ALGORITMO DE COLAÇÃO UNICODE

1. Introdução

O consórcio Unicode publica um padrão denominado **Algoritmo de Colação Unicode** (**UCA**[1]). Este algoritmo recebe como entrada um *string* de caracteres Unicode e uma **tabela de elementos de colação** e tem como saída uma sequência de bytes, denominada **chave de ordenação**. Esta sequência de bytes pode ser comparada, byte a byte, com outra resultante da aplicação do mesmo algoritmo em outro *string*.

A tabela de elementos de colação deve levar em consideração informações sobre localidade (v. **Seção 5.2**), de modo que se possa obter, ao final do processo de comparação, a ordenação de *strings* esperada. Assim, *strings* podem ser ordenados de modos diferentes usando-se diferentes tabelas de elementos de colação. Para levar em consideração a ocorrência de caracteres expansíveis e contraíveis, além de caracteres comuns, é necessária uma tabela de elementos de colação capaz de fazer mapeamentos um-para-um, muitos-para-um e muitos-para-muitos.

Cada **elemento de colação** que aparece numa tabela de elementos de colação consiste de uma sequência de três ou mais pesos inteiros e a ordem com que os pesos aparecem corresponde aos seus níveis. Isto é, o primeiro valor inteiro corresponde ao peso primário (mais forte), o segundo valor corresponde ao peso secundário, e assim por diante, de modo que o último valor corresponde ao peso mais fraco (i.e., o peso de menor influência na ordenação). A ordenação dos elementos de colação é feita da seguinte maneira: se os pesos primários forem diferentes, a ordem dos elementos é definida pela ordem deste peso; se esses pesos forem iguais, a ordem dos pesos secundários é usada, e assim por diante. Um peso com valor `0000` significa que o elemento de colação é ignorável neste nível de peso.

Usando-se apenas três níveis de pesos, dois *strings* podem ser considerados iguais sem que sejam literalmente iguais. Assim, o algoritmo UCA inclui um quarto nível de comparação. Neste nível, o peso atribuído a um caractere é exatamente seu ponto de código. Assim, se dois caracteres são idênticos até o terceiro nível de comparação, seu ponto de código é utilizado como critério de desempate.

Caracteres de controle e formatação são totalmente ignorados na ordenação padrão utilizada pelo algoritmo UCA. Excetuam-se desta regra os caracteres de quebra de linha (`'\n'`) e tabulação horizontal (`'\t'`), que são considerados espaços em branco.

---

[1] O acrônimo UCA é derivado de *Unicode Collation Algorithm*.

2. Elementos de Colação Variáveis

Alguns caracteres, tais como caracteres de espaço, pontuação e muitos símbolos, possuem elementos de colação variáveis. Um **elemento de colação variável** tem pesos pré-definidos, mas também podem usar **pesos alternativos**. Existem quatro opções para tratamento de caracteres com pesos alternativos, que recebem as seguintes denominações: não-ignorável, ignorável, alterado e alterado-reduzido.

Quando um caractere é considerado **não-ignorável**, os mesmos pesos associados a ele na tabela de elementos de colação são utilizados, como no caso de elementos de colação que não são variáveis. Por exemplo, a seguinte lista está ordenada de acordo com este critério[2]:

e-mail *[Hífen: U+002D]*
e—mail *[Travessão: U+2014]*
eleitor
email
exercício

Quando a opção utilizada é **ignorável**, os pesos primário, secundário e terciário do caractere são todos considerados iguais a zero e seu peso quaternário continua sendo igual ao seu ponto de código. Esta escolha faz com que o caractere seja totalmente ignorado se a comparação prosseguir até o terceiro nível de pesos. Se a comparação continuar além deste nível, o ponto de código do caractere (peso quaternário) é usado como critério de desempate. A mesma lista de palavras apresentada acima seria ordenada da maneira a seguir de acordo com esta opção:

eleitor
e–mail *[Hífen]*
email
e—mail *[Travessão]*
exercício

Neste último exemplo, as três variantes de *email* são consideradas iguais até o terceiro nível de comparação, pois, até este nível, o hífen e o travessão são invisíveis ao

[2] Foram acrescentados comentários entre colchetes para ajudar o leitor a distinguir hífen de travessão, já que esta diferença pode não ser prontamente visível. Também, do ponto de vista de estilo de redação, não é recomendável o uso de travessão nesta situação. Mas o propósito aqui é apenas demonstrar como ele seria ordenado, se fosse o caso, nesta situação.

algoritmo de comparação. No quarto nível de comparação, os pontos de código são usados para resultar na ordenação final dos três *strings*[3].

Na opção **alterado**, os três primeiros pesos de um caractere com peso alternativo recebem o valor `0x0000`, como no caso ignorável, mas seu peso primário original passa a ser seu peso quaternário. Neste caso, qualquer caractere que não tenha pesos variáveis recebe o maior valor possível como peso quaternário (i.e., `0xFFFF`). O uso desta opção resulta numa ordenação mais intuitiva quando *strings* são considerados iguais até o terceiro nível de comparação porque assegura-se que todos os *strings* quase iguais, cada um dos quais contendo um caractere ignorável numa mesma posição, aparecem juntos na ordenação final, em vez de separados como na ordenação do último exemplo. Utilizando-se esta opção, os mesmos *strings* dos exemplos anteriores seriam ordenados como:

`eleitor`
`e-mail` *[Hífen]*
`e—mail` *[Travessão]*
`email`
`exercício`

No último exemplo, as três variantes de *email* são consideradas iguais até o terceiro nível de comparação, como ocorreu no exemplo anterior. Mas, agora, no quarto nível de comparação, os pesos (i.e., pontos de código) do hífen e do travessão são comparados com o peso de valor `0xFFFF` atribuído à letra *m*. Assim, as duas versões de *email* com caracteres ignoráveis aparecem juntas.

A opção **alterado-reduzido** é muito parecida com a opção alterado. A diferença é que, na opção alterado-reduzido, os caracteres que não têm pesos alternativos recebem `0x0001` como pesos quaternários (em vez de `0xFFFF`). Esta alteração faz com que os *strings* com caracteres ignoráveis continuem sendo ordenados juntos, como na opção alterado, mas sejam ordenados depois de *strings* sem caracteres ignoráveis considerados iguais até o terceiro nível de comparação. Por exemplo, a mesma lista de *strings* dos exemplos anteriores seria agora ordenada como:

`eleitor`
`email`
`e-mail` *[Hífen]*
`e—mail` *[Travessão]*
`exercício`

[3] Os caracteres comparados no quarto nível e seus respectivos pontos de código são hífen (`U+002D`), travessão (`U+2014`) e letra *m* (`U+006D`). Assim, tem-se a seguinte ordem: o hífen precede a letra *m* que precede o travessão.

3. Chaves de Ordenação

Seguindo o algoritmo UCA, a chave de ordenação criada para cada *string* a ser comparado consiste de uma sequência de pesos. Para obtê-la, examinam-se todos os caracteres do *string* e copiam-se todas as chaves primárias para a chave de ordenação; se um caractere for ignorável (e.g., hífen), seu peso não é copiado para a chave; um caractere expansível (e.g., *æ*) tem mais de um peso acrescentado à chave e caracteres que se contraem (e.g., *ch* em espanhol) têm apenas um peso acrescentado à chave. Depois que todos os pesos primários são acrescentados à chave de ordenação, acrescenta-se a ela um valor de separação (v. adiante) e repete-se o processo para os pesos secundário e terciário de cada caractere que compõe o *string*.

Após a criação das chaves de ordenação de dois *strings*, o processo de ordenação deles resume-se a comparar os pesos constantes nas duas chaves nas respectivas ordens. A comparação de dois pesos é feita do mesmo modo que se comparam números inteiros.

O **valor de separação** que é acrescentado a uma chave de ordenação entre a sequência de pesos primários e a sequência de pesos secundários e entre estes e a sequência de pesos terciários deve ter um valor menor do que o menor de todos os pesos envolvidos. Tipicamente, o separador de pesos é escolhido como sendo zero. Isto faz com que *strings* mais curtos precedam *strings* mais longos quando os caracteres do menor *string* coincidem com o início de um *string* maior (e.g., `"manga"` precede `"mangaba"`). Quando se deseja o efeito contrário (e.g., `"mangaba"` precedendo `"manga"`), utiliza-se um valor maior do que o valor de qualquer peso.

Um conjunto de pesos que faz parte de uma chave de ordenação é denominado **elemento de ordenação** ou **elemento de colação**. Na prática, usualmente, um elemento de colação é obtido por meio da concatenação, num inteiro de 32 bits, de todos os pesos de cada caractere (ou sequência de caracteres de contração)[4]. Utilizando esta abordagem, uma chave de ordenação pode ser obtida por meio de uma única passagem pelo *string*. Por outro lado, o algoritmo de ordenação pode fazer mais de uma passagem, comparando, em cada uma delas, os pesos primários, secundários ou terciários. Em cada passagem apenas um conjunto de pesos torna-se visível por meio de uma operação de mascaramento (v. **Capítulo 13 – Volume I**).

[4] Esta concatenação de pesos pode ser compactada num único inteiro de 32 bits se os pesos utilizarem os intervalos sugeridos pelo padrão Unicode. Na realidade, usando-se estes intervalos, apenas 24 bits são necessários para armazenar cada conjunto de pesos: 16 bits para o peso primário, 4 bits para o peso secundário e 4 bits para o peso terciário. O peso quaternário sugerido pelo padrão Unicode não precisa ser armazenado porque ele corresponde exatamente ao ponto de código do caractere. A compactação dos pesos num único inteiro pode ser obtida usando-se operações de mascaramento (v. **Capítulo 13 – Volume I**).

Uma consideração de natureza prática é que, usualmente, não é necessária a criação de chaves de ordenação antes da comparação de dois *strings*. Isto é, os elementos de colação dos *strings* podem ser criados e imediatamente comparados. Assim, estes elementos de colação só são criados até que seja encontrada uma diferença entre os eles.

4. A Tabela DUCET

O algoritmo UCA apresenta uma tabela de elementos colação padrão denominada **DUCET**[5]. Esta tabela pode ser usada como tal ou pode servir como base para a definição de uma tabela de colação para uma necessidade específica.

A ordenação padrão especificada pela tabela DUCET é inserida num arquivo-texto denominado `allkeys.txt`, que pode ser facilmente encontrado no site do consórcio Unicode na internet (`http://www.unicode.org`).

O arquivo `allkeys.txt` contém uma lista de mapeamentos de pontos de código em sequências de pesos. Cada mapeamento consiste de um ponto de código ou uma sequência de pontos de código, seguido por ponto-e-vírgula e por uma ou mais sequências de pesos. Cada mapeamento ocupa, no arquivo, uma linha que termina com um breve comentário, começando com *#*, que descreve o respectivo caractere sendo mapeado. Cada ponto de código é apresentado como uma sequência de quatro dígitos hexadecimais[6]. Cada elemento de colação (i.e., sequência de pesos) é apresentado por quatro valores hexadecimais de quatro dígitos separados por ponto e envolvidos por colchetes. Estes valores representam respectivamente os pesos primário, secundário, terciário e quaternário do respectivo caractere sendo mapeado. Por exemplo, a linha do arquivo `allkeys.txt` a seguir:

```
0061 ; [.0861.0020.0002.0061] # LATIN SMALL LETTER A
```

faz o mapeamento do caractere Unicode, cujo ponto de código é `U+0061`, com um único elemento de colação contendo os pesos `0x0861`, `0x0020`, `0x0002` e `0x0061`. A linha termina com um comentário informando que o caractere ora mapeado é a letra *a* minúscula. Note que o valor do peso quaternário (`0x0061`) coincide com o ponto de código do caractere.

Se o conteúdo entre colchetes começar com ponto, o elemento de colação é considerado normal. Mas, se em vez de ponto, houver um asterisco, o elemento de colação é variável. Elementos de colação variáveis têm pesos definidos, mas uma dada

[5] DUCET é uma abreviação derivada de *Default Unicode Collation Element Table*.
[6] O padrão Unicode não utiliza a notação da linguagem C para representar valores hexadecimais; i.e., em Unicode estes valores não começam com *0x*.

implementação do algoritmo UCA pode usar pesos alternativos, conforme já foi discutido antes. Por exemplo, a seguinte linha na tabela DUCET:

```
002D  ; [*0221.0020.0002.002D] # HYPHEN-MINUS
```

indica que o hífen é um elemento de colação variável cujo peso primário é `0x0221`. Este peso faz com que esse caractere seja ordenado antes de qualquer letra. Assim, por exemplo, o *string* `"pré-natal"` seria ordenado antes do *string* `"prédio"`, uma vez que o peso primário da letra *d* é `0x1182`. Ainda considerando o mesmo exemplo, se o peso primário do hífen fosse ignorado, usando como peso alternativo `0x0000`, obter-se-ía a ordenação usualmente encontrada em dicionários.

Sequências de contração e expansão são incluídas no arquivo `allkeys.txt` para assegurar que formas de normalização diferentes de um mesmo caractere ou sequência de caracteres sejam tratadas da mesma maneira.

Uma entrada de caractere expansível no arquivo `allkeys.txt` é semelhante ao seguinte[7]:

```
2474 ; [*027A.0020.0004.2474][.0A0C.0020.0004.2474]
       [*027B.0020.001F.2474] # PARENTHESIZED DIGIT ONE; COMPATSEQ
```

Esta entrada informa que o caractere ⑴, com ponto de código `U+2474`, é ordenado como se fosse composto de abre-parênteses, seguido do número um e terminado com fecha-parênteses (i.e., tal qual o formato do caractere sugere). Os pesos deste caractere são os mesmos dos referidos caracteres da composição, exceto a partir do terceiro nível de cada elemento de colação, o que assegura que o caractere ⑴ não seja o mesmo que o *string* `"(1)"` composto de três caracteres.

Os valores dos pesos na tabela DUCET não são normativos; i.e, quaisquer pesos podem ser utilizados, desde que eles façam com que os *strings* sejam ordenados na mesma ordem que seriam se utilizassem os pesos propostos nesta tabela.

5. Normalização

[7] Algumas vezes, um comentário termina com um rótulo informativo que tem como objetivo identificar a respectiva entrada na tabela. Por exemplo, o rótulo *COMPATSEQ* identifica uma um caractere associado a uma sequência de elementos de colação (i.e., um caractere expansível).

Em Unicode, muitos caracteres e sequências de caracteres possuem múltiplas representações. Um dos requisitos para aderência ao padrão Unicode é que todas as representações de um caractere sejam tratadas da mesma maneira. Por exemplo, a letra *u* pode ser representado como:

```
U+00FC # LATIN SMALL LETTER U WITH DIAERESIS
```

Ou como a combinação de caracteres:

```
U+0075 # LATIN SMALL LETTER U
U+0308 # COMBINING DIAERESIS
```

Assim, um algoritmo de busca ou ordenação que siga o padrão Unicode deve considerar `U+00FC` e a combinação de `U+0075` com `U+0308` como iguais. Para obter este efeito, os *strings* a ser comparados são convertidos numa **forma normalizada**.

Tipicamente, utiliza-se a **Forma Normalizada D**, na qual cada caractere é totalmente decomposto na representação equivalente com o maior número de componentes[8]. Uma razão para a escolha desta forma normalizada, em detrimento a outras sugeridas pelo padrão Unicode, é que, usualmente, é mais simples tratar acentos isoladamente como caracteres, que são ignoráveis no primeiro nível de comparação, do que como parte integrante de outras letras. Usando-se esta última opção, pode ser preciso considerar mais caracteres como expansíveis (v. **Seção 7.7.3**).

No algoritmo UCA, o processo de normalização é efetuado durante o processo de mapeamento de *strings* em elementos de colação.

6. Resumo do Algoritmo UCA

Em resumo, o algoritmo UCA apresenta as seguintes características:

- Quatro níveis de pesos são usados, com os pesos no quarto nível correspondendo a pontos de código.
- É requerido suporte para caracteres ignoráveis, caracteres com sequências de contração, caracteres com expansão e uso de pesos variáveis.
- É sugerido suporte para ordenação de acentos franceses.
- É requerido suporte para o uso de pesos alternativos.

[8] Em alguns scripts, há letras com um número muito maior de representações equivalentes do que o exemplo apresentado.

- Uma ordenação padrão (DUCET) é especificada para todos os caracteres do repertório Unicode. Na ausência de alterações desta ordenação, todas as implementações do algoritmo devem ordenar qualquer conjunto de *strings* da mesma maneira.
- Para qualquer alteração feita na ordenação padrão, qualquer implementação do algoritmo deve apresentar o mesmo resultado para um mesmo conjunto de *strings*.

A comparação de *strings* de acordo com o algoritmo UCA pode ser resumida nos seguintes passos:

1. **Decomposição e reordenação de caracteres**. Neste passo, os *strings* são colocados na Forma Normalizada D e, se necessário, reordenados[9].
2. **Mapeamento de *strings* normalizados e reordenados em elementos de colação**. Este passo pode envolver o mapeamento de caracteres em múltiplos elementos de colação, mapeamento de sequências de caracteres num único elemento de colação ou mapeamento de sequências de caracteres em sequências de elementos de colação.
3. **Criação das chaves de ordenação**. Aqui, os pesos são reorganizados de modo que todos o pesos primários apareçam primeiro, seguidos por um valor de separação, seguido de todos os pesos secundários, e assim por diante[10].
4. **Comparação das chaves de ordenação**. Esta comparação é realizada naturalmente byte a byte.

7. Busca

Além de ordenação, outra operação importante na qual colação é essencial é busca. Quando se faz uma operação de busca na qual se tenta encontrar um casamento exato entre *strings*, a comparação pode ser feita utilizando pontos de código. Por outro lado, quando se buscam *strings* equivalentes (e.g., quando se ignoram distinções entre maiúsculas e minúsculas), o uso da abordagem delineada pelo algoritmo UCA é mais apropriada. Isto é, neste caso, em vez de comparar pontos de código, constroem-se

[9] Reordenação é um processo necessário em algumas línguas asiáticas. Maiores considerações sobre este assunto específico estão além do escopo deste livro e pode ser encontrada na bibliografia apresentada ao final deste volume.

[10] Se ordenação francesa de caracteres acentuados for desejada, a inversão de pesos secundários para produzir a ordem desejada é feita neste passo.

chaves de ordenação e comparam-se os respectivos elementos de colação. A exatidão do casamento entre os *strings* depende dos níveis em que ocorrem as comparações.

Utilizando essa última abordagem, para ignorar, por exemplo, diferenças entre maiúsculas e minúsculas, basta não levar em consideração os pesos terciários das chaves de ordenação. Por outro lado, se for desejado ignorar diferenças produzidas por acentuação, ignoram-se os pesos secundários.

Uma complicação no processo de busca descrito até aqui é que, se for encontrado um casamento entre elementos de colação dos dois *strings*, ele só é considerada bem sucedido se começar e terminar nos limites de um agrupamento de grafemas. Por exemplo, se o *string* `"café"` estiver presente no texto em sua forma decomposta (i.e., se a letra *é* aparecer como a letra *e* seguida de acento agudo), `"cafe"` casará com a porção inicial de `"café"`. Mas, se diferenças causadas por acentuação forem importantes, `"cafe"` não deve casar com `"café"` porque o possível casamento termina no meio de um agrupamento de grafema; i.e., o casamento inclui a letra *e*, mas não inclui o acento do *string* `"café"`.

8. Exemplo

Para ilustrar o processo de comparação de *strings*, considere como exemplo a comparação dos *strings* `"Maca"` e `"maçã"`. Este processo começa com os *strings* em seus estados originais e o quadro a seguir apresenta-os com seus respectivos pontos de código.

```
Maca  0x004D 0x0061 0x0063 0x0061
maçã  0x006D 0x0061 0x00E7 0x00E3
```

O primeiro passo no processo de comparação consiste em mapear os caracteres dos dois *strings* em suas respectivas formas normalizadas. O resultado é apresentado no quadro a seguir[11]:

```
Maca  0x004D 0x0061 0x0063 0x0061
maçã  0x006D 0x0061 0x0063 0x0327 0x0061 0x0303
```

Em seguida, os pontos de código são mapeados em elementos de colação, como mostra o quadro a seguir[12]:

[11] Observe que apenas as letras ç e ã são expandidas na passagem para a Forma Normalizada D.
[12] Os pesos foram obtidos da tabela DUCET e são separados por pontos.

```
Maca  [0x1291.0x0020.0x0008.0x004D]
      [0x1141.0x0020.0x0002.0x0061]
      [0x116F.0x0020.0x0002.0x0063]
      [0x1141.0x0020.0x0002.0x0061]

maçã  [0x1291.0x0020.0x0002.0x006D]
      [0x1141.0x0020.0x0002.0x0061]
      [0x116F.0x0020.0x0002.0x0063]
      [0x0000.0x0056.0x0002.0x0327]
      [0x1141.0x0020.0x0002.0x0061]
      [0x0000.0x004E.0x0002.0x0303]
```

Então, os pesos de cada caractere são reunidos, os valores ignoráveis são descartados e separadores de pesos são inseridos entre os conjuntos de pesos[13]. O resultado é apresentado no quadro a seguir:

```
Maca  0x1291 0x1141 0x116F 0x1141 0x0000
      0x0020 0x0020 0x0020 0x0020 0x0000
      0x0008 0x0002 0x0002 0x0002 0x0000
      0x004D 0x0061 0x0063 0x0061

maçã  0x1291 0x1141 0x116F 0x1141 0x0000
      0x0020 0x0020 0x0020 0x0056 0x0020 0x004E 0x0000
      0x0002 0x0002 0x0002 0x0002 0x0002 0x0002 0x0000
      0x006D 0x0061 0x0063 0x0327 0x0061 0x0303
```

Finalmente, os elementos de colação são comparados um a um. Os dois primeiros elementos de colação que apresentam diferença são apresentados em negrito e sublinhados no quadro a seguir.

[13] O valor utilizado para separadores de pesos foi `0x0000`.

```
Maca  0x1291 0x1141 0x116F 0x1141 0x0000
      0x0020 0x0020 0x0020 0x0020 0x0000
      0x0008 0x0002 0x0002 0x0002 0x0000
      0x004D 0x0061 0x0063 0x0061

maçã  0x1291 0x1141 0x116F 0x1141 0x0000
      0x0020 0x0020 0x0020 0x0056 0x0020 0x004E 0x0000
      0x0002 0x0002 0x0002 0x0002 0x0002 0x0002 0x0000
      0x006D 0x0061 0x0063 0x0327 0x0061 0x0303
```

Conforme mostra o último quadro, os primeiros elementos de colação diferentes são `0x0020`, no *string* `"Maca"`, e `0x0056`, no *string* `"maçã"`. Como, evidentemente, `0x0020` é menor do que `0x0056`, tem-se que `"Maca"` deve preceder `"maçã"`.

Concluindo o exemplo acima, note que os dois *strings* são considerados iguais no nível primário de comparação e que, devido à normalização, o *string* `"maçã"` tem dois conjuntos de pesos a mais que o *string* `"Maca"` no início do processo. Mas, os pesos primários dos caracteres til e cedilha são ignoráveis no primeiro nível, de modo que os dois *strings* passam a ter o mesmo conjunto de pesos primários. A primeira diferença encontrada (i.e., os elementos de colação `0x0020` e `0x0056`) correspondem, respectivamente, aos pesos secundários do segundo *a* de `"Maca"` e do cedilha de `"maçã"`. A diferença entre a letra maiúscula inicial de `"Maca"` e a letra minúscula inicial de `"maçã"` só aparece em seus respectivos pesos terciários; isto é, *M* tem peso terciário `0x0008`, enquanto que *m* tem peso terciário `0x0002`. Se os *strings* fossem essencialmente os mesmos e diferissem apenas no uso de maiúsculas e minúsculas, esta seria a primeira diferença encontrada e o *string* que começasse com letra maiúscula precederia o outro.

O processo de comparação em quatro passos apresentado no exemplo acima tem propósito ilustrativo e didático. Na prática, estes passos são intercalados de modo que uma diferença primária no primeiro caractere de cada *string* seja encontrada logo no início do processo em vez de após executar muitas tarefas que serão, enfim, desnecessárias.