UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação

Disciplina	Turmas
Algoritmos e Estruturas de Dados II	3 e 4

Professores	Monitores
Erickson Rangel do Nascimento William Robson Schwartz	Bernardo Biesseck Gabriel Resende Gonçalves

Data de entrega: 30/10/2015 **até as** 23:55 **via** ***Moodle***

Trabalho Prático em Linguagem C (TP1)

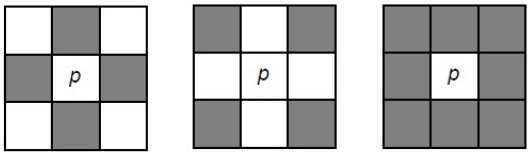
Especificação

O objetivo deste trabalho é implementar e aplicar o algoritmo **flood fill** recursivo e iterativo em imagens digitais no formato PGM. O flood fill realiza a coloração de uma região da imagem, com uma cor qualquer, circundada por pixels conectados de acordo com algum critério de vizinhança. Normalmente programas de desenho aplicam o flood fill como a ferramenta *balde de tinta*.

Vizinhança de pixels

- **Vizinhança 4:** dado um pixel p com coordenadas (x,y) seus vizinhos são os pixels adjacentes na horizontal e na vertical;
- **Vizinhança diagonal:** dado um pixel p com coordenadas (x,y) seus vizinhos são os pixels adjacentes nas diagonais;
- **Vizinhança 8:** dado um pixel p com coordenadas (x,y) seus vizinhos são os pixels adjacentes na horizontal, na vertical e nas diagonais;

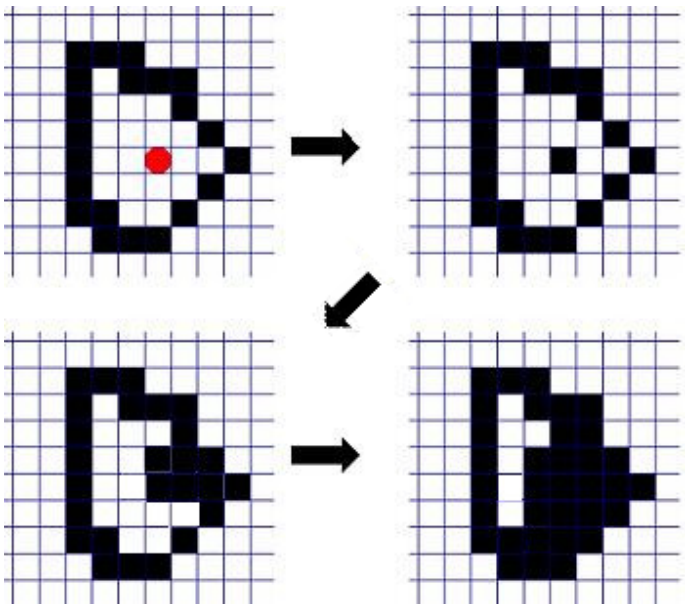
A figura abaixo exemplifica os pixels vizinhos de p pintados de cinza.



Vizinhança 4 *Vizinhança diagonal* *Vizinhança 8*

Algoritmo

Para realizar o flood fill são necessários uma imagem e um pixel inicial. Começando por ele o algoritmo deverá visitar seus vizinhos (de acordo com o critério especificado) pintando aqueles que possuem uma intensidade especificada com uma nova intensidade estabelecida. O algoritmo deve parar quando não houverem mais pixels para visitar ou quando não houver nenhum pixel vizinho com intensidade igual ou próxima da cor especificada inicialmente. A figura a seguir ilustra o flood fill em execução e o resultado final.



Para facilitar o entendimento este LINK contém um GIF animado extraído do Wikipedia que ilustra o seu funcionamento.

Abaixo o pseudo-código do algoritmo flood fill recursivo. Você poderá basear sua implementação neste exemplo.

Algorithm 1 Flood Fill

```
procedure FLOODFILL(img, x, y, corAtual, novaCor)
    if (pixel(x,y)-corAtual) < limiar then
        pixel(x,y) ← novaCor
        FLOODFILL(img, x+1, y, corAtual, novaCor)
        FLOODFILL(img, x, y+1, corAtual, novaCor)
        FLOODFILL(img, x-1, y, corAtual, novaCor)
        FLOODFILL(img, x, y-1, corAtual, novaCor)
    end if
end procedure
```

Detalhes de implementação

A imagem deverá ser preenchida com um valor qualquer de intensidade. Nas imagens de teste, utilize 127. Utilize a mesma estrutura de dados PGM utilizada no TP-0 para armazenar todas as imagens do programa.

```
typedef struct {
   int c;                   // número de colunas na imagem
   int l;                   // número de linhas na imagem
   unsigned char maximo;     // valor máximo para cada pixel
   unsigned char **imagem;    // variável para armazenar os pixels da imagem (matriz)
} PGM;
```

O que deve ser feito

1. Implementar o algoritmo flood fill recursivo e aplicá-lo sobre uma imagem a partir de um pixel inicial p. A imagem resultante deve ser salva em outro arquivo pgm.

2. Implementar o flood fill iterativo utilizando a estrutura Pilha implementada por você para simular a recursão do algoritmo ao visitar cada pixel e seus vizinhos. Ou seja, a ordem de visita e coloração dos pixels deve ser a mesma do algoritmo recursivo. Depois aplique-o sobre uma imagem a partir de um pixel inicial p e a imagem resultante deve ser salva em outro arquivo pgm.

3. Os nomes dos arquivos de entrada, o pixel inicial, e saída deverão ser passados por argumentos para o programa.

   ```
   ./exec entrada.pgm 0 0 saida.pgm
   ```

   onde *exec* é o nome do executável, *entrada.pgm* representa o nome do arquivo de entrada (imagem no formato PGM), 0 e 0 indicam as coordenadas pixel inicial do algoritmo e *saida.pgm* é o nome do arquivo de saída (formato PGM).

4. OBS: Utilizar um vetor de tamanho fixo para armazenar os dados da pilha causa desperdício de memória quando poucos dados são colocados na pilha. *Logo, implemente a pilha usando alocação encadeada.*

5. Semelhante ao TP-0, o programa deve ter a função `PGM *LerPGM(char* entrada)` que recebe o nome do arquivo como entrada, realiza a leitura da imagem e retorna um ponteiro para uma estrutura de dados PGM alocada dinamicamente.

6. O programa deve ter a função `void SalvarPGM(PGM *img, char* saida)` que escreve a imagem apontada por **img* em um arquivo no formato PGM com o nome descrito na variável *saida*.

7. O programa deverá ter a função `void FloodFill(PGM *entrada, int x, int y, unsigned char corAtual, unsigned char novaCor, PGM *saida)` que realiza o flood fill na *entrada*, começando pelo pixel (x, y) e armazena o resultado em *saida*.

8. Todas as funções implementadas devem possuir um cabeçalho conforme o exemplo a seguir:

```
/*-----------------------------------------------------------------------
Protótipo: PGM *LerPGM(char* entrada)
Função: Realiza a operação ...
Entrada: estruturas PGM contendo ...
Saída: ...
-----------------------------------------------------------------------*/
```

9. Seu programa não deverá ter nenhum *leak* de memória, ou seja, tudo que for alocado de forma dinâmica, deverá ser desalocado com a função "free()"antes do término da execução.

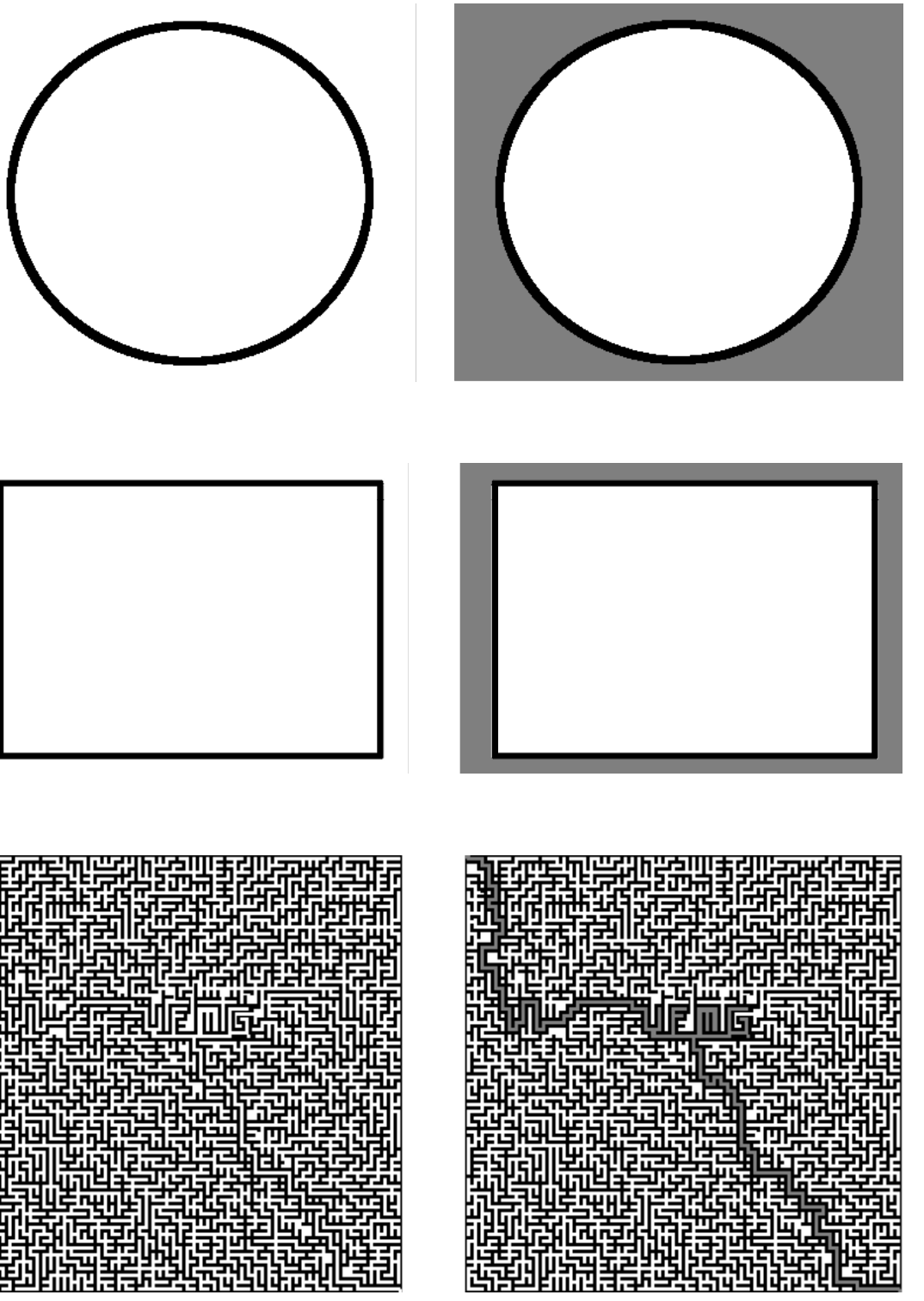
10. O programa deverá ser possível de ser compilado no Linux, portanto ele não deverá conter nenhuma biblioteca que seja específica do sistema operacional Windows.

11. Você deverá implementar o trabalho na IDE Code::Blocks (*disponível em: http://www.codeblocks.org/*). Deve ser submetido ao moodle o diretório do projeto criado no Code::Blocks em um arquivo compactado contendo os arquivos ".h"(com as declarações das funções para ler e salvar imagens PGM) e ".c"(um com as implementações das funções descritas nesse documento e outro com a função *main*. Além disso, a submissão também deverá conter o relatório do trabalho em formato PDF.

12. O trabalho deverá ser entregue via *Moodle* até as 23:55 horas do dia 30/10/2015. **Trabalhos que forem entregues fora do prazo não serão aceitos em nenhuma hipótese.**

Exemplos para teste

Para avaliar o funcionamento do seu programa disponibilizamos as imagens *quadrado.pgm*, *circulo.pgm* e *maze.pgm*. As imagens preenchidas, iniciadas do pixels (0, 0) e com preenchimento igual a 127, estão ilustrados abaixo.



Documentação

Escreva um documento explicando o seu código e avaliando o desempenho de sua implementação. Separe-o em cinco seções: introdução, implementação, resultados, conclusão e referências. Seja claro e objetivo.

- **Introdução**: Faça uma breve introdução o problema, definindo-o com as *suas* palavras.
- **Implementação**: Explique quais foram as estratégias adotadas para a leitura das imagens em formato PGM, execução do flood fill e a escrita da imagem PGM em disco. Descreva qual o papel de cada função, seus parâmetros de entrada e de saída. **Faça a análise da sua implementação em termos de tempo e espaço indicando as funções e as ordens de complexidade de cada um. Responda as perguntas: o seu algoritmo tem complexidade polinomial ou exponencial? Por que? Você consegue pensar em alguma forma de resolver o problema do flood fill com um custo mais baixo do que o algoritmo passado neste documento?**
- **Resultados**: Faça testes com seu programa utilizando várias imagens, **inclusive algumas além das passadas nos exemplos.** Apresente os resultados obtidos e faça uma breve discussão sobre eles. Serão disponibilizadas três imagens para teste.
- **Conclusão**: Explique quais foram as dificuldades encontradas durante o desenvolvimento e as conclusões obtidas.
- **Referências**: Se você utilizou informações adicionais além das especificadas neste documento, cite as fontes.

Avaliação

- **4 pontos** pela implementação, onde serão avaliados, além do funcionamento adequado do programa: identação correta do código, comentários das funções, alocações de desalocações dinãmicas bem feitas e modularização.
- **4 pontos** pelos testes, onde serão avaliados os resultados obtidos.
- **6 pontos** pela documentação, onde serão avaliados a clareza das seções e a discussão dos resultados obtidos.
- ***Lembramos que será utilizado um software de detecção de cópias e qualquer tipo de plágio detectado resultará em nota 0 para ambos trabalhos!***

Pontos Extras

Todos os alunos da UFMG são bons programadores, mas sabemos que alguns destes alunos se destacam entre os demais. Para que estes alunos não fiquem entediados após terminarem o TP, vamos distribuir 2 (dois) pontos extras em algumas tarefas. **Nota.** Você precisa ter feito todo o TP para poder receber os pontos extras!

(1 pontos): Implemente o algoritmo flood fill com vizinhança 8 utilizando a sua estrutura Pilha, ou seja, levando em consideração também os vizinhos da diagonal. Responda às perguntas: a complexidade é diferente do algoritmo com quatro vizinhos? Por que?

(1 pontos): Implemente o algoritmo flood fill utilizando a estrutura Fila no lugar da Pilha. Responda às perguntas: a complexidade do algoritmo é diferente? E os resultados nas imagens de saída?