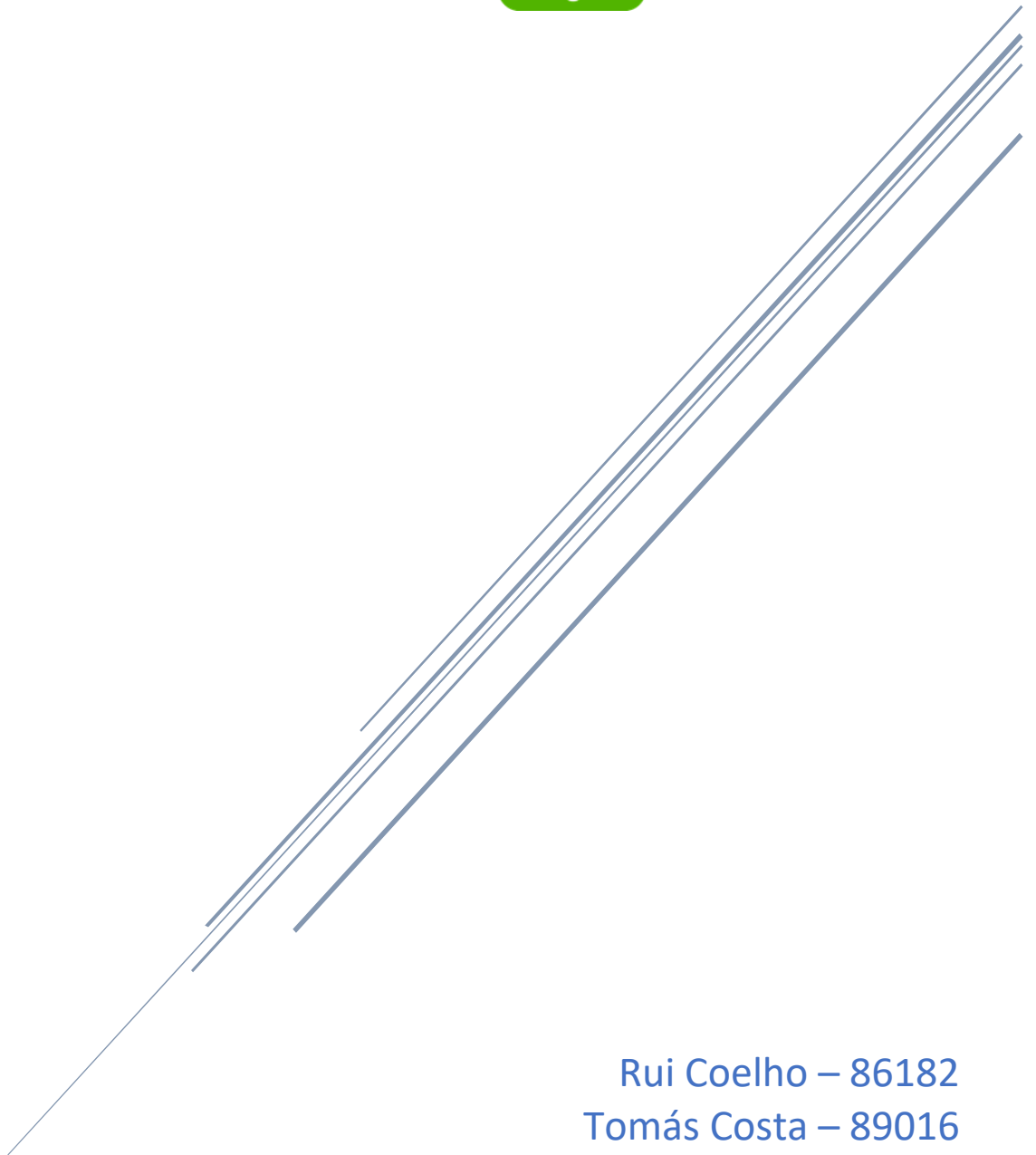


CONNECTIVITY USING UNION-FIND

Algoritmos e Estruturas de dados



universidade
de aveiro



Rui Coelho – 86182
Tomás Costa – 89016
João Carvalho - 89059

Índice

Union-Find.....	3
Problema	4
Resolução do problema.....	4
Representação da solução – Matlab	5
Solução do problema	5
Aproximação em C	5
Aproximação em Matlab.....	6
Validação de implementação.....	7
Grelha 500 x 500	7
Referências bibliográficas	9
Netgrafia.....	9
Anexos.....	10
Código C	10
Código Matlab	19

Índice de Ilustrações

<i>Figura 1 - Exemplo de funcionamento</i>	3
<i>Figura 2 - Figura representativa do problema</i>	4
<i>Figura 3 - Array de representantes e Matrix A (30x20)</i>	5
<i>Figura 4 - Matriz B e tempos de execução (30x20)</i>	6
<i>Figura 5 - Caminho identificado com a grelha 30 x 20</i>	6
<i>Figura 6 - Blocos retirados da grelha 30 x 20</i>	6
<i>Figura 7 - Execução para a grelha de 500 x 500</i>	7
<i>Figura 9 - Caminho identificado com a grelha 500 x 500</i>	7
<i>Figura 8 - Blocos retirados da grelha 500 x 500</i>	7

Union-Find

A estrutura de dados “Union-Find” é uma maneira eficiente de acompanhar os componentes conectados de um gráfico não direcionado. (Existem outras aplicações desta estrutura de dados.) As suas principais operações são:

- ✚ [Union] Substituir os componentes conectados contendo vértices p e q com sua união (se os vértices pertencem ao mesmo componente, então não há nada a fazer. Caso contrário, junte os dois componentes);
- ✚ [Find] Dado um vértice p , encontre o vértice q que representa o componente conectado que pertence. Para manter a informação de conectividade de um gráfico atualizado para cada inserção de aresta, faça uma operação de união de seus vértices de incidente. Depois verifica-se se os dois vértices estão conectados para verificar se os representantes dos componentes conectados são os mesmos.

No exemplo a seguir, uma aresta entre os vértices 1 e 2 é inserida no gráfico. Neste caso, a operação de união correspondente junta dois componentes conectados.

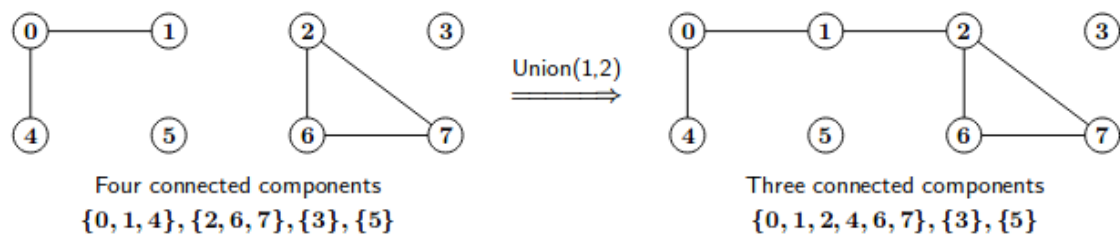


Figura 1 - Exemplo de funcionamento

Um dos vértices de cada componente conectado será o vértice representativo desse componente. Para um gráfico sem arestas, cada vértice é o vértice representativo de seu componente (cada componente possui apenas um vértice).

Cada operação de união faz com que o representante do componente de seu segundo argumento seja o representante do primeiro.

Problema

Duas regiões estão separadas por uma grelha $W \times H$ feita de quadrados (com $W = 30$, $H = 20$). Cada quadrado da grelha pode ser branco ou preto. Inicialmente, todos os quadrados são pretos. Os quadrados pretos, escolhidos ao acaso, são então transformados em brancos. Em média, quantos quadrados pretos precisam ser transformados em brancos até que haja um caminho feito inteiramente de quadrados brancos entre as duas regiões?

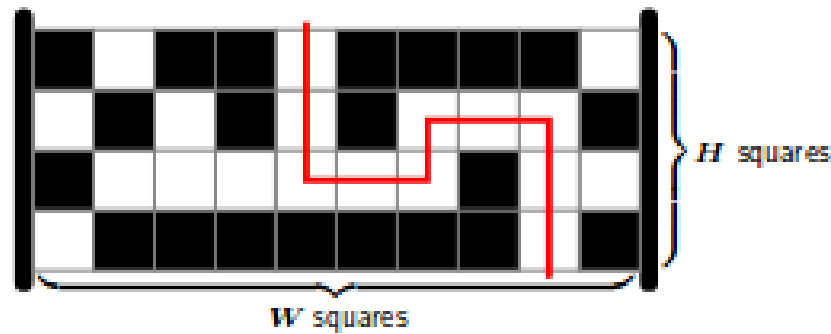


Figura 2 - Figura representativa do problema

Resolução do problema

Para resolver este problema optamos pela criação de 3 arrays sendo:

- ✚ Representantes de cada ponto – Inicializado com o próprio ponto.
- ✚ Tamanhos das árvores – Inicializado a 1 para otimizar a função “Union” tendo em conta os tamanhos de cada árvore.
- ✚ Array bidimensional – Dimensões da imagem (que será usado em Matlab para fazer o processamento da imagem e ser mais fácil de visualizar a solução). Este array resulta da junção entre dois arrays unidimensionais através da função `getArray()` que converte as coordenadas bidimensionais para o índice correspondente.

Após inicializados os dados implementámos as funções “Union” e “Find”.

Na nossa função “Find” que devolve a raiz de um determinado ponto e utilizamos uma técnica chamada “path compression” na sua forma recursiva que permite encurtar o caminho de um ponto para a sua raiz colocando no lugar do representante do ponto a raiz.

Na função “union(uni)” aplicámos aquilo que se chama o “weighted-union” que liga sempre a menor árvore à maior para tornar o processo da função “Find” mais rápido.

Para finalizar criámos as funções `check()` e `adjacent()` que são chamadas a cada iteração do nosso ciclo principal. A função `adjacent()` verifica se existem pontos a branco adjacentes a um dado ponto e faz o “Union” caso existam de facto pontos a branco.

A função `check()` é utilizada no final de cada iteração para verificar se já foi encontrada solução. Isto é feito percorrendo a primeira e última linha da imagem e procurar se existem raízes iguais no topo e no fundo. Se houver, quer dizer que existem componentes ligadas e há pelo menos um caminho do topo ao fundo.

Representação da solução – Matlab

Solução do problema

Aproximação em C

Para além de serem impressos na consola estes valores da Matriz A e Matriz B são guardados nos seus respetivos ficheiros (matrizA.txt e matrizB.txt) para posteriormente serem carregados e processados no Matlab.

- 📊 Número médio de pontos por imagem
- 📊 Média de pontos preenchidos por imagem
- 📊 Percentagem de pontos a branco

[illegible]

Figura 3 - Array de representantes e Matrix A (30x20)

```

rc@cube: ~/Rui/Universidade/5_semestre/AED/Union_Find
Matriz A:
0 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 0 1 1 1
0 1 1 1 0 1 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 1 0 0 1 1 1 0 0 1 1 0 0 0 0 0 1 1 1 1
1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0
0 1 0 1 0 0 0 1 1 1 1 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1 0 1
1 1 0 0 1 0 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 0 0 0 1 1
1 1 0 0 0 1 1 1 1 1 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 1 1
0 1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 0 0
1 1 1 1 0 0 0 1 1 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 1
0 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 1 1 0 1 0 0 1 1 1 0 1
1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 1 2
1 1 0 0 1 1 0 0 1 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0 1 1 1
0 1 1 0 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1
0 1 1 1 1 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 1 1 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 1 0 0 1 1 0 1 0 1 0
1 1 1 0 1 1 0 0 0 1 1 0 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0
0 1 1 1 1 0 0 1 1 1 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
1 0 0 1 0 0 1 1 1 0 0 1 0 0 1 1 0 0 1 1 1 1 0 0 0 1 1 1
1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 0 1 0 1 1 1 1 1 0 1
1 1 0 0 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 0 0
Matriz B:
0 1 1 1 1 0 0 0 2 2 2 2 2 2 2 0 0 0 1 1 0 1 0 0 1 0 1 1 1
0 1 1 1 0 1 1 0 2 2 2 2 2 2 2 0 0 1 1 1 1 1 1 0 0 0 0 0 0
1 1 0 1 0 0 1 1 0 2 2 2 2 2 2 0 0 1 1 0 0 0 0 0 1 1 1 1 1
1 1 1 1 1 1 0 1 1 0 0 0 2 2 2 2 2 0 1 0 1 0 1 0 0 0 0 1 0 1 0
0 1 0 1 0 0 0 0 1 1 1 0 0 2 2 2 2 2 0 1 1 1 0 0 0 1 0 0 0
1 1 0 0 1 0 1 1 1 1 1 1 1 0 2 2 2 2 0 0 1 1 1 1 1 0 0 0 1 1
1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 2 2 2 0 1 0 0 0 1 1 1 0 1 1 1 1
0 2 0 0 1 0 1 1 1 0 1 1 0 2 2 2 2 0 1 1 1 0 1 1 1 1 0 0 0
2 2 2 2 2 0 0 0 1 1 1 1 0 0 1 0 2 2 2 0 0 1 0 1 0 1 0 0 0 1 1
0 0 0 2 2 0 2 0 0 1 1 0 2 0 0 2 2 2 0 1 0 0 1 0 1 1 1 0 1
1 0 0 2 2 0 2 2 2 2 0 0 2 2 0 2 2 0 1 1 0 0 1 0 1 1 1
1 1 1 0 2 2 0 0 2 0 0 2 2 2 2 2 2 0 0 1 0 1 1 0 0 1 1 1
0 1 1 0 2 2 0 2 2 2 0 2 2 2 2 2 2 0 2 0 1 0 1 1 0 0 1 1
0 1 1 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 0 0 0
0 1 1 1 0 0 2 2 2 2 2 0 0 2 2 2 2 2 2 2 0 0 1 1 0 1 0 1 0
1 1 1 0 1 1 0 0 0 2 2 0 2 2 2 2 2 2 2 2 2 0 1 0 0 0 1 0 0
0 0 1 1 1 1 0 0 2 2 2 2 2 2 2 0 0 0 2 0 0 0 0 1 1 0 0
1 1 0 0 1 0 0 1 1 1 0 0 2 2 2 0 0 2 2 0 0 2 2 2 0 0 1 1 1 1
1 1 0 1 0 1 0 1 1 0 2 2 2 0 2 2 0 2 2 2 2 0 2 0 1 1 1 1 0 1
1 1 1 0 1 0 0 1 1 0 2 2 2 2 2 2 2 0 2 2 0 1 1 1 1 0 0 0
Media de pontos gerados por imagem: 504.456938
Media pontos preenchidos por imagem: 340.127546
Percentagem de pontos a branco: 56.687924
661.73user 0.79system 9.24.06elapsed 99%CPU (0avgtext+0avgdata 1680maxresident)k
0inputs+160outputs (0major+64minor)pagefaults 0swaps

```

Figura 4 - Matriz B e tempos de execução (30x20)

Aproximação em Matlab

Após obtermos os valores em C demos início ao processamento em matlab sendo geradas sempre duas figuras, uma primeira na qual são representados os blocos retirados da grelha e na segunda o caminho identificado.

Para grelha de 30 de comprimento e 20 de altura foram obtidas as imagens abaixo.



Figura 6 - Blocos retirados da grelha 30 x 20

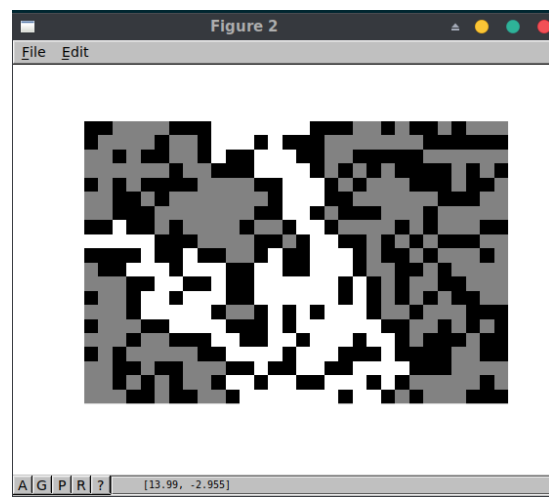




Figura 5 - Caminho identificado com a grelha 30 x 20

Validação de implementação

Para validar a implementação da nossa solução optamos por realizar mais testes, realizados testes para grelhas maiores tendo sido executados os seguintes:

-  Grelha 500 x 500 com dois nós de execução
-  Grelha 2000 x 2000 com 1 nó de execução

Grelha 500 x 500

Tal como aconteceu na resolução do problema proposto começamos por executar o nosso algoritmo em C tendo gerado a matriz A e B e o array dos representantes.

```
Média de pontos gerados por imagem:221904.000000
Média pontos preenchidos por imagem:147097.000000
Percentagem de pontos a branco:58.838800

126.87user 0.01system 2:06.89elapsed 99%CPU (0avgtext+0avgdata 4600maxresident)k
0inputs+7128outputs (0major+796minor)pagefaults 0swaps
~/Rui/Universidade/5 semestre/AED/Union Find/testes/Grelha 500x500 > master ➔
```

Figura 7 - Execução para a grelha de 500 x 500

Posteriormente processamos ambas as matrizes em Matlab obtendo assim as representações desejadas.

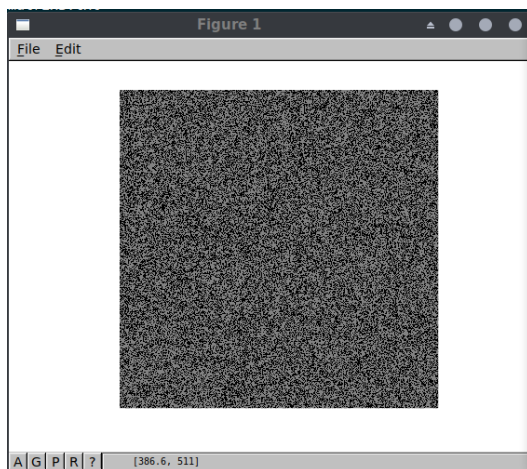


Figura 9 - Blocos retirados da grelha 500 x 500

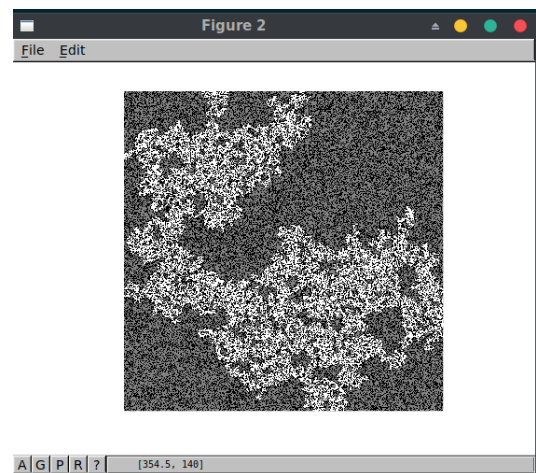


Figura 8 - Caminho identificado com a grelha 500 x 500

Referências bibliográficas

1. The Algorithm Design Manual, Steven S. Skiena, second edition, Springer, 2008.
Consultado a 17 de dezembro de 2018
2. Introduction to Algorithms, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, third edition, The MIT press, 2009.
Consultado a 18 de dezembro de 2018
3. Algorithms, Robert Sedgewick and Kevin Wayne, fourth edition, Addison Wesley, 2011
Consultado a 20 de dezembro de 2018

Netgrafia

1. https://elearning.ua.pt/pluginfile.php/214852/mod_resource/content/48/AED.pdf, Tomás Oliveira e Silva, Universidade de Aveiro, Portugal
2. <https://algs4.cs.princeton.edu/lectures/42DirectedGraphs.pdf>, Sedgewick and Wayne, Princeton University, USA
3. <https://algs4.cs.princeton.edu/lectures/15UnionFind.pdf>, Sedgewick and Wayne, Princeton University, USA
4. <https://algs4.cs.princeton.edu/lectures/41UndirectedGraphs.pdf>, Sedgewick and Wayne, Princeton University, USA

Anexos

Código C

```
#include <stdio.h>
```

```
#include <assert.h>
```

```
#include <stdlib.h>
```

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

```
/**
```

```
 * Comprimento da tabela.
```

```
 */
```

```
#define W 30
```

```
/**
```

```
 * Altura da tabela.
```

```
 */
```

```
#define H 20
```

```
/**
```

```
 * Nós de execução.
```

```
 */
```

```
#define EXEC 1e6
```

```
/**
```

```
 * Array com pixels da imagem.
```

```
 * 0 - Pixel preto
```

```
 * 1 - Pixel branco
```

```
 *
```

```
 * A inicialização é feita (por defeito) a 0
```

```
 */
```

```
int image[H][W];
```

```

/**
 * Array unidimensional de representantes de cada pixel.
 */
int array[W*H];

/**
 * Tamanho do conjunto com essa raíz.
 * Usado para manter os unions equilibrados.
 */
int size[W*H];

/**
 * Transforma coordenadas 2D num índice de um array unidimensional.
 */
int getArray( int x, int y ) {
    return array[x*W+y];
}

/**
 * Imprime o array de representantes na consola.
 */
void show(int arr[])
{
    printf("[");
    for(int i = 0; i < H*W; i++)
        printf("%d ", arr[i]);
    printf("]\n");
}

```

```

/**
 * Gera a estrutura dos arrays para matlab.
 * Estes arrays são guardados em dois ficheiros:
 *
 * matrixA.txt
 * matrizB.txt
 *
 * Após tudo gerado será tudo processado em Matlab / Octave.
 */

```

```

void showImage(int arr[H][W], int a)
{
    if (a == 1){
        FILE *fp = fopen("matrixA.txt","w");
        printf("Matriz A - \n");
        if (fp == NULL)
        {
            printf("Error opening file!\n");
            exit(1);
        }
        for(int i = 0; i < H; i++) {
            for(int j = 0; j < W; j++){
                fprintf(fp, "%d ", arr[i][j]);
                printf("%d ", arr[i][j]);
            }
            if(i==H-1){
                fprintf(fp, "\n");
                printf("\n");
            }
            else{
                fprintf(fp, "\n");
                printf("\n");
            }
        }
    }
}

```

```
/**
 *          CONECTIVITY USING UNION-FINDING
 */
```

```
/**
 * Inicializa os representantes de todos os píxeis
 * como eles próprios e a imagem a zeros.
 */
```

```
void init()
{
    for(int i = 0; i < H; i++) {
        for(int j = 0; j < W; j++) {
            image[i][j] = 0;
            array[i*W+j] = i*W+j;
            size[i*W+j] = 1;
        }
    }
}
```

```
/**
 * Função que encontra a raíz de um píxel e comprime
 * o caminho definindo a raíz do conjunto como um
 * novo representante.
 */
```

```
int find(int i)
{
    if(array[i] != i)
        array[i] = find(array[i]);

    return array[i];
}
```

```
/**  
 * Procura a raíz de 2 pixeis e une os conjuntos tendo  
 * em conta o peso de cada um, unindo sempre o maior  
 * ao menor para obter finds mais rápidos.  
 */
```

```
void uni(int x, int y)  
{  
    int raiz_x = find(x);  
    int raiz_y = find(y);  
    if(size[raiz_x] > size[raiz_y]) {  
        array[raiz_y] = raiz_x;  
        size[raiz_x] += size[raiz_y];  
    }  
  
    else{  
        array[raiz_x] = raiz_y;  
        size[raiz_y] += size[raiz_x];  
    }  
}
```

```

/**
 * Função que verifica a vizinhança de um píxel
 * tendo sempre em conta os limites da imagem.
 */
void adjacent(int x, int y)
{
    /**
     * Verificar de baixo.
     */
    if(x > 0 && image[x-1][y] != 0)
        uni(getArray(x-1, y), getArray(x,y));
    /**
     * Verificar pixel de cima.
     */
    if(x < H-1 && image[x+1][y] != 0)
        uni(getArray(x+1, y), getArray(x,y));
    /**
     * Verificar pixel à esquerda.
     */
    if(y > 0 && image[x][y-1] != 0)
        uni(getArray(x, y-1), getArray(x,y));
    /**
     * Verificar pixel à direita.
     */
    if(y < W-1 && image[x][y+1] != 0)
        uni(getArray(x, y+1), getArray(x,y));
}

```



```

/**
 * Verifica se existe algum conjunto que esteja
 * ao mesmo tempo na 1ª e na última linha,
 * caso exista um conjunto nessa situação quer
 * dizer que há pelo menos um caminho entre o topo
 * e o fundo.
 */
int check(void)
{
    for(int i = 0; i < W; i++) {
        for(int j = (H-1)*W; j < H*W; j++) {
            if (find(i) == find(j)) {
                return find(i);
            }
        }
    }
    return 0;
}

```

```

/**
 * Desenha o caminho na imagem.
 * Este desenho será processado em matlab com o
 * ficheiro resuults.m.
 */
void draw(int h) {
    for(int i = 0; i < H; i++) {
        for(int j = 0; j < W; j++)
            if (find(getArray(i,j)) == h)
                image[i][j] = 2;
    }
}

```

```
/**
 * Main function
 */
int main(int argc, char **argv)
{
    /**
     * Seed para obter sempre o mesmo resultado.
     */
    srand(80013);

    /**
     * Declarações.
     */
    int x, y, a, cont, unicos, total = 0, resolvido, total_gerados = 0;

    /**
     * Ciclo principal do programa.
     */
    for(int i = 0; i < EXEC; i++) {
        // inicializar os arrays.
        init();
        unicos = 0;
        resolvido = 0;
        cont = 0;
```

```

/** Enquanto que não houver um caminho entre o topo e o fundo. */
while(resolvido == 0) {
    cont++;

    /** Gerar novas coordenadas. */
    x = rand() % H;
    y = rand() % W;

    /* Se esse ponto ainda não tinha sido gerado */
    if (image[x][y] == 0) {
        image[x][y] = 1;
        adjacent(x, y);
        unicos++;
        resolvido = check();
    }
}

total += unicos;
total_gerados += cont;
}

double media_gerados = total_gerados/EXEC;
double media = total/EXEC;
double percentagem = media/(W*H)*100;

a = 1;
printf("Array de representantes: \n");
show(array);
showImage(image, a);
draw(resolvido);

a = 2;
showImage(image, a);

printf("Média de pontos gerados por imagem:%lf\nMédia pontos preenchidos por
imagem:%lf\nPercentagem de pontos a branco:%lf\n",media_gerados, media, percentagem);
}

```

Código Matlab

```
%% Ler matriz A.  
b = load("matrixA.txt");  
%%  
%% Ler matriz B.  
a = load("matrixB.txt");  
%%  
%% Display da primeira imagem.  
figure(1)  
%%  
%% Passar os pixels para uma escala de 0 a 1  
b=b/2;  
%%  
imshow(b);  
%% Passar os pixels para uma escala de 0 a 1  
a = a/2;  
%%  
%% Os pixels que originalmente tinham 0 ficam a preto,  
%% os que tinham 1 ficam a cinza e os que tinham 2  
%% ficam a branco  
figure(2)  
%%  
imshow(a);
```