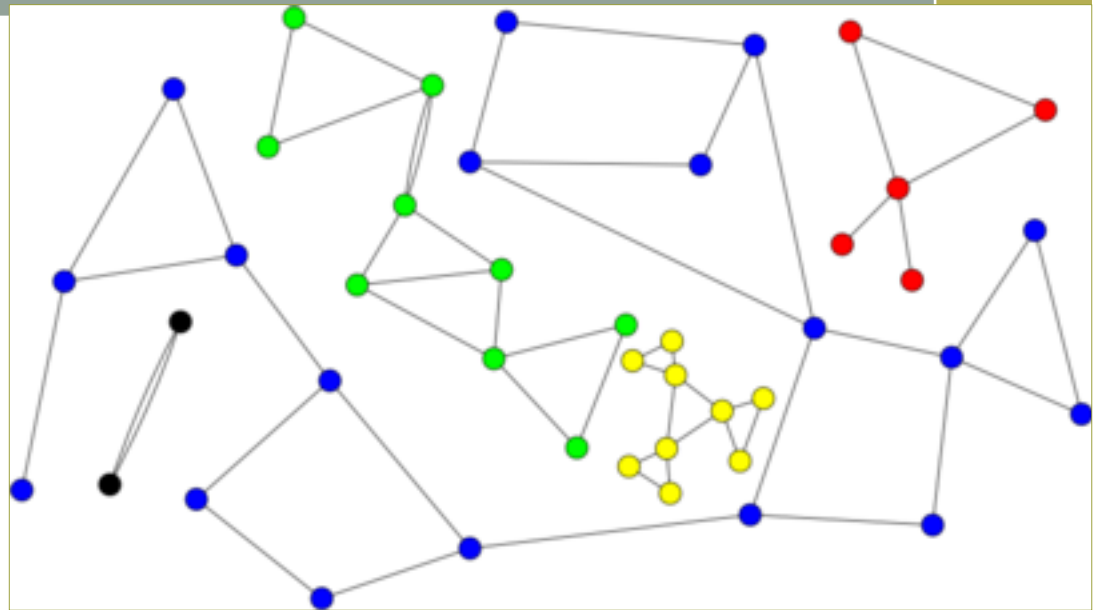


2011

Trabalho de EDA



INDICE

Apresentação	3
Objectivos.....	4
Algoritmo.....	5
Connected Components Labeling	5
Primeira passagem	5
Segunda Passagem	7
União com todos.....	8
Disjoint-set forest	8
Path compression	9
Union By Rank	10
Path Compression and Union By Rank	10
Resultados obtidos.....	12
Recursos.....	12
Imagens.....	12
Resultados.....	13
Imagens obtidas	17
Conclusões	19
Bibliografia	20

APRESENTAÇÃO



Bruno Alexandre da Silva Moreira

Aluno nº 6170

Curso de Eng^a Informática

Escola Superior de Tecnologia e Gestão

OBJECTIVOS

Trabalho individual desenvolvido no âmbito da disciplina de Estruturas de Dados e Algoritmos, da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Beja, curso de Informática de Gestão. Este tem como objectivo criar um programa que permita ler imagens binárias através do algoritmo *connected components* e identifique os elementos conexos de uma imagem. Após esta identificação deverá ser gerada uma nova imagem com os elementos conexos pintados de cores diferentes e analisado o tempo de processamento para diferentes imagens. Após esta análise deverá ser deduzida a complexidade do tempo de processamento do programa que foi desenvolvido.

ALGORITMO

O algoritmo utilizado para o desenvolvimento deste programa é o *connected components labelling*, tal como descrito no livro “Introduction to Algorithms” 2nd Ed.

Este algoritmo, normalmente é utilizado para identificar áreas conexas em imagens binárias, e consiste em efectuar duas passagens na imagem, detectando as áreas não conexas de cada imagem atribuindo-lhe uma marca diferente.

CONNECTED COMPONENTS LABELING

O algoritmo *connected components labeling*, consiste em analisar pixel a pixel as imagens por duas vezes, sendo que na primeira identifica vários marcadores par zonas distintas e correspondências de zonas que se tocam

Abaixo é apresentado o pseudocódigo utilizado para o desenvolvimento do referido algoritmo.

```
algorithm TwoPass(data)
    linked = []
    labels = structure with dimensions of data, initialized with the value of Background

    First pass
    for row in data:
        for column in row:
            if data[row][col] is not Background

                neighbors = connected elements with the current element's label

                if neighbors is empty
                    linked[NextLabel] = set containing NextLabel
                    labels[row][column] = NextLabel
                    NextLabel += 1

                else
                    Find the smallest label
                    L = neighbors labels
                    labels[row][column] = min(L)
                    for label in L
                        linked[label] = union(linked[label], L)

    Second pass
    for row in data
        for column in row
            if labels[row][column] is not Background
                labels[row][column] = min(linked[labels[row][column]])

    return labels
```

Ilustração 1 - Pseudocódigo para connected components labeling

PRIMEIRA PASSAGEM

Em primeiro lugar é definida a cor de fundo da imagem, sendo que os pixéis identificados como tal não são considerados neste algoritmo.

Esta primeira passagem consiste em atribuir diferentes marcadores às áreas detectadas, que se crêem serem áreas não conexas.

Todos os pixéis são analisados da esquerda para a direita até atingir o final de cada linha, passando então para linha seguinte até que se tenha analisado todos os pixéis da imagem.

Relativamente ao *pixel* a ser analisado são identificados os pixéis vizinhos, nomeadamente o pixel imediatamente à sua esquerda, e na linha de cima os pixéis à esquerda, direita e cima, tal como demonstrado na ilustração 1, onde os vizinhos estão identificados a verde.

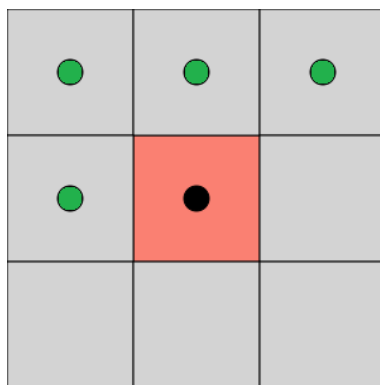


Ilustração 2 - Vizinhos

Assim, no caso de os pixéis vizinhos não terem qualquer marca, o *pixel* actual é considerado uma nova área e consequentemente é-lhe atribuída uma nova marca, por exemplo “1”.

No entanto caso os seus vizinhos tenham alguma marca atribuída, significa que o pixel que está a ser analisado é conexo relativamente a uma área anteriormente identificada, e consequentemente não será criada uma nova marca. Assim o *pixel* actual terá a marca correspondente à marca com valor mínimo da lista de marcas encontradas, por exemplo, no caso de os seus vizinhos terem as marcas “2”, “3” e “7”, ao *pixel* actual seria atribuída a marca “2”.

Como as marcas “2”, “3” e “7” são vizinhas entre si, significa que estas são conexas logo trata-se da mesma área. Desta forma é necessário criar uma lista de equivalências onde fique identificado que as marcas anteriormente indicadas fazem parte da mesma área, assim como as marcas que já existiam na lista de equivalências das marcas identificadas.

Por exemplo, no caso de apenas a marca “2” já ter uma equivalência identificada, e esta ser a marca “5”, significa que a lista de equivalências da marca “2” passe a ser “2”, “3”, “5” e “7”. Mas como as marcas “3”, “5” e “7” são equivalentes à marca “2”, também elas têm que ter como equivalências as marcas equivalentes a “2”, ou seja “2”, “3”, “5” e “7”.

SEGUNDA PASSAGEM

A segunda passagem consiste em percorrer novamente cada *pixel*, e tal como na primeira passagem ignorando os pixéis identificados como fundo.

Sempre que o *pixel* tem uma marca, é validada a lista de equivalências, para saber se essa marca equivale a outra.

No caso de haver uma lista de equivalências a marca atribuída a ao *pixel* analisado será igual à menor marca da lista de equivalências. Por exemplo se o pixel analisado tem a marca “3”, mas a sua lista de equivalências é “2”, “3”, “5” e “7”, a sua marca será substituída pela marca “2”.

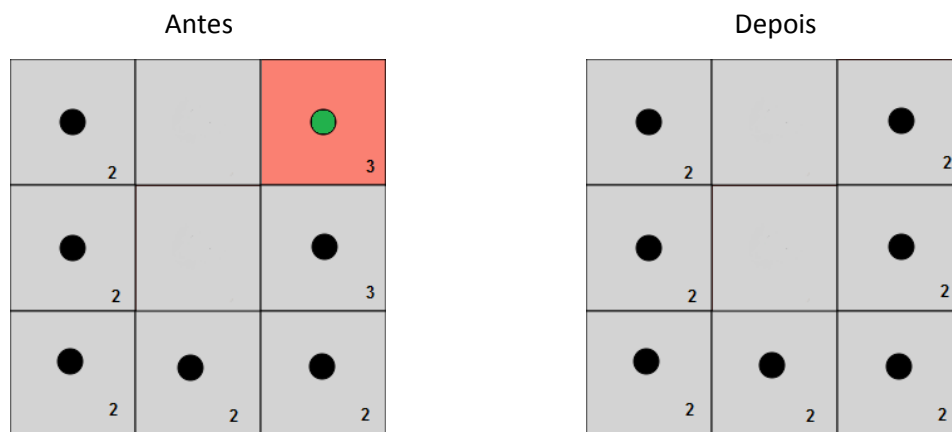


Ilustração 3 - Exemplo da segunda passagem

UNIÃO COM TODOS

Esta método aplica à risca o algoritmo connected components labelling, sendo que para cada marca faz corresponder uma lista de todos os vizinhos, e para cada vizinho todos os seus vizinhos.

Apesar de ser um código relativamente simples e rápido de implementar, este demostra ser tanto mais lento quanto mais forem as marcas encontradas, pois para cada vizinho são identificadas as correspondências anteriormente identificadas para esse vizinho e agrupar essas correspondências num novo conjunto de correspondências

Por sua vez para cada marca desse novo conjunto tem que se actualizar a lista das correspondências com o novo conjunto anteriormente criado.

Este método apresenta um tempo de processamento de $O(n^2)$.

Abaixo é apresentado o código que faz a união das correspondências.

```
#-----  
# joins sets of identified neighbors  
#  
# neighbors      - set containing neighbor labels found  
#-----  
def union(self, neighbors):  
    # prepare set of labels  
    new_set = set()  
  
    # build new set with neighbor elements  
    for n in neighbors:  
        new_set.update(self.equivalences[n-1])  
  
    # add set to all identified labels  
    for n in new_set:  
        self.equivalences[n-1].update(new_set)  
    pass  
pass
```

Ilustração 4 - União com todos, código do método union

DISJOINT-SET FOREST

Este algoritmo é mais rápido que o anterior, pois consiste não em criar constantemente uma lista de equivalências e actualizar essa lista pelas marcas equivalentes, mas sim em identificar as marcas equivalentes em forma de árvore, permitindo assim que para a identificação da marca menor não seja necessário passar por todas as equivalências.

Abaixo na ilustração 5 é exemplificado este algoritmo, que referencia os vários filhos da marca “f”.

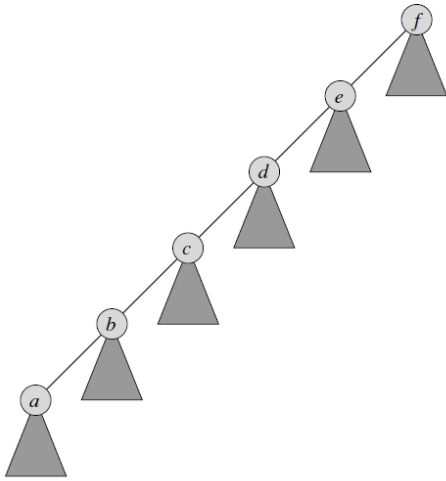


Ilustração 5 - Disjoint Set Forest

É no entanto possível fazer pelo menos dois melhoramentos ao algoritmo, atribuindo um ranking a cada marca e comprimindo o caminho até chegar ao pai das equivalências.

PATH COMPRESSION

Este melhoramento consiste em fazer corresponder a cada elemento do conjunto o pai directamente, por forma a não haver necessidade de passar por cada um dos filhos até chegar ao pai.

Desta forma a pesquisa do pai fica muito mais rápida, abaixo na ilustração 6 é exemplificada a ligação de 5 filhos directamente ao pai “f”.

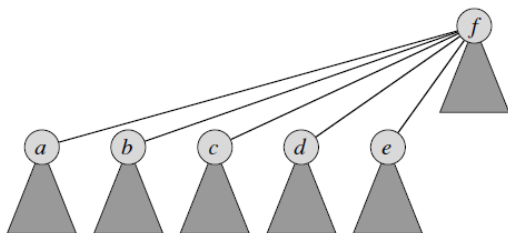


Ilustração 6 - Disjoint Set Forest com Path Compression

UNION BY RANK

Este melhoramento consiste em associar a árvore mais pequena à raiz da árvore maior, em vez do inverso. Tendo em conta que é a profundidade das árvores que afecta o tempo de processamento, a árvore com a menor profundidade é adicionada à raiz da árvore com maior profundidade, desta forma apenas aumenta a profundidade se as árvores forem iguais.

Na ilustração é demonstrada uma união por ranking.

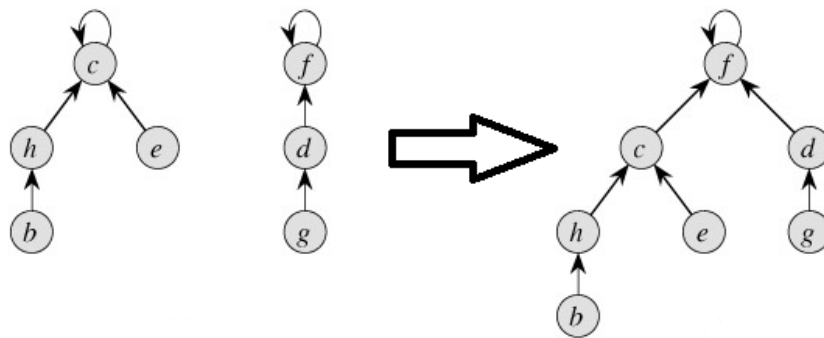


Ilustração 7 - Efeito da união por ranking

PATH COMPRESSION AND UNION BY RANK

Para obter uma optimização ainda maior é possível conjugar os dois métodos acima mencionados, union by rank e path compression.

Desta forma é possível obter uma optimização de tempo de processamento na ordem de $O(m \alpha(n))$ para m operações do tipo Disjoint-Set.

De acordo com a descrição que Cormen faz em Introduction to Algorithms, 2ed, a função $\alpha(n)$ é igual a 4 para valores de n entre 2048 e um número muito grande, logo cresce muito devagar.

Tendo em conta que $\alpha(n)$ cresce muito devagar, função $O(m \alpha(n))$ tem um crescimento linear directamente proporcional ao número de operações Disjoint-Set.

Sendo esta aplicação muito eficiente no tratamento das Disjoint-Set Structures

$$\alpha(n) = \begin{cases} 0 & \text{for } 0 \leq n \leq 2, \\ 1 & \text{for } n = 3, \\ 2 & \text{for } 4 \leq n \leq 7, \\ 3 & \text{for } 8 \leq n \leq 2047, \\ 4 & \text{for } 2048 \leq n \leq A_4(1). \end{cases}$$

Ilustração 8 - Descrição dos valores possíveis de $\alpha(n)$

Abaixo na ilustração é descrito o pseudocódigo da junção dos dois métodos.

```
MAKE-SET( $x$ )
1   $p[x] \leftarrow x$ 
2   $rank[x] \leftarrow 0$ 

UNION( $x, y$ )
1  LINK(FIND-SET( $x$ ), FIND-SET( $y$ ))

LINK( $x, y$ )
1  if  $rank[x] > rank[y]$ 
2    then  $p[y] \leftarrow x$ 
3  else  $p[x] \leftarrow y$ 
4      if  $rank[x] = rank[y]$ 
5          then  $rank[y] \leftarrow rank[y] + 1$ 

The FIND-SET procedure with path compression is quite simple.

FIND-SET( $x$ )
1  if  $x \neq p[x]$ 
2    then  $p[x] \leftarrow \text{FIND-SET}(p[x])$ 
3  return  $p[x]$ 
```

Ilustração 9 - Pseudocódigo para Disjoint-set forest

RESULTADOS OBTIDOS

Para cada experiencia foram utilizadas imagens com diferentes padrões e tamanhos, sendo que foram realizados 100 testes sobre as mesmas, por forma a garantir resultados credíveis, tendo em conta que os vários testes podem ter variações de desempenho na máquina, causadas por alterações nas solicitações efectuadas ao processador por outros programas ao mesmo tempo que os testes realizados estão a decorrer.

RECURSOS

Os testes foram executados recorrendo aos seguintes recursos:

Computador

- Processador: Intel® Pentium® Dual CPU T3400 @ 2.16GHz 2.17GHz
- Memória RAM 3GB
- Sistema Operativo: Windows 7 Ultimate 32 Bit
- Disco Rígido: Western Digital Scorpio Blue WD3200BEVT 320GB 5400 RPM 8MB Cache 2.5" SATA 3.0Gb/s

Software

- Microsoft Office Professional Plus 2010 32 Bit
- Python's Integrated DeveLopment Environment(IDLE)
- Python version 2.7.1

IMAGENS

As imagens utilizadas foram escolhidas de forma crescente para se ter noção das tendências dos algoritmos à medida que a complexidade das imagens aumentava.

Os testes foram corridos sobre as seguintes imagens:

Nome	Tamanho LxA(em pixéis)	Pixéis não background
01_255x255.jpg	255x255	12230
02_272x272.jpg	272x272	13620
03_300x300.jpg	300x300	17820
04_381x378.jpg	381x381	29178
05_500x500.jpg	500x500	121634
06_630x475.jpg	630x475	89814

Tabela 1 - Imagens utilizadas no estudo

RESULTADOS

Abaixo serão apresentados os resultados dos 100 testes executados a cada uma das 6 imagens utilizando os dois algoritmos.

Tabela 2 - Comparação de médias obtidas por método aplicado a cada imagem

Imagens	01_255x255.jpg	02_272x272.jpg	03_300x300.jpg	04_381x378.jpg	05_500x500.jpg	06_630x475.jpg
Disjoin-Set Forest	0,7643	0,8575	0,9809	2,0819	6,1226	5,1555
União com todos	0,4870	0,5102	0,4815	10,9176	3,4699	11,9119

Tabela 3 - Comparação de média e desvio padrão entre métodos

	Média	Desvio Padrão
Disjoin-Set Forest	2,6605	2,1685
União com todos	4,6297	4,9203

Tabela 4 - Relação entre médias obtidas e número de pixels preenchidos

Pixels preenchidos	12230	13620	17820	29178	89814	121634
Disjoin-Set Forest	0,76433	0,85755	0,98090	2,08185	5,15553	6,12263
União com todos	0,48697	0,51015	0,48149	10,91761	11,91190	3,46987

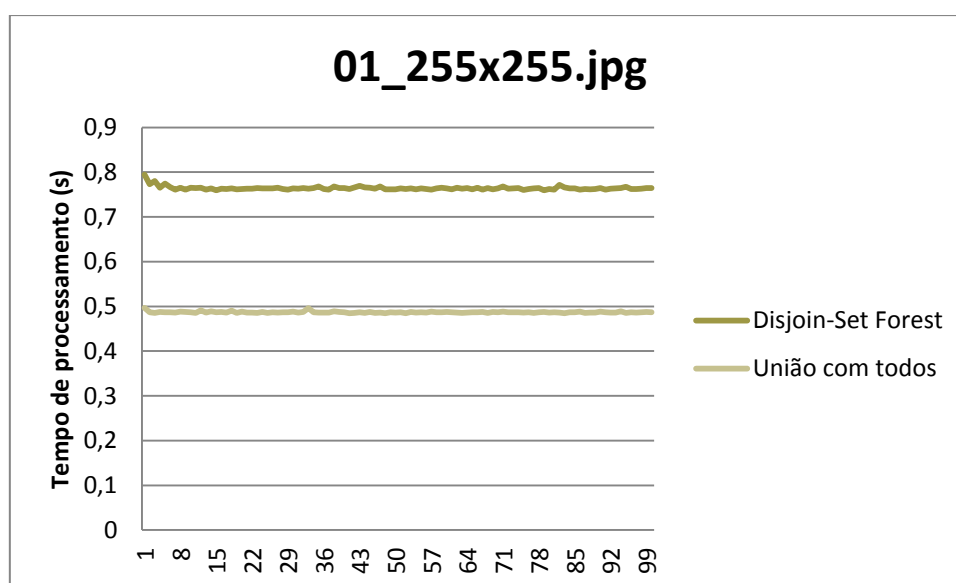


Gráfico 1 - Resultados dos testes realizados à imagem 01_255x255.jpg

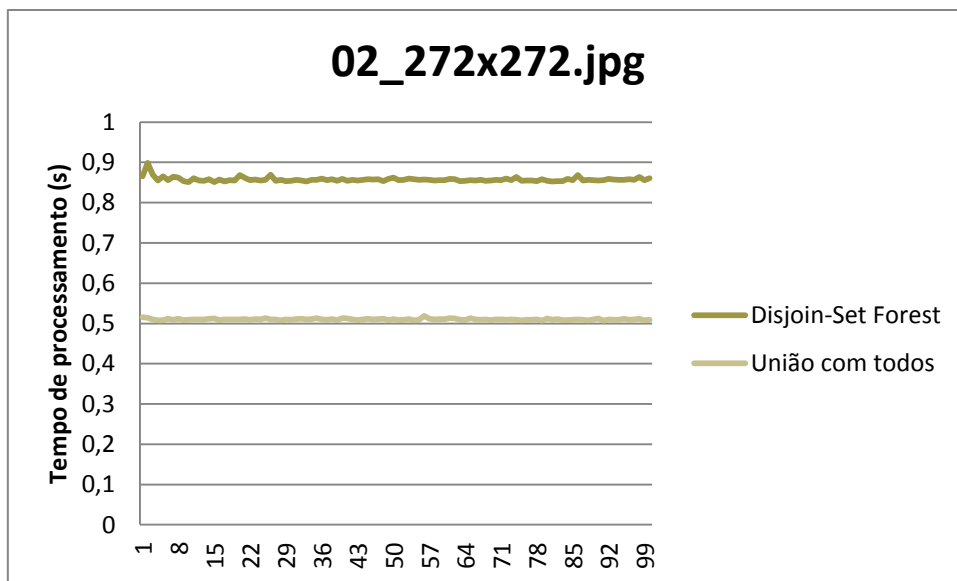


Gráfico 2 - Resultados dos testes realizados à imagem 02_272x272.jpg

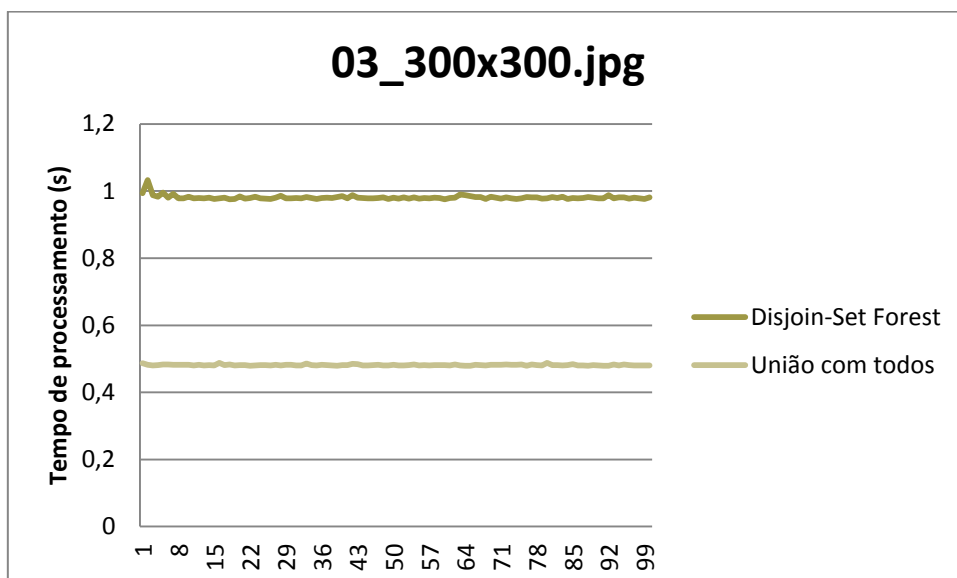


Gráfico 3 - Resultados dos testes realizados à imagem 03_300x300.jpg

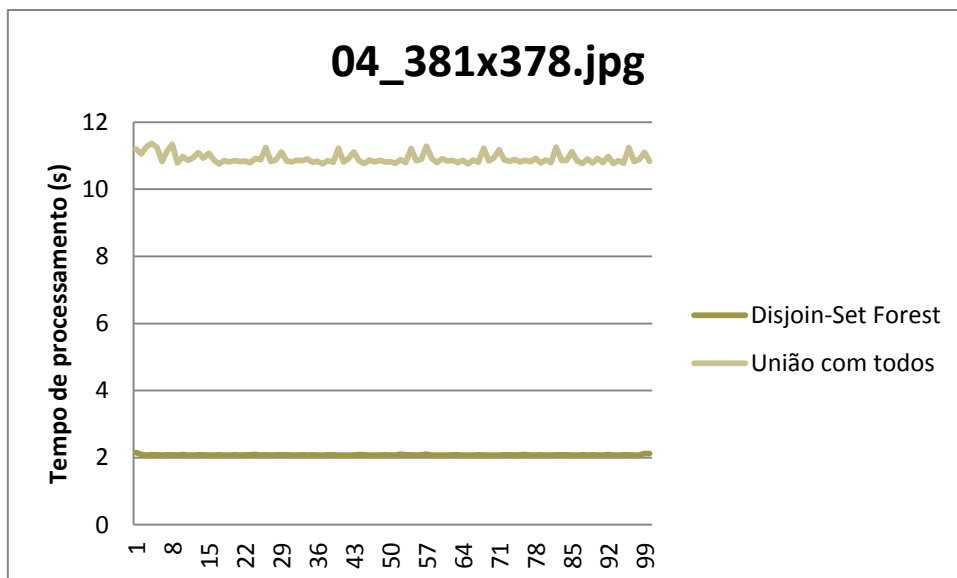


Gráfico 4 - Resultados dos testes realizados à imagem 04_381x378.jpg

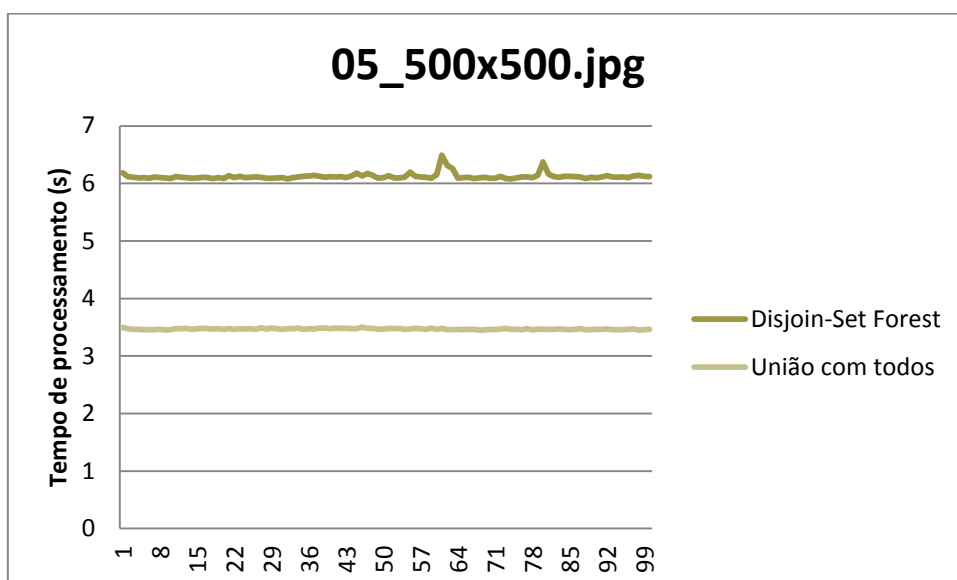


Gráfico 5 - Resultados dos testes realizados à imagem 05_500x500.jpg

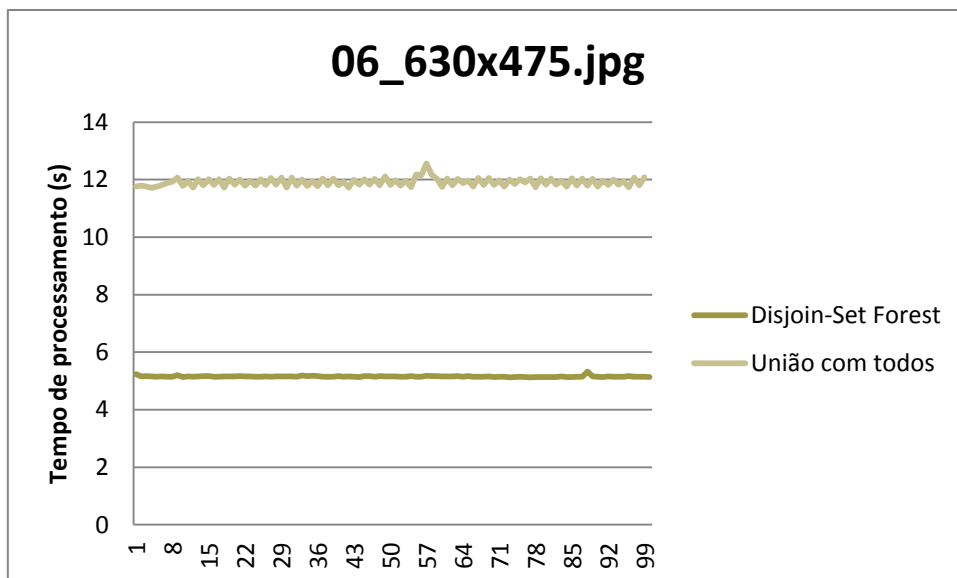


Gráfico 6 - Resultados dos testes realizados à imagem 06_630x475.jpg

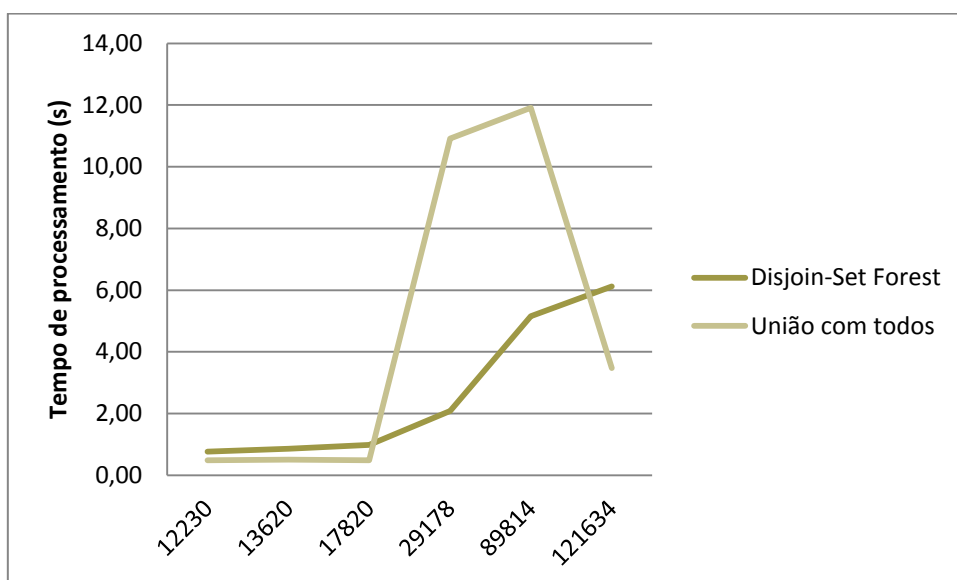
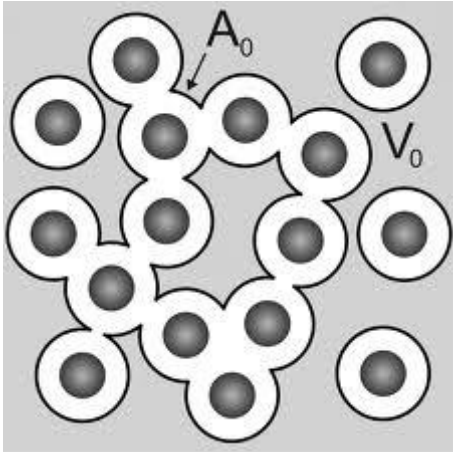
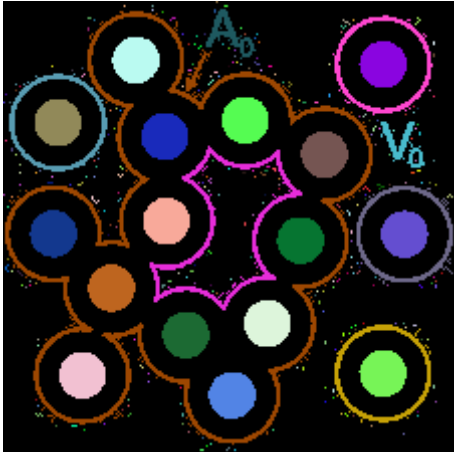

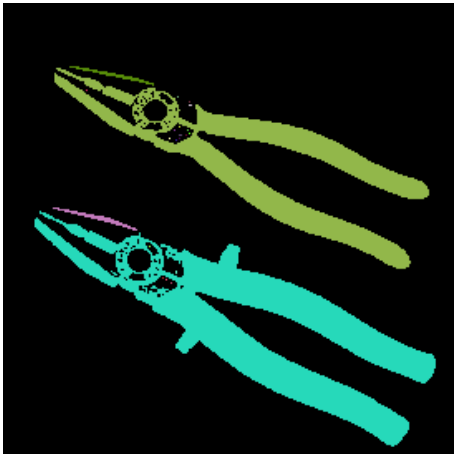


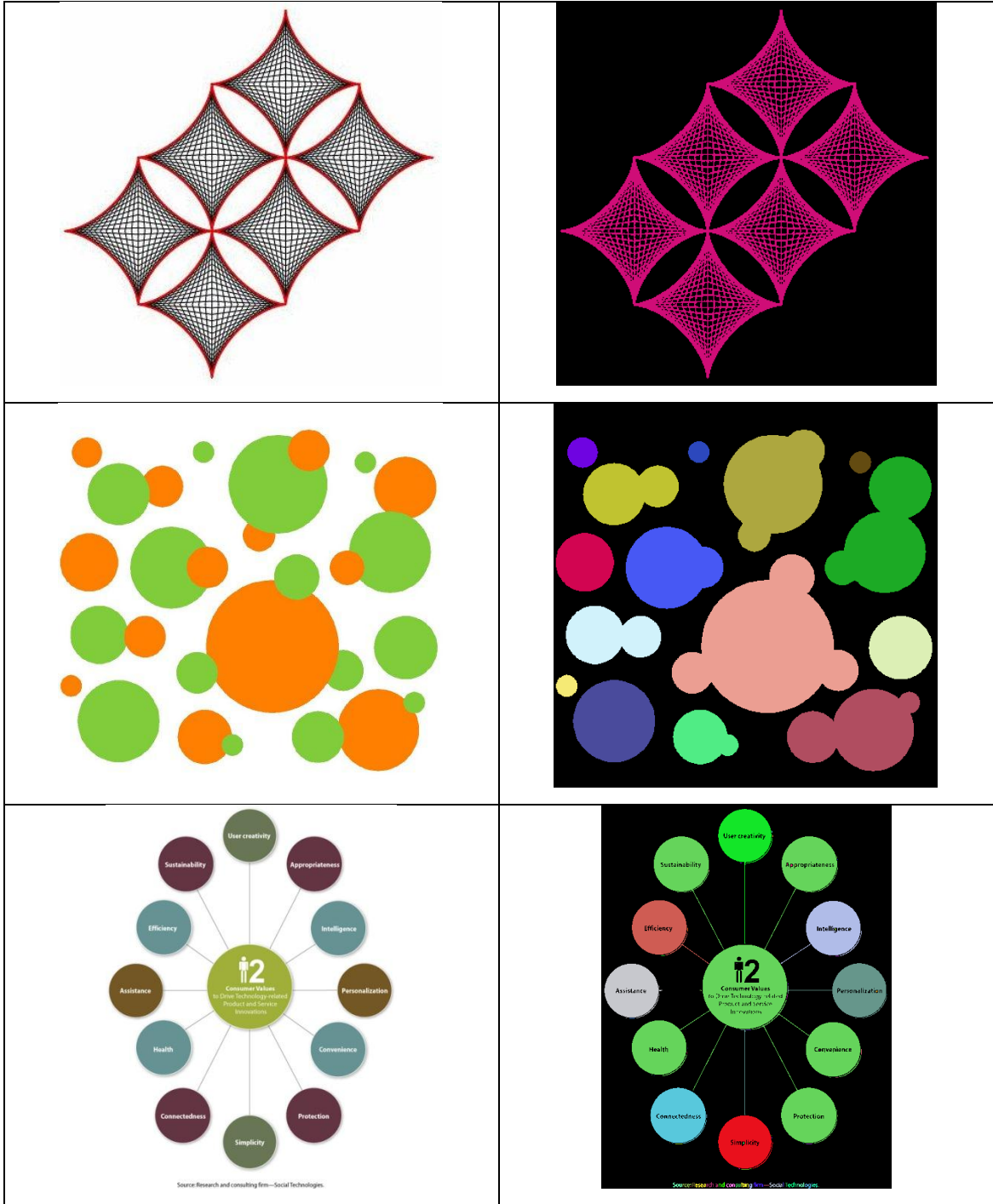


Gráfico 7 - Relação entre o tempo de processamento e o nº de pixéis preenchidos

IMAGENS OBTIDAS

Abaixo são apresentadas as imagens obtidas antes e depois da identificação das zonas conexas.

Antes	Depois
	
	
	



CONCLUSÕES

Foi cumprido o objectivo de criar um programa que permitisse identificar as zonas conexas de uma imagem.

Este trabalho foi muito interessante na medida em que me permitiu estudar as diferenças entre escolher entre um algoritmo ou outro mais eficiente, sendo o resultado da escolha muito importante no desfecho final do desenvolvimento de uma qualquer aplicação.

Foi um pouco complicado entender como implementar os algoritmos por forma a obter o resultado pretendido, pois no início tudo parecia muito abstracto e o manual sugerido para a aula não tem uma curva de aprendizagem muito rápida.

BIBLIOGRAFIA

1. Connected Components Labeling, Wikipédia,
http://en.wikipedia.org/wiki/Connected_component_labeling
2. Cormen, Thomas H. et al. Introduction to Algorithms. USA: McGraw-Hill, 2001