Sumário

Lista de Tabelas

Algoritmos

1 Tabelas

tipo	bits	min...max	precisao
char	8	0..127	2
signed char	8	-128..127	2
unsigned char	8	0..255	2
short	16	-32.768 .. 32.767	4
unsigned short	16	0 .. 65.535	4
int	32	-2x10**9 .. 2 x 10**9	9
unsigned int	32	0 .. 4x10**9	9
int64_t	64	-9 x 10**18 .. 9 x 10**18	18
uint64_t	64	0 .. 18 x 10**18	19

Tabela 1: Limites de representação de dados

0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5.040
8! = 40.320
9! = 362.880
10! = 3.628.800
11! = 39.916.800
12! = 479.001.600 [limite do (unsigned) int]
13! = 6.227.020.800
14! = 87.178.291.200
15! = 1.307.674.368.000
16! = 20.922.789.888.000
17! = 355.687.428.096.000
18! = 6.402.373.705.728.000
19! = 121.645.100.408.832.000
20! = 2.432.902.008.176.640.000 [limite do (u)int64_t]

Tabela 2: Fatorial

Tipo	%
char	c
int	d
float	e, E, f, g, G
int (octal)	o
int (hexa)	x, X
uint	u
char*	s

Tabela 3: scanf() - %[*][width][modifiers]type

modifiers	tipo
h	short int (d, i, n), or unsigned short int (o, u, x)
l	long int (d, i, n), or unsigned long int (o, u, x), or double (e, f, g)
ll	long long int (d, i, n), or unsigned long long int (o, u, x)
L	long double (e, f, g)

Tabela 4: scanf() %[*][width][modifiers]type

função	descrição
atof	Convert string to double
atoi	Convert string to integer
atol	Convert string to long integer
strtod	Convert string to double
strtol	Convert string to long integer
strtoul	Convert string to unsigned long integer

Tabela 5: stdlib

função	descrição
cos	Compute cosine
sin	Compute sine
tan	Compute tangent
acos	Compute arc cosine
asin	Compute arc sine
atan	Compute arc tangent
atan2	Compute arc tangent with two parameters
cosh	Compute hyperbolic cosine
sinh	Compute hyperbolic sine
tanh	Compute hyperbolic tangent
exp	Compute exponential function
frexp	Get significand and exponent
ldexp	Generate number from significand and exponent
log	Compute natural logarithm
log10	Compute common logarithm
modf	Break into fractional and integral parts
pow	Raise to power
sqrt	Compute square root
ceil	Round up value
fabs	Compute absolute value
floor	Round down value
fmod	Compute remainder of division

Tabela 6: math (angulos em radianos)

2 Codigos

2.1 Exemplos

Código 1: Modelo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include <inttypes.h>
#include <ctype.h>
#include <limits.h>

#include <algorithm>
#include <utility>
#include <iostream>

#include <map>
#include <set>
#include <vector>
#include <list>
#include <queue>
#include <sstream>

using namespace std;

#define abs(a) ((a) > 0 ? (a) : -(a))

int main()
{
    int n;

    cin >> n;

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

    }

    while (cin >> n)
    {

    }
    return 0;
}
```

Código 2: comparcao de ponto flutuante

```
/**
 * -1 se x < y
 * 0 se x = y
 * 1 se x > y
 */
```

```
const double EPS = 1e-10;
#define _inline(f...) f() __attribute__((always_inline)); f
_inline(int cmp)(double x, double y = 0, double tol = EPS)
{
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}
```

Código 3: .vimrc para a configuração do vim

```
set ai noet ts=4 sw=4 bs=2
set cindent
```

Código 4: função que acelara o cin. Não deve ser usada com printf

```
std::cout.sync_with_stdio(false);
```

Código 5: printf

```
/* printf example */
#include <stdio.h>

int main()
{
    printf ("Characters: %c %c \n", 'a', 65);
    printf ("Decimals: %d %ld\n", 1977, 650000L);
    printf ("Preceding with blanks: %10d \n", 1977);
    printf ("Preceding with zeros: %010d \n", 1977);
    printf ("Some different radixes: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100);
    printf ("floats: %4.2f %+.0e %E %4.2f\n", 3.1416, 3.1416, 3.1416, 3.1);
    printf ("Width trick: %*d \n", 5, 10);
    printf ("%s \n", "A string");
    return 0;
}
/* %[flags (-, +, etc)][width][.precision][length (h,l,L)]specifier
Characters: a A
Decimals: 1977 650000
Preceding with blanks:       1977
Preceding with zeros: 0000001977
Some different radixes: 100 64 144 0x64 0144
floats: 3.14 +3e+000 3.141600E+000 3.10
Width trick:    10
A string
*/
```

Código 6: exemplo de map

```
#include <iostream>
#include <map>
using namespace std;

int main ()
{
  map<char,int> mymap;
  map<char,int>::iterator it;
  pair<map<char,int>::iterator,bool> ret;

  // first insert function version (single parameter):
  mymap.insert ( pair<char,int>('a',100) );
  mymap.insert ( pair<char,int>('z',200) );

  ret=mymap.insert (pair<char,int>('z',500) );
  if (ret.second==false)
  {
    cout << "element 'z' already existed";
    cout << " with a value of " << ret.first->second << endl;
  }

  // third insert function version (range insertion):
  map<char,int> anothermap;
  anothermap.insert(mymap.begin(),mymap.find('c'));

  // showing contents:
  cout << "mymap contains:\n";
  for ( it=mymap.begin() ; it != mymap.end(); it++ )
    cout << (*it).first << " => " << (*it).second << endl;

  map<char,string> mymap;
  mymap['a']="an element";
  if (mymap.count('a') > 0)
      cout << mymap['a'] << " is an element of mymap.\n";

  while (!mymap.empty())
  {
     cout << mymap.begin()->first << " => ";
     cout << mymap.begin()->second << endl;
     map<char, int>::iterator erasedelement = mymap.erase(mymap.begin());
  }

  return 0;
}
```

Código 7: exemplo de set e multset

```
#include <iostream>
#include <set>
using namespace std;

int main ()
{
  multiset<int> mymultiset;
  multiset<int>::iterator it;

  // set some initial values:
  for (int i=1; i<=5; i++) mymultiset.insert(i*10);   // 10 20 30 40 50

  cout << "size: " << (int) mymultiset.size() << endl;
  cout << "count: " << (int) mymultiset.count(10) << endl;

  it=mymultiset.find(20);
  mymultiset.erase (it);

  if (! mymultiset.empty)
   mymultiset.erase (mymultiset.find(40));

  for (it=mymultiset.begin(); it!=mymultiset.end(); it++)
    cout << " " << *it;

  int myints[]={19,72,4,36,20,20};
  multiset<int> first (myints,myints+3);      // 4,19,72
  multiset<int> second (myints+3,myints+6);   // 20,20,36

  first.swap(second); // troca conteudo. o primeiro fica [20,20,36] e o segundo [4,19,72]

  return 0;
}
```

Código 8: exemplo de list

```
#include <iostream>
#include <list>
using namespace std;

int main ()
{
  list<int> mylist (2,100);          // two ints with a value of 100
  mylist.push_front (200);
  mylist.push_back (300);

  it = mylist.begin();
  mylist.insert (it,10);
  mylist.insert (it,2,20); // two ints with a value of 20

  mylist.reverse(); // Reverses the order of the elements in the list.

  cout << "mylist contains:";
  for (list<int>::iterator it=mylist.begin(); it!=mylist.end(); ++it)
    cout << " " << *it;

  cout << "Popping out the elements in mylist:";
  while (!mylist.empty())
  {
    cout << " " << mylist.front();
    mylist.pop_front();
  }

  while (!mylist.empty())
  {
    cout << " " << mylist.back();
    mylist.pop_back();
  }

  cout << mylist.size() << endl;

  return 0;
}
```

Código 9: exemplo de queue

```
#include <iostream>
#include <queue>
using namespace std;

int main ()
{
  queue<int> myqueue;
  int sum (0);

  for (int i=1;i<=10;i++) myqueue.push(i);

  myqueue.back() -= myqueue.front();

  cout << "size: " << (int) myqueue.size() << endl;

  while (!myqueue.empty())
  {
     sum += myqueue.front();
     myqueue.pop();
  }

  cout << "total: " << sum << endl;

  return 0;
}
```

Código 10: exemplo de priority queue

```
#include <iostream>
#include <queue>
using namespace std;

int main ()
{
  priority_queue<int> mypq;

  mypq.push(30);
  mypq.push(100);
  mypq.push(25);
  mypq.push(40);

  cout << "size: " << (int) mypq.size() << endl;

  cout << "Popping out elements...";
  while (!mypq.empty())
  {
     cout << " " << mypq.top();
     mypq.pop();
  }
  cout << endl;

  return 0;
}
```

Código 11: exemplo de stack

```
#include <iostream>
#include <stack>
using namespace std;

int main ()
{
  stack<int> mystack;
  int sum = 0;

  mystack.push(10);
  mystack.push(20);

  mystack.top() -= 5;

  while (!mystack.empty())
  {
     sum += mystack.top();
     mystack.pop();
  }

  cout << "size: " << (int) mystack.size() << endl;

  return 0;
}
```

Código 12: exemplo de vector

```
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
  vector<int> myvector (3,100); // (100 100 100)
  vector<int>::iterator it;

  myvector.reserve(100);

  for (int i=0; i<myvector.size(); i++)
    myvector.at(i)=i; // = myvector[i] = i

  it = myvector.begin();
  it = myvector.insert ( it , 200 );
  myvector.insert (it,2,300);

  vector<int> anothervector (2,400);
  int myarray [] = { 501,502,503 };
  vector<int> initializer(myarray/*PointerInicio*/, myarray+sizeof(
      myarray)/sizeof(int)/*PointerFim*/);
  myvector.insert (it+2,anothervector.begin(),anothervector.end());
  myvector.insert (myvector.begin(), myarray, myarray+3);

  cout << "myvector contains:";
  for (it=myvector.begin(); it<myvector.end(); it++)
    cout << " " << *it;
  cout << endl;

  // erase the 6th element
  myvector.erase (myvector.begin()+5);
  int sum;
  while (!myvector.empty())
  {
     sum += myvector.back();
     myvector.pop_back();
  }

  return 0;
}
```

Código 13: exemplo de string

```
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string str ("There are two needles in this haystack with needles.");
  string str2 ("needle");
  size_t found;

  // different member versions of find in the same order as above:
  found=str.find(str2);
  if (found!=string::npos)
    cout << "first 'needle' found at: " << int(found) << endl;

  found=str.find("needles are small",found+1,6);
  if (found!=string::npos)
    cout << "second 'needle' found at: " << int(found) << endl;

  found=str.find("haystack");
  if (found!=string::npos)
    cout << "'haystack' also found at: " << int(found) << endl;

  found=str.find('.');
  if (found!=string::npos)
    cout << "Period found at: " << int(found) << endl;

  // let's replace the first needle:
  str.replace(str.find(str2),str2.length(),"preposition");
  cout << str << endl;

  string str="We think in generalities, but we live in details.";
```

```
                              // quoting Alfred N. Whitehead
  string str2, str3;
  size_t pos;

  str2 = str.substr (12,12); // ”generalities”

  pos = str.find("live");    // position of ”live” in str
  str3 = str.substr (pos);   // get from ”live” to the end

  cout << str2 << ' ' << str3 << endl;


  return 0;
}
/*
first 'needle' found at: 14
second 'needle' found at: 44
'haystack' also found at: 30
Period found at: 51
There are two prepositions in this haystack with needles.
generalities live in details.
*/
```

Código 14: exemplo de stringstream

```
#include <string>
#include <sstream>
#include <iostream>


using namespace std;


int main() {
   string linha="Olah mundo";
   stringstream separador(linha); // Tokenizador
   string word;
   separador >> word; // word=Olah
   char mu[4];
   separador.readsome(mu, 3); // Le ” mu” (inclui espaco)
   cout << separador.tellg() << endl; // 7 (posicao de leitura)
   cout << separador.tellp() << endl; // 0, posicao de escrita

   separador.seekp(separador.str().size());
   separador << " cruel"; // separador = ”Olah mundo cruel”

   separador.seekp(5);
   separador << "doido"; // separador = ”Olah doido cruel” (sobrescrito)
   return 0;
}
```

Código 15: exemplo de ordenação

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool myfunction (int i,int j) { return (i<j); }

struct myclass {
  bool operator() (int i,int j) { return (i<j);}
} myobject;

int compare (const void * a, const void * b)
{
  return ( *(int*)a - *(int*)b );
}


int main () {
  int myints[] = {32,71,12,45,26,80,53,33};
  vector<int> myvector (myints, myints+8);               // 32 71 12 45 26 80 53 33

  // using default comparison (operator <):
  sort (myvector.begin(), myvector.begin()+4);           //(12 32 45 71) 26 80 53 33
  // using function as comp
  sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45 71(26 33 53 80)
  // using object as comp
  sort (myvector.begin(), myvector.end(), myobject);     //(12 26 32 33 45 53 71 80)

  // if stable is need
  stable_sort (myvector.begin(), myvector.end(), myfunction);

  // Rearranges the elements in the range [first,last), in such a way that the subrange [first,middle)
  // contains the smallest elements of the entire range sorted in ascending order, and the subrange
  // [middle,end) contains the remaining elements without any specific order.
  partial_sort (myvector.begin(), myvector.begin()+3, myvector.end());

  qsort (myints, 8, sizeof(int), compare);

  return 0;
}
```

Código 16: pesquisa binária

```
int compareMyType (const void * a, const void * b)
{
  if ( *(MyType*)a >  *(MyType*)b ) return 1;
  if ( *(MyType*)a == *(MyType*)b ) return 0;
  if ( *(MyType*)a <  *(MyType*)b ) return -1;
}

int key = 40;
```

```
item = (int*) bsearch (&key, values, n, sizeof(int), compareMyType);
```

Código 17: Arredondamento e output em outras bases

```
#include <iostream>
#include <iomanip> // setprecision()
using namespace std;

int main () {
  double a = 3.1415926534;
  double b = 2006.0;
  double c = 1.0e-10;

  // setprecision(1) => 1 casa decimal apos a virgula
  cout << fixed << setprecision(1) << 9.09090901 << endl;
  cout << fixed << setprecision(2) << 9.09090901 << endl;
  cout << fixed << setprecision(3) << 9.09090901 << endl;
  cout << fixed << setprecision(2) << 9.1 << endl;

  // anula o efeito de setprecision
  cout.unsetf(ios::floatfield);

  // 5 digitos no maximo
  cout.precision(5);

  cout << a << '\t' << b << '\t' << c << endl;
  cout << fixed << a << '\t' << b << '\t' << c << endl;
  cout << scientific << a << '\t' << b << '\t' << c << endl;

  // Sets the basefield format flag for the str stream to dec, hex or oct.
  int n =70;
  cout << dec << n << endl;
  cout << hex << n << endl;
  cout << oct << n << endl;

  return 0;
}
/* output
9.1
9.09
9.091
9.10
3.1416   2006     1e-10
3.14159  2006.00000       0.00000
3.14159e+00       2.00600e+03       1.00000e-10
70
46
106
*/
```

2.2 Teoria dos números

Código 18: máximo divisor comum e mínimo multiplo comum

```
int gcd(int x, int y)
{
   return y ? gcd(y, x % y) : abs(x);
}
uint64_t lcm(int x, int y)
{
   if (x && y) return abs(x) / gcd(x, y) * uint64_t(abs(y));
   else return uint64_t(abs(x | y));
}
```

Código 19: decide se um número é primo

```
bool isPrime(int n)
{
   if (n < 0) return isPrime(-n);
   if (n == 1) return true;
   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;
}
```

Código 20: Retorna a fatoração em números primos de abs(n).

```
typedef map<int, int> prime_map;
void squeeze(prime_map& M, int& n, int p)
{
   for (; n % p == 0; n /= p) M[p]++;
}
void factor(int n, prime_map& M)
{
   if (n < 0) { factor(-n, M); return; }
   if (n < 2) return;

   squeeze(M, n, 2);
   squeeze(M, n, 3);

   int maxP = sqrt(n) + 2;
   for (int p = 5; p < maxP; p += 6)
   {
      squeeze(M, n, p);
      squeeze(M, n, p+2);
   }
   if (n > 1) M[n]++;
}
```

Código 21: Calcula Valor de $a^b mod$ n de forma rápida.

```
int mpow(int a, int b, int n = 10)
{
  if(b == 0)
    return 1;
  else {
    long long res = mpow(a, b/2, n);
    res = (res*res) % n;
    if(b%2 == 1)
      res = (res*a) % n;
    return (int) res;
  }
}
```

Código 22: Calcula (a*b)%c de forma rápida.

```
long long mulmod(long long a, long long b, long long c)
{
    long long x = 0;
    long long y = a % c;
    while(b > 0)
    {
        if(b & 1ll) x = (x + y) % c;
        y = (y << 1) % c;
        b >>= 1;
    }
    return x % c;
}
```

Código 23: Computa x tal que a*x = b (mod c). Quando a equação não tem solução, retorna algum valor arbitrário errado, mas basta conferir o resultado.

```
long long axbmodc(long long a, long long b, long long c)
{
    return a ? (axbmodc(c % a, (a - b % a) % a, a) * c + b) / a : 0;
}
```

Código 24: **Baby-step Giant-step algorithm** Calcula o menor valor de e para $b^e = n$ mod p. Retorna -1 se eh impossivel

```
#define inv_mult( a, n ) axbmodc(a, 1, n)

long long discreteLlogarithm( long long b, long long n, long long p )
{
    if ( n == 1 ) return 0;

    map < long long, int > table;
    long long m = sqrt(p) + 1, pot = 1, pot2 = 1;

    for (int j = 0; j < m; j++)
    {
        if ( pot == n ) return j;
        table[( n * inv_mult( pot, p ) ) % p] = j;
        pot = ( pot * b ) % p;
    }

    for (int i = 0; i < m; i++)
    {
        if ( table.find( pot2 ) != table.end() ) return i * m + table[pot2
            ];
        pot2 = ( pot * pot2 ) % p;
    }

    return -1;
}
```

2.3 Estruturas de dados

Código 25: Números de precisão harbitrária.

```
const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 1000;

struct BigInt
{
    int num[TAM], numDigits;
    BigInt(int x = 0): numDigits(1)
    {
        memset(num, 0, sizeof(num));
        num[numDigits++] = x; fixInvariant();
    }
    BigInt(char *s): numDigits(1)
    {
        memset(num, 0, sizeof(num));
        int sign = 1;

        while (*s && !isdigit(*s))
        {
            if (*s++ == '-') sign *= -1;
        }

        char *t = strdup(s), *p = t + strlen(t);

        while (p > t)
        {
            *p = 0; p = max(t, p - DIG);
            sscanf(p, "%d", &num[numDigits]);
            num[numDigits++] *= sign;
        }

        free(t);
        fixInvariant();
    }

    BigInt& fixInvariant(int m = 0)
    {
        numDigits = max(m, numDigits);
        int sign = 0;

        for (int i = 1, carry = 0; i <= numDigits || carry && (numDigits = i); i++)
        {
            num[i] += carry;
            carry = num[i] / BASE;
            num[i] %= BASE;
            if (num[i]) sign = (num[i] > 0) ? 1 : -1;
        }

        for (int i = 1; i < numDigits; i++)
        {
            if (num[i] * sign < 0)
            {
                num[i] += sign * BASE;
                num[i+1] -= sign;
            }
        }

        while (numDigits && !num[numDigits]) numDigits--;
        return *this;
    }

    //Comparacao
    int cmp(const BigInt& x = 0) const
    {
        int i = max(numDigits, x.numDigits), t = 0;
        while (1)
        {
            if ((t = ::cmp(num[i], x.num[i])) || i-- == 0) return t;
        }
    }

    bool operator <(const BigInt& x) const { return cmp(x) < 0; }
    bool operator >(const BigInt& x) const { return cmp(x) > 0; }
    bool operator <=(const BigInt& x) const { return cmp(x) <= 0; }
    bool operator >=(const BigInt& x) const { return cmp(x) >= 0; }
    bool operator ==(const BigInt& x) const { return cmp(x) == 0; }
    bool operator !=(const BigInt& x) const { return cmp(x) != 0; }

    //operacoes fundamentais
    BigInt& operator +=(const BigInt& x)
    {
        for (int i = 1; i <= x.numDigits; i++) num[i] += x.num[i];
        return fixInvariant(x.numDigits);
    }
    BigInt& operator -=(const BigInt& x)
    {
        for (int i = 1; i <= x.numDigits; i++) num[i] -= x.num[i];
        return fixInvariant(x.numDigits);
    }

    void multiAndAcumWithShift(const BigInt& x, int m, int b)
    { // *this += (x * m) << b;
        for (int i = 1, carry = 0; (i <= x.numDigits || carry) && (numDigits = i + b); i++)
        {
            num[i+b] += x.num[i] * m + carry;
            carry = num[i+b] / BASE;
            num[i+b] %= BASE;
        }
    }

    BigInt operator *(const BigInt& x) const
    {
        BigInt r;
        for (int i = 1; i <= numDigits; i++) r.multiAndAcumWithShift(x, num[i], i-1);
        return r;
    }

    BigInt div(const BigInt& x)
    {
        if (x == 0) return 0;

        BigInt q;
        q.numDigits = max(numDigits - x.numDigits + 1, 0);
        int d = x.num[x.numDigits] * BASE + x.num[x.numDigits-1];

        for (int i = q.numDigits; i > 0; i--)
        {
            int j = x.numDigits + i - 1;
            q.num[i] = int((num[j] * double(BASE) + num[j-1]) / d);
            multiAndAcumWithShift(x, -q.num[i], i-1);
            if (i == 1 || j == 1) break;
            num[j-1] += BASE * num[j];
            num[j] = 0;
        }

        fixInvariant(x.numDigits);
        return q.fixInvariant();
    }

    BigInt& operator *=(const BigInt& x) { return *this = (*this) * x; }
    BigInt operator +(const BigInt& x) { return BigInt(*this) += x; }
    BigInt operator -(const BigInt& x) { return BigInt(*this) -= x; }
    BigInt operator -() { BigInt r = 0; return r -= *this; }
    BigInt& operator /=(const BigInt& x) { return *this = div(x); }
    BigInt& operator %=(const BigInt& x) { div(x); return *this; }
    BigInt operator /(const BigInt& x) { return BigInt(*this).div(x); }
    BigInt operator %(const BigInt& x) { return BigInt(*this) %= x; }

    // I/O
    operator string() const
    {
        ostringstream s; s << num[numDigits];
        for (int i = numDigits - 1; i > 0; i--)
        {
            s.width(DIG);
            s.fill('0');
            s << abs(num[i]);
        }

        return s.str();
    }

    friend ostream& operator <<(ostream& o, const BigInt& x)
    {
        return o << (string) x;
    }

    friend istream& operator >>(istream& in, BigInt& x)
    {
        string num;
        in >> num;
        x = BigInt((char*) num.c_str());
        return in;
    }

    // potencia e raiz
    BigInt pow(int x)
    {
        if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
        BigInt r = 1;
        for (int i = 0; i < x; i++) r *= *this;
        return r;
    }

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

        return a;
    }
};
```

2.4 Programação Dinâmica

Código 26: **Sub Set Sum**: Verifica se há um sobconjunto dos elementos do vetor cuja soma seja igual a soma pedida.

```
//soma maxima dos elementos do vetor
#define MAX_SUM 10000
int n;
int vet[TAM];
bool m[MAX_SUM];

//M->soma maxima dos elementos do vetor c->soma procurada
bool subSetSum(int M, int c)
{
    for (int i = 0; i <= M; i++) m[i] = false;
    m[0] = true;

    for(int i = 0; i < n; i++)
    {
        for(int j = M; j >= vet[i]; j--)
        {
            m[j] |= m[j - vet[i]];
        }
    }

    return m[c];
}
```

Código 27: **Lis: longest increasing (decreasing) subsequence** O(n^2)

```
#define TAM 10000
int c[TAM];
int A[TAM];
int H[TAM];

void ssctf(int n)
{
    for (int m = 1; m <= n; m++)
    {
        c[m] = H[m];
        for (int i = m -1; i > 0; i--)
        {
            if (A[i] < A[m] && c[i] + H[m] > c[m])
            {
                c[m] = c[i] + H[m];
            }
        }
    }
}

void ssdtf(int n)
{
    for (int m = 1; m <= n; m++)
    {
        c[m] = H[m];
        for (int i = m -1; i > 0; i--)
        {
            if (A[i] > A[m] && c[i] + H[m] > c[m])
            {
                c[m] = c[i] + H[m];
            }
        }
    }
}

int lis1d(int n, bool inc = true)
```

```
{
    if (inc) ssctf(n);
    else ssdtf(n);

    int max = 0;

    for (int i = 1; i <= n; i++)
        if (max < c[i])
            max = c[i];

    return max;
}
```

Código 28: **Lis: longest increasing subsequence** O(n*logn)

```
// Longest Increasing Subsequence - LIS O(n log n)
#define fori(i, n) for ( int i = 0; i < (n); ++i )
void lis( const vector< int > & v, vector< int > & asw )
{
    vector<int> pd(v.size(),0), pd_index(v.size()), pred(v.size());
    int maxi = 0, x, j, ind;

    fori(i,v.size())
    {
        x = v[i];
        j = lower_bound( pd.begin(), pd.begin() + maxi, x ) - pd.begin();
        //j = upper_bound( pd.begin(), pd.begin() + maxi, x ) - pd.begin() ; para lds
        pd[j] = x;
        pd_index[j] = i;
        if( j == maxi ) { maxi++; ind = i; }
        pred[i] = j ? pd_index[j-1] : -1;
    }
    // return maxi; se a sequencia nao precisa ser refeita

    int pos = maxi-1, k = v[ind];
    asw.resize( maxi );

    while ( pos >= 0 )
    {
        asw[pos--] = k;
        ind = pred[ind];
        k = v[ind];
    }
}
```

2.5 Grafos

Código 29: Verifica se o grafo é aciclico.

```
#define TAM 100
#define BRANCO 0
#define CINZA 1
#define PRETO 2
bool grafo[TAM][TAM];
int pass[TAM];

bool dfs(int v)
{
    pass[v] = CINZA;

    for (int i = 0; i < TAM; i++)
    {
        if (grafo[v][i])
        {
            if (pass[i] == CINZA) return false;
            if (pass[i] == BRANCO && !dfs(i)) return false;
        }
    }

    pass[v] = PRETO;
    return true;
}

bool aciclico()
{
    memset(pass, BRANCO, TAM*sizeof(int));

    for (int i = 0; i < TAM; i++)
    {
        if (pass[i] == BRANCO)
        {
            if (!dfs(i)) return false;
        }
    }

    return true;
}
```

Código 30: **Dijkstra** Caminho minimo 1 para todos pesos positivos.

```
#include <queue>

typedef vector<map<int, int> > AdjList;
typedef AdjList Grafo;

int dist[MAX_VERTICES];
int prev[MAX_VERTICES]; // para recuperar o caminho usando um dijoint forest set

void dijkstra(Grafo& grafo, int source)
{
    for (int i = 0; i < grafo.size(); i++)
    {
        dist[i] = INF;
        prev[i] = -1;
    }

    dist[source] = 0;
    priority_queue<pair<int, int> > heap;
    heap.push(make_pair(0, source));

    while (!heap.empty())
    {
        int u = heap.top().second;
        heap.pop();

        // para cada vizinho de u
        for (map<int,int>::iterator i = grafo[u].begin(); i != grafo[u].end(); i++)
        {
            int totalDist = dist[u] + (*i).second;
            if (totalDist <= dist[(*i).first])
            {
                dist[(*i).first] = totalDist;
                heap.push(make_pair(totalDist, (*i).first));
                prev[(*i).first] = u;
            }
        }
    }
}
```

Código 31: Floresta dijunta de arvores

```
#define SIZE 100

struct dsf
{
     int element_count;
     int parent[SIZE];
     int rank[SIZE];
};
typedef struct dsf * disjoint_set_forest_p;

void dsf_init(disjoint_set_forest_p forest, int element_count)
{
    forest->element_count = element_count;
    memset(forest->parent, 0, element_count*sizeof(int));
    memset(forest->rank, 0, element_count*sizeof(int));

    for (int i = 0; i < element_count; ++i)
        forest->parent[i] = i;
}

int dsf_find_set(disjoint_set_forest_p forest, int i)
{
     if (i != forest->parent[i])
     {
         forest->parent[i] = dsf_find_set(forest, forest->parent[i]);
     }
     return forest->parent[i];
}

void dsf_union(disjoint_set_forest_p forest, int i, int j)
{
     int x = dsf_find_set(forest, i);
     int y = dsf_find_set(forest, j);

     if (forest->rank[x] > forest->rank[y])
     {
         forest->parent[y] = x;
     }
     else
     {
         forest->parent[x] = y;
         if (forest->rank[x] == forest->rank[y])
         {
             forest->rank[y]++;
         }
     }
}
```

Código 32: **Kruskal** Arvore geradora mínima kruskal

```
typedef vector<map<int, int> > AdjList;
struct Grafo
{
    int edgeCnt;
    AdjList adj;
};

struct edge
{
     int u;
     int v;
     int weight;
};

int edge_compare(const void * e1, const void * e2)
{
     struct edge * p1 = (struct edge *) e1;
     struct edge * p2 = (struct edge *) e2;
     int f = p1->weight - p2->weight;
     if (f < 0)
     {
         return -1;
     }
     else  if (f == 0)
     {
         return edge_compare1(e1, e2);
     }
     else
     {
         return 1;
     }
}

struct edge * get_edge_list(Grafo& graph)
{
     int edge_count = graph.edgeCnt;
    struct edge *edges = (struct edge*) malloc(edge_count * sizeof(struct edge));

    int current_edge = 0;

    for (int i = 0; i < graph.adj.size(); ++i)
    {
        for (map<int, int>::iterator j = graph.adj[i].begin(); j != graph.adj[i].end(); j++)
        {
            struct edge e;
            e.u = i < (*j).first ? i : (*j).first;
            e.v = i > (*j).first ? i : (*j).first;
            e.weight = (*j).second;
            edges[current_edge++] = e;
        }
    }

    return edges;
}

void kruskal(Grafo& graph, Grafo& mst)
{
    // Obtain a list of edges and sort it by weight in O(E lg E) time
    int edge_count = graph.edgeCnt;
    struct edge *edges = get_edge_list(graph);
    qsort(edges, edge_count, sizeof(struct edge), edge_compare);

    disjoint_set_forest dsf;
   dsf_init(&dsf, edge_count);

    for (int i = 0; i < edge_count; ++i)
    {
        struct edge e = edges[i];
        int uset = dsf_find_set(dsf, e.u);
        int vset = dsf_find_set(dsf, e.v);
        if (uset != vset)
        {
            mst.adj[e.u][e.v] = e.weight;
         mst.edgeCnt++;
            dsf_union(dsf, uset, vset);
        }
    }

    free(edges);
}
```

Código 33: verifica se um grafo é bipartido

```
#define TAM 200

bool grafo[TAM][TAM];
int pass[TAM];
int n;

bool bipartido(int v, int color = 1)
{
    pass[v] = color;
    int thisColor = color;
    bool ret = true;

    color = color == 1 ? 2 : 1;

    for (int i = 0; i < n; i++)
    {
        if (grafo[v][i])
        {
            if (!pass[i]) ret = dfs(i, color);
            else if (pass[i] == thisColor) return false;

            if (!ret) return false;
        }
    }

    return ret;
}
```

Código 34: faz a ordenação topológica de um grafo acíclico

```
#define UNVISITED -1

int grafo[SIZE][SIZE];
int prof[SIZE];
int sorted[SIZE];
int nordem;

void dfsTopsort(int no)
{
    for (int viz = 0; viz < SIZE; viz++)
    {
        if (grafo[no][viz])
        {
            if (prof[viz] == UNVISITED)
            {
                prof[viz] = prof[no] + 1;
                dfsTopsort(viz);
            }
        }
    }

    sorted[nordem--] = no;
}

void topSort(int nvt)
{
    memset(prof, UNVISITED, nvt*sizeof(int));
    nordem = nvt - 1;

    for (int i = 0; i < nvt; i++)
    {
        if (prof[i] == UNVISITED)
        {
            prof[i] = 0;
            dfsTopsort(i);
        }
    }
}
```

Código 35: calcula fluxo máximo, **Ford-Fulkerson**

```
#define TAM 1000
#define MAX_INT 1000000

int grafo[TAM][TAM];
int pred[TAM];
int f[TAM][TAM];
bool visitados[TAM];
int fila[TAM];

bool bfs(int n, int ini, int fim)
{
    int no, s = 0, e = 0;
    fila[e++] = ini;

    while (s != e)
    {
        no = fila[s++];

        if (visitados[no] == 2) continue;
        visitados[no] = 2;

        for (int i = 0; i < n; i++)
        {
            if (visitados[i] < 2)
            {
                if(grafo[no][i] - f[no][i] > 0)
                {
                    pred[i] = no;
                    if (i == fim) return true;
                    if(visitados[i] == 0)
                    {
                        fila[e++] = i;
                        visitados[i] = 1;
                    }
                }
            }
        }
    }

    return false;
}

bool dfs(int s, int t, int size)
{
    visitados[s] = true;
    if(s == t) return true;

    for(int v = 0; v < size; v++)
    {
        if(!visitados[v] && grafo[s][v] - f[s][v] > 0)
        {
            pred[v] = s;
            if(dfs(v, t, size)) return true;
        }
    }

    return false;
}

bool findPath(int s, int t, int size)
{
    memset(visitados, false, sizeof(bool)*size);
    pred[s] = s;
    // Aqui pode ser usado tanto busca em largura quanto em profundidade.
    // busca em largura geralmente apresenta tempos de execucao bem menores.
    return bfs(size, s, t);
    //return dfs(s, t, size);
}

int maxFlow(int size, int s, int t)
{
    int delta;

    for(int i = 0; i < size; i++)
    {
        memset(f[i], 0, sizeof(int)*size);
    }

    while(1)
    {
        bool path = findPath(s, t, size);
        if (!path) break;

        delta = MAX_INT;
        for(int c = t; pred[c] != c; c = pred[c])
        {
            delta = min(delta, grafo[pred[c]][c] - f[pred[c]][c]);
        }

        for(int c = t; pred[c] != c; c = pred[c])
        {
            f[pred[c]][c] += delta;
            f[c][pred[c]] -= delta;
        }
    }

    int soma = 0;

    for(int i = 0; i < size; i++)
    {
        soma += f[i][t];
    }

    return soma;
}
```

Código 36: calcula fluxo máximo, algoritmo mais eficiente porém muito maior em tempo de codificação

```
const int VT = 100;
const int AR = VT * VT;

struct grafo
{
    // lista de adjacencias representada na forma de vetor
    int nvt, nar;
    int dest[2 * AR];
    int adj[VT][2 * VT];
    int nadj[VT];

    int cap[AR]; // capacidade do arco
    int fluxo[AR];
    int ent[VT];

    int padj[VT], lim[VT], nivel[VT], qtd[VT];

    int inv(int a) { return a ^ 0x1; }
    int orig(int a) { return dest[inv(a)]; }
    int capres(int a) { return cap[a] - fluxo[a]; }

    void inic(int n = 0)
    {
        nvt = n;
        nar = 0;
        memset(nadj, 0, sizeof(nadj));
    }

    //////////////////////////////////////////////////////////
    // Adiciona uma aresta ao grafo.
    //
    // "int u" apenas para Fluxos;
    //
    int aresta(int i, int j, int u = 0)
    {
        int ar = nar;
        cap[nar] = u;
        dest[nar] = j;
        adj[i][nadj[i]] = nar++;
        nadj[i]++;

        cap[nar] = 0;
        dest[nar] = i;
        adj[j][nadj[j]] = nar++;
        nadj[j]++;
        return ar;
    }

    void revbfs(int ini, int fim)
    {
        int i, no, viz, ar;
        queue<int> fila;

        memset(nivel, NULO, sizeof(nivel));
        memset(qtd, 0, sizeof(qtd));

        nivel[fim] = 0;
        fila.push(fim);

        while (!fila.empty())
        {
            no = fila.front();
            fila.pop();
            qtd[nivel[no]]++;

            for (i = 0; i < nadj[no]; i++)
            {
                ar = adj[no][i];
                viz = dest[ar];

                if (cap[ar] == 0 && nivel[viz] == NULO)
                {
                    nivel[viz] = nivel[no] + 1;
                    fila.push(viz);
                }
            }
        }
    }

    int admissivel(int no)
    {
        while (padj[no] < nadj[no])
        {
            int ar = adj[no][padj[no]];
            if (nivel[no] == nivel[dest[ar]] + 1 && capres(ar) > 0) return
                 ar;
            padj[no]++;
        }

        padj[no] = 0;
        return NULO;
    }

    int retrocede(int no)
    {
        int i, ar, viz, menor = NULO;

        if (--qtd[nivel[no]] == 0) return NULO;

        for (i = 0; i < nadj[no]; i++)
        {
            ar = adj[no][i]; viz = dest[ar];
            if (capres(ar) <= 0) continue;
            if (menor == NULO || nivel[viz] < nivel[menor]) menor = viz;
        }

```

```
      if (menor != NULO) nivel[no] = nivel[menor];
      qtd[++nivel[no]]++;

      return ((ent[no] == NULO) ? no : orig(ent[no]));
   }

   int avanca(int no, int ar)
   {
      int viz = dest[ar];
      ent[viz] = ar;
      lim[viz] = min(lim[no], capres(ar));
      return viz;
   }

   int aumenta(int ini, int fim)
   {
      int ar, no = fim, fmax = lim[fim];

      while (no != ini)
      {
         fluxo[ar = ent[no]] += fmax;
         fluxo[inv(ar)] -= fmax;
         no = orig(ar);
      }

      return fmax;
   }

   int maxflow(int ini, int fim)
   {
      int ar, no = ini, fmax = 0;

      memset(fluxo, 0, sizeof(fluxo));
      memset(padj, 0, sizeof(padj));

      revbfs(ini, fim);

      lim[ini] = INF;
      ent[ini] = NULO;

      while (nivel[ini] < nvt && no != NULO)
      {
         if ((ar = admissivel(no)) == NULO)
         {
            no = retrocede(no);
         }
         else if ((no = avanca(no, ar)) == fim)
         {
            fmax += aumenta(ini, fim);
            no = ini;
         }
      }
      return fmax;
   }
};
```

2.6 Geometria

Código 37: ponto e poligono

```
struct point
{
   double x, y;
   double z; // para pontos no espaco
   point(double x = 0, double y = 0, double z = 0): x(x), y(y), z(z) {}

   point operator +(point q) { return point(x + q.x, y + q.y, z + q.z); }
   point operator -(point q) { return point(x - q.x, y - q.y, z - q.z); }
   point operator *(double t) { return point(x * t, y * t, z * t); }
   point operator /(double t) { return point(x / t, y / t, z / t); }
   double operator *(point q) { return x * q.x + y * q.y + z * q.z; }
   point vec(point q) { return point(y * q.z - z * q.y, z * q.x - x * q.z, x * q.y - y * q.x); }
   double operator %(point q) { return x * q.y - y * q.x; }

   int cmp(point q) const
   {
      if (int t = ::cmp(x, q.x)) return t;
      else if (int t = ::cmp(y, q.y)) return t;
      return ::cmp(z, q.z);
   }

   bool operator ==(point q) const { return cmp(q) == 0; }
   bool operator !=(point q) const { return cmp(q) != 0; }
   bool operator < (point q) const { return cmp(q) < 0; }

   friend ostream& operator <<(ostream& o, point p) {
     return o << "(" << p.x << ", " << p.y << ", " << p.z << ")";
   }
   static point pivot;
};

// para pontos 2D
double abs(point p) { return hypot(p.x, p.y); }
double arg(point p) { return atan2(p.y, p.x); }

point point::pivot;

typedef vector<point> polygon;

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

double angle(point p, point q, point r)
{
   point u = p - q, v = r - q;
   return atan2(u % v, u * v);
}
```

Código 38: Decide se q está sobre o segmento fechado pr.

```
bool between(point p, point q, point r)
{
    return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
}
```

Código 39: Decide se os segmentos fechados pq e rs têm pontos em comum.

```
bool seg_intersect(point p, point q, point r, point s)
{
    point A = q - p;
    point B = s - r;
    point C = r - p;
    point D = s - q;

    int a = cmp(A % C) + 2 * cmp(A % D);
    int b = cmp(B % C) + 2 * cmp(B % D);

    if (a == 3 || a == -3 || b == 3 || b == -3) return false;
    if (a || b || p == r || p == s || q == r || q == s) return true;

    int t = (p < r) + (p < s) + (q < r) + (q < s);
    return t != 0 && t != 4;
}
```

Código 40: Calcula a distância do ponto r ao segmento pq.

```
double seg_distance(point p, point q, point r)
{
    point A = r - q;
    point B = r - p;
    point C = q - p;

    double a = A * A, b = B * B, c = C * C;

    if (cmp(b, a + c) >= 0) return sqrt(a);
    else if (cmp(a, b + c) >= 0) return sqrt(b);
    else return fabs(A % B) / sqrt(c);
}
```

Código 41: Classifica o ponto p em relação ao polígono T. Retorna 0, -1 ou 1 dependendo se p está no exterior, na fronteira ou no interior de T, respectivamente.

```
int in_poly(point p, polygon& T)
{
    double a = 0;
    int N = T.size();
    for (int i = 0; i < N; i++)
    {
        if (between(T[i], p, T[(i+1) % N])) return -1;
        a += angle(T[i], p, T[(i+1) % N]);
    }
    return cmp(a) != 0;
}
```

2.7 Casamento de strings

Código 42: String matching - Algoritmo **KMP** - O(n + m)

```
// F[i] - size of the largest prefix of pattern[0..i] that is also a
// suffix of pattern[1..i]. Ex: pattern = {a,b,a,c,a,b}, F =
    {0,0,1,0,1,2}
#define MAX_PATTERN_SIZE 10010
int F[MAX_PATTERN_SIZE];
void build_failure_function( const string & pattern )
{
    int m = pattern.size();
    F[0] = -1;
    for (int i = 0; i < m; i++)
    {
        F[i+1] = F[i] + 1;
        while ( F[i+1] > 0 && pattern[i] != pattern[ F[i+1]-1 ] )
            F[i+1] = F[ F[i+1]-1 ] + 1;
    }
}

// retorna a posicao inicial de cada ocorrencia de pattern em text
vector<int> KMP( const string & text, const string & pattern )
{
    build_failure_function( pattern );
    vector<int> start_positions;
    int j = 0, m = pattern.size(), n = text.size();

    for (int i = 0; i < n; i++)
    {
        while ( true )
        {
            if ( text[i] == pattern[j] )
            {
                if ( ++j == m )
                {
                    start_positions.push_back( i - m + 1 );
                    j = F[j];
                }
                break;
            }

            if ( j == 0 ) break;
            j = F[j];
        }
    }

    return start_positions;
}
```

2.8 Outros

Código 43: josephus problem

```cpp
/**
The Josephus problem (or Josephus permutation) is a theoretical problem related to a certain counting-out game. There are people standing in a circle waiting to be executed. After the first man is executed, certain number of people are skipped and one man is executed. Then again, people are skipped and a man is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last man remains, who is given freedom. The task is to choose the place in the initial circle so that you are the last one remaining and so survive.
 */

using namespace std;

int josephus(int n, int m)
{
    int res = 0;
    vector <int> people;
    int loc = 0;

    for (int i = 0; i < n; i++) people.push_back(i+1);

    while (people.size() > 1)
    {
        if (loc >= people.size())
            loc %= people.size();

        people.erase(people.begin()+loc);
        loc += (m-1);
    }

    return people[0];
}
```

Código 44: Simplex

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

#define fori(i,n) for(int i=0; i < (n); ++i)
#define forr(i,a,b) for(int i=(a); i <= (b); ++i)
#define ford(i,a,b) for(int i=(a); i >= (b); --i)
#define sz size()

const double EPS=1e-9; const int INF = 0x3f3f3f3f;

#define all(x) (x).begin(),(x).end()

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

struct simplex {
    // max c * x, s.t: A * x <= b; x >= 0
    simplex( const vector< vector< double > > & A_, const vector< double > & b_,
            const vector< double > & c_ ) : A( A_ ), b( b_ ), c( c_ ) {}
    vector< vector< double > > A; vector< double > b, c, sol;
    vector< bool > N; vector< int > kt; int m, n;
    void pivot( int k, int l, int e ) {
        int x = kt[l]; double p = A[l][e];
        fori(i,k) A[l][i] /= p;
        b[l] /= p; N[e] = false;
        fori(i,m) if (i != l) {b[i] -= A[i][e]*b[l]; A[i][x] = -A[i][e]*A[l][x];}
        fori(j,k) if ( N[j] ) {
            c[j] -= c[e] * A[l][j];
            fori(i,m) if ( i != l ) A[i][j] -= A[i][e] * A[l][j];
        }
        kt[l] = e; N[x] = true; c[x] = -c[e] * A[l][x];
    }
    vector< double > go( int k ) {
        vector< double > res;
        while ( 1 ) {
            int e = -1, l = -1;
            fori(i,k) if ( N[i] && cmpD( c[i] ) > 0 ) { e = i; break; }
            if ( e == -1 ) break;
            fori(i,m) if ( cmpD(A[i][e]) > 0 && ( l == -1 || cmpD( b[i] / A[i][e],
                          b[l] / A[l][e], 1e-20 ) < 0 ) ) l = i;
            if ( l == -1 ) return vector< double >(); // unbounded
            pivot( k, l, e );
        }
        res.resize( k, 0 );
        fori(i,m) res[kt[i]] = b[i];
        return res;
    }
    vector< double > solve() {
       m = A.sz; n = A[0].sz; int k = m+n+1;
       N = vector< bool >( k, true ); vector< double > c_copy = c;
       c.resize(n+m); kt.resize(m);
       fori(i,m) {
          A[i].resize(k); A[i][n+i] = 1; A[i][k-1] = -1;
          kt[i] = n+i; N[kt[i]] = false;
       }
       int l = min_element(all(b)) - b.begin();
       if(cmpD(b[1] ) < 0 ) {
          c = vector<double>(k,0);
          c[k-1] = -1; pivot(k, 1, k-1); sol=go(k);
          if(cmpD(sol[k-1])>0) return vector<double>(); // infeasible
          fori(i,m) if(kt[i] == k-1) {
             fori(j,k-1) if(N[j] && cmpD( A[i][j] ) != 0 ) {
                pivot( k, i, j); break;
             }
            }
            c=c_copy; c.resize(k,0);
            fori(i,m) fori(j,k) if(N[j]) c[j] -= c[kt[i]]*A[i][j];
        }
        sol = go(k-1);
        if(!sol.empty()) sol.resize(n);
        return sol;
    }
};

// Como usar
int main() {
    /* Exemplo: Maximize cx Subject to Ax <= b */
    vector<vector<double> > A(9);
    double Av[][3] = {{1,1,0}, {0,0,-1}, {-1,-1,0},
                      {0,0,1}, {1,0,0}, {0,1,0},
                      {0,0,1}, {1,0,1}, {0,1,0}};

    for(int i=0; i < 9; i++) {
        A[i].insert(A[i].begin(), &(Av[i][0]), &(Av[i][3])); // Sim, [3]! Ou seja, idx-final+1
    }

    vector<double > c(3, 1); // c=[1 1 1]
    double bv[] = {2,-1,-2,1,2,1,1,2,1};
    vector<double > b(bv, bv+sizeof(bv)/sizeof(double));

    simplex sim(A,b,c);
    vector<double> s = sim.solve();
    if(!s.size()) cout << "Impossivel\n";
    else
    for(int i=0; i < s.size(); i++) {
        cout << s[i] << endl;
    }
}
```

Código 45: Gera as permutações dos elementos da string

```
bool nextPermutation(string& number)
{
        bool isBigger = true;
        int i, j;

        for (i = number.size() - 1; i >= 0; i--)
        {
                if (number[i] < number[i+1]) break;
        }

        if (i != -1)
        {
                isBigger = false;

                for (j = number.size() - 1; j >= i+1; j--)
                {
                        if (number[j] > number[i])
                        {
                                break;
                        }
                }

                int tmp = number[i];
                number[i] = number[j];
                number[j] = tmp;

                j = number.size() -1;
                i++;

                while (i < j)
                {
                        tmp = number[i];
                        number[i] = number[j];
                        number[j] = tmp;
                        i++;
                        j--;
                }
        }

        return isBigger;
}
```

3 Biblioteca C/C++

3.1 I/O

Ignorando entradas na família scanf:

Código 46: Ignora os dois floats do meio. Retornará 2 no sucesso.

```
scanf("%f %*f %*f %d", &a, &b);
```

3.2 Map

Código 47: Referencias map

```
#include <map>
#include <string>
#include <cstdio>

using namespace std; // USE ISTO!!!

class Comparadora;

class Pessoa {
   int idade;
   string nome;
   friend class Comparadora;
public:
```

```cpp
   Pessoa(string nome, int idade) {
      this->idade = idade;
      this->nome = nome;
   }
   void print() const {
      printf("Nome: %s Idade: %d\n", nome.c_str(), idade);
   }
};

class Comparadora { // Ordena crescentemente
public: // <- IMPORTANTE
   bool operator() (const Pessoa &a, const Pessoa &b)
   {
      int idDif = a.idade-b.idade;
      if(idDif < 0) return true;
      else if(idDif==0) return a.nome.compare(b.nome) < 0 ? true : false;
      else return false;
   }
};

int main() {
   Pessoa r("Rangelz", 86);
   Pessoa r2("Rangelzao", 86);

   map<Pessoa, string, Comparadora> alunos;
   alunos[r]="UFMG";
   alunos[r2]="PUC";
   // Iterator
   for(map<Pessoa, string, Comparadora>::iterator it=alunos.begin(); it != alunos.end(); it++) {
      it->first.print();
      printf("\t%s\n\n", it->second.c_str());
   }
   // Find
   if(alunos.find(Pessoa("Rangelz", 86)) != alunos.end()) { // Achou!
      printf("Achei Rangel!\n");
   }
   return 0;
}
```