UNIVERSIDADE DO RIO GRANDE DO NORTE

Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Bacharelado em Ciência da Computação

MOISÉS FERREIRA DE LIMA

Estudo Dirigido 5 - Desempenho e Otimização de Código e Padrões de Projeto

NATAL/RN
2024

MOISÉS FERREIRA DE LIMA

Estudo Dirigido 5 - Desempenho e Otimização de Código e Padrões de Projeto

Relátorio sobre o projeto do estudo dirigido 5 implementado durante a disciplina Boas Práticas de Programação, ministrada pelo professor Dr. Edson Jackson de Medeiros Neto no Curso de Ciência da Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte – Campus Central (Natal).

NATAL/RN
2024

Sumário

1 INTRODUÇÃO **3**
1.1 Visão geral . . . 3
1.2 Contexto do problema . . . 3
1.3 Objetivos . . . 3

2 DESCRIÇÃO DO PADRÃO STRATEGY **4**
2.1 O padrão de projeto Strategy . . . 4
2.2 Aplicação do padrão Strategy no contexto do inventário de produtos . . . 4

3 IMPLEMENTAÇÃO **5**
3.1 Implementação da interface SearchStrategy . . . 5
3.2 Implementação da classe concreta LinearSearchStrategy . . . 5
3.3 Implementação da classe concreta BinarySearchStrategy . . . 5
3.4 Implementação da classe de contexto InventorySearchContext . . . 6

4 ANÁLISE DE DESEMPENHO **8**
4.1 Complexidade de tempo das duas estratégias de busca . . . 8
4.2 Metodologia utilizada para realizar a análise de desempenho . . . 8
4.3 Resultados obtidos da análise de desempenho . . . 8
4.3.1 Primeiro teste . . . 8
4.3.2 Segundo teste . . . 9

5 DISCUSSÃO SOBRE ESTRUTURA DE DADOS **11**
5.1 Uso dos arrays . . . 11
5.2 A Tabela Hash . . . 11

6 CONCLUSÃO **12**

7 REFERÊNCIAS **13**

1 INTRODUÇÃO

1.1 Visão geral

Os algoritmos de busca são ferramentas importantes e muito utilizadas em diversos contextos para verificar se um item está presente em uma determinada lista de itens e para retornar informações sobre ele, caso o item esteja na lista. Neste relatório, serão apresentadas duas estratégias de busca, a busca sequencial e a busca binária, aplicadas a um problema de inventário de produtos, seguindo o padrão de projeto Strategy.

1.2 Contexto do problema

A aplicação desenvolvida durante a realização do estudo dirigido 5 consiste em um sistema de inventário de produtos para uma pequena loja. Nesse projeto, um array contendo códigos de produtos cadastrados é mantido pelo sistema. Nesse contexto, o problema que o programa implementado visa resolver consiste em permitir ao usuário escolher a estratégia de busca que ele considerar mais eficiente para realizar a procura de um determinado produto no inventário, através do seu código.

1.3 Objetivos

O projeto desenvolvido possui como objetivos:

- Melhorar o entendimento do padrão de projeto Strategy através da implementação de diferentes algoritmos de busca como estratégias intercambiáveis;
- Analisar o tempo de execução das buscas para diferentes tamanhos de vetores;
- Discutir sobre a estrutura de dados, explicando as vantagens e desvantagens do uso de arrays em determinados cenários e evidenciando a importância dos vetores estarem ordenados para a realização da busca binária;
- Mostrar como escolhas de algoritmos e estruturas de dados impactam o desempenho de um sistema de inventário de produtos.

2 DESCRIÇÃO DO PADRÃO STRATEGY

2.1 O padrão de projeto Strategy

O Strategy é um padrão de projeto que visa encapsular uma família de algoritmos em um conjunto de classes que são intercambiáveis, onde cada uma dessas classes implementa de formas diferentes uma mesma funcionalidade. Com isso, o padrão Strategy permite que o usuário escolha qual algoritmo (ou estratégia) vai ser usada em tempo de execução.

A estrutura do padrão Strategy contém três componentes principais:

- Strategy: É uma interface comum a todos os algoritmos (estratégias) suportados. Nela se declara métodos abstratos que serão utilizado pelas classes de estratégia concreta para implementar as diferentes formas de uma determinada funcionalidade.
- ConcreteStrategy: São classes que, usando a interface Strategy, implementam diferentes variações de um algoritmo.
- Context: É uma classe que mantém uma referência para umas das estratégias concretas. Ela permite o usuário trocar de estratégia em tempo de execução e utiliza a interface Strategy para executar uma estratégia.

2.2 Aplicação do padrão Strategy no contexto do inventário de produtos

No contexto apresentado nesse relatório, o padrão de projeto Strategy foi implementado através da construção quatro classes:

- SearchStrategy: Classe que representa a interface, onde se declara o método search, que realiza a busca do código de um produto no inventário e retorna o índice da localização do código no array caso o produto esteja no inventário ou o valor -1 caso o produto não se faça presente no array.
- LinearSearchStrategy: É uma classe que representa uma das estratégias concretas. Ela herda o método search da interface SearchStrategy e implementa a busca através do algoritmo de busca linear.
- BinarySearchStrategy: É a classe que representa a outra estratégia concreta. Assim como a LinearSearchStrategy, a classe BinarySearchStrategy também herda o método search da interface SearchStrategy, porém ele implementa o algoritmo de busca utilizando a estratégia da busca binária.
- InventorySearchContext: Classe que reprenta o Contexto. Essa classe possui uma referêcia para a estratégia de busca atual (aquela que será aplicada caso o usuário queira realizar a busca). Além disso, essa classe contém um método chamado setStrategy, no qual o usuário pode passar uma estratégia específica, alterando assim a forma de busca, e um método chamado performSearch, que aplicará a estratégia de busca atual no inventário.

3 IMPLEMENTAÇÃO

A implementação do padrão Strategy foi feita com a linguagem de programação C++, porém apenas os pseudocódigos dos algoritmos serão apresentados para que o entendimento dos algoritmos não seja prejudicado pelas especificidades da linguagem.

3.1 Implementação da interface SearchStrategy

Na interface foi definido o método `search` que terá seus diferentes comportamentos implementados nas classes concretas. Esse método possui como parâmetros o array `productCodes`, que armazena os códigos dos produtos, o inteiro `size`, que representa o tamanho do array `productCodes` e o inteiro `key`, código que será buscado no array.

Algorithm 1: Definição da Interface `SearchStrategy`

```
1  Interface SearchStrategy:
       // Função para buscar um código de produto
       // Deve ser implementada pelas classes concretas
2      Método search(productCodes, size, key):
3      end
4  end
```

3.2 Implementação da classe concreta LinearSearchStrategy

Nessa classe, o algoritmo de busca linear foi implementado. Esse algoritmo consiste em percorrer todo o array até encontrar o valor procurado. Caso o valor esteja no array, será retornado o índice da posição desse valor no array. Porém, se o produto não for encontrado, o valor -1 será retornado.

Algorithm 2: Definição da Classe `LinearSearchStrategy`

```
1  Classe LinearSearchStrategy implementa SearchStrategy:
2      Método search(productCodes, size, key):
3          for i ← 0 to size − 1 do
4              if productCodes[i] = key then
5                  return i;
6              end
7          end
8          return −1;
9      end
10 end
```

3.3 Implementação da classe concreta BinarySearchStrategy

Foi implementado nessa classe a estratégia de busca binária. Para a realização dessa busca é necessário que o array passado esteja ordenado, pois esse algoritmo utiliza a ordenação do vetor para realizar uma busca de maneira mais rápida.

Nesse algoritmo, primeiramente se verifica o valor localizado no meio do array. Se o valor for igual ao código buscado, será retornado o índice da posição desse valor. Caso

esse valor seja menor que o código procurado, será realizado a busca na metade direita do array, descartando a metade esquerda, já que, como o vetor está ordenado, é garantido que todos os outros valores dessa parte são menores que o código desejado. De maneira análoga, caso o valor do meio do array seja maior que o código procurado, a busca será realizada na metade esquerda do vetor e a outra metade será descarta. A busca na metade do vetor seguirá a mesma lógica: O valor do meio do subarray restante será analisado, se for igual ao código desejado, sua posição no array será retornada; caso contrário, seguirá os mesmos passos descritos anteriormente até que se encontre o código do produto ou se verifique que o código não está presente no array.

Algorithm 3: Definição da Classe `BinarySearchStrategy`

```
1  Classe BinarySearchStrategy implementa SearchStrategy:
2      Método search(productCodes, size, key):
3          begin ← 0;
4          end ← size;
5          while begin < end do
6              mid ← begin + (end − begin)/2;
7              if productCodes[mid] = key then
8                  return mid;
9              end
10             else if productCodes[mid] < key then
11                 begin ← mid + 1;
12             end
13             else
14                 end ← mid;
15             end
16         end
17         return −1;
18     end
19 end
```

3.4 Implementação da classe de contexto InventorySearchContext

Essa classe vai manter uma referência para um dos objetos estratégias. Além disso ela terá um método chamado `setStrategy`, que vai permitir a troca de estratégia durante o tempo de execução, e um método `performSearch`, que delega a busca ao método cuja estratégia está sendo referenciada atualmente.

Algorithm 4: Definição da Classe `InventorySearchContext`

Classe *InventorySearchContext*:
  private: `strategy`: Referência para uma estratégia concreta;
  Método *`setStrategy(desiredStrategy: Referência para a estratégia que se deseja aplicar)`*:
    $strategy = desiredStrategy$;
  end
  Método *`performSearch(productCodes, size, key)`*:
    return $strategy.search(productCodes, size, key)$;
  end
end

4 ANÁLISE DE DESEMPENHO

4.1 Complexidade de tempo das duas estratégias de busca

Consideremos agora a complexidade de tempo do pior caso dos dois algoritmos de busca: a busca linear e a busca binária. Na busca linear, a complexidade de tempo do pior caso é $\mathcal{O}(n)$, onde n é o tmanho do array, pois, nesse caso, o algoritmo irá percorrer todo o array, verificando cada um dos n elementos dele.

Na busca binária, o fato do algoritmo descartar metade do array restante em cada iteração da busca faz com que a complexidade de tempo do pior caso dessa estratégia seja $\mathcal{O}(\log n)$.

4.2 Metodologia utilizada para realizar a análise de desempenho

Com o objetivo de avaliar o desempenho das estratégias de busca, foi realizado a medição do tempo de execução de cada algoritmo utilizando 61 amostras de arrays de tamanhos diferentes entre si, igualmente espaçadas entre a menor e a maior amostra. Neste projeto, a menor amostra tem 50 elementos e cada vetor sequinte possui 5000 elementos a mais do que o anterior, fazendo com que a amostra com maior quantidade de valores armazenados tenha 300050 elementos. Vale ressaltar que todos os elementos dos arrays eram inteiros positivos (uma vez que se tratavam de códigos de produtos) e que o valor que foi buscado nos arrays era um número negativo e, portanto, que não estava presente no vetor. A escolha desse valor negativo teve como intenção sempre considerar a análise de desmpenho do pior caso da busca.

Com base no que foi dito no parágrafo anterior, dois testes foram realizados. No primeiro, a medição do tempo de execução dos algoritmos de busca foi feita com todas as amostras de vetores estando ordenadas. No segundo teste, a medição foi realizada usando vetores desordenados. Nesse último, foi medido o tempo tanto da ordenação do array quanto da busca binária com a finalidade de analisar em quais situações haveriam maior benefício em utilizar a busca linear em detrimento a busca binária e vice-versa.

Para facilitar a análise da eficiência dos algoritmos de busca, o software gnuplot foi utilizado, possibilitando a criação de gráficos com informações relevantes sobre os dados.

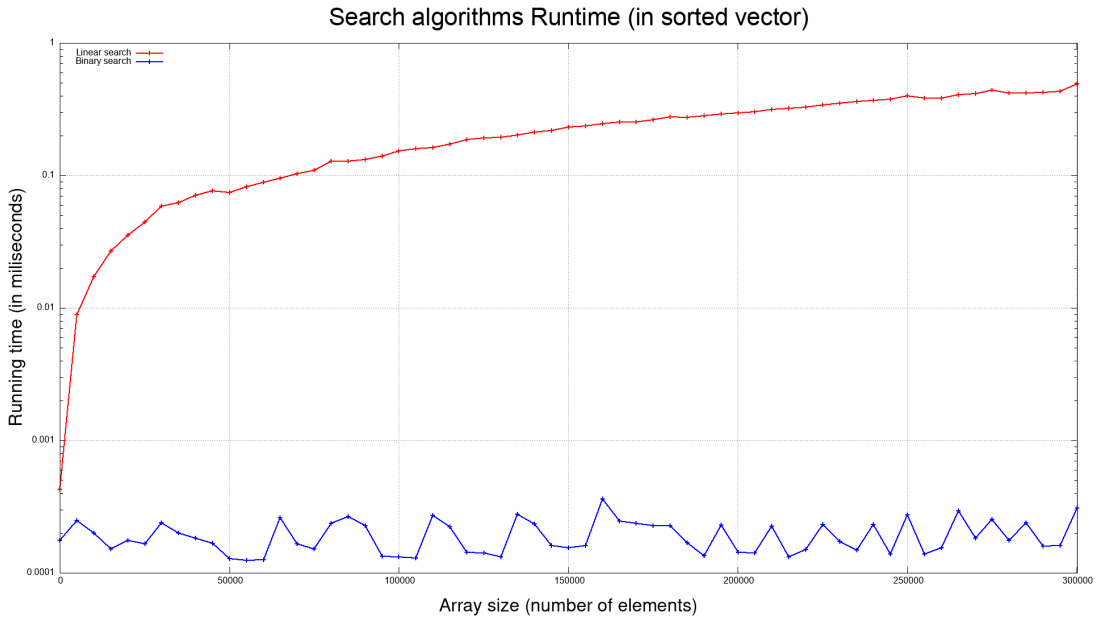
4.3 Resultados obtidos da análise de desempenho

4.3.1 Primeiro teste

No primeiro teste, utilizando o gnuplot para traçar os dados obtidos durante a medição do tempo das estratégias de busca, foi possível construir o seguinte gráfico apresentado na Figura 1:

Com base na Figura 1, pode-se perceber que a busca binária é bem mais rápida do que a busca linear, fato que está em conformidade com a teoria, uma vez que, no pior caso, a complexidade de tempo da busca linear é $\mathcal{O}(n)$, enquanto a da busca binária é $\mathcal{O}(\log n)$. Além disso, é possível concluir ainda que na medida em que o tamanho do array vai aumentando, o tempo de execução da busca linear também aumenta. Teoricamente, a busca binária também deveria apresentar esse mesmo comportamento da busca linear, porém, como o tempo da execução da busca binária é muito pequeno, outros programas que estão sendo executados concorrentemente no computador acabam atrapalhando a

Figure 1: Desempenho dos algoritmos de busca em vetores ordenados



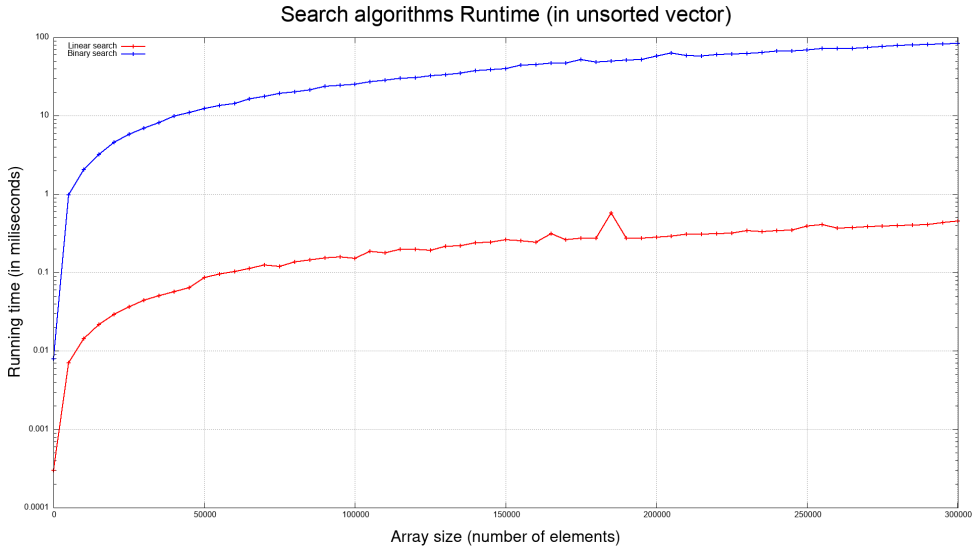
Fonte: Gerado no gnuplot (2024).

medição dessa estratégia, gerando essas oscilações do tempo medido apresentadas na Figura 1.

4.3.2 Segundo teste

Agora com todos os vetores da amostra estando desordenados, o gnuplot permitiu a realização do esboço do gráfico da Figura 2, possibilitando uma análise mais profunda sobre o custo da ordenação necessária para a realização da busca binária em vetores desordenados.

Figure 2: Desempenho dos algoritmos de busca em vetores não ordenados



Fonte: Gerado no gnuplot (2024).

Baseado na Figura 2, é possível observar que busca binária se torna bastante custosa em relação ao tempo quando o array está desordenado, uma vez que é necessário primeiramente ordenar o vetor para depois realizar a busca. O algoritmo de ordenação utilizado é o Introspective Sort, que possui complexidade de pior caso $\mathcal{O}(n^2)$ quando o tamanho do vetor é pequeno e $\mathcal{O}(n \log n)$ quando o vetor é grande, fazendo com que a busca linear acabe sendo uma melhor opção quando o vetor está desordenado.

No entanto, apesar da melhor eficiência da busca linear em vetores não ordenados, podem haver situações em que ainda seja mais benéfico utilizar a busca binária. Essa escolha de estratégia depende de quais operações são mais realizadas na aplicação que um determinado desenvolvedor quer produzir. Se essa aplicação exigir, por exemplo, grande quantidade de inserções no vetor, operação que pode acabar desordenando essa estrutura de dados com muita frequência, então não seria viável o uso da busca binária, por causa do uso recorrente do algoritmo de ordenação. Porém, se a aplicação tiver como operação predominante a busca de um valor no vetor, então a busca binária seria mais eficiente, uma vez que o algoritmo de ordenação seria utilizado com menos frequência e as buscas seriam mais frequentes.

5 DISCUSSÃO SOBRE ESTRUTURA DE DADOS

5.1 Uso dos arrays

O array é uma estrutura de dados que armazena um conjunto de elementos em que cada um pode ser identificado pelo seu índice. Eles são fáceis de utilizar e apresentam um bom desempenho em relação a operação de busca quando são usados em aplicações onde o número de elementos presente neles não é tão grande. Além disso, essa eficiência na busca pode melhorar consideravelmente se for garantido que o array é ordenado devido ao algoritmo de busca binária como já foi visto nos itens 4.1 e 4.3.1 desse relatório. No entanto, quando o número de elementos começa a aumentar muito, o desempenho da busca no array pode não ser tão eficiente, fazendo com que outras estruturas de dados, tais como a Tabela Hash, sejam mais eficientes em aplicações que precisem armazenar grandes quantidades de dados.

5.2 A Tabela Hash

A tabela Hash pode ser uma boa escolha em aplicações que exijam o armazenamento de grandes quantidades de dados. Essa estrutura de dados que associa chaves a valores através de uma determinada função, chamada função hash. Logo, quando se procura uma chave nessa estrutura, a função hash mapeia a chave para um índice na tabela hash, permitindo encontrar o valor desejado. Pode ocorrer que ao se adicionar um item na tabela hash, a função hash mapeie a chave desse item para o mesmo índice de outra chave na tabela, fato chamado de colisão. Existem alguns métodos para lidar com colisões, mas não será apresentado aqui, pois foge do objetivo desse relatório.

Dependendo de eficiência da função hash, a operação de busca pode chegar a ser mais eficiente do que a busca binária, podendo apresentar uma complexidade de tempo média $\mathcal{O}(1)$, oque faz dela uma estrutura de alto desempenho em relação ao tempo de execução.

6 CONCLUSÃO

Com base na análise das informações contidas nesse relatório, foi possível conceber experimentalmente que, quando o vetor está ordenado, a busca binária é muito mais rápida do que a busca linear. Foi possível analisar também que, apesar da inviabilidade do uso da busca binária em arrays ordenados, podem haver aplicações em que o uso da busca binária pode ser mais eficiente, como em situações onde a busca é realizada várias vezes e as operações que podem alterar a ordenação da lista ocorrem com pouca frequência.

Além disso, esse trabalho permitiu um compreendimento melhor do padrão de projeto Strategy, tanto de maneira teórica, quanto prática através da implementação do padrão em um problema real. Por fim, uma possível extensão desse projeto poderia ser a implementação de uma hash table para analisar o desempenho das buscas linear e binária em um vetor em relação a operação de busca em uma tabela hash. Essa nova implementação ajudaria entender melhor em quais situações o uso da tabela hash seria mais vantajoso do que a utilização do array.

7 REFERÊNCIAS

https://refactoring.guru/pt-br/design-patterns/strategy

https://www-geeksforgeeks-org.translate.goog/introsort-or-introspective-sort/?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc

https://pt.wikipedia.org/wiki/Arranjo_(computa%C3%A7%C3%A3o)

https://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node26.html

https://www.dio.me/articles/estrutura-de-dados-tabela-hash

https://pt.wikipedia.org/wiki/Tabela_de_dispers%C3%A3o

https://www-modernescpp-com.translate.goog/index.php/the-strategy-pattern/?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc

https://refactoring.guru/pt-br/design-patterns/strategy/cpp/example

https://www.dio.me/articles/padrao-de-projeto-strategy-a-arte-de-trocar-comportamentos

https://brainly.com.br/tarefa/52321667

https://www.devmedia.com.br/estudo-e-aplicacao-do-padrao-de-projeto-strategy/25856