La Máquina de Humo

29 de septiembre de 2018

Resumen

Códigos de problemas resueltos. Problemas de Uri, TAP 2016, TAP 2017. Documentación de librerías y ayuda de sintaxis

1. Problemas de Uri

1219

```
#include <iostream>
#include <iomanip>
#include <math.h>

using namespace std;

int main(){
long double a,b,c;
long double radioS, radioR;
long double areaT;
long double areaS, areaV, areaR;
while(cin >> a >> b >> c){
long double perimetro = a + b + c;
long double p = perimetro / 2;
areaT = sqrt(p*(p-a)*(p-b)*(p-c) );
radioS = (a*b*c)/(4*areaT);
radioR = areaT/p;
areaR = M_PI * radioR * radioR;
areaV = areaT - areaR;
areaS = M_PI * radioS * radioS - areaT;
cout << fixed;
cout << setprecision(4);
cout << areaS << " " << areaV << " " << areaR << endl;
//cout << EOF << endl;
}
 return 0;
}
```

2. Problemas de TAP 2016

A

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

using namespace std;

int main (){
string s;
bool r;
while(cin >> s){
r = true;
bool r = s.find('i') != string::npos || s.find('I') != string::npos;
if (!r)
cout << 'S' << endl;
else
cout << 'N' << endl;
}
return 0;
}
```

C

```
#include <bits/stdc++.h>

long contador = 0;
using namespace std;

bool check (int x, bool *mat, bool *reg, int n){
bool r = true;
for(int i = 0; i < n; i++){
//cout << "Linkeado:" << *((mat+i*n) + (x - 1))
//<< "," << i << "," << x -1 << endl;
if(*((mat+i*n) + (x - 1))){
r = r && *(reg+i);
//cout << "Registrado:" <<*(reg+i) << endl;
}
if (!r) break;
}
/*delete mat;
delete reg;*/
return r;
}

void registrar (int x, bool *mat, bool *reg, int n, bool *pen){
*(reg + (x-1)) = 1;
pen[x-1] = 0; // no se si va
//cout << "Se registro:" << x << endl;
contador++;
for(int i = 0; i < n; i++){
//cout << *((mat+(x-1)*n) + i) <<endl;
if(*((mat+(x-1)*n) + i) && !*(reg+(i)) && *(pen+i)){
if(check(i +1 , mat, reg, n)){
*(pen+i) = 0;
registrar(i + 1, mat, reg, n, pen);
}

}
}
/*delete mat;
delete reg;
delete pen;*/
}

int main (){
int n,m, a, b, x;
while(cin >> n >> m)
{
a = b = x = contador = 0;
bool mat[n][n];
bool reg[n], pen[n];

//Inicializa en 0
for(int i = 0; i < n; i++){
for(int j = 0; j < n; j++){
mat[i][j] = 0;

}
reg[i] = 0;
pen[i] = 0;
}
//Lee datos de correlativas
for(int i = 0; i < m; i++){
cin >> a >> b;
mat[a - 1][b - 1] = 1;
}
for(int i = 0; i < n; i++){
cin >> x;
//cout << check(x, (int *)mat, (int *)reg, n)  << endl;
if(check(x, (bool *)mat, (bool *)reg, n)){
registrar(x, (bool *)mat, (bool *)reg, n, (bool *)pen);
/*reg[x-1] = 1;
pen[x-1] = 0;
contador++;*/
}else{
pen[x-1] = 1;
}
cout << contador  << endl;
}
}
return 0;
}
```

C2

```
#include <iostream>
#include <map>
#include <iterator>

using namespace std;
long contador = 0;
multimap <int, int> mat;
multimap <int, int> matA;
//vector<int> G[100]
map <int, bool> reg;
map <int, bool> pen;
typedef multimap<int, int>::iterator MMAPIterator;

bool check (int x, int n){
map <int, bool> :: iterator itr;
pair<MMAPIterator, MMAPIterator> result;
bool r = true;
result = matA.equal_range(x);
for (MMAPIterator it = result.first; it != result.second; it++){
itr = reg.find(it->second);
r = r && itr->second;
if (!r) break;
}
return r;
}

void registrar (int x, int n){
map <int, bool> :: iterator itr;
pair<MMAPIterator, MMAPIterator> result;
reg.insert(pair <int,bool> (x, 1));
pen.erase(x);
contador++;
result = mat.equal_range(x);
for (MMAPIterator it = result.first; it != result.second; it++){
itr = pen.find(it->second);
if(itr->second){
```

```
if (check(it->second, n))
registrar(it->second, n);
}
}
}

int main (){
int n,m, a, b, x;
while(cin >> n >> m)
{
mat.clear();
matA.clear();
reg.clear();
pen.clear();
a = b = x = contador = 0;
//Lee datos de correlativas
for(int i = 0; i < m; i++){
cin >> a >> b;
mat.insert(pair <int, int> (a,b));
matA.insert(pair <int, int> (b,a));
}

for(int i = 0; i < n; i++){
cin >> x;
//cout << check(x, (int *)mat, (int *)reg, n)  << endl;
if(check(x,n)){
registrar(x, n);
}else{
pen.insert(pair <int, bool> (x, 1));
}
cout << contador  << endl;
}
}
return 0;
}
```

J

```
#include <bits/stdc++.h>

using namespace std;

bool comp(int a, int b){
return a>b;
}

int main (){
long int  n,l,c;
long int  k;
bool r;
while(cin >> n >> l >> c){
k = 0;
r = true;
long int pruebas[n];
for(int i = 0; i < n; i++){
cin >> pruebas[i];
}
sort(pruebas, pruebas + n, comp);
while(k < n && r){
//cout << pruebas[k] << " " << c;
if(pruebas[k]<= c ){
c -=pruebas[k];
k += l;
}else{
r = false;
}
}
if(r)
cout << 'S' << endl;
else
cout << 'N' << endl;
}

return 0;
}
```

3. Problemas de TAP 2017

A

```
#include <bits/stdc++.h>
#include <string>

using namespace std;

int main (){
int s;
string nota;
int notaReal;
string escala[] =
{"DO", "DO#", "RE", "RE#", "MI", "FA", "FA#", "SOL",
 "SOL#","LA", "LA#", "SI"};
cin >> s;
cin >> nota;
for(int i = 0; i < 12; i++){
if (escala[i] == nota){
notaReal = i - s;
if(notaReal < 0)
notaReal += 12;
}
}
cout << escala[notaReal] << endl;
return 0;
}
```

E

```
#include <bits/stdc++.h>

using namespace std;

int main (){
int a,b,c, d =0;
int cartas[] = {0,0,0,0,0,0,0};
int contrarias[3] = {0, 0, 0};
bool r = false;
cin >> a >> b >> c;
cartas[a- 1] = 1;
cartas[b-1] = 1;
cartas[c-1] = 1;
r = (cartas[0] && cartas[2]) ||
(cartas[1] && cartas[2] && cartas[3]) ||
(cartas[0] && cartas[3] && cartas[4]);
if(r)
cout << 'S' << endl;
else
cout << 'N' << endl;
return 0;
}
```

F

```
#include <bits/stdc++.h>
#include <map>

using namespace std;

int main (){
int n,a,b,c,d,x;
int count;
cin >> n;
vector <int> banda;
for(int i = 0; i < n; i++)
{
cin >> x;
if(x == 1){
cin >> a >> b >> c;
banda.push_back(a);
banda.push_back(b);
banda.push_back(c);
}
if(x == 2){
cin >> a;
banda[3*a - 1] = 0;
}
if(x == 3){
count = 0;
cin >> c >> d;
for(int i = 0; i < (banda.size() / 3); i++){
if (!(d <= banda[i*3] || c >= banda[i*3 + 1])){
count += banda[i*3 + 2];
}
}
cout << count << endl;
}
}
return 0;
}
```

H

```
#include <bits/stdc++.h>

using namespace std;

int main (){
int n,x, y, c = 0;
int i = 0;
cin >> n;
cin >> y;
bool r = y > 0;
for(int i = 0; i < (n -1); i++){
cin >> x;
if(x < y && r){
c++;
r = false;
}else{
if(x>y){
r = true;
}
}
y = x;
}
c += (r) ? 1 : 0;
cout << c << endl;
return 0;
}
```

I

```
#include <bits/stdc++.h>
#include <math>

using namespace std;

int main (){
int n;

cin >> n;
bool r = true;
int polT [n][2];
int polO [n][2];
int vectT[n][2];
int vectO[n][2];
int modO[n];
int modT[n];
for (int i = 0; i < n; i++){
cin >> polO[i][0] << polO[i][1];
}
for (int i = 0; i < n; i++){
cin >> polT[i][0] << polO[i][1];
}
for (int i = 0; i < n-1; i++){
vectO[i][0] = polO[i+1][0] - polO[i][0];
vectO[i][1] = polO[i+1][1] - polO[i][1];


vectT[i][0] = polT[i+1][0] - polT[i][0];
vectT[i][1] = polT[i+1][1] - polT[i][1];
}

vectO[n-1][0] = polO[0][0] - polO[n-1][0];
vectO[n-1][1] = polO[0][1] - polO[n-1][1];

vectT[n-1][0] = polT[0][0] - polT[n-1][0];
vectT[n-1][1] = polT[0][1] - polT[n-1][1];

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

```
modO[i] = vectO[i][0]*vectO[i][0] + vectO[i][1]*vectO[i][1];
modT[i] = vectT[i][0]*vectT[i][0] + vectT[i][1]*vectT[i][1];
}
sort(modO, modO + n);
sort(modT, modT + n);
double cociente = modO[0] / modT[0];
for(int i = 1; i < n; i++){
r = r && (cociente == (modO[i]/modT[i]));
}
if
return 0;
}
```

4. Segment Tree

C

```
// C program to show segment tree operations like construction, query
// and update
#include <stdio.h>
#include <math.h>

// A utility function to get the middle index from corner indexes.
int getMid(int s, int e) {  return s + (e -s)/2;  }

/*  A recursive function to get the sum of values in given range
    of the array. The following are parameters for this function.

    st    --> Pointer to segment tree
    si    --> Index of current node in the segment tree. Initially
              0 is passed as root is always at index 0
    ss & se  --> Starting and ending indexes of the segment represented
                 by current node, i.e., st[si]
    qs & qe  --> Starting and ending indexes of query range */
int getSumUtil(int *st, int ss, int se, int qs, int qe, int si)
{
    // If segment of this node is a part of given range, then return
    // the sum of the segment
    if (qs <= ss && qe >= se)
        return st[si];

    // If segment of this node is outside the given range
    if (se < qs || ss > qe)
        return 0;

    // If a part of this segment overlaps with the given range
    int mid = getMid(ss, se);
    return getSumUtil(st, ss, mid, qs, qe, 2*si+1) +
           getSumUtil(st, mid+1, se, qs, qe, 2*si+2);
}

/* A recursive function to update the nodes which have the given
   index in their range. The following are parameters
```

```
    st, si, ss and se are same as getSumUtil()
    i    --> index of the element to be updated. This index is
             in the input array.
   diff --> Value to be added to all nodes which have i in range */
void updateValueUtil(int *st, int ss, int se, int i, int diff, int si)
{
    // Base Case: If the input index lies outside the range of
    // this segment
    if (i < ss || i > se)
        return;

    // If the input index is in range of this node, then update
    // the value of the node and its children
    st[si] = st[si] + diff;
    if (se != ss)
    {
        int mid = getMid(ss, se);
        updateValueUtil(st, ss, mid, i, diff, 2*si + 1);
        updateValueUtil(st, mid+1, se, i, diff, 2*si + 2);
    }
}

// The function to update a value in input array and segment tree.
// It uses updateValueUtil() to update the value in segment tree
void updateValue(int arr[], int *st, int n, int i, int new_val)
{
    // Check for erroneous input index
    if (i < 0 || i > n-1)
    {
        printf("Invalid Input");
        return;
    }

    // Get the difference between new value and old value
    int diff = new_val - arr[i];

    // Update the value in array
    arr[i] = new_val;

    // Update the values of nodes in segment tree
    updateValueUtil(st, 0, n-1, i, diff, 0);
```

```
}

// Return sum of elements in range from index qs (quey start)
// to qe (query end).  It mainly uses getSumUtil()
int getSum(int *st, int n, int qs, int qe)
{
    // Check for erroneous input values
    if (qs < 0 || qe > n-1 || qs > qe)
    {
        printf("Invalid Input");
        return -1;
    }

    return getSumUtil(st, 0, n-1, qs, qe, 0);
}

// A recursive function that constructs Segment Tree for array[ss..se].
// si is index of current node in segment tree st
int constructSTUtil(int arr[], int ss, int se, int *st, int si)
{
    // If there is one element in array, store it in current node of
    // segment tree and return
    if (ss == se)
    {
        st[si] = arr[ss];
        return arr[ss];
    }

    // If there are more than one elements, then recur for left and
    // right subtrees and store the sum of values in this node
    int mid = getMid(ss, se);
    st[si] =  constructSTUtil(arr, ss, mid, st, si*2+1) +
              constructSTUtil(arr, mid+1, se, st, si*2+2);
    return st[si];
}

/* Function to construct segment tree from given array. This function
   allocates memory for segment tree and calls constructSTUtil() to
   fill the allocated memory */
int *constructST(int arr[], int n)
{
    // Allocate memory for the segment tree

    //Height of segment tree
    int x = (int)(ceil(log2(n)));

    //Maximum size of segment tree
    int max_size = 2*(int)pow(2, x) - 1;

    // Allocate memory
    int *st = new int[max_size];

    // Fill the allocated memory st
    constructSTUtil(arr, 0, n-1, st, 0);

    // Return the constructed segment tree
    return st;
}

// Driver program to test above functions
int main()
{
    int arr[] = {1, 3, 5, 7, 9, 11};
    int n = sizeof(arr)/sizeof(arr[0]);

    // Build segment tree from given array
    int *st = constructST(arr, n);

    // Print sum of values in array from index 1 to 3
    printf("Sum of values in given range = %dn",
            getSum(st, n, 1, 3));

    // Update: set arr[1] = 10 and update corresponding
    // segment tree nodes
    updateValue(arr, st, n, 1, 10);

    // Find sum after the value is updated
    printf("Updated sum of values in given range = %dn",
             getSum(st, n, 1, 3));
    return 0;
}
```

5. STL

Vector

```
-Member functions
(constructor)
Construct vector (public member function )
(destructor)
Vector destructor (public member function )
operator=
Assign content (public member function )

-Iterators:
begin
Return iterator to beginning (public member function )
end
Return iterator to end (public member function )
rbegin
Return reverse iterator to reverse beginning (public member function )
rend
Return reverse iterator to reverse end (public member function )
cbegin
Return const_iterator to beginning (public member function )
cend
Return const_iterator to end (public member function )
crbegin
Return const_reverse_iterator to reverse beginning (public member function )
crend
Return const_reverse_iterator to reverse end (public member function )

-Capacity:
size
Return size (public member function )
max_size
Return maximum size (public member function )
resize
Change size (public member function )
capacity
Return size of allocated storage capacity (public member function )
```

```
empty
Test whether vector is empty (public member function )
reserve
Request a change in capacity (public member function )
shrink_to_fit
Shrink to fit (public member function )

-Element access:
operator[]
Access element (public member function )
at
Access element (public member function )
front
Access first element (public member function )
back
Access last element (public member function )
data
Access data (public member function )

-Modifiers:
assign
Assign vector content (public member function )
push_back
Add element at the end (public member function )
pop_back
Delete last element (public member function )
insert
Insert elements (public member function )
erase
Erase elements (public member function )
swap
Swap content (public member function )
clear
Clear content (public member function )
emplace
Construct and insert element (public member function )
emplace_back
Construct and insert element at the end (public member function )

-Allocator:
get_allocator
```

```
Get allocator (public member function )

-Non-member function overloads:
relational operators
Relational operators for vector (function template )
swap Exchange contents of vectors (function template )
```

Map

-Member functions
(constructor)
Construct map (public member function)
(destructor)
Map destructor (public member function)
operator=
Copy container content (public member function)

-Iterators:
begin
Return iterator to beginning (public member function)
end
Return iterator to end (public member function)
rbegin
Return reverse iterator to reverse beginning (public member function)
rend
Return reverse iterator to reverse end (public member function)
cbegin
Return const_iterator to beginning (public member function)
cend
Return const_iterator to end (public member function)
crbegin
Return const_reverse_iterator to reverse beginning (public member function)
crend
Return const_reverse_iterator to reverse end (public member function)

-Capacity:
empty
Test whether container is empty (public member function)
size
Return container size (public member function)
max_size
Return maximum size (public member function)

-Element access:
operator[]
Access element (public member function)
at

```
Access element (public member function )

-Modifiers:
insert
Insert elements (public member function )
erase
Erase elements (public member function )
swap
Swap content (public member function )
clear
Clear content (public member function )
emplace
Construct and insert element (public member function )
emplace_hint
Construct and insert element with hint (public member function )

-Observers:
key_comp
Return key comparison object (public member function )
value_comp
Return value comparison object (public member function )

-Operations:
find
Get iterator to element (public member function )
count
Count elements with a specific key (public member function )
lower_bound
Return iterator to lower bound (public member function )
upper_bound
Return iterator to upper bound (public member function )
equal_range
Get range of equal elements (public member function )

-Allocator:
get_allocator
Get allocator (public member function )
```

Multimap

```
-Member functions
(constructor)
Construct multimap (public member function )
(destructor)
Multimap destructor (public member function )
operator=
Copy container content (public member function )

-Iterators:
begin
Return iterator to beginning (public member function )
end
Return iterator to end (public member function )
rbegin
Return reverse iterator to reverse beginning (public member function )
rend
Return reverse iterator to reverse end (public member function )
cbegin
Return const_iterator to beginning (public member function )
cend
Return const_iterator to end (public member function )
crbegin
Return const_reverse_iterator to reverse beginning (public member function )
crend
Return const_reverse_iterator to reverse end (public member function )

-Capacity:
empty
Test whether container is empty (public member function )
size
Return container size (public member function )
max_size
Return maximum size (public member function )

-Modifiers:
insert
Insert element (public member function )
erase
```

Erase elements (public member function)
swap
Swap content (public member function)
clear
Clear content (public member function)
emplace
Construct and insert element (public member function)
emplace_hint
Construct and insert element with hint (public member function)

-Observers:
key_comp
Return key comparison object (public member function)
value_comp
Return value comparison object (public member function)

-Operations:
find
Get iterator to element (public member function)
count
Count elements with a specific key (public member function)
lower_bound
Return iterator to lower bound (public member function)
upper_bound
Return iterator to upper bound (public member function)
equal_range
Get range of equal elements (public member function)

-Allocator:
get_allocator
Get allocator (public member function)

Set

```
-Member functions
(constructor)
Construct set (public member function )
(destructor)
Set destructor (public member function )
operator=
Copy container content (public member function )

-Iterators:
begin
Return iterator to beginning (public member function )
end
Return iterator to end (public member function )
rbegin
Return reverse iterator to reverse beginning (public member function )
rend
Return reverse iterator to reverse end (public member function )
cbegin
Return const_iterator to beginning (public member function )
cend
Return const_iterator to end (public member function )
crbegin
Return const_reverse_iterator to reverse beginning (public member function )
crend
Return const_reverse_iterator to reverse end (public member function )

-Capacity:
empty
Test whether container is empty (public member function )
size
Return container size (public member function )
max_size
Return maximum size (public member function )

Modifiers:
insert
Insert element (public member function )
erase
```

```
Erase elements (public member function )
swap
Swap content (public member function )
clear
Clear content (public member function )
emplace
Construct and insert element (public member function )
emplace_hint
Construct and insert element with hint (public member function )

-Observers:
key_comp
Return comparison object (public member function )
value_comp
Return comparison object (public member function )

-Operations:
find
Get iterator to element (public member function )
count
Count elements with a specific value (public member function )
lower_bound
Return iterator to lower bound (public member function )
upper_bound
Return iterator to upper bound (public member function )
equal_range
Get range of equal elements (public member function )

-Allocator:
get_allocator
Get allocator (public member function )
```