

Sumário

1	Tabelas	2
2	Codigos	3
2.1	Exemplos	3
2.2	Teoria dos números	8
2.3	Programação Dinâmica	9
2.4	Grafos	10
2.5	Geometria	15
2.6	Casamento de strings	16
2.7	Outros	16
3	Biblioteca C/C++	18
3.1	I/O	18
3.2	Map	18

Lista de Tabelas

1	Limites de representação de dados	2
2	Fatorial	2
3	scanf() - %[*][width][modifiers]type	2
4	scanf() %[*][width][modifiers]type	2
5	stdlib	2
6	math (angulos em radianos)	3

Algoritmos

1	Modelo	3
2	comparcao de ponto flutuante	3
3	.vimrc para a configuração do vim	4
4	função que acelara o cin. Não deve ser usada com printf	4
5	printf	4
6	exemplo de map	4
7	exemplo de set e multiset	4
8	exemplo de list	5
9	exemplo de queue	5
10	exemplo de priority queue	5
11	exemplo de stack	6
12	exemplo de vector	6
13	exemplo de string	6
14	exemplo de stringstream	7
15	exemplo de ordenação	7
16	pesquisa binária	7

17	Arredondamento e output em outras bases	8
18	máximo divisor comum e mínimo multiplo comum	8
19	decide se um número é primo	8
20	Retorna a fatoração em números primos de abs(n).	8
21	Calcula Valor de $a^b \bmod n$ de forma rápida.	8
22	Sub Set Sum : Verifica se há um subconjunto dos elementos do vetor cuja soma seja igual a soma pedida.	9
23	Lis: longest increasing (decreasing) subsequence $O(n^2)$	9
24	Lis: longest increasing subsequence $O(n \cdot \log n)$	9
25	Verifica se o grafo é aciclico.	10
26	Dijkstra Caminho minimo 1 para todos pesos positivos.	10
27	Floresta dijunta de arvores	10
28	Kruskal Arvore geradora mínima kruskal	11
29	verifica se um grafo é bipartido	12
30	faz a ordenação topológica de um grafo acíclico	12
31	calcula fluxo máximo, Ford-Fulkerson	12
32	calcula fluxo máximo, algoritmo mais eficiente porém muito maior em tempo de codificação	13
33	ponto e poligono	15
34	Decide se q está sobre o segmento fechado pr.	15
35	Decide se os segmentos fechados pq e rs têm pontos em comum.	15
36	Calcula a distância do ponto r ao segmento pq.	16
37	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.	16
38	String matching - Algoritmo KMP - $O(n + m)$	16
39	josephus problem	16
40	Simplex	17
41	Gera as permutações dos elementos da string	18
42	Ignora os dois floats do meio. Retornará 2 no sucesso.	18
43	Referencias map	18

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

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <math.h>
5
6 #include <inttypes.h>
7 #include <ctype.h>
8 #include <limits.h>
9
10 #include <algorithm>
11 #include <utility>
12 #include <iostream>
13
14 #include <map>
15 #include <set>
16 #include <vector>
17 #include <list>
18 #include <queue>
19 #include <sstream>
20
21 using namespace std;
22
23 #define abs(a) ((a) > 0 ? (a) : -(a))
24
25 int main()
26 {
27     int n;
28
29     cin >> n;
30
31     for (int i = 0; i < n; i++)
32     {
33
34     }
35
36     while (cin >> n)
37     {
38
39     }
40     return 0;
41 }

```

Código 2: comparcao de ponto flutuante

```

1 const double EPS = 1e-10;
2 /**
3  * -1 se x < y
4  * 0 se x = y
5  * 1 se x > y

```

```

6  */
7  inline int cmp (double x, double y = 0, double tol = EPS)
8  {
9      return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
10 }

```

Código 3: .vimrc para a configuração do vim

```

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

```

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

```

1 std::cout.sync_with_stdio(false);

```

Código 5: printf

```

1  /* printf example */
2  #include <stdio.h>
3
4  int main()
5  {
6      printf ("Characters: %c %c \n", 'a', 65);
7      printf ("Decimals: %d %ld\n", 1977, 650000L);
8      printf ("Preceding with blanks: %10d \n", 1977);
9      printf ("Preceding with zeros: %010d \n", 1977);
10     printf ("Some different radixes: %d %x %o %#x %#o \n", 100, 100, 100,
11             100, 100);
12     printf ("floats: %4.2f %+0e %E %4.2f\n", 3.1416, 3.1416, 3.1416,
13             3.1);
14     printf ("Width trick: %*d \n", 5, 10);
15     printf ("%s \n", "A string");
16     return 0;
17 }
18
19 /* %[flags (-, +, etc)][width][.precision][length (h,l,L)]specifier
20 Characters: a A
21 Decimals: 1977 650000
22 Preceding with blanks:          1977
23 Preceding with zeros: 0000001977
24 Some different radixes: 100 64 144 0x64 0144
25 floats: 3.14 +3e+000 3.141600E+000 3.10
26 Width trick:      10
27 A string
28 */

```

Código 6: exemplo de map

```

1 #include <iostream>
2 #include <map>
3 using namespace std;
4
5 int main ()
6 {

```

```

7  map<char,int> mymap;
8  map<char,int>::iterator it;
9  pair<map<char,int>::iterator,bool> ret;
10
11  // first insert function version (single parameter):
12  mymap.insert ( pair<char,int>('a',100) );
13  mymap.insert ( pair<char,int>('z',200) );
14
15  ret=mymap.insert ( pair<char,int>('z',500) );
16  if (ret.second==false)
17  {
18      cout << "element 'z' already existed";
19      cout << " with a value of " << ret.first->second << endl;
20  }
21
22  // third insert function version (range insertion):
23  map<char,int> anothermap;
24  anothermap.insert(mymap.begin(),mymap.find('c'));
25
26  // showing contents:
27  cout << "mymap contains:\n";
28  for ( it=mymap.begin(); it != mymap.end(); it++ )
29      cout << (*it).first << " => " << (*it).second << endl;
30
31  map<char,string> mymap;
32  mymap['a']="an element";
33  if (mymap.count('a') > 0)
34      cout << mymap['a'] << " is an element of mymap.\n";
35
36  while (!mymap.empty())
37  {
38      cout << mymap.begin()->first << " => ";
39      cout << mymap.begin()->second << endl;
40      map<char, int>::iterator erasedelement = mymap.erase(mymap.begin());
41      ;
42  }
43
44  return 0;
45 }

```

Código 7: exemplo de set e multiset

```

1 #include <iostream>
2 #include <set>
3 using namespace std;
4
5 int main ()
6 {
7     multiset<int> mymultiset;
8     multiset<int>::iterator it;
9
10    // set some initial values:
11    for (int i=1; i<=5; i++) mymultiset.insert(i*10);    // 10 20 30 40 50
12
13    cout << "size: " << (int) mymultiset.size() << endl;
14    cout << "count: " << (int) mymultiset.count(10) << endl;

```

```

15     it=mymultiset.find(20);
16     mymultiset.erase (it);
17
18     if (! mymultiset.empty)
19         mymultiset.erase (mymultiset.find(40));
20
21     for (it=mymultiset.begin(); it!=mymultiset.end(); it++)
22         cout << " " << *it;
23
24     int myints[]={19,72,4,36,20,20};
25     multiset<int> first (myints,myints+3);    // 4,19,72
26     multiset<int> second (myints+3,myints+6); // 20,20,36
27
28     first.swap(second); // troca conteudo. o primeiro fica [20,20,36] e o
29                          segundo [4,19,72]
30
31     return 0;
32 }

```

Código 8: exemplo de list

```

1 #include <iostream>
2 #include <list>
3 using namespace std;
4
5 int main ()
6 {
7     list<int> mylist (2,100);           // two ints with a value of 100
8     mylist.push_front (200);
9     mylist.push_back (300);
10
11     it = mylist.begin();
12     mylist.insert (it,10);
13     mylist.insert (it,2,20); // two ints with a value of 20
14
15     mylist.reverse(); // Reverses the order of the elements in the list.
16
17     cout << "mylist contains:";
18     for (list<int>::iterator it=mylist.begin(); it!=mylist.end(); ++it)
19         cout << " " << *it;
20
21     cout << "Popping out the elements in mylist:";
22     while (!mylist.empty())
23     {
24         cout << " " << mylist.front();
25         mylist.pop_front();
26     }
27
28     while (!mylist.empty())
29     {
30         cout << " " << mylist.back();
31         mylist.pop_back();
32     }
33
34     cout << mylist.size() << endl;

```

```

35
36     return 0;
37 }

```

Código 9: exemplo de queue

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4
5 int main ()
6 {
7     queue<int> myqueue;
8     int sum (0);
9
10    for (int i=1;i<=10;i++) myqueue.push(i);
11
12    myqueue.back() -= myqueue.front();
13
14    cout << "size: " << (int) myqueue.size() << endl;
15
16    while (!myqueue.empty())
17    {
18        sum += myqueue.front();
19        myqueue.pop();
20    }
21
22    cout << "total: " << sum << endl;
23
24    return 0;
25 }

```

Código 10: exemplo de priority queue

```

1 #include <iostream>
2 #include <queue>
3 using namespace std;
4
5 int main ()
6 {
7     priority_queue<int> mypq;
8
9     mypq.push(30);
10    mypq.push(100);
11    mypq.push(25);
12    mypq.push(40);
13
14    cout << "size: " << (int) mypq.size() << endl;
15
16    cout << "Popping out elements...";
17    while (!mypq.empty())
18    {
19        cout << " " << mypq.top();
20        mypq.pop();
21    }

```

```

22  cout << endl;
23
24  return 0;
25 }

```

Código 11: exemplo de stack

```

1  #include <iostream>
2  #include <stack>
3  using namespace std;
4
5  int main ()
6  {
7      stack<int> mystack;
8      int sum = 0;
9
10     mystack.push(10);
11     mystack.push(20);
12
13     mystack.top() -= 5;
14
15     while (!mystack.empty())
16     {
17         sum += mystack.top();
18         mystack.pop();
19     }
20
21     cout << "size: " << (int) mystack.size() << endl;
22
23     return 0;
24 }

```

Código 12: exemplo de vector

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main ()
6  {
7      vector<int> myvector (3,100); // (100 100 100)
8      vector<int>::iterator it;
9
10     myvector.reserve(100);
11
12     for (int i=0; i<myvector.size(); i++)
13         myvector.at(i)=i; // = myvector[i] = i
14
15     it = myvector.begin();
16     it = myvector.insert ( it , 200 );
17     myvector.insert (it,2,300);
18
19     vector<int> anothervector (2,400);
20     int myarray [] = { 501,502,503 };

```

```

21     vector<int> initializer (myarray/*PointerInicio*/, myarray+sizeof(
        myarray)/sizeof(int)/*PointerFim*/);
22     myvector.insert (it+2,anothervector.begin(),anothervector.end());
23     myvector.insert (myvector.begin(), myarray, myarray+3);
24
25     cout << "myvector contains:";
26     for (it=myvector.begin(); it<myvector.end(); it++)
27         cout << " " << *it;
28     cout << endl;
29
30     // erase the 6th element
31     myvector.erase (myvector.begin()+5);
32     int sum;
33     while (!myvector.empty())
34     {
35         sum += myvector.back();
36         myvector.pop_back();
37     }
38
39     return 0;
40 }

```

Código 13: exemplo de string

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main ()
6  {
7      string str ("There are two needles in this haystack with needles.");
8      string str2 ("needle");
9      size_t found;
10
11     // different member versions of find in the same order as above:
12     found=str.find(str2);
13     if (found!=string::npos)
14         cout << "first 'needle' found at: " << int(found) << endl;
15
16     found=str.find("needles are small",found+1,6);
17     if (found!=string::npos)
18         cout << "second 'needle' found at: " << int(found) << endl;
19
20     found=str.find("haystack");
21     if (found!=string::npos)
22         cout << "'haystack' also found at: " << int(found) << endl;
23
24     found=str.find('.');
25     if (found!=string::npos)
26         cout << "Period found at: " << int(found) << endl;
27
28     // let's replace the first needle:
29     str.replace(str.find(str2),str2.length(),"preposition");
30     cout << str << endl;
31
32     string str="We think in generalities, but we live in details.";

```

```

33          // quoting Alfred N. Whitehead
34  string str2, str3;
35  size_t pos;
36
37  str2 = str.substr (12,12); // "generalities"
38
39  pos = str.find("live");    // position of "live" in str
40  str3 = str.substr (pos);   // get from "live" to the end
41
42  cout << str2 << ' ' << str3 << endl;
43
44
45  return 0;
46 }
47 /*
48 first 'needle' found at: 14
49 second 'needle' found at: 44
50 'haystack' also found at: 30
51 Period found at: 51
52 There are two prepositions in this haystack with needles.
53 generalities live in details.
54 */

```

Código 14: exemplo de stringstream

```

1 #include <string>
2 #include <sstream>
3 #include <iostream>
4
5
6 using namespace std;
7
8
9 int main() {
10  string linha="Olah mundo";
11  stringstream separador(linha); // Tokenizador
12  string word;
13  separador >> word; // word=Olah
14  char mu[4];
15  separador.readsome(mu, 3); // Le " mu" (inclui espaco)
16  cout << separador.tellg() << endl; // 7 (posicao de leitura)
17  cout << separador.tellp() << endl; // 0, posicao de escrita
18
19  separador.seekp(separador.str().size());
20  separador << " cruel"; // separador = "Olah mundo cruel"
21
22  separador.seekp(5);
23  separador << "doido"; // separador = "Olah doido cruel" (sobrescrito)
24  return 0;
25 }

```

Código 15: exemplo de ordenação

```

1 #include <iostream>
2 #include <algorithm>

```

```

3 #include <vector>
4 using namespace std;
5
6 bool myfunction (int i,int j) { return (i<j); }
7
8 struct myclass {
9  bool operator() (int i,int j) { return (i<j); }
10 } myobject;
11
12 int compare (const void * a, const void * b)
13 {
14  return ( *(int*)a - *(int*)b );
15 }
16
17
18 int main () {
19  int myints[] = {32,71,12,45,26,80,53,33};
20  vector<int> myvector (myints, myints+8); // 32 71 12 45
21                                           26 80 53 33
22  // using default comparison (operator <):
23  sort (myvector.begin(), myvector.begin()+4); // (12 32 45 71)
24                                           26 80 53 33
25  // using function as comp
26  sort (myvector.begin()+4, myvector.end(), myfunction); // 12 32 45
27                                           71(26 33 53 80)
28  // using object as comp
29  sort (myvector.begin(), myvector.end(), myobject); // (12 26 32 33
30                                           45 53 71 80)
31
32  // if stable is need
33  stable_sort (myvector.begin(), myvector.end(), myfunction);
34
35  // Rearranges the elements in the range [first,last), in such a way
36  // that the subrange [first,middle)
37  // contains the smallest elements of the entire range sorted in
38  // ascending order, and the subrange
39  // [middle,end) contains the remaining elements without any specific
40  // order.
41  partial_sort (myvector.begin(), myvector.begin()+3, myvector.end());
42
43  qsort (myints, 8, sizeof(int), compare);
44
45  return 0;
46 }

```

Código 16: pesquisa binária

```

1 int compareMyType (const void * a, const void * b)
2 {
3  if ( *(MyType*)a > *(MyType*)b ) return 1;
4  if ( *(MyType*)a == *(MyType*)b ) return 0;
5  if ( *(MyType*)a < *(MyType*)b ) return -1;
6 }
7
8 int key = 40;

```

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

Código 17: Arredondamento e output em outras bases

```
1 #include <iostream>
2 #include <iomanip> // setprecision()
3 using namespace std;
4
5 int main () {
6     double a = 3.1415926534;
7     double b = 2006.0;
8     double c = 1.0e-10;
9
10    // setprecision(1) => 1 casa decimal apos a virgula
11    cout << fixed << setprecision(1) << 9.09090901 << endl;
12    cout << fixed << setprecision(2) << 9.09090901 << endl;
13    cout << fixed << setprecision(3) << 9.09090901 << endl;
14    cout << fixed << setprecision(2) << 9.1 << endl;
15
16    // anula o efeito de setprecision
17    cout.unsetf(ios::floatfield);
18
19    // 5 digitos no maximo
20    cout.precision(5);
21
22    cout << a << '\t' << b << '\t' << c << endl;
23    cout << fixed << a << '\t' << b << '\t' << c << endl;
24    cout << scientific << a << '\t' << b << '\t' << c << endl;
25
26    // Sets the basefield format flag for the str stream to dec, hex or
27    // oct.
28    int n = 70;
29    cout << dec << n << endl;
30    cout << hex << n << endl;
31    cout << oct << n << endl;
32
33    return 0;
34 }
35 /* output
36 9.1
37 9.09
38 9.10
39 3.1416 2006 1e-10
40 3.14159 2006.00000 0.00000
41 3.14159e+00 2.00600e+03 1.00000e-10
42 70
43 46
44 106
45 */
```

Código 18: máximo divisor comum e mínimo múltiplo comum

```
1 int gcd(int x, int y)
2 {
3     return y ? gcd(y, x % y) : abs(x);
4 }
5 uint64_t lcm(int x, int y)
6 {
7     if (x && y) return abs(x) / gcd(x, y) * uint64_t(abs(y));
8     else return uint64_t(abs(x | y));
9 }
```

Código 19: decide se um número é primo

```
1 bool isPrime(int n)
2 {
3     if (n < 0) return isPrime(-n);
4     if (n == 1) return true;
5     if (n < 5 || n % 2 == 0 || n % 3 == 0) return (n == 2 || n == 3);
6
7     int maxP = sqrt(n) + 2;
8     for (int p = 5; p < maxP; p += 6)
9     {
10        if (n % p == 0 || n % (p+2) == 0) return false;
11    }
12    return true;
13 }
```

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

```
1 typedef map<int, int> prime_map;
2 void squeeze(prime_map& M, int& n, int p)
3 {
4     for (; n % p == 0; n /= p) M[p]++;
5 }
6 void factor(int n, prime_map& M)
7 {
8     if (n < 0) { factor(-n, M); return; }
9     if (n < 2) return;
10
11    squeeze(M, n, 2);
12    squeeze(M, n, 3);
13
14    int maxP = sqrt(n) + 2;
15    for (int p = 5; p < maxP; p += 6)
16    {
17        squeeze(M, n, p);
18        squeeze(M, n, p+2);
19    }
20    if (n > 1) M[n]++;
21 }
```

2.2 Teoria dos números

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


```

1 int mpow(int a, int b, int n = 10)
2 {
3     if(b == 0)
4         return 1;
5     else {
6         long long res = mpow(a, b/2, n);
7         res = (res*res) % n;
8         if(b%2 == 1)
9             res = (res*a) % n;
10        return (int) res;
11    }
12 }

```

2.3 Programação Dinâmica

Código 22: **Sub Set Sum**: Verifica se há um subconjunto dos elementos do vetor cuja soma seja igual a soma pedida.

```

1 //soma maxima dos elementos do vetor
2 #define MAXSUM 10000
3 int n;
4 int vet[TAM];
5 bool m[MAXSUM];
6
7 //M->soma maxima dos elementos do vetor c->soma procurada
8 bool subSetSum(int M, int c)
9 {
10     for (int i = 0; i <= M; i++) m[i] = false;
11     m[0] = true;
12
13     for(int i = 0; i < n; i++)
14     {
15         for(int j = M; j >= vet[i]; j--)
16             {
17                 m[j] |= m[j - vet[i]];
18             }
19     }
20
21     return m[c];
22 }

```

Código 23: **Lis: longest increasing (decreasing) subsequence** $O(n^2)$

```

1 #define TAM 10000
2 int c[TAM];
3 int A[TAM];
4 int H[TAM];
5
6 void ssctf(int n)
7 {
8     for (int m = 1; m <= n; m++)
9     {
10         c[m] = H[m];

```

```

11         for (int i = m - 1; i > 0; i--)
12         {
13             if (A[i] < A[m] && c[i] + H[m] > c[m])
14             {
15                 c[m] = c[i] + H[m];
16             }
17         }
18     }
19 }
20
21 void ssdtf(int n)
22 {
23     for (int m = 1; m <= n; m++)
24     {
25         c[m] = H[m];
26         for (int i = m - 1; i > 0; i--)
27         {
28             if (A[i] > A[m] && c[i] + H[m] > c[m])
29             {
30                 c[m] = c[i] + H[m];
31             }
32         }
33     }
34 }
35
36 int lis1d(int n, bool inc = true)
37 {
38     if (inc) ssctf(n);
39     else ssdtf(n);
40
41     int max = 0;
42
43     for (int i = 1; i <= n; i++)
44         if (max < c[i])
45             max = c[i];
46
47     return max;
48 }

```

Código 24: **Lis: longest increasing subsequence** $O(n \log n)$

```

1 // Longest Increasing Subsequence - LIS  $O(n \log n)$ 
2 #define fori(i, n) for (int i = 0; i < (n); ++i)
3 void lis(const vector<int> &v, vector<int> &asw)
4 {
5     vector<int> pd(v.size(), 0), pd_index(v.size()), pred(v.size());
6     int maxi = 0, x, j, ind;
7
8     fori(i, v.size())
9     {
10         x = v[i];
11         j = lower_bound(pd.begin(), pd.begin() + maxi, x) - pd.begin();
12         //j = upper_bound(pd.begin(), pd.begin() + maxi, x) - pd.begin()
13         ; para lds
14         pd[j] = x;
15         pd_index[j] = i;

```

```

15     if( j == maxi ) { maxi++; ind = i; }
16     pred[i] = j ? pd_index[j-1] : -1;
17 }
18 // return maxi; se a sequencia nao precisa ser refeita
19
20 int pos = maxi-1, k = v[ind];
21 asw.resize( maxi );
22
23 while ( pos >= 0 )
24 {
25     asw[pos--] = k;
26     ind = pred[ind];
27     k = v[ind];
28 }
29 }

```

2.4 Grafos

Código 25: Verifica se o grafo é aciclico.

```

1 #define TAM 100
2 #define BRANCO 0
3 #define CINZA 1
4 #define PRETO 2
5 bool grafo[TAM][TAM];
6 int pass[TAM];
7
8 bool dfs(int v)
9 {
10     pass[v] = CINZA;
11
12     for (int i = 0; i < TAM; i++)
13     {
14         if (grafo[v][i])
15         {
16             if (pass[i] == CINZA) return false;
17             if (pass[i] == BRANCO && !dfs(i)) return false;
18         }
19     }
20
21     pass[v] = PRETO;
22     return true;
23 }
24
25 bool aciclico()
26 {
27     memset(pass, BRANCO, TAM*sizeof(int));
28
29     for (int i = 0; i < TAM; i++)
30     {
31         if (pass[i] == BRANCO)
32         {
33             if (!dfs(i)) return false;
34         }
35     }
36 }

```

```

35 }
36
37 return true;
38 }

```

Código 26: Dijkstra Caminho minimo 1 para todos pesos positivos.

```

1 #include <queue>
2
3 typedef vector<map<int, int> > AdjList;
4 typedef AdjList Grafo;
5
6 int dist[MAX_VERTICES];
7 int prev[MAX_VERTICES]; // para recuperar o caminho usando um disjoint
                        forest set
8
9 void dijkstra(Grafo& grafo, int source)
10 {
11     for (int i = 0; i < grafo.size(); i++)
12     {
13         dist[i] = INF;
14         prev[i] = -1;
15     }
16
17     dist[source] = 0;
18     priority_queue<pair<int, int> > heap;
19     heap.push(make_pair(0, source));
20
21     while (!heap.empty())
22     {
23         int u = heap.top().second;
24         heap.pop();
25
26         // para cada vizinho de u
27         for (map<int, int>::iterator i = grafo[u].begin(); i != grafo[u].
                        end(); i++)
28         {
29             int totalDist = dist[u] + (*i).second;
30             if (totalDist <= dist[(*i).first])
31             {
32                 dist[(*i).first] = totalDist;
33                 heap.push(make_pair(totalDist, (*i).first));
34                 prev[(*i).first] = u;
35             }
36         }
37     }
38 }

```

Código 27: Floresta dijunta de arvores

```

1 #define SIZE 100
2
3 struct dsf
4 {
5     int element_count;

```

```

6     int parent[SIZE];
7     int rank[SIZE];
8 };
9 typedef struct dsf * disjoint_set_forest_p;
10
11 void dsf_init(disjoint_set_forest_p forest, int element_count)
12 {
13     forest->element_count = element_count;
14     memset(forest->parent, 0, element_count*sizeof(int));
15     memset(forest->rank, 0, element_count*sizeof(int));
16
17     for (int i = 0; i < element_count; ++i)
18         forest->parent[i] = i;
19 }
20
21 int dsf_find_set(disjoint_set_forest_p forest, int i)
22 {
23     if (i != forest->parent[i])
24     {
25         forest->parent[i] = dsf_find_set(forest, forest->parent[i]);
26     }
27     return forest->parent[i];
28 }
29
30 void dsf_union(disjoint_set_forest_p forest, int i, int j)
31 {
32     int x = dsf_find_set(forest, i);
33     int y = dsf_find_set(forest, j);
34
35     if (forest->rank[x] > forest->rank[y])
36     {
37         forest->parent[y] = x;
38     }
39     else
40     {
41         forest->parent[x] = y;
42         if (forest->rank[x] == forest->rank[y])
43         {
44             forest->rank[y]++;
45         }
46     }
47 }

```

Código 28: Kruskal Arvore geradora mínima kruskal

```

1 typedef vector<map<int, int>> AdjList;
2 struct Grafo
3 {
4     int edgeCnt;
5     AdjList adj;
6 };
7
8 struct edge
9 {
10     int u;
11     int v;

```

```

12     int weight;
13 };
14
15 int edge_compare(const void * e1, const void * e2)
16 {
17     struct edge * p1 = (struct edge *) e1;
18     struct edge * p2 = (struct edge *) e2;
19     int f = p1->weight - p2->weight;
20     if (f < 0)
21     {
22         return -1;
23     }
24     else if (f == 0)
25     {
26         return edge_compare1(e1, e2);
27     }
28     else
29     {
30         return 1;
31     }
32 }
33
34 struct edge * get_edge_list(Grafo& graph)
35 {
36     int edge_count = graph.edgeCnt;
37     struct edge *edges = (struct edge*) malloc(edge_count * sizeof(
38         struct edge));
39
40     int current_edge = 0;
41
42     for (int i = 0; i < graph.adj.size(); ++i)
43     {
44         for (map<int, int>::iterator j = graph.adj[i].begin(); j !=
45             graph.adj[i].end(); j++)
46         {
47             struct edge e;
48             e.u = i < (*j).first ? i : (*j).first;
49             e.v = i > (*j).first ? i : (*j).first;
50             e.weight = (*j).second;
51             edges[current_edge++] = e;
52         }
53     }
54
55     return edges;
56 }
57
58 void kruskal(Grafo& graph, Grafo& mst)
59 {
60     // Obtain a list of edges and sort it by weight in O(E lg E) time
61     int edge_count = graph.edgeCnt;
62     struct edge *edges = get_edge_list(graph);
63     qsort(edges, edge_count, sizeof(struct edge), edge_compare);
64
65     disjoint_set_forest dsf;
66     dsf_init(&dsf, edge_count);

```

```

66     for (int i = 0; i < edge_count; ++i)
67     {
68         struct edge e = edges[i];
69         int u = dsf_find_set(dsf, e.u);
70         int v = dsf_find_set(dsf, e.v);
71         if (u != v)
72         {
73             mst.adj[e.u][e.v] = e.weight;
74             mst.edgeCnt++;
75             dsf_union(dsf, u, v);
76         }
77     }
78     free(edges);
79 }
80 }

```

Código 29: verifica se um grafo é bipartido

```

1 #define TAM 200
2
3 bool grafo[TAM][TAM];
4 int pass[TAM];
5 int n;
6
7 bool bipartido(int v, int color = 1)
8 {
9     pass[v] = color;
10    int thisColor = color;
11    bool ret = true;
12
13    color = color == 1 ? 2 : 1;
14
15    for (int i = 0; i < n; i++)
16    {
17        if (grafo[v][i])
18        {
19            if (!pass[i]) ret = dfs(i, color);
20            else if (pass[i] == thisColor) return false;
21
22            if (!ret) return false;
23        }
24    }
25
26    return ret;
27 }

```

Código 30: faz a ordenação topológica de um grafo acíclico

```

1 #define UNVISITED -1
2
3 int grafo[SIZE][SIZE];
4 int prof[SIZE];
5 int sorted[SIZE];
6 int nordem;
7

```

```

8 void dfsTopsort(int no)
9 {
10    for (int viz = 0; viz < SIZE; viz++)
11    {
12        if (grafo[no][viz])
13        {
14            if (prof[viz] == UNVISITED)
15            {
16                prof[viz] = prof[no] + 1;
17                dfsTopsort(viz);
18            }
19        }
20    }
21
22    sorted[nordem++] = no;
23 }
24
25 void topSort(int nvt)
26 {
27     memset(prof, UNVISITED, nvt*sizeof(int));
28     nordem = 0;
29
30     for (int i = 0; i < nvt; i++)
31     {
32         if (prof[i] == UNVISITED)
33         {
34             prof[i] = 0;
35             dfsTopsort(i);
36         }
37     }
38 }

```

Código 31: calcula fluxo máximo, Ford-Fulkerson

```

1 #define TAM 1000
2 #define MAX_INT 1000000
3
4 int grafo[TAM][TAM];
5 int pred[TAM];
6 int f[TAM][TAM];
7 bool visitados[TAM];
8 int fila[TAM];
9
10 bool bfs(int n, int ini, int fim)
11 {
12     int no, s = 0, e = 0;
13     fila[e++] = ini;
14
15     while (s != e)
16     {
17         no = fila[s++];
18
19         if (visitados[no] == 2) continue;
20         visitados[no] = 2;
21
22         for (int i = 0; i < n; i++)

```

```

23     {
24         if (visitados[i] < 2)
25         {
26             if(grafo[no][i] - f[no][i] > 0)
27             {
28                 pred[i] = no;
29                 if (i == fim) return true;
30                 if(visitados[i] == 0)
31                 {
32                     fila[e++] = i;
33                     visitados[i] = 1;
34                 }
35             }
36         }
37     }
38 }
39
40 return false;
41 }
42
43 bool dfs(int s, int t, int size)
44 {
45     visitados[s] = true;
46     if(s == t) return true;
47
48     for(int v = 0; v < size; v++)
49     {
50         if(!visitados[v] && grafo[s][v] - f[s][v] > 0)
51         {
52             pred[v] = s;
53             if(dfs(v, t, size)) return true;
54         }
55     }
56
57     return false;
58 }
59
60 bool findPath(int s, int t, int size)
61 {
62     memset(visitados, false, sizeof(bool)*size);
63     pred[s] = s;
64     // Aqui pode ser usado tanto busca em largura quanto em profundidade.
65     // busca em largura geralmente apresenta tempos de execucao bem
66     // menores.
67     return bfs(size, s, t);
68     //return dfs(s, t, size);
69 }
70
71 int maxFlow(int size, int s, int t)
72 {
73     int delta;
74
75     for(int i = 0; i < size; i++)
76     {
77         memset(f[i], 0, sizeof(int)*size);

```

```

78
79     while(1)
80     {
81         bool path = findPath(s, t, size);
82         if (!path) break;
83
84         delta = MAX_INT;
85         for(int c = t; pred[c] != c; c = pred[c])
86         {
87             delta = min(delta, grafo[pred[c]][c] - f[pred[c]][c]);
88         }
89
90         for(int c = t; pred[c] != c; c = pred[c])
91         {
92             f[pred[c]][c] += delta;
93             f[c][pred[c]] -= delta;
94         }
95     }
96
97     int soma = 0;
98
99     for(int i = 0; i < size; i++)
100     {
101         soma += f[i][t];
102     }
103
104     return soma;
105 }

```

Código 32: calcula fluxo máximo, algoritmo mais eficiente porém muito maior em tempo de codificação

```

1  const int VT = 100;
2  const int AR = VT * VT;
3
4  struct grafo
5  {
6      // lista de adjacencias representada na forma de vetor
7      int nvt, nar;
8      int dest[2 * AR];
9      int adj[VT][2 * VT];
10     int nadj[VT];
11
12     int cap[AR]; // capacidade do arco
13     int fluxo[AR];
14     int ent[VT];
15
16     int padj[VT], lim[VT], nivel[VT], qtd[VT];
17
18     int inv(int a) { return a ^ 0x1; }
19     int orig(int a) { return dest[inv(a)]; }
20     int capres(int a) { return cap[a] - fluxo[a]; }
21
22     void inic(int n = 0)
23     {
24         nvt = n;

```

```

25     nar = 0;
26     memset(nadj, 0, sizeof(nadj));
27 }
28
29 //////////////////////////////////////////////////
30 // Adiciona uma aresta ao grafo.
31 //
32 // "int u" apenas para Fluxos;
33 //
34 int aresta(int i, int j, int u = 0)
35 {
36     int ar = nar;
37     cap[nar] = u;
38     dest[nar] = j;
39     adj[i][nadj[i]] = nar++;
40     nadj[i]++;
41
42     cap[nar] = 0;
43     dest[nar] = i;
44     adj[j][nadj[j]] = nar++;
45     nadj[j]++;
46     return ar;
47 }
48
49 void revbfs(int ini, int fim)
50 {
51     int i, no, viz, ar;
52     queue<int> fila;
53
54     memset(nivel, NULO, sizeof(nivel));
55     memset(qtd, 0, sizeof(qtd));
56
57     nivel[fim] = 0;
58     fila.push(fim);
59
60     while (!fila.empty())
61     {
62         no = fila.front();
63         fila.pop();
64         qtd[nivel[no]]++;
65
66         for (i = 0; i < nadj[no]; i++)
67         {
68             ar = adj[no][i];
69             viz = dest[ar];
70
71             if (cap[ar] == 0 && nivel[viz] == NULO)
72             {
73                 nivel[viz] = nivel[no] + 1;
74                 fila.push(viz);
75             }
76         }
77     }
78 }
79
80 int admissivel(int no)

```

```

81 {
82     while (padj[no] < nadj[no])
83     {
84         int ar = adj[no][padj[no]];
85         if (nivel[no] == nivel[dest[ar]] + 1 && capres(ar) > 0) return
            ar;
86         padj[no]++;
87     }
88
89     padj[no] = 0;
90     return NULO;
91 }
92
93 int retrocede(int no)
94 {
95     int i, ar, viz, menor = NULO;
96
97     if (--qtd[nivel[no]] == 0) return NULO;
98
99     for (i = 0; i < nadj[no]; i++)
100     {
101         ar = adj[no][i]; viz = dest[ar];
102         if (capres(ar) <= 0) continue;
103         if (menor == NULO || nivel[viz] < nivel[menor]) menor = viz;
104     }
105
106     if (menor != NULO) nivel[no] = nivel[menor];
107     qtd[++nivel[no]]++;
108
109     return ((ent[no] == NULO) ? no : orig(ent[no]));
110 }
111
112 int avanca(int no, int ar)
113 {
114     int viz = dest[ar];
115     ent[viz] = ar;
116     lim[viz] = min(lim[no], capres(ar));
117     return viz;
118 }
119
120 int aumenta(int ini, int fim)
121 {
122     int ar, no = fim, fmax = lim[fim];
123
124     while (no != ini)
125     {
126         fluxo[ar = ent[no]] += fmax;
127         fluxo[inv(ar)] -= fmax;
128         no = orig(ar);
129     }
130
131     return fmax;
132 }
133
134 int maxflow(int ini, int fim)
135 {

```

```

136     int ar, no = ini, fmax = 0;
137
138     memset(fluxo, 0, sizeof(fluxo));
139     memset(padj, 0, sizeof(padj));
140
141     revbfs(ini, fim);
142
143     lim[ini] = INF;
144     ent[ini] = NULO;
145
146     while (nivel[ini] < nvt && no != NULO)
147     {
148         if ((ar = admissivel(no)) == NULO)
149         {
150             no = retrocede(no);
151         }
152         else if ((no = avanca(no, ar)) == fim)
153         {
154             fmax += aumenta(ini, fim);
155             no = ini;
156         }
157     }
158     return fmax;
159 }
160 };

```

2.5 Geometria

Código 33: ponto e poligono

```

1 struct point
2 {
3     double x, y;
4     double z; // para pontos no espaco
5     point(double x = 0, double y = 0, double z = 0): x(x), y(y), z(z) {}
6
7     point operator +(point q) { return point(x + q.x, y + q.y, z + q.z); }
8     point operator -(point q) { return point(x - q.x, y - q.y, z - q.z); }
9     point operator *(double t) { return point(x * t, y * t, z * t); }
10    point operator /(double t) { return point(x / t, y / t, z / t); }
11    double operator *(point q) { return x * q.x + y * q.y + z * q.z; }
12    point vec(point q) { return point(y * q.z - z * q.y, z * q.x - x * q.z, x * q.y - y * q.x); }
13    double operator %(point q) { return x * q.y - y * q.x; }
14
15    int cmp(point q) const
16    {
17        if (int t = ::cmp(x, q.x)) return t;
18        else if (int t = ::cmp(y, q.y)) return t;
19        return ::cmp(z, q.z);
20    }
21

```

```

22    bool operator ==(point q) const { return cmp(q) == 0; }
23    bool operator !=(point q) const { return cmp(q) != 0; }
24    bool operator < (point q) const { return cmp(q) < 0; }
25
26    friend ostream& operator <<(ostream& o, point p) {
27        return o << "(" << p.x << ", " << p.y << ", " << p.z << ")";
28    }
29    static point pivot;
30 };
31
32 // para pontos 2D
33 double abs(point p) { return hypot(p.x, p.y); }
34 double arg(point p) { return atan2(p.y, p.x); }
35
36 point point::pivot;
37
38 typedef vector<point> polygon;
39
40 int ccw(point p, point q, point r)
41 {
42     return cmp((p - r) % (q - r));
43 }
44
45 double angle(point p, point q, point r)
46 {
47     point u = p - q, v = r - q;
48     return atan2(u % v, u * v);
49 }

```

Código 34: Decide se q está sobre o segmento fechado pr.

```

1 bool between(point p, point q, point r)
2 {
3     return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
4 }

```

Código 35: Decide se os segmentos fechados pq e rs têm pontos em comum.

```

1 bool seg_intersect(point p, point q, point r, point s)
2 {
3     point A = q - p;
4     point B = s - r;
5     point C = r - p;
6     point D = s - q;
7
8     int a = cmp(A % C) + 2 * cmp(A % D);
9     int b = cmp(B % C) + 2 * cmp(B % D);
10
11     if (a == 3 || a == -3 || b == 3 || b == -3) return false;
12     if (a || b || p == r || p == s || q == r || q == s) return true;
13
14     int t = (p < r) + (p < s) + (q < r) + (q < s);
15     return t != 0 && t != 4;
16 }

```

Código 36: Calcula a distância do ponto r ao segmento pq.

```

1 double seg_distance(point p, point q, point r)
2 {
3     point A = r - q;
4     point B = r - p;
5     point C = q - p;
6
7     double a = A * A, b = B * B, c = C * C;
8
9     if (cmp(b, a + c) >= 0) return sqrt(a);
10    else if (cmp(a, b + c) >= 0) return sqrt(b);
11    else return fabs(A % B) / sqrt(c);
12 }

```

Código 37: 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.

```

1 int in_poly(point p, polygon& T)
2 {
3     double a = 0;
4     int N = T.size();
5     for (int i = 0; i < N; i++)
6     {
7         if (between(T[i], p, T[(i+1) % N])) return -1;
8         a += angle(T[i], p, T[(i+1) % N]);
9     }
10    return cmp(a) != 0;
11 }

```

2.6 Casamento de strings

Código 38: String matching - Algoritmo KMP - $O(n + m)$

```

1 // F[i] - size of the largest prefix of pattern[0..i] that is also a
2 // suffix of pattern[1..i]. Ex: pattern = {a,b,a,c,a,b}, F =
3 // {0,0,1,0,1,2}
4 #define MAX_PATTERN_SIZE 10010
5 int F[MAX_PATTERN_SIZE];
6 void build_failure_function( const string & pattern )
7 {
8     int m = pattern.size();
9     F[0] = -1;
10    for (int i = 0; i < m; i++)
11    {
12        F[i+1] = F[i] + 1;
13        while ( F[i+1] > 0 && pattern[i] != pattern[ F[i+1]-1 ] )
14            F[i+1] = F[ F[i+1]-1 ] + 1;
15    }
16
17    // retorna a posicao inicial de cada ocorrencia de pattern em text
18    vector<int> KMP( const string & text, const string & pattern )

```

```

19 {
20     build_failure_function( pattern );
21     vector<int> start_positions;
22     int j = 0, m = pattern.size(), n = text.size();
23
24     for (int i = 0; i < n; i++)
25     {
26         while ( true )
27         {
28             if ( text[i] == pattern[j] )
29             {
30                 if ( ++j == m )
31                 {
32                     start_positions.push_back( i - m + 1 );
33                     j = F[j];
34                 }
35                 break;
36             }
37
38             if ( j == 0 ) break;
39             j = F[j];
40         }
41     }
42
43     return start_positions;
44 }

```

2.7 Outros

Código 39: josephus problem

```

1 /**
2  * The Josephus problem (or Josephus permutation) is a theoretical problem
3  * related to a certain counting-out game. There are people standing in
4  * a circle waiting to be executed. After the first man is executed,
5  * certain number of people are skipped and one man is executed. Then
6  * again, people are skipped and a man is executed. The elimination
7  * proceeds around the circle (which is becoming smaller and smaller as
8  * the executed people are removed), until only the last man remains,
9  * who is given freedom. The task is to choose the place in the initial
10    circle so that you are the last one remaining and so survive.
11
12    */
13
14    using namespace std;
15
16    int josephus(int n, int m)
17    {
18        int res = 0;
19        vector<int> people;
20        int loc = 0;
21
22        for (int i = 0; i < n; i++) people.push_back(i+1);
23
24        while (people.size() > 1)

```

```

16 {
17     if (loc >= people.size())
18         loc %= people.size();
19
20     people.erase(people.begin()+loc);
21     loc += (m-1);
22 }
23
24 return people[0];
25 }

```

Código 40: Simplex

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 #define fori(i,n) for(int i=0; i < (n); ++i)
8 #define forr(i,a,b) for(int i=(a); i <= (b); ++i)
9 #define ford(i,a,b) for(int i=(a); i >= (b); --i)
10 #define sz size()
11
12 const double EPS=1e-9; const int INF = 0x3f3f3f3f;
13
14 #define all(x) (x).begin(),(x).end()
15
16 int cmpD(double x, double y=0, double tol=EPS) {
17     return (x <= y+tol) ? (x+tol<y) ? -1 : 0 : 1;
18 }
19
20 struct simplex {
21     // max c * x, s.t: A * x <= b; x >= 0
22     simplex( const vector< vector< double > > & A_, const vector< double
23         > & b_,
24         const vector< double > & c_ ) : A( A_ ), b( b_ ), c( c_ ) {}
25     vector< vector< double > > A; vector< double > b, c, sol;
26     vector< bool > N; vector< int > kt; int m, n;
27     void pivot( int k, int l, int e ) {
28         int x = kt[l]; double p = A[l][e];
29         fori(i,k) A[l][i] /= p;
30         b[l] /= p; N[e] = false;
31         fori(i,m) if (i != l) {b[i] -= A[i][e]*b[l]; A[i][x] = -A[i][e]*A[
32             l][x];}
33         fori(j,k) if ( N[j] ) {
34             c[j] -= c[e] * A[l][j];
35             fori(i,m) if ( i != l ) A[i][j] -= A[i][e] * A[l][j];
36         }
37         kt[l] = e; N[x] = true; c[x] = -c[e] * A[l][x];
38     }
39     vector< double > go( int k ) {
40         vector< double > res;
41         while ( 1 ) {
42             int e = -1, l = -1;
43             fori(i,k) if ( N[i] && cmpD( c[i] ) > 0 ) { e = i; break; }

```

```

42         if ( e == -1 ) break;
43         fori(i,m) if ( cmpD(A[i][e]) > 0 && ( l == -1 || cmpD( b[i] / A
44             [i][e],
45                 b[l] / A[l][e], 1e-20 ) < 0 ) ) l = i;
46         if ( l == -1 ) return vector< double >(); // unbounded
47         pivot( k, l, e );
48     }
49     res.resize( k, 0 );
50     fori(i,m) res[kt[i]] = b[i];
51     return res;
52 }
53 vector< double > solve() {
54     m = A.sz; n = A[0].sz; int k = m+n+1;
55     N = vector< bool >( k, true ); vector< double > c_copy = c;
56     c.resize(n+m); kt.resize(m);
57     fori(i,m) {
58         A[i].resize(k); A[i][n+i] = 1; A[i][k-1] = -1;
59         kt[i] = n+i; N[kt[i]] = false;
60     }
61     int l = min_element(all(b)) - b.begin();
62     if(cmpD(b[l]) < 0 ) {
63         c = vector<double>(k,0);
64         c[k-1] = -1; pivot(k, l, k-1); sol=go(k);
65         if(cmpD(sol[k-1])>0) return vector<double>(); // infeasible
66         fori(i,m) if(kt[i] == k-1) {
67             fori(j,k-1) if(N[j] && cmpD( A[i][j] ) != 0 ) {
68                 pivot( k, i, j); break;
69             }
70         }
71         c=c_copy; c.resize(k,0);
72         fori(i,m) fori(j,k) if(N[j]) c[j] -= c[kt[i]]*A[i][j];
73     }
74     sol = go(k-1);
75     if(!sol.empty()) sol.resize(n);
76     return sol;
77 }
78
79 // Como usar
80 int main() {
81     /* Exemplo: Maximize cx Subject to Ax <= b */
82     vector<vector<double>> A(9);
83     double Av[][3] = {{1,1,0}, {0,0,-1}, {-1,-1,0},
84         {0,0,1}, {1,0,0}, {0,1,0},
85         {0,0,1}, {1,0,1}, {0,1,0}};
86
87     for(int i=0; i < 9; i++) {
88         A[i].insert(A[i].begin(), &(Av[i][0]), &(Av[i][3])); // Sim, [3]!
89         Ou seja, idx-final+1
90     }
91
92     vector<double> c(3, 1); // c=[1 1 1]
93     double bv[] = {2,-1,-2,1,2,1,1,2,1};
94     vector<double> b(bv, bv+sizeof(bv)/sizeof(double));
95
96     simplex sim(A,b,c);

```

```

96 vector<double> s = sim.solve();
97 if(!s.size()) cout << "Impossible\n";
98 else
99 for(int i=0; i < s.size(); i++) {
100     cout << s[i] << endl;
101 }
102 }

```

Código 41: Gera as permutações dos elementos da string

```

1 bool nextPermutation(string& number)
2 {
3     bool isBigger = true;
4     int i, j;
5
6     for (i = number.size() - 1; i >= 0; i--)
7     {
8         if (number[i] < number[i+1]) break;
9     }
10
11    if (i != -1)
12    {
13        isBigger = false;
14
15        for (j = number.size() - 1; j >= i+1; j--)
16        {
17            if (number[j] > number[i])
18            {
19                break;
20            }
21        }
22
23        int tmp = number[i];
24        number[i] = number[j];
25        number[j] = tmp;
26
27        j = number.size() - 1;
28        i++;
29
30        while (i < j)
31        {
32            tmp = number[i];
33            number[i] = number[j];
34            number[j] = tmp;
35            i++;
36            j--;
37        }
38    }
39
40    return isBigger;
41 }

```

3 Biblioteca C/C++

3.1 I/O

Ignorando entradas na família scanf:

Código 42: Ignora os dois floats do meio. Retornará 2 no sucesso.

```

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

```

3.2 Map

Código 43: Referencias map

```

1 #include <map>
2 #include <string>
3 #include <cstdio>
4
5 using namespace std; // USE ISTO!!!
6
7 class Comparadora;
8
9 class Pessoa {
10     int idade;
11     string nome;
12     friend class Comparadora;
13 public:
14     Pessoa(string nome, int idade) {
15         this->idade = idade;
16         this->nome = nome;
17     }
18     void print() const {
19         printf("Nome: %s Idade: %d\n", nome.c_str(), idade);
20     }
21 };
22
23 class Comparadora { // Ordena crescentemente
24 public: // <- IMPORTANTE
25     bool operator() (const Pessoa &a, const Pessoa &b)
26     {
27         int idDif = a.idade-b.idade;
28         if(idDif < 0) return true;
29         else if(idDif==0) return a.nome.compare(b.nome) < 0 ? true : false
30             ;
31         else return false;
32     }
33 };
34
35 int main() {
36     Pessoa r("Rangelz", 86);
37     Pessoa r2("Rangelzao", 86);
38
39     map<Pessoa, string, Comparadora> alunos;
40     alunos[r]="UFMG";

```

```
40     alunos[r2]="PUC";
41     // Iterator
42     for(map<Pessoa, string, Comparadora>::iterator it=alunos.begin(); it
43         != alunos.end(); it++) {
44         it->first.print();
45         printf("\t%s\n\n", it->second.c_str());
46     }
47     // Find
48     if(alunos.find(Pessoa("Rangelz", 86)) != alunos.end()) { // Achou!
49         printf("Achei Rangel!\n");
50     }
51     return 0;
52 }
```
