Departamento de Ciência da Computação – IME-USP

MAC0121 Algoritmos e Estruturas de Dados I
Segundo Semestre de 2019

EP1 - Dormedorme Turtle
Data máxima para entrega: 24/AGO/2019

Todo EP é estritamente individual.

"To infinity . . . and beyond!"

Buzz Lightyear

1 Objetivo

Pretendemos com este EP reforçar conceitos e boas práticas de programação vistos em MAC0110:

- a manipulação de arquivos, strings, vetores e matrizes;
- a criação e utilização de funções com essas estruturas; e
- o desenvolvimento e teste de funções e programas.

Mais adiante, nas aulas, veremos os problemas tratados neste EP do ponto de vista MAC0121.

Este EP é inspirado no jogo eletrônico *Lights out*, cuja descrição pode ser vista em

`http://en.wikipedia.org/wiki/Lights_Out_(game)`.

Este EP é muito legal e podemos aprender muitas coisas com ele.

2 Turtlenautas, turtledorms, tapinhas e guardiões

Os **turtlenautas**, após vagarem perdidos pelo espaço por um tempo, finalmente encontraram o planeta *Turtlon*, a origem de sua espécie. Nesse planeta, um aspecto curioso é que os **turtles** dormem em lugares especiais denominados *turtledorms*. Um **turtledorm** é um arranjo de `n` × `n` cubículos individuais em uma área quadrada, sendo que cada cubículo comporta apenas um turtle.

Um turtle num turtledorm ou está desperto ou está dormindo. Vamos supor também que todos os cubículos estão ocupados. A figura 1(a) mostra um turtledorm `5` × `5`.

Um pequeno **tapinha** é suficiente para fazer com que um turtle desperto caia imediatamente no sono e, similarmente, para fazer com que um turtle que esteja dormindo desperte imediatamente. Então um **guardião** de turtledorm pode achar que fazer com que todos os turtles durmam é a coisa mais fácil do mundo. Ledo engano!

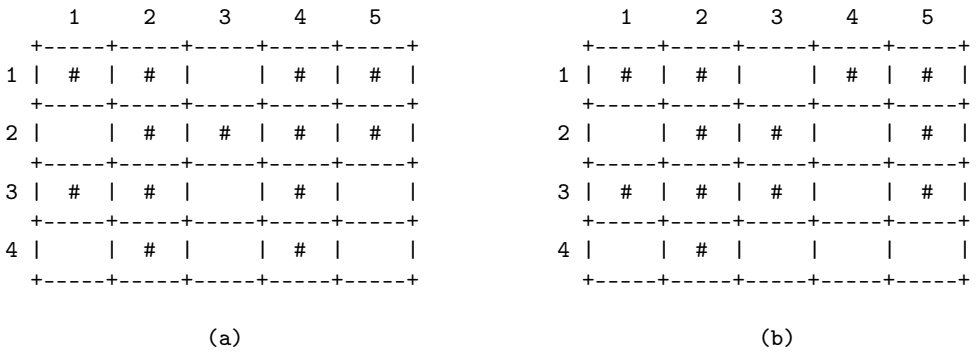

Figura 1: (a) Mostra um turtledorm de dimensão 4×5. Para fins de visualização, um caractere `'#'` indica um turtle desperto e a ausência indica um turtle dormindo. (b) Turtledorm após um tapinha no turtle da posição `[3][4]`).

Quando um turtle recebe um tapinha, ele acorda se estiver dormindo e dorme se estiver desperto. Além disso, ele se espreguiça e cutuca usando suas 4 patinhas o seu vizinho de cima, de baixo e dos lados. Uma cutucada acorda ou faz dormir o vizinho, mas só um tapinha faz ele se espreguiçar. Assim, quando o guardião coloca um turtle para dormir (ou despertar) com um tapinha, esse turtle só afeta o sono de seus 4 vizinhos. Aqueles que estavam despertos dormem, e os que estavam dormindo são despertados! A figura 1(b) mostra o que ocorre quando o turtle na posição `[3][4]` recebe um tapinha do guardião.

Uma **solução** de um turtledorm é uma sequência ou **lista de tapinhas** que coloca todos os turtles para dormir, ou seja, o guardião dando tapinhas nos turtles das posições na lista coloca todos os turtles para dormir. Chamaremos uma solução de um turtledorm de **ótima** ou **menos violenta** se, dentre todas as soluções para o turtledorm, ela é uma das que usa o menor número de tapinhas possível.

Todos os dias o guardião de um turtledorm recebe a missão de colocar todos os turtles para dormir. Neste EP, você fará um programa para simular a tarefa do guardião.

3 O que deve ser feito neste EP

Você deve escrever um programa em `C` que

(i) permita que o usuário jogue uma partida de *Dormedorme Turtle* no papel de guardião, e

(ii) encontre e exiba uma solução ótima do turtledorm para ajudar o guardião.

Inicialmente, o programa deve carregar um turtledorm e exibi-lo ao guardião. Em seguida, o programa deve realizar uma sequência de iterações. Em cada iteração, uma posição onde o guardião deseja **dar um tapinha** deve ser lida, esse tapinha deve ser aplicado e o turtledorm resultante deve ser mostrado. O jogo acaba quando o guardião consegue fazer com que todos os turtles durmam.

Além de dar tapinhas, o guardião também poderá **desistir da missão**, ou pedir para o programa **mostrar uma solução ótima** do turtledorm atual, caso exista uma solução, ou ainda o guardião poderá pedir para o programa **gravar o turtledorm** atual em um arquivo para, eventualmente, ser carregado em uma próxima oportunidade.

4 Representações de posições, turtledorms e listas de tapinhas

Uma **posição** em um turtledorm será representada por um par de inteiros que correspondem a sua linha e sua coluna. Dizemos que uma matriz é **binária** se cada uma de suas posições contém um 0 ou um 1.

A configuração de um turtledorm de dimensão `nLin` × `nCol` deverá ser **representada** por dois inteiros `nLin` e `nCol` e por uma matriz binária `tdorm` de dimensão `nLin` × `nCol`. Na matriz, uma posição com um 1 corresponde a uma posição do turtledorm em que o turtle está desperto, já uma posição com um 0 corresponde a um turtle dormindo.

Uma **lista de tapinhas** poderá ser representada por uma matriz binária `tapinhas` de dimensão `nLin` × `nCol`. As posições na matriz com um 1 correspondem àquelas na lista de tapinhas.

5 Comportamento do programa

Para carregar um turtledorm, o programa deve oferecer ao usuário duas opções: **sortear** uma configuração ou **ler** uma configuração de um arquivo. Assim, a primeira coisa que o seu programa deve fazer é perguntar:

```
(s)ortear ou (l)er um turtledorm de arquivo: l
```

Você deve supor que a resposta do usuário a esta pergunta é sempre um caractere `l` ou `s`.

Caso a resposta seja `l` para "`l`er um turtledorm de arquivo", o programa deve então perguntar pelo nome do arquivo e carregá-lo. Um arquivo de entrada deve conter os valores de `nLin` e `nCol`, e uma matriz binária de dimensão `nLin` × `nCol`.

Caso a resposta seja `s`, o programa deve perguntar em seguida e exatamente nessa ordem:

- qual a dimensão do turtledorm? (dois números inteiros)
- qual a semente para sorteio? (outro número inteiro)
- qual o nível de dificuldade [f/m/d]? (um caractere '`f`', '`m`', ou '`d`'para (`f`)ácil, (`m`)édio ou (`d`)ifícil).

Com esses 4 parâmetros (3 inteiros e um caractere), o seu programa deve então construir um turtledorm apropriado.

Uma vez lido ou sorteado, o turtledorm deve ser mostrado na tela, como na figura 1(a), para que o jogo se inicie. O guardião então poderá **dar tapinha** em uma posição, ou **desistir** ou **pedir ajuda** para ver uma solução ótima do turtledorm ou **gravar** o turtledorm atual em um arquivo. Assim, deve-se ler do teclado:

- uma posição [`lin`][`col`] (o usuário digita dois inteiros separados por um espaço); ou
- um caractere '`d`' (o usuário digita `d`) para (`d`)esistir da partida;
- um caractere '`a`' (o usuário digita `a`) para (`a`)juda, que mostra uma solução ótima da partida;
- um caractere '`g`' (o usuário digita `g`) para (`g`)ravar o turtledorm atual em um arquivo.

Caso a entrada seja uma posição [`lin`][`col`], o seu programa deverá executar um tapinha no turtle que ocupa o cubículo de coordenada [`lin`][`col`], alterar o estado dos turtles de acordo com a regra do jogo, e

mostrar a nova configuração do turtledorm na tela. Suponha `nLin` × `nCol` é a dimensão do turtledorm. A posição [`lin`][`col`] pode ter valores entre 1 e `nLin` e 1 e `nCol`, respectivamente. Ou seja, $1 \leq$ `lin` $\leq$ `nLin` e $1 \leq$ `col` $\leq$ `nCol`.

Se após o tapinha não restarem turtles despertos, o programa deve terminar após imprimir o número de jogadas efetuadas e avisar o usuário que ele ganhou a partida.

Caso a entrada seja '`d`', o programa deverá imprimir o número de jogadas efetuadas até aquele momento e terminar.

Caso a entrada seja '`a`', o programa deve calcular uma solução ótima a partir da situação atual do turtledorm, imprimir o número de tapinhas na solução calculada e exibir uma matriz de mesma dimensão indicando quais são as posições onde o guardião deve dar os tapinhas (veja a figura 2). O programa deve então continuar, pedindo ao usuário que entre com o próximo lance.

Caso a entrada seja '`g`', o programa deverá gravar o turtledorm em um arquivo e continuar, pedindo ao usuário que entre com o próximo lance.

Para facilitar o desenvolvimento, sugerimos que você resolva esse EP uma parte de cada vez, testando cada função antes de passar a desenvolver a próxima parte. Maiores detalhes das três partes propostas, mais os aspectos de organização do seu código, estão descritos a seguir e valem nota conforme especificado. Todas as funções descritas a seguir são **obrigatórias**.

6 Descrição da Parte I - Jogo básico (vale 60% da nota)

Na parte I você vai desenvolver o jogo básico, que corresponde à leitura do turtledorm de um arquivo e à implementação do comportamento geral do jogo. Nessa parte, não se preocupe com as opções de sorteio e ajuda para resolver o problema. O seu programa deve ser capaz de ler os lances do usuário e realizar o processamento correspondente. Para isso você deve implementar ao menos (ou seja, você pode implementar outras funções se desejar), as seguintes funções em `C`.

Considere as seguintes declarações:

```
/* tamanho máximo de um tdorm */
#define MAX      128

#define ACORDADO  '#'
#define DORMINDO  ' '
#define TAPINHA   'T'

#define TRUE  1
#define FALSE 0
```

1. `int main(int argc, char *argv[]);`
   Programa principal, responsável pelo comportamento geral do jogo, recebendo os lances e comandos do guardião. Ao final, o programa deve imprimir se o guardião ganhou ou desistiu e, em ambos os casos, quantos lances foram realizados.
2. `void leiaTurtledorm(int *nLin, int *nCol, int tDorm[][MAX]);`
   Lê de um arquivo a configuração de um turtledorm e devolve em `*nLin` e `*nCol` a sua dimensão e em

tDorm a sua representação. O arquivo de entrada deve conter no seu início o valor de *nLin e *nCol e em seguida os valores de uma matriz binária de dimensão *nLin × *nCol.

3. `void mostreTurtledorm(int nLin, int nCol, int tDorm[][MAX], char c);`
   Mostra na tela o turtledorm representado por nLin, nCol e pela matriz binária tDorm. O caractere c deve ser usado para exibir as posições com 1 (um), e um espaço para as posições com 0 (zero). Para exibir os turtles despertos, usamos c = ACORDADO e, para exibir os tapinhas de uma solução, usamos c = TAPINHA.

4. `void tapinhaTurtle(int nLin, int nCol, int tDorm[][MAX], int lin, int col);`
   Atualiza a configuração do turtledorm representado por nLin, nCol e pela matriz binária tDorm de forma a refletir a mudança de estados provocada por um tapinha no turtle no cubículo da posição [lin][col].

5. `int todosDormindo(int nLin, int nCol, int tDorm[][MAX]);`
   Verifica se todos os turtles no turtledorm representado por nLin, nCol e pela matriz binária tDorm estão dormindo. Retorna TRUE em caso afirmativo e FALSE em caso contrário.

6. `int graveTurtledorm(int nLin, int nCol, int tDorm[][MAX]);`
   Esta função pergunta ao guardião o nome de um arquivo e cria e grava nesse arquivo o turtledorm representado por nLin, nCol e pela matriz tDorm. A função retorna EXIT_FAILURE se houve algum problema e EXIT_SUCCESS se o turtledorm foi gravado.

7 Descrição da Parte II - Níveis de dificuldade (vale 15% da nota)

Na parte II você vai deixar o jogo um pouco mais interessante, permitindo o sorteio de uma configuração inicial com a escolha entre 3 níveis de dificuldade.

1. `void sorteieTurtledorm(int *nLin, int *nCol, int tDorm[][MAX]);`

   Essa função lê do teclado os valores para *nLin, *nCol, semente e dificuldade. Esta função deve devolver, através do parâmetro tDorm, a representação de um turtledorm de dimensão *nLin × *nCol construído de maneira aleatória como descrito a seguir.

   Inicialmente a função deve inicializar as posições de tDorm com zeros. Em seguida a função deve sortear posições para dar tapinhas nas turtles dessas posições. Finalmente, a função deve imprimir o número de tapinhas dados e de turtles despertos no turtledorm, conforme detalhado a seguir.

   O valor de semente deve ser utilizado apenas uma vez, para inicializar o gerador de números pseudo-aleatórios:

   ```
   /* inicializa o gerador */
   srand(semente);
   ```

   Essa semente não seria necessária para um jogo de verdade, mas nos será muito útil na depuração do programa e também na correção.

   Com todas as posições de tDorm contendo inicialmente zero, o programa deve passar a sortear uma linha e uma coluna do turtledorm para dar um tapinha no turtle dessa posição. Para evitar que o guardião saiba exatamente quantos tapinhas ele precisa dar para chegar a uma solução, um nível de dificuldade será usado para definir um número mínimo e máximo de tapinhas a serem dados em função do número de cubículos no turtledorm.

O nível de dificuldade é dado pelo valor lido em `dificuldade`. Esse valor é um caractere e pode ser `'f'` de `f`ácil, `'m'` de `m`édio ou `'d'` de `d`ifícil. Seu programa deve utilizar os níveis de dificuldade definidos na tabela abaixo:

nível de dificuldade	número de tapinhas
`f`ácil	entre 5% e 20% dos cubículos
`m`édio	entre 25% e 50% dos cubículos
`d`ifícil	entre 55% e 85% dos cubículos

Assim, se o guardião deseja resolver um problema difícil em um turtledorm de dimensão 5×5, o seu programa deve sortear inicialmente um número inteiro `nTapinhas` entre `(int)(0.55*5*5)` e `(int)(0.85*5*5)`, incluindo os extremos. Em seguida, a função deve sortear `nTapinhas` posições e dar um tapinha em cada uma delas. Você não deve se preocupar se mais de um tapinha é dado em uma mesma posição.

A função `sorteieTurtledorm` deve imprimir o número sorteado `nTapinhas` de tapinhas e o número de turtles que ficaram despertos após a aplicação dos `nTapinhas` tapinhas. Ao final, a função deve devolver a matriz resultante da aplicação dos tapinhas sorteados através de `tDorm`.

O procedimento descrito garante que o turtledorm produzido tem solução.

Para sortear um número inteiro em um determinado intervalo utilize a função

```
int randomInteger(int low, int high);
```

descrita na página Números aleatórios de Paulo Feofiloff.

8 Descrição da Parte III - Solução ótima (vale 25% da nota)

Na parte III você vai terminar o EP, implementando o cálculo de uma solução ótima (para ajudar o guardião a colocar os turtles para dormir com esforço mínimo).

- `void resolvaTurtledorm(int nLin, int nCol, int tDorm[][MAX]);`

  Tenta resolver o turtledorm representado por `nLin`, `nCol` e pela matriz `tDorm`. Caso exista alguma solução do turtledorm, a função mostra na tela uma **solução ótima**, na forma de uma matriz (veja a figura 2), e imprime uma mensagem indicando o número de tapinhas da solução ótima encontrada. Caso não exista solução, a função deve imprimir uma mensagem avisando o usuário sobre isso.

O exemplo da figura 2 ilustra como uma solução deve ser mostrada ao guardião, na forma de uma matriz em que as posições com `'T'` representam onde os tapinhas precisam ser aplicados.

A solução que você precisa implementar para essa função é de sua livre escolha. Algumas formas possíveis (mas talvez particulares para tabuleiros de tamanho 5×5) estão descritas em

http://en.wikipedia.org/wiki/Lights_Out_(game).

A sua solução precisa funcionar para um turtledorm de tamanho arbitrário `nLin` $\times$ `nCol` (e não, por exemplo, apenas para 5×5). Soluções específicas não serão consideradas e receberão nota zero nessa parte.

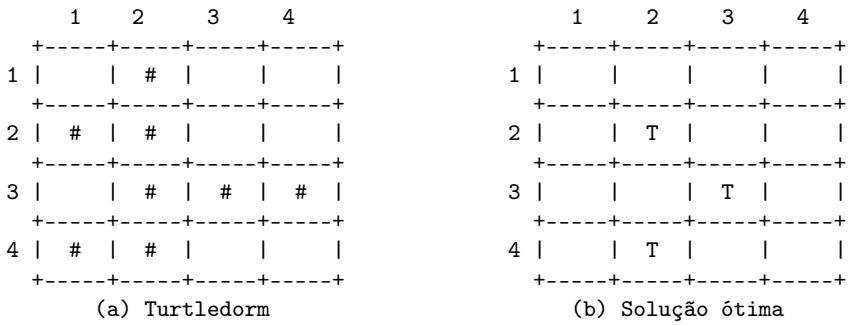

Figura 2: (a) Mostra um turtledorm 4×4. (b) Mostra a solução para esse turtledorm, ou seja, onde os tapinhas (`T`) precisam ser aplicados. Observe que os tapinhas podem ser aplicados em qualquer ordem.

Para evitar plágio, caso você se baseie em alguma dessas (ou outra) soluções, forneça a referência. Alias, você sempre deve procurar dar o crédito a quem é devido, isto, além de ser ético blá-blá-blá . . . , é uma questão de boa educação.

Você pode se basear em uma ideia qualquer e discutir suas ideias com os colegas, mas não deve compartilhar seu código, copiar código ou traduzir código (como de uma solução já pronta em Java).

9 Outros critérios para nota

Um programa com erros de sintaxe será severamente punido, podendo, inclusive, nem ser corrigido (nota zero).

Além de implementar cada parte corretamente, seu programa deve estar bem documentado. Um programa sem documentação ou com documentação muito ruim pode perder até 30% da nota.

Para saber o que é uma boa documentação, leia a página Documentação escrita por Paulo Feofiloff.

10 Observações finais

1. Antes de começar a desenvolver a sua solução, dê uma olhada nos vários exemplos de execução do jogo, bem como nos arquivos de entrada que estão disponíveis na página desse EP.
2. Você pode supor que todos os lances e comandos fornecidos pelo guardião são válidos.
3. Você não deve alterar nenhum protótipo de função.
4. Você deve implementar e entregar todas as funções obrigatórias em seu programa, mesmo que você não as utilize... mas...
5. Caso você NÃO utilize uma das funções obrigatórias, justifique no cabeçalho do seu EP porque a sua solução sem o uso dessa(s) função(s) obrigatória(s) é melhor. Você não precisa justificar a sua solução caso venha a utilizar todas as funções obrigatórias.
6. Outras observações importantes sobre a entrega do exercício estão contidas na página www.ime.usp.br/~cris/mac121/eps/infoeps.
7. Esse exercício é individual. Você pode discutir ideias de soluções com seus colegas, mas não deve compartilhar, copiar ou traduzir código.