Busca Binária e Sequencial para solucionar o problema Comedores de Pipocas

Francisco Renan da Silva Menezes[1], Wladimir Araujo Tavares[1]

[1]Universidade Federal do Ceará (UFC) - Campus Quixadá
Av. José de Freitas Queiroz, 5003 – Cedro – Quixadá – Ceará 63902-580
renansilva@alu.ufc.br, wladimirufc@gmail.com

Abstract. *This paper aims to present the problem of the Brazilian Popcorn Marathon of the Programming Marathon of the SBC (Brazilian Computer Society) of the Subregional Brazil stage and its solution with binary search and sequential search*

Resumo*. Esse artigo tem a finalidade de apresentar o problema da Maratona Brasileira de Comedores de pipocas da Maratona de Programação da SBC(Sociedade Brasileira da Computação) da etapa Sub-regional Brasil e a sua solução com busca binária e busca sequencial.*

1. Introdução

Na Universidade Federal do Ceará, o Programa Bolsa de Iniciação Acadêmica (BIA) que tem como objetivo propiciar aos estudantes de cursos de Graduação Presenciais da UFC, especialmente dos semestres iniciais, condições financeiras para a sua permanência e desempenho acadêmico satisfatório mediante atuação em atividades de iniciação acadêmica. Uma dessas atividades é o Grupo de Estudos para a Maratona de Programação (GEMP) que tem como objetivo desenvolver uma preparação para os estudantes do Campus de Quixadá para a Maratona de Programação.

A Maratona de Programação é um evento da Sociedade Brasileira de Computação(SBC) que existe desde o ano de 1996. Desde sua primeira edição o evento vem sendo realizado com a intenção de atrair mais jovens para a computação e classificar competidores para as finais mundiais do concurso de programação, o *International Collegiatte Programming Contest*.

A Maratona se destina a estudantes de graduação e início de pós-graduação das áreas de computação e afins exigindo desses alunos capacidade de trabalho em equipe, criatividade e conhecimento de algoritmos clássicos. Os times são compostos por três estudantes que tentarão resolver dentro de 5 horas o maior número possível das 10 ou mais questões. Estes estudantes têm a sua disposição um único computador e material impresso para vencer a batalha contra o relógio e os problemas propostos.

Durante o semestre, os bolsistas do BIA que fazem parte do GEMP foram expostos a vários problemas que devem ser resolvidos computacionalmente vencendo os limites de tempo e memória estipulados pelo problema. O problema que será analisado é o problema Comedores de Pipocas apresentado na fase regional da Maratona de Programação de 2019.

Na seção 2, apresentaremos o problema Comedores de Pipoca. Na seção 3 em diante, apresentaremos a sua resolução com busca binária e sequencial, a distribuição dos elementos do vetor e como realizar o teste para saber se a distribuição está correta, também iremos discutir qual a melhor busca.

2. Apresentação do Problema

A Maratona Brasileira de Comedores de pipocas consiste em uma competição onde N sacos são colocados com uma quantidade arbitrária de pipoca. A competição é em equipe, cada uma composta por C competidores onde cada um só pode comer no máximo T pipocas por segundo. A comissão organizadora definiu que cada competidor deve comer uma sequência contígua de sacos de pipoca e todas as pipocas do mesmo saco devem ser comida por um único competidor. O objetivo da competição é comer todas as pipocas no menor tempo possível, dado que os C competidores podem comer em paralelo.

2.1 Entrada

A primeira linha contém três inteiros N, C e T ($1 \leq N \leq 10^5$, $1 \leq C \leq 10^5$ e $1 \leq T \leq 50$), representando a quantidade de sacos de pipoca, a quantidade de competidores de uma mesma equipe e quantidade máxima de pipoca por segundo que um competidor pode comer. A segunda linha conterá N inteiros P_i ($1 \leq P_i \leq 10^4$), sendo estes a quantidade de pipoca em cada um dos N sacos.

2.2 Saída

O programa deve produzir uma única linha com um inteiro representando a quantidade mínima de segundos necessário para a equipe comer todas as pipocas se ela se organizar da melhor maneira possível.

3. Resolução do Problema Com Busca Sequencial

Para solucionarmos problema podemos utilizar uma busca sequencial ou busca binária. Nesta seção vamos começar resolvendo com busca sequencial. Na busca sequencial deve ser feita a verificação da quantidade mínima de segundos S em que podemos comer as pipocas, a variável S começa igual a 1 e é incrementada +1 quando não satisfaz um tempo possível onde possa ser comido todos os sacos de pipocas considerando o tempo T de pipocas que podem ser comidas por segundo, caso contrario, retorna a variável.

A chamada da *função Busca Sequencial* deve ser feita passando o vetor de pipocas, a quantidade máxima de pipoca por segundo que um competidor pode comer e o número de competidores. Temos que declarar uma variável, onde deve ser testado como a quantidade mínima de segundos necessário para a equipe comer todo o saco de pipoca, para isso a variável a ser testada deve iniciar em 1, o menor tempo possível, e sempre incrementando +1 quando o valor não satisfazer essa condição. Para realizar o teste deve ser chamada uma outra função para testar a variável S.

3.1 Algoritmo

```
def busca_sequencial(vetor, velocidade, numeroCompetidores):
    S=1
    while ( True ):
        if ( testa(vetor, velocidade, numeroCompetidores, S ) ):
            return s
        else:
            S=S+1
```

A *função Busca sequencial* deve sempre chamar uma *função teste* para avaliar se o segundo S é suficiente para os competidores comerem todas as pipocas. Para isso nossa nova função deve receber o vetor de pipocas, a quantidade máxima de pipocas que devem ser comida em 1 segundo, o número de competidores e o segundo que a função deve avaliar retornando falso ou verdadeiro.

4. Resolução do Problema Com Busca Binária

Na Busca Binária definimos um limite entre limite inferior, que pode ser iniciado pelo menor tempo possível(1 segundo), até o limite superior, que passa a ser a soma da quantidade de pipoca dentro de todos os sacos dividido pela quantidade máxima de pipoca por segundo que um competidor pode comer. A busca deve ser feita a partir da variável meio que é igual a soma dos limites dividido por 2, onde quando o meio não satisfaz a condição, significa que a quantidade mínima de segundos necessário para a equipe comer todas as pipocas está para o seu lado direito entre as variáveis meio e limite superior, caso contrário, está a sua esquerda entre as variáveis limite inferior e meio. As três variáveis são atualizadas e a verificação é feita até que os limites estejam iguais.

A chamada da *função busca binária* deve ser feita passando o vetor de pipocas, a quantidade máxima de pipoca por segundo que um competidor pode comer e o número de competidores. O limite inferior pode ser definido em pelo menor tempo possível(1 segundo) e para calcular o limite superior pode ser chamado a *função limite superior* que pode ser feita da seguinte forma:

```
def limiteSuperior (vetor, quantidade)
    soma=0
    for x in vetor:
        soma = soma + 1
    if ( soma %t == 0 ):
        return soma/quantidade
    else:
        return soma/quantidade +1
```

Não podemos trabalhar com números decimais, por isso a nossa divisão pela quantidade deve dar um número inteiro e resto 0, caso contrário devemos sempre arredondar esse número para cima.

Com os limites inferior e superior definidos, a busca binária deve iniciar um laço com a condição do Limite Inferior ser menor que o Limite Superior, após isso uma variável que define o meio dos limites deve ser inicializada com a soma dos limites dividido por 2. A função teste deve ser sempre chamada para testar se a variável que está definindo o meio dos limites é uma quantidade de segundo suficientes para que a equipe possa comer toda a quantidade de pipocas, caso retorne verdadeiro significa que a quantidade mínima de segundos necessário para a equipe comer todas as pipocas é exatamente o valor do meio ou está para a sua esquerda entre o meio e o limite inferior. Caso contrário, significa que a quantidade mínima está para a direita entre o meio e limite superior.

4.1 Algoritmo

```
def buscaBinaria (vetor, quantidade, numeroCompetidores):
    inferior=0
    superior=limiteSuperior(vetor, quantidade)

    while (inferior < superior):
        meio=(inferior < superior)/2
        if ( teste( vetor, quantidade, numeroCompetidores, meio ) ):
```

```
                superior=meio
        else:
                inferior = meio + 1
    return inferior
```

Se a função teste retorna verdadeiro, o limite superior deve receber o valor do meio, caso contrário, se a função teste retorna falso o limite inferior deve receber o meio+1. Assim, os limites são sempre atualizados e o laço é sempre executado até que os ponteiros limite inferior e limite superior se encontrem e a função de busca binária possa retornar o valor de limite inferior ou limite superior que será a quantidade mínima de segundos necessário para a equipe comer todas as pipocas se ela se organizar da melhor maneira possível.

5. Função Teste

A função teste é chamada passando o vetor, a quantidade máxima de pipocas que podem ser comidas em 1 segundo, o número de competidores e o valor a ser testado. A função distribui os sacos de pipocas aos competidores sempre respeitando o fato de que os competidores só podem comer em uma sequência contígua dentro do limite máximo de pipocas por segundo que podem ser comidos. O retorno será falso caso não consiga comer todas as pipocas e verdadeiro caso coma todas as pipocas

5.1 Algoritmo

```
def teste(vetor, quantidade, numeroCompetidores, valor):
    pipocas = valor * quantidade

    contCompetidores = 0
    sacos = 0
    tamanho = len(vetor)

    while (  contCompetidores < numeroCompetidores ):
        comeu = 0
        while ( sacos < tamanho and comeu + vetor[sacos] <= pipocas ):
            comeu = comeu + vetor[sacos]
            sacos = sacos + 1
        contCompetidores = contCompetidores + 1
    if ( sacos  == tamanho ):
        return True
    else:
        return False
```

É declarado uma variável pipoca para receber o valor a ser testado multiplicado pela quantidade máxima que se pode comer de pipocas por segundo. Para satisfazer o nosso limite o número de pipocas a serem comidos não podem ultrapassar o número da variável pipocas. Para distribuir as pipocas deve ser usado dois laços, onde o primeiro percorre o número de competidores, passando para o próximo só quando o competidor comer a maior quantidade possível. O segundo laço serve para garantir que o competidor coma a maior quantidade possível de uma maneira contígua. Para isso existe a condição de que o número do saco seja menor que o tamanho do vetor dos sacos de pipocas e que a quantidade que competidor já comeu somado ao próximo saco de

pipoca a ser comido seja menor que a variável pipocas, dessa forma o competidor pode comer o saco de pipoca e passar para o próximo saco, caso não satisfaça essa condição o algoritmo sair do laço e incrementa mais um na variável contCompetidores para que o laço possa passar para o próximo competidor.

Quando contCompetidores for igual a numeroCompetidores ambos os laços não precisam mais serem executados, mas sim a condição para avaliar se foi possível comer toda a quantidade de pipocas ou não. O if verifica se a variável sacos é igual ao tamanho do vetor, ou seja, se os laços conseguiram ir até o final. Se o algoritmo entrar no if é retornado verdadeiro, caso contrário entra no else e retorna falso.

6. Conclusão e Resultados

Utilizando busca em sequência verificou-se que assumimos o risco de percorrer todo o nosso limite, a busca binária mostrou que pode ser bem mais eficiente por nunca ter que percorrer todo o limite pois sempre realiza uma busca seguindo o paradigma de divisão e conquista. A busca binária considera que temos um vetor ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado com o elemento do meio do vetor diminuindo assim o número de comparações a serem feitas resultando em um menor tempo de execução. A busca em sequência só ganha em alguns casos específicos, como por exemplo, quando o elemento que estamos procurando for o primeiro da sequência a ser avaliada.

Referências

Maratona de Programação. http://maratona.ime.usp.br/. Acesso 2019-10-22.

Busca em sequência e binária. https://panda.ime.usp.br/aulasPython/static/aulasPython/aula23.html /. Acesso 2019-10-22