Atividade 9: Sequência de DeBruijn

Filipe Moreira Mateus

Como Testar:

O código para gerar as sequências de DeBruijn foi escrito em Go. Existem duas formas para testar:

- A forma mais fácil de ver o funcionamento do código é entrando no seguinte link: https://play.golang.org/p/4jGwWdQGeHz e clicando em Run. Essa é uma versão alternativa do código (pasta alt) que não permite entrada do usuário, mas facilita por não ser necessário instalar nada localmente. (É possível testar valores diferentes alterando os valores de `r` e `s` do código). Exemplo de execução:

```
Tamanho da palavra:  2
Número de símbolos:  2
sequencia:  0011
    ----
Tamanho da palavra:  3
Número de símbolos:  2
sequencia:  00011101
    ----
Tamanho da palavra:  4
Número de símbolos:  2
sequencia:  0000100110101111
    ----
Tamanho da palavra:  5
Número de símbolos:  2
sequencia:  00000100011001010011101011011111
    ----
Tamanho da palavra:  2
Número de símbolos:  3
sequencia:  001120221
    ----
Tamanho da palavra:  2
Número de símbolos:  4
sequencia:  0012021311032233
    ----
Tamanho da palavra:  3
Número de símbolos:  3
sequencia:  000100201101202102211121222
    ----
Tamanho da palavra:  4
Número de símbolos:  3
sequencia:
000010102002010002100110111121012200120220112210211022111002222020212012122212112
```

```
    ----
Tamanho da palavra:  3
Número de símbolos:  4
sequencia:
000103300221003011012223023102013031112323312021131320321213332 2
    ----
Tamanho da palavra:  6
Número de símbolos:  2
sequencia:
000001000011000101000111001001011001101001111010101110110111111 0
    ----
Tamanho da palavra:  3
Número de símbolos:  7
sequencia:
004014624105106005234022511403411540412155242035006100015661160161226023512
362525332613204216262053422062101024321103630304333434405413352631501112 4
414151345423055353644245543544365142545603163221255561430633662223104445025
623233002645152003246346465213550331136045224064164265301266356361531201 3
020214466431314532504605654616551665050666065 64
    ----
```

- Para testar uma versão interativa do código, onde é possível entrar com os valores que desejar, é necessário:
  - Instalar a Linguagem Go - https://golang.org/.
  - Executar o seguinte comando para instalar a biblioteca utilizada para encontrar o caminho Euleriano: `go get github.com/yourbasic/graph`
  - Executar seguinte comando para executar o programa: `go run main.go`
  - Exemplo de execução:

```
Insira o tamanho da palavra: 4
Insira o número de símbolos: 2
sequencia:  0000100111101011

Insira o tamanho da palavra: 2
Insira o número de símbolos: 5
sequencia:  0304001022311241321433442

Insira o tamanho da palavra: 3
Insira o número de símbolos: 6
sequencia:
000120403102404131132054144502535530200210321145154241554312122230334055503
410425440035043224201335105051150044110052551231342135202250140151304521 4
124330531422030101112514343532324423315221533325234540235455245344 43
```

Solução

O primeiro passo é encontrar as "palavras" de `s` símbolos e tamanho `r-1`. Isso foi feito usando a seguite função recursiva:

- se `r = 1`, adiciona as `s` strings diferentes de tamanho 1.
- se `r > 1`, para cada palavra de tamanho `r-1` adiciona cada um dos `s` símbolos diferentes ao início dela.

A função `codifica` transforma um número inteiro em um caractere, onde o inteiro representa o "offset" a partir de um caracter inicial, por exemplo: a sequência 0110 se tornaria ABBA se trocarmos '0' por 'A'.

```
func codifica(n int) string {
    return string(rune('0' + n))
}

func geraPalavras(r, s int) []string {
    if r > 1 {
        palavras := geraPalavras(r-1, s)
        res := []string{}
        for _, palavra := range palavras {
            for j := 0; j < s; j++ {
                res = append(res, fmt.Sprintf("%s%s", codifica(j),
palavra))
            }
        }
        return res

    } else if r == 1 {
        res := []string{}
        for i := 0; i < s; i++ {
            res = append(res, codifica(i))
        }
        return res
    }

    return nil
}
```

Depois é necessário gerar grafo que servirá como modelo para o problema. Foi utilizada uma biblioteca especializada em grafos.

- Primeiro é gerado um grafo com o número de vértices igual ao número de subpalavras que foi gerado anteriormente.
- As palavras são percorridas em dois laços de repetição aninhados, se retirando o primeiro símbolo de uma palavra A, e o último símbolo de uma palavra B, criamos uma aresta entre o vértice dado pelo índice da palavra A, e o vértice dado pelo índice da palavra B.

```
func geraGrafoDeBruijn(palavras []string, r int) *graph.Mutable {
    g := graph.New(len(palavras))

    for indiceA, palavraA := range palavras {
        for indiceB, palavraB := range palavras {
```

```
            // se retirando primeiro simbolo de A e último de B temos
mesma sequencia então cria aresta A --> B
            if palavraA[1:] == palavraB[:r-2] {
                g.Add(indiceA, indiceB)
            }
        }
    }

    return g
}
```

Por último geramos o caminho Euleriano usando a biblioteca importada, e exibimos a sequência de DeBruijin resultante. É importante ressaltar que existem várias sequências corretas para cada entrada, e o algoritmo que calcula o caminho Euleriano não é determinístico nesse caso, então podemos ter soluções diferentes para as mesmas entradas.

```
func imprime(caminhoEuleriano []int, palavras []string) {
    var deBrujin string
    for _, indiceVertice := range caminhoEuleriano {
        deBrujin = fmt.Sprintf("%s%s", deBrujin, palavras[indiceVertice]
[:1])
    }
    fmt.Println("sequencia: ", deBrujin[:len(deBrujin)-1])
}
```