

Otimização temporal / espacial

Por: Gustavo Xavier Pereira

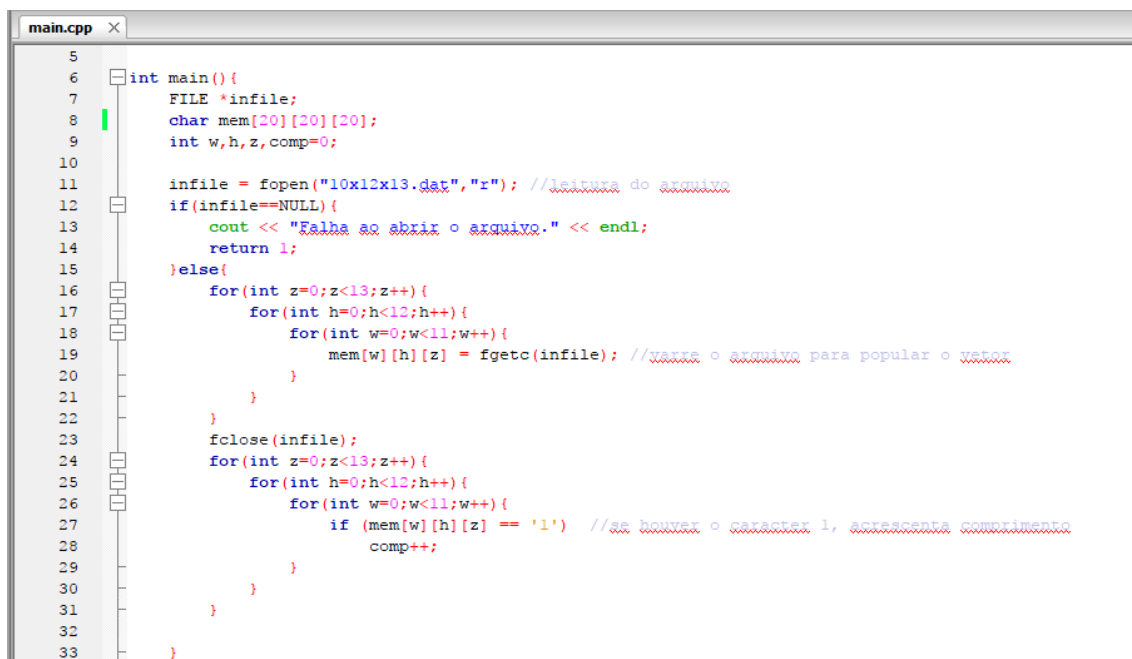
Release #1

O primeiro release consiste apenas da resolução do problema, portanto está livre de qualquer otimização.

OBS: O arquivo foi modificado para melhor leitura pelo algoritmo, já que os espaços estavam sendo lidos, assim prejudicando a determinação das coordenadas.

OBS2: Não foi possível alocar os arquivos 100x150x200.dat e 300x400x450.dat, pois os arquivos são demasiados grandes, e o algoritmo não suporta uma matriz tridimensional de tais dimensões, além minha falta de conhecimento sólido em programação e da linguagem c++.

Segue Screenshots do código:



```
5
6 int main(){
7     FILE *infile;
8     char mem[20][20][20];
9     int w,h,z,comp=0;
10
11     infile = fopen("10x12x13.dat","r"); //leitura do arquivo
12     if(infile==NULL){
13         cout << "Falha ao abrir o arquivo." << endl;
14         return 1;
15     }else{
16         for(int z=0;z<13;z++){
17             for(int h=0;h<12;h++){
18                 for(int w=0;w<11;w++){
19                     mem[w][h][z] = fgetc(infile); //varre o arquivo para popular o vetor
20                 }
21             }
22         }
23         fclose(infile);
24         for(int z=0;z<13;z++){
25             for(int h=0;h<12;h++){
26                 for(int w=0;w<11;w++){
27                     if (mem[w][h][z] == '1') //se houver o caracter 1, aumenta o comprimento
28                         comp++;
29                 }
30             }
31         }
32     }
33 }
```

Nessa screenshot estão a leitura do arquivo e dois laços, no qual um é responsável pela leitura do arquivo e alocação dos caracteres na memória; enquanto outro é responsável por checar os caracteres '1', para assim determinar o comprimento.

```
main.cpp x
39 do{
40     if (w<11 && mem[w+1][h][z] == '1' && coordw == true){
41         i++;
42         w++;
43         coordh = true;
44         coordz = true;
45     }
46     else
47     if (w>0 && mem[w-1][h][z] == '1'){
48         w--;
49         i++;
50         coordw = false;
51         coordh = true;
52         coordz = true;
53     }
54     if (h<12 && mem[w][h+1][z] == '1' && coordh == true){
55         i++;
56         h++;
57         coordw = true;
58         coordz = true;
59     }
60     else
61     if (h>0 && mem[w][h-1][z] == '1'){
62         i++;
63         h--;
64         coordw = true;
65         coordh = false;
66         coordz = true;
67     }
68     if (z<13 && mem[w][h][z+1] == '1' && coordz == true){
```

```
main.cpp X
63         h--;
64         coordw = true;
65         coordh = false;
66         coordz = true;
67     }
68     if (z<13 && mem[w][h][z+1] == '1' && coordz == true)
69         i++;
70         z++;
71         coordw = true;
72         coordh = true;
73     }
74     else
75     if (z>0 && mem[w][h][z-1] == '1'){
76         i++;
77         z--;
78         coordw = true;
79         coordh = true;
80         coordz = false;
81     }
82     }while (i<comp - 1);
83
84     cout << "Comprimento: " << comp << endl;
85     cout << "W = " << w << endl;
86     cout << "H = " << h << endl;
87     cout << "Z = " << z << endl;
88
89     return 0;
90 }
91
```

Nessas duas screenshots está o algoritmo que percorre o caminho, um laço principal de FAÇA-ENQUANTO, tendo como parâmetro comparativo uma variável $i < comp$, ou seja, entrará nos *if's* a cada andamento do comprimento, procurando por caracteres "1".