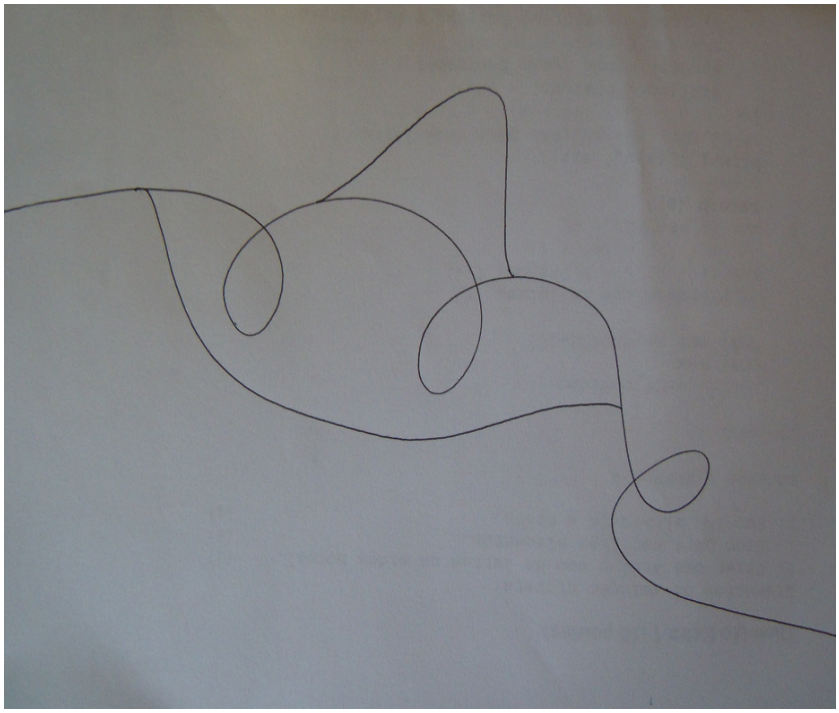
Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Informática
ICSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu
Profa. Leyza Baldo Dorini
Prof. Daniel Fernando Pigatto

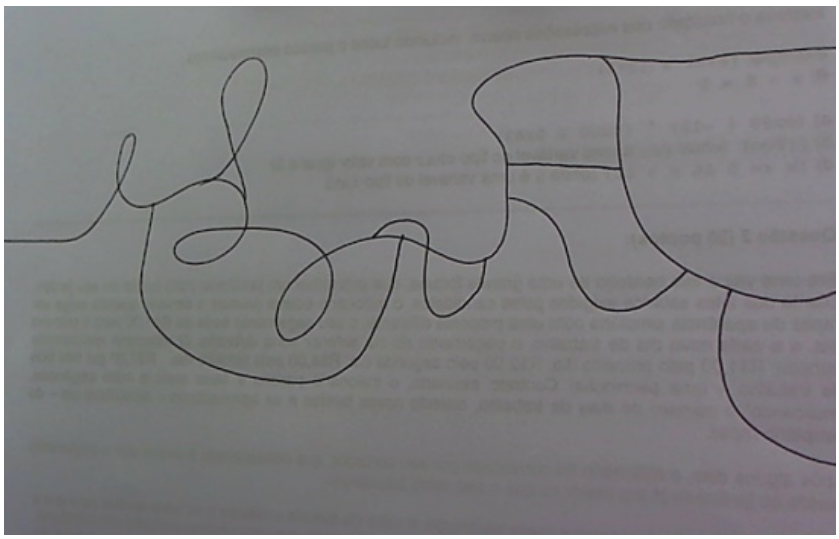
Trabalho 3

I) Descrição geral

Este trabalho tem como objetivo encontrar a saída de um “labirinto” desenhado em papel e fotografado. Apesar de ser um exercício abstrato, ele envolverá conceitos básicos de processamento digital de imagens e a busca por um caminho mínimo. Desta forma, as ideias por trás do trabalho estão presentes em ou se relacionam com softwares como Photoshop, sistemas para análise de imagens de satélite, e problemas como a definição do caminho a ser seguido por personagens de um game.

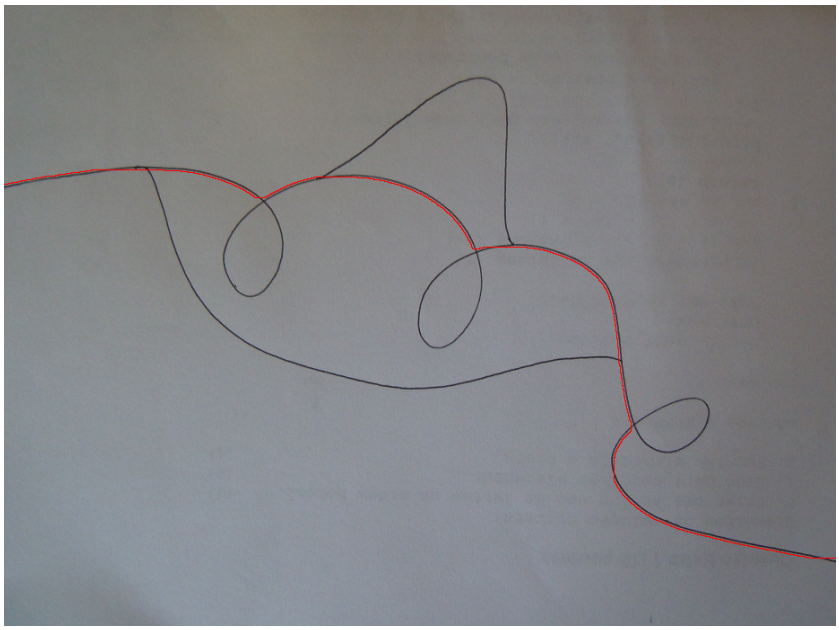
A entrada do sistema é um conjunto de imagens como as mostradas abaixo:

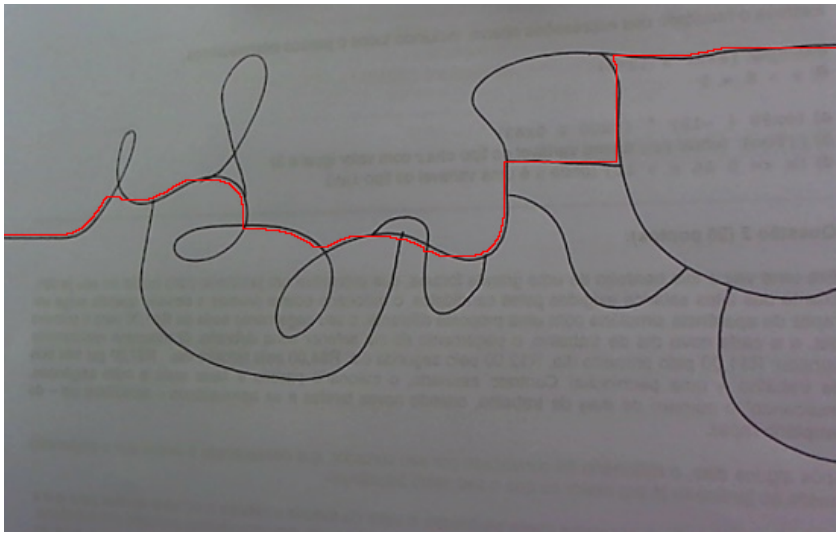




Cada imagem corresponde a um caso de teste, e foi obtida desenhando-se um “labirinto” sobre um papel branco, usando caneta preta. O desenho foi então fotografado e a imagem cortada, de forma que existe uma única “entrada” à esquerda e pelo menos uma “saída” à direita. Por se tratarem de fotografias, as imagens não contêm exatamente um caminho preto sobre uma superfície branca – as imagens são afetadas por sombras e pela iluminação, e em alguns casos o conteúdo no verso da folha pode ser visto.

O desafio do trabalho é determinar, para cada caso de teste, o menor caminho possível entre a entrada à esquerda e uma das saídas à direita, tendo como base apenas a própria imagem do caso de teste. O caminho deve passar sobre ou próximo às linhas escuras, como mostrado em vermelho nas figuras abaixo:





O seu trabalho é criar uma função com o protótipo abaixo:

```
int encontraCaminho (Imagem1C* img, Coordenada** caminho);
```

A função recebe como parâmetro de entrada uma imagem em escala de cinza, e retorna o número de passos do caminho entre a entrada e a saída mais próxima. A função também recebe um parâmetro de saída, que é um ponteiro para um vetor de coordenadas. O vetor deve ser alocado dentro da função, e deve ter exatamente o número de posições retornado. Um caminho válido começa com x=0, termina com x=largura-1, e tem a cada passo um vizinho do passo anterior. Uma coordenada é vizinha de outra se estiver imediatamente acima, abaixo, à direita ou à esquerda (mas não em diagonal!).

Para criar o trabalho, será fornecido um “pacote” contendo os seguintes arquivos:

- main.c: contém uma função `main` (i.e. um programa) que testa o trabalho. No começo do arquivo, existe uma macro `SALVA_SAIDA`: se o valor desta macro for diferente de 0, o programa salva imagens de saída destacando a trajetória achada, como mostrado nas figuras acima (note que o seu trabalho não é desenhar estes caminhos – a trajetória encontrada pela sua função é representada por um vetor de coordenadas). Também podem-se encontrar uma macro `N_ARQUIVOS`, que indica o número de casos de teste, e um vetor de strings chamado `ARQUIVOS`, que indica o arquivo usado em cada teste.

- imagem.c e imagem.h: contêm a declaração e a implementação de tipos e funções para manipulação de imagens no formato bmp.

- trabalho3.h: contém as declarações da função `encontraCaminho` e do tipo `Coordenada`. Este arquivo deve ser incluído (através da diretiva `#include`) no arquivo contendo a implementação da sua função.

- 11 casos de teste (diretório img).

A montagem do projeto, a forma de se representar e manipular imagens, e as estratégias que podem ser usadas para resolver o problema serão discutidos em aula.

II) Equipes

O trabalho deve ser feito em equipes de 3 pessoas. Equipes menores só serão aceitas se devidamente justificadas e previamente autorizadas pelos professores. A justificativa deverá ser um motivo de força maior: questões pessoais como “prefiro trabalhar sozinho” ou “não conheço ninguém da turma” não serão aceitas. Cada equipe deve apresentar sua própria solução para o problema. Indícios de fraude (cópia) podem levar à anulação dos trabalhos.

III) Entrega

O prazo de entrega é 25/06/2024. Trabalhos entregues após esta data terão sua nota final reduzida em 0,00025% para cada segundo de atraso. Cada equipe deve entregar, através da página da disciplina no Classroom, três arquivos (separados, sem compressão):

- Um arquivo chamado *t3-x-y-z.c*, onde *x, y* e *z* são os números de matrícula dos alunos. O arquivo deve conter a implementação da função `encontraCaminho` (com cabeçalho idêntico ao especificado), assim como de quaisquer outras funções auxiliares usadas no trabalho. Funções da biblioteca-padrão também podem ser usadas. Os autores do arquivo devem estar identificados no início, através de comentários. Separe as sub-tarefas em funções, para organizar o seu código.

IMPORTANTE: as funções pedidas não envolvem interação com usuários. Elas não devem imprimir mensagens (por exemplo, através da função `printf`), nem bloquear a execução enquanto esperam entradas (por exemplo, através da função `scanf`). O arquivo também não deve ter uma função `main`.

- Um arquivo no formato PDF chamado *t3-x-y-z.pdf*, onde *x, y* e *z* são os números de matrícula dos alunos. Este arquivo deve conter um relatório breve (em torno de 2 a 3 páginas), descrevendo em detalhes a contribuição de cada membro da equipe, os desafios encontrados, e a forma como eles foram superados. Não é preciso seguir uma formatação específica. Os autores do arquivo devem estar identificados no início.

- Uma imagem com nome *t3-x-y-z.bmp*, onde *x, y e z* são os números de matrícula dos alunos. Esta imagem deve conter um labirinto criado pela equipe. O algoritmo proposto pela equipe deve funcionar para esta imagem, e ela deve ser lida pelas funções fornecidas nos arquivos imagem.h e imagem.c – se isso não ocorrer, a equipe será penalizada. A largura da imagem deve estar entre 512 e 1024 pixels, e a altura entre 256 e 768 pixels.

IV) Avaliação (normal)

Todos os testes serão feitos usando a IDE Code::Blocks. Certifique-se de que o seu trabalho pode ser compilado e executado a partir dela.

Os trabalhos serão avaliados por meio de baterias de testes automatizados. Os testes estão implementados no arquivo main.c fornecido. A nota final será composta por uma nota base e descontos aplicados por conta de problemas encontrados. A nota base será decidida por uma "competição" envolvendo todos os trabalhos entregues, além de uma implementação feita pelo professor, sem otimizações ou "truques" sofisticados. As notas específicas serão atribuídas com base na colocação de cada trabalho na competição. Todos os trabalhos que obtiverem desempenho superior à implementação do professor terão a nota base superior a 100.

O desempenho será medido objetivamente através de uma métrica que beneficia trajetórias que se mantêm próximas a regiões escuras da imagem (i.e. próximas ao caminho desenhado). Devido às ambiguidades causadas pelo uso de fotografias (como determinar se um pixel pode ou não ser usado?), a melhor trajetória por vezes pode ser uma que "corta caminho" através de uma região clara da imgem.

O programa de testes fornecido realiza os cálculos das pontuações, e coloca os resultados em um arquivo out.txt, com cada linha correspondendo à pontuação obtida para um caso de teste. Quanto mais baixa a pontuação, melhor é o desempenho da função. Os valores absolutos não fazem sentido isoladamente, pois serão posteriormente normalizados de acordo com o desempenho dos programas entregues. Em caso de empate, o critério de desempate será o tempo de execução. Note que a métrica usada no teste pode sofrer alterações, portanto procure avaliar suas soluções visualmente, observando se o caminho descoberto se mantém sempre próximo das linhas desenhadas.

Os descontos sobre a nota base serão aplicados com base nos seguintes critérios:

IV.a) Compilação e corretude. Cada erro que impeça a compilação do arquivo ou que cause um erro de execução será corrigido pelo professor. Erros cuja correção seja complexa demais serão tratados como fracassos para os casos de teste afetados. Um trabalho que exija *n* correções terá sua nota base multiplicada por 0.9^n. Certifique-se que seu código compila!!!

IV.b) Atendimento da especificação. Serão descontados até 10 pontos para cada item que não esteja de acordo com esta especificação, tais como nomes de arquivos e funções fora do padrão.

IV.c) Documentação. Descreva através de comentários o funcionamento das suas funções. Não é preciso detalhar tudo linha por linha, mas forneça uma descrição geral da sua abordagem, assim como comentários sobre estratégias que não fiquem claras na leitura das linhas individuais do programa. Uma função sem comentários terá sua nota reduzida em até 25%.

IV.d) Organização e legibilidade. Use a indentação para tornar seu código legível, e mantenha a estrutura do programa clara. Evite construções como *loops* infinitos terminados somente por `break`, ou *loops* `for` com varías inicializações e incrementos, mas sem corpo. Evite também replicar blocos de programa que poderiam ser melhor descritos por repetições. Um trabalho desorganizado ou cuja legibilidade esteja comprometida terá sua nota reduzida em até 30%.

IV.e) Variáveis com nomes significativos. Use nomes significativos para as variáveis – lembre-se que `n` ou `x` podem ser nomes aceitáveis para um parâmetro de entrada que é um número, mas uma variável representando uma "soma total" será muito melhor identificada como `soma_total`, `soma` ou `total`; e não como `t`, `aux2` ou `foo`. Uma função cujas variáveis internas não tenham nomes significativos terá sua nota reduzida em até 15%.

IV.f) Siga as convenções para nomear constantes, funções e variáveis. Um trabalho que tenha identificadores fora das convenções pode ter sua nota reduzida em até 10%.

IV.g) Crie funções para organizar o seu programa, tendo em mente a modularização e a legibilidade. Um trabalho monolítico ou tendo funções muito longas/confusas terá sua nota reduzida em até 25%.

IV.h) Desempenho. Se a estrutura lógica de uma função for pouco eficiente, por exemplo, realizando muitas operações desnecessárias ou redundantes, a sua nota pode ser reduzida em até 20%.