Algoritmos

Força Bruta recursiva - backtracking

Lista de exercícios

1. Considere uma função que determine a quantidade de ocorrências de um algarismo em um número inteiro não negativo n.

   Exemplo: O algarismo 4 *ocorre* 2 *vezes em* 13412563 4.

   Faça:

   (a) Escreva uma função recursiva que retorne a quantidade de ocorrências de um algarismo x em um inteiro (`long long`) n.
   Dica: *Use o resto da divisão pare recuperar o algarismo*

   (b) Determine a complexidade da função implementada usando a notação *big-Oh*.

2. Sobre a função a seguir:

```
int f1(int n){
  if (n<=0) return 0;
  return (n&1) + f1(n>>1);
}
```

   (a) Determine o que a função faz.

   (b) Determine a complexidade usando a notação *big-Oh*.

   IMPORTANTE*: Para todos os problemas de* ***labirinto*** *descritos a seguir considere o tamanho máximo do labirinto de* 7 *linhas e* 7 *colunas.*

3. Descreva e Escreva, em ANSI C, uma função recursiva que determine a quantidade de diferentes caminhos que encontram a saída de um labirinto. A entrada da labirinto é a célula $(0, 0)$ e a saída é a célula $(L - 1, C - 1)$, onde L é a quantidade de linhas e C é a quantidade de colunas.

4. Descreva e escreva, em ANSI C, uma função recursiva que determine o melhor caminho para encontrar a saída de um labirinto. O melhor caminho é aquele passa pela **menor** quantidade de células.

5. Descreva e escreva, em ANSI C, uma função recursiva que determine se é possível se deslocar entre duas diferentes células em um labirinto. A entrada contém, além da descrição do labirinto, dois pares de valores inteiros, que representam, respectivamente, a célula inicial, e a célula final. No exemplo de entrada a seguir o 0 representa célula livre e 1 representa barreira. O algoritmo para ao encontrar a primeira solução, caso exista. Não há restrições sobre a localização do célula inicial nem da célula final, ou seja, eles podem estar no meio do labirinto.

   Exemplo de entrada:

```
7 7                     /* Quantidade de linhas e colunas */
0 0                     /* Célula inicial */
4 6                     /* Célula final */
0 0 1 1 0 0 0
0 0 1 1 0 1 0
0 0 0 1 0 1 0
1 1 0 0 0 1 0
0 0 0 1 0 1 0
0 0 1 0 0 0 0
0 0 0 0 1 0 0
```

6. Considere um labirinto onde cada célula possui um nível de dificuldade para ser atravessada, representado por um valor D ($0 \leq D \leq 1000$). Células que não podem ser atravessadas possuem o valor -1. Escreva um programa, em ANSI C, que leia um labirinto e determine qual o melhor caminho a ser percorrido.

   Exemplo de entrada:

```
 7   7                          /* Quantidade de linhas e colunas */
 0   0                          /* Célula inicial */
 4   6                          /* Célula final */
 0  10  19  89  99   5   6
 5  10  12   4  -1   1  23
81  32  32   1  -1   1   2
 5  10  12  72  92  84  13
15  15  -1  48  -1  -1   0
 1  81  18   6  -1  25   5
 5  21  12  12  39  -1   1
```

7. Para o problema da **sublista de soma** s, em vetores de números inteiros não negativos:

   (a) Escreva um conjunto de casos de teste, considerando tamanhos de 2 até 30 elementos e construa uma tabela com o tempo de execução para cada tamanho do *array*.

   (b) Faça um gráfico e verifique se a taxa de crescimento corresponde a análise feita.

   (c) Sabendo que o algoritmo usa *backtraking*, descartando soluções não viáveis, construa um conjunto de casos de teste, de tamanhos variando de 10 a 30, onde a função não testa todos os casos.

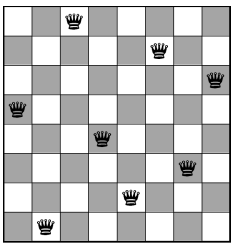
   (d) Implemente um algoritmo de força bruta que determine a **quantidade de diferentes soluções** do problema de sublista de soma s.

8. Com relação ao problema da mochila, escreva, em ANSI C, um programa que mostre, além do valor máximo a ser carregado na mochila, o peso total dos itens carregados.

9. Com relação ao problema da mochila, escreva, em ANSI C, um programa que liste todos os itens que devem ser colocados na mochila para carregar o maior valor possível.

   DICA: Crie uma `'struct'` *para cada item e armazene a solução em um vetor.*

10. **Problema das 8 rainhas**: Em um tabuleiro de xadrez é possível colocar 8 rainhas de tal forma que nenhuma rainha ataque outra rainha, como mostrado no exemplo a seguir.



Faça:

(a) Descreva um algoritmo de força bruta que determine as posições das 8 rainhas sem que elas se ataquem entre sí e determine a complexidade. Descreva o algoritmo que considere **TODAS** as soluções.

(b) Implemente, em ANSI C, uma função usando a técnica de *backtracking* que mostre uma solução do problema das 8 rainhas.

Links para pesquisa sobre o problema das 8 rainhas:

- `https://pt.wikipedia.org/wiki/Problema_das_oito_damas`
- `https://www.ime.usp.br/~coelho/mac0122-2012/aulas/aula19.pdf`
- `http://www.vision.ime.usp.br/~pmiranda/mac122_2s14/aulas/aula20/aula20.html`
- `https://www.ic.unicamp.br/~mc202abcd/03backTrack2pp.pdf`