Desenvolvimento de um Sistema de Gerenciamento de Pontuação e Ranking para Maratonas de Programação

Marcos Vinicius Dias Oliveira[1]

[1]Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana – BA – Brasil

marcosvdiaso@gmail.com

Resumo. *Este relatório descreve o desenvolvimento de um sistema para gerenciamento de pontuação e ranking para maratonas de programação. O sistema é capaz de armazenar a pontuação das questões, cadastrar os problemas resolvidos e então exibir um relatório final com base nos dados informados. O programa foi desenvolvido utilizando a linguagem de programação Python e houve êxito em seu desenvolvimento.*

Palavras-chave: Maratona de Programação. Python. Sistema.

1. Introdução

As maratonas de programação são eventos em que equipes ou indivíduos são desafiados a resolver problemas envolvendo algoritmos, matemática e estruturas de dados. Durante a competição, as soluções computam pontos para a equipe, e o tempo gasto na resolução também é registrado. Isso exige que as equipes não apenas resolvam os problemas corretamente, mas também de maneira eficiente e rápida. Com o objetivo de gerenciar de forma eficaz um desses eventos que será organizado pela UEFS, armazenando dados como tempo e pontuação de diferentes equipes, além de poder comparar esses dados, foi solicitado aos alunos do 1º semestre o desenvolvimento de um sistema de gerenciamento de pontuação e ranking, que atenda a requisitos específicos e seja reutilizável.

Este relatório tem como objetivo descrever o processo de desenvolvimento de um sistema em *Python*. O sistema foi projetado para registrar a pontuação de cada questão com base em sua dificuldade (classificadas como fácil, média e difícil). Em seguida, permite o cadastro dos problemas resolvidos por equipe, sendo o sistema configurado, por padrão, para cinco equipes. Após o cadastro de todos os problemas, o sistema gera um relatório final com as principais informações da maratona, como ranking, pontuação média das equipes e o número de problemas resolvidos por cada equipe.

O sistema foi desenvolvido utilizando a linguagem de programação *Python* 3.12, com o *PyCharm* 2024.2.1 como Ambiente Integrado de Desenvolvimento (*IDE*) e o sistema operacional *Windows* 11.

2. Metodologia

Nesta seção, serão detalhados os requisitos e funcionalidades do sistema, conforme o que foi solicitado no problema original, as discussões realizadas durante as sessões tutoriais e com base nos sistemas e ferramentas utilizados na elaboração do projeto.

Serão abordadas as necessidades identificadas, as soluções propostas e como essas soluções foram implementadas para atender aos requisitos estabelecidos.

2.1. Requisitos

O desenvolvimento do programa exigiu o cumprimento de alguns requisitos:

- Os valores inseridos pelo usuário devem passar por validação, a fim de verificar se foram inseridos valores válidos ou não.
- O programa deve iniciar solicitando ao usuário quanto cada dificuldade de problema irá ter de pontuação, esses valores são inseridos pelo usuário pois o programa deve ser customizável com o propósito de poder ser reutilizável.
- Durante o cadastro dos problemas, o programa deve coletar as seguintes informações:
    - A equipe que resolveu o problema;
    - A dificuldade do problema resolvido;
    - O tempo gasto para resolver o problema.

Após o cadastro de todos os problemas, o programa deve gerar e exibir o relatório final da maratona.

Com relação a forma que o tempo é registrado no programa, foi decidido durante as sessões que cada um decidiria qual forma acharia melhor de implementar. No programa que esse relatório aborda, o tempo é armazenado em horas, minutos e segundos, com a finalidade de deixar a entrada desses dados mais simples para o usuário, para que ele não precise fazer cálculos de conversão de tempo, cálculos esses que são feitos pelo próprio programa.

2.2. Pontuação das questões

A primeira etapa do desenvolvimento foi fazer a entrada dos valores que cada uma das dificuldades de questões teriam como pontuação, junto com a validão para esses valores. Para tal, foram utilizados *inputs* dentro de um *loop*, que rodam até que um valor válido seja inserido.

```
verificador = False
while verificador == False:
    facil = input("Qual será a pontuação das questões fáceis? ")
    facil_float = facil.replace( __old: ".", __new: "", __count: 1)
    if facil_float.isdigit():
        facil = float(facil)
        verificador = True
    else:
        print("ERRO! O que foi inserido não é um número válido, tente novamente.")
```

Figura 1. Validação das questões fáceis.

Como mostrado na figura acima, primeiramente é declarada uma variável booleana “verificador” com o valor *False,* após isso o programa entra em um *loop*, onde solicita ao usuário por meio de um input qual a pontuação do tipo de questão especificado. Enquanto for digitado algo que não for um número válido, como letras ou número negativos, o valor não será aprovado na função *isdigit*, e com isso será exibida uma mensagem de erro para o usuário e será solicitada nova entrada. Quando o usuário

inserir um número válido, esse número é armazenado em uma segunda variável junto a função *replace*, para que possa ser removido o ponto caso seja um número decimal, então a segunda variável é verificada por meio da função *isdigit*, após isso a variável original é convertida a *float* e o "verificador" tem seu valor alterado para *True*, assim encerrando o *loop*.

Nas validações de pontuação para questões médias e difíceis é utilizada a mesma lógica, com apenas uma alteração, nesses casos é verificado se o valor inserido é menor ou igual que o valor da categoria anterior, caso seja menor, é apresentado um erro e é solicitado um novo valor, como pode ser visto na figura abaixo.

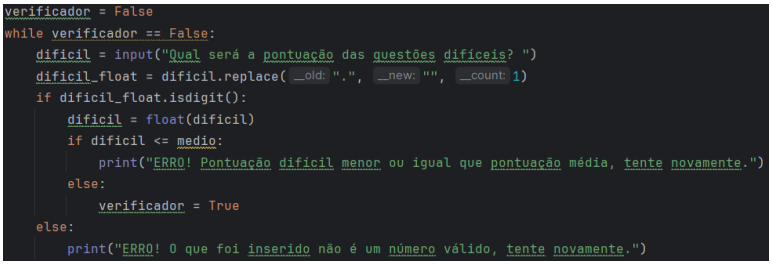


Figura 2. Validação das questões difíceis.

2.3. Cadastro de problemas resolvidos

Primeiramente é solicitado ao usuário quantas resoluções de problemas serão cadastradas, esse valor é validado por meio de *isdigit*, sendo aceito apenas valores inteiros. Segue a demonstração:

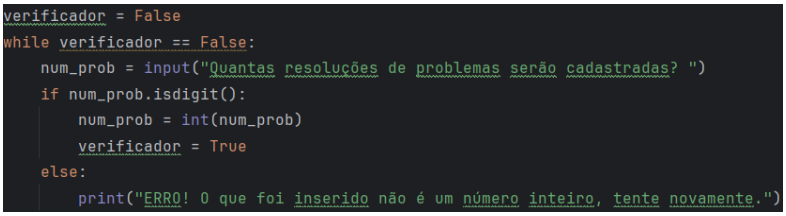
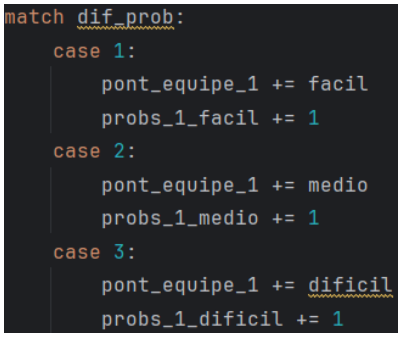


Figura 3. Validação do número de problemas.

Após o registro dessa informação, é iniciado um *loop for*, que itera de 1 até o número de problemas informado pelo usuário. Dentro do *loop*, a primeira coisa que é solicitado ao usuário é o número da equipe que resolveu o problema, com o usuário tendo que informar um número de 1 a 5, cada número representando uma equipe. Após o usuário inserir um valor válido para a equipe, o programa entra em uma condicional *match case* específica da equipe.

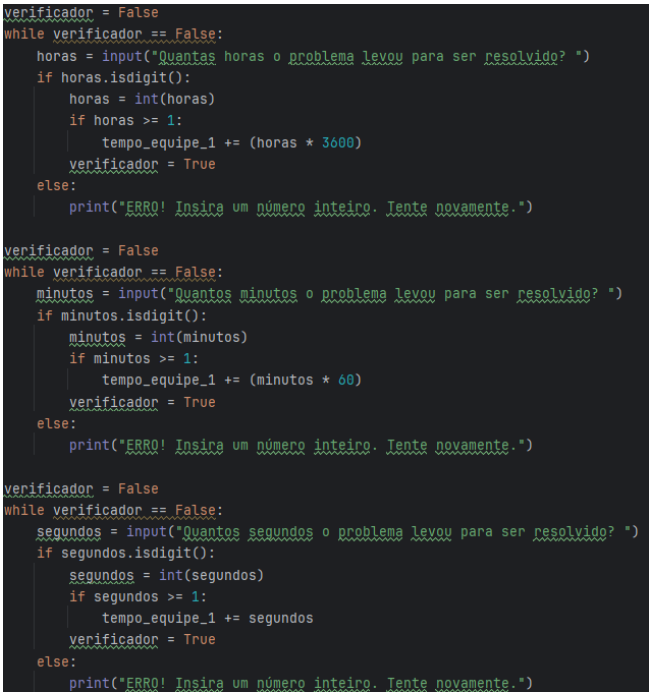
Dentro do *match case* de cada equipe, a primeira coisa que acontece é o incremento de uma variável contadora referente ao número de problemas resolvidos, assim contabilizando que a equipe em questão resolveu mais um problema. Após isso é solicitado ao usuário qual foi a dificuldade do problema, onde o usuário deve inserir um valor de 1 a 3, cada número referente a uma dificuldade entre fácil, médio e difícil. Após o usuário inserir um valor válido para a dificuldade da questão, o programa entra em outro *match case*, onde dependendo da dificuldade informada, é adicionado mais 1 ao contador de problemas fáceis, médios ou difíceis da equipe além de ser somada a pontuação da dificuldade à pontuação total da equipe. A figura exibida abaixo, retrata esse trecho do código.



```
match dif_prob:
    case 1:
        pont_equipe_1 += facil
        probs_1_facil += 1
    case 2:
        pont_equipe_1 += medio
        probs_1_medio += 1
    case 3:
        pont_equipe_1 += dificil
        probs_1_dificil += 1
```

Figura 5. Exemplo de *Match case* de dificuldade do problema.

Após isso é feito o cadastro do tempo em que o problema foi resolvido, primeiramente é solicitado ao usuário quantas horas a questão levou para ser resolvida, caso o valor seja maior que 1, ele é multiplicado por 3600, para transformar em segundos, e é somado à variável de tempo da equipe. Depois é solicitado os minutos, que passa pelo mesmo processo, sendo multiplicado por 60. E então são solicitados os segundos. A escolha de transformar todo o tempo em segundos dentro do programa, foi para tornar mais simples a comparação do tempo posteriormente. Na figura abaixo é possível ver a validação e conversão das variáveis de tempo.



```
verificador = False
while verificador == False:
    horas = input("Quantas horas o problema levou para ser resolvido? ")
    if horas.isdigit():
        horas = int(horas)
        if horas >= 1:
            tempo_equipe_1 += (horas * 3600)
        verificador = True
    else:
        print("ERRO! Insira um número inteiro. Tente novamente.")

verificador = False
while verificador == False:
    minutos = input("Quantos minutos o problema levou para ser resolvido? ")
    if minutos.isdigit():
        minutos = int(minutos)
        if minutos >= 1:
            tempo_equipe_1 += (minutos * 60)
        verificador = True
    else:
        print("ERRO! Insira um número inteiro. Tente novamente.")

verificador = False
while verificador == False:
    segundos = input("Quantos segundos o problema levou para ser resolvido? ")
    if segundos.isdigit():
        segundos = int(segundos)
        if segundos >= 1:
            tempo_equipe_1 += segundos
        verificador = True
    else:
        print("ERRO! Insira um número inteiro. Tente novamente.")
```

Figura 6. Exemplo de cadastro de tempo.

Após o cadastro da equipe que resolveu o problema, da dificuldade e do tempo, o programa retorna ao loop para cadastro de mais um problema resolvido ou encerra o loop se for o caso.

2.4. Exibição do relatório

Após o cadastro de todos os problemas, o programa entra na fase de exibição de relatório. A primeira coisa a ser exibida é o *ranking* das equipes, para organização do *ranking* foram utilizadas 5 novas variáveis, denominadas como “posX”, onde X é o

número de cada equipe. Cada equipe é comparada individualmente uma com a outra dentro de uma série de condicionais, onde são comparadas primeiramente as variáveis de pontuação, caso uma das duas equipes tenha maior pontuação, sua variável "posX" tem um incremento de 1, caso as pontuações sejam iguais, são comparados os números de problemas difíceis, onde caso uma equipe tenha resolvido mais problemas difíceis que a outra, sua variável tem o incremento de 1. Caso novamente haja um empate, então são comparados os tempos, se uma equipe tiver resolvido os problemas em menos tempo que a outra, sua variável tem um incremento de mais 1. Em caso de empate em todas as 3 situações, ambas as equipes têm um incremento de mais 1 em suas variáveis. Na demonstração a seguir é exibido o trecho da comparação entre a equipe 2 e a equipe 4.

```
if pont_equipe_2 > pont_equipe_4:
    pos2 += 1
elif pont_equipe_2 < pont_equipe_4:
    pos4 += 1
elif pont_equipe_2 == pont_equipe_4:
    if probs_2_dificil > probs_4_dificil:
        pos2 += 1
    elif probs_2_dificil < probs_4_dificil:
        pos4 += 1
    elif probs_2_dificil == probs_4_dificil:
        if tempo_equipe_2 < tempo_equipe_4:
            pos2 += 1
        elif tempo_equipe_2 > tempo_equipe_4:
            pos4 += 1
        elif tempo_equipe_2 == tempo_equipe_4:
            pos2 += 1
            pos4 += 1
```

Figura 7. Exemplo de comparação entre equipes.

Após todas as comparações entre as equipes, o código entra em uma série de condicionais que compara todas as variáveis "posX", onde as variáveis de maior valor ficam na frente no *ranking* e as menores ficam atrás, assim determinando o *ranking* com base na pontuação dentro dessa variável. Em caso de empate no valor dessa variável, a ordem do ranking é determinada pelo número da equipe, então por exemplo a equipe 1 ficaria na frente da equipe 4 em caso de empate. Na figura abaixo há um pequeno trecho, representando um possível *ranking* com base nas variáveis "posX".

```
if pos1 >= pos2 and pos1 >= pos3 and pos1 >= pos4 and pos1 >= pos5:
    primeiro_lugar = 1
    pont_primeiro = pont_equipe_1
    if pos2 >= pos3 and pos2 >= pos4 and pos2 >= pos5:
        segundo_lugar = 2
        pont_segundo = pont_equipe_2
        if pos3 >= pos4 and pos3 >= pos5:
            terceiro_lugar = 3
            pont_terceiro = pont_equipe_3
            if pos4 >= pos5:
                quarto_lugar = 4
                quinto_lugar = 5
                pont_quarto = pont_equipe_4
                pont_quinto = pont_equipe_5
            elif pos5 >= pos4:
                quarto_lugar = 5
                quinto_lugar = 4
                pont_quarto = pont_equipe_5
                pont_quinto = pont_equipe_4
```

Figura 8. Exemplo de possível ranking com base na comparação das variáveis.

A segunda coisa exibida no relatório final é a quantidade de problemas resolvidos por equipe, separados por dificuldade. Para isso foram utilizadas variáveis

acumuladoras, que ao decorrer do *loop* vão acumulando o número de problemas resolvidos, a exibição ocorre como no exemplo abaixo:

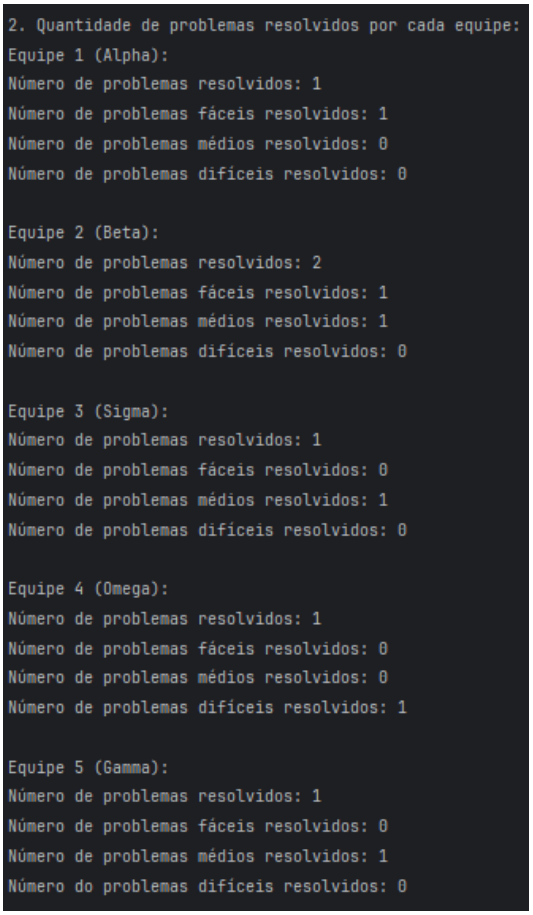


Figura 9. Exemplo de exibição da quantidade de problemas resolvidos.

A próxima exibição é da equipe vencedora e sua pontuação total, para isso é verificado por meio de uma condicional qual equipe ocupa o primeiro lugar, e com isso é exibido o nome e pontuação da equipe em questão. A figura a seguir representa como foi feita essa verificação:

```
print("3. Equipe vencedora e sua pontuação total:")
if primeiro_lugar == 1:
    print(f"A equipe vencedora foi a Equipe 1 ({equipe_1}) com {pont_equipe_1} pontos!")
elif primeiro_lugar == 2:
    print(f"A equipe vencedora foi a Equipe 2 ({equipe_2}) com {pont_equipe_2} pontos!")
elif primeiro_lugar == 3:
    print(f"A equipe vencedora foi a Equipe 3 ({equipe_3}) com {pont_equipe_3} pontos!")
elif primeiro_lugar == 4:
    print(f"A equipe vencedora foi a Equipe 4 ({equipe_4}) com {pont_equipe_4} pontos!")
elif primeiro_lugar == 5:
    print(f"A equipe vencedora foi a Equipe 5 ({equipe_5}) com {pont_equipe_5} pontos!")
print()
```

Figura 10. Verificação da equipe vencedora.

A quarta exibição é referente a equipe que resolveu o maior número de problemas difíceis. Para isso foi utilizado uma série de condicionais que verificam as variáveis acumuladoras de problemas difíceis resolvidos de cada equipe e no fim exibe a equipe que tem essa variável maior. A condicional verifica também possíveis empates nessa variável.

Para a exibição da média de pontos da equipe, primeiramente é verificado se o número de problemas resolvidos é maior que 1, para evitar o erro de dividir por 0, caso a resposta seja positiva é exibido então a pontuação total da equipe dividida pelo número de problemas resolvidos, assim resultando na média.

Para a última exibição, do tempo da equipe vencedora, o tempo de todas as equipes é novamente convertido de segundos para horas, minutos e segundos, a fim de ter uma melhor exibição para o usuário, e então assim como na exibição da equipe vencedora, é verificado qual equipe ficou em primeiro lugar, e então são exibidas as variáveis de tempo dessa equipe.

3. Resultados e Discussões

Basicamente o sistema é dividido em 3 partes principais: configuração inicial, cadastro de problemas e exibição do relatório final. O sistema inicia solicitando ao usuário a pontuação de cada uma das dificuldades de problemas, o nome das equipes e a quantidade de problemas que serão cadastrados. Em seguida o programa entra no loop de cadastros de problemas, onde será solicitado ao usuário o número da equipe, a dificuldade do problema e o tempo que a equipe levou para resolver o problema, o loop se repete a quantidade de vezes que o usuário inseriu em número de problemas cadastrados. Após a condição do loop ser atendida, é iniciada a exibição do relatório final, onde é exibido em ordem: 1) *Ranking* das equipes, 2) Quantidade de problemas resolvidos por cada equipe, separados por dificuldade, 3) Equipe vencedora e sua pontuação total, 4) Equipe que resolveu o maior número de problemas difíceis, 5) Média de pontos por equipe e 6) Tempo total gasto pela equipe vencedora. Como mencionado anteriormente, em caso de empate na pontuação, o critério de desempate para exibição no ranking é, em ordem: maior número de questões difíceis resolvidas e menor tempo total gasto. O programa funciona exclusivamente pelo terminal e precisa apenas do teclado. É necessário um interpretador *Python* instalado na máquina para execução do programa.

3.1. Testes

Para fins de demonstração, segue abaixo teste da validação da pontuação das questões, onde são demonstrados os valores considerados inválidos e os válidos.

```
!!! CONFIGURAÇÃO DE PONTUAÇÃO !!!
!!! PARA NÚMEROS DECIMAIS UTILIZAR SEPARADOR "." !!!
Qual será a pontuação das questões fáceis? -1
ERRO! O que foi inserido não é um número válido, tente novamente.
Qual será a pontuação das questões fáceis? a
ERRO! O que foi inserido não é um número válido, tente novamente.
Qual será a pontuação das questões fáceis?
ERRO! O que foi inserido não é um número válido, tente novamente.
Qual será a pontuação das questões fáceis? @
ERRO! O que foi inserido não é um número válido, tente novamente.
Qual será a pontuação das questões fáceis? 5
Qual será a pontuação das questões médias? 3
ERRO! Pontuação média menor ou igual que pontuação fácil, tente novamente.
Qual será a pontuação das questões médias? 7.5
Qual será a pontuação das questões difíceis? 7.3
ERRO! Pontuação difícil menor ou igual que pontuação média, tente novamente.
Qual será a pontuação das questões difíceis? 10.5
```

Figura 11. Teste de validação.

Segue demonstração de ranking abaixo:

```
1. Ranking das equipes:
Primeiro lugar: Equipe 4 com 10.0 pontos
Segundo lugar: Equipe 2 com 10.0 pontos
Terceiro lugar: Equipe 3 com 6.0 pontos
Quarto lugar: Equipe 5 com 6.0 pontos
Quinto lugar: Equipe 1 com 4.0 pontos
```

Figura 12. Teste de *ranking*.

No teste acima a pontuação das questões foi respectivamente 4, 6 e 10 e foram cadastrados 6 problemas:

- Um problema fácil para a equipe 1, com tempo de 30 minutos
- Um problema médio para a equipe 2, com tempo de 1 hora
- Um problema fácil para a equipe 2, com tempo de 30 minutos
- Um problema médio para a equipe 3, com tempo de 15 minutos
- Um problema difícil para a equipe 4, com tempo de 1 hora
- Um problema médio para a equipe 5, com tempo de 30 minutos

É possível perceber que, apesar da mesma pontuação entre a equipe 4 e 2, a equipe 4 ficou na frente pois acertou mais problemas difíceis. Um desempate também ocorre entre as equipes 3 e 5, onde a equipe 3 ficou na frente pois teve um tempo de conclusão menor.

4. Conclusão

Em resumo o programa foi bem-sucedido, todos os requisitos foram atendidos e não foram encontrados *bugs* ou erros que prejudiquem o funcionamento do programa. Todas as validações funcionam perfeitamente e o sistema de ranking e desempate consegue identificar a pontuação, número de problemas difíceis realizados e tempo das equipes e comparar de forma satisfatória todos esses dados e exibir corretamente.

Uma possível melhoria para o programa, seria a otimização das linhas de código, o código ficou com linhas em demasiado que poderiam ser simplificadas em blocos de código mais simples, assim deixando o código com mais legibilidade, sendo menos repetitivo.

5. Referências Bibliográficas

W3Schools, “Python String isdigit() Method”, disponível em: https://www.w3schools.com/python/ref_string_replace.asp; Acesso em 25 de Agosto de 2024

W3Schools, “Python String replace() Method”, disponível em: https://www.w3schools.com/python/ref_string_replace.asp; Acesso em 08 de Setembro de 2024