Nome: Wayne Nascimento Souza
Matrícula: 0016183

Paradigma da Programação Dinâmica

A Programação Dinâmica é uma metodologia de desenvolvimento que divide problemas complexos em questões mais simples e os resolve repetidamente. Segundo Ferreira e Matheus (2021) a principal inovação dessa abordagem é o uso de memória para armazenar os subresultados, que são reutilizados ao longo do cálculo da solução geral em diferentes momentos.

Conforme o site Geeksforgeeks (2022), a subestrutura ótima e a sobreposição de soluçõessão as bases desta abordagem. A subestrutura ótima diz que a melhor solução para um problema é composta pelas melhores soluções para seus subproblemas dependentes. A sobreposição de soluções mostra que a resolução de subproblemas que surgem duas ou mais vezes é a solução ideal.

De acordo com Feofiloff (2020) a Programação Dinâmica oferece uma maneira eficaz de resolver problemas, usando uma tabela que armazena as soluções das várias subinstâncias. O tamanho da tabela geralmente corresponde ao tempo gasto pelo algoritmo.

O problema deve ter uma estrutura recursiva, com soluções de subinstâncias para cada instância, para que o paradigma em questão possa ser usado, afirma Kleina (2021). A Programação Dinâmica também é útil quando subproblemas são repetidos, pois as soluções calculadas podem ser armazenadas e reutilizadas, evitando cálculos inúteis. Segundo o site Geeksforgeeks (2023), para acelerar os programas de computador, a memoização é uma estratégia que elimina computações repetitivas e evita chamadas repetidas de funções que processam a mesma entrada. Como resultado, a memoização e a programação dinâmica são ferramentas poderosas para lidar com problemas complexos e otimizar o desempenho de algoritmos, especialmente quando vários problemas se sobrepõem.

Geralmente tem a ver com encontrar o máximo e o mínimo intervalo da consulta algorítmica, declara BasuMallick (2022), e alguns exemplos de problema clássicos que podem ser solucionados usando a Programação Dinâmica segundo o site Medium (2018) são: Subsequência comum mais curta, subsequência comum mais longa, multiplicação de cadeias de matrizes, problema binário da mochila entre outros.

Modelagem da solução

Para realizar a modelagem de uma possível solução para o problema, primeiramente foi definido com base no enunciado de como seriam a entrada e a saída. A entrada é definida pelo número de etapas na linha de produção, dois números inteiros referentes ao tempo gasto para a entrada em cada uma das linhas de produção, duas listas com os tempos gastos em cada linha, duas listas com valores para realizar a transição entre as duas linhas e dois números inteiros referentes ao tempo de saída de cada linha de produção. Com isso, foi possível determinar o formato dos candidatos e desenvolver um algoritmo baseado no paradigma da Programação Dinâmica para solucionar o problema da linha de montagem (2769) presente na plataforma Beecrowd.

A aplicação do paradigma da Programação Dinâmica como parte da solução possui as seguintes etapas: identificação do problema, definição da entrada e saída, identificação dos subproblemas, elaboração da recursividade e o cálculo do tempo mínimo de saída.

Na primeira etapa, é necessário identificar se o problema pode ser resolvido usando este paradigma. Como o objetivo deste problema é a minimização do tempo total de produção em duas linhas de montagem, então é possível utilizar.

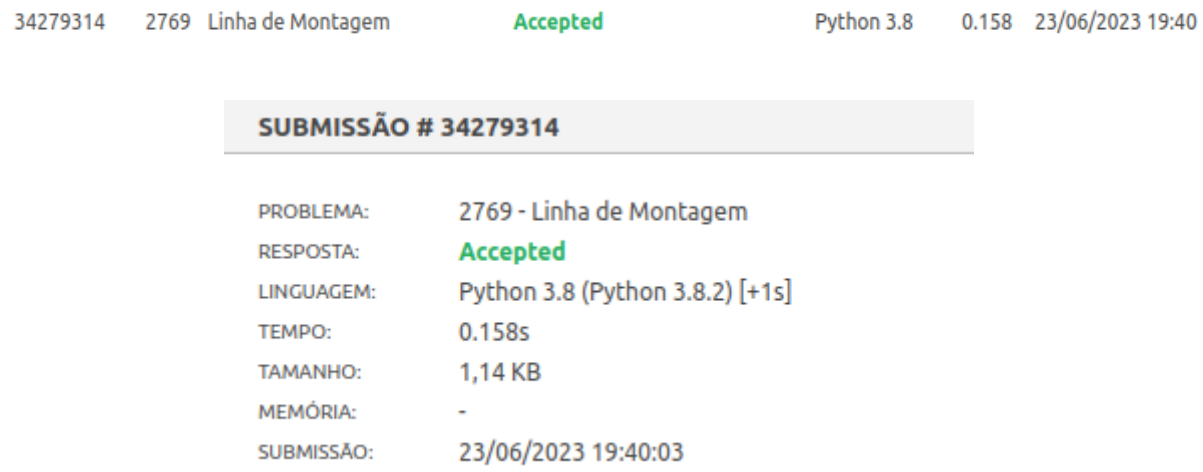
A etapa de definição da entrada e saída, para solucionar o problema é utilizada a leitura de entrada para obter o número de etapas da linha de produção (N), dois números inteiros (e1 e e2) referentes ao tempo gasto para a entrada em cada uma das linhas de produção, duas listas (a1 e a2) com os tempos gastos em cada linha, duas listas com valores (t1 e t2) para realizar a transição entre as duas linhas e dois números inteiros (x1 e x2) referentes ao tempo de saída de cada linha de produção. Já a saída é um número que representa o menor tempo total de produção.

A identificação dos subproblemas é a terceira etapa, onde é analisada a necessidade de calcular os tempos mínimos para cada estágio em ambas as linhas de produção.

A etapa da elaboração da recursividade é dada pela formulação baseada nos tempos mínimos para cada estágio, considerando os tempos anteriores. A recursividade é definida a partir de dois cenários: na linha de montagem 1 e na linha montagem 2, mas com a mesma lógica. Como são usadas duas listas (T1 e T2) para representar a estrutura de dados para armaezenar as soluções dos subproblemas, no primeiro estágio é armazenado em cada uma a soma entre o tempo gasto de entrada e o primeiro valor da lista com os tempos gastos e, posteriormente usando uma estrutura de repetição ocorre o cálculo para encontrar o tempo mínimo considerando o tempo anteiror daquela linha ou o tempo anterior na outra linha somado com o tempo de transição.

E por fim, a etapa do cálculo do tempo mínimo de saída onde é calculado o mínimo entre a

soma do tempo no último estágio com o tempo de saída da linha de montagem 1 e a soma do tempo no último estágio com o tempo de saída da linha de montagem 2.
Com base nessa modelagem, foi possível resolver o problema 2769 do Beecrowd. Logo abaixo estão algumas evidências que o código oriundo desta modelagem foi aceito pela bateria de testes e no tempo definido pela plataforma.



Análise e custo do algoritmo

Para analisar o custo do algoritmo será dividido em trechos de códigos, apontando a linha inicial e a final e consequentemente o custo associado.

- 06-07: é a inicialização das listas T1 e T2 e tem custo baseado em N (N + N).
- 10-11: atribuição de valor para a primeira posição da lista e tem custo constante (1).
- 14-16: dentro de uma estrutura de repetição são realizadas operações de comparação e atribuição de valores mínimos (N + 1).
- 19-21: atribuição e retorno, custo constante (1).
- 25-33: leitura da entrada que é proporcional ao número de linhas (N).

A soma de todos esses custos resulta na equação recorrência abaixo:
$T(n) = O(n) + O(n) + O(1) + O(n) + O(1) + O(1) + O(n)$
Como o n é a complexidade assintótica dominante dentre estas, podemos usar a equação simplificada $T(n) = O(n)$.
Não é possível aplicar o Teorema Mestre pois não temos a equação de recorrência na forma $T(n) = aT(\frac{n}{b}) + f(n)$.

Logo vamos considerar que o custo deste algoritmo é $O(n)$.

Referências

MEDIUM. Top 10 Dynamic programming problems for interviews, 2018. Disponível em: https://medium.com/techie-delight/top-10-dynamic-programming-problems-5da486eeb360. Acesso em: 25 de jun. de 2023.

FERREIRA, Ennio; MATHEUS, Victor. Programação Dinâmica, 2021. Disponível em: https://lamfo.unb.br/wp-content/uploads/2021/03/Programa%C3%A7%C3%A3o-Din%C3%A2mica.pdf. Acesso em: 25 de jun. de 2023.

FEOFILOFF, Paulo. Programação Dinâmica, 2020. Disponível em: https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamic-programming.html. Acesso em: 25 de jun. de 2023.

KLEINA, Mariana. Programação Dinâmica, 2021. Disponível em: https://docs.ufpr.br/~marianakleina/PD-aula2-exemplos.pdf. Acesso em: 25 de jun. de 2023.

GEEKSFORGEEKS. Overlapping Subproblems Property in Dynamic Programming, 2022. Disponível em: https://www.geeksforgeeks.org/overlapping-subproblems-property-in-dynamic-programming-dp-1. Acesso em: 25 de jun. de 2023.

GEEKSFORGEEKS. Optimal Substructure Property in Dynamic Programming, 2022. Disponível em: https://www.geeksforgeeks.org/optimal-substructure-property-in-dynamic-programming-dp-2. Acesso em: 25 de jun. de 2023.

GEEKSFORGEEKS. What is memoization? A Complete tutorial, 2023. Disponível em: https://www.geeksforgeeks.org/what-is-memoization-a-complete-tutorial. Acesso em: 25 de jun. de 2023.

BASUMALLICK, Chiradeep. What is Dynamic Programming? Working, Algorithms, and Examples, 2022. Disponível em:

https://www.spiceworks.com/tech/devops/articles/what-is-dynamic-programming/. Acesso em: 25 de jun. de 2023.