

Recursión

Recursión es Recursión

PCFIM

17 de enero de 2023

Recursión

¿Qué es una recursión?

¿Qué es una recursión?

Recursión

Una recursión es una técnica matemática utilizada en programación. Se dice que una función es recursiva cuando se llama a sí misma.

Una recursión puede definirse como una:

——recursión

Una recursión simple

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  int Fibo(int n){
5      if(n <= 1) return n;
6      return Fibo(n-1) + Fibo(n-2);
7  }
8
9  int main(){
10     cout << Fibo(60);
11     return 0;
12 }
```

Problemas:

- Muy lento: $O(\varphi^n)$.
- Recursión muy profunda.
- Overflow (?)

Mejorando la recursión

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  #define N 10000
4  int f[N];
5  int Fibo(int n){
6      if(n <= 1) return n;
7      if(f[n] != 0) return f[n];
8      f[n] = Fibo(n-1) + Fibo(n-2);
9      return f[n];
10 }
11
12 int main(){
13     cout << Fibo(60);
14     return 0;
15 }
```

Problemas:

- Muy lento: $\mathcal{O}(\phi^n)$.
- Recursión muy profunda.
- Overflow (?)

Problemas :D

Dado un entero N , imprimir el fractal de Sierpinski de profundidad N .

Sierpinski de profundidad 1:



A Sierpinski triangle of depth 1, represented by three lines forming a triangle: a top line with a backslash on the left and a forward slash on the right, and a bottom line with a forward slash on the left and a backslash on the right.

Sierpinski de profundidad 2:



A Sierpinski triangle of depth 2, represented by a larger triangle composed of four smaller triangles of depth 1. The top line is a backslash on the left and a forward slash on the right. The second line has a forward slash on the left and a backslash on the right. The third line consists of two smaller triangles, each with a backslash on the left and a forward slash on the right. The bottom line consists of two smaller triangles, each with a forward slash on the left and a backslash on the right.

Fractal de Sierpinski

```
1 void fractal(int n, int h, int l){
2     if (n == 1){
3         mat[h][l] = '/'; mat[h][l+1] = '\\';
4         mat[h+1][l-1] = '/'; mat[h+1][l] = '_';
5         mat[h+1][l+1] = '_'; mat[h+1][l+2] = '\\';
6         return;
7     }
8     fractal(n-1, h, l);
9     fractal(n-1, h + (1<<n-1), l - (1<<n-1));
10    fractal(n-1, h + (1<<n-1), l + (1<<n-1));
11 }
12
13 int main(){
14     memset(mat, ' ', sizeof mat);
15     int n = 5;
16     fractal(n, 1, 1<<n);
17     for (int i = 1; i <= (1<<n); i++){
18         for (int j = 1; j <= (1<<n) + i; j++) cout << mat[i][j];
19         cout << '\n';
20     }
21     cout << '\n';
22     return 0;
23 }
```


Thanos Sort

Thanos sort es un algoritmo de ordenación de supervillanos, que funciona de la siguiente manera: si el array no está ordenado, chasquea los dedos para eliminar la primera o la segunda mitad de los elementos y repite el proceso. Dado un array de entrada, ¿cuál es el tamaño del array ordenado más largo que puede obtener usando Thanos sort?

Se requiere Infinity Gauntlet.

Thanos Sort

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int solve(int l, int r, vector <int> &arr){
4      if(r - l == 0) return 0;
5      bool found = true;
6      for(int i=l; i<r; i++){
7          if(arr[i] > arr[i+1]){
8              found = false;
9              break;
10         }
11     }
12     if(found) return r-l;
13     else return(max(solve(l, l+(r-l)/2, arr), solve(l+(r-l)/2+1, r, arr)));
14 }
15 int main(){
16     int n;
17     cin >> n;
18     vector <int> arr(n);
19     for(int i=0; i<n; i++) cin >> arr[i];
20     cout << 1+solve(0, n-1, arr);
21     return 0;
22 }
```

Un poquito de backtracking

Dado un tablero de ajedrez ordinario (8×8), de cuantas formas es posible colocar 8 reinas en el tablero, tal que ninguna esté atacando a otra.

N Reinas

```
1 void solve(vector <int> &table, int p){
2     if(p == N-1){
3         print(); return;
4     }
5     for(int j=0; j<N; j++){
6         table[p+1] = j;
7         bool cancel = false;
8         for(int i=0; i<p+1; i++){
9             if(table[i] == table[p+1]){
10                 cancel = true; break;
11             }
12             if(abs(table[p+1] - table[i]) == p+1-i){
13                 cancel = true; break;
14             }
15         }
16         if(cancel) continue;
17         solve(table, p+1);
18     }
19     table[p+1] = -1;
20     return;
21 }
```
