2[l2]='\0';
}
void subs(char v1[], char v2[])
// v1=v1-v2;
{
        int i,d,c=0;
        int l1=strlen(v1);
        int l2=strlen(v2);
        for(i=l2;i<l1;i++) v2[i]='0';
        for(i=0;i<l1;i++)
        {
                d=(v1[i]-'0'-c)-(v2[i]-'0');
                if(d<0) {d+=10; c=1;} else c=0;
                v1[i]='0'+d;
        }
        v2[l2]='\0';
    i=l1-1;
    while(i>0 && v1[i]=='0') i--;
    v1[i+1]='\0';
}
//return the sign of v1-v2
int comp(char v1[], char v2[]) {
    int i;
    int l1=strlen(v1);
    int l2=strlen(v2);
    if(l1>l2) return(1);
    if(l1<l2) return(-1);
    for(i=l1-1;i>=0;i--)
    {
        if(v1[i]>v2[i]) return(1);
        if(v1[i]<v2[i]) return(-1);
    }
    return(0);
}
char tmp[10000]; char tmp1[10000]; char tmpd[10][10000]; char
bs[10000];
void multi(char v1[], char v2[])
// v1=v1*v2;
{
    bs[0]='\0';
    int l2=strlen(v2);
    int i;
    strcpy(tmpd[0],"0");
    for(i=1;i<10;i++)
    {
        strcpy(tmpd[i],tmpd[i-1]);
```

```
            add(tmpd[i],v1);
        }
        strcpy(v1,"0");
        for(i=0;i<l2;i++)
        {
            strcpy(tmp,bs); bs[i]='0';bs[i+1]='\0';
            strcat(tmp,tmpd[v2[i]-'0']);
            add(v1,tmp);
        }
}
void multi(char v1[], char v2[], char v3[])
//make sure v1 is not v3
// v3=v1*v2;
{
        bs[0]='\0';
        int l2=strlen(v2);
        int i;
        strcpy(tmpd[0],"0");
        for(i=1;i<10;i++)
        {
            strcpy(tmpd[i],tmpd[i-1]);
            add(tmpd[i],v1);
        }
        strcpy(v3,"0");
        for(i=0;i<l2;i++)
        {
            strcpy(tmp,bs); bs[i]='0';bs[i+1]='\0';
            strcat(tmp,tmpd[v2[i]-'0']);
            add(v3,tmp);
        }
}
void divi(char v1[], char v2[], char v3[], char v4[])
//v1/v2=v3...v4
// make sure v3, v4 are different from v1, v2
{
        int i;
        if(strcmp(v2, "1")==0)
        {
            strcpy(v3, v1);
            strcpy(v4, "0");
            return;
        }
        if(strcmp(v1, "0")==0)
        {
            strcpy(v3, "0");
            strcpy(v4, "0");
            return;
        }
        for(i=0;v1[i];i++) v3[i]='0';
        v3[i]='\0';
        int ff=1;
        int l=i;
        for(i=l-1;i>=0;i--)
        {
            while(1)
            {
                if(v3[i]=='9') break;
                v3[i]++;
                multi(v3, v2, v4);
                if(comp(v4, v1)>0)
```

```
            {
                v3[i]--;
                break;
            }
            ff=0;
        }
        if(ff && i) v3[i]='\0';
        //simplify(v3);
    }
    multi(v2, v3, tmp1);
    strcpy(v4, v1);
    subs(v4, tmp1);
}
void showBigint(char v[]) {
    simplify(v);
    int l=strlen(v);
    int i;
    for(i=l-1;i>=0;i--) cout<<v[i];
}
void rev(char v[]) {
    int l=strlen(v);
    int i; char cc;
    for(i=0;i<l-1-i;i++)
    {
        cc=v[i];v[i]=v[l-1-i];v[l-i-1]=cc;
    }
}
```

4.3 Sieve de Números Primos

```
#include <cstring>

#define MAXN  100000000  /* valor maximo de N */
#define P1    1562501    /* = ceil(MAXN/64) + 1 */
#define P2    50000000   /* = ceil(MAXN/2) */
#define P3    5000       /* = ceil(ceil(sqrt(MAXN))/2) */

unsigned s[P1];

#define GET(b) ((s[(b) >> 5] >> ((b) & 31)) & 1)

void make() {
    register unsigned i, j, k;
    memset(s, 0, sizeof(s));
    for(k = 1; k <= P3; k++)
        if(GET(k) == 0)
            for(j = 2*k+1, i = 2*k*(k+1); i < P2; i += j)
                s[i >> 5] |= 1 << (i & 31);
}

int isprime(int p) {
    return p == 2 || (p > 2 && (p & 1) == 1 && (GET((p-1) >> 1) == 0));
}
```

4.4 GCD e LCM

```
long long gcd(long long a, long long b) {
```

```
    if(a % b == 0) return b;
    return gcd(b, a % b);
}

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

4.5 Algoritmo de Euclides

```
struct Triple{
    int d, x, y;
    Triple() {}
    Triple( int q, int w, int e ) : d( q ), x( w ), y( e ) {}
};
/* Retorna uma tripla T, onde
** gcd(a,b) = T.d = T.x * a + T.y * b
*/
Triple egcd( int a, int b ){
    if(!b) return Triple(a, 1, 0);
    Triple q = egcd(b, a%b);
    return Triple(q.d, q.y, q.x-a/b*q.y);
}
/* Retorna x tal que ax = 1 (mod n) */
int invmod( int a, int n )
{
    Triple t = egcd( a, n );
    if( t.d > 1 ) return 0;
    int r = t.x % n;
    return( r < 0 ? r + n : r );
}
```

4.6 Combinação

$$C_n^k = \frac{n!}{k! \cdot (n-k)!}$$

```
void divbygcd(long long& a, long long& b) {
    long long g = gcd(a, b);
    a /= g;
    b /= g;
}
long long C(int n, int k){
    long long numerator=1, denominator=1, toMul, toDiv, i;
    if(k > n/2) k = n - k;                /* use smaller k */
    for(i = k; i; i--) {
        toMul = n - k + i;
        toDiv = i;
        divbygcd(toMul, toDiv);           /* always divide before multiply */
        divbygcd(numerator, toDiv);
        divbygcd(toMul, denominator);
        numerator *= toMul;
        denominator *= toDiv;
    }
    return numerator / denominator;
}
```

4.7 Floyd's Cycle-Finding

```
int f(int x) { return (x + 3) % 7; } /* exemplo de funcao */

int floyd(int x0) {
        int t, h, len, first;

        /* encontra repeticao */
        t = f(x0);
        h = f(f(x0));
        while(t != h) {
            t = f(t);
            h = f(f(h));
        }

        /* encontra a posicao do primeiro ciclo (opcional) */
        first = 0;
        h = t;
        t = x0;
        while(t != h) {
            t = f(t);
            h = f(h);
            first++;
        }

        /* encontra o tamanho ciclo (opcional) */
        length = 1;
        h = f(t);
        while(t != h) {
            h = f(h);
            length++;
        }

        return length;
}
```

4.8 Matrizes

4.8.1 Decomposição LUP

Determina uma decomposição LUP para a matriz A: $PA = LU$ Tempo: $\Theta(n^3)$

```
/* LUP-DECOMPOSITION (Cormen, p. 752) */
/* Retorna Psgn = det P (ou 0, se a matriz for singular) */
int lupdecomp(double a[NN][NN], double lu[NN][NN], int p[NN], int n) {
    int i, j, k, psgn = 1;
    /* copia A para LU */
    for(i = 0; i < n; i++) {
        p[i] = i;
        for(j = 0; j < n; j++) lu[i][j] = a[i][j];
    }
    for(k = 0; k < n; k++) {
        double v = 0.0;
        int kl;
        /* encontra o maior elemento lu[kl][k] na coluna k */
        for(i = k; i < n; i++) {
            if(fabs(lu[i][k]) - v > EPS) {
                v = fabs(lu[i][k]);
                kl = i;
            }
```

```
        }
        if(fabs(v) < EPS) return 0; /* matriz singular */
        if(k != kl) {
            /* troca linhas k e kl */
            std::swap(p[k], p[kl]), psgn *= -1;
            for(i = 0; i < n; i++)
                std::swap(lu[k][i], lu[kl][i]);
        }
        /* computa o complemento Schur */
        for(i = k+1; i < n; i++) {
            lu[i][k] /= lu[k][k];
            for(j = k+1; j < n; j++)
                lu[i][j] -= lu[i][k]*lu[k][j];
        }
    }
    return psgn;
}
```

4.8.2 Determinante

```
double det(double a[NN][NN], int n) {
    int p[NN], detp;
    double lu[NN][NN], d = 1.0;
    detp = lupdecomp(a, lu, p, n);
    if(detp == 0) return 0; /* matriz singular */
    for(int i = 0; i < n; i++) d *= a[i][i];
    return d/detp;
}
```

4.8.3 Sistemas Lineares

Resolve um sistema de equações lineares da forma $Ax = B$

```
/* LUP-SOLVE (Cormen, p. 745) */
void linearsolve(double a[NN][NN], double b[NN], double x[NN], int n) {
    double lu[NN][NN], y[NN], sum;
    int i, j, p[NN];
    lupdecomp(a, lu, p, n); /* cuidado, supoe que eh invertivel */
    sum = 0.0;
    for(i = 0; i < n; i++) {
        y[i] = b[p[i]];
        for(j = 0; j < i; j++) y[i] -= lu[i][j]*y[j];
    }
    for(i = n-1; i >= 0; i--) {
        x[i] = y[i];
        for(j = i+1; j < n; j++) x[i] -= lu[i][j]*x[j];
        x[i] /= lu[i][i];
    }
}
```

5 Grafos

5.1 Strongly Connected Components

5.1.1 Kosaraju

Tempo: $O(V + E)$

```
int n;
int adj[NN][NN], deg[NN]; /* G */
int jda[NN][NN], ged[NN]; /* G^T */
int vis[NN], num[NN], scc[NN];
int cnt, scccnt;

void dfs(int v) {
    int i;
    vis[v] = 1;
    for(i = 0; i < deg[v]; i++)
        if(!vis[adj[v][i]]) dfs(adj[v][i]);
    num[cnt++] = v;
}

void dfst(int v) {
    int i;
    vis[v] = 1;
    scc[v] = scccnt;
    for(i = 0; i < ged[v]; i++)
        if(!vis[jda[v][i]]) dfst(jda[v][i]);
}

void findscc() {
    int i;
    /* dfs em G */
    for(i = 0; i < n; i++) vis[i] = 0;
    cnt = 0;
    for(i = 0; i < n; i++)
        if(!vis[i])
            dfs(i);

    /* dfs em G^T */
    for(i = 0; i < n; i++) vis[i] = 0;
    scccnt = 0;
    for(i = n-1; i >= 0; i--)
        if(!vis[num[i]])
            dfst(num[i]), scccnt++;
}
```

5.1.2 Tarjan

```
int idx[4000], nnnn, low[4000], SCC[4000], stack[4000], scc_count, top;
bool stacked[4000], NO;
void push(int a){
    stack[top++] = a;
    stacked[a] = 1;
}
int pop(){
    stacked[stack[--top]] = 0;
    return stack[top];
}
void tarjan(int k){

    idx[k] = low[k] = nnnn = nnnn+1;
    stack[top++] = k;
    stacked[k] = 1;
    for(typeof(adj[k].begin())it = adj[k].begin(); it != adj[k].end(); it++){
        int nadj = (*it);
```

```
            if(idx[nadj] == -1){
                tarjan(nadj);
                low[k] = MIN(low[k],low[nadj]);
            } else if(stacked[nadj]) {
                low[k] = MIN(low[k],idx[nadj]);
            }
        }
        if(low[k] == idx[k]){
            scc_count++;
            int x = -1;
            while(x != k)SCC[x = pop()] = scc_count;
        }
}
```

5.2 Caminho mínimo

5.2.1 Floyd-Warshall

Tempo: $\Theta(n^3)$

```
int w[MAX_NODES][MAX_NODES];           /* adjacencias */
int d[MAX_NODES][MAX_NODES];           /* distancias */
int p[MAX_NODES][MAX_NODES];           /* predecessores */

/* algoritmo */
void floyd_warshall() {
    for(i = 0; i < n; i++) {
        for(j = 0; j < n; j++) {
            d[i][j] = (i == j ? 0 : w[i][j]);
            p[i][j] = i;
        }
    }

    for(k = 0; k < n; k++) {
        for(i = 0; i < n; i++) {
            if(i != k && d[i][k] < INF) { // opcional, deixa + rapido
                for(j = 0; j < n; j++) {
                    if(d[i][k] + d[k][j] < d[i][j]) {
                        d[i][j] = d[i][k] + d[k][j];
                        p[i][j] = p[k][j];
                    }
                } // for j
            } // opcional
        } // for i
    } // for k
}

/* reconstruir caminho entre i e j */
void print_path(int i, int j) {
    if(i != j) print_path(i, p[i][j]);
    print(j);
}
```

5.2.2 Dijkstra

Tempo: $O(|V|^2)$

```
int n;
int w[NN][NN];
```

```

int dijkstra(int s, int t) {
    int in[NN], d[NN];
    int i, u;

    for(i = 0; i < n; i++)
        d[i] = w[s][i], in[i] = 0;
    d[s] = 0;

    while(!in[t]) {
        int best = INF;
        for(i = 0; i < n; i++)
            if(!in[i] && best > d[i])
                best = d[u = i];
        if(best == INF) break;

        in[u] = 1;

        for(i = 0; i < n; i++)
            if(!in[i] && w[u][i] < INF && d[i] > d[u] + w[u][i])
                d[i] = d[u] + w[u][i];
    }

    return d[t];
}
```

5.2.3 Dijkstra com Heap

Tempo: $O(|E| + |V| \log |V|)$

5.2.4 Bellman-Ford

Tempo: $O(|V||E|)$

```
struct { int u, v, w; } e[MM]; /* arestas (u,v) com peso w */

int bellmanford(int s, int t) {
    int di[NN], p[NN]; /* distancias e predecessores */
    int i, j;

    for(i = 0; i < n; i++)
        di[i] = INF, p[i] = -1;

    di[s] = 0;

    for(j = 0; j < n; j++) {
        bool trocou = false;
        for(i = 0; i < m; i++) {
            int u = e[i].u, v = e[i].v;
            if(di[v] > di[u] + e.w) {
                di[v] = di[u] + e.w;
                p[v] = u;
                trocou = true;
            }
        }
        if(!trocou) break;
    }

    return di[t];
```

```
}
```

5.3 Árvore Geradora Mínima

5.3.1 Prim

5.3.2 Kruskal

Tempo: $O(|E| \log |V|)$ ou $O(|E| \cdot \alpha(|V|))$ caso as arestas estejam pré-ordenadas.

```
int n; /* numero de vertices */

double kruskal() {
    priority_queue<pair<double, pair<int, int> > > q; /* fila de adjacencias */
    double total;
    int c; /* numero de componentes desconexas do grafo */

    /* inserir adjacencias na fila! */

    c = n;
    for(i = 0; i < n; i++) makeset(i);

    total = 0.0;
    while(c > 1) {
        double w = -q.top().first;
        int a = q.top().second.second;
        int b = q.top().second.second;
        q.pop();
        if(findset(a) != findset(b)) {
            /* aresta a-b faz parte da MST */
            unionset(a, b);
            total += w;
            c--;
        }
    }

    return total;
}
```

5.4 Fluxo Máximo e Corte Mínimo

5.4.1 Edmonds-Karp

Tempo: $O(n!)$

```
#define MAX_NODES 100

int c[MAX_NODES][MAX_NODES];    /* capacidade */
int f[MAX_NODES][MAX_NODES];    /* fluxo */
int v[MAX_NODES];               /* visitados */
int p[MAX_NODES];               /* predecessores no caminho */
int n;                          /* numero de vertices */

bool bfs(int s, int t) {
    queue<int> q;
    int i, j;
    for(i = 0; i < n; i++)
        v[i] = 0;
    q.push(s);
```

```
    v[s] = 1;
    p[s] = -1;
    while(!q.empty() && v[t] == 0) {
        i = q.front();
        q.pop();
        for(j = 0; j < n; j++) {
            if(v[j] == 0 && c[i][j] - f[i][j] > 0) {
                q.push(j);
                v[j] = 1;
                p[j] = i;
            }
        }
    }
    return v[t] == 1;
}

int max_flow(int s, int t) {
    int mf = 0;
    int i, j;

    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            f[i][j] = 0;

    while(bfs(s, t)) {
        int cf = INF;

        for(i = t; p[i] >= 0; i = p[i])
            cf = min(cf, c[p[i]][i] - f[p[i]][i]);

        for(i = t; p[i] >= 0; i = p[i]) {
            f[p[i]][i] += cf;
            f[i][p[i]] -= cf;
        }

        mf += cf;
    }

    return mf;
}
```

5.4.2 Stoer-Wagner

Algoritmo para encontrar o corte mínimo (não s-t) em um grafo. A implementação abaixo é $O(n^3)$.

```
int n;
int adj[MAX_NODES][MAX_NODES];

/* Stoer-Wagner {{{ */
int v[MAX_NODES], cn;

int mincutphase() {
    int w[MAX_NODES], a[MAX_NODES];
    int i, j;

    /* inicializa conjunto A e pesos dos vertices */
    a[v[0]] = 1;

    for(i = 1; i < cn; i++) {
```

```
            a[v[i]] = 0;
            w[i] = adj[v[0]][v[i]];
    }

    /* insere os outros vertices */
    int prev = v[0];
    for(i = 1; i < cn; i++) {
        /* encontra o "most tightly connected vertex" nao pertencente a A */
        int zj = -1;
        for(j = 1; j < cn; j++)
            if(!a[v[j]] && (zj < 0 || w[j] > w[zj]))
                zj = j;

        /* insere o vertice no conjunto A */
        a[v[zj]] = 1;

        /* ultimo vertice? */
        if(i == cn - 1) {
            /* merge prev e v[zj] */
            for(i = 0; i < cn; i++)
                adj[v[i]][prev] = adj[prev][v[i]] += adj[v[zj]][v[i]];
            v[zj] = v[--cn];

            return w[zj];
        }
        prev = v[zj];

        /* atualiza os pesos dos vizinhos */
        for(j = 1; j < cn; j++)
            if(!a[v[j]])
                w[j] += adj[v[zj]][v[j]];
    }

    return INF;
}

int mincut() {
    int best = INF;

    cn = n;
    for(int i = 0; i < n; i++) v[i] = i;

    while(cn > 1)
        best = min(mincutphase(), best);

    return best;
}
/* }}} */
```

5.4.3 Min-Cost Max-Flow

Tempo: $O(min(n^2m \cdot fcost,\ nm \cdot flow))$

```
int n;
int f[NN][NN], cap[NN][NN], cost[NN][NN]; /* fluxo, capacidade, custo */
list< pair<int, int> > edges; /* lista de adjacencias */

int p[NN], d[NN]; /* previous, dists do bellman-ford */

int bellmanford(int s, int t) {
```

```
    int i;

    for(i = 0; i < n; i++) d[i] = INF;

    d[s] = 0;
    p[s] = -1;

    for(i = 0; i < n; i++) {
        bool troca = false;
        foreach(e, edges) {
            int u = e->first, v = e->second;

            /* tenta reduzir fluxo v->u */
            if(f[v][u] && d[v] > d[u] - cost[v][u]) {
                p[v] = u;
                d[v] = d[u] - cost[v][u];
                troca = true;
            }

            /* tenta reforcar fluxo u->v */
            if(f[u][v] < cap[u][v] && d[v] > d[u] + cost[u][v]) {
                p[v] = u;
                d[v] = d[u] + cost[u][v];
                troca = true;
            }
        }
        if(!troca) break;
    }

    return d[t];
}

void mincostmaxflow(int &fcost, int &flow, int s, int t) {
    int i, j;

    /* inicia rede de fluxos */
    fcost = flow = 0;
    for(i = 0; i < n; i++)
        for(j = 0; j < n; j++)
            f[i][j] = 0;

    while(bellmanford(s, t) < INF) {
        int cf = INF;

        /* encontra gargalo */
        for(i = t; p[i] >= 0; i = p[i])
            if(f[i][p[i]]) cf = min(cf, f[i][p[i]]);
            else cf = min(cf, cap[p[i]][i] - f[p[i]][i]);

        /* atualiza rede residual */
        for(i = t; p[i] >= 0; i = p[i]) {
            if(f[i][p[i]]) {
                f[i][p[i]] -= cf;
                fcost -= cf * cost[i][p[i]];
            } else {
                f[p[i]][i] += cf;
                fcost += cf * cost[p[i]][i];
            }
        }

```

```
            flow += cf;
        }
}
```

5.4.4 Kuhn-Munkres

Dado um grafo completo bipartido com pesos $G = (X \cup Y, X \times Y)$, encontra um matching M de X para Y com peso total máximo. Para encontrar com peso mínimo, basta negar todos os custos.

Tempo: $O(n^3)$

```
int n;
int cost[NN][NN];

int x[NN], y[NN], lx[NN], ly[NN], prev[NN], q[NN];

int hungarian() {
    int i, j, k, s, head, tail;
    int ret = 0;

    /* init */
    for(i = 0; i < n; i++) {
        lx[i] = ly[i] = 0;
        for(j = 0; j < n; j++) lx[i] = max(lx[i], cost[i][j]);
        x[i] = y[i] = -1;
    }

    /* go! go! go! */
    for(i = 0; i < n; i++) {
        /* procura caminho aumentante (BFS) */
        for(j = 0; j < n; j++) prev[j] = -1;
        for(q[0] = i, head = 0, tail = 1; head < tail && x[i] < 0; head++) {
            s = q[head];
            for(j = 0; j < n; j++) {
                if(x[i] >= 0) break;
                if(lx[s] + ly[j] > cost[s][j] || prev[j] >= 0) continue;
                q[tail++] = y[j];
                prev[j] = s;
                if(y[j] < 0) while(j >= 0) {
                    s = y[j] = prev[j];
                    k = x[s]; /* swap(x[s], j) */
                    x[s] = j;
                    j = k;
                }
            }
        }

        /* se nao encontrou caminho, atualiza labels e tenta de novo */
        if(x[i] < 0) {
            k = INF;
            for(head = 0; head < tail; head++)
                for(j = 0; j < n; j++)
                    if(prev[j] == -1)
                        k = min(k, lx[q[head]] + ly[j] - cost[q[head]][j]);
            for(j = 0; j < tail; j++)
                lx[q[j]] -= k;
            for(j = 0; j < n; j++)
                if(prev[j] >= 0)
                    ly[j] += k;
```

```
            i--;
        }
    }

    /* calcula soma */
    for(i = 0; i < n; i++)
        if(x[i] >= 0)
            ret += cost[i][x[i]];

    return ret;
}
```

5.5 Aresta de Corte

```
int w[MAX_NODES][MAX_NODES];      /* matriz de adjacencia */
int num_nodes;                    /* numero de nos */

/* variaveis do bridgeR */
int cnt, lbl[MAX_NODES], low[MAX_NODES], parnt[MAX_NODES];
int bcnt;                         /* numero de bridges */

void bridgeR(int v) {
    int x;

    lbl[v] = cnt++;
    low[v] = lbl[v];

    for(x = 0; x < num_nodes; x++) {
        if(!w[v][x])
            /* se x nao eh adjacente a v, continue */
            continue;

        if(lbl[x] == -1) {
            /* se x nao foi visitado ainda, visite */
            parnt[x] = v;
            bridgeR(x);

            if(low[v] > low[x]) low[v] = low[x];
            if(low[x] == lbl[x]) {
                /* v-x eh uma ponte */
                printf("%d-%d\n", v, x);
                bcnt++;
            }
        } else if(x != parnt[v] && low[v] > lbl[x]) {
            low[v] = lbl[x];
        }
    }
}

void all_bridges() {
    int v;

    bcnt = cnt = 0;

    for(v = 0; v < num_nodes; v++)
        lbl[v] = -1;

    for(v = 0; v < num_nodes; v++) {
        if(lbl[v] == -1) {
```

```
            parnt[v] = v;
            bridgeR(v);
        }
    }
}
```

5.6 Ponto de Articulação

```
int nos, nArvore;                 /* inicializar nArvore = 0 */
int grafo[MAX_NOS][MAX_NOS];
int tras[MAX_NOS];
int arvore[MAX_NOS];              /* iniciar em -1 */
int raiz, arcosArvoreRaiz;

void pontoArt(int no) {
    nArvore++;
    arvore[no] = nArvore;
    tras[no] = arvore[no];

    for(int i = 0; i < nos; i++) {
        if(grafo[no][i] && i != no) {
            if(arvore[i] == -1) {
                if(no == raiz)
                    arcosArvoreRaiz++;
                pontoArt(i);
                if(tras[i] >= arvore[no] && ((no == raiz && arcosArvoreRaiz > 1)
                    || no != raiz)) {
                    /* 'no' eh um ponto de articulacao */
                    printf("%d ", no);
                } else {
                    tras[no] = min(tras[no], tras[i]);
                }
            } else {
                tras[no] = min(tras[no], arvore[i]);
            }
        }
    }
}

void all_pontoarts() {
    int v;

    nArvore = 0;

    for(v = 0; v < nos; v++)
        arvore[v] = -1;

    for(v = 0; v < nos; v++) {
        if(arvore[v] == -1) {
            arcosArvoreRaiz = 0;
            raiz = v;
            pontoArt(v);
        }
    }
}
```

5.7 Lowest Common Ancestor

5.8 Caminho Euleriano

5.9 Ciclo Euleriano

5.10 Fatos Interessantes

1. Um grafo é bipartido se e somente se não tem ciclo ímpar.

2. **Teorema de Euler**. Para qualquer grafo planar, $V - E + F = 1 + C$, onde V é o número de vértices do grafo, E é o número de arestas, F é o número de faces do grafo em um desenho planar e C é o número de componentes conexas.

3. **Teorema de Kirchhoff**. Seja a matriz $T = [t_{ij}]$, onte t_{ij} é o número de arestas entre i e j, para $i \neq j$, e $t_{ii} = -deg_i$. O número de árvores geradoras de um grafo é igual ao determinante de uma matriz obtida ao deletar qualquer k-ésima linha e k-ésima coluna de T.

6 Programação Dinâmica

6.1 Longest Common Subsequence

Tempo: $O(nm)$

```
char a[MAX], b[MAX];
int n, m; /* n = strlen(a), m = strlen(b) */
int c[MAX+1][MAX+1], d[MAX+1][MAX+1];

void printlcs(int i, int j) {
    if(i == 0 || j == 0) return;
    if(d[i][j] == 1) {
        printlcs(i-1, j-1);
        printf("%c", a[i-1]);
    } else if(d[i][j] == 2) printlcs(i-1, j);
    else printlcs(i, j-1);
}

void computelcs() {
    int i, j;
    for(i = 0; i <= n; i++) c[i][0] = 0;
    for(j = 0; j <= m; j++) c[0][j] = 0;
    for(i = 1; i <= n; i++) {
        for(j = 1; j <= m; j++) {
            if(a[i-1] == b[j-1]) {
                c[i][j] = c[i-1][j-1] + 1;
                d[i][j] = 1;
            } else if(c[i-1][j] >= c[i][j-1]) {
                c[i][j] = c[i-1][j];
                d[i][j] = 2;
            } else {
                c[i][j] = c[i][j-1];
                d[i][j] = 0;
            }
        }
    }
}
```

6.2 Longest Increasing Subsequence

Tempo: $O(n \log n)$

```
int b[MAX]; // ARRAY DE INDICES DA LIS
int p[MAX]; // ARARY DE PREDECESSORES
int nums[MAX]; // ARRAY DE NUMEROS
void lis(int n){
    if (n < 1) return;
    int i, u, v, tail = -1;
    b[++tail] = 0;
    memset(p,-1,sizeof(int)*n);
    for (i = 1; i < n; i++) {
        /*
         * Se o numero atual(nums[i])for maior
         * que o ultimo elemento da LIS
         * insira-o no fim da LIS e va
         * para o proximo numero
         */
        if (nums[b[tail]] < nums[i]) {
            p[i] = b[tail];
            b[++tail] = i;
            continue;
        }
        /*
         * Busca binaria para encontrar
         * o melhor lugar para inserir
         * o numero atual (nums[i])
         */
        for (u = 0, v = tail; u < v;) {
            int c = (u + v)>>1;
            if (nums[b[c]] < nums[i]) u=c+1; else v=c;
        }
        /*
         * Substitui o elemento da posicao
         * u da LIS se, e somente se,
         * o numero atual for menor que ele
         */
        if (nums[i] < nums[b[u]]) {
            if (u) p[i] = b[u-1];
            b[u] = i;
        }
    }
    /* arruma o array de predecessores */
    for (u = tail+1, v = b[tail]; u--; v = p[v]) b[u] = v;
    /* print */
    for(i =0; i <= tail; i++) printf("%d\n",nums[b[i]]);
}
```

6.3 Matrix-chain Multiplication

Tempo: $O(n^3)$

```
int num_arrays;
int p[MAX_ARRAYS];
int m[MAX_ARRAYS][MAX_ARRAYS];
int s[MAX_ARRAYS][MAX_ARRAYS];

/* MATRIX-CHAIN-ORDER (Cormen, p. 336) */
void matrix_chain_order() {
```

```
    int l, i, j, k, q;
    for(l = 2; l <= num_arrays; l++) {
        for(i = 1; i <= num_arrays - l + 1; i++) {
            j = i + l - 1;
            m[i][j] = INT_MAX;
            for(k = i; k <= j - 1; k++) {
                q = m[i][k] + m[k + 1][j] + p[i-1]*p[k]*p[j];
                if(q < m[i][j]) {
                    m[i][j] = q;
                    s[i][j] = k;
                }
            }
        }
    }
}


/* PRINT-OPTIMAL-PARENS (Cormen, p. 338) */
void print_optimal_parens(int i, int j) {
    if(i == j)
        printf("A%d", i);
    else {
        printf("(");
        print_optimal_parens(i, s[i][j]);
        printf(" x ");
        print_optimal_parens(s[i][j] + 1, j);
        printf(")");
    }
}
```

6.4 0-1 Knapsack

6.5 CYK

Tempo: $O(n^3)$

7 Geometria Computacional

```
struct point { int x, y; };
struct polygon { point p[MAX_VERTS]; int n; };
struct circle { point c; double r; };

int distsqr(point a, point b) {
    return (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y);
}

int cross(point a, point b, point c) {
    return (b.x - a.x)*(c.y - a.y) - (c.x - a.x)*(b.y - a.y);
}
```

7.1 Círculos

```
/* c = circulo que passa por a, b e tem raio r (ordem a, b importa) */
bool circle2ptsRad(point a, point b, double r, circle &c) {
    double det = r*r/distsqr(a, b) - 0.25;
```

```
    if(det < 0) return false;
    double h = sqrt(det);
    c.r = r;
    c.c.x = (a.x + b.x)*0.5 + (a.y - b.y)*h;
    c.c.y = (a.y + b.y)*0.5 + (b.x - a.x)*h;
    return true;
}

/* circulo por 3 pontos (shygypsy), TODO: mudar pro formato do nosso manual */
bool lineIntersect(double x[], double y[], double r[]) {
    double n[2]; n[0] = y[3] - y[2]; n[1] = x[2] - x[3];
    double denom = n[0] * (x[1] - x[0]) + n[1] * (y[1] - y[0]);
    if(fabs(denom) < EPS) return false;
    double num = n[0] * (x[0] - x[2]) + n[1] * (y[0] - y[2]);
    double t = -num / denom;
    r[0] = x[0] + t * (x[1] - x[0]);
    r[1] = y[0] + t * (y[1] - y[0]);
    return true;
}

double circle3pts(double x[], double y[], double r[]) {
    double lix[4], liy[4];
    lix[0] = 0.5 * (x[0] + x[1]); liy[0] = 0.5 * (y[0] + y[1]);
    lix[1] = lix[0] + y[1] - y[0];  liy[1] = liy[0] + x[0] - x[1];
    lix[2] = 0.5 * (x[1] + x[2]); liy[2] = 0.5 * (y[1] + y[2]);
    lix[3] = lix[2] + y[2] - y[1];  liy[3] = liy[2] + x[1] - x[2];
    if(!lineIntersect(lix, liy, r)) return -1.0;
    return sqrt((r[0] - x[0]) * (r[0] - x[0]) + (r[1] - y[0]) * (r[1] - y[0]));
}
```

7.2 Intersecção de Segmentos

```
bool on_segment(point i, point j, point k) {
    return min(i.x, j.x) <= k.x && k.x <= max(i.x, j.x)
           && min(i.y, j.y) <= k.y && k.y <= max(i.y, j.y);
}

int direction(point a, point b, point c) {
    return cross(a, c, b);
}

/* true se os segmentos ab e cd intercedem */
bool segments_intersect(point a, point b, point c, point d) {
    int d1 = direction(c, d, a);
    int d2 = direction(c, d, b);
    int d3 = direction(a, b, c);
    int d4 = direction(a, b, d);

    if(((d1 > 0 && d2 < 0) || (d1 < 0 && d2 > 0))
           && ((d3 > 0 && d4 < 0) || (d3 < 0 && d4 > 0))) return true;
    if(d1 == 0 && on_segment(c, d, a)) return true;
    if(d2 == 0 && on_segment(c, d, b)) return true;
    if(d3 == 0 && on_segment(a, b, c)) return true;
    if(d4 == 0 && on_segment(a, b, d)) return true;

    return false;
}
```

7.3 Ponto dentro de Polígono

Constante de tempo baixa, não faz uso de funções de trigonometria. Retorna *true* se o ponto está sob um dos lados do polígono (facilmente alterável).

```
bool inpoly(polygon q, point p) {
    bool in = false;
    point a, b;
    int i, j, t;

    for(i = 0; i < q.n; i++) {
        j = (i+1) % q.n;
        if(q.p[j].x > q.p[i].x) {
            a = q.p[i];
            b = q.p[j];
        } else {
            a = q.p[j];
            b = q.p[i];
        }
        t = cross(a, b, p);
        if(t == 0 && on_segment(a, b, p)) return true; /* LADO! */
        if((q.p[j].x < p.x) == (p.x <= q.p[i].x) && t < 0)
            in = !in;
    }

    return in;
}
```

7.4 Teste de Convexidade

Algoritmo para testar se um polígono é convexo ou não.

7.5 Área de um Polígono

7.5.1 Convexo

Se o polígono está no sentido anti-horário, a área é positiva. Se estiver no sentido horário, a área é negativa.

```
double polygon_area(polygon q) {
    double r = 0.0;
    int i, j;
    for(i = 0; i < q.n; i++) {
        j = (i+1) % q.n;
        r += q.p[i].x * q.p[j].y - q.p[j].x * q.p[i].y;
    }
    return r/2.0;
}
```

7.5.2 Qualquer

```
double polygon_area(polygon q) {
    double r = 0.0;
    int i;
    for(i = 1; i < q.n-1; i++){
        int x1 = q.p[i].x - q.p[0].x;
        int y1 = q.p[i].y - q.p[0].y;
        int x2 = q.p[i+1].x - q.p[0].x;
```

```
        int y2 = q.p[i+1].y - q.p[0].y;
        r += x1*y2 - x2*y1;
    }
    return fabs(r/2.0);
}
```

7.6 Convex Hull

7.6.1 Graham Scan

Tempo: $O(n \log n)$

```
#include <algorithm>
#include <vector>

point pivot;

bool graham_cmp(point a, point b) {
    int t = cross(pivot, b, a);
    if(t < 0) return true;
    if(t == 0) return distsqr(pivot, a) < distsqr(pivot, b);
    return false;
}

/* q = CH(p) */
void graham_scan(polygon &q, polygon &p) {
    vector<point> s(p.p, p.p+p.n);
    int i;

    /* pivot = ponto mais abaixo e mais a esquerda */
    pivot = p.p[0];
    for(i = 1; i < p.n; i++)
        if(p.p[i].y < pivot.y || (p.p[i].y == pivot.y && p.p[i].x < pivot.x))
            pivot = p.p[i];

    /* ordena s no sentido anti-horario, com relacao ao pivot */
    sort(s.begin(), s.end(), graham_cmp);

    /* remove elementos repetidos de s */
    for(i = 2; i < (int)s.size(); i++)
        if(cross(pivot, s[i], s[i-1]) == 0)
            s.erase(s.begin() + --i);

    /* here comes the fun */
    q.p[0] = s[0];
    q.p[1] = s[1];
    q.p[2] = s[2];
    q.n = 3;
    for(i = 3; i < (int)s.size(); i++) {
        while(cross(q.p[q.n-2], q.p[q.n-1], s[i]) <= 0) q.n--;
        q.p[q.n++] = s[i];
    }
}
```

7.6.2 Jarvis March

Tempo: $O(nh)$

```
/* q = CH(p) */
```

```
void jarvis_march(polygon &q, polygon &p) {
    bool used[M];
    int top, bottom;
    int current, next;
    int i;

    q.n = 0;

    top = bottom = 0;
    for(i = 0; i < p.n; i++) {
        used[i] = 0;
        if(p.p[top].y < p.p[i].y) top = i;
        if(p.p[bottom].y > p.p[i].y || (p.p[bottom].y == p.p[i].y && p.p[bottom
            ].x > p.p[i].x)) bottom = i;
    }

    /* right-chain */
    current = bottom;
    while(current != top) {
        used[current] = 1;
        q.p[q.n].x = p.p[current].x;
        q.p[q.n].y = p.p[current].y;
        q.n++;
        next = top;
        for(i = 0; i < p.n; i++) {
            if(used[i]) continue;
            int t = cross(p.p[current], p.p[i], p.p[next]);
            if(t > 0) {
                next = i;
            } else if(t == 0 && distsqr(p.p[i], p.p[current]) > distsqr(p.p[next
                ], p.p[current])) {
                next = i;
            }
        }
        current = next;
    }

    /* left-chain */
    current = top; /* ! */
    while(current != bottom) { /* ! */
        used[current] = 1;
        q.p[q.n].x = p.p[current].x;
        q.p[q.n].y = p.p[current].y;
        q.n++;
        next = bottom; /* ! */
        for(i = 0; i < p.n; i++) {
            if(used[i]) continue;
            int t = cross(p.p[current], p.p[i], p.p[next]);
            if(t > 0) {
                next = i;
            } else if(t == 0 && distsqr(p.p[i], p.p[current]) > distsqr(p.p[next
                ], p.p[current])) {
                next = i;
            }
        }
        current = next;
    }
}
```

7.7 Closest Pair of Points

8 Outros

8.1 2-SAT

$$(a_1 \vee a_2) \wedge (b_1 \vee b_2) \wedge \ldots \wedge (z_1 \vee z_2)$$

Para determinar se uma fórmula com duas literais por cláusula é satisfazível, basta:

1. Trocar cada cláusula $(x_1 \vee x_2)$ por $(\neg x_1 \rightarrow x_2) \wedge (\neg x_2 \rightarrow x_1)$.
2. Criar um grafo G:
   (a) Para cada literal x, criar um nó x e $\neg x$.
   (b) Para cada cláusula $(a \rightarrow b)$, criar uma aresta (a, b) no grafo.
3. Encontrar SCC do grafo G.
4. Se todo literal x estiver em uma SCC diferente de $\neg x$, a formula é satisfazível.

Caso exista uma cláusula com apenas um literal x, basta trocá-la por $(1 \vee x)$ e, portanto, criar os nós 0 e 1 e as arestas:

1. $(0, 1)$.
2. $(0, a)$, $(0, \neg a)$, $(a, 1)$ e $(\neg a, 1)$ para *todos* os literais a da fórmula.
3. Obviamente, $(1, x)$ e $(\neg x, 0)$.

8.2 Sist. de restrições de diferenças

Dado um sistema de n incógnitas x_i $(1 \leq i \leq n)$ com m inequações na forma $x_j - x_i \leq b_k$, modela-se um grafo com $V = \{v_0, v_1, \ldots, v_n\}$, uma aresta (v_i, v_j) para cada inequação $x_j - x_i \leq b_k$, com $w(v_i, v_j) = v_k$ e uma aresta (v_0, v_i), com $w(v_0, v_i) = 0$ para cada $v_i \in V - \{v_0\}$.

Por fim, rodar Bellman-Ford no grafo construído, caso existam ciclos negativos, não há solução para o sistema. Caso não existam ciclos negativos, uma solução factível para o sistema é $(x_1 = \delta(v_0, v_1), x_2 = \delta(v_0, v_1), \ldots, x_n = \delta(v_0, v_n))$.

O algoritmo minimiza $\sum_{i=1}^{n} x_i$ e $\max\{x_i\} - \min\{x_i\}$, sujeito às restrições e $x_i \leq 0$.

Mais informações no Cormen, p. 602.

8.3 Kadane 1D

Tempo: $\Theta(n)$

```
int kadane(int a[], int n) {
    int max = 0, s = 0;
    int i;

    for(i = 0; i < n; i++) {
        s += a[i];
        if(s < 0)
            s = 0;
        if(s > max)
            max = s;
```

```
    }

    return max;
}
```

8.4 Kadane 2D

Tempo: $O(n^3)$

```
int kadane2d(int a[N][N], int n) {
    int i, j, k, maxsum = 0;

    for(i = 0; i < n; i++) {
        int p[N];
        for(j = 0; j < n; j++) p[j] = 0;
        for(j = i; j < n; j++) {
            int sum;
            for(k = 0; k < n; k++) p[k] += a[j][k];
            /* p[k] = soma da coluna k, da linha i ate a linha j */
            sum = kadane(p, n);
            if(sum > maxsum) maxsum = sum;
        }
    }

    return maxsum;
}
```

8.5 Majority Problem

Encontra o elemento que aparece em mais da metade de um array em $\Theta(n)$

```
int findmajority(int a[], int n) {
    int x, y, i;

    x = a[0];
    y = 1;

    for(i = 1; i < n; i++) {
        if(y >= 1) {
            if(a[i] == x) y++;
            else y--;
        } else {
            x = a[i];
            y = 1;
        }
    }

    /* verifica se eh majoritario (caso nao haja) */
    int c = 0;
    for(i = 0; i < n; i++)
        if(a[i] == x) c++;

    if(c > n/2) return x; /* tem */
    else return -1; /* nao tem */
}
```

8.6 Stable Marriage

Given n men and n women, where each person has ranked all members of the opposite sex with a unique number between 1 and n in order of preference, marry the men and women off such that there are no two people of opposite sex who would both rather have each other than their current partners. If there are no such people, all the marriages are *stable*.

8.6.1 Gale-Shapley

Tempo: $O(n^2)$

```
int n;
int mpref[N][N], wpref[N][N]; /* preencher com lista de preferencias */
int mmatch[N], wmatch[N]; /* iniciar com -1 */

void smp_solve() {
    while(true) {
        int m = -1, w;
        for(i = 0; i < n && m == -1; i++)
            if(mmatch[i] == -1) m = i;
        if(m == -1) break; /* fim, nao tem nenhum homem nao-casado */

        for(j = 0; j < n; j++) {
            w = mpref[m][j];

            if(wmatch[w] == -1) {
                /* mulher w esta livre, faz (m, w) */
                mmatch[m] = w;
                wmatch[w] = m;
                break;
            } else {
                int ml = wmatch[w];
                int in_m, in_ml;

                /* encontra indices de ml e m */
                for(i = 0; i < n; i++) {
                    if(wpref[w][i] == m) in_m = i;
                    else if(wpref[w][i] == ml) in_ml = i;
                }

                if(in_m < in_ml) {
                    /* desfaz (ml, w) e faz (m, w) */
                    mmatch[ml] = -1;
                    mmatch[m] = w;
                    wmatch[w] = m;
                    break;
                }
            }
        }
    }
}
```

8.7 Números Romanos

```
string fill(char c, int n) {
    string s;
    while(n--) s += c;
    return s;
```

```
}

string toRoman(int n) {
    if(n < 4) return fill('i', n);
    if(n < 6) return fill('i', 5 - n) + "v";
    if(n < 9) return string("v") + fill('i', n - 5);
    if(n < 11) return fill('i', 10 - n) + "x";
    if(n < 40) return fill('x', n / 10) + toRoman(n % 10);
    if(n < 60) return fill('x', 5 - n / 10) + 'l' + toRoman(n % 10);
    if(n < 90) return string("l") + fill('x', n / 10 - 5) + toRoman(n % 10);
    if(n < 110) return fill('x', 10 - n / 10) + "c" + toRoman(n % 10);
    if(n < 400) return fill('c', n / 100) + toRoman(n % 100);
    if(n < 600) return fill('c', 5 - n / 100) + 'd' + toRoman(n % 100);
    if(n < 900) return string("d") + fill('c', n / 100 - 5) + toRoman(n % 100);
    if(n < 1100) return fill('c', 10 - n / 100) + "m" + toRoman(n % 100);
    if(n < 4000) return fill('m', n / 1000) + toRoman(n % 1000);
    return "?";
}
```

8.8 Calendário

```
#define isBissexto(y) ((y % 4 == 0 && y % 100 != 0) || (y % 400 == 0))

int diasAno(int y) {
    return isBissexto(y) ? 366 : 365;
}

int diasMes(int m, int y) {
    switch(m) {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            return 31;
        case 4:
        case 6:
        case 9:
        case 11:
            return 30;
        case 2:
            return isBissexto(y) ? 29 : 28;
    }
    return -1;
}

int getdia(int d, int m, int y) {
    int ano, mes, dia, r;
    r = 0;
    for(ano = 1900; ano < y; ano++) r += diasAno(ano);
    for(mes = 1; mes < m; mes++) r += diasMes(mes, ano);
    r += d-1;
    return r;
}
```

8.9 Josephus Problem

```
int joseph(int n){
    vector< int >  v;
    int dir, died, i, m;
    /* dir = 1  := horario
     * dir = 0  := anti-horario
     * m        := passos para matar o proximo
     * died     := guarda quem vai morrer
     */
    for(i = 0; i < n; i++) v.push_back(i);
    /* Inicializacao:
     * se o sentido for horario
     * died = 1, senao died = 0
     * m = 1 (mata todos em sequencia)
     */
    died = dir;
    m = 1;
    /* mata ate o penultimo */
    for(i = 0; i < (n-1); i++){
        if(dir) died = (died-1+m)%(n-i);
        else {
            m = -m;
            died = (died+m);
            while(died < 0) died += (n-i);
            died%=(n-i);
        }
        m = 0; // atualiza o valor da proxima contagem
        v.erase(v.begin()+died);
    }
    /* retorna o numero do sobrevivente */
    return *v.begin();
}
```

9 Containers STL

10 Formulário

10.1 Somatórios

$$\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}$$

$$\sum_{k=1}^{n} k^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = \left[\frac{n \cdot (n+1)}{2}\right]^2$$

10.2 Geometria

10.2.1 Triângulos

Área, dados lados a, b, c

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

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

Área, dadas medianas u, v, w

$$S = \frac{4}{3}\sqrt{q(q-u)(q-v)(q-w)}$$

$$q = \frac{u+v+w}{2}$$

11 Troubleshooting

11.1 Erros Comuns

1. **Overflow**?
2. **Inicializar** todas as variáveis.
3. Verificar **iteradores** de todos os fors.
4. Verificar **limites de arrays**.
5. Modificando container STL enquanto iterando sobre ele? Cuidado!
6. *Nunca* fazer $strcmp == -1$, sempre $strcmp < 0$!
7. Tem alguma função que era pra retornar algo que não tá retornando nada? Cuidado, C++ não reclama da falta de *return*!

11.1.1 Wrong Answer

1. Overflow?
2. As variáveis estão todas sendo inicializadas/zeradas? O programa dá a mesma resposta para casos repetidos?
3. Array pequeno demais? Programa pode estar escrevendo no endereço de memória de outra variável e fazendo algo bizarro.

11.1.2 Runtime Error

1. Dividiu por zero? Fez módulo por zero?
2. Array out of bounds?
3. Arrays grandes demais? Usando memória demais?
4. Função recursiva que não termina nunca?
5. Stack overflow? Arrays grandes demais? Colocar os arrays gigantes como globais, nunca locais.
6. Alocando memória sem liberar? Memory leak?

11.1.3 Time Limit Exceeded

1. Loop infinito?
2. Evitar recálculo (i/w, i%w)
3. Usando STL desnecessariamente?
4. Descubra qual parte do codigo demora mais.

11.2 Grafos

1. O grafo é dirigido?
2. O grafo pode ser desconexo?
3. O grafo possui arestas paralelas? Arestas repetidas? Arestas (v_i, v_i)?
4. As arestas podem ter pesos negativos?
5. Dá pra otimizar o Dijkstra? Usar heap no braço? Alguma heurística pra transformar em A*?
6. Zerou $deg[]$, $vis[]$, etc?