Estudo Teórico e Experimental de Abordagens Recursivas e Dinâmicas Aplicadas ao Problema do Corte de Hastes

Luciano Sousa Barbosa[1], **Pedro Henrique Silva Rodrigues**[1], **Tiago Lima de Moura**[1]

[1]Departamento de Sistemas de Informação – Universidade Federal do Piauí (UFPI)
64.600-000 – Picos – PI – Brasil

{luciano.barbosa,pedro.rodrigues.pr,tiago.lima}@ufpi.edu.br

Resumo. *Este trabalho apresenta uma análise comparativa das abordagens de programação recursiva e programação dinâmica aplicadas ao problema do corte de hastes, considerando aspectos teóricos e experimentais. As soluções foram implementadas na linguagem C e avaliadas sob diferentes tamanhos da barra, permitindo observar o impacto do crescimento da instância sobre o desempenho computacional. Os resultados obtidos possibilitam avaliar o comportamento prático de cada abordagem e relacioná-lo com suas respectivas características teóricas.*

1. Introdução

A análise de algoritmos é essencial para compreender o impacto das técnicas de projeto no desempenho computacional de soluções para problemas de otimização [Yang 2011]. Em especial, algoritmos baseados em recursão e programação dinâmica apresentam diferenças significativas quanto ao custo computacional, apesar de partirem frequentemente de formulações conceituais semelhantes [Kabanov and Vene 2006]. A comparação entre essas abordagens permite avaliar como decisões de projeto influenciam diretamente a eficiência dos algoritmos [Cormen et al. 2022].

A programação dinâmica fundamenta-se no Princípio da Otimalidade de Bellman, segundo o qual uma solução ótima global é composta por soluções ótimas de seus subproblemas [Bellman and Lee 1984]. Esse teorema fornece a base teórica para a decomposição eficiente de problemas complexos, evitando a recomputação de resultados intermediários [Bellman and Dreyfus 2015]. Em contraste, abordagens recursivas ingênuas tendem a recalcular subproblemas repetidamente, o que pode resultar em crescimento exponencial do tempo de execução [Kleinberg and Tardos 2006].

O problema do corte de hastes destaca-se como um exemplo clássico para ilustrar essas diferenças de desempenho. O objetivo do problema é determinar a forma mais lucrativa de dividir uma haste de comprimento fixo, a partir de uma tabela de preços previamente definida [Dasgupta et al. 2007]. Embora simples em sua formulação, o problema evidencia claramente a sobreposição de subproblemas e a presença de subestrutura ótima.

A aplicação de programação dinâmica ao problema do corte de hastes permite explorar essas propriedades de forma sistemática e eficiente [Chu and Antonio 1999]. Ao armazenar soluções parciais, o algoritmo reduz significativamente o número de operações necessárias em comparação à abordagem puramente recursiva [Tanır et al. 2019]. Dessa forma, o problema torna-se um cenário adequado para avaliar empiricamente os ganhos proporcionados por essa técnica.

Neste contexto, o presente trabalho visa analisar e comparar soluções recursivas e baseadas em programação dinâmica para o problema do corte de hastes. A comparação é realizada a partir da implementação das diferentes abordagens e da avaliação de seu desempenho computacional. Por fim, busca-se relacionar os resultados empíricos com as previsões teóricas, contribuindo para uma compreensão mais sólida sobre a aplicação desses algoritmos em diferentes cenários computacionais.

2. Referencial Teórico

Nesta seção, são apresentados os fundamentos teóricos relacionados às técnicas de programação recursiva e programação dinâmica aplicadas ao problema do corte de hastes, abordando seus princípios de funcionamento, características estruturais e particularidades operacionais. São discutidos os aspectos conceituais que orientam o comportamento dessas abordagens. Por fim, são introduzidos conceitos práticos que permitem relacionar a fundamentação teórica aos resultados experimentais apresentados nas seções subsequentes.

2.1. Programação Dinâmica

A programação dinâmica é uma técnica de projeto de algoritmos utilizada para resolver problemas de otimização que apresentam subestrutura ótima e sobreposição de subproblemas. Essa abordagem baseia-se no Princípio da Otimalidade de Bellman, segundo o qual uma solução ótima global pode ser construída a partir de soluções ótimas de instâncias menores do mesmo problema. Dessa forma, a programação dinâmica busca reduzir o custo computacional por meio do armazenamento e reaproveitamento de resultados intermediários [Bellman and Lee 1984].

A aplicação da programação dinâmica envolve a decomposição sistemática do problema original em subproblemas menores, cujas soluções são armazenadas para evitar recomputações desnecessárias. Essa estratégia pode ser implementada por meio de abordagens bottom-up ou top-down com memorização, dependendo da estrutura do problema.

No contexto do problema do corte de hastes, a programação dinâmica explora a relação entre o lucro máximo de uma haste de comprimento n e os lucros ótimos obtidos para comprimentos menores. A solução é construída iterativamente a partir dos menores comprimentos até atingir o tamanho desejado, garantindo a utilização das melhores soluções parciais. A figura 1 abaixo é apresentada a implementação do problema do corte de hastes em pseudocódigo, evidenciando a construção incremental da solução.

Listing 1. Implementação em Pseudocódigo do Corte de Hastes Usando Programação Dinâmica.

```
FUNCAO CorteDeHastesDP(n, precos[], r[])
    // Caso base: haste tamanho 0 tem lucro 0
    r[0] <- 0

    // Para cada tamanho de haste de 1 ate n (bottom-up)
    PARA j DE 1 ATE n FAZER
        // Inicializa com o menor valor possivel
        lucro <- -INFINITO

        // Testa todos os cortes possiveis para haste de tamanho j
        // i = tamanho do primeiro pedaco cortado
        PARA i DE 1 ATE j FAZER
```

```
            // Lucro = preco do pedaco i + melhor lucro do restante (j-i)
            valor <- precos[i] + r[j - i]

            // Atualiza melhor lucro encontrado para tamanho j
            SE valor > lucro ENTAO
                lucro <- valor
            FIM SE
        FIM PARA

        // Armazena solucao otima para tamanho j na tabela r
        r[j] <- lucro
    FIM PARA

    // Retorna solucao otima para o tamanho original n
    RETORNE r[n]
FIM FUNCAO
```

A eficiência da programação dinâmica no problema do corte de hastes decorre da eliminação da redundância presente em abordagens recursivas ingênuas. Ao armazenar os resultados de cada subproblema, o algoritmo evita a explosão combinatória típica de soluções sem reaproveitamento. Esse comportamento teórico reflete-se diretamente na redução do tempo de execução observada nos experimentos.

Além de sua eficiência computacional, a programação dinâmica oferece uma estrutura conceitual clara para a modelagem de problemas de otimização. A definição explícita dos estados e das transições facilita a análise do algoritmo e sua implementação prática.

2.2. Abordagem Recursiva

A programação recursiva é uma técnica de resolução de problemas na qual uma função é definida em termos de chamadas a si mesma, reduzindo progressivamente o tamanho da instância original. Essa abordagem é especialmente adequada para problemas que podem ser naturalmente decompostos em subproblemas semelhantes, como aqueles definidos por relações de recorrência. Contudo, a ausência de mecanismos de reaproveitamento de resultados pode impactar significativamente o desempenho computacional [Aho and Hopcroft 1974].

No problema do corte de hastes, a formulação recursiva baseia-se na ideia de testar todas as possibilidades de corte possíveis para uma barra de comprimento n. A solução ótima é obtida comparando os lucros resultantes de cada possível divisão da haste, considerando recursivamente os comprimentos remanescentes. A figura 2 apresenta a implementação do problema do corte de hastes em pseudocódigo, ilustrando essa estratégia de decomposição recursiva.

Listing 2. Implementação em Pseudocódigo do Corte de Hastes Usando Abordagem Recursiva.

```
FUNCAO CorteDeHastesRec(n, precos[])
    // Caso base: haste de tamanho 0
    SE n == 0 ENTAO
        RETORNE 0
    FIM SE

    // Inicializa lucro com valor minimo
    lucro <- -INFINITO
```

```
    // Para todos os possiveis tamanhos do primeiro corte
    PARA i DE 1 ATE n FAZER
        // Calcula lucro cortando pedaco de tamanho i
        // e resolvendo RECURSIVAMENTE o resto (n-i)
        valor <- precos[i] + CorteDeHastesRec(n - i, precos)

        // Mantem o maior lucro encontrado
        SE valor > lucro ENTAO
            lucro <- valor
        FIM SE
    FIM PARA

    // Retorna melhor lucro para haste de tamanho n
    RETORNE lucro
FIM FUNCAO
```

Embora conceitualmente simples, a abordagem recursiva apresenta uma grande sobreposição de subproblemas, uma vez que os mesmos comprimentos são avaliados repetidas vezes. Essa característica resulta em um crescimento exponencial do número de chamadas recursivas à medida que o tamanho da barra aumenta. Como consequência, o tempo de execução torna-se rapidamente impraticável para instâncias maiores do problema.

Do ponto de vista do uso de memória, a programação recursiva demanda espaço proporcional à profundidade da pilha de chamadas. No pior caso, a profundidade da recursão é linear em relação ao tamanho da barra, o que pode levar a limitações práticas associadas à capacidade da pilha. Esse aspecto torna a abordagem sensível a restrições de memória, especialmente em ambientes com recursos limitados.

Apesar dessas limitações, a solução recursiva desempenha um papel fundamental na compreensão teórica do problema do corte de hastes. Ela fornece uma formulação direta da relação de recorrência que descreve o problema, servindo como base conceitual para o desenvolvimento de soluções mais eficientes.

3. Metodologia

Nesta seção, são descritos o ambiente de execução empregado e os aspectos técnicos relacionados à implementação dos algoritmos, como bibliotecas utilizadas e características do hardware e do sistema operacional, de modo a assegurar a reprodutibilidade dos experimentos. Também são apresentados os mecanismos adotados para a coleta e o tratamento dos dados experimentais, bem como as ferramentas auxiliares empregadas na medição e análise de desempenho. Por fim, são explicitados os critérios e procedimentos experimentais utilizados para comparar o desempenho e o comportamento funcional dos algoritmos de ordenação sob diferentes configurações de entrada.

3.1. Especificações Técnicas

Esta subseção apresenta a configuração da máquina utilizada para a execução dos testes, fornecendo informações essenciais para a reprodutibilidade dos resultados. Essas informações permitem contextualizar os resultados obtidos, evidenciando as condições sob as quais os experimentos foram realizados. Observe a tabela 1.

Tabela 1. Especificações Técnicas da Máquina Utilizada nos Testes

Especificação	Descrição
Processador	Intel Core i5-12450H 12º Gen (2.00 GHz) (8 núcleos, 12 threads, 12 MB cache)
Memória RAM	16,0 GB @ 3200 MHz (utilizável: 15,7 GB)
SO	Windows 11 Home Single Language (Executado no Ubuntu 24.04.3 LTS via WSL2)

3.2. Especificações Práticas

Na condução dos experimentos, foram desenvolvidas implementações específicas em linguagem C para a resolução do problema do corte de hastes utilizando programação recursiva e programação dinâmica. Em ambas as abordagens, a geração da instância do problema baseou-se em uma função determinística de preços, permitindo controlar e padronizar os valores associados a cada comprimento possível da barra. Essa padronização possibilitou isolar o impacto da técnica de projeto de algoritmos sobre o desempenho computacional observado.

Os cenários experimentais consideraram diferentes tamanhos da barra, especificamente comprimentos iguais a 10, 20 e 30 unidades, respeitando as limitações práticas da abordagem recursiva quanto ao tempo de execução. Cada tamanho foi avaliado de forma independente nas duas abordagens, assegurando condições equivalentes de entrada para a comparação. Essa variação sistemática do tamanho do problema permitiu analisar o comportamento dos algoritmos à medida que a instância cresce.

Para a medição do tempo de execução, foi utilizada a função *clock_gettime* com o relógio *CLOCK_MONOTONIC*, garantindo maior precisão e independência de ajustes externos do sistema. Cada experimento foi executado 11 vezes consecutivas, sendo a primeira execução descartada como *warm-up*, a fim de reduzir interferências associadas à inicialização do ambiente. O tempo médio das execuções restantes foi então calculado em milissegundos e utilizado como métrica principal de desempenho.

Além do tempo de execução, foram coletadas métricas adicionais relacionadas ao uso de memória e ao comportamento interno dos algoritmos. Na versão recursiva, foi contabilizado o número total de chamadas recursivas, permitindo estimar o consumo de memória na pilha a partir de uma estimativa média do tamanho do *frame* de ativação. Já na versão baseada em programação dinâmica, o consumo de memória foi determinado pelo tamanho do vetor auxiliar utilizado para armazenar os resultados dos subproblemas.

Os resultados das 10 execuções válidas foram armazenados em arquivos CSV, os quais serviram de base para a geração dos gráficos por meio de um *script* em Python utilizando as bibliotecas *pandas* e *matplotlib.pyplot*.

4. Resultados e Discussão

Nesta seção, são apresentados e analisados os dados obtidos nos testes de desempenho das abordagens baseadas em programação recursiva e programação dinâmica aplicadas ao problema do corte de hastes, considerando diferentes cenários definidos pelo tamanho da barra. Além disso, os resultados são discutidos à luz dos fundamentos teóricos de

cada técnica, buscando relacionar o comportamento observado nos experimentos com as características esperadas em termos de eficiência computacional.

4.1. Tempo Médio de Execução

Os dados exibidos no gráfico 1 evidenciam uma diferença de desempenho abismal entre os dois paradigmas. Para apenas 30 hastes, a solução recursiva consome mais de 4 segundos, enquanto a dinâmica finaliza em 0.002 ms. Essa disparidade cresce de forma não linear com o tamanho da entrada, indicando complexidades assintóticas radicalmente distintas.

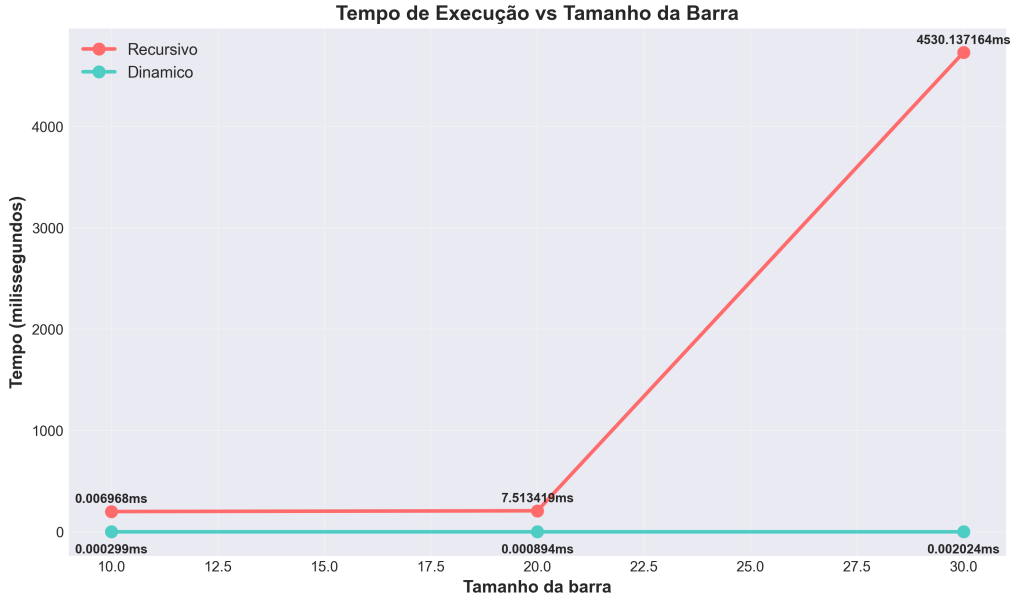

Figura 1. Comparação de Tempo Médio de Execução: Abordagem Dinâmica x Recursiva

Este comportamento é uma confirmação experimental direta da teoria da complexidade de algoritmos. A solução recursiva pura apresenta crescimento exponencial típico, aproximando-se de $O(2^n)$, devido à recalculação massiva de subproblemas idênticos. Em contraste, o tempo da programação dinâmica escala de forma polinomial, condizente com sua complexidade $O(n^2)$ bem estabelecida.

A explicação fundamental reside no uso da memorização pela abordagem dinâmica. Ela armazena soluções parciais em uma tabela, eliminando toda a redundância computacional inerente à árvore de recursão. Cada subproblema é resolvido uma única vez, transformando a explosão exponencial em um custo quadrático gerenciável.

Os resultados práticos ilustram perfeitamente o *trade-off* clássico entre tempo e espaço na análise de algoritmos. A programação dinâmica introduz um custo de memória adicional $O(n)$ para armazenar a tabela. Esse pequeno *overhead* é insignificante perante o ganho de tempo de várias ordens de magnitude, tornando-se uma troca absolutamente vantajosa.

Portanto, conclui-se que a recursão é intratável mesmo para instâncias moderadas deste problema. A programação dinâmica se consolida como a única alternativa viável,

demonstrando a importância crucial de se identificar e explorar a subestrutura ótima e a sobreposição de subproblemas no projeto de algoritmos eficientes.

4.2. Estimativa de Memória Utilizada

O gráfico 2 traz o consumo de memória, apresentando uma relação linear crescente para ambos os algoritmos, porém com inclinações distintas. A programação dinâmica partiu de 44 bytes (n=10) e atingiu 124 bytes (n=30), mostrando um crescimento suave. Em contraste, a recursão iniciou em 480 bytes e escalou para 1440 bytes no mesmo intervalo, evidenciando uma taxa de crescimento muito mais acentuada.

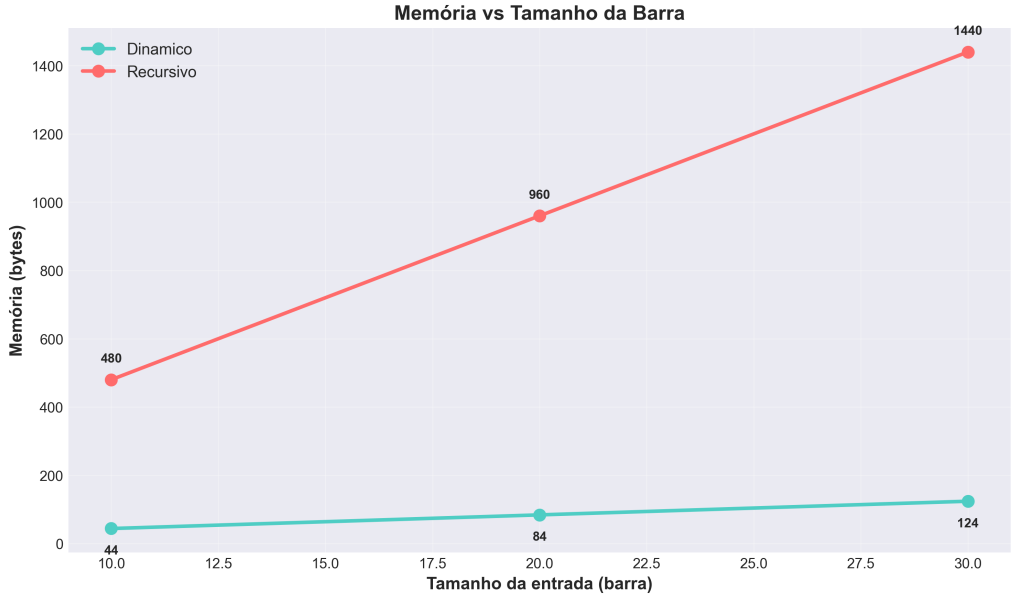

Figura 2. Comparação de Memória Utilizada: Abordagem Dinâmica x Recursiva

Este padrão é uma manifestação direta das estruturas de dados subjacentes a cada implementação. A solução dinâmica aloca uma única estrutura de tamanho $O(n)$ para armazenar os resultados intermediários. A solução recursiva, por sua vez, mantém uma pilha de chamadas ativas proporcional à profundidade da recursão, que também é $O(n)$ no pior caso.

A diferença nas constantes multiplicativas, no entanto, é significativa e explicável. A memória da recursão é aproximadamente uma ordem de grandeza maior devido ao custo fixo de cada *stack frame*, que armazena parâmetros, endereços de retorno e variáveis locais. Cada chamada recursiva gera um novo *frame*, acumulando-se rapidamente.

Portanto, observa-se um *trade-off* clássico entre os paradigmas. A recursão pura demanda menos memória conceitual, mas sua implementação prática incorre em alto *overhead* da pilha. A programação dinâmica, ao preencher uma tabela iterativamente, troca um aumento pequeno e previsível de memória de trabalho por um ganho monumental em tempo de execução.

Conclui-se que, para este problema, o uso de memória não é o fator limitante para a abordagem dinâmica. Seu crescimento linear modesto é um custo amplamente

justificado pela eliminação da explosão exponencial no tempo, consolidando-a como a escolha algorítmica eficiente e escalável.

5. Conclusão

O presente trabalho demonstrou, de forma empírica e contundente, a superioridade da programação dinâmica sobre a recursão para resolver o problema do corte de hastes. Os resultados revelaram uma diferença de desempenho que evolui de ordens de grandeza para uma disparidade intransponível conforme a entrada cresce. Esta conclusão valida o princípio teórico de que a sobreposição de subproblemas exige o uso de memorização para se obter eficiência prática.

A análise conjunta de tempo e memória comprovou o *trade-off* fundamental envolvido nessa escolha algorítmica. A abordagem dinâmica introduz um custo de espaço linear e previsível para armazenar soluções parciais em uma tabela. Este pequeno *overhead* é a chave para transformar uma complexidade temporal exponencial em uma complexidade polinomial, representando uma troca extremamente vantajosa.

Portanto, os resultados consolidam a programação dinâmica como o paradigma obrigatório para instâncias não triviais deste problema. A recursão, apesar de ser uma representação mental direta da equação de recorrência, mostrou-se completamente impraticável. Este estudo serve como um caso paradigmático da importância crucial de se analisar a complexidade assintótica e de se aplicar técnicas de projeto de algoritmos para dominar a explosão combinatória.

Referências

Aho, A. V. and Hopcroft, J. E. (1974). *The design and analysis of computer algorithms*. Pearson Education India.

Bellman, R. and Lee, E. (1984). History and development of dynamic programming. *IEEE Control Systems Magazine*, 4(4):24–28.

Bellman, R. E. and Dreyfus, S. E. (2015). *Applied dynamic programming*. Princeton university press.

Chu, C. and Antonio, J. (1999). Approximation algorithms to solve real-life multicriteria cutting stock problems. *Operations Research*, 47(4):495–508.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to algorithms*. MIT press.

Dasgupta, S., Papadimitriou, C., and Vazirani, U. (2007). Dynamic programming. *Algorithms*, 1:169–199.

Kabanov, J. and Vene, V. (2006). Recursion schemes for dynamic programming. In *International Conference on Mathematics of Program Construction*, pages 235–252. Springer.

Kleinberg, J. and Tardos, E. (2006). *Algorithm design*. Pearson Education India.

Tanır, D., Ugurlu, O., Guler, A., and Nuriyev, U. (2019). One-dimensional cutting stock problem with divisible items: A case study in steel industry. *TWMS Journal of Applied and Engineering Mathematics*, 9(3):473–484.

Yang, X.-S. (2011). Metaheuristic optimization: algorithm analysis and open problems. In *International symposium on experimental algorithms*, pages 21–32. Springer.