MAC0122 - Relatório EP3

Felipe Fernandes de Lima
Nº: 9345321

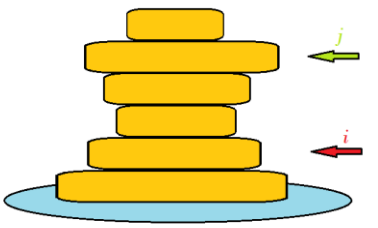
• O ALGORITMO

O algoritmo implementado neste EP foi razoavelmente simples, mas de bom funcionamento. Partindo do fato que cada operação de flip interfere na posição de todos os elementos acima dela, porém mantém as dos outros que estão abaixo, a ideia consiste em ordenar o vetor preenchendo cada posição i começando de 0 (posição inferior da pilha de panquecas) até o topo n com o seu respectivo elemento.

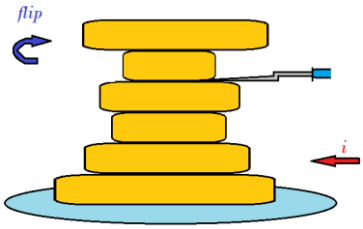
Uma etapa da ordenação é iniciada pela procura do maior elemento do vetor, na qual são percorridas $n - i$ posições, partindo de i até o seu topo. A posição do maior elemento encontrado é guardada e então é feita a operação de ordenação. Cada ordenação consiste em uma operação de até dois flips: se a posição do maior elemento localizado for diferente de i e de $n - 1$ (se o elemento não estiver no topo), o primeiro flip ocorre onde ele está para levá-lo ao topo da pilha, então é feito um novo flip na posição i para que o maior elemento acabe lá. Se o elemento já estiver no topo, então será feito um único flip para levá-lo à posição i.

O funcionamento do algoritmo se justifica pela relação invariante que a cada nova iteração iniciada em i, todas as posições de 0 a i já estão ordenadas e não sofrerão mais alteração, por estarem abaixo de onde ainda ocorrerão novos flips.

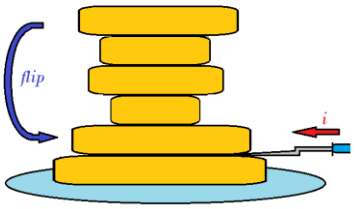
A saída é feita durante as iterações, sempre que um flip é realizado.



O batedor *j* procura pela maior panqueca na pilha



O primeiro flip joga a maior panqueca encontrada para o topo do vetor.



O segundo flip inverte a pilha pela posição *i* para que a panqueca selecionada fique em seu lugar.

• COMPLEXIDADE

```
void ordenaPanquecas(int *pilha, int n){
    int i, j, maior;

    for(i = 0; i < n; i++){
        maior = i;
        for(j = i; j < n; j++){
            if(pilha[j] >= pilha[maior])
                maior = j;
        }
        if(maior != i){
            if(maior != (n-1))
                flip(maior, pilha, n);
            flip(i, pilha, n);
        }
    }
    return;
}
```

Analisando a função *ordenaPanquecas* vemos que a maior parte de sua complexidade de tempo está contida nos dois laços do tipo *for* que percorrem o vetor.

O primeiro deles leva a variável i a percorrer toda a pilha de panquecas à medida que o algoritmo é executado. Portanto, para n panquecas, o código no interior do primeiro laço será executado n vezes.

Já o segundo laço leva a variável j a percorrer toda a pilha a partir da posição i determinada a cada ciclo do primeiro. Portanto, o que ocorre em seu interior será repetido um número decrescente de vezes a cada ciclo, o que significa que suas repetições são dadas por uma Progressão Aritmética de início n, final 1 e razão -1. A complexidade total do código da função será dada pela soma dos termos da PA:

$$S = \frac{(n+1)n}{2} = \frac{n^2+n}{2} \Rightarrow O(n^2)$$

Assim, a complexidade avaliada para *ordenaPanquecas* é de $O(n^2)$.

• NÚMERO DE FLIPS

• MÁXIMOS E MÍNIMOS:

Para o algoritmo implementado pela função *ordenaPanquecas*, no geral, o número de flips realizados vai depender do tamanho da pilha, mas, dependendo da disposição inicial dos tamanhos das panquecas, o total de flips para completar a ordenação pode variar.

Como já foi dito na descrição do algoritmo, durante a ordenação de uma pilha de panquecas, podem ocorrer até dois flips para que cada posição seja ordenada. Portanto, o pior caso, com número máximo de flips, poderia ser quando é necessário fazer, para uma pilha de n panquecas, $2n$ flips. Porém, quando $n-2$ panquecas já estiverem no lugar, para que ainda não esteja ordenado (pois estamos considerando o pior caso) elas deverão estar invertidas. Assim, a maior panqueca não ordenada ocupará o topo da pilha, sendo ordenada em 1 flip, e a panqueca restante não necessitará de flips para ser ordenada. Finalmente, o número máximo de flips para n panquecas será $2n-3$.

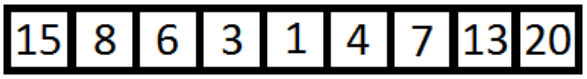
Entretanto, algumas condições verificadas a cada nova iteração (novo valor de i) podem reduzir as inversões. Primeiramente, é testado se a panqueca na posição i já tem o maior tamanho encontrado de i até o final da pilha. Caso isso ocorra, não será necessário fazer inversões, pois a posição estudada já tem o elemento certo. Caso contrário, ainda é possível perguntar se a maior panqueca já não ocupa a posição do topo da pilha, pois, se sim, apenas uma inversão em i será necessária para pôr a maior panqueca da vez em seu lugar. Assim, o exemplo de pilha onde ocorrerá o menor número de flips é aquela que já está ordenada.

• EXEMPLOS:

Exemplo 1:

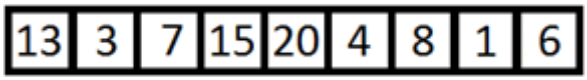
Nesse caso, a cada nova etapa da ordenação, será verificado que a posição i, já possui maior elemento (olhando dela até o topo) e, portanto, nenhum flip será necessário.

Exemplo 2:



Esse não é um exemplo de mínimo ou máximo número de flips, mas também demonstra um caso interessante. Aqui, a cada iteração, o segundo teste se verifica (o maior elemento já está no topo), e é realizado apenas um flip por posição, totalizando $n - 1$ flips. É possível perceber que, nesse caso, o menor elemento está no centro e o vetor cresce a partir dele uma posição à esquerda, depois uma posição à direita. É essa alternância que produz o efeito descrito no número de flips.

Exemplo 3:



Esse é um exemplo de pior caso para este algoritmo. Para essa disposição de elementos será necessário, exceto para os dois últimos elementos, fazer dois flips para cada etapa da ordenação. Ao todo são necessários $2 \text{ x } 9 - 3 = 15$ flips para ordená-lo.

Achar um pior caso para este problema pode ser bastante complicado. Dependendo da complexidade ou engenhosidade do algoritmo de resolução, há poucos casos que precisariam de um número máximo de flips. Como o algoritmo descrito aqui está bem longe de ser o mais engenhoso, ele ainda dá boa margem para achar casos que necessitem de grande número de inversões.

Se, fazendo o caminho contrário, explorássemos todas as possibilidades de "desordenação" para, por exemplo, uma pilha de apenas três panquecas (como no diagrama ao lado), é possível perceber que de todas as $3! = 6$ possibilidades de disposição inicial para o vetor, apenas uma gera o pior caso. Aplicando essa mesma ideia para vetores maiores, da mesma forma, é possível descobrir que o número de casos que levam a um grande número de flips é proporcionalmente pequeno. Porém ele tende a aumentar conforme a simplicidade do algoritmo utilizado.

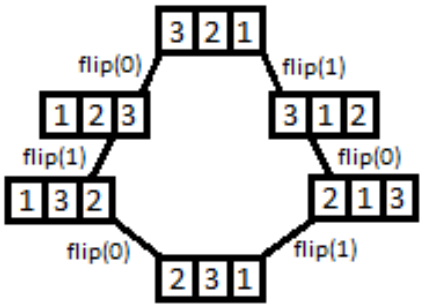


Diagrama de possibilidades para pilha de três panquecas