## Bugs do milênio

Erros teóricos:  
-Não ler o enunciado do problema com calma.  
-Assumir algum fato sobre a solução na pressa.  
-Não reler os limites do problema antes de submeter.  
-Quando adaptar um algoritmo, atentar a todos os detalhes da estrutura do algoritmo, se devem (ou não) ser modificados (ex:marcação de vértices/estados).  
-O problema pode ser NP, disfarçado ou mesmo sem limites especificados. Nesse caso a solução é bronca mesmo, não é hora de tentar ganhar o prêmio nobel.  
  
Erros com valor máximo de variável:  
-Verificar com calma (fazer as contas direito) para ver se o infinito é tão infinito quanto parece.   
-Verificar se operacoes com infinito estouram 31 bits.  
-Usar multiplicacao de int's e estourar 32 bits (por exemplo checar sinais usando a\*b > 0).  
  
Erros de casos extremos:  
-Testou caso n=0? n=1? n=MAXN? Muitas vezes tem que tratar separado.  
-Pense em todos os casos que podem ser considerados casos extremos ou casos isolados.  
-Casos extremos podem atrapalhar não só no algoritmo, mas em coisas como construir alguma estrutura (ex: lista de adj em grafos).  
-Não esquecer de self-loops ou multiarestas em grafos.  
-Em problemas de caminho Euleriano, verificar se o grafo é conexo.

Erros de desatenção em implementação:  
-Errar ctrl-C/ctrl-V em código. Muito comum.  
-Colocar igualdade dentro de if? (if(a = 0) continue;)

-Esquecer de inicializar variável.  
-Trocar break por continue (ou vice-versa)

-Declarar variável global e variável local com mesmo nome (é pedir pra dar merda...)  
  
Erros de implementação:  
-Definir variavel com tipo errado (int por double, int por char).  
-Não usar variável com nome max e min.  
-Não esquecer que .size() é unsigned.  
-Lembrar que 1 é int, ou seja, se fizer *long long a = 1 << 40;*não irá funcionar (o ideal é fazer long long a = 1LL << 40;).  
  
Erros em limites:  
-Qual o ordem do tempo e memória? 10^8 é uma referencia para tempo. Sempre verificar rapidamente a memoria, apesar de que o limite costuma ser bem grande.  
-A constante pode ser muito diminuida, com um algoritmo melhor (ex: húngaro no lugar de fluxo) ou operações mais rápidas (ex: divisões são lentas, bitwise é rápido)?  
-O exercicio é um caso particular que pode (e está precisando) ser otimizado e nao usar direto a biblioteca?  
  
Erros em doubles:  
-Primeiro, evitar (a não ser que seja necessário ou mais simples a solução) usar float/double. E.g. conta que só precisa de 2 casas decimais pode ser feita com inteiro e depois % 100.  
-Sempre usar double, não float (a não ser que o enunciado peça explicitamente);  
-Testar igualdade com tolerância (absoluta, e talvez relativa).  
-Cuidado com erros de imprecisão, em particular evitar ao máximo subtrair dois números praticamente iguais.  
  
Outros erros:  
-Evitar (a não ser que seja necessário) alocação dinâmica de memória.  
-Não usar STL desnecessariamente (ex: vector quando um array normal dá na mesma), mas usar se facilitar (ex: nomes associados a vértices de um grafo - map<string,int>) ou se precisar (ex: um algoritmo nlogn q usa <set> é necessário para passar no tempo).  
-Não inicializar variavel a cada teste (zerou vetores? zerou variavel que soma algo? zerou com zero? era pra zerar com zero ou com -1 ou com INF?).  
-Saída está formatada corretamente?  
-Declarou vetor com tamanho suficiente?  
-Cuidado ao tirar o módulo de número negativo. Ex.: x%n não dá o resultado esperado se x é negativo, fazer (x%n + n)%n.

Referência

2 Operações com bits

3 Combinatória

MDC, MMC, Binomial, permutações

4 Numéricos

Exponenciação, teste de primalidade, inverso mod n

4 Busca Binária

bmin, bmax

4 Referência C++ e STL

6 Ordenação

STL, Count sort, radix sort

6 Bignum

8 Probabilidade / Estatística

Distribuição binomial, hipergeométrica, poisson

9 Matrizes

9 Estruturas de dados caseiras

Linked list, union-find, heap, hash table, interval tree

12 Geométricos

13 Grafos

17 Strings

18 Códigos diversos

19 Anexos

Tabela ASCII, Bugs do milênio, exemplos de grafos

TODO:

- selection (k-ésimo elemento)

- revisar algoritmos em grafos

max flow, min cost flow, min cut, MST, ordenação topológica, pontes e vértices de articulação, componentes biconexas

- algoritmos geométricos

Não esquecer

* Testar o compilador do jugde no warmup
* Casos particulares (N = 0, N = 1, a == b, )
* Inicializar variáveis
* Possibilidades de overflow
* Variáveis com nomes iguais
* using namespace std;
* size() é **unsigned int** -> Cuidado com overflow
* Usar reserve() quando usar vector
* Queue e Stack **NÃO** têm .clear()

Operações com bits

**#define** ON(a, i) a |= 1<<(i)

**#define** OFF(a, i) a &= ~(1<<(i))

**#define** ISON(a, i) (a & (1<<(i)))

// Quando você quer mais que 32 bits

**unsigned int** v[1000];

**#define** ON(v, i) v[i>>5] |= 1<<((i)&31)

**#define** OFF(v, i) v[i>>5] &= ~(1<<((i)&31))

**#define** ISON(v, i) (v[i>>5] & (1<<((i)&31)))

Desligar todos os bits: a = 0;

Ligar todos os bits: a = ~0;

Inverter todos os bits: a = ~a;

Isolar o bit menos significativo: a = (a & -a);

Contagem de “1” ones()

// Calcula quantos “1” tem a

unsigned ones(register int a){

a = (a & 0x55555555) + ((a >> 1) & 0x55555555);

a = (a & 0x33333333) + ((a >> 2) & 0x33333333);

a = (a & 0x0F0F0F0F) + ((a >> 4) & 0x0F0F0F0F);

a = (a & 0x00FF00FF) + ((a >> 8) & 0x00FF00FF);

a = (a & 0x0000FFFF) + ((a >>16) & 0x0000FFFF);

**return** a;

}

Combinatória

Maior Divisor Comum gcd()

// Calcula o gcd(a,b) para **a > b**

unsigned int gcd(int a, int b){

int k = a % b; if (!k) **return** b;

**return** **gcd(b, k)**;

}

Você Sabia? O número de pontos no reticulado sobre a reta que liga (1,1) e (m, n) é gcd(m, n).

**unsigned int** **safe\_gdc**(int a, int b){

**if** (**!**b)

**return** max(a, 1);

**else**

**return** gcd(a, b);

}

**void** **normaliza**(int &a, int &b){

**int** k = safe\_gdc(a, b);

a /= k;

b /= k;

}

void normaliza(int &a, int &b, int &c){

int k = safe\_gdc(safe\_gdc(a, b), c);

a /= k;

b /= k;

c /= k;

}

Binomial com triângulo de Pascal Bipascal()

// **unsigned int** => overflow em (35 15) e (35 16)

// **unsigned long long** => overflow em (63 29)

**#define** NMAX 35

unsigned int tri[NMAX][NMAX];

void calculaPascal() {

for (int i=0; i < NMAX; i++) tri[i][0]=1;

tri[1][1] = 1;

for (int i = 2; i < NMAX; i++)

for (int j = 1; j <= i; j++)

tri[i][j] = tri[i-1][j] + tri[i-1][j-1];

}

int Bi\_pascal (int n, int k) {

return tri[n][k];

}

Binomial recursivo Bi()

// O(min(n,k)) para calculo do Binomial

unsigned int Bi (int n, int k){

**if** (k>n-k) **return** **Bi(n, n-k)**;

**if** (!n || !k) **return** 1;

**return** n\***Bi(n-1, k-1)**/k;

}

Binomial recursivo sem overflow Bi()

// Calculo do Binomial evitando overflow

// needs **gcd()**

unsigned int Bi\_safe (int n, int k){

**unsigned int** m, q;

**if** (k > n-k) **return** **Bi\_safe (n, n-k)**;

**if** (!n || !k) **return** 1;

m = Bi\_safe(n-1, k-1);

q = **gcd(n,k)**; k/=q;

**return** (m / k) \* (n / q);

}

**Quantidade de permutações** perms()

// retorna a quantidade de permutacões de e[n]

// leva em conta repetição de elementos

// elementos podem estar desordenados

// perms([e0, e1, ..., en]) = n \* perms([e1, ..., en]) / freq(e0)

// O (n­2)

unsigned int perms (char \*e, int n){

if (n == 1) return 1;

char x = e[0];

unsigned int contagem = 1;

for (int i = 1; i < n; i++)

if (x == e[i]) contagem++;

return n \* perms(&e[1], n-1) / contagem;

}

Ordem de uma permutação permton()

#include <algorithm>

using namespace std;

// retorna a ordem da permutacao de e[n]

// needs **perms()**

int permton (char \*e, int n){

**if** (n == 1) **return** 1;

**char** c[128];

**for** (int i=0; i<128; i++)

c[i] = 0;

**int** ret = 0;

**for** (**int** i=1; i < n; i++){

**if** (e[0] > e[i] && !c[e[i]]){

c[e[i]] = 1;

troca(e[0], e[i]);

ret += perms(&e[1], n-1);

troca(e[0], e[i]);

}

}

**return** ret + **permton**(&e[1], n-1);

}

k-ésima permutação ntoperm()

#include <algorithm>  
#include <string.h>  
using namespace std;

// retorna a k-esima permutacao dos elementos de e[n]

// e[n] deve vir ordenado, **e[n] é destruído**!!!

// needs **perms()**

string ntoperm (char \*e, int n, unsigned int k){

**if** (n == 1)

**return** string(1, e[0]);

**char** c[128];

**for** (**int** i=0; i<128; i++)

c[i] = 0;

**unsigned int** t = 0, j = perms(&e[1], n-1);

**int** i = 0;

c[e[0]] = 1;

**while**(j < k){

i++;

if(!c[e[i]]){

c[e[i]] = 1;

**troca** (e[0], e[i]);

t = j;

j += **perms(&e[1], n-1)**;

**troca** (e[0], e[i]);

}

}

**for** (;i>0; i--)

**troca** (e[i-1], e[i]);

**return** string(1, e[0]) + **ntoperm(&e[1], n-1, k-t)**;

}

Numéricos

Potência módulo n modpot()

typedef **unsigned long long** ull;

// calcula (b^p mod n) em O(log(p)) multiplicações

**unsigned int modpot (ull b, int p, int n)**{

ull ret;

**for** (ret = 1; p>0; p = (p>>1), b = (b \* b) % n)

**if** (p & 1) ret = (ret \* b) % n;

**return** (**unsigned int**) ret;

}

Teste de primalidade is\_prime()

// Teste de primalidade by Fabinho

// needs **math.h**

**bool is\_prime (int n)**{

**if** (n < 0) **return** **is\_prime(-n)**;

**if** (n < 5 || n % 2 == 0 || n % 3==0)

**return** (n == 2 || n == 3);

**int** maxP = **sqrt(n)** + 2;

**for** (int p = 5; p < maxP; p += 6)

**if** (n % p == 0 || n % (p + 2)==0) **return** **false**;

**return** **true**;

}

Teste de primalidade - Miller-Rabin mr()

typedef unsigned long long ull;

// Subrotina do teste de Miller-Rabin

// needs **modpot()**

**bool mrtest (int a, int s, int d, int n)**{

ull x = modpot(a, d, n);

**if** (x == 1 | x == n-1) **return** **true**;

**for** (; s > 0 ;s--)

**if** ((x = (x \* x) % n) == n-1) **return** **true**;

**return** **false**;

}

// Teste de primalidade de Miller-Rabin, bizu para números gigantes

// Garantido p/ *n* < 4,759,123,141,  *a* = 2, 7, and 61 (**unsigned int**)

// Melhor que o teste normal para n > 100,000

bool mr (int n){

int s, d;

**if** (n < 5 || n % 2 == 0 || n % 3 == 0)

**return** (n == 2 || n == 3);

**for** (s = 0, d = n-1; d % 2 == 0; s++, d /= 2);

**return** **mrtest (2, s, d, n)** &&

(n <= 7 || **mrtest (7, s, d, n)**) &&

(n <= 61 || **mrtest (61, s, d, n)**);

}

If n < 1,373,653 is a both 2 and 3-SPRP, then n is prime.

If n < 25,326,001 is a 2, 3 and 5-SPRP, then n is prime.

If n < 118,670,087,467 is a 2, 3, 5 and 7-SPRP, then either n = 3,215,031,751 or n is prime.

If n < 2,152,302,898,747 is a 2, 3, 5, 7 and 11-SPRP, then n is prime.

If n < 3,474,749,660,383 is a 2, 3, 5, 7, 11 and 13-SPRP, then n is prime.

If n < 341,550,071,728,321 is a 2, 3, 5, 7, 11, 13 and 17-SPRP, then n is prime.

Teste de primalidade – Erastosthenes

// Primalidade com o Crivo de Erastosthenes

// pelo menos 4x mais rápido que o teste de Miller Rabin (excluindo o

// tempo de pré-processamento)

**#define** MAXN 100000000 /\* maximum value of N \*/

**#define** P1 1562501 /\* = ceil(MAXN/64) \*/

**#define** P2 50000000 /\* = ceil(MAXN/2) \*/

**#define** P3 5000 /\* = ceil(ceil(sqrt(MAXN))/2) \*/

**unsigned** sieve[P1];

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

**void** make\_sieve(){

**register** **unsigned** i, j, k;

memset(sieve, 0, **sizeof**(sieve));

**for** (k = 1; k <= P3; k++)

**if** (GET(k)==0) for(j=2\*k+1,i=2\*k\*(k+1);i<P2;i+=j)

sieve[i>>5]|=1<<(i&31);

}

**int** isprime(**int** p) {

**return** p==2 || (p>2 && (p&1)==1 && (GET((p-1)>>1)==0));

}

Inverso módulo m inv()

// calcula o inverso modular de x, módulo m

// mdc(x, m) deve ser 1

**int** **inv**(**int** x, **int** m){

**int** s = m, t = x;

**int** a = 0, b = 1;

**while** (t) {

**int** q = s / t, r = s % t;

s = t, t = r;

**int** temp = (a – b \* q) % m;

a = b, b = temp;

}

**if** (a > 0) **return** a;

**return** a + m;

}

Busca Binária

// f é uma função não-decrescente com domínio [first, last)

// bmin retorna o **maior x** em [first, last) tal que **f(x) == val**.

// caso não exista, retorna o menor x tal que f(x) > val.

**long long bmin(long long first, long long last, int val)**{

**long long** len = last - first;

**while** (len > 0){

**long long** half = len >> 1, middle = first + half;

**if** (**f**(middle) < val){

first = middle+1;

len -= half + 1;

} else {

len = half;

}

}

**return** first;

}

// f é uma função não-decrescente com domínio [first, last)

// bmax retorna o **menor x** em [first, last) tal que **f(x) > val**

// se f(x) <= val, [first, last), então bmax retorna last

**long long bmax(long long first, long long last, int val)**{

**long long** len = last - first;

**while** (len > 0){

**long long** half = len >> 1, middle = first + half;

**if** (val < **f**(middle)) {

len = half;

} else {

first = middle+1;

len -= half + 1;

}

}

**return** first;

}

Propriedade: se a = e b = **então** f(x) = y, em [a, b)

Referência C++ e STL

String

#include <string>

string( size\_type length, const char& ch );

Membros de string

**begin, end, rbegin, rend, clear, empty, size**

**c\_str** returns a standard C character array version of the string

**copy** copies characters from a string into an array

**empty** true if the string has no elements

**end** returns an iterator just past the last element of a string

**erase** removes elements from a string

**find** find characters in the string

**find\_first\_not\_of** find first absence of characters

**find\_first\_of** find first occurrence of characters

**find\_last\_not\_of** find last absence of characters

**find\_last\_of** find last occurrence of characters

**getline** read data from an I/O stream into a string

**insert**  insert characters into a string

**push\_back** add an element to the end of the string

**replace** replace characters in the string

**rfind**  find the last occurrence of a substring

**substr** **return**s a certain substring

**swap** swap the contents of this string with another

size\_type **find**(const string& str, size\_type index );

size\_type **find**(const char\* str, size\_type index );

size\_type **find**(const char\* str, size\_type index, size\_type length );

size\_type **find**(char ch, size\_type index );

returns the **first occurrence of str within the current string**, starting at index, string::npos if nothing is found,

if the length parameter is given, then find() returns the first occurrence of the first length characters of str within the current string, starting at index, string::npos if nothing is found,

or returns the index of the first occurrence ch within the current string, starting at index, string::npos if nothing is found.

size\_type **find\_first\_not\_of**(const string& str, size\_type index = 0 );

size\_type **find\_first\_of**(const string &str, size\_type index = 0 );

size\_type **find\_last\_of**(const string& str, size\_type index = npos );

size\_type **find\_last\_not\_of**(const string& str, size\_type index = npos );

**obs**.: Essas funções retornam **string::npos** caso não for encontrada nenhuma ocorrência

iterator erase( iterator loc );

iterator erase( iterator start, iterator end );

string& erase( size\_type index = 0, size\_type num = npos );

erase removes num characters from the current string, starting at index, and **return**s \*this.

string substr(size\_type index, size\_type num = npos);

The **substr()** function **return**s a substring of the current string, starting at index, and num characters long. If num is omitted, it will default to string::npos, and the substr() function will simply **return** the remainder of the string starting at index.

Stringstream

#include <sstream>

string str;

getline(cin,str);

istringstream stream(str);

while(stream >> s)

cout << s;

Complex

typedef complex<double> point;

point P;

// cria ponto

double *a, b*;

P = point(a, b);

struct MS{

int v;

MS(int x = 0){v = x;}

};

bool operator < (const MS &X, const MS &Y){

**return** X.v < Y.v;

}

Pair

#include <utility>

pair<*tipo1*, *tipo2*> P;

Membros de pair

*tipo1* **first**;

*tipo2* **second**;

List

**list<Elem> c** Creates an empty list without any elements

**list<Elem> c1(c2)** Creates a copy of another list of the same type (all elements are copied)

**list<Elem> c(n)** Creates a list with n elements that are created by the default constructor

**list<Elem> c(n,elem)** Creates a list initialized with n copies of element elem

**list<Elem> c(beg,end)** Creates a list initialized with the elements of the range [beg,end)

**c.~list<Elem>()** Destroys all elements and frees the memory

Membros de list

**begin, end, rbegin, rend, size, empty, clear, swap**

**front** Returns the first element

**back** Returns the last element

**push\_back** Appends a copy of elem at the end

**pop\_back** Removes the last element (does not return it)

**push\_front** Inserts a copy of elem at the beginning

**pop\_front** Removes the first element (does not return it)

**erase (pos)** Removes the element at iterator position pos and returns the position of the next element

**erase (beg,end)** Removes all elements of the range [beg,end) and returns the position of the next element

if (**!**coll.empty()) {

std::cout << coll.back(); *// OK*

}

Vector

#include <vector>

vector<*tipo*> V;

tipo x;

V[50] – Acesso randômico

Membros de vector

**begin, end, rbegin, rend, size, empty, clear, swap**

**reserve** sets the minimum capacity of the vector

**front** returns a reference to the first element of a vector

**back** returns a reference to last element of a vector

**erase** removes elements from a vector

**pop\_back** removes the last element of a vector

**push\_back** add an element to the end of the vector

Deque

#include <queue>

deque<*tipo*> Q;

Q[50] – Acesso randômico

Membros de deque

**begin, end, rbegin, rend, size, empty, clear, swap**

**front** returns a reference to the first element of a dequeue

**back** returns a reference to last element of a dequeue

**erase** removes elements from a dequeue

**pop\_back** removes the last element of a dequeue

**pop\_front** removes the first element of the dequeue

**push\_back** add an element to the end of the dequeue

**push\_front** add an element to the front of the dequeue

Queue

#include <queue>

queue<*tipo*> Q;

Membros de queue

**back**  returns a reference to last element of a queue

**empty**  true if the queue has no elements

**front** returns a reference to the first element of a queue

**pop** removes the first element of a queue

**push**  adds an element to the end of the queue

**size** returns the number of items in the queue

Stack

#include <stack>

stack<tipo> P;

Membros de stack

**empty** true if the stack has no elements

**pop** removes the top element of a stack

**push** adds an element to the top of the stack

**size** returns the number of items in the stack

**top** returns the top element of the stack

Map

find leva O(log(n))

#include <**map**>

#include <**string**>

map<**string**, **int**> *si*;

si[“Fofao”] = 0;

**if** (si.find("Fofao") == si.end())

cout << "Fofao nao tem inteiro";

**else**

cout << "Fofao tem int igual a " << si["Fofão”];

si.erase(si.find("Fofao"));

si.erase("Fofao");

Membros de map

**begin, end, rbegin, rend, size, empty, clear, swap**

**erase** removes elements from a map

**find** returns an iterator to specific elements

**lower\_bound** returns an iterator to the first element greater than or equal to a certain value

**upper\_bound** returns an iterator to the first element greater than a certain value

Map é um set de pair, ao iterar pelos elementos de map, i->first é a chave e i->second é o valor.

**// map com comparador personalizado**

#include <map>

#include <string.h>

using namespace std;

struct word {

char s[10];

} buf;

struct cmp {

bool operator()(const word &a, const word &b){

return strcmp(a.s, b.s) < 0;

}

};

map<word, int, cmp> M;

typedef map<word, int, cmp>::iterator M\_it;

int main(){

while(scanf("%s", buf.s) == 1)

++M[buf];

for (M\_it i = M.begin(); i != M.end(); ++i)

printf("%s: %d\n", i->first.s, i->second);

return 0;

}

Set

#include <set>  
set<*tipo*> S;

Membros de set

**begin, end, rbegin, rend, size, empty, clear, swap**

**erase** removes elements from a set

**find** returns an iterator to spec**if**ic elements

**insert** insert items into a set

**lower\_bound** returns an iterator to the first element greater than or equal to a certain value

**upper\_bound** returns an iterator to the first element greater than a certain value

**// Criando set com comparador personalizado**

// importante: set considera que um elemento é igual a outro se ele não é nem menor e nem maior.

**int** d[100];

**class** cmp {

public:

**int operator()(int x, int y)**{

**if**(d[x] != d[y])

**return** d[x] < d[y];

**return** x < y;

}

};

**set**<**int**, **cmp**> S;

Lista de prioridade com set

Existem duas maneiras de criar uma lista de prioridade com set. Você pode criar usando pair ou uma função de comparação personalizada.

**Lista de prioridade com pair**

typedef pair<int,int> ii;

set<ii> Q;

**// Adicionando o valor x com prioridade p**

Q.insert(ii(p, x));

**// Pegando o menor elemento**

ii top = \*Q.begin();

x = top->second;

p = top->first;

**// Removendo o menor elemento**

Q.erase(Q.begin());

**// Atualizando o valor de um elemento**

Q.erase(ii(p, x));

Q.insert(ii(pnovo, x));

Hash map (Hash set)

#include "stl\_hash.h"

using std::hash\_map;

#include <hash\_map>

using std::hash\_map;

#include <ext/hash\_map>

using \_\_gnu\_cxx::hash\_map;

#include <algorithm>

int a = max(5,13);//retorna o maior valor (templatizado)

int a = min(5,13);//retorna o menor valor

vector <int> v;

sort(v.begin(), v.end());//ordena de acordo com <

int vv[1234];

stable\_sort(&v[0], &v[1234]);//ordena mantendo a ordem de elementos iguais

reverse(v.begin(), v.end()); //inverte a ordem

//poe v como a proxima permutacao, retorna false se não existir

next\_permutation(v.begin(), v.end());

prev\_permutation(v.begin(), v.end());

Ordenação

sort() da STL

#include <algorithm>

**void** sort( iterator start, iterator end );

**void** sort( iterator start, iterator end, StrictWeakOrdering cmp );

* **sort com comparador customizado**

// cmp: ordenação em ordem **crescente**

**bool** cmp( **int** a, **int** b ) {

**return** a **<** b;

}

**sort** (V.begin(), V.end(), cmp);

obs.: também tem o stable\_sort, que ordena mantendo a ordem relativa dos elementos não-diferentes.

// sort com array

**int** in[MAXN], n;

**sort** (&in[0], &in[n]);

Counting sort

**#define** MAXVALUE 10000000

**#define** MAXN 1000000

// O(n + maxValue)

int B[MAXN], C[MAXVALUE];

**void** countSort(int \*A, int n, int maxValue){

for(int i = 0; i <= maxValue; i++)

C[i] = 0;

for(int i = 0; i < n; i++)

C[A[i]]++;

for(int i = 1; i <= maxValue; i++)

C[i] += C[i - 1];

for(int i = n - 1; i >= 0; i--){

B[--C[A[i]]] = A[i];

// B[--C[A[i]]] = i;

// para obter um vetor com os indices ordenados

}

}

Radix sort

Sort d-digits numbers:

Radix-sort (A, d)

1. **for** 1 🡨 1 **to** d:
2. **do** use a stable sort to sort array A on digit i

Counting sort will do.

* bizu para n grande (mais bizu que STL para n > 500)

// Radix sort de 8bits

#include <algorithm>

**#define** **CTE** 0x80000000 // para lidar com números negativos

int temp[MAXN];

int bucket[256];

**void** **radixSort**(**int** \*A, **int** n){

**int** \*prev = temp, \*next = B;

**for** (**int** i = 0; i < n; i++)

prev[i] = A[i] ^ **CTE**;

**for** (**unsigned int** bit = **255**, s = 0; bit; bit <<= **8**, s += **8**,

swap(prev, next)){

**for** (**int** i = 0; i < **256**; i++)

bucket[i] = 0;

**for** (**int** i = 0; i < n; i++)

bucket[(prev[i] & bit) >> s]++;

**for** (**int** i = 1; i < **256**; i++){

**if** (bucket[i] == n) **goto** brasil;

bucket[i] += bucket[i - 1];

}

**for** (**int** i = n - 1; i >= 0; i--)

next[--bucket[(prev[i] & bit) >> s]] = prev[i];

**brasil**: continue;

}

**for** (**int** i = 0; i < n; i++)

B[i] = prev[i] ^ **CTE**;

}

BigNum

**Regra do envelope:**

3 dígitos decimais = 10 dígitos binários (10^3 ≈ 2^10)

301 dígitos decimais = 100 dígitos binários (log­­10(2) ≈ 0,301)

#include <stdlib.h>

#include <algorithm>

#include <stdio.h>

#include <string.h>

using namespace std;

**#define** MAXN 100 // nao esquecer deste valor

//int digs = 4; // quantidade de digitos na base 256

//int bits = 32; // **8\*digs**, numero de bits do numero

**#define** digs 13

**#define** bits 104

**#define** ISON(a, i) (a & (1 << (i)))

**#define** ON(a, i) a |= 1 << (i)

typedef **unsigned char** uchar;

// divide um valor bcd por 2 e retorna o resto.

int **bcddiv2**(**unsigned char** \*d, **int** n){

**unsigned char** a = 0, b;

**for** (**int** i = 0; i < n; i++){

b = d[i] & 1;

d[i] >>= 1;

**if** (a) d[i] += 5;

a = b;

}

**return** a;

}

// multiplica um valor bcd por 2 e soma 0 ou 1.

// 2\*d + a, a = 0,1

**void** **bcdmul2**(**unsigned char** \*d, **int** n, **char** a){

**char** b;

**for** (**int** i = n-1; i >= 0; i--){

d[i] <<= 1;

**if** (a) d[i]++;

**if** (d[i] >= 10){

a = 1;

d[i] -= 10;

} **else** a = 0;

}

}

**struct** num{

**unsigned char** d[MAXN];

num(){ memset(d, 0, sizeof(d)); }

// s = numero na base 10 em ascii, needs **bcddiv2**()

**num** (**char** \*s){

**int** n = strlen(s);

**bool** neg = **false**;

**if** (s[0] == '-'){

s++; n--;

neg = **true**;

}

**unsigned char** bcd[n];

memset(d, 0, sizeof(d));

**for** (**int** i = 0; i < n; i++)

bcd[i] = s[i] - '0';

**for** (**int** i = bits-1; i >= 0; i--)

**if** (bcddiv2(bcd, n))

ON(d[i>>3], 7 - i&7);

**if** (neg) \*this = -\*this;

}

**num** (**int** x){

**bool** neg = **false**;

memset(d, 0, sizeof(d));

**if** (x < 0) {neg = **true**; x = -x;}

d[digs-1] = x & 0xff; x >>= 8;

d[digs-2] = x & 0xff; x >>= 8;

d[digs-3] = x & 0xff; x >>= 8;

d[digs-4] = x & 0xff;

**if** (neg) \*this = -\*this;

}

**bool** isneg(){ **return** d[0] & 128 != 0; }

**num** **operator** **-**(); // operador -this, complemento de 2 + 1

**void** binprint(); // imprime o numero na forma binaria

**void** uprint(); // unsigned print

**void** print(); // imprime o numero com sinal, needs **operator -**

// Operacoes de soma

**num operator** +=(**num** x);

**num operator** + (**num** x) {num ret = \*this; ret += x; **return** ret;}

**num operator** -= (**num** x) {**return** \*this += (-x);}

**num operator** - (**num** x) {**return** \*this + (-x);}

**num operator** ++(); // ++THIS

**num operator** --(); // --THIS

**char** div2(); // divide o numero por 2

**void** mul2(char a); // multiplica por 2 e soma 0 ou 1

**void** mul256(**unsigned char** a); // multiplica por 256 e soma a

**num operator** \*(num x); // multiplicacao, precisa de "**mul2**"

**num operator** \*=(num x){\*this = \*this \* x; **return** \*this; }

**num** udiv(**num** di, **num** &r); // divisao sem sinal (**mul2, uless**)

**num** div(**num** di, **num** &r); // com sinal (**isneg, ++, --, udiv**)

**num operator** /(**num** di) { num r; **return** div(di, r); }

**num operator** %(**num** di){ num r; div(di, r); **return** r; }

**bool** uless(num &x); // comparacao sem sinal, retorna this >= x

**bool operator** < (**num** x);

**bool operator** >= (**num** x){ **return** **!**operator < (x); }

**bool operator** > (**num** x){ **return** x < \*this; }

**bool operator** <= (**num** x){ **return** **!**(x < \*this); }

**bool operator** == (**num** x);

};

// divide o numero por 2

**char num::div2()**{

**unsigned char** a = 0, b;

for(char i = 0; i < digs; i++){

b = d[i] & 1;

d[i] = (d[i] >> 1) | (a << 7);

a = b;

}

**return** a;

}

// multiplica por 2, somando a (0 ou 1)

**void num::mul2(char a)**{

char b = 0;

**if**(a) a = 1;

**for**(char i = digs-1; i >= 0; i--){

b = d[i] >> 7;

d[i] = (d[i] << 1) + a;

a = b;

}

}

// multiplica por 256 e soma a

**void num::mul256(unsigned char a)**{

**for**(char i = 0; i < digs - 1; i++)

d[i] = d[i+1];

d[digs-1] = a;

}

// Operador ++this, incremento

**num num::operator ++()**{

**register char** c = 1;

for(int i = digs-1; c && i >= 0; i--){

if(c && d[i] == 255)

d[i] = 0;

**else**{

d[i]++;

break;

}

}

**return** \*this;

}

//operador --this, decrementa o numero

**num num::operator --()**{

**register char** c = 1;

**for**(int i = digs - 1; i >= 0; i--){

**if**(c && d[i] == 0)

d[i] = 255;

**else**{

d[i]--;

**break**;

}

}

**return** \*this;

}

// operador -this, retorna o complemento de 2 + 1

**num num::operator -()**{

**unsigned char** c = 1;

num ret = num();

**for**(int i = digs-1; i >= 0; i--){

ret.d[i] = ~d[i] + c;

**if**(c) c = !ret.d[i];

}

**return** ret;

}

**num num::operator +=(num x)**{

**register short c** = 0, r;

for(int i = digs - 1; i >= 0; i--){

r = d[i] + x.d[i] + c;

if(r >= 256){

d[i] += x.d[i] + c; c = 1;

}else{

d[i] += x.d[i] + c; c = 0;

}

}

// overflow se (ambos numeros sao positivos e c = 1)

// ou (se ambos sao negativos e c = 0)

**return** \*this;

}

// multiplica por x, precisa de "+=" e "mul2"

num num::operator \*(num x){

num ret = num();

for(int i = digs - 1; i >= 0; i--){

**unsigned char** a = d[i];

for(int j = 0; j < 8; j++){

if(a & 1)

ret += x;

a >>= 1;

x.mul2(0);

}

}

**return** ret;

}

//////////////////////////////////////

// unsigned division, needs mul2, uless

**num num::udiv(num di, num &r)**{

num q = num();

r = num();

**for**(int i = 0; i < digs; i++){

**unsigned char** x = d[i];

**for**(int j = 7; j >= 0; j--){

r.mul2(x & 128);

x <<= 1;

**if**(r.uless(di)){

q.mul2(1);

r -= di;

} **else**

q.mul2(0);

}

}

**return** q;

}

// signed division

**num num::div(num di, num &r)**{

**if**(isneg()){

num x = -(\*this);

num q = x.div(-di, r);

if (di.isneg()){

r = -(di + r);

**return** ++q;

} **else** {

r = di - r;

**return** --q;

}

}

**if** (di.isneg()){

num q = udiv(-di, r);

**return** -q;

}

**return** udiv(di, r);

}

////////////////////////////////////////////////////

// imprime o numero na forma binaria

**void** **num::binprint()**{

for(int i=0; i<bits; i++)

putchar(ISON(d[i>>3], 7 - i&7) ? '1' : '0');

}

// unsigned print

**void** **num::uprint()**{

int n = 3 + bits/3;

**unsigned char** bcd[n];

memset(bcd, 0, sizeof(bcd));

**for**(**int** i = 0; i < bits; i++)

bcdmul2(bcd, n, ISON(d[i>>3], 7 - i&7));

int i = 0;

**while**(**!**bcd[i++] && i < n);

**for**(i--; i < n; i++)

putchar('0' + bcd[i]);

}

// imprime o numero com sinal, needs operator -

**void** **num::print()**{

**if**(isneg()){

putchar('-');

num z = -(\*this);

z.uprint();

} **else**

uprint();

}

////////////////////////////////////////////////////

// comparacao sem sinal, retorna this >= x

**bool num::uless(num &x)**{

**for**(int i = 0; i < digs; i++){

**if**(d[i] < x.d[i])

**return** **false**;

**else** if(d[i] > x.d[i])

**return** **true**;

}

**return** **true**;

}

// needs uless

**bool num::operator < (num x)**{

**if**(isneg() != x.isneg())

**return** isneg();

**else** if(isneg())

**return** x.uless(\*this);

**else** **return** !uless(x);

}

**bool num::operator ==(num x)**{

**for**(int i=0; i<digs; i++)

**if**(d[i] != x.d[i]) **return** **false**;

**return** **true**;

}

Probabilidade

Distribuição binomial

In general, if the random variable *K* follows the binomial distribution with parameters *n* and *p*, we write *K* ~ B(*n*, *p*). The probability of getting exactly *k* successes in *n* trials is given by:

 \Pr(K = k) = f(k;n,p) 

 \Pr(K = k) = {n\choose k}p^k(1-p)^{n-k}

for *k* = 0, 1, 2, ..., *n* and where

{n\choose k}=\frac{n!}{k!(n-k)!}

(n+1)p-1 < m \leq (n+1)p.\,

As a function of *k*, the expression *ƒ*(*k*; *n*, *p*) **is monotone increasing for *k* < *m*** and **monotone decreasing for *k*> *m***, with the **exception** of one **case where (*n* + 1)*p* is an integer**. In this case, there are two maximum values for *m* = (*n* + 1)*p* and *m* − 1. *m* is known as the *most probable* (*most likely*) outcome of Bernoulli trials. Note that the probability of it occurring can be fairly small.

#include <math.h>

**double** **logfat**[1000000];

// calcula valores da distribuição binomial

// aproxima as contas com o log dos fatoriais.

**double** bindist(**int** n, **int** k, **double** p){

**double** resp;

// (n escolhe k) \* pk \* (1 - p)(n - k)

resp = logfat[n] - logfat[k] - logfat[n-k] + k \* log(p) + (n - k) \* log(1.0 - p);

**return** exp(resp);

}

int main(){

**logfat**[0] = **logfat**[1] = 0.0;

**for**(**int** i=2; i <= **1000000**; i++)

**logfat**[i] = **logfat**[i-1] + **log**(i);  
}

Distribuição multinomial

Distribuição multinomial:

 \begin{align}
f(x_1,\ldots,x_k;n,p_1,\ldots,p_k) & {} = \Pr(X_1 = x_1\mbox{ and }\dots\mbox{ and }X_k = x_k) \\  \\
& {} = \begin{cases} { \displaystyle {n! \over x_1!\cdots x_k!}p_1^{x_1}\cdots p_k^{x_k}}, \quad &
\mbox{when } \sum_{i=1}^k x_i=n \\  \\
0 & \mbox{otherwise,} \end{cases}
\end{align}

for non-negative integers *x*1, ..., *xk*.

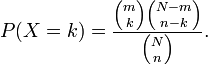
p_1, \ldots p_k event probabilities (Σpi = 1)

Distribuição Hypergeometrica

The hypergeometric distribution is a discrete [probability distribution](http://en.wikipedia.org/wiki/Probability_distribution) that describes the number of successes in a sequence of **n draws** from a finite [population](http://en.wikipedia.org/wiki/Population) **without** **replacement**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **drawn** | **not** **drawn** | **total** |
| **successes** | **k** | m − k | **m** |
| **failures** | **n − k** | N + k − n − m | N − m |
| total | **n** | N − n | **N** |

A [random variable](http://en.wikipedia.org/wiki/Random_variable) X follows the hypergeometric distribution with parameters N, m and n if the probability is given by



\frac{\prod_{i=1}^{c} \binom{m_i}{k_i}}{\binom{N}{n}} N = \sum_{i=1}^c m_i Distribuição hipergeométrica multivariada.

Distribuição de Poisson

If the expected number of occurrences in an interval is λ, then the probability that there are exactly k occurrences (k being a non-negative [integer](http://en.wikipedia.org/wiki/Integer), k = 0, 1, 2, ...) is equal to

f(k; \lambda)=\frac{\lambda^k e^{-\lambda}}{k!},\,\!

Matrizes

#include <string.h>

using namespace std;

**#define** MS 100

**#define** **fu**(i, n) **for**(**int** i=0; i<n; i++)

**struct mat**{

int m[**MS**][**MS**];

**void** **clear**(){

**memset**(m, 0, sizeof(m));

}

};

**mat operator+(mat &a, mat &b)**{

**mat** ret;

**fu**(i, **MS**) **fu**(j, **MS**) ret.m[i][j] = a.m[i][j] + b.m[i][j];

**return** ret;

}

**mat operator-(mat &a, mat &b)**{

mat ret;

**fu**(i, **MS**) **fu**(j, **MS**) ret.m[i][j] = a.m[i][j] - b.m[i][j];

**return** ret;

}

// needs +

**mat operator\*(mat &a, mat &b)**{

**mat** ret;

ret.clear();

**fu**(i, **MS**) **fu**(j, **MS**) **fu**(k, **MS**)

ret.m[i][j] = (ret.m[i][j] + a.m[i][k] \* b.m[k][j]);

**return** ret;

}

// needs + and \*

**mat pow**(**mat** a, **int** n){

**mat** ret;

ret.clear();

**fu**(i, **MS**) ret.m[i][i] = 1;

**while** (n > 0){

**if** (n & 1) ret = ret \* a;

a = a \* a;

n >>= 1;

}

**return** ret;

}

Estruturas de dados caseiras

Linked list simples

#define NMAX 25000000

struct **node**{

struct node \*next;

int value;

};

// alocação estática de nós.

**node** nodes[NMAX+10];

**int** nalloc = 0;

**node** \*newNode(){

**return** &nodes[nalloc++];

}

// adiciona nó na frente da lista.

**void** addList(**node** \*&head, **int** value){

**node** \*i = newNode();

i->next = head;

i->value = value;

head = i;

}

// iterando numa lista encadeada, NULL = lista vazia

**for** (**node** \***i** = head; **i** **!=** **NULL**; **i** = **i**->next){

*… i->value …*

}

Union-find

**#define** MAX 1000  
int p[MAX], rank[MAX];  
  
// **Inicializa um set x  
void** make\_set(int x) {  
   p[x] = x;  
  rank[x] = 0;  
}  
  
**void** link(int x,int y) {  
   if (rank[x] > rank[y])  
     p[y] = x;  
  else {  
      p[x] = y;  
      if (rank[x] == rank[y])  
        rank[y] = rank[y] + 1;  
  }  
}  
  
**// Find the set ID of item x**  
int find\_set(int x) {  
  if (x != p[x])  
     p[x] = find\_set(p[x]);  
   **return** p[x];  
}  
  
**// Union two set containing item x and item y**  
**void** union\_set(int x,int y) {  
  link (find\_set(x), find\_set(y));  
}

Heap

// HEAP TRADICIONAL, é 2x mais rápido que heaps com set, além de suportar atualização

**#define** **troca**(a, b) {int \_x; \_x=a; a=b; b=\_x;}

**#define** **MAXN** 10001

**int** dist[MAXN];

**int** heap[MAXN], heapsize=0;

**int** inv[MAXN]; // retorna a posicao de um vertice no heap

// esse heap da mais prioridade para as menores distancias

**int** comp(int a, int b){

**return** dist[heap[a]] - dist[heap[b]];

}

// faz um elemento ir na direcao das folhas

**void** sifup(int n){

**int** k = 2\*n;

**if** (k > heapsize) **return**;

**if** (k < heapsize && comp(k, k+1) > 0) k++;

**if** (comp(n, k) > 0){

troca (heap[n], heap[k]);

inv[ heap[n] ] = n;

inv[ heap[k] ] = k;

sifup(k);

}

}

// faz um elemento ir na direcao da raiz

**void** sifdown(**int** n){

int k = n/2;

**if** (!k) **return**;

**if** (comp(n, k) < 0){

troca (heap[n], heap[k]);

inv[ heap[n] ] = n;

inv[ heap[k] ] = k;

sifdown(k);

}

}

**void** addHeap(int a){

heap[++heapsize] = a;

inv[a] = heapsize;

sifdown(heapsize);

}

**int** getHeap(){

int k;

**if** (heapsize <= 0) **return** -1;

k = heap[1];

heap[1] = heap[heapsize--];

inv[heap[1]] = 1;

sifup(1);

**return** k;

}

**void** resetHeap(){

heapsize = 0;

}

//////////////////////

// “Heap” com set e comparador customizado

#include <set>

**class** cmp {

public:

**int** operator()(int x, int y){

**if** (dist[x] != dist[y])

**return** dist[x] < dist[y];

**return** x < y;

}

};

**set**<int, cmp> heap;

//////////////////////

// “Heap” com pair

#include <set>

typedef **pair**<**int**, **int**> ii;

**set**<ii> heap;

Hash Table (com list da STL)

#include <list>

#define **SIZE** 1000003

list<int> M[**SIZE**];

char word[NMAX][17];

**void** resetHashTable(){

**for** (**int** i = 0; i < **SIZE**; i++)

M[i].**clear**();

}

**int** H(**const** **char** \*s) {

**unsigned** **int** hash = 0;

**for** (**int** i = 0; s[i]; i++) {

hash = (hash **<<** 6) **+** (hash **<<** 16) **-** hash **+** s[i];

}

**return** hash % **SIZE**;

}

------🡪 Rotating hash (bem rápido)

**int** H(**const** **char** \*s){

**unsigned** **int** hash, i;

**for** (hash = 0x7A3D7F9, i=0; s[i]; ++i)

hash = (hash<<4) ^ (hash>>28) ^ s[i];

return hash % **SIZE**;

}

// se usar a lista encadeada caseira fica 2x mais rápido.

**int** find(**const** **char** \*s){

**int** key = H(s);

**list**<**int**> b = M[key];

**for** (list<**int**>::iterator it = b.begin(); it != b.end(); ++it){

**if** (!strcmp(word[\*it], s))

**return** \*it;

}

return -1;

}

**void** add(**int** i){

**int** key = H(word[i]);

M[key].push\_front(i);

}

Rolling Hash (*Rabin-Karp*)

// requer linked list simples

*char text[NMAX];*

node \*M[HSIZE];

#define CTE 29943829

unsigned int H(const char \*s, int len) {

unsigned int hash = 0;

for (int i = 0; i < len; i++) {

hash = CTE \* hash + s[i];

}

return hash;

}

int find (unsigned int key, const char \*s){

key %= CTE;

for (node \*i = M[key]; i != NULL; i = i->next){

*if (!strncmp(&text[i->value], s, n))*

return i->value;

}

return -1;

}

void add(int i, unsigned int key){

key %= CTE;

addList(M[key], i);

}

void resetHashTable(){

nalloc = 0; // linked list simples

for (int i=0; i<HSIZE; i++)

M[i] = NULL;

}

unsigned int fix = 1;

for (int i = 0; i < n; i++)

fix \*= CTE;

unsigned int fkey = H(text, n);

add (0, fkey);

**for** (**int** i=n; text[i]; i++){

// remove text[i-n] e adiciona text[i];

fkey = **CTE** \* fkey - fix \* text[i-n] + text[i];

***int*** *v = find(fkey, &text[i-n+1]);*

*if(v == -1){*

*add(i, fkey);*

*………………*

*} else …*

}

Árvore de Segmentos

// Use somente para intervalos de **1** até **maxi** ≤ **MAXN**

// **maxi** é definido em tempo de execução

// **MAXN** é definido em tempo de compilação

Obs.: (**idx** & -**idx**) isola o bit (1) menos significativo de **idx**

**#define** MAXN 4000

int tree[MAXN], maxi;

// retorna a frequencia acumulada de **1** até **i**

**int** read (**int** i){

**int** ret = 0;

**while**(i > 0){

ret += tree[i];

i -= (i & -i);

}

**return** ret;

}

// adiciona **v** na posição **i**

**void** add (**int** i, **int** v){

**while** (i <= maxi){

tree[i] += v;

i += (i & -i);

}

}

// retorna o valor em **i**

**int** readS (**int** i){

**int** sum = tree[i];

**if** (i > 0){

**int** z = i - (i & -i);

i--;

**while** (i != z){

sum -= tree[i];

i -= (i & -i);

}

}

**return** sum;

}

--------------------------------------

#include <stdio.h>

#include <algorithm>

#include <limits.h>

**using** **namespace** **std**;

**struct** intervalo{

**int** ini, fim, v; // Representa o intervalo [ini, fim]

struct **intervalo** \*esquerda, \*direita;

***int*** *menor; // para findmenor, findmaior, findmin e findmax*

***int*** *maior; // para findmenor, findmaior, findmin e findmax*

// Adiciona dif ao intervalo [a, b], O(H)

**void** add(**int** a, **int** b, **int** dif);

// Retorna o valor em x, O(H)

**int** valor(int x);

// Retorna soma do intervalo [a, b], O(H)

**int** soma(int a, int b);

// Retorna o menor x tal que valor(x) == k, O(N)

**int** findmenor(int k);

// Retorna o maior x tal que valor(x) == k

**int** findmaior(int k);

// Retorna o menor valor(x) no intervalo [a, b], O(N)

**int** findmin(int a, int b);

// Retorna o maior valor(x) no intervalo [a, b], O(N)

**int** findmax(int a, int b);

};

#define **MAXNODE** 1000000

**intervalo** intervalos[**MAXNODE**];

**int** intcount = 0;

**void** resetIntervalos(){

**int** count = 0;

}

**intervalo** \*newIntervalo(**int** ini, **int** fim){

**intervalo** \*ret = &intervalos[intcount++];

ret->ini = ini;

ret->fim = fim;

ret->v = 0;

ret->esquerda = ret->direita = NULL;

*// para findmenor, findmaior, findmin e findmax*

*ret->menor = 0;*

*ret->maior = 0;*

***return*** *ret; // lazy mode*

// Inicializacao O(n): eager mode

**if** (ini < fim){

int mid = (ini + fim)/2;

ret->esquerda = newIntervalo(ini, mid);

ret->direita = newIntervalo(mid+1, fim);

} **else** {

ret->esquerda = ret->direita = 0;

}

**return** ret;

}

**void** intervalo::add(int a, int b, int dif){

**if** (b < ini || a > fim) return;

**if** (a < ini) a = ini;

**if** (b > fim) b = fim;

**if** (a == ini && b == fim){

v += dif;

menor += dif;

maior += dif;

} **else** {

*// essas duas linhas soh sao necessarias no lazy mode*

*if(!esquerda) esquerda = newIntervalo(ini, (ini + fim)/2);*

*if(!direita) direita = newIntervalo((ini + fim)/2 + 1, fim);*

esquerda->add(a, b, dif);

direita->add(a, b, dif);

menor = **min**(esquerda->menor, direita->menor) + v;

maior = **max**(esquerda->maior, direita->maior) + v;

}

}

**int** intervalo::valor(**int** x){

**if** (x < ini || x > fim) **return** 0;

**int** ret = 0;

**if** (ini <= x && x <= fim)

ret += v;

**if** (ini < fim){

*// if soh eh necessario no lazy mode*

*if(esquerda)* ret += esquerda->valor(x);

*if(direita)* ret += direita->valor(x);

}

**return** ret;

}

**int** intervalo::soma(**int** a, **int** b){

**if** (b < ini || a > fim) **return** 0;

**if** (a < ini) a = ini;

**if** (b > fim) b = fim;

**int** ret = (b - a + 1) \* v;

**if** (ini < fim){

*// if soh eh necessario no lazy mode*

*if(esquerda)* ret += esquerda->soma(a, b);

*if(direita)* ret += direita->soma(a, b);

}

**return** ret;

}

**int** intervalo::findmenor(**int** k){

**if** (ini == fim) **return** k == v ? ini : -1;

**if** (menor > k || maior < k) **return** -1; // nao esta aqui

**if** (menor == maior) **return** ini;

**int** ret = -1;

*// if soh eh necessario no lazy mode*

*if (esquerda)* ret = esquerda->findmenor(k - v);

**if** (ret == -1)

*if (direita)* ret = direita->findmenor(k - v);

**return** ret;

}

**int** intervalo::findmaior(**int** k){

**if** (ini == fim) return k == v ? ini : -1;

**if** (menor > k || maior < k) **return** -1; // nao esta aqui

**if** (menor == maior) **return** fim;

**int** ret = -1;

*// if soh eh necessario no lazy mode*

*if (direita)* ret = direita->findmaior(k - v);

**if** (ret == -1)

*if (esquerda)* ret = esquerda->findmaior(k - v);

**return** ret;

}

**int** intervalo::findmin(**int** a, **int** b){

**if** (a > fim || b < ini) **return** **INT**\_**MAX**/2;

**if** (a < ini) a = ini;

**if** (b > fim) b = fim;

**if** (ini == a && b == fim)

**return** menor;

*// modo lazy*

***int*** *ret = maior;*

***if*** *(esquerda) ret =* ***min****(ret, v + esquerda->findmin(a, b));*

***if*** *(direita) ret =* ***min****(ret, v + direita->findmin(a, b));*

***return*** *ret;*

// modo eager

**return** v + **min**(esquerda->findmin(a, b), direita->findmin(a, b));

}

**int** intervalo::findmax(**int** a, **int** b){

**if** (a > fim || b < ini) **return** -**INT**\_**MAX**/2;

**if** (a < ini) a = ini;

**if** (b > fim) b = fim;

**if** (ini == a && b == fim)

**return** maior;

*// modo lazy*

*int ret = menor;*

***if*** *(esquerda) ret = max(ret, v + esquerda->findmax(a, b));*

***if*** *(direita) ret = max(ret, v + direita->findmax(a, b));*

***return*** *ret;*

// modo eager

**return** v + max(esquerda->findmax(a, b), direita->findmax(a, b));

}

Algoritmos Geométricos

Usar bizuário geométrico do bart...

#include <complex>  
#include <algorithm>

#include <vector>  
#include <queue>

using namespace std;

**#define** fu(i, n) for(int i=0; i<n; i++)

typedef complex<double> point;

typedef vector<point> poly;

Calcular reta get\_reta()

**typedef** struct{ int x, y; } **ponto**;

**typedef** struct{ int a, b, c; } **reta**;

// calcula coeficientes (a, b, c) da reta definida pelos pontos p e q

// needs normaliza();z

**void** get\_reta(**ponto** &p, **ponto** &q, **reta** &r){

r.a = p.y - q.y;

r.b = q.x - p.x;

r.c = p.y \* q.x - q.y \* p.x;

**normaliza**(r.a, r.b, r.c);

}

Calcular area area()

// CALCULA ÁREA

double area(poly &p){

double ret=0;

for (int i = 0; i < p.size(); ++i)

ret += imag(p[i]\*conj(p[(i+1)%p.size()]));

**return** abs(ret)\*0.5;

}

Testa ponto em segmento onSegment()

// Verifica se p está no segmento definido pelos

// pontos a e b.

bool onSegment(point p, point a, point b){

if(a == b) **return** p==a;

point x=b-a; p \*= conj(x);

**return** p.imag()==0 && 0 <= p.real() && p.real()<=norm(x);

}

Distância entre segmentos distSeg()

// Calcula distancia entre os segmentos ab e cd

// Use somente com complex<double>

**#define** alt(m, n) **if**(0<m.real() && m.real()<n) \

ret = min(ret,\abs(m.imag())/sqrt(n));

double distSeg(point a, point b, point c, point d){

point x=conj(b-a), y=conj(d-c);

point va=(a-c)\*y, vb=(b-c)\*y, vc=(c-a)\*x, vd=(d-a)\*x;

**if**(va.imag()\*vb.imag()<0 && vc.imag()\*vd.imag()<0) **return** 0.0;

double ret=min(min(abs(a-c), abs(a-d)), min(abs(b-c), abs(b-d)));

double nx=norm(x), ny=norm(y);

alt(va, ny); alt(vb, ny); alt(vc, nx); alt(vd, nx);

**return** ret;

}

Testa ponto em poly pointInsidePolygon()

// Ponto dentro do poly? Usa winding number

// 1=dentro, 0=em cima, -1=fora

int pointInsidePolygon(point2 p, const poly &P){

int wn = 0;

for(int j=0, i=P.size()-1; j<P.size(); i=j++){

**if**(onSegment(p, P[i], P[j])) **return** 0;

point2 a = P[i], b = P[j];

**if**(a.imag() < b.imag()) swap(a, b);

**if**(b.imag()<=p.imag() && p.imag()<a.imag() &&

((p-b)\*conj(a-b)).imag()>0)

**if**(a==P[i]) wn++; else wn--;

}

**if**(!wn) **return** -1;

**return** 1;

}

typedef pair<double,point> circle;

// Circunferencia que passa pelos pontos A, B e C

circle circumcirle(point A, point B, point C){

double a = norm(C-B), b = norm(C-A), c = norm(A-B);

double aa = a \* (b+c-a), bb = b \* (c+a-b), cc = c \* (a+b-c);

point centre = (A\*aa + B\*bb + C\*cc) / (aa + bb + cc);

**if** (a == 0) centre = (A+B) \* 0.5;

**if** (b == 0) centre = (A+B) \* 0.5;

**if** (c == 0) centre = (A+C) \* 0.5;

**return** circle(abs(A-centre), centre);

}

// include para random\_shuffle()

#include <utility>

**#define** EPS 1E-8

**#define** CIRCLE(a, b) circle(abs(a-b)/2, (a+b)\*0.5)

**#define** atalho(i,j) fu(i,j)**if**(abs(p[i]-D.second)-EPS>D.first)

// Calcula o menor circumcirculo de n pontos em O(n)

// mindisk1 e mindisk2 sao funcoes auxiliares

circle mindisk(poly &p){

int s = p.size();

**if**(s==1) **return** circle(0.0, p[0]);

circle D=CIRCLE(p[0],p[1]);

atalho(i,s){D=CIRCLE(p[0],p[i]);

atalho(j, i){D=CIRCLE(p[j], p[i]);

atalho(k,j) D=circumcirle(p[j], p[k], p[i]);

}

}

**return** D;

}

Algoritmos em Grafos

Gowen – Min Cost Flow

/\* Min cost flow em O(w \* n²)

\* Entrada:

\* - n: número de vertices da rede

\* - s: vértice iniciall (fonte)

\* - t: vértice final (destino)

\* - cap: Capacidade das arestas

\* - cost: Custo unitário por fluxo

\* Saída:

\* Retorna valores diferentes de -1 caso configuração

\* de custo mínimo.

\* Vértices enumerados de 1 a n.

\*/

**#define** MIN(a,b) ((a)<(b)?(a):(b))

**#define** MAX\_SIZE 1024

**#define** INF 1E10

**#define** NORMAL 1

**#define** REVERSE -1

**int** n, s, t;

**int** cap[MAX\_SIZE][MAX\_SIZE], flow[MAX\_SIZE][MAX\_SIZE], tp[MAX\_SIZE], p[MAX\_SIZE];

**double** cost[MAX\_SIZE][MAX\_SIZE], dist[MAX\_SIZE];

int floyd(){

**int** i, j, k;

**double** d[MAX\_SIZE][MAX\_SIZE];

**for** (i=1; i<=n; i++){

**for** (j=1; j<=n; j++)

**if** (cap[i][j] > 0)

d[i][j] = cost[i][j];

**else**

d[i][j] = INF;

d[i][i] = 0;

}

**for** (k=1; k<=n; k++)

**for** (i=1; i<=n; i++)

**for** (j=1; j<=n; j++){

**if** (d[i][j] + d[j][i] < 0)

**return** 0;

**if** (d[i][j] > d[i][k] + d[k][j])

d[i][j] = d[i][k] + d[k][j];

}

**for** (i=1; i<=n; i++)

dist[i] = d[s][i];

**return** 1;

}

int dijkstra(){

**int** i, j, min, nrex;

**int** mark[MAX\_SIZE];

**double** d[MAX\_SIZE];

**for** (i=1; i<=n; i++){

d[i] = dist[i];

dist[i] = INF;

mark[i] = 0;

}

p[s] = s;

dist[s] = 0;

mark[s] = 1;

i = s; nrex = n;

**while** (--nrex){

min = -1;

**for** (j=1; j<=n; j++){

**if** (mark[j])

continue;

**if** (cap[i][j] > flow[i][j])

**if** (dist[j] > dist[i] + cost[i][j] + d[i] - d[j]){

dist[j] = dist[i] + cost[i][j] + d[i] - d[j];

p[j] = i;

tp[j] = NORMAL;

}

**if** (flow[j][i] > 0)

**if** (dist[j] > dist[i]-cost[j][i]+d[i]-d[j]){

dist[j] = dist[i]-cost[j][i]+d[i]-d[j];

p[j] = i;

tp[j] = REVERSE;

}

**if** (min == -1)

min = j;

**if** (dist[min] > dist[j])

min = j;

}

mark[min] = 1;

i = min;

}

**if** (dist[t] == INF) **return** 0;

**for** (i=1; i<=n; i++)

dist[i] += d[i];

**return** 1;

}

double gowen(int wanted\_flow){

**int** i, j, f = 0, aug;

**double** c = 0;

**for** (i=1; i<=n; i++){

dist[i] = 0;

**for** (j=1; j<=n; j++)

flow[i][j] = 0;

}

**if** (**!**floyd()) **return** -1;

**while** (f < wanted\_flow){

**if** (!dijkstra()) **return** -1;

aug = wanted\_flow - f;

**for** (i = t; i != s; i = p[i]){

**if** (tp[i] == NORMAL)

aug = MIN(aug, cap[p[i]][i]-flow[p[i]][i]);

**else**

aug = MIN(aug, flow[i][p[i]]);

}

**for** (i = t; i != s; i = p[i]){

**if** (tp[i] == NORMAL){

flow[p[i]][i] += aug;

c += cost[p[i]][i] \* aug;

} **else** {

flow[i][p[i]] -= aug;

c -= cost[i][p[i]] \* aug;

}

}

f += aug;

}

**return** c;

}

Componentes biconexas e pontes

/\* Componentes Biconectadas e pontes

\* Entrada:

\* - G: **Lista de adjacencia** do grafo nao-direcionado

\* - g: grau dos vertices

\* - N: Quantidade de vértices

\* Saída:

\* A gosto do freguês.

\* Obs: vértices de 0 a N-1

\* Componentes biconexas não têm pontes, mas podem

\* ter pontos de articulação!

\*/

**int** vis[NMAX];

void reset(){

**for** (int i=0; i<N\_vertices; i++)

vis[i] = 0;  
}

**int** biponte(**int** v, **int** pai){

if(vis[v]) **return** vis[v];

**int** R = vis[v] = intime++;

**for**(**int** i=0; i<g[v]; i++){

**if**(G[v][i] == pai)

**continue**;

**int** r = biponte(G[v][i], v);

**if**(r < R){

// v está na mesma componente biconexa de G[v][i]

// union\_set(v, G[v][i]);

R = r;

}

}

**if**(R >= vis[v] && pai != -1)

printf("ponte -> (%d, %d)\n", v+1, pai+1);

// Se o grafo permitir multiplas arestas, verificar

// se a aresta (v+1, pai+1) ocorre mais de uma vez.

**return** R;

}

**void** biconexo(){

intime = 1;

**for** (**int** i=0; i<N; i++)

**biponte**(i, -1);

}

Componentes fortemente conexas

/\* Determina as CFC em O(V+E)

\* Entrada:

\* - G: **Matriz de adjacência** do grafo direcionado

\* - n: Quantidade de vértices

\* Saída:

\* A gosto do freguês, modifique a função CFC().

\* Obs: vértices de 0 a n-1

\*/

#include <stack>

#include <vector>

using namespace std;

**int** n;

**int** G[NMAX][NMAX];

**bool** foi[NMAX];

**stack**<**int**> P;

**vector**<**int**> V;

**void** **dfs**(**int** i){

foi[i] = **true**;

**for** (**int** j=0;j<n;j++)

**if** (G[i][j] && !foi[j])

**dfs**(j);

P.push(i);

}

**void** dfsT(**int** i){

foi[i] = **true**;

V.push\_back(i);

**for**(**int** j=0;j<n;j++)

**if** (G[j][i] && **!**foi[j]){

**// j está na mesma componente conexa que i**

dfsT(j);

}

}

**void** CFC(){

memset(foi,0,sizeof(foi));

**for**(**int** i=0;i<n;i++)

**if** (!foi[i])

dfs(i);

memset(foi,0,sizeof(foi));

**while**(!P.empty()){

**int** u = P.top(); P.pop();

**if** (!foi[u]){

V.clear();

dfsT(u);

// V contém os vértices de um CFC

}

}

}

Lista de adjacência

Na hora de implementar, arrancar fora as structs... (não deu tempo de revisar a parte de grafos).

#include <string.h>

template<class tipo >

struct gLista{

int G[NMAX][NMAX], grau[NMAX];

tipo w[NMAX][NMAX];

int n;

**void** clear(){

**memset**(grau, 0, sizeof(grau));

**memset**(w, 0, sizeof(w));

}

};

Matriz de adjacência

Na hora de implementar, arrancar fora as structs... (não deu tempo de revisar a parte de grafos).

#include <string.h>

template<class tipo>

struct gMatriz{

tipo G[NMAX][NMAX];

int n,s,t;

**void** clear(){

**memset**(G,0,sizeof(G));

}

};

Ordenação topológica – lista topologica()

// Ordenação topológica com lista de adj O(V + E)

// Vértices enumerados de 0 até n-1

// Entrada:

// 1 - g: lista de adjacencia

// Saída:

// stack<int> com os elementos na ordem topológica

#include <stack>

**typedef** gLista<int > grafo;

**bool** foi[NMAX];

**void** **topo\_dfs**(int i, grafo &g, stack<int> Q){

foi[i] = 1;

**for** (**int** j = 0; j < g.grau[i]; j++)

**if** (!foi[g.G[i][j]])

**topo\_dfs**(g.G[i][j], g, Q);

Q.push(i);

}

**stack**<**int**> **topologica**(grafo &g){

**stack**<**int**> ret;

**memset**(foi, 0, sizeof(foi));

**for** (int i = 0; i < g.n; i++)

**if** (!foi[i])

**topo\_dfs**(i, g, ret);

}

Caminho mínimo em DAG dagShortestPath()

#include <algorithm>

#include <limits.h>

// Dag Shortest Path com lista de adj O(V + E)

// Determina caminho mais curto de origem s

// em um GRAFO ACICLIO ORIENTADO

// Vértices enumerados de 0 até n-1

// Entrada:

// 1 - g: **lista de adjacencia**

// 2 - s: Vértice de origem

// 3 - d[]: Vetor que comportara as menores distâncias

// Saída:

// vetor d[] com as distâncias

// obs.: Aceita pesos negativos

// needs topologica()

typedef gLista<int > grafo;

**void** **dagShortestPath**(grafo &g, int s, int d[]){

**for** (int i=0; i < g.n; i++)

d[i] = INT\_MAX/2;

d[s] = 0;

**for**(stack<int> Q = **topologica**(g); !Q.empty(); Q.pop()){

**int** i = Q.top();

**for** (**int** j = 0; j < g.grau[i]; j++)

d[g.G[i][j]] = **min**(d[g.G[i][j]], d[i] + g.w[i][g.G[i][j]]);

}

}

Weighted Bipartite Matching – maxmatch()

// **Maximal** Weighted Bipartite Matching (Hungarian)

// Complexidade: O(n^3) - Kuhn-Munkres

// Entrada: w[][] = matriz com os pesos

// n = numero de vértices

// Saída: m[] matriz com o matching

// Valor de retorno: Soma do matching

// Obs.: vértices de 0 a n-1

// Se quiser o matching mínimo, passe a matriz -w.

#include <vector>

#include <numeric>

using namespace std;

**#define** NMAX 30

**#define** INF (1<<20)

**#define** SIZE (2\*NMAX+1)

**int** n = 0;

**int** w[NMAX][NMAX], m[SIZE];

**int** maxmatch(){

**int** L[SIZE], slack[SIZE], pai[SIZE];

**char** s[SIZE], t[SIZE];

**int** nmatch = 0;

**for** (**int** i = 0, j = n; I < n; i++, j++){

L[i] = \*max\_element(&w[i][0], &w[i][n]);

L[j] = 0;

m[i] = m[j] = -1;

}

**while**(nmatch < n){

**for**(**int** i = 0, j = n; i < n; i++, j++){

s[i] = 0; t[j] = 0;

slack[j] = INF;

pai[i] = i; pai[j] = j;

}

**int** u, y;

vector<**int**> V;

**for**(u=0; u<n; u++)

**if**(m[u] == -1){

s[u]++;

V.push\_back(u);

break;

}

**for** (;;){

**for** (**int** i = 0; i < V.size(); i++){

u = V[i];

**for** (y = 2\*n - 1; y >= n; y--)

**if** (!t[y]){

**int** d = L[u] + L[y] - w[u][y-n];

**if** (!d){

pai[y] = u; *// aumenta a hungarian tree*

**if** (m[y] == -1){ *// augmenting path encontrado*

**for** (; pai[y] != y; y = pai[pai[y]]){

m[y] = pai[y];

m[pai[y]] = y;

}

nmatch++;

**goto** fora;

}

t[y]++; u = m[y];

s[u]++; V.push\_back(u);

pai[u] = y;

}

**else** **if** (slack[y] > d) slack[y] = d;

}

}

**int** a = INF;

**for** (y = 2\*n - 1; y >= n; y--)

**if**(!t[y] && a > slack[y]) a = slack[y];

**for** (**int** i=0, j=n; i<n; i++, j++){

**if**(s[i]) L[i] -= a;

**if**(t[j]) L[j] += a;

**else** slack[j] -= a;

}

}

fora: continue;

}

**return** accumulate(&L[0], &L[2\*n], 0);

}

int main(){

printf ("Maximal Match = %d\n", **maxmatch()** );

printf ("Matchings:\n");

for (int i = 0; i < n; i++)

printf("linha = %d, coluna = %d: valor = %d\n",

**m[i] - n**, **i**, w[m[i] - n][i]);

}

Floyd-Warshall

// all pairs shortest path

12 **procedure** *FloydWarshall* ()

13 **for** *k*: = 1 **to** *n*

14 **for each** (*i*, *j*) **in** {1,..,*n*}2

15 path[i][j] = min ( path[i][j], path[i][k]+path[k][j] );

-----

for (int i = 0; i < N; i++)  
    for (int j = 0; j < N; j++)  
        next[i][j] = j;  
  
**for** (**int** k = 0; k < N; k++)  
  **for** (**int** i = 0; i < N; i++)  
    **for** (**int** j = 0; j < N; j++) {  
      **if** (dist[i][k] + dist[k][j] < dist[i][j]) {  
        dist[i][j] = dist[i][k] + dist[k][j];  
        next[i][j] = next[i][k];  
      }  
  
// print a path from s to t  
**for** (**int** x = s; x != t; x = next[x][t]) printf("%d ", x);  
printf("%d\n", t);

Fluxo máximo - Golgberg goldberg()

/\* Goldberg (push-relabel) com matriz O(V^3)

\* Determina MAX FLOW entre s e t

\* Vértices enumerados de 0 até n-1

\* Entrada:

\* 1 - G: Matriz das capacidades

\* 2 - G.s: Vértice de origem

\* 3 - G.t: Vértice de destino

\* 4 - G.n: total de vértices

\* Saída:

\* 1 - <tipo> max flow.

\* 2 - matriz flow (fluxos).

\*/

#include <algorithm>

#include <queue>

#include <string.h>

using namespace std;

**#define** fu(i,n) for(int i = 0; i < (n); i++)

**#define** MIN(a,b) ((a) < (b) ? (a) : (b))

**#define** NMAX 100

typedef struct gMatriz{

int G[NMAX][NMAX]; // tipo

int n, s, t;

} grafo;

deque<int> Q;

bool foi[NMAX];

**void** enqueue(int v, int relabel){

if (foi[v] == 1) **return**;

foi[v] = 1;

if (relabel) Q.push\_front(v);

else Q.push\_back(v);

}

int dequeue(){

int v = Q.front(); Q.pop\_front();

foi[v] = 0;

**return** v;

}

int flow[NMAX][NMAX]; // tipo

int goldberg(grafo &g){ // tipo

int ex[NMAX], h[NMAX]; // tipo

int delta; // tipo

int u, v, mind;

memset(foi, 0, sizeof(foi));

memset(flow, 0, sizeof(flow));

h[g.s] = g.n;

fu (u,g.n){

if (u == g.s) continue;

h[u] = 0;

flow[g.s][u] = g.G[g.s][u];

flow[u][g.s] = -flow[g.s][u];

ex[u] = flow[g.s][u];

if (ex[u] > 0 && u != g.t)

enqueue(u, 1);

}

while (!Q.empty()){

u = dequeue(); mind = 2\*g.n;

fu (v,g.n){

if (!ex[u]) break;

if(flow[u][v] == g.G[u][v] || u == v)

continue;

if (h[v] == h[u] - 1){

delta = min(ex[u], g.G[u][v]-flow[u][v]);

flow[u][v] += delta;

flow[v][u] -= delta;

ex[u] -= delta;

ex[v] += delta;

if(v != g.t && v != g.s)

enqueue(v,0);

}

else if (mind > h[v]) mind = h[v];

}

if (ex[u] > 0){

h[u] = mind + 1;

enqueue(u, 1);

}

}

**return** ex[g.t];

}

Bellman-Ford bellmanFord()

procedure BellmanFord(list vertices, list edges, vertex source)

// Step 1: Initialize graph

for each vertex v in vertices:

if v is source then v.distance := 0

else v.distance := infinity

v.predecessor := null

// Step 2: relax edges repeatedly

for i from 1 to size(vertices) - 1:

for each edge uv in edges: // uv is the edge from u to v

u := uv.source

v := uv.destination

if u.distance + uv.weight < v.distance:

v.distance := u.distance + uv.weight

v.predecessor := u

// Step 3: check for negative-weight cycles

for each edge uv in edges:

u := uv.source

v := uv.destination

if u.distance + uv.weight < v.distance:

error "Graph contains a negative-weight cycle"

/\* Bellman-Ford com lista de adj O(V\*E)

\* Determina caminho mais curto de origem s

\* Vértices enumerados de 0 até n-1

\* Entrada:

\* 1 - g: lista de adjacencia

\* 2 - s: Vértice de origem

\* 3 - dist[]: Vetor que comportara

\* as menores distâncias

\* Saída:

\* **true** - se existir ciclo negativo

\* **false** - se não existir ciclo negativo

\* vetor dist[] com as distâncias

\*/

#include <algorithm>

#include <limits.h>

typedef gLista<int > grafo;

bool bellmanFord(grafo &g, int s, int d[]){

for(int i = 0; i < g.n; i++)

d[i] = INT\_MAX/2;

d[s] = 0;

for (int k = 0; k < g.n – 1; k++)

for (int i = 0; i < g.n; i++)

for (int j = 0; j < g.grau[i]; j++)

d[g.G[i][j]] = **min**(d[g.G[i][j]], d[i] + g.w[i][g.G[i][j]]);

// so para dectar ciclo negativo alcansável a partir de s

for (int i = 0; i < g.n; i++)

for (int j = 0; j < g.grau[i]; j++)

if (d[g.G[i][j]] > d[i] + g.w[i][g.G[i][j]])

**return true**;

**return** **false**;

}

**// Bellman-Ford com lista de arestas**

int n, m;

int d[1010];

**typedef** **struct**{int s, t, w;} **aresta**;

**aresta** A[2010];

**bool** **negcircle**(){

**for**(**int** i = 0; i < n; i++)

d[i] = INT\_MAX/2;

d[0] = 0;

**for** (**int** k = 0; k < n - 1; k++)

**for**(**int** i=0; i<m; i++)

d[A[i].t] = **min**(d[A[i].t], d[A[i].s] + A[i].w);

// para detectar ciclo negativo alcansável a partir de s

**for**(**int** i=0; i<m; i++)

**if** (d[A[i].t] > d[A[i].s] + A[i].w)

**return** **true**;

**return** **false**;

}

Caminho mínimo – Dijkstra - matriz

dijkstra()

/\* Dijkstra com matriz sem heap O(n^2)

\* Determina menor distância entre s e t

\* Vértices enumerados de 0 até n-1

\* Entrada:

\* 1 - G: Matriz das distâncias

\* 2 - s: Vértice de origem

\* 3 - t: Vértice de destino

\* Saída:

\* 1 - d[]: Vetor com as distâncias mínimas (t = -1)

\* 2 - pai[]

\* Auxiliar:

\* 1 - foi[]

\*/

#include <string.h>

**#define** INF (1<<30)-1

**#define** NMAX 100

typedef gMatriz<int> grafo;

int d[NMAX], pai[NMAX];

bool foi[NMAX];

int getmin(grafo &g){

int imin = -1;

int dmin = INF;

fu(i, g.n)

if(d[i] < dmin && !foi[i]){

imin = i; dmin = d[i];

}

**return** imin;

}

int dijkstra(grafo &g, int s, int t = -1){

for (int i = 0; i < g.n; i++){

d[i] = INF;

foi[i] = 0;

}

**memset**(pai,0xFF,sizeof(pai));

d[s] = 0;

int i;

while((i=**getmin**(g)) != -1){

foi[i] = 1;

if(i == t) **return** d[i];

for (int j = 0; j < g.n; j++)

if(g.G[i][j] && !foi[j] && d[j] > d[i] + g.G[i][j]){

d[j] = d[i] + g.G[i][j];

pai[j] = i;

}

}

**return** -1;

}

Caminho mínimo – Dijkstra – lista + set dijkstra()

/\* Dijkstra com matriz com heap O((e+n)\*log(n))

\* Determina menor distância entre s e t

\* Vértices enumerados de 0 até n-1

\* Entrada:

\* 1 - G: Matriz das distâncias

\* 2 - s: Vértice de origem

\* 3 - t: Vértice de destino

\* Saída:

\* 1 - d[]: Vetor com as distâncias mínimas (t = -1)

\* 2 - pai[]

\* Auxiliar:

\* 1 - foi[]

\*/

#include <set>

#include <utility>

**#define** INF (1<<30)-1

typedef gLista<int> grafo;

typedef pair<int,int> ii;

int d[NMAX], pai[NMAX];

int dijkstra(grafo &g, int s, int t = -1){

bool foi[NMAX];

set<ii> H;

for(int i = 0; i < g.n; i++){

d[i] = INF;

foi[i] = 0;

}

memset(pai,0xFF,sizeof(pai));

H.**insert**(ii(0,s));

d[s] = 0;

**while** (!H.**empty**()){

int i = H.**begin**()->second;

H.**erase**(H.**begin**());

foi[i] = 1;

**if** (i == t) **return** d[i];

**for**(int j = 0; j < g.grau[i]; j++){

int u = g.G[i][j];

**if** (!foi[u] && d[u] > d[i] + g.w[i][u]){

**if** (d[u] != INF) H.**erase**(ii(d[u], u));

d[u] = d[i] + g.w[i][u];

pai[u] = i;

H.**insert**(ii(d[u], u));

}

}

}

**return** -1;

}

Euler Tour

* An undirected graph has a closed Euler tour iff it is connected and each vertex has an even degree.
* An undirected graph has an open Euler tour iff it is connected, and each vertex, except for exactly two vertices, has an even degree. The two vertices of odd degree have to be the endpoints of the tour.
* A directed graph has a closed Euler tour iff it is strongly connected and the in-degree of each vertex is equal to its out-degree.
* Similarly, a directed graph has an open Euler tour iff it is strongly connected and for each vertex the difference between its in-degree and out-degree is 0, except for two vertices. In one of them the difference has to be +1 (this will be the beginning of the tour) and in the other one the difference has to be -1 (this will be its end).

'tour' is a stack

find\_tour(u):

for each edge e=(u,v) in E:

remove e from E

find\_tour(v)

**prepend** u to tour

to find the tour, clear stack 'tour' and call find\_tour(u),

where u is any vertex with a non-zero degree.

Algoritmos com strings

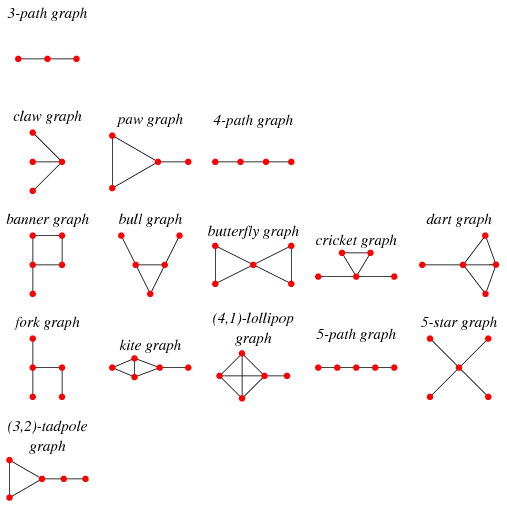
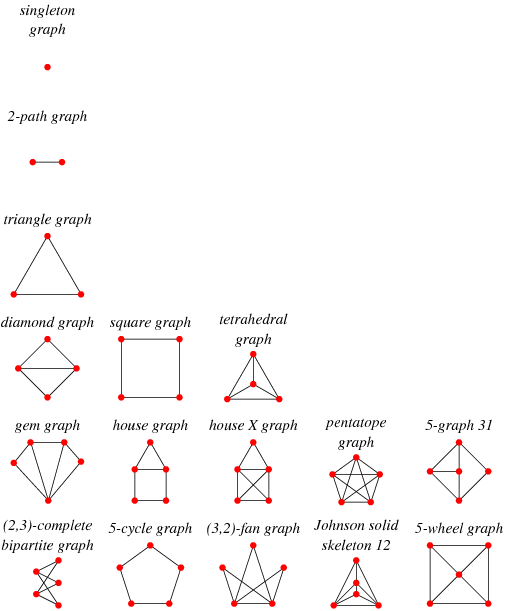


Tabela ASCII

Char Dec Oct Hex | Char Dec Oct Hex | Char Dec Oct Hex | Char Dec Oct Hex

-------------------------------------------------------------------------------------

(nul) 0 0000 0x00 | (sp) 32 0040 0x20 | @ 64 0100 0x40 | ` 96 0140 0x60

(soh) 1 0001 0x01 | ! 33 0041 0x21 | A 65 0101 0x41 | a 97 0141 0x61

(stx) 2 0002 0x02 | " 34 0042 0x22 | B 66 0102 0x42 | b 98 0142 0x62

(etx) 3 0003 0x03 | # 35 0043 0x23 | C 67 0103 0x43 | c 99 0143 0x63

(eot) 4 0004 0x04 | $ 36 0044 0x24 | D 68 0104 0x44 | d 100 0144 0x64

(enq) 5 0005 0x05 | % 37 0045 0x25 | E 69 0105 0x45 | e 101 0145 0x65

(ack) 6 0006 0x06 | & 38 0046 0x26 | F 70 0106 0x46 | f 102 0146 0x66

(bel) 7 0007 0x07 | ' 39 0047 0x27 | G 71 0107 0x47 | g 103 0147 0x67

(bs) 8 0010 0x08 | ( 40 0050 0x28 | H 72 0110 0x48 | h 104 0150 0x68

(ht) 9 0011 0x09 | ) 41 0051 0x29 | I 73 0111 0x49 | i 105 0151 0x69

(nl) 10 0012 0x0a | \* 42 0052 0x2a | J 74 0112 0x4a | j 106 0152 0x6a

(vt) 11 0013 0x0b | + 43 0053 0x2b | K 75 0113 0x4b | k 107 0153 0x6b

(np) 12 0014 0x0c | , 44 0054 0x2c | L 76 0114 0x4c | l 108 0154 0x6c

(cr) 13 0015 0x0d | - 45 0055 0x2d | M 77 0115 0x4d | m 109 0155 0x6d

(so) 14 0016 0x0e | . 46 0056 0x2e | N 78 0116 0x4e | n 110 0156 0x6e

(si) 15 0017 0x0f | / 47 0057 0x2f | O 79 0117 0x4f | o 111 0157 0x6f

(dle) 16 0020 0x10 | 0 48 0060 0x30 | P 80 0120 0x50 | p 112 0160 0x70

(dc1) 17 0021 0x11 | 1 49 0061 0x31 | Q 81 0121 0x51 | q 113 0161 0x71

(dc2) 18 0022 0x12 | 2 50 0062 0x32 | R 82 0122 0x52 | r 114 0162 0x72

(dc3) 19 0023 0x13 | 3 51 0063 0x33 | S 83 0123 0x53 | s 115 0163 0x73

(dc4) 20 0024 0x14 | 4 52 0064 0x34 | T 84 0124 0x54 | t 116 0164 0x74

(nak) 21 0025 0x15 | 5 53 0065 0x35 | U 85 0125 0x55 | u 117 0165 0x75

(syn) 22 0026 0x16 | 6 54 0066 0x36 | V 86 0126 0x56 | v 118 0166 0x76

(etb) 23 0027 0x17 | 7 55 0067 0x37 | W 87 0127 0x57 | w 119 0167 0x77

(can) 24 0030 0x18 | 8 56 0070 0x38 | X 88 0130 0x58 | x 120 0170 0x78

(em) 25 0031 0x19 | 9 57 0071 0x39 | Y 89 0131 0x59 | y 121 0171 0x79

(sub) 26 0032 0x1a | : 58 0072 0x3a | Z 90 0132 0x5a | z 122 0172 0x7a

(esc) 27 0033 0x1b | ; 59 0073 0x3b | [ 91 0133 0x5b | { 123 0173 0x7b

(fs) 28 0034 0x1c | < 60 0074 0x3c | \ 92 0134 0x5c | | 124 0174 0x7c

(gs) 29 0035 0x1d | = 61 0075 0x3d | ] 93 0135 0x5d | } 125 0175 0x7d

(rs) 30 0036 0x1e | > 62 0076 0x3e | ^ 94 0136 0x5e | ~ 126 0176 0x7e

(us) 31 0037 0x1f | ? 63 0077 0x3f | \_ 95 0137 0x5f | (del) 127 0177 0x7f