INE5452 - Tópicos Especiais em Algoritmos II

Terceiro simulado - Questões extra-URI

Entrega: até 14 de outubro de 2020 (até 23:55h via Moodle)

Este Exercício-Programa (EP) é individual. Todos devem entregar as seguintes tarefas (além daquelas disponíveis via sistema externo URI):

- Implementação de algoritmos que apliquem a técnica *Backtracking* para resolver os problemas:
  - *Ordem das Permutações* (mod 26); e
  - *Rainhas Amigas com uma Intriga*.
- Implementação dos algoritmos de Divisão-e-Conquista
  - *Karatsuba*; e
  - *Strassen*.

1 O que deve ser entregue?

O EP pode ser entregue até dia 14 de outubro de 2020 (até 23:55h via Moodle). Cada aluno (aluna) deverá entregar um conjunto de arquivos com:

1. um conjunto de arquivos que implementam uma solução para a *Ordem das Permutações* (mod 26);
2. um *Makefile* para compilação/interpretação/execução da *Ordem das Permutações* (mod 26);
3. um conjunto de arquivos que implementam uma solução para a *Rainhas Amigas com uma Intriga*;
4. um *Makefile* para compilação/interpretação/execução da *Rainhas Amigas com uma Intriga*;
5. um conjunto de arquivos que implementam o algoritmo de *Karatusuba*;
6. um *Makefile* para compilação/interpretação/execução do algoritmo de *Karatsuba*;
7. um conjunto de arquivos que implementam o algoritmo de *Strassen*;
8. um *Makefile* para compilação/interpretação/execução do algoritmo de *Strassen*;

Orientações para construção de um *Makefile*: `https://www.gnu.org/software/make/manual/make.html`

2 Sobre as compilações/interpretações/execuções das tarefas

No momento da execução dos programas desenvolvidos, a presença (virtual) do aluno que o desenvolveu poderá ser necessária para a efetiva avaliação.

3 O que será avaliado na execução/uso do analisador léxico

Abaixo listamos itens importantes com relação a essa avaliação.

- A existência de *Makefiles* (se não existir algum, então a nota das tarefas é 0);
- A execução correta dos *Makefiles* (se algum não executar corretamente, então a nota das tarefas é 0);
- A compilação/interpretação dos programas desenvolvidos (se houver erros de compilação/interpretação, então haverá descontos na nota);

4 Sobre a definição dos problemas para *Backtracking* e os algoritmos de Divisão-e-Conquista

4.1 *Ordem das Permutações* (mod 26)

O problema da *Ordem das Permutações* (mod 26) é definido da seguinte forma. Dado um número inteiro positivo n, fixe duas listas de arranjos: uma crescente $c = [1, 2, \ldots, n]$ e uma decrescente $d = [n, n-1, \ldots, 1]$. Depois, para cada maneira r de arranjar os números $1, 2, \ldots, n$, encontre duas listas de valores C' e D' onde $C'_i = r_i + c_i$ e $D'_i = r_i + d_i$ para todo $i = 1, 2, \ldots n$. Finalmente, encontre $C'' = \prod_{i=1}^{n} C'_i \pmod{26}$ e $D'' = \prod_{i=1}^{n} D'_i \pmod{26}$. Dessa forma, cada maneira r de arranjar n números possui valores $C''(r)$ e $D''(r)$. O problema pede para escrever os valores $C''(r^1), C''(r^2), \ldots C''(r^{n!})$ seguido por $D''(r^1), D''(r^2), \ldots, D''(r^{n!})$ tal que $C''(r^1) \leq C''(r^2) \leq \cdots \leq C''(r^{n!})$ (se algum $C''(r^i) = C''(r^{i+1})$ então, $C''(r^i)$ virá antes de $C''(r^{i+1})$, se $D''(r^i) \leq D''(r^{i+1})$). Por exemplo, para $n = 3$ temos $c = [1, 2, 3]$ e $d = [3, 2, 1]$. Teremos os arranjos $r_1 = [1, 2, 3]$, $r_2 = [1, 3, 2]$, $r_3 = [3, 1, 2]$, $r_4 = [3, 2, 1]$, $r_5 = [2, 1, 3]$, $r_6 = [2, 3, 1]$. Com isso, podemos calcular C', D', C'' e D''. Veja os valores na próxima tabela.

r	C'	D'	C''	D''
1, 2, 3	2, 4, 6	4, 4, 4	22	12
1, 3, 2	2, 5, 5	4, 5, 3	24	8
3, 1, 2	4, 3, 5	6, 3, 3	8	2
3, 2, 1	4, 4, 4	6, 4, 2	12	22
2, 1, 3	3, 3, 6	5, 3, 4	2	8
2, 3, 1	3, 5, 4	5, 5, 2	8	24

Portanto, a solução para este caso é: 2, 8, 8, 12, 22, 24 seguido por 8, 2, 24, 22, 12, 8.

4.2 *Rainhas Amigas com uma Intriga*

No problema das *Rainhas Amigas com uma Intriga*, é dado um inteiro positivo n. Dizemos que uma rainha é *amiga* de outra se elas não se atacam. Existirá uma *intriga* quando uma rainha escolhe onde ficar no tabuleiro. Uma rainha causadora de intriga quer se estabelecer em uma posição do tabuleiro. Ela quer encontrar **todas** as posições de um tabuleiro $n \times n$ de tal forma a minimizar o número de *rainhas amigas* no tabuleiro. Note que todas as rainhas no tabuleiro devem ser amigas, mas uma causa intriga. Por exemplo, para $n = 3$, então única posição onde uma rainha causadora de intriga pode se estabelecer de tal forma a ficar com o tabuleiro só para

ela é a posição $(2, 2)$. Para este caso, o número de rainhas amigas no tabuleiro é 1 (veja Figura 1).

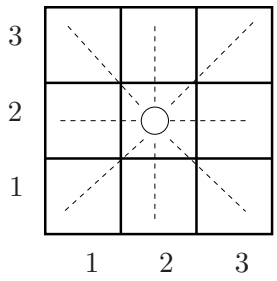

Figura 1: Uma rainha em $(2, 2)$ minimiza o número de rainhas amigas em um tabuleiro 3×3

4.3 Algoritmo de Karatsuba

O algoritmo de Karatsuba deve ser implementado para multiplicar números inteiros positivos grandes (com até milhares de dígitos e na base 10). Você deve usar uma estrutura de dados para guardar cada dígito dos números.

4.4 Algoritmo de Strassen

O algoritmo de Strassen deve ser implementado para multiplicar matrizes quadradas ($n \times n$). Cada entrada da matriz deve guardar um número inteiro. Você pode supor que n sempre será uma potência de 2.

5 Sobre a entrada e a saída dos dados

5.1 *Ordem das Permutações* (mod 26)

A entrada para este problema será um número inteiro n.
Exemplo de entrada:

```
3
```

A saída correspondente terá duas linha com $n!$ números inteiros cada. A primeira contém os valores C'' em ordem, e a segunda contém os valores D'' correspondentes.

Exemplo de saída:

```
2 8 8 12 22 24
8 2 24 22 12 8
```

5.2 *Rainhas Amigas com uma Intriga*

A entrada para este problema será um número inteiro n.
Exemplo de entrada:

```
3
```

A saída correspondente terá todas as posições do tabuleiro que minimiza o número de rainhas amigas no tabuleiro.

Exemplo de saída:
```
(2, 2)
```

5.3 Algoritmo de Karatsuba

A entrada para este problema será um número inteiro positivo n. Nas próximas duas linhas, dois números inteiros positivos com n dígitos
Exemplo de entrada:
```
4
1234
4321
```

A saída correspondente será a multiplicação dos números inteiros dados na entrada.

Exemplo de saída:
```
5332114
```

5.4 Algoritmo de Strassen

A entrada para este problema será um número inteiro positivo n (potência de 2). Nas próximas linhas, duas matrizes A e B, $(n \times n)$, com números inteiros positivos.
Exemplo de entrada:
```
4
1 2 3 4
4 3 2 1
2 3 4 1
3 2 1 4
5 6 7 8
8 7 6 5
6 7 8 5
7 6 5 8
```

A saída será uma matriz $(n \times n)$ que corresponde ao produto AB.

Exemplo de saída:
```
67 65 63 65
63 65 67 65
65 67 69 59
65 63 61 71
```

Observações importantes:

1. Os programas podem ser escritos em C (compatível com compilador gcc versão 5.4.0), C++ (compatível com compilador g++ versão 5.4.0), Java (compatível com compilador javac versão 1.8.0_31) ou Python (compatível com versão 3.7.3), e deve ser compatível com Linux/Unix.
2. Se for desenvolver em Python, então especifique (nos *Makefiles* principalmente) qual é a versão que está usando. Prepare seus *Makefiles* considerando a versão usada.
3. Exercícios-Programas atrasados **não** serão aceitos.
4. Programas com *warning* na compilação terão diminuição da nota.
5. Escreva o seu programa de maneira a destacar a sua formatação.
6. Os programas devem começar com um cabeçalho contendo o seu nome e quais pontos são resolvidos.
7. Coloque comentários em pontos convenientes do programa, e faça uma saída clara.
8. A entrega do Exercício-Programa deverá ser feita no Moodle.
9. **A implementação de Karatsuba vale 2 pontos**.
10. **A implementação de Strassen vale 2 pontos**.
11. **O número total de pontos do simulado 3 (URI e extra-URI) é 13. A pontação excedente será considerada no simulado 4**.
12. O Exercício-Programa é individual. Não copie o programa de outro aluno (aluna), não copie os casos de teste de outro aluno (aluna), não empreste o seu programa para outro aluno (aluna), e tome cuidado para que não copiem seu programa sem a sua permissão. Todos os programas envolvidos em cópias terão nota igual a ZERO.

Bom trabalho!