Trabalho Prático 2

Algoritmos e Estruturas de Dados III

17 de Outubro de 2018

Entrega: 06/11/2018 até 23:55.

1 Descrição

Um poderoso sheik chamado Mohammed Kaystor visitou a cidade de "Fabulouston" e ficou encantado com a quantidade de cassinos naquela cidade. "Fabulouston" é o destino favorito de jogadores e apostadores de todo mundo! Lá você pode encontrar mesas de pôquer, crupiês (*dealers*) experientes, roletas, diferentes máquinas caça-níqueis e um incrível jogo chamado "CameL-bet". Com tantos jogos você pode sair da cidade milionário. Os jogos não são o único entretenimento da cidade, mas são o carro-chefe e a base de todo o turismo em "Fabulouston".

Mohammed ficou fascinado pelo jogo "CameL-bet" devido a adrenalina causada pelas apostas, o jogador está sujeito a oscilações repetitivas e suaves durante o jogo, mas tais ocilações também podem ocorrer de forma brusca fazendo com que o jogador não ganhe nada. O jogador precisa pensar rapidamente para decidir se vai ou não participar da rodada do jogo. O desafio não é dos mais fáceis, por isso, alguns entusiastas preferem, as vezes, apenas aproveitar o ambiente e a musica apenas para se divertir e conhecer novas pessoas.

A tarefa principal do trabalho é conseguir resolver o jogo "Camel-bet" dizendo se é possível ou não ganhar, além disso dizer também qual é *melhor solução para o jogo e o maior valor que você consegue alcançar, a decisão deve ser feita o mais rápido possível*. Só assim você conseguirá ganhar alguns milhões de dólares ao participar deste jogo.

2 Definição

Neste trabalho, nós estamos interessados em encontrar se é possível ou não ganhar no jogo "CameL-bet" e qual a melhor solução para o jogo que o jogador recebeu. É claro que o jogador não quer sair ganhando pouco, ele sempre buscará obter o maior valor possível antes de sair da mesa de apostas. Mas como funciona o jogo? Veja a Seção 2.1.

2.1 O jogo"CameL-bet"

É necessário compreender o funcionamento do jogo "CameL-bet". Existem as seguintes regras:

- No começo do jogo, o jogador recebe um valor inicial V que está entre 0 e X (inclusive). O jogador também recebe um valor M, que determina o valor mínimo que ele deverá alcançar para ganhar o jogo, que está nesse mesmo intervalo $[0, X]$.

- Em seguida o jogador recebe uma sequência de números S, cada um desses valores também está no intervalo $[0, X]$. Os valores da sequência são sempre positivos.

Nesse momento, o jogador é obrigado a apostar. O jogador deve decidir se vai utilizar o valor da sequência como soma ou subtração. Ao realizar a jogada para cada um dos números na sequência S, o jogador deve ficar atento à:

- Se o valor do número da sequência S será somado ou substraído do *montante corrente* (MC). Antes da primeira jogada, o montante corrente é igual ao valor inicial V, e ele vai sendo atualizado a cada jogada dependendo da decisão do jogador (soma ou substração).

- O *montante corrente* (MC) não pode sair do intervalo $[0, X]$. Se em algum momento esta regra for violada, o jogador perde imediatamente, sendo necessário sair da mesa que está.

- Ao final do jogo, se o *montante corrente final* (MCF) arrecado for maior ou igual ao valor mínimo M. O jogador ganha e recebe o valor máximo que ele obteve. Caso o MCF seja menor que o valor mínimo M e esteja fora do intervalo $[0, X]$, o jogador perde. Se o jogador obteve um MCF menor que o valor mínimo M e o MCF esta dentro do intervalo $[0, X]$, é necessário apresentar o maior valor possível.

Sabemos que é quase impossível solucionar isso devido a quantidade de escolhas existentes utilizando um algoritmo de força bruta. Devido a grandeza desse espaço de possibilidades, os donos dos cassinos acreditam que ninguém conseguirá resolver e desvendar as jogadas do "CameL-bet" em poucos segundos. Mas, eles não sabem que existe uma técnica poderosa para resolver esse problema em tempo hábil muito menor. E mais, você conhece a técnica e está prestes a ganhar milhões de doláres e se tornar o novo milionário em "Fabulouston" !!!.

2.2 Objetivos

O objetivo desse trabalho prático é o desenvolvimento de 2 programas utilizando as técnicas e os conceitos de paradigmas de programação.

- **Força bruta**

- **Programação Dinâmica** ou **Algoritmo Guloso**. Note que apenas um desses dois paradigmas resolve o problema.

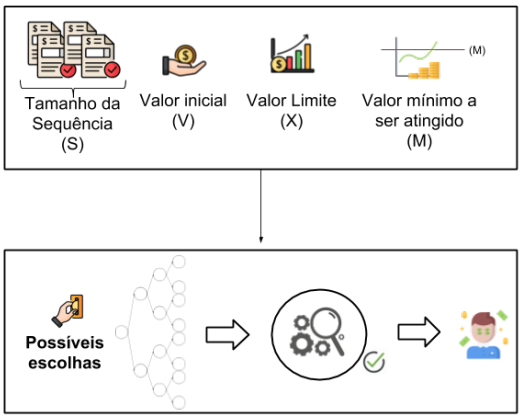

Figura 1: Etapas necessárias para validar uma solução.

3 Entrada

Seu programa deve ser capaz de ler a entrada de dados através da entrada padrão (*stdin*) e gravar a saída na saída padrão (*stdout*). A entrada de teste é composta por vários casos de teste.

Para cada caso de teste, a primeira linha contém um inteiro I ($1 \leq I \leq 10$) que corresponde ao número de instâncias. Cada instâncias é composta por duas linhas. A primeira contém quatro inteiros, que representam respectivamente: o tamanho da sequência S ($1 \leq S \leq 10^3$), o valor inicial V ($0 \leq V \leq X$) do jogo para o jogador, o valor limite X ($1 \leq X \leq 10^3$)e o valor mínimo M ($1 \leq S \leq 10^6$)a ser atingido pelo jogador. A segunda linha contém os valores da sequência S na ordem em que eles devem ser usados. **Nenhuma ordem deve ser alterada**.

Um exemplo de entrada para o prgrama está na Tabela 1

Tabela 1: Exemplo *Toy* para ilustrar a entrada do problema. Esse teste está disponível em um arquivo no moodle.

input
3
3 5 10 10
5 3 7
4 8 20 10
15 2 9 10
8 516 668 442
79 417 337 349 40 41 45 42

4 Saída

Para cada instância do problema, deve ser impresso na saída dois valores separados por vírgula. Do seguinte modo:

1º valor:
Se é possível ganhar o jogo, "S" para Sim e "N" para Não;

2º valor:
O valor máximo que pode ser atingido naquele caso de teste. Caso este valor esteja fora do intervalo apresentado na Seção 2, deverá ser impresso -1.

Não pode existir espaço entre a resposta "S"/"N" e o valor máximo caso seja possível ganhar o jogo ou -1 para valores fora do intervalo de $[0, X]$. Um exemplo de saída para o programa está na Tabela 2.

Tabela 2: Exemplo *Toy* para ilustrar a saída do problema. Esse teste está disponível em um arquivo no moodle.

output
S,10
N,-1
N,334

Observação: Fique atento quanto a complexidade do seu algoritmo, para cada caso de teste é esperado que ele execute em no máximo 1 segundo. Além disso, utilize o **diff** do Linux para conferir se o seu algoritmo está com a saída correta e de acordo com as respostas de exemplo.

5 Entrega

- A data de entrega desse trabalho é até às 23:55 do dia **06/11/2018** (06 de Novembro de 2018).
- A penalização por atraso obedece à formula apresentada na Equação 1, onde d são os dias úteis de atraso na entrega do projeto.

$$\frac{2^{d-1}}{0.32\%} \tag{1}$$

- Submeta apenas um arquivo chamado *<número matricula>_<nome>.zip*. Não utilize espaços no nome do arquivo. Ao invés disso utilize o caractere "_". Não envie o arquivo compactado .rar, o formato de arquivo ZIP é mais acessível que o RAR.
- Não inclua arquivos compilados ou gerados por IDEs. **Apenas** os arquivos abaixo devem estar presentes no arquivo zip:

  - Makefile;
  - Arquivos fonte (*.c e *.h);
  - documentacao.pdf.

- Não inclua **nenhuma** pasta. Coloque todos os arquivos na raiz do zip.
- Siga rigorosamente o formato do arquivo de saída descrito na especificação. Tome cuidado com *whitespaces* e formatação dos dados de saída.
- **NÃO SERÁ NECESSÁRIO ENTREGAR DOCUMENTAÇÃO IMPRESSA!**
- A sua nota final será 0 se uma das partes (**documentação ou execução**) não for apresentada.

6 Documentação

A documentação não deve exceder o limite de **12 páginas**, sendo submetida no formato PDF e deve conter pelo menos os seguintes itens:

- Uma introdução do problema em questão com uma explicação clara e objetiva;
- Modelagem e solução proposta para o problema. O algoritmo deve ser explicado de forma clara. Se necessário, utilize pseudo-código e/ou esquemas ilustrativos. **Apresentar qual das duas técnicas não é aplicável ao problema e mostrar o porquê**;
- Análise de complexidade de tempo e espaço da solução usando o formalismo da notação assintótica visto em sala de aula;
- Análise experimental, variando-se o tamanho da entrada e quaisquer outros parâmetros que afetem significativamente a execução do algoritmo;
  - **Cabe a você gerar as entradas para esses experimentos**.
- Especificação da máquina utilizada nos experimentos realizados;
- Uma breve conclusão do trabalho implementado.

Um tutorial para elaboração da documentação e também um exemplo são apresentados no *minha.ufmg*[1]. Para visualizar o tutorial para elaboração da documentação (**clique aqui**), já para visualizar o exemplo de uma documentação pronta (**clique aqui**).

[1] `https://sistemas.ufmg.br/idp/login.jsp`

7 Código

O código deve ser obrigatoriamente escrito na **linguagem C** ou **C++**. A utilização da biblioteca STL do C++ é permitida apenas para estruturas vistas em disciplinas anteriores (AEDS1 e AEDS2). Os algoritmos ensinados em sala de aula devem ser implementados sem o auxilio da biblioteca STL. Ele deve compilar e executar corretamente nas máquinas Linux[2] dos laboratórios de graduação.

- O utilitário ***Make*** deve ser utilizado para auxiliar a compilação, um arquivo *Makefile* deve portanto ser incluído no código submetido. O utilitário deverá gerar um executável com o nome **tp2** que deverá **obrigatoriamente** ser capaz de receber o nome do arquivo de entrada e do arquivo de saída. O comando para executá-lo deverá seguir o exemplo abaixo:
  /.tp2-forcabruta input.txt output.txt
  /.tp2-pd input.txt output.txt

- As estruturas de dados devem ser **alocadas dinamicamente** e o código deve ser **modularizado** (divisão em múltiplos arquivos fonte e uso de arquivos cabeçalho *.h*)

- **Variáveis globais** devem ser evitadas.

- Parte da correção poderá ser feita de forma automatizada, portanto **siga rigorosamente os padrões de saída especificados**, caso contrário sua nota pode ser prejudicada.

- **Legibilidade e boas práticas** de programação serão avaliadas.

- **Não é permitido o uso de bibliotecas de terceiros**[3].

- **Não é permitido o compartilhamento de código entre os estudantes**. Indícios de plágio serão investigados e, caso confirmados, serão severamente punidos.

- **Seja honesto**, você não aprende nada copiando código de terceiros nem pedindo a outra pessoa que faça o trabalho por você.

Bom trabalho!

[2] `http://www.crc.dcc.ufmg.br/infraestrutura/laboratorios/labs_unix`
[3] Dúvidas quanto a utilização de uma biblioteca em específico deverá ser sanada com o monitor responsável.

Referências

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[2] Steven Halim and Felix Halim. *Competitive Programming 3*. Lulu Independent Publish, 2013.

[3] Nivio Ziviani. Projeto de algoritmos com implementação em pascal e c. 2ª. *São Paulo: Thomson*, 2004.