Randomized Search

João Amaral – nº 65772

***Resumo* – Este relatório encontra-se dividido em cindo capítulos. Sendo o primeiro capítulo direcionado sobre o trabalho e os objectivos. O segundo irá conter informação e explicação de cada algoritmo utilizado. O terceiro capitulo irá conter os teste efectudos que servem como apoio as conclusões tirandas neste relatório. O quarto capitulo, encontra-se a conclusão que contém as conclusões sobre os resultados e dos teste efectuados. Sendo o ultimo capitulo reservado para as referencias.**

***Abstract* - This report is divided into five chapters. Being the first chapter focused on work and goals. The second will contain information and explanation of each algorithm used. The third chapter will contain the actual tests that serve to support the conclusions drawn from this report. The fourth chapter contains the conclusion which contains the conclusions on the results and the tests carried out. Being the last chapter reserved for the references.**

I. INTRODUÇÃO

O trabalho prátrico realizado e descrito neste relatório pretence ao âmbito da disciplina de Algoritmos Avançados que pretence ao Mestrado de Engenharia de Informática da Universidade de Aveiro.

O trabalho proposto pelo docente da disciplina foi efectuar a comparação do desempenho computacional do algoritmo de Procura Linear e do algoritmo de Procura Probabilística ("randomized search") quando aplicados a conjuntos de dados aleatórios.

Será efectuada uma análise da eficiência computacional e das limitaçãos dos algoritmos desenvolvidos.

II. COMPLEXIDADE DOS ALGORITMOS UTILIZADOS

Como foi indicado na introdução foi utilizado um algoritmo de Procura Linear e um de Procura Probabilística neste trabalho. Para efectuar a procura probabilistica foi escolhido o algoritmo de *Monte Carlo*.

A. Algoritmo de Procura Linear

A procura linear é uma pesquisa em listas de modo sequencial ou seja, lê elemento a elemento, de modo que a função de tempo em relação ao número de elementos cresca proporcionalmente.

Apesar de não se saber em concreto o tempo de execução do algoritmo visto que depende do tamanho do ficheiro/lista de elementos, existe situações preferenciais como também situações incómodas.

Numa situação preferencial, é necessário que o elemento que se deseje pesquisar esteja no principio ou até mesmo ser o primeiro elemento na lista, aumentando assim a velocidade do algoritmo durante a pesquisa.

Relativamente a uma situação incómoda, é quando o elemento pretendido se encontra na última posição da lista/*array* que se está a ler. Isto deve-se ao facto da necessidade de percorrer os N elementos até chegar ao elemento pretendido, caso a lista contenha N elementos.

Contudo em média, o elemento é encontrado após (n+1)/2 comparações.

O algoritmo de procura linear é um algoritmo O(n), o que indica que é de complexidade de tempo linear.[1][2]

1) Fluxograma

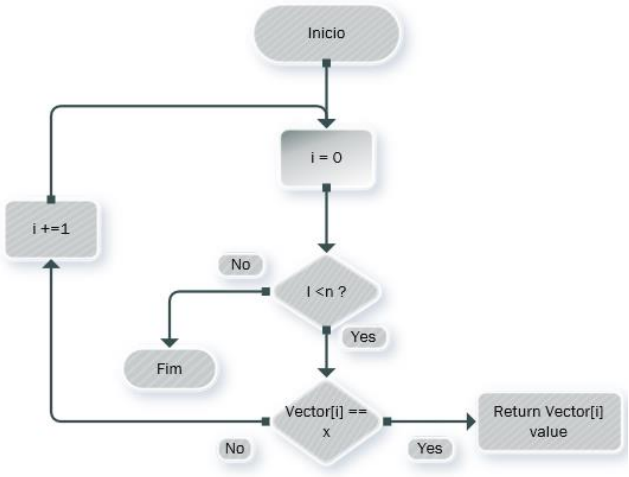

Figura 1 - Fluxograma Procura Linear

2) Algoritmo

```
i=1
while (i<=n and x not equal ai)
    i: i+1
    if i<n, then location:=1
    else location = 0
```

3) Exemplo

Elemento desejado: 4

Casos:

- A = [3, 1, 2, 10, -1] → Irá sair no fim da lista
- B = [4, 2, 5, 8, 6] → Irá sair no princípio da lista
- C = [1, 4, 3, 2] → Irá sair depois do Segundo elemento
- D = [1, 2, 4, 9, 10, 11, 12, 15, 17, 18, 100, 0]

A lista B irá terminar primeiro em comparação com as listas A, C e D, apesar da lista D necessitar de mais tempo para terminar.

4) Imagem do código

```
#Linear search
def linearSearch(numSearch):
    x = 1
    for numero in arrayNumeros:
        if numero == int(numSearch):
            return "Encontrou o elemtno " + numSearch + \
                   " em " + str(x) + " tentativas"
        x+=1
    return "O elemento desejado nao se encontra no ficheiro/array"
```

Figura 2 - Função linearSearch

B. *Algoritmo Monte Carlo*

Monte Carlo é um algoritmo aleatório cujo resultado pode-se encontrar incorrecto com uma certa probabilidade, ou seja, se o número de tentativas for pequeno.

O nome refere-se ao grande casino em Monte Carlo, Mónaco, que é conhecido pelos jogos do azar.

O algoritmo é garantidamente rápido, tendo em consideração que o tempo de execução do algoritmo é fixo (o tempo de execução é O(1)), contudo poderá não encontrar a resposta correcta. O que torna este algoritmo interessante é que a probabilidade de encontrar a resposta correcta pode ser controlada pelo utilizador através de parâmetros no algoritmo.

Em certos casos, a probabilidade do erro poderá ser pequena tendo em conta o tempo de execução eficiente, mas o resultado poderá ser errado com pequena margem de erro probabilístico.

Os exemplos mais comuns do algoritmo *Monte Carlo* são problemas de decisão, ou seja, problemas sim e não ou True ou False.

Para este tipo de problemas, os algoritmos probabilísticos podem ser *true-biased* ou *false-biased*. Se a resposta recebida pelo algoritmo for *true-biased*, o algoritmo é *true* o que se conclui que a resposta é a correcta. Caso contrário, a resposta seria *false-biased* e a resposta dada será errada com a probabilidade dada.

Apesar de ter utlizado o algoritmo *Monte Carlo*, devido à sua similariadade, não posso deixar de referenciar o algoritmo *Las Vegas*. Este algoritmo é igualmente um algoritmo aleatório mas difere visto que produz sempre a resposta correcta independentemente do tempo de execução.

O algoritmo de *Monte Carlo* poderá ser convertido num algoritmo *Las Vegas* sempre que exista um procedimento para verificar se o resultado pretendido pelo algoritmo é de facto correcto. Tendo isto em conta, posso concluir que o algoritmo *Monte Carlo* e o algoritmo *Las Vegas* são muito próximos visto que o de *Las Vegas* é meramente um algoritmo que executa repetitivamente o algoritmo de *Monte Carlo* até que seja produzido um output correcto.

Enquanto o output recebido por um algoritmo determinista é suposto estar sempre correcto, o que não é o caso do algoritmo de *Monte Carlo*. [3][4]

1) *Fluxograma*

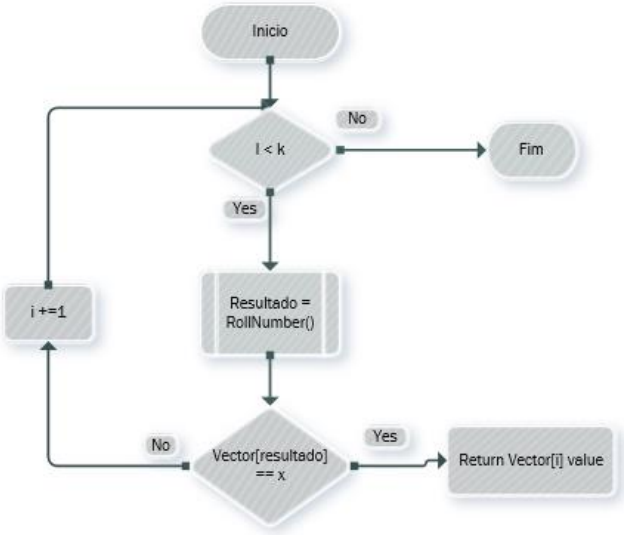

Figura 2 – Fluxograma Procura Probabilística

2) *Algoritmo*

```
i=1
repeat
   randomly select an array element
   i = i +1
until i==k or 'a' is found
```

3) *Exemplo*

Amostra A:

- [5, 3, 9, 2, 5, 1]

Número de tentativas:

- K = 4

Elemento desejado:

- 9

Resultados:

- 1 → A[1] = 3 != 9 Falso

- 0 → A[0] = 5 != 9 Falso
- 4 → A[4] = 5 != 9 Falso
- 2 → A[2] = 9 = 9 O elemento existe na amostra A.

4) Imagem do código

```
#Randomized Search Monte carlo
def randomSearchMonteCarlo(numsearch, tamanhomax):
    x = 1
    while x<tamanhomax:
        result = rollnumber(tamanhomax)
        if(arrayNumeros[result]==int(numsearch)):
            return "Encontrou o valor " + numSearch +\
                   " em " + str(x) + " tentativas"
        x+=1
    return "Falhou em produzir um resutaldo"
```

Figura 4 - Função randomSearchMonteCarlo

III. TESTES

Para os testes foram realizados os seguintes ficheiros:

- aleatorios02500.txt
- aleatorios05000.txt

Nos dois casos foram selecionados 3 numeros, um no principio, no meio e do fim do documento.

1. Ficheiro aleatorios02500.txt

Elemento pretendido: 4497814
Posição: 7º linha
MonteCarlo numero de tentativas: 2256

```
---------------------------
Pesquisa Linear:
Encontrou o elemento 4497814 em 7 tentativas
Tempo de processamento: 0.00008 segundos
---------------------------
Pesquisa random (Monte Carlo):
Encontrou o valor 4497814 em 1967 tentativas
Tempo de processamento: 0.01426 segundos
---------------------------
```

Elemento pretendido: 20436751
Posição: 1270º linha
MonteCarlo numero de tentativas: 2256

```
---------------------------
Pesquisa Linear:
Encontrou o elemento 20436751 em 1270 tentativas
Tempo de processamento: 0.00111 segundos
---------------------------
Pesquisa random (Monte Carlo):
Falhou em produzir um resultado
Tempo de processamento: 0.01206 segundos
---------------------------
```

Elemento pretendido: 10910482
Posição: 2381º linha
MonteCarlo numero de tentativas: 500

```
---------------------------
Pesquisa Linear:
Encontrou o elemento 10910482 em 2381 tentativas
Tempo de processamento: 0.00139 segundos
---------------------------
Pesquisa random (Monte Carlo):
Encontrou o valor 10910482 em 335 tentativas
Tempo de processamento: 0.00176 segundos
---------------------------
```

2. Ficheiro aleatorios05000.txt

Elemento pretendido: 18788528
Posição: 6º linha
MonteCarlo numero de tentativas: 4000

```
---------------------------
Pesquisa Linear:
Encontrou o elemento 18788528 em 6 tentativas
Tempo de processamento: 0.00007 segundos
---------------------------
Pesquisa random (Monte Carlo):
Encontrou o valor 18788528 em 1317 tentativas
Tempo de processamento: 0.00641 segundos
---------------------------
```

Elemento pretendido: 11867303
Posição: 1808º linha
MonteCarlo numero de tentativas: 2256

```
---------------------------
Pesquisa Linear:
Encontrou o elemento 11867303 em 1808 tentativas
Tempo de processamento: 0.00102 segundos
---------------------------
Pesquisa random (Monte Carlo):
Falhou em produzir um resultado
Tempo de processamento: 0.00958 segundos
---------------------------
```

Elemento pretendido: 12545598
Posição: 4583º linha
MonteCarlo numero de tentativas: 500

```
---------------------------
Pesquisa Linear:
Encontrou o elemento 12545598 em 4583 tentativas
Tempo de processamento: 0.00348 segundos
---------------------------
Pesquisa random (Monte Carlo):
Falhou em produzir um resultado
Tempo de processamento: 0.00234 segundos
---------------------------
```

IV. CONCLUSAO

Durante a realização deste trabalho foi possível estudar o algoritmo de procura linear e dois algoritmos de Procura probabilística apesar de ter sido pedido no trabalho o estudo de só um deles.

Durante a pesquisa e a realização do trabalho conclui que no algoritmo *Monte Carlo* quanto maior a probabilidade o utilizador lhe insere, mais favoralmente o algoritmo conseguirá devolver a reposta correcta ao contrário do algoritmo de *Las Vegas*, que irá dar sempre a resposta correcta independentemente do tempo que irá demorar.

Também se pode concluir que a velocidade do algoritmo de procura linear é influenciado pelo tamanho da amostra inserida e da posição do elemento desejado nessa mesma amostra, demorando o tempo máximo caso o elemento desejado se encontrar no fim da amostra ou até mesmo caso não se encontre na amostra desejada visto que seria necessário percorrer todos os elementos para verificar se algum deles seria o desejado.

V. REFERENCES

[1] www.dcc.fc.up.pt/~nam/aulas/0001/pi/slides/slipi0012/node3.html

[2] www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node19.html

[3] Vrajitoru and Knight, "Algorithms and Probabilities"

[4] http://elearning.ua.pt/pluginfile.php/368963/mod_resource/content/0/AA_08_Intro_Randomized_Algorithms_III.pdf