# Análise e Projeto de Algoritmos: Analisando jogos de civilizações antigas

**Ana Beatriz Stahl, Gabriela Bley Rodrigues**

Universidade do Vale do Rio dos Sinos (UNISINOS) – São Leopoldo, RS - Brazil

ana.beatriz.stahl@gmail.com, bleyrodrigues.gabriela@gmail.com

***Resumo.*** *Este artigo aborda a combinação de pontuações em jogos de civilizações antigas, visando equilibrar somas de pontuações de jogadores em campeonatos. Dado um conjunto de placas, busca-se organizá-las para que as somas em ambos os lados sejam iguais, com possibilidade de descartar uma placa para maximizar o equilíbrio. Foram propostas duas abordagens: uma solução de força bruta e uma otimização com programação dinâmica.*

## 1. Introdução

Jogos e competições desempenham papéis centrais na cultura humana, inclusive em civilizações antigas. Esses jogos eram frequentemente registrados em placas que documentavam as pontuações de dois jogadores em cada rodada. A descoberta recente de placas antigas contendo esses registros permitiu que se investigasse a estrutura desses campeonatos, revelando um problema interessante: reorganizar as placas para que as somas das pontuações dos jogadores, em um arranjo lado a lado, sejam equilibradas (Huizinga, 1944; Schädler, 2007).

Este trabalho apresenta uma análise e solução para o problema de encontrar uma combinação de placas que possibilite esse equilíbrio nas somas, considerando que, caso o equilíbrio perfeito não seja alcançável, uma placa pode ser descartada. Este descarte deve ser feito de forma a maximizar o valor da soma e minimizar o valor da placa descartada, respeitando critérios específicos de seleção (Papadimitriou e Steiglitz, 1998).

Para resolver o problema, foram desenvolvidas duas abordagens principais: uma solução inicial baseada em força bruta e uma segunda solução, otimizada com programação dinâmica, que visa aumentar a eficiência para conjuntos maiores de placas. Este artigo discute ambas as soluções e compara seu desempenho, destacando as vantagens e limitações de cada uma (Demaine e Hearn, 2009).

## 2. Primeira solução: Força bruta

Ao elaborar uma solução mais bruta para o problema, pode-se dividir o algoritmo em três etapas principais: leitura de entrada, verificação de combinações e identificação de solução ou impossibilidade. Na etapa de entrada, os dados são lidos de arquivos contendo $n$ placas, cada uma representada por pares $x$ e $y$. As placas são armazenadas em uma lista para processamento subsequente.

Na verificação de combinações, todas as combinações de placas $2^n$ são geradas e avaliadas. Para cada combinação, calcula-se as somas superiores e soma inferior. Caso elas sejam iguais, a solução é encontrada. Se nenhuma combinação atender à condição, o algoritmo irá descartar iterativamente cada placa e recalcula as somas para o subconjunto

restante, de forma a respeitar a potência $2^{n-1}$. Se nenhuma solução for encontrada mesmo com descartes, o resultado é dado como impossível.

Por fim, a análise de complexidade é dividida em duas partes: cálculo de combinações e descarte de placas. O número de combinações possíveis de placas é $2^n$. Para cada combinação, o cálculo das somas envolve uma operação linear $O(n)$ no número de placas. Assim, a complexidade para verificar todas as combinações é $O(2^n.\ n)$. No caso de descarte, o algoritmo tenta remover cada uma das $n$ placas e recalcular as combinações para as $n-1$ restantes. A complexidade dessa etapa se simplifica para $O(n^2.\ 2^n)$.

## 3. Segunda solução: Programação dinâmica

Uma das maneiras para solucionar o problema, é através de programação dinâmica, o algoritmo pode ser dividido em quatro partes: leitura de entrada, criação da matriz $dp$, busca da melhor soma, verificação de descarte de placas. Na etapa de entrada é lido os casos de teste e convertido cada matriz em uma lista de tuplas.

A matriz $dp$ é criada através de um algoritmo de *Subset Sum Problem Bottom-Up Approach*, Krishnan (2020), para rastrear todas as somas possíveis de pares usando as placas disponíveis, para cada par de valores de uma placa, novos estados possíveis são calculados e adicionados ao conjunto. Na etapa de busca pela melhor soma, é feita uma busca pela maior soma balanceada que é possível alcançar, percorrendo do maior valor possível até 0.

A verificação de descarte avalia o descarte de placas em um contexto em que se busca a melhor soma balanceada. O procedimento consiste em recalcular, para cada placa, uma versão da matriz de programação dinâmica excluindo a placa em análise. O objetivo é verificar se a exclusão da placa mantém a possibilidade de alcançar a soma balanceada ideal.

No geral, o processo executa os passos mencionados para cada matriz lida no arquivo de input. A complexidade de cada parte do algoritmo é descrita na tabela a seguir, considerando $d$ o número de matrizes, $p$ o número médio de placas por matriz, e $S$ o valor máximo de soma:

| Passo | Complexidade |
|---|---|
| Leitura do arquivo | $O(n)$ |
| Criação da matriz $dp$ | $O(p.S^2)$ |
| Busca pela melhor soma | $O(S)$ |
| Verificação de descarte | $O(p^2.S^2)$ |
| Processamento total | $O\big(d.(p^2.S^2)\big)$ |

(Fonte: As autoras)

## 4. Resultados

Os resultados obtidos demonstram uma diferença de desempenho entre as abordagens de força bruta e programação dinâmica na resolução do problema de balanceamento de somas de placas.

Durante a análise dos resultados gerados pelo algoritmo de força bruta e sua comparação com os casos de teste, observou-se que os arquivos de saída *out, out1, out2, out5, out6 e out7* apresentaram resultados corretos. Entretanto, os arquivos *out3* e *out4* - que coincidem com as entradas de maior volume de dados - apresentaram algumas inconsistências, com determinadas saídas marcadas como impossíveis. Apesar disso, a maioria dos resultados desses dois casos estava correta, exceto por algumas linhas específicas. Devido à complexidade (muitos dados) envolvida no processo de debug, não foi possível identificar a causa do erro. Outro detalhe importante a ser mencionado é que não foi possível executar todas as análises de forma sequencial. O computador despendia muito tempo nas entradas de maior dado, tornando a execução desse tipo de algoritmo inviável para entradas grandes no nível de *in3* e *in4*. A tabela abaixo explicita o tempo tomado nas instâncias em que o algoritmo foi executado:

| **Arquivos executados** | **Tempo (ms)** |
|---|---|
| in, in1, in2, in5, in6, in7 | ~119ms |
| in3 | ~600000ms |
| in4 | Não finalizou |

(Fonte: As autoras)

No caso dos resultados obtidos com o algoritmo de programação dinâmica, observou-se que mesmo sendo mais eficiente para rodar casos menores como o dos arquivos de entrada *in, in1, in2 e in6*, é necessário um tempo considerável para rodar casos maiores como o *in7*. A execução de algumas entradas foram inviáveis mesmo utilizando programação dinâmica, como nos casos *in3, e in4*. A tabela abaixo explicita o tempo tomado nas instâncias em que o algoritmo foi executado.

| Arquivos executados | Tempo (ms) |
|---|---|
| in, in1, in2, in6, | ~0.800ms |
| in7 | ~4137ms |
| in5 | MemoryError |
| in3, in4 | Não finalizou |

(Fonte: As autoras)

Como visto acima, surgiu uma exceção durante a implementação. O *MemoryError* é gerado quando uma operação fica sem memória. Seguindo a análise, foi possível observar que o *buffering* de saída - que é uma técnica usada por sistemas operacionais e bibliotecas de E/S para otimizar a escrita de dados, armazenando-os temporariamente em memória antes de exibi-los no terminal ou gravá-los em arquivos - teve influência nas medições de tempo de execução do código, pois tempos de execução medidos podem não refletir com precisão a duração real do processamento, especialmente se o buffer não for esvaziado imediatamente. Para evitar esse cenário, existem primitivas para eliminar este problema.

## 5. Conclusão

Em termos práticos, a abordagem de força bruta é adequada apenas para instâncias pequenas devido à sua complexidade. Para aplicações em escala maior, seria necessário explorar heurísticas ou algoritmos mais eficientes, como técnicas aproximativas.

## 6. Referências

HUIZINGA, Johan. Homo Ludens: A Study of the Play-Element in Culture. Routledge, 1944.

KRISHNAN, M. Dynamic Programming Part 3: Subset Sum problem. Disponível em: https://medium.com/@muralikrishnan.official/dynamic-programming-part-3-subset-sum-problem-9d063b5ef1b5. Acesso em: 17 out. 2024.

SCHÄDLER, Ulrich. Board Games in Antiquity: A Survey of the Near East and Mediterranean. In: FAWKNER, Irving (Ed.). Ancient Board Games in Perspective. London: The British Museum Press, 2007.

PAPADIMITRIOU, Christos H.; STEIGLITZ, Kenneth. Combinatorial Optimization: Algorithms and Complexity. Dover Publications, 1998.

DEMAINE, Erik D.; HEARN, Robert A. Games, Puzzles, and Computation. In: GAL, Shira (Ed.). Game Theory and Applications. Nova Science Publishers, 2009.