BCC202 – Estruturas de Dados I (2024-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)

Trabalho Prático I (TP I) - 10 pontos, peso 1.

- Submissão com data e hora de entrega disponíveis na plataforma da disciplina. O que vale é o horário do *Moodle*, e não do *seu*, ou do *meu* relógio!!!

- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.

- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.

- A avaliação considerará o tempo de execução e o percentual de respostas corretas.

- Eventualmente serão realizadas entrevistas sobre o trabalho para complementar a avaliação;

- O trabalho é em grupo de até 2 (duas) pessoas.

- Será aceito trabalhos após a data de entrega, todavia com um decréscimo de 0,05 a cada 10min.

- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.

- Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.

- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos

- Não serão considerados algoritmos parcialmente implementados.

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Você deve submeter os arquivos *.h*, *.c* e o *.pdf* (relatório) na raiz do arquivo *.zip*. Use os nomes dos arquivos *.h* e *.c* exatamente como pedido.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

- Bom trabalho!

O Mistério das Grades Secretas

Em uma cidade antiga, há muitos séculos, um grupo de artesãos dedicados guardava um segredo curioso e fascinante. Eles criavam belas obras de arte em mosaico, pequenas imagens compostas de pedras e azulejos coloridos, que retratavam cenas do cotidiano, figuras mitológicas e paisagens de tirar o fôlego. Contudo, essas imagens tinham uma peculiaridade: não podiam ser vistas de imediato. Elas estavam ocultas em padrões de pedra que, à primeira vista, pareciam um enigma indecifrável.

Reza a lenda que essas imagens secretas eram encomendadas por reis, mercadores e viajantes em busca de sabedoria, mas apenas aqueles que usassem a lógica e paciência poderiam revelá-las. Para desvendá-las, um observador precisava interpretar sequências de números que acompanhavam as linhas e colunas da grade de pedra, descobrindo assim a posição das cores que completavam o quadro. Cada sequência representava a quantidade de pedras de uma mesma cor que aparecia em cada linha ou coluna, mas sem revelar exatamente onde deveriam ser colocadas.

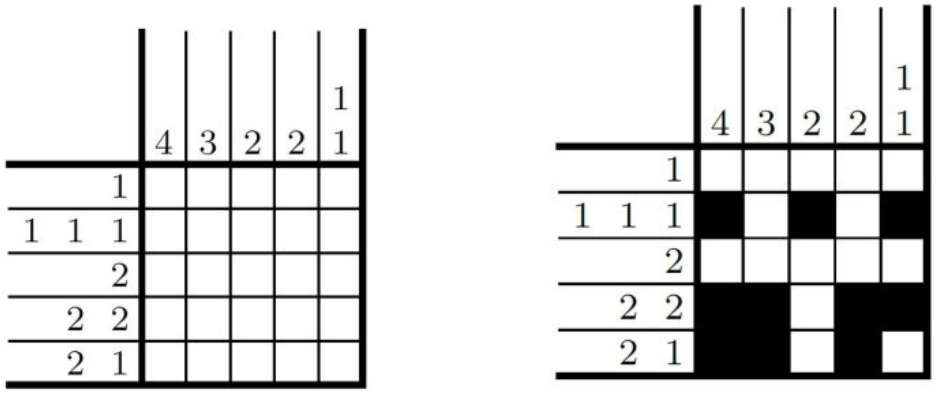
Conforme os séculos passaram, esses mosaicos foram esquecidos pela maioria e se tornaram apenas uma lenda entre estudiosos e artistas. Mas um jovem arqueólogo, fascinado pelas histórias, acabou por encontrar, em meio a ruínas abandonadas, uma dessas grades misteriosas. Intrigado, ele decidiu decifrar o código e revelar a imagem oculta, desvendando o segredo dos antigos artesãos.

Ele observou a grade com cuidado. Ao lado de cada linha e coluna, viu números esculpidos, os quais indicavam quantas pedras coloridas deveriam aparecer. Ele percebeu que, em cada linha ou coluna, os grupos de pedras coloridas estariam separados por intervalos em branco, e então começou a experimentar, pintando e deixando espaços vazios, tentando desvendar a imagem oculta.

Com paciência, ele foi preenchendo a grade até que uma figura começou a emergir — uma cena detalhada da vida na cidade antiga. E assim, aquele mosaico perdido renasceu, revelando ao jovem arqueólogo os segredos daquela civilização esquecida.

E foi assim que o enigma dos Nonogramas surgiu, inspirado pelos mistérios antigos de grades que ocultavam imagens e que, através da lógica, eram trazidas à luz. Você pode testar e jogar o Nonograma no seguinte link: `https://br.puzzle-nonograms.com/`.

De forma prática, no Nonograma, células de uma grade devem ser preenchidas de acordo com dicas representadas por números dispostos nas bordas das linhas e colunas. Esses números indicam a quantidade de células consecutivas que devem ser preenchidas em cada linha ou coluna. Quando mais de um número é apresentado para uma linha ou coluna, os diferentes grupos de células preenchidas devem ser separados por pelo menos uma célula em branco. A Figura 1(a) apresenta um exemplo do jogo inicial e a Figura 1(b) a solução do mesmo.



(a) Nonograma inicial (b) Nonograma resolvido

Figura 1: Exemplo do jogo Nonograma.

Agora, é a sua vez. Você deve desenvolver um programa que implemente o jogo de quebra-cabeça, permitindo que o usuário descubra uma imagem oculta em uma grade através da aplicação de lógica. O programa deverá interpretar as dicas numéricas fornecidas e validar a solução apresentada pelo jogador.

Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada).
- Um grande número de *Warnings* ocasionará a redução na nota final.

O que deve ser entregue

- Código fonte do programa em C (**bem identado e comentado**).
- Relatório do trabalho (relatório[1]). A documentação deve conter:
  1. **Introdução:** descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. **Implementação:** descrição sobre a implementação do programa. **Não faça** "*print screens*" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
  3. **Testes:** descrição dos testes realizados e listagem da saída (não edite os resultados).
  4. **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho. Por exemplo, avaliar o tempo gasto de acordo com o tamanho do problema.
  5. **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  6. **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.
  7. **Formato:** PDF ou HTML.

Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *Moodle*, um arquivo **.ZIP** com o nome e sobrenome do aluno. Esse arquivo deve conter: (i) os arquivos *.c* e *.h* utilizados na implementação, (ii) instruções de como compilar e executar o programa no terminal, e (iii) o relatório em **PDF**.

Detalhes da implementação

Para atingir o seu objetivo, você deverá construir um Tipo Abstrato de Dados (TAD) `Nonogram` como representação do tabuleiro do campo minado que você irá implementar. O seu TAD deve possuir no mínimo os seguintes campos: dimensão, o tabuleiro em si e as dicas (*hints*) fornecidas via terminal. O TAD deverá implementar, **pelo menos**, as seguintes operações:

- `NonogramAllocate`: que aloca o TAD `Nonogram` e as células do tabuleiro que pode ser a matriz.
- `NonogramFree`: que libera tudo que foi alocado para o TAD `Nonogram`.
- `NonogramRead`: lê o tabuleiro a partir dos dados do terminal.
- `NonogramPlay`: resolve o tabuleiro informado por terminal e retorna a quantidade de soluções encontradas.
- `NonogramPrint`: imprime o tabuleiro.

O tabuleiro pode ser implementado de acordo com o desejo do aluno, todavia, o TAD deve ser implementado utilizando a separação interface no `.h` e implementação `.c` bem como as convenções de tradução.

[1]Exemplo de relatório: `https://www.overleaf.com/latex/templates/modelo-relatorio/vprmcsdgmcgd`.

Considerações

O código-fonte deve ser modularizado corretamente em três arquivos: `main.c`, `nonogram.h` e `nonogram.c`. O arquivo `main.c` deve apenas invocar e tratar as respostas das funções e procedimentos definidos no arquivo `nonogram.h`. A separação das operações em funções e procedimentos está a cargo do aluno, porém, não deve haver acúmulo de operações dentro de uma mesma função/procedimento.

O limite de tempo para solução de cada caso de teste é de apenas **um segundo**. Além disso, o seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada). ***Warnings* ocasionarão a redução pela metade da nota final**. Assim sendo, utilize suas habilidades de programação e de análise de algoritmos para desenvolver um algoritmo correto e rápido!

Entrada e Saída

A entrada é dada por meio do terminal. Para facilitar, a entrada será fornecida por meio de arquivos. **Use eles como entrada no terminal com o seguinte comando:** `./executavel < arquivo_teste.in`.

A primeira linha especifica o tamanho n do tabuleiro. Após isso, serão apresentadas n linhas informando a combinação de números de cada coluna, onde o primeiro número corresponde a quantidade de grupos naquela coluna. Após as n colunas, são informadas as informações das n linhas seguindo o mesmo padrão.

A saída consiste nas várias soluções possíveis para um tabuleiro e a quantidade de soluções encontradas. Será usado o caractere '.' para representar onde não estiver marcado e 'x', onde estiver. Se o tabuleiro não tiver solução, deve ser impressa a mensagem "*No solution was found!*".

Exemplo de caso de teste

Exemplos de saídas esperadas dada uma entrada:

Entrada	Saída
3 0 1 1 0 1 2 0 0	No solution was found!

Entrada	Saída
5 0 1 4 2 1 1 1 1 0 0 1 2 2 1 1 1 2 1 1	SOLUTION 1: . . . . . . x x . . . x . x . . x x . . . x . . . Total of solutions: 1

Entrada	Saída
5 2 2 1 1 4 1 1 2 1 2 1 3 1 1 2 2 1 2 2 1 2 1 2 2 2 2	SOLUTION 1: * . . . . * * . * . . * * . * . * . * * * * . * * SOLUTION 2: . . * . . * * . * . * * . . * . * . * * * * . * * Total of solutions: 2

A SAÍDA DA SUA IMPLEMENTAÇÃO DEVE SEGUIR EXATAMENTE A SAÍDA PROPOSTA.

Diretivas de Compilação

As seguintes diretivas de compilação devem ser usadas (essas são as mesmas usadas no *Moodle*).

```
$ gcc -c nonogram.c -Wall
$ gcc -c main.c -Wall
$ gcc main.o nonogram.o -o exe -lm
```

Também é fornecido um arquivo `Makefile`.

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta *valgrind*. O *valgrind* é um *framework* de instrumentação para análise dinâmica de um código e é muito útil para resolver dois problemas em seus programas: **vazamento de memória e acesso a posições inválidas de memória** (o que pode levar a *segmentation fault*). Um exemplo de uso é:

```
1 gcc -g -o exe arquivo1.c arquivo2.c -Wall
2 valgrind --leak-check=full -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
1 ==xxxxxx== All heap blocks were freed -- no leaks are possible
2 ==xxxxxx==
3 ==xxxxxx== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0))
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.

O SEU CÓDIGO SERÁ TESTADO NOS COMPUTADORES DO LABORATÓRIO (AMBIENTE LINUX)