https://eventos.utfpr.edu.br//sicite/sicite2020

Desenvolvimento de uma ferramenta computacional para correção de rotinas

Development of a computational tool to correct routines

Felipe Gimenez da Silva
felipegimenezsilva@gmail.com
Universidade Tecnológica Federal do Paraná, Ponta Grossa, Paraná, Brasil

Erikson Freitas de Morais
emorais@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Ponta Grossa, Paraná, Brasil

Iara da Cunha Ribeiro da Silva
iarasilva@utfpr.edu.br
Universidade Tecnológica Federal do Paraná, Ponta Grossa, Paraná, Brasil

RESUMO

Durante as aulas na UTFPR, muitos professores utilizam linguagens de programação, como ocorre com a linguagem GNU Octave. Para auxiliar os professores na correção das tarefas sugeridas em aulas, está sendo realizada uma plataforma de envios, com correções automáticas. Este relatório trata da forma de avaliação utilizada internamente pela plataforma, e do cálculo utilizado para obter a nota do aluno. Além disso, este relatório também mostra os resultados obtidos dos processamentos de casos reais submetidos por alunos. Com a utilização da plataforma, o professor e os alunos poderão analisar o resultado de suas submissões mais rapidamente.

PALAVRAS-CHAVE: Computação. Automação. Cálculos numéricos.

ABSTRACT

During classes at UTFPR, many teachers use programming languages, as with the GNU Octave language. To assist teachers in the correction of tasks suggested in classes, a submission platform is being carried out, with automatic corrections. This report deals with the form of assessment used internally by the platform, and the calculation used to obtain the student's grade. In addition, this report also shows the results obtained from the processing of real cases submitted by students. Using the platform, the teacher and students will be able to analyze the result of their submissions more quickly.

KEYWORDS: Computing. Automation.Numerical calculations.

Recebido: 19 ago. 2020.
Aprovado: 01 out. 2020.

INTRODUÇÃO

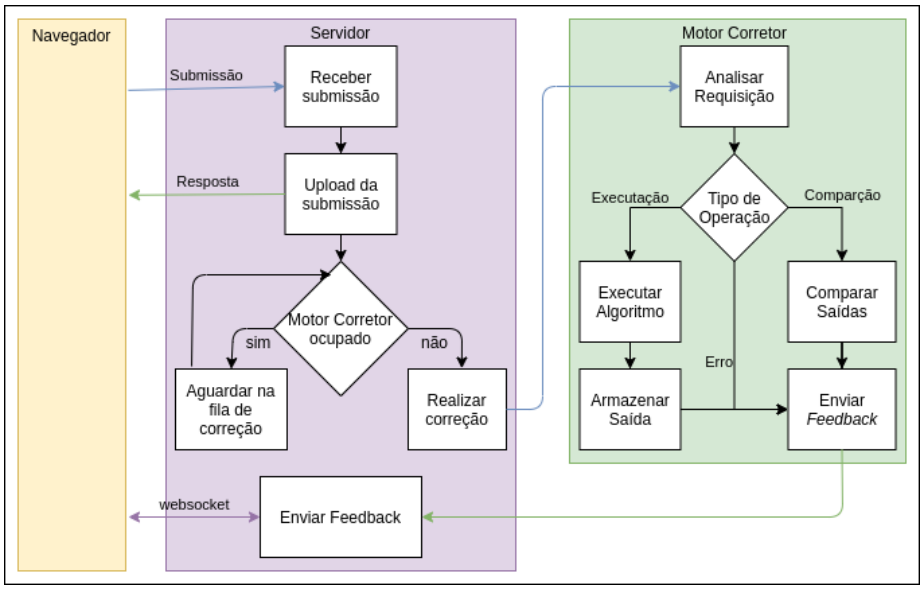
Na disciplina de cálculo numérico ministrada na UTFPR campus Ponta Grossa, padronizou-se como linguagem de programação o uso da linguagem Octave. Toda semana são aplicados exercícios computacionais, onde os alunos precisam implementar algoritmos numéricos. Para auxiliar as correções dessas rotinas computacionais está sendo construída na UTFPR uma plataforma de submissão e correção das tarefas realizadas pelos alunos.

Este relatório trata do desenvolvimento da parte de correção de rotinas computacionais, e contém a descrição dos passos seguidos para avaliar rotinas programadas em Octave, calculando notas entre zero a dez. A pontuação é usada em conjunto com a plataforma para submissões de exercícios, com o objetivo de promover um feedback rápido para os alunos ao realizar uma tarefa, além de auxiliar os professores na avaliação de atividades. O software realizado apenas atribui uma nota ao algoritmo enviado, portanto não realiza correções no código.

METODOLOGIA

Esta seção contém os passos para a execução automatizada da avaliação de rotinas programadas em Octave e a forma de comparação dos resultados obtidos por cada rotina. A execução e avaliação automatizada, é usada em conjunto com uma plataforma de submissão, seguindo a estrutura da Figura 1.

Figura 1 - Estrutura



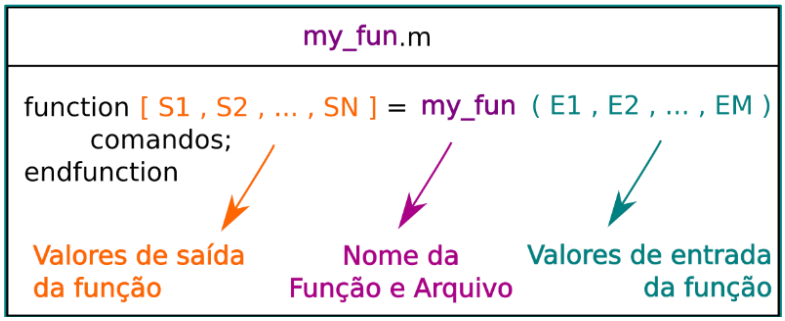
Fonte: Autoria própria.

A execução automática considera que as funções de alunos e de professores a serem chamadas estejam previamente salvas em arquivos, e que as mesmas estejam de acordo com os padrões de função em arquivos do Octave [1], como mostra a Figura 2.

A linguagem Octave permite que o programador salve suas funções em arquivos, possibilitando o reuso em outras partes de código. Para evitar a

ocorrência de erros, o arquivo armazenado deve seguir um padrão [1], como mostra a Figura 2. Ao receber um comando para executar a chamada de uma função que não está compilada, o Octave faz a busca de um arquivo que tenha o mesmo nome da função desejada [1]. Caso exista um arquivo com os requisitos necessários, a função será compilada e executada. Esse padrão de nomenclatura permite encontrar a função salva sem a necessidade de carregar todos os arquivos na memória RAM, além de possibilitar a descoberta do nome da rotina utilizando apenas a URL do arquivo [1].

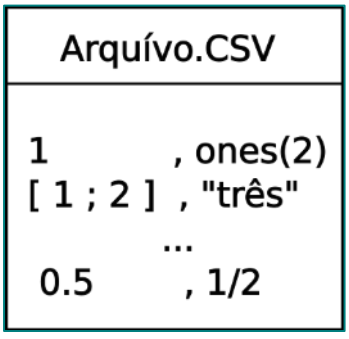
Figura 2 - Padrão de função em arquivos do octave



Fonte: Autoria própria.

Para a inserção das entradas automaticamente, foi criado um padrão de arquivo inspirado no formato de arquivos de valores separados por vírgula (*comma-separated values*, CSV [4]), como mostra a Figura 3. Cada linha do arquivo é referente a um conjunto de parâmetros de entrada para ser utilizado em uma função. O arquivo se difere de um CSV comum para aceitar parâmetros que contenham vírgulas, como ocorre na *string* "olá, sou um único parâmetro". Sem a alteração no formato CSV original causaria a separação entre "olá" e "sou um único parâmetro" durante a interpretação do arquivo, tornando-se duas entradas distintas.

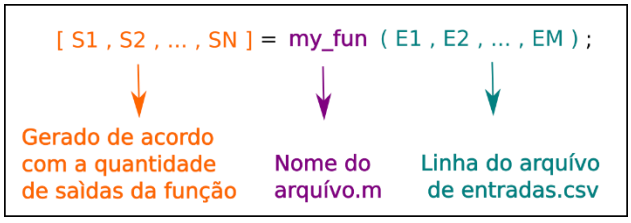
Figura 3 - Arquivo de entradas padronizado



Fonte: Autoria própria.

Para obter a quantidade de saídas de uma função válida do Octave, foi utilizado o comando "nargout('<nome da função>');" da linguagem Octave. Dessa forma foi possível gerar automaticamente os *scripts* para chamadas de função, seguindo o padrão da linguagem Octave [1], como mostra a Figura 4. Para simplificar o entendimento, foram omitidos os comandos de tratamentos de erros da linguagem, como "try".

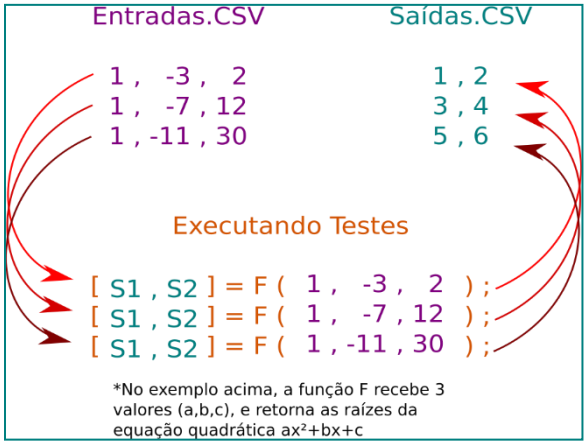
Figura 4 - Chamada de função gerado automaticamente



Fonte: Autoria própria.

Após a geração dos *scripts* de chamadas de funções é feita a execução dos mesmos. Cada conjunto de saída foi escrito em um arquivo seguindo o mesmo padrão do arquivo de entradas (CSV modificado), como mostra a Figura 5.

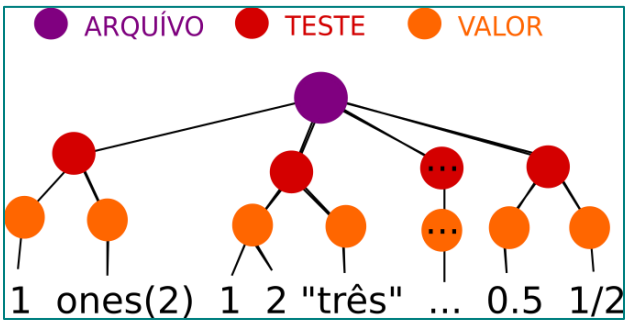
Figura 5 - Ciclo de execução dos *scripts* automáticos



Fonte: Autoria própria.

Ao executar duas funções que têm o mesmo objetivo, com as mesmas entradas, é esperado que as funções obtenham valores iguais ou próximos. A comparação de dois arquivos de saída mostra a distância entre os resultados obtidos por duas rotinas. Para que a análise dos resultados ocorra, o arquivo de saída foi convertido em uma árvore, como mostra a Figura 6.

Figura 6 - Árvore de um arquivo de entradas ou saídas

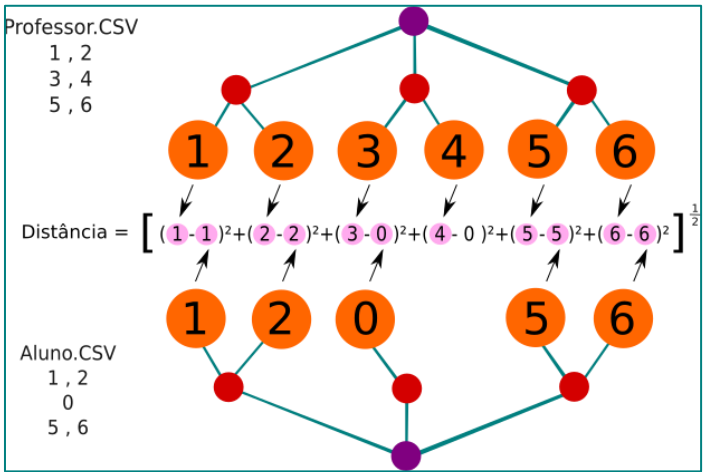


Fonte: Autoria própria.

A comparação é feita considerando dois arquivos de saídas, onde um é o gabarito e outro é a saída que será avaliada. Ao criar a árvore dos dois arquivos, como é mostrado na Figura 7, cada nó folha é comparado e a distância é obtida através do cálculo de distância euclidiana [2]. A distância calculada informa se o

algoritmo analisado obteve boas saídas nos testes. Quando a distância se aproxima de zero, a nota do algoritmo se aproxima de dez.

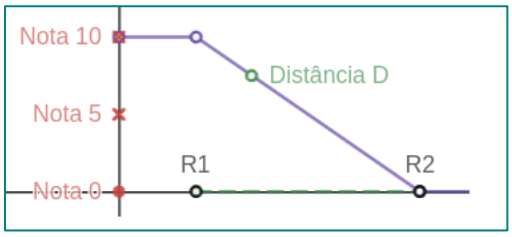
Figura 7 - Cálculo da distância entre dois arquivos de saídas



Fonte: Autoria própria.

Para definir a nota final é preciso controlar o erro mínimo e o erro máximo permitido dentro da avaliação automática. O erro mínimo (R1) indica o valor de erro que o aluno pode ter sem prejudicar a sua nota, obtendo a nota integral. O erro máximo (R2), indica qual o valor de erro que o aluno pode ter para ser avaliado, pois qualquer valor maior que R2 implicará na nota 0. Ao utilizar o erro mínimo, máximo, a distância entre o algoritmo analisado e o algoritmo de referência, é possível definir a tolerância da correção automática, além de calcular uma nota entre zero e dez, como mostra a Figura 8.

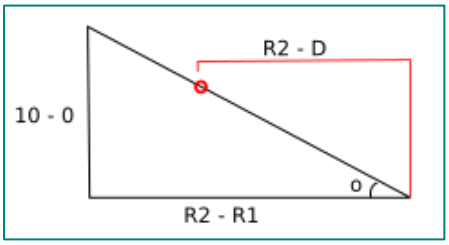
Figura 8 - Relação entre Distância, R1 e R2 com a nota calculada



Fonte: Autoria própria.

A Figura 8 foi utilizada para obter a equação das notas. Com ela, foi possível explorar a relação entre as variáveis R1, R2, D para a composição da nota, como mostra a Figura 9.

Figura 9- Análise da Figura 8, no intervalo [R1, R2]



Fonte: Autoria própria.

A equação das notas, foi obtida por:

$$Nota = \tan(o) * (R2 - D) \quad (1)$$

Para o cálculo da tangente de 'o', é utilizado :

$$\tan(o) = \frac{\text{cateto oposto}}{\text{cateto adjacente}} \quad (2)$$

Então, ao inserir na equação 2 os valores vistos na Figura 8, temos que:

$$\tan(o) = \frac{10}{R2 - R1} \quad (3)$$

Com isso, temos que a equação 1 será :

$$Nota = \frac{10}{R2 - R1} * (R2 - D) \quad (4)$$

A equação 4 é utilizada sempre que a distância se encontra no intervalo [R1,R2]. Caso a distância não esteja no intervalo, a nota conterá valores constantes, como pode ser observado na Figura 8.

RESULTADOS E DISCUSSÕES

Foi utilizada a comparação numérica, que interfere significativamente no resultado. Se considerar, por exemplo, as saídas "0001" e "0.99", a comparação dos caracteres informa que apenas o primeiro símbolo está correto (o símbolo "0"), enquanto a comparação numérica informa que a distância entre os números é de 0.01 . Caso a comparação seja feita com outros símbolos é aplicada a tabela ASCII. Essa forma de comparação ocorre de modo diferente de sistemas como URI Online Judge [3].

Os resultados foram obtidos pela análise de 41 atividades propostas na Universidade Tecnológica Federal do Paraná Câmpus Ponta Grossa antes da pandemia de Covid-19, totalizando 859 rotinas, incluindo rotinas de alunos (submissões) e de professores (gabaritos). Cada atividade possui uma quantidade diferente de testes, causando múltiplas execuções de cada rotina. O processamento ocorreu durante 47 minutos, utilizando um processador Intel(R) Pentium(R) CPU P6200, e 2 GB de memória RAM.

A fila de processamento foi formada por 859 requisições de execução, uma para cada rotina submetida, seguida por 826 requisições de comparação (Tabela 1), comparando a saída da rotina do professor com as saídas das rotinas dos alunos. Ocorreram 85 falhas durante a execução dos algoritmos, devido a tempo excedido, erros sintáticos em rotinas e formatos inválidos. É esperado que o aluno corrija os problemas descritos ao ser alertado na plataforma de submissões, realizando uma nova submissão. Houve também 85 falhas durante a comparação, em consequência dos erros ocorridos durante a execução.

Tabela 1 – Status de requisições

Requisição	Quantidade	Quantidade de falhas	Quantidade de sucessos	Implicações em casos de falhas
Execução de Algoritmos	859	85	774	Não gera saída
Comparação de Saídas	826	85	741	Nota 0 ao comparar saída não existente
Total	1685	170	1515	

Fonte: autoria própria.

Alguns casos de comparação resultaram em valores como *NaN* (*not a number*) e *Inf* (*Infinity*), mostrados na Tabela 2, obtidos quando ocorre *overflow* durante o processamento da distância. Isso indica que a rotina analisada possui resultados muito distantes dos valores presentes no gabarito. Para esses casos, a nota obtida será zero, pois é considerado que a distância ultrapassou o erro máximo (R2).

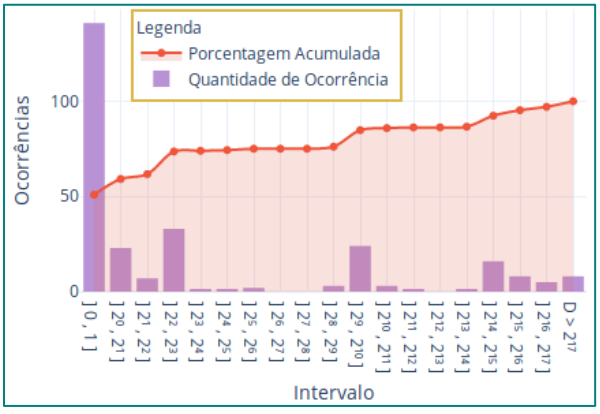
Tabela 2 – Notas

Nota	Quantidade	Motivo
10	456	Saídas exatamente iguais ao gabarito
0	85	Rotina não executável
0	7	Distância resultou em *inf*
0	1	Distância resultou em *NaN*
Dependente dos critérios adotados	277	Saídas com divergências (Grandes ou pequenas)

Fonte: autoria própria.

Ocorreram 277 casos (mostrado na Tabela 2) onde a nota só pode ser obtida com os valores de configuração de R1 e R2. Essas notas podem variar de acordo com a configuração adotada pelo professor. Porém, ainda é possível analisar as distâncias obtidas, para estimar os próximos resultados. A Figura 10 mostra o intervalo que os casos se encaixam, além de mostrar também que a maioria das distâncias entre as rotinas de alunos e de professores são de baixo valor.

Figura 10 – Quantidade de ocorrências em um intervalo



Fonte: Autoria própria.

CONCLUSÕES

Os parâmetros de controle (R1 e R2) são escolhidos de modo manual, permitindo que o professor torne mais flexíveis ou mais rígidas as correções a qualquer momento.

A utilização do cálculo de distância euclidiana trata todas as saídas com a mesma importância. Em outras palavras, o erro obtido por cada saída é tratado igualmente, e não há pesos que os diferem. Inserir pesos pode melhorar o controle da correção, mas não foi inserido dentro da plataforma inicialmente.

Como a comparação utiliza duas árvores de saídas (Figura 7), a ordem de saída dos parâmetros é importante. Caso as saídas não estejam ordenadas, a comparação não obterá bons resultados. O mesmo ocorre com os parâmetros de entrada, que devem seguir a mesma ordem em ambas rotinas.

O modo de comparação se mostrou efetivo para as rotinas testadas, porém a escolha dos parâmetros de controle ainda são muito abstratas e inicialmente podem ser difíceis ao usuário, necessitando mais de uma tentativa para configurá-lo corretamente.

A correção de 859 algoritmos durou 47 minutos, mostrando ser uma velocidade muito superior se comparado ao tempo utilizado por um professor.

REFERÊNCIAS

[1] GNU OCTAVE. GNU Octave: Scientific Programming Language. Disponível em: <https://www.gnu.org/software/octave/>. Acesso em: 13 de julho de 2020.

[2] STRANG, G. Linear Algebra and Its Applications, 4o. ed, Thomson, Brooks/Cole, 2006.

[3] URI ONLINE JUDGE. University of Rhode Island. Disponível em: <https://www.urionlinejudge.com.br/judge/en/login>. Acesso em: 13 de julho de 2020.

[4] SOLIDMATRIX TECNOLIGIES. Common Format and MIME Type for Comma-Separated Values (CSV) Files, RFC 4180. Disponível em : <https://tools.ietf.org/html/rfc4180> . Acesso em: 30 de julho de 2020.