Relatório T1
Análise de complexidade do problema proposto

Max Franke*
Escola Politécnica — PUCRS

21 de março de 2019

Resumo

Este artigo descreve duas alternativas de solução para o primeiro problema proposto na disciplina de Algoritmos Avançados. Que tem como objetivo separar os números ímpares e pares dentro de um vetor, onde a entrada de dados é fornecido um vetor de inteiros desorganizado. É apresentada apenas duas soluções para o problema e a suas eficiências são analisadas. Por fim, os resultados são apresentados.

Introdução

Dado um vetor com n números inteiros, deve-se fazer dois métodos onde ambos recebem vetores com números inteiros sortidos e ambos devem retornar um vetor com os números pares a frente do vetor e os ímpares ao final. A diferença entre os dois métodos é a estratégia que deve ser adotada para resolver o método. Na primeira solução, utilizaremos um vetor auxiliar para facilitar a separação dos números. Já na segunda solução, iremos operar sobre o mesmo vetor, sem acrescentar um vetor extra para auxiliar.

Analisaremos as duas alternativas de solução. Em seguida os resultados obtidos serão apresentados, bem como as conclusões obtidas no decorrer do trabalho.

Algoritmo com vetor auxiliar

Esta primeira solução é a mais simples de ser pensada, pois, ao utilizarmos um vetor auxiliar, basta verificar cada número no vetor e adicionar na posição correspondente no vetor auxiliar. Para descobrir qual posição o número deve ser "copiado", utilizaremos dois contadores:

- contInicio - onde está "apontando"para a primeira posição do vetor auxiliar, e a cada número par que ele copia do vetor original, o contInicio vai para a próxima posição.
- contFim - onde está "apontando"para a ultima posição do vetor auxiliar, e a cada número impar que ele copia do vetor original, o contFim ele decresce uma posição, indo para "trás".

Com os dois "contadores", o vetor auxiliar vai sendo preenchido com números pares à esquerda, e ímpares à direita. Por fim, retornaremos o vetor auxiliar

*max.franke@acad.pucrs.br

```java
public static int[] separaParImpar(int[] vet) {
   int[] vetAux = new int[vet.length] ;
   int contInicio = 0 ;
   int contFim = vet.length − 1 ;
   for (int i = 0 ; i < vet.length ; i++) {
      if (vet[i] % 2 == 0) {
         vetAux[contInicio] = vet[i] ;
         contInicio++ ;
      } else {
         vetAux[contFim] = vet[i] ;
         contFim−− ;
      }
   }
return vetAux ;
}
```

Listing 1: Java

Esse algoritmo obrigatoriamente sempre irá percorrer o tamanho do vetor até o final, independentemente dos números inteiros dentro dispostos. A complexidade do algoritmo é $\mathcal{O}(n)$ e $\Omega(n)$, consequentemente $\Theta(n)$, sendo n o tamanho do vetor passado por parâmetro. O pior caso e o melhor caso são idênticos e independem dos números que irão ser organizados, pois as clausulas "if"e "else"manipulam o vetor custando o mesmo preço de processamento.

Algoritmo sem vetor auxiliar

Após se pensar em uma solução com vetor auxiliar, ficará mais simples em pensar-se um algoritmo que não utilize uma estrutura auxiliar. A ideia é semelhante à primeira solução, continuaremos utilizando dois contadores, entretanto eles estarão em uma iteração que será capaz de fazer com que eles avancem casas até acharem os números (pares ou ímpares) alvos:

- contInicio - onde está "apontando"para a primeira posição do vetor original que possua um número ímpar. Ou seja, o número ímpar mais a esquerda do vetor.
- contFim - onde está "apontando"para a ultima posição do vetor original que possua um número par. Ou seja, o número par mais a direita do vetor.

Quando o "contInicio"estiver em um número ímpar mais a esquerda, e o "contFim"estiver em um par mais a direita. Os números são trocados de posições. Com isso, resta-se apenas números pares à esquerda, e ímpares à direita.

```java
public static int[] separaParImpar2(int vet[]) {
   int contInicio = 0 ;
   int contFim = vet.length − 1 ;
   while (contInicio < contFim) {
      while (vet[contInicio] % 2 == 0 and contInicio < contFim) {
         contInicio++ ;
      }

      while (vet[contFim] % 2 == 1 and contInicio < contFim) {
         contFim−− ;
      }

      if (contInicio < contFim) {
         int aux = vet[contInicio] ;
         vet[contInicio] = vet[contFim] ;
         vet[contFim] = aux ;
         contInicio++ ;
```

```
18              contFim−− ;
19          }
20      }
21      return vet ;
22  }
```

Listing 2: Java

Por mais que o pensamento para se resolver o problema seja parecido com a primeira solução, uma das grandes diferenças já começa na linha 4, onde o laço iterativo frequentemente não irá percorrer todo o tamanho do vetor, e sim uma parte dele. Os laços das linhas 5 e 9 são apenas para adiantar e recuar os contadores. Já o condicional "if"da linha 13 é onde ocorre a troca, e onde o peso computacional é maior.

Surpreendentemente, o melhor caso ocorre quando o algoritmo executa o laço interno até o fim (linha 5). Isso apenas ocorre quando o vetor passado para parâmetro contêm apenas números pares, pois ao final dele, com o "contInicio"sendo igual ao "contFim", ignorará as condicionais da linha 9 e 13 e terminará o laço externo.

Já o pior caso ocorre quando o vetor está totalmente dividido entre impares à esquerda e pares à direita, com todos os números trocados. Com isso é ignorado as instruções dos laços 5 e 9 (entretanto a análise é feita mesmo assim, mesmo que seja falsa), mas faz-se a condicional da linha 13 toda iteração. Ao analisarmos essa condicional, percebe-se que ambos os contadores movimentam-se 1 casa, portanto essa troca apenas será realizada $\frac{n}{2}$ vezes. Sendo n o tamanho do vetor.

Por mais que os $whiles$ encadeados passem a ideia de que se trata de uma notação $\mathcal{O}(n^2)$, vimos que se trata de uma notação $\mathcal{O}(n)$ e $\Omega(n)$, e consequentemente uma notação $\Theta(n)$, sendo n o tamanho do vetor passado por parâmetro.

Resultados

Depois de implementar o algoritmo acima em Java e executá-lo, obtivemos os seguintes resultados:

```
100 elementos, 50 pares e 50 ímpares;
Iterações com vetor aux:  1006
Iterações sem vetor aux:  701


100 elementos, 20 pares e 80 ímpares;
Iterações com vetor aux:  1006
Iterações sem vetor aux:  781


100 elementos, 80 pares e 20 ímpares;
Iterações com vetor aux:  1006
Iterações sem vetor aux:  638


100000 elementos, 50000 pares e 50000 ímpares;
Iterações com vetor aux:  1000006
Iterações sem vetor aux:  674949


100000 elementos, 20000 pares e 80000 ímpares;
Iterações com vetor aux:  1000006
Iterações sem vetor aux:  809881


100000 elementos, 80000 pares e 20000 ímpares;
Iterações com vetor aux:  1000006
Iterações sem vetor aux:  639998
```

Conclusões

Fazendo-se teste de mesa conclui-se que a solução utilizando vetor auxiliar segue o padrão da função $f(n) = 10n + 6$ onde n é o tamanho do vetor passado por parâmetro. Para todo e qualquer tipo de inteiro, o algoritmo irá seguir esta função, sendo ele o melhor caso ou o pior.

Já na solução que não utiliza um vetor auxiliar, o pior caso descreve a função: $f(n) = \frac{18n}{2} + 6$ que pode ser simplificada como: $f(n) = 9n + 6$ sendo n o tamanho do vetor e as condições já foram descritas anteriormente.

E o melhor caso pode ser descrito como: $f(n) = 6n+8$ sendo n o tamanho do vetor e as condições também foram descritas anteriormente.

Tendo-se essas funções, conclui-se que, diante das duas soluções apresentadas, a solução que não utiliza vetor auxiliar é a que consome menos operações para se chegar ao resultado esperado.