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

2 Codigos

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

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

```
const double EPS = 1e-10;
/**
 * -1 se x < y
 * 0 se x = y
```

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)
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)

```
 * 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

```
/* 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

```
#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

```
#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));
```

```cpp

  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 6: exemplo de set e multset

```cpp
#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 7: exemplo de list

```cpp
#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 8: exemplo de queue

```cpp
#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 9: exemplo de priority queue

```cpp
#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 10: exemplo de stack

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

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

```cpp
#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 12: 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 13: 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 14: Arredondamento e output em outras bases

```
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 15: 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 16: 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 17: Retorna a fatoração em números primos de abs(n).

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

```
struct dsf
{
    int element_count;
    int *parent;
    int *rank;
};
typedef struct dsf * disjoint_set_forest_p;

disjoint_set_forest_p dsf_alloc(int element_count)
{
    disjoint_set_forest_p forest = (disjoint_set_forest_p) malloc(sizeof(
        struct dsf));

    forest->element_count = element_count;
    forest->parent = (int*) calloc(element_count, sizeof(int));
    forest->rank = (int*) calloc(element_count, sizeof(int));

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

    return forest;
}

void dsf_free(disjoint_set_forest_p forest)
{
    if (forest)
    {
        free(forest->parent);
        free(forest->rank);
        forest->element_count = 0;
        forest->parent = NULL;
        forest->rank = NULL;
        free(forest);
    }
}

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 19: 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_p dsf = dsf_alloc(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);
        }
    }

    dsf_free(dsf);
    free(edges);
}
```

Código 20: 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 21: verifica se um grafo é bipartido

```
/**
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 22: josephus problem