02

Dividindo e conquistando na busca

Transcrição

Temos uma lista de presença com todos os nomes ordenados. Como farei para procurar o meu nome entre os demais? Meu nome é Guilherme. Começo a procurar os nomes pelas letras iniciais: A, B, C, D, F, G... O meu nome estará na letra G! É uma maneira de pesquisar meu nome. O mesmo pode ser feito com nossa lista:

Os alunos foram ordenados pela nota. Se eu perguntar: alguém tirou a nota 5? Vamos verificar: O Jonas tirou 3, o André tirou 4, e a Mariana tirou 5... Temos a nota 5 na lista!

Nova pergunta: tem alguém com a nota 5.7? Vamos verificar. O Jonas tirou 3, o André tirou 4, a Mariana tirou 5 e a Juliana tirou 6.7. Então, não temos um elemento com a nota 5.7.

Em seguida vamos verificar se tem alguém com a nota 9.9? O Jonas tirou 3, o André tirou 4, a Mariana tirou 5 e a Juliana tirou 6.7, o Gui tirou 7, o Carlos tirou 8.5, o Paulo tirou 9, a Lúcia tirou 9.3 e a Ana tirou 10. Não temos a nota 9.9 na lista. Mas esta é forma que você irá procurar se eu perguntar sobre a nota 9.9?

Observe novamente a relação das notas dos alunos. Se eu fizer a pergunta: alguém tirou a nota 8.6? Em que direção começaremos a olhar? Vou perguntar por outra pontuação: tem alguém com a nota 3.3? Por onde você começaria a analisar na lista? Tem alguém com a nota 5.4? Por onde você começaria a analisar? Última pergunta: tem alguém com a nota 6.6? Por onde você começaria a analisar?

Perceba que nós não começamos a analisar no sentido da esquerda para a direita. Por onde começamos a analisar quando procuramos um número dentro de um *array*? Da mesma forma, por onde começamos a analisar quando procuramos alguém em uma lista de nomes ordenados?

Dividindo para buscar

Temos a lista com todas as notas ordenadas:

Poderia ser uma lista de presença com todos os nomes ordenados. Ou um catálogo telefônico com todos os nomes ordenados. Queremos procurar algo em específico dentro de uma lista. Se eu pedir que você procure alguém com a nota 3, em que direção você irá olhar na lista? No começo do *array*. Ou se eu disser "procure alguém com a nota 7", em que direção você irá olhar? Será no começo do *array*... Se eu pedir que você procure alguém com a nota 7, você irá começar a analisar a partir do fim do *array*. Se eu perguntar sobre alguém com a nota 5, você analisará já a partir do meio do *array*.

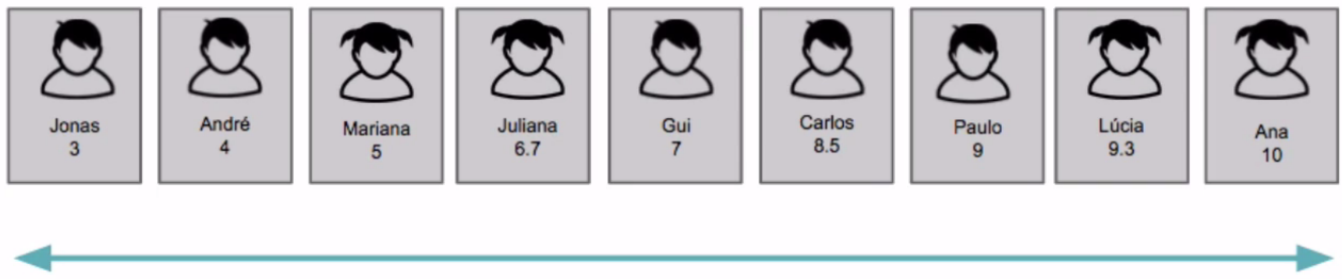
Por que direcionamos o nosso olhar desta maneira? Por que, automaticamente, o nosso olhar se dirige para um determinado lado ou para o meio, dependendo do elemento que estivermos procurando? Porque sabemos que se procurarmos do menor para o maior da lista, e o item que estivermos buscando tiver um valor grande, demoraremos para encontrá-lo. Se procurarmos um número pequeno, e começarmos a analisar a lista do maior elemento para o menor, iremos demorar para encontrá-lo. Nós tentaremos otimizar o processo, e tornaremos a busca mais rápida do que se passássemos por todos os elementos. Nós só passaremos pelos elementos com uma localização aproximada de onde pode estar o que estamos procurando.

Então, quando recebemos uma lista de presença, não começaremos a procurar o Guilherme pelo fim da relação de nomes. Também não iremos começar pelo início. Analisaremos a lista a partir do meio. Ao encontrarmos a posição da letra I, saberemos que o Guilherme estará em uma posição anterior. Se olharmos para a esquerda, veremos os nomes com a letra G, como Gabriel. Se encontramos o Gabriel, o elemento que buscamos estará posicionado à direita. Logo, encontraremos o Guilherme.

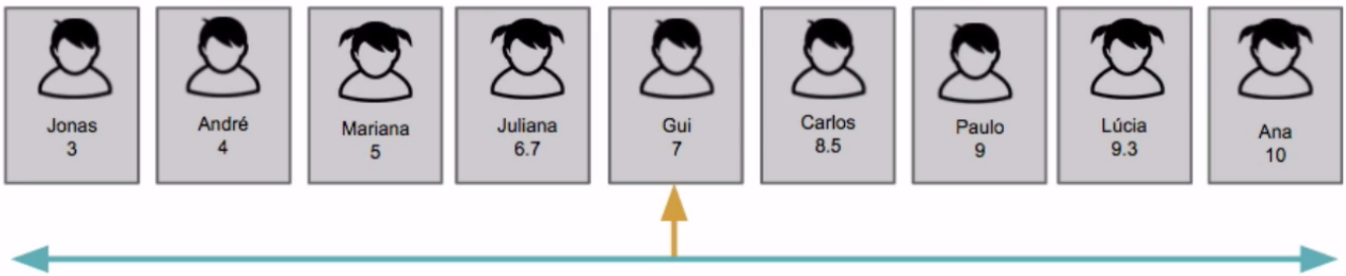
Iremos detectar aproximadamente onde está posicionado o elemento. Em uma lista com 100 mil nomes, não iremos analisar todos os elementos, se o nome que buscamos começa Z. Não somos loucos de fazer isto, nem devemos ensinar esta forma de busca para o computador. Nós o ensinaremos a fazer da mesma maneira como fazemos, uma maneira de procurar mais inteligente.

Porém, não conseguiremos que o computador fique analisando a todo momento se um número é pequeno. Na verdade, seria uma tarefa possível. Mas se ele tivesse que saber identificar "este é um número pequeno, devemos começar pela esquerda. Ou este é um número grande, iremos analisar a partir da direita" seria um processo mais complexo. Nós queremos encontrar uma forma mais simples de especificar para o programa "ao invés de procurar por todos os elementos, busque em apenas uma parte".

Nós já trabalhamos com pedaços de um *array* diversas vezes. Por exemplo, dividimos uma pilha de baralho em dois, para que duas pessoas organizassem cada parte. Nós também já particionamos um *array* em dois, e mandávamos executar uma ordenação em cada um dos pedaços. Se estamos procurando um elemento que esteja entre o primeiro e o último item, o que poderemos fazer? Nós podemos dividir.



Tente dividir a nossa lista e encontrar o elemento 8.5.



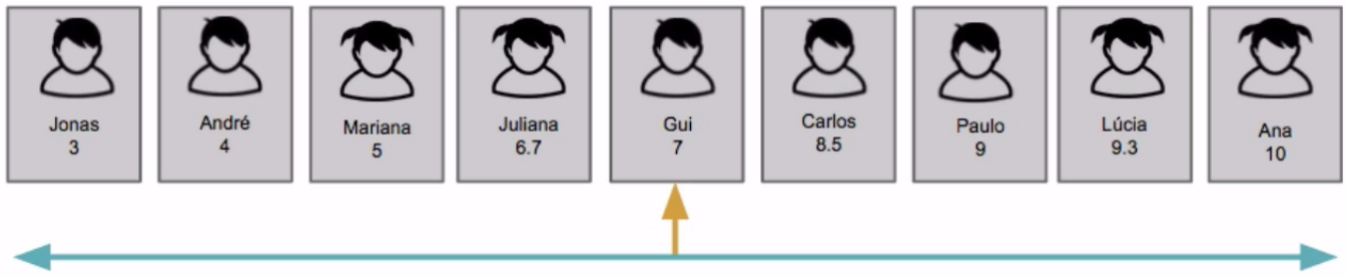
Depois que você fizer a divisão, qual será o processo para encontrar a nota? Em que direção você irá olhar.

E se estivéssemos procurando o elemento 3.4? Após dividir o total de elementos, por onde você começaria analisar? Se procurássemos a nota 17, em que direção você iria olhar na lista? Se procurássemos o elemento 1, em que direção você iria olhar? Você percebeu alguma regra? Qual foi?

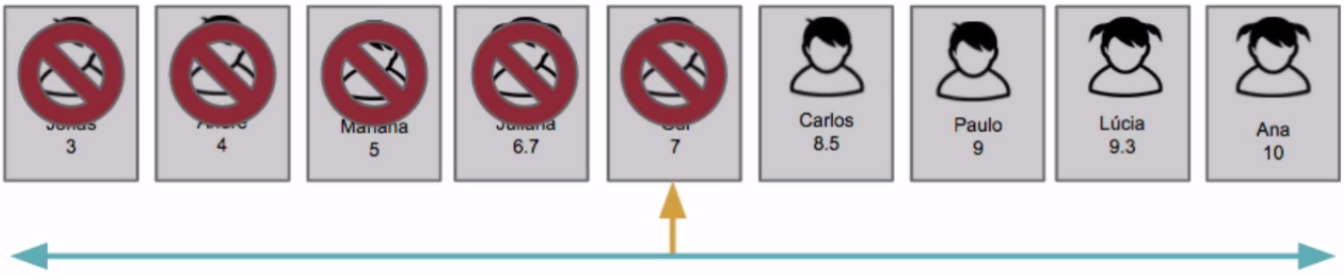
Dividindo dividindo dividindo

Dado um *array* com diversos itens já ordenados, quero saber se um elemento específico está dentro dele. Independentemente se procuro o meu nome ou a nota de um aluno, como realizaremos a busca? Podemos procurar entre todos os elementos da esquerda para direita - o que iria demorar bastante. Ou podemos quebrar a lista em duas partes.

Iremos procurar a nota 9.3. Será que ela está na lista? Por isso, iremos quebrar a lista em dois.



O elemento que ficará no meio será o Gui. Ele tirou 9.3? Não. Inclusive o Guilherme tirou uma nota menor do que 9.3. Se ele é menor, adiantaria procurar entre os elementos posicionados à esquerda? Não. O elemento procurado está posicionado à direita. Podemos ignorar todos os itens à esquerda e o Gui também.



Por que buscaríamos entre os elementos da esquerda? Agora, sobraram apenas os elementos da direita.

Existe o elemento 9.3 entre os que estão posicionado à direita? Sim. Vamos quebrar os elementos que sobraram no meio.

O Paulo é o novo pivô.

Ele tirou 9.3? Não. Ele é menor? Sim. Então, o elemento que estamos buscando estará à direita. Iremos ignorar também os que ficaram à esquerda.

Nós iremos buscar novamente.

Qual elemento no meio? A Lúcia. Ela tirou 9.3? Sim. Encontramos o elemento! Quantas comparações nós fizemos? Comparamos com a nota 7, 9 e 9.3. Fizemos três comparações praticamente. Ao invés de fazermos oito comparações, fizemos apenas três. Parece ter sido um processo mais rápido. Por quê? Porque a cada vez que observamos o *array*, nós eliminávamos metade dos elementos. Não precisamos mais analisar cada elemento e eliminar os menores. Começamos a análise pelo meio. Nós comparamos com um elemento que era menor, depois eliminamos o que não servia. Repetimos o processo e fomos eliminando metade dos elementos em cada parte. Assim encontramos o resultado rapidamente.

Se tivermos **dezesseis** elementos e com uma comparação, conseguirmos eliminar a metade, sobrarão apenas **oito**. Com duas comparações, sobrarão **quatro**. Com três comparações, sobrarão **dois**. Com quatro comparações, sobrará **um**. Se tivermos dezesseis elementos, com quatro comparações chegaremos ao resultado. Se tivermos 1.024 elementos, no pior caso, teremos que passar por cada um deles. Precisaremos de 1.024 comparações. E se usarmos o algoritmo que divide no meio? Com **uma**

comparação, sobrarão 512 elementos, que estarão à direita ou à esquerda. Com **duas** comparações, sobrarão 256. Com **três** comparações, sobrarão 128, que poderão ser da direita ou da esquerda. Com quatro comparações, sobrarão 64. Com **cinco** comparações, sobrarão 32. Com **seis** comparações, sobrarão 16. Com **sete** comparações, sobrarão **oito**. Com **oito** comparações, sobrarão **quatro**. Com **nove** comparações, sobrarão **dois**. E com apenas mais uma comparação, sobrará apenas **um**. Em **10** comparações conseguimos percorrer todos os elementos que nos interessam em um *array* de 1.024 elementos e encontrar (ou não) o item que procurávamos. Dez comparado com 1.024, existe uma grande diferença.

O algoritmo de busca que dividimos o *array* em duas partes, e seguimos procurando apenas no pedaço que nos interessa, é mais rápido do que o que estamos acostumados a utilizar. Nós estamos interessados em implementar este algoritmo agora.

Implementando a busca pela metade

Nós implementamos a busca, mas vimos que varrer o *array* inteiro para procurar um elemento em um que tenha 1.024 itens, por exemplo, é um processo muito lento. Teríamos que passar por todos os elementos. Sendo que podemos estimar se o que estamos buscando está posicionado na direita ou na esquerda. Podemos ir cortando pedaços do *array* e buscar apenas onde temos interesse. Vamos alterar o nosso código para que ele faça isto?

```
private static int busca(Nota[] notas, int de, int de, int ate, double buscando) {
for(int atual = de; atual < ate; atual++) {
    if(notas[atual].getValor() == buscando) {
        return atual;
    }
}
return -1;

}
```

Então, vamos tirar o `for` e irei tentar descobrir onde está o elemento. Mas precisamos ter um base. Será que ele está à esquerda ou à direita do meio? Então, podemos usar o **meio** como base. O meio será o `de` mais o `ate` dividido por 2.

```
private static int busca(Nota[] notas, int de, int de, int ate, double buscando) {
    int meio = (de + ate) / 2;
    }
```

Qual é a nota que está no meio? Será o `notas[meio]`. Adicionaremos mais uma linha:

```
private static int busca(Nota[] notas, int de, int de, int ate, double buscando) {
    int meio = (de + ate) / 2;
    Nota nota = notas[meio];

}
```

Nós iremos querer verificar se a nota do meio é a que estamos procurando. Vamos criar um `if` que verifique se o `buscando` é igual a `nota.getValor`

```
if(buscando == nota.getValor()) {

}
```

Caso seja, iremos ficar satisfeitos em saber que encontramos o resultado. O `return` está na posição do `meio`.

```
if(buscando == nota.getValor()) {
    return meio;
}
```

Mas se o elemento não estiver na posição do `meio`, teremos que seguir procurando. Onde será que ele está? No lado direito ou esquerdo? Se a nota que estamos `buscando` for menor que a nota do meio (`nota.getValor`), então ela estará na esquerda.

```
if(buscando < nota.getValor()) {
    ta na esquerda

}
```

Se `buscando` não for igual ou menor, a nota estará na direita.

```
if(buscando < nota.getValor()) {
    ta na esquerda

}
ta na direita
```

Veremos como ficou o nosso código:

```
private static int busca(Nota[] notas, int de, int de, int ate, double buscando) {
    int meio = (de + ate) / 2;
    Nota nota = notas[meio];
    if(buscando == nota.getValor()) {
        return meio;

    }
    if(buscando < nota.getValor()) {
        ta na esquerda
    }
    ta na direita
}
```

Então, temos algumas possibilidades para estimar a posição do elemento. A nota que buscamos pode estar exatamente no meio. Caso não esteja, ela pode ser menor e estar à esquerda. Ou pode ser maior do que o `meio` e estar à direita. Temos estes três casos.

Como faremos para buscar na esquerda?

```
if(buscando < nota.getValor()) {
    ta na esquerda
}
```

Pediremos que o algoritmo retorne (`return`) busque nas minhas notas, de a posição inicial `de` até o `meio`. Porém, sabemos que neste caso, o `meio` não é a posição certa, então iremos diminuir -1 e eliminar a posição do meio. Nosso objetivo é

descobrir o `buscando`.

```
if(buscando < nota.getValor()) {
    return busca(notas, de, meio -1, buscando)
}
```

E como faremos para buscar apenas na metade da direita? Para buscar na parte da direita, iremos adicionar a seguinte linha:

```
return busca(notas, meio + 1, ate, buscando);
```

O `if` ficará assim:

```
 if(buscando == nota.getValor()) {
        return meio;

    }
    if(buscando < nota.getValor()) {
        return busca(notas, de, meio -1, buscando)
    }
        return busca(notas, meio + 1, ate, buscando);
```

Vale lembrar que temos certeza de que o elemento não está no meio. Se estivesse, teríamos interrompido a busca mais acima.

Vamos testar o código e ver se ele irá encontrar a nota 9.3? Ao rodarmos o programa, ele encontrará o seguinte resultado:

```
Encontrei a nota em 7.

jonas 3.0

andre 4.0

mariana 5.0

juliana 6.7

guilherme 7.0

carlos 8.5

paulo 9.0

lucia 9.3

ana 10.0
```

Ele encontrou a nota na posição 7. A Lúcia, que tirou 9.3 está na posição 7.