MAC0115 - Introdução à Computação para Ciências Exatas e Tecnologia

Alfredo Goldman e Beatriz Viana

Setembro de 2024

EP 2

Simulação de Apostas

Neste EP, você deverá desenvolver um código para simular apostas aleatórias. O código deve utilizar as bibliotecas `Random` e `Plots` do Julia. Essas bibliotecas são essenciais para a geração de números aleatórios, plotagem de gráficos e manipulação de dados.

Instalação das Bibliotecas

Para utilizar as bibliotecas mencionadas, você deve instalá-las no terminal do Julia. Siga os passos abaixo:

1. Abra o terminal do Julia.
2. Inicie o modo de gerenciamento de pacotes digitando `]`. O prompt mudará para `pkg>`.
3. Digite os seguintes comandos para instalar as bibliotecas necessárias:

```
pkg> add Random
pkg> add Plots
```

4. Após a instalação, você pode voltar ao modo normal digitando `Backspace` ou `Ctrl+C`.

Ambientes para Visualização

Para visualizar os gráficos gerados, você pode utilizar o seguinte ambiente:

- **Julia REPL:** O terminal do Julia que permite executar código diretamente e visualizar gráficos em uma janela externa.

Ou pode também usar o comando `savefig("nome_da_imagem.png")` para salvar o gráfico gerado no mesmo diretório do script que está sendo executado.

Requisitos

Você deve implementar as seguintes funções:

1. `gerar_aposta(numeros::Int)`:

   Esta função é responsável por gerar uma aposta aleatória. O parâmetro `numeros` define quantos números devem ser selecionados aleatoriamente, sendo um valor entre 6 e 10. A função deve escolher números únicos no intervalo de 1 a 60 e retorná-los em um vetor ordenado.

2. `simular_apostas(numeros_selecionados::VectorInt, max_numeros::Int)`:

   Esta função simula a geração de apostas até que todos os números especificados em `numeros_selecionados` sejam acertados. O vetor de números que o usuário deseja acertar é passado como argumento, assim como o parâmetro `max_numeros`, que indica o número máximo de seleções permitidas. A função deve contar quantas apostas foram necessárias para acertar todos os números e retornar esse total.

3. `plota_grafico(resultados::VectorInt)`:

   Esta função tem como objetivo visualizar a distribuição do número de apostas necessárias para acertar todos os números selecionados em uma série de simulações. A função recebe um vetor de inteiros, resultados, que contém o total de apostas realizadas em cada simulação.

   Essa função chama a função `gerar_aposta()`.

   Descrição do Funcionamento:

   1. **Definição de Intervalos:**
      A função começa definindo um conjunto de intervalos que será utilizado para agrupar os resultados. Esses intervalos são especificados como uma lista de valores, representando faixas de número de apostas, como [0, 10000, 100000, 1000000, 10000000, 100000000].

   2. **Contagem de Frequências:**
      Em seguida, a função inicializa um vetor frequencias_intervalos para armazenar a contagem de resultados em cada intervalo. Um loop percorre cada resultado no vetor resultados, e outro loop verifica em qual intervalo cada resultado se encaixa, incrementando a contagem correspondente.

   3. **Impressão das Frequências:**
      Após a contagem, a função imprime o vetor de frequências, permitindo uma verificação rápida do número de apostas em cada faixa.

   4. **Criação do Gráfico:**
      A função utiliza a biblioteca de plotagem para gerar um gráfico de barras. No gráfico, os intervalos de número de apostas são exibidos no eixo x, enquanto as frequências correspondentes aparecem no eixo y. O gráfico é rotulado com títulos e eixos apropriados para melhor compreensão.

   5. **Salvamento do Gráfico:**
      Por fim, o gráfico é salvo como um arquivo PNG chamado distribuicao_apostas.png, permitindo que o usuário visualize os resultados fora do ambiente de execução.

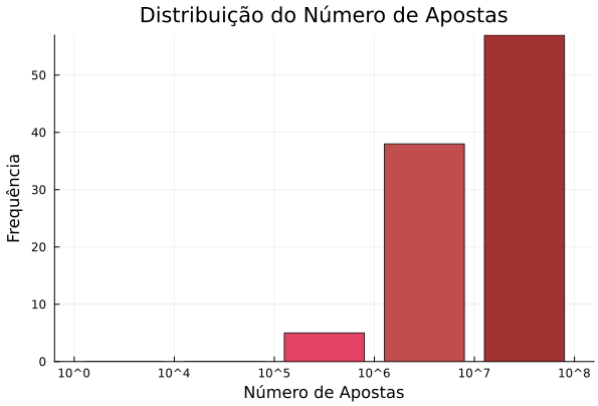

Figura 1: Distribuição da quantidade de tentativas para 100 simulações.

4. `main()`:

   Esta é a função principal que irá coordenar a execução do código. Ela deve definir os parâmetros para a simulação, como o número máximo de seleções permitidas e o número de simulações a serem realizadas. A função deve invocar `simular_apostas` e coletar os resultados.

   Nessa função, você deve simular um total de 100 apostas. Além disso, será necessário definir os parâmetros que serão passados para as demais funções, incluindo o vetor `numeros_selecionados` e o valor de `max_numeros`.

   O vetor `numeros_selecionados` pode ser gerado de forma aleatória ou definido manualmente, desde que contenha entre 6 e 10 números, todos distintos e no intervalo de 1 a 60. O parâmetro `max_numeros` deve ser fixado em 10, pois as apostas podem incluir entre 6 e 10 números.

   Essa função chama as funções `simular_apostas()` e `plotar_grafico()`.

Exemplo de Uso

Este é um exemplo de gráfico gerado para 100 simulações:

Entrega

Um código-fonte foi disponibilizado com o cabeçalho de cada função, você deve utilizar este código como base para desenvolver sua solução. Por fim, o código-fonte da solução deve ser submetido, com a extensão de Julia, `.jl`.

Não esqueça de fazer um código bem escrito e indentado :).

Extra

As simulações podem ser demoradas, especialmente ao testar a lógica das funções `simular_apostas()` e `plotar_grafico()`. Para otimizar o processo de desenvolvimento

e garantir que você possa validar rapidamente a funcionalidade do código, recomenda-se utilizar vetores de valores fixos e/ou testar para poucas simulações.

Por exemplo, você pode criar um vetor com um conjunto pré-definido de resultados que simula o comportamento esperado das apostas. Isso permite que você teste a lógica da função `simular_apostas()` sem depender de simulações aleatórias.

Além disso, ao testar a função `plotar_grafico()`, você pode usar dados de entrada conhecidos para garantir que a visualização está correta. Com isso, você poderá verificar se o gráfico gerado corresponde à distribuição esperada, facilitando a identificação de quaisquer problemas na lógica de contagem ou na geração do gráfico.

Utilizar valores fixos durante a fase de testes não só acelera o processo, mas também torna mais fácil reproduzir e corrigir erros, uma vez que você pode controlar todos os aspectos da entrada e verificar a saída de maneira mais eficaz.

Entretanto, é importante lembrar que, ao final do processo de desenvolvimento e testes, você deve deixar os vetores como variáveis, garantindo que o código possa executar simulações reais com dados dinâmicos :).

Referências

Estas são algumas referências que podem ajudar:

1. Biblioteca Random;
2. Bar Plot;