

Aula Prática 04 –Recursividade

Data de entrega: 30/04/2018 até 23:55.

Atenção: *O que vale é o horário do RunCodes, e não do seu, ou do meu, relógio.*

Procedimento para a entrega:

1. Para cada questão, implemente sua solução.
 2. Para testar suas soluções, implemente um único método *main()*, que poderá conter, por exemplo, um *Menu* de interações e possibilidades do usuário especificar os dados de entrada.
 3. Especifique o *Makefile* com as instruções necessárias para compilação e execução do seu código, sendo que o *Makefile* deve conter também o redirecionamento da entrada e da saída (e.g., `./prog.exe < input.txt > output.txt`.)
 4. Compacte em um único diretório o seu código fonte juntamente com o *Makefile*, o arquivo de entrada e o arquivo de saída usados para testes (e.g., *input.txt*, *output.txt*).
 5. Faça a entrega do arquivo compactado, obrigatoriamente em formato *.zip*, no *RunCodes*, na tarefa correspondente.
- Não utilize caracteres acentuados ou especiais para nomes de pastas, arquivos e na especificação de comentários no código.
 - Implemente em conformidade com boas práticas para reuso e modularização do código.

- Bom trabalho!

1. Implemente uma função recursiva que receba por parâmetro dois valores inteiros x e y e retorne o resultado de x^y (x elevado a y)
2. Implemente uma função recursiva que receba um valor inteiro x e o retorne invertido. Exemplo: se $x = 123$, a função deve retornar 321.
3. Implemente uma função recursiva que receba um valor inteiro em base decimal e o imprima em base binária.
4. Implemente uma função recursiva que retorne o menor elemento em um vetor. Implemente a versão iterativa da sua função. Especifique a complexidade de tempo dos dois algoritmos (usando notação O) implementados. Qual dos algoritmos é mais eficiente? Justifique sua resposta.
5. Implemente uma função recursiva para imprimir todos os números naturais de 0 até N em ordem crescente. Implemente a versão iterativa da sua função. Especifique a complexidade de tempo dos dois algoritmos (usando notação O) implementados. Qual dos algoritmos é mais eficiente? Justifique sua resposta.
6. A pesquisa ou busca binária (em inglês, “binary search algorithm”) é um algoritmo de busca em vetores que segue o paradigma de “divisão e conquista”. Ela assume que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso retornando o índice da posição da chave. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua recursivamente na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua recursivamente na metade inferior do vetor.
 - [a] Implemente a busca binária usando uma função recursiva.
 - [b] Formule e resolva a equação de recorrência do seu algoritmo e defina a ordem de complexidade da sua função.
 - [c] Implemente a versão iterativa da sua função de busca.
 - [d] Especifique a complexidade de tempo dos dois algoritmos (usando notação O) implementados.

Importante: O protótipo de todas as funções podem ser definidos num arquivo `funcoes-recursivas.h(pp)` e suas implementações num arquivo `funcoes-recursivas.cpp`, implementando assim, uma biblioteca de funções recursivas. Não será necessário implementar TADs neste caso. Implementa também um método `main()` que teste a implementação das funções anteriores. Respostas discursivas devem ser entregues num arquivo. pdf, compactado junto com o código fonte.