ACH2023 - Primeiro Exercício-Programa
Maratona de Programação

Prof. Luciano Antonio Digiampietri

Prazo máximo para a entrega: 19/10/2024

Neste primeiro EP, vocês devem desenvolver, de forma individual, um sistema básico para gerenciar a Maratona de Programação.

1 Sobre a Maratona de Programação

"*A Maratona de Programação é um evento da Sociedade Brasileira de Computação que existe desde o ano de 1996. Nasceu das competições regionais classificatórias para as etapas mundiais da competição de programação, o International Collegiate Programming Contest, e é parte da super-regional latino-americana do evento. Ela se destina a alunos e alunas de cursos de graduação e início de pós-graduação na área de Computação e afins (Ciência da Computação, Engenharia de Computação, Sistemas de Informação, Matemática, etc.). A competição promove nos estudantes a criatividade, a capacidade de trabalho em equipe, a busca de novas soluções de software e a habilidade de resolver problemas sob pressão. A cada ano, observa-se que as instituições de ensino e, principalmente, as grandes empresas da área têm valorizado os alunos que participam do evento.*

Os times são compostos por três estudantes, que tentarão resolver durante cinco horas o maior número possível dos dez ou mais problemas fornecidos. Eles têm à sua disposição apenas um computador e material impresso (livros, listagens, manuais, etc.) para vencer a batalha contra o relógio e a prova proposta. Os competidores do time devem colaborar para descobrir os problemas mais fáceis, projetar os testes, e construir as soluções que sejam aprovadas pelos juízes da competição. Certos problemas requerem apenas compreensão, outros conhecimento de técnicas mais sofisticadas, e alguns podem ser realmente muito difíceis de serem resolvidos.

O julgamento é estrito. Nos enunciados dos problemas constam exemplos dos casos de testes, mas os times não têm acesso às instâncias verificadas pelos juízes. A cada submissão incorreta de um problema (ou seja, que a solução proposta apresenta resposta incorreta a uma das instâncias dos juízes) é atribuída uma penalidade de tempo. O time que conseguir resolver o maior número de problemas (no menor tempo acumulado com as penalidades) é declarado o vencedor."[1]

[1] `https://maratona.sbc.org.br/sobre/`, acessado em 12/08/2024

2 O Problema

Do ponto de vista de Algoritmos e Estruturas de Dados, este EP focará em **listas duplamente ligadas, ordenadas, não circulares e com nó cabeça**.

A Maratona de Programação que você gerenciará tem um número fixo de problemas definido na constante *PROBLEMAS*. Os problemas da maratona serão numerados de 0 (zero) até PROBLEMAS-1. Haverá uma lista ligada ordenada pela classificação dos times, como será discutido ao longo do enunciado.

Há três estruturas (*structs*) envolvidas neste EP: TIME, ELEMENTO e MARATONA.

A estrutura **TIME** foi projetada para armazenar as informações de cada time participante da Maratona de Programação. Ela é composta por quatro campos:

- *id* – identificador numérico do time, começando por 1 (um);
- *resolvidos* – arranjo booleano com tamanho igual ao número de problemas da maratona, que conterá o valor *true*, caso o tinha tenha resolvido (corretamente) o problema e *false*, caso contrário;
- *tentativas* – arranjo de inteiros com tamanho igual ao número de problemas da maratona, que conterá o número de tentativas (submissões) do respectivo problema que o time fez até conseguir resolver o problema; e
- *minutos* – arranjo de inteiros com tamanho igual ao número de problemas da maratona, que conterá o número de minutos passados desde o início da maratona até o momento em que o time submeteu a solução correta do respectivo problema. Para os problemas em que o time não submeteu uma solução correta, o valor 0 (zero) deve ser armazenado na respectiva posição do arranjo.

```
typedef struct {
 int id;
 bool resolvidos[PROBLEMAS];
 int tentativas[PROBLEMAS];
 int minutos[PROBLEMAS];
} TIME;
```

A estrutura **ELEMENTO** foi projetada para armazenar cada elemento da lista duplamente ligada, ordenada, com nó cabeça e não circular que você gerenciará. Esta estrutura é composta por três campos:

- *time* – ponteiro (ou referência) para o time participante da maratona;
- *ant* – ponteiro (ou referência) para o elemento anterior da lista ligada; e
- *prox* – ponteiro (ou referência) para o elemento seguinte da lista ligada.

```
typedef struct aux{
 TIME* time;
 struct aux* ant;
 struct aux* prox;
} ELEMENTO, *PONT;
```

A estrutura **MARATONA** foi projetada para representar a Maratona de Programação. Esta estrutura é composta por dois campos:

- *cabeca* – ponteiro (ou referência) para o nó cabeça da lista duplamente ligada;
- *numTimes* – campo do tipo inteiro contendo o número de times que estão participando da maratona.

```
typedef struct {
 PONT cabeca;
 int numTimes;
} MARATONA;
```

O gerenciamento da Maratona de Programação exige a implementação de diversas funções. Várias delas já estão implementadas no código fornecido do EP, você deve completar a implementação daqueles que estão em azul.

bool inicializarMaratona(MARATONA* mar, int numTimes): função que recebe como parâmetro o endereço de uma MARATONA e o número de times que participarão da competição e realiza a inicialização da maratona. Isto é, cria a lista duplamente ligada com todos os times e o nó cabeça, preenche o valor de todos os campos necessários (valores dos times, elementos e da maratona propriamente dita) e retorna *true*. Porém, se os parâmetros recebidos forem inválidos, retorna *false*, sem realizar a inicialização. Destaca-se que durante a inicialização, a lista ordenada dos times é criada e, neste momento, o time com *id* igual a 1 está em primeiro lugar, com *id* igual a 2 em segundo e assim por diante (se a competição acabar sem nenhuma resolução dos problemas, esta será a classificação dos times).

void mostrarClassificacao(MARATONA* mar): função que recebe o endereço de uma MARATONA e exibe informações básicas dos times de acordo com a ordem de classificação (que é a ordem em que os times estão na lista ligada).

void exibirTime(TIME* t): função que recebe o endereço de um TIME e exibe informações básicas do time (identificador, número de problemas resolvidos e penalidade total).

void exibirTodosTimes(MARATONA* mar): função que recebe o endereço de uma MARATONA e exibe as informações de todos os times (utilizando a função *exibirTime*).

int calcularProblemasResolvidosDoTime(TIME* time): função que recebe o endereço de um TIME e, se esse endereço for igual a NULL, retorna -1. Caso contrário, calcula e retorna o número total de problemas resolvidos pelo respectivo time (o arranjo *resolvidos* armazena, para cada problema, se ele foi ou não resolvido).

int calcularPenalidade(TIME* time): função que recebe o endereço de um TIME e, se esse endereço for igual a NULL, retorna -1. Caso contrário, calcula e retorna a penalidade total (em minutos) recebida pelo time. A penalidade total de um time é dada pela soma dos minutos que o time levou para resolver cada problema (estes valores são armazenados no arranjo *minutos*), considerando apenas os problemas efetivamente resolvidos, mais 20 minutos de penalização para cada submissão incorreta, considerando, novamente, apenas os problemas que o time conseguiu resolver. Por exemplo, se o time conseguiu resolver o problema número 3 apenas na quinta tentativa e a tentativa correta ocorreu no minuto 110, haverá uma penalização de 80 minutos referentes às tentativas incorretas, mais 110 minutos referentes ao tempo que o time levou para resolver esse problema. Assim, haverá uma penalidade de 190 minutos referente ao problema de número 3 (a penalidade total do time corresponde a soma das penalidades do time para todos os problemas da maratona). Destaca-se que problemas não resolvidos corretamente têm penalidade igual a zero.

int calcularProblemasNaoResolvidos(MARATONA* mar): função que recebe o endereço de uma MARATONA e, se esse endereço for igual a NULL, retorna -1. Caso contrário, calcula e retorna o número total de problemas que nenhum time ainda conseguiu resolver. Isto é, se a maratona tem 10 problemas e há três problemas que nenhum time resolveu, deverá retornar 3.

bool tratarSubmissao(MARATONA* mar, int id, int problema, int tempo, bool acerto): função que recebe cinco parâmetros: *mar* - endereço de uma MARATONA, *id* – identificador de um time, *problema* – identificador de um problema, *tempo* – tempo, em minutos, da submissão, *acerto* – parâmetro booleano que indica se o problema foi resolvido corretamente ou não. Esta função deverá retornar *false* caso algum parâmetro seja inválido, isto é *MAR* igual a NULL, ou *id* fora do intervalo que vai de 1 até o número de times da respectiva maratona, ou *problema* fora do intervalo válido dos problemas (que vai de 0 a PROBLEMAS – 1). A função também deverá retornar *false* caso o respectivo time já tenha resolvido corretamente o problema cujo número é dado pelo parâmetro *problema*. Isto é, submissões de problemas já resolvidos pelo time devem ser ignoradas. Caso contrário, a função deve atualizar as seguintes informações do time: (i) aumentar em uma unidade o número de

tentativas de resolução pelo time para o respectivo problema; (ii) caso o time tenha acertado a solução (parâmetro *acerto* = true), (ii.a) atualizar para verdadeiro o respectivo valor do arranjo *resolvidos*; (ii.b) atualizar o valor do arranjo *minutos*, atualizando o valor correspondente ao problema atual com o tempo passado como parâmetro; (ii.c) reposicionar, se necessário, o ELEMENTO correspondente ao time atual na lista duplamente ligada, ordenada, com nó cabeça e não circular. **A lista é ordenada pela classificação do time na maratona, a qual segue estas regras:** a lista é ordenada de forma decrescente do time que resolveu mais problemas para o time que resolveu menos problemas e tendo como critério de desempate as penalidades recebidas pelos times (dados dois times que resolveram a mesma quantidade de problemas, ficará na frente da lista [mais para o início da lista] aquele com menor penalidade). Se dois times empatarem nestes dois critérios, deve ficar à frente da lista aquele que já estava à frente antes da atualização (isto é, se o time A já tinha resolvido dois problemas e tinha penalidade 200 e a função *tratarSubmissao* foi chamada para o time B que, agora, resolveu seu segundo problema e ficou com penalidade 200, o time A deve permanecer à frente na classificação em relação ao time B [como já estava antes desta submissão]). Durante a atualização da posição dos elementos da lista ligada preste bastante atenção para atualizar todos os ponteiros necessários e não esqueça que a lista é duplamente ligada e possui nó cabeça.

2.1 Material a Ser Entregue

Um arquivo, denominado *`NUSP`*`.c` (sendo NUSP o seu número USP, por exemplo: 123456789.c), contendo seu código, incluindo todas as funções solicitadas e qualquer outra função adicional que ache necessário. Para sua conveniência, `completeERenomeie.c` será fornecido, cabendo a você então completá-lo e renomeá-lo para a submissão.

Atenção!

1. Não modifique as assinaturas das funções já implementadas e/ou que você deverá completar!
2. Para avaliação, as diferentes funções serão invocadas diretamente (individualmente ou em conjunto com outras funções). Em especial, qualquer código dentro da função `main()` será ignorado.

3 Entrega

A entrega será feita única e exclusivamente via sistema e-Disciplinas, até a data final marcada. Deverá ser postado no sistema um arquivo .c, tendo como nome seu número USP:

```
seuNumeroUSP.c   (por exemplo, 12345678.c)
```

Não esqueça de preencher o cabeçalho constante do arquivo .c, com seu nome, número USP, turma etc.

A responsabilidade da postagem é exclusivamente sua. Por isso, submeta e certifique-se de que o arquivo submetido é o correto (fazendo seu download, por exemplo). Problemas referentes ao uso do sistema devem ser resolvidos com antecedência.

4 Avaliação

A nota atribuída ao EP será baseada nas funcionalidades solicitadas, porém não esqueça de se atentar aos seguintes aspectos:

1. Documentação: se há comentários explicando o que se faz nos passos mais importantes e para que serve o programa (tanto a função quanto o programa em que está inserida);
2. Apresentação visual: se o código está legível, identado etc;
3. Corretude: se o programa funciona.

Além disso, algumas observações pertinentes ao trabalho, que influenciam em sua nota, são:

- Este exercício-programa deve ser elaborado individualmente;
- Não será tolerado plágio;
- Exercícios com erro de sintaxe (ou seja, erros de compilação), receberão nota ZERO.