Introdução a Análise de Algoritmos
2º semestre de 2014 - Turmas 02 e 14
Lista de exercícios 3

1. Implemente um método iterativo (o uso de instruções de repetição está liberado) que recebe um vetor `c` de caracteres, e imprima todos os arranjos de tamanho 2 que podem ser formados com os caracteres presentes em `c`. Por exemplo, para `c` = { '`a`', '`f`' }, seu programa deve imprimir a seguinte saída:

   ```
   aa
   af
   fa
   ff
   ```

2. Idem a questão anterior, mas que imprima todos os arranjos com tamanho igual a 3.

3. Implemente agora um método que recebe um vetor `c` de caracteres, um valor inteiro `n` e imprima todos os arranjos de tamanho `n` que podem ser formados pelos caracteres presentes em `c`. Observe que, para este exercício, um algoritmo iterativo com um número fixo de laços encadeados não é viável (justamente a forma mais direta de resolver as questões 4 e 5). Embora seja possível elaborar um algoritmo iterativo, o mais recomendado é que vocês implementem uma versão recursiva. Apesar de este não ser exatamente um problema de tentativa e erro (pois não estamos interessados em encontrar uma solução dentre o universo de todas as tentativas possíveis), a estrutura do algoritmo recursivo será semelhante à dos algoritmos estudados para os problemas de tentativa e erro uma vez que gerar todos os arranjos possíveis é um problema equivalente a gerar todas as tentativas possíveis a serem analisadas em um problema resolvido através de uma estratégia de tentativa e erro.

4. Empregando a estratégia de tentativa e erro, implemente um método que encontre um caminho ligando um ponto de partida a um ponto de chegada em um mapa. O caminho encontrado não precisa ser necessariamente o mais curto, ou seja, qualquer caminho que ligue a origem ao destino é válido. Assuma que o mapa é representado por uma matriz de caracteres onde o caractere '.' indica caminho livre e o caractere 'X' caminho bloqueado. Assuma ainda que os pontos de partida e destino são dados pelas suas coordenadas na matriz, e que os movimentos podem ser executados apenas nas seguintes direções: para cima, para baixo, para a esquerda ou para a direita.

5. Modifique o método implementado na questão anterior para que ele encontre, obrigatoriamente, o caminho mais curto ligando a origem ao destino.

6. Implemente um método que resolva o problema de encontrar um caminho no mapa usando uma estratégia gulosa. Assim como no exercício 4, não se preocupe em encontrar o caminho mais

curto. Além disso, não se preocupe se seu algoritmo não conseguir encontrar uma solução em todos os casos.

7. Implemente um método recursivo que realiza uma busca sequencial que possua profundidade máxima de recursão igual a $\Theta(n)$. Qual a complexidade, em relação ao tempo de execução, do método implementado no pior caso? E no melhor?

8. Idem à questão anterior, mas desta vez implementando um método cuja profundidade de recursão máxima seja $\Theta(lg\ n)$.

9. Considere o código **BuscaBin.java** no qual é implementado um algoritmo recursivo de busca binária. Determine a equação de recorrência para a função $T(n)$ (função que expressa o tempo de execução, baseado no tamanho da entrada) no pior caso. Em seguida, resolva a equação de recorrência para determinar a complexidade assintótica do algoritmo.

10. O que acontece com a complexidade assintótica, no pior caso, do algoritmo considerado na questão anterior se os valores numéricos estiverem armazenados em uma lista ligada ao invés de um *array*? Considere a implementação de lista ligada fornecida no código **ListaLigada.java** para fazer tal análise.