Análise e Complexidade de Algoritmos: *Backtracking* e Força Bruta

Raphael Augusto Nascimento Rodrigues[1], Millys Fabrielle Araújo Carvalhaes[2]

[1,2] Bacharelados em Computação - Centro Universitário de Anápolis (UniEVANGÉLICA) – Anápolis, GO

[1]millys.carvalhaes@docente.unievangelica.edu.br

Resumo. *Este trabalho apresenta a análise dos algoritmos Backtracking e Força Bruta utilizando um problema de ordenação de elementos de um vetor.*

1. Introdução

A análise de complexidade de algoritmos tem como função determinar os recursos necessários para executar um dado algoritmo. Fazer um estudo detalhado da eficiência de um algoritmo, da quantidade de memória utilizada, do tempo de execução expedido, pode-se obter um melhor desempenho para o funcionamento do algoritmo, e é importante que um estudo seja feito em cima de algoritmos de *backtracking* e força bruta, para que se obtenha o melhor desempenho e eficiência.

2. Backtracking

Backtracking é um método para iterar (percorrer) todas as possíveis configurações em um espaço qualquer. É um algoritmo geral que poderá ser personalizado para cada tipo de aplicação. De modo geral, a solução será um vetor $a = (a1, a2, \dots, a - n)$, sendo cada elemento $a - i$ selecionado de um conjunto Si. Os exemplos mais comuns, os quais serão mostrados posteriormente, são a criação de permutações e de subconjuntos.

A cada passo do algoritmo, começa-se com uma solução parcial, diga-se $a = (a1, a2, \dots, a - k)$, e tenta-se aumentá-la, adicionando outro elemento ao seu fim. Após adicioná-lo, é preciso testar se há uma solução completa – se sim, deve-se imprimi-la, contá-la ou qualquer ação necessária. Se não é necessário checar se essa solução parcial é ainda expansível para alguma completa. Se for, continue testando, se não, então apague o último elemento de a e teste outra possibilidade para aquela posição.

2.1. Características do Backtracking

Algoritmos tentativa e erro não seguem regra fixa de computação:

- Passos em direção à solução final são tentados e registrados;
- Caso esses passos tomados não levem à solução final, eles podem ser retirados e apagados do registro.
- Quando a pesquisa na árvore de soluções cresce rapidamente é necessário usar algoritmos aproximados ou heurísticas que não garantem a solução ótima, mas são rápidas.
- O número de escolhas cresce pelo menos exponencialmente com o tamanho da instância.
- Constrói a solução, um componente por vez, tentando terminar o processo tão logo quanto for possível identificar que uma solução não poderá ser feita obtida em razão das escolhas feitas.

- Esta técnica torna possível a resolução de muitos problemas NP-difícil com instâncias grandes em um tempo aceitável.
- Elimina um nó se for garantido que ele não levará a obtenção de uma solução para o problema.
- Faz busca em profundidade.

O *Backtracking* assegura o acerto por enumerar todas as possibilidades sem visitar nunca o mesmo estado, sendo também eficiente. A recursividade promove a elegância e a fácil implementação desse algoritmo, porque o vetor de novos candidatos, c, é alocado com um procedimento recursivo. As principais aplicações do *backtracking* são da criação de todos os subconjuntos de um conjunto S e na criação de todas as suas permutações.

3. Força Bruta

A força bruta é um algoritmo trivial, mas de uso muito geral que consiste em enumerar todos os possíveis candidatos de uma solução e verificar se cada um satisfaz o problema. Por exemplo, um algoritmo para encontrar os divisores de um número natural n é enumerar todos os inteiros de 1 a n, e verificar para cada um se ele dividido por n resulta em resto 0.

Esse algoritmo possui uma implementação muito simples, e sempre encontrará uma solução se ela existir. Entretanto, seu custo computacional é proporcional ao número de candidatos a solução, que, em problemas reais, tende a crescer exponencialmente. Portanto, a força bruta é tipicamente usada em problemas cujo tamanho é limitado, ou quando há uma heurística usada para reduzir o conjunto de candidatos para um espaço aceitável. Também pode ser usado quando a simplicidade da implementação é mais importante que a velocidade de execução, como nos casos de aplicações críticas em que os erros de algoritmo possuem em sérias consequências.

3.1. *Aplicação na ordenação*

Um exemplo clássico de aplicação de algoritmos da classe da força bruta é a ordenação, que pode ser aplicada nos mais diferentes tipos de dados como *Selection Sort*, *Bubble Sort*, entre outros. Por exemplo, uma das formas de resolver o problema consiste em procurar o menor elemento e colocá-lo na primeira posição, operando sucessivamente com os demais elementos da lista a ser classificada até finalizar a operação, um método conhecido como ordenação por inserção.

Figura 1. Algoritmo de busca.

```
rotina BuscaPadrao(texto[0 ... n-1], padrao[0 ... m-1])
    para i ← 0 até n – m faça
        j ← 0
        enquanto j < m e padrao[j] = texto[i + j] faça
            j ← j + 1
        seSe j = m então
            retorne i
retorne -1
```

Fonte: Autores

Outro exemplo é a busca de padrões. Dada uma cadeia de caracteres de tamanho n (o "texto") e outra cadeia com tamanho m menor ou igual chamada "padrão". É realizada uma

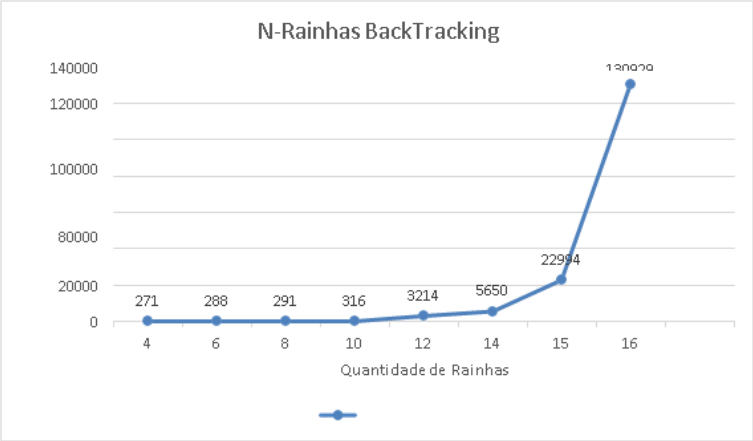
procura no "texto" pelo "padrão", verificando-se todo o texto em busca de múltiplas ocorrências. O funcionamento da busca de padrão é simples: se o primeiro caractere é idêntico à um referente no texto, todos os sucessores devem ser idênticos também, até finalizar o padrão. Caso ocorra que um caractere não seja igual, desloca-se o padrão em um caractere até chegar ao fim do texto ou encontrar o padrão, conforme apresentado pela Figura 1.

O algoritmo apresentado na Figura 1 se desloca após a comparação do primeiro caractere, porém nem sempre é assim. O pior caso é "muito pior": o algoritmo tem que comparar todos os m caracteres antes de se deslocar, e isso pode ocorrer para cada uma das $n - m + 1$ tentativas. Portanto, o pior caso para este algoritmo está em $\theta(n.m)$. Para textos de linguagem natural, o caso médio é melhor (pois ocorre nas primeiras comparações). Até em textos aleatórios, o comportamento se mostra linear, $\theta(n + m) = \theta(n)$.

4. Resultados Obtidos

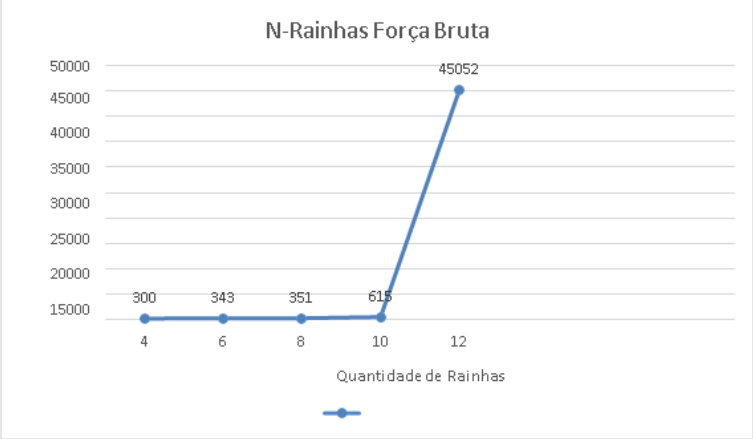
Os gráficos a seguir demonstram o comportamento dos algoritmos de *backtracking* e força bruta (n rainhas) em várias situações, para tamanho inicial de $n = 4$. Os eixos verticais correspondem ao tempo de execução, em milissegundos, e nos eixos horizontais, a quantidade de rainhas. Os resultados foram obtidos a partir de 3 execuções de cada algoritmo em cada situação e calculada a média das 3 execuções.

Figura 2. Execução do Algoritmo n-Rainhas – *BackTracking*.



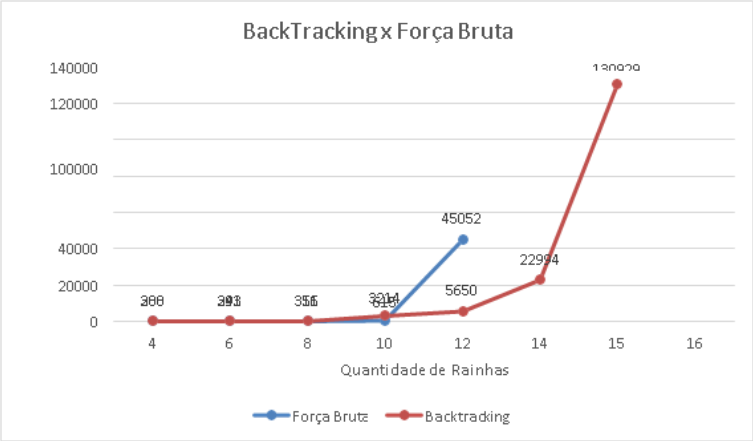
Fonte: Autores

Figura 3. Algoritmo n-Rainhas Força Bruta.



Fonte: Autores

Figura 4. Comparação entre Força Bruta e *Backtracking*.



Fonte: Autores

5. Considerações Finais

No contexto geral, o algoritmo de *backtracking* tem um melhor desempenho e calcula com maior eficiência um maior número de rainhas. Para n menor que 10 os resultados são quase idênticos. Para n maior que 12 o algoritmo de força bruta não apresenta desempenho satisfatório. O algoritmo foi executado por mais de 2 horas e não apresentou resultados. O mesmo vale para o algoritmo de *backtracking*, porém este, apresentou resultados aceitáveis até n igual a 16.

Referências

Duarte, H (2017). “Backtracking”. http://heltonduarte.com/2009/04/09/programando-melhor-aula-5-backtracking/. Março

Barbosa, F. R. S., Sousa, F. R. J., Vilela, L. R. (2017). “Trabalho de Análise de Algoritmos: Problema da mochila”. http://www.ic.uff.br/~jsilva/artigo_problema_da_mochila.pdf. Março.