Sumário

Lista de Tabelas

Lista de Listagens

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 dou
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

```cpp
#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 <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 1: Modelo

```cpp
const double EPS = 1e-10;
/**
 * -1 se x < y
 * 0 se x = y
 * 1 se x > y
 */
inline int cmp (double x, double y = 0, double tol = EPS)
{
    return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}
```

Código 2: comparcao de ponto flutuante

```
set ai noet ts=4 sw=4 bs=2
syn on
mat Keyword "\<foreach\>"
```

Código 3: .vimrc para a configuração do vim

```cpp
/* 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 4: printf

```cpp
#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 5: exemplo de map

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

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

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

Código 6: exemplo de set e multset

```
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 7: exemplo de list

```
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 8: exemplo de 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 9: exemplo de priority queue

```
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 10: exemplo de stack

```
1   #include <iostream>
2   #include <vector>
3   using namespace std;
4
```

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

  myvector.reserve(100);

  for (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 };
  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 11: exemplo de vector

```
#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 12: exemplo de string

```
#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 13: exemplo de ordenação

```
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 14: pesquisa binária

```
#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
*/
```

Código 15: Arredondamento e output em outras bases

2.2 Teoria dos números

```
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 16: máximo divisor comum e mínimo multiplo comum

```
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 17: decide se um número é primo

```
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) return n = -n;
   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 18: Retorna a fatoração em números primos de abs(n).

2.3 Grafos

```
#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 19: Verifica se o grafo é aciclico.

```
#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 20: Caminho minimo 1 para todos pesos positivos.

```
#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 21: Floresta dijunta de arvores

```
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 22: Arvore geradora mínima kruskal

```
#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 23: verifica se um grafo é bipartido

```
#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 24: faz a ordenação topológica de um grafo acíclico

```
#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]) continue;
      visitados[no] = true;

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

   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 25: calcula fluxo máximo, Ford-Fulkerson

```
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;
    }
};
```

Código 26: calcula fluxo máximo, algoritmo mais eficiente porém muito maior em tempo de codificação

2.4 Geometria

```
struct point
{
    double x, y;
    double z; // para pontos no espa o
    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 27: ponto e poligono

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

Código 28: Decide se q está sobre o segmento fechado pr.

```
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 29: Decide se os segmentos fechados pq e rs têm pontos em comum.

```
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 30: Calcula a distância do ponto r ao segmento pq.

```
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;
}
```

Código 31: 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.

2.5 Outros

```
/**
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 32: josephus problem

```
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;
}
```

Código 33: Gera as permutações dos elementos da string

3 Biblioteca C/C++

3.1 I/O

Ignorando entradas na família scanf:

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

Código 34: Ignora os dois floats do meio. Retornará 2 no sucesso.

3.2 Map

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

using namespace std; // USE ISTO!!!

class Comparadora;

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

```
34 int main() {
35     Pessoa r("Rangelz", 86);
36     Pessoa r2("Rangelzao", 86);
37 
38     map<Pessoa, string, Comparadora> alunos;
39     alunos[r]="UFMG";
40     alunos[r2]="PUC";
41     // Iterator
42     for(map<Pessoa, string, Comparadora>::iterator it
           =alunos.begin(); it != alunos.end(); it++) {
43         it->first.print();
44         printf("\t%s\n\n", it->second.c_str());
45     }
46     // Find
47     if(alunos.find(Pessoa("Rangelz", 86)) != alunos.
           end()) { // Achou!
48         printf("Achei Rangel!\n");
49     }
50     return 0;
51 }
```

Código 35: Referencias map