Algoritmos e Estruturas de Dados I
Prof. Alexandre Freire
Segundo Exercício Programa
Data de Entrega: 22 de Outubro de 2018

Compressor de Ziv e Lempel

Em 1977 e 1978, Jacob Ziv e Abraham Lempel propuseram dois métodos de compressão de arquivos inovadores. Existem diversas variantes desses métodos implementados em programas conhecidos de compressão, como o zip/unzip, o gzip, o compress e outros compressores do UNIX.

Neste EP, estamos interessados no método proposto em 1978, que é conhecido como LZ78. Ilustramos a técnica básica deste método por meio de um exemplo. Considere um arquivo binário com o seguinte conteúdo:

000101110001000000010011

Basicamente, a idéia é particionar a cadeia de bits do arquivo em trechos, de maneira que cada trecho corresponda à menor subsequência que não apareceu entre os trechos encontrados anteriormente. De acordo com esta definição, o conteúdo acima seria particionado da seguinte maneira:

0|00|1|01|11|000|10|0000|001|0011|

O método LZ78 é composto por dois algoritmos: o compressor e o descompressor.

O compressor

Para a compressão, determina-se os trechos da cadeia de bits conforme a regra descrita acima, e estes são numerados sequencialmente. Por conveniência, define-se um trecho vazio, que ocorre no início da cadeia e cujo índice é 0. No exemplo acima, o trecho “0” recebe índice 1, o trecho “00” recebe índice 2, e assim por diante. Cada trecho T é então associado a um par, composto pelo índice do trecho dentre os anteriores que corresponde ao prefixo de T, e o bit (0 ou 1) que finaliza T. Na tabela abaixo são mostrados os pares associados a cada trecho do exemplo ilustrado acima.

Índice do Trecho	1	2	3	4	5	6	7	8	9	10
Subsequência	“0”	“00”	“1”	“01”	“11”	“000”	“10”	“0000”	“001”	“ 0011”
Prefixo	Vazio	“0”	Vazio	“0”	“1”	“00”	“1”	“000”	“00”	“001”
(Índice do Prefixo, Bit Final)	(0, 0)	(1, 0)	(0, 1)	(1, 1)	(3, 1)	(2, 0)	(3, 0)	(6, 0)	(2, 1)	(9, 1)

Basicamente, a sequência de pares obtida é a “codificação” da cadeia de bits (obs.: a codificação deve ser convertida para a base binária, conforme será explicado na próxima seção). Note que, à medida que avançamos na cadeia, os pares representarão subsequências cada vez mais longas, o que tende a tornar a codificação mais compacta do que a cadeia que a gerou, pois os prefixos dos trechos serão substituídos pelos seus respectivos índices. Apesar disso, pode ocorrer do arquivo compactado ficar maior que o original (geralmente, ocorre quando o arquivo original é muito pequeno).

Representação binária da codificação

Para produzir o arquivo binário de saída (que corresponde ao arquivo de entrada compactado), deve-se converter os índices de prefixo de cada par de base decimal para base binária. O número de bits usados ao se escrever o índice do prefixo do *n*-ésimo par em binário corresponde ao número de bits necessários para representar o número *n-1* (que é o maior valor que o índice do prefixo do *n*-ésimo par pode assumir) em binário. O *n*-ésimo par possuirá, portanto, *n* bits, sendo *n-1* bits para representar o índice do prefixo e *1* bit para representar o último bit do trecho correspondente. Veja o exemplo mostrado na tabela abaixo.

Índice do Trecho	1	2	3	4	5	6	7	8	9	10
Par	(0, 0)	(1, 0)	(0, 1)	(1, 1)	(3, 1)	(2, 0)	(3, 0)	(6, 0)	(2, 1)	(9, 1)
Núm. de Bits Necessários	0+1	1+1	2+1	2+1	3+1	3+1	3+1	3+1	4+1	4+1
Par Codificado	“0”	“10”	“001”	“011”	“0111”	“0100”	“0110”	“1100”	“00101”	“10011”

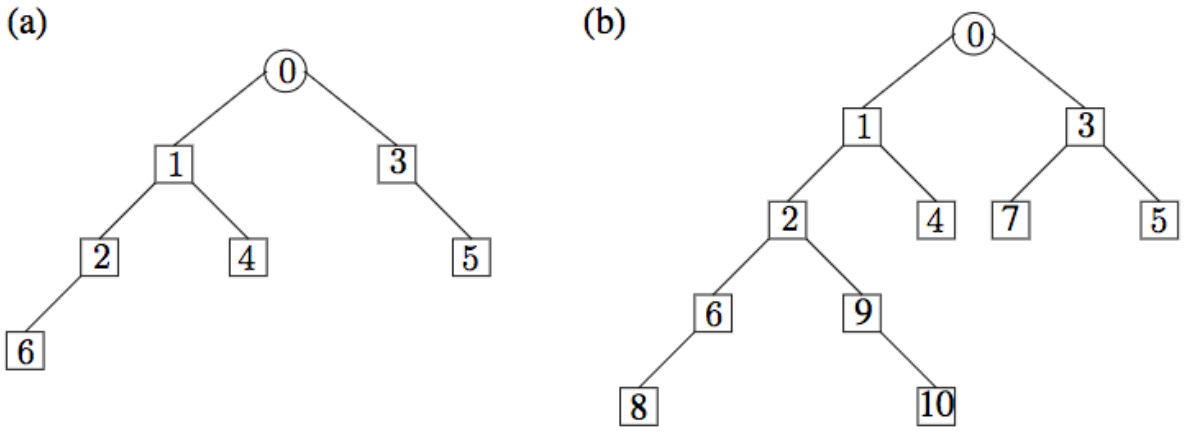
Ou seja, a codificação da cadeia de bits do exemplo fica assim:

01000101101110100011011000010110011

Estrutura de dados

Para implementar o algoritmo de codificação, será utilizada uma variante da estrutura de dados conhecida como *trie* (que se pronuncia como a palavra try em inglês, apesar do nome ser derivado da palavra *retrieval*). Uma *trie* é uma árvore (no nosso caso, binária) na qual, em cada subárvore estão armazenadas as chaves que têm um certo prefixo comum.

Cada nó da *trie* estará associado a um dos trechos da cadeia de bits do arquivo de entrada e será rotulado com o índice do respectivo trecho. A *trie* será construída de forma incremental. Mais precisamente, inicialmente a trie terá um único nó, representando o trecho inicial, que corresponde à subsequência vazia. Para representar o próximo trecho da cadeia, um novo nó é adicionado à *trie,* conforme será detalhado abaixo. Veja a *trie* resultante da inserção dos 6 primeiros trechos (11 primeiros bits) da cadeia do exemplo no item (a) da figura abaixo.



Observe que, percorrendo-se o caminho da raiz até um nó, determina-se os bits do trecho cujo índice o rotula concatenando-se os “rótulos” das arestas percorridas, onde o “rótulo” de uma aresta é 0 ou 1, dependendo se ela vai para a esquerda ou para a direita na árvore, respectivamente. Por exemplo: o trecho 4 corresponde à subsequência “01”, já que para alcançar o nó de rótulo 4 a partir da raiz percorre-se uma aresta à esquerda e depois uma à direita. O nó rotulado por 6 corresponde à subsequência “000” (3 arestas à esquerda, a partir da raiz), e assim por diante.

Suponha que foram lidos os bits correspondentes aos *i* primeiros trechos da cadeia. A partir da *trie* gerada na *i*-ésima iteração, é fácil determinar o final do próximo trecho da cadeia de bits: basta percorrer a *trie* usando os próximos bits da cadeia para decidir qual dos ramos da *trie* seguir; se chegarmos a um bit para o qual não existe um ramo correspondente, então esse é o último bit do próximo trecho. Adiciona-se, então, um novo nó à *trie* para representar o *(i+1)*-ésimo trecho.

No exemplo acima, percorrendo a *trie a* partir da raiz, começando com o 12° bit da cadeia (início do 7° trecho), chega-se ao nó rotulado por 3 e não há nesse nó uma aresta para a esquerda. Portanto, inserimos um novo nó (de rótulo 7) na *trie* como filho esquerdo do nó rotulado por 3. Repetindo o processo para os demais bits da cadeia, chegamos à *trie* exibida no item (b) da figura acima.

A trie deve ser construída à medida que a cadeia de bits é percorrida, ao mesmo tempo em que se constrói a sequência de pares da codifição (ou diretamente o arquivo binário resultante da compactação).

Um detalhe importante

É preciso tomar um cuidado especial no final da cadeia a ser comprimida, do contrário o trecho final da cadeia pode vir a ser o prefixo de um trecho anterior.

Dica: Preencha o final da cadeia com o mesmo bit (1, por exemplo) até que o trecho final não seja um prefixo de trechos anteriores. Reserve o primeiro byte do arquivo compactado para indicar quantos bits foram adicionados no final da cadeia. Ou seja, a cadeia codificada iniciará a partir do 9º bit e o último trecho, após decodificado, deve ser truncado de forma a desprezar os últimos *k* bits, onde *k* corresponde ao número contido no primeiro byte do arquivo compactado.

O descompressor

Uma vez que o número de bits para representar cada par da cadeia codificada está bem definido ($\log_2 n + 1$ bits para o *n*-ésimo par), o descompressor consegue facilmente obter os pares a partir do arquivo binário compactado. A partir dos pares, é possível reconstruir a *trie* e, por conseguinte, obter a cadeia de bits original.

Como um teste, veja se você consegue descomprimir a seguinte cadeia codificada usando o esquema acima. Após decodificá-la, interprete cada 8 bits da sequência como o código US ASCII de um caracter e leia a mensagem até o símbolo $ aparecer, que indica que a mensagem terminou.

```
00101011110000011010101001000001110100110011011001110110101010110010111001100
10010001100001001010101101101001000110100000101100100110011011000001110011101
01001011000000000110110000110101111001111110010000111000010000001010100
```

O que deve ser feito

Você deve escrever um programa que comprime e descomprime arquivos de qualquer tipo (texto, imagem, som, etc) de acordo com o algoritmo LZ78. O nome do arquivo a ser comprimido/descomprimido deve ser dado na linha de comando. A opção `-x` indica que o arquivo dado deve ser descomprimido, e a ausência dela indica que o arquivo dado deve ser comprimido. Ao comprimir um arquivo de nome `abacaxi.xxx`, seu

programa deve gerar um arquivo de nome `abacaxi.xxx.cod`. Quando a opção `-x` é dada, o arquivo dado na linha de comando deve ter a extensão `.cod`, por exemplo, `abacaxi.xxx.cod`, e a saída do seu programa será um arquivo cujo nome substitui a extensão `.cod` por `.dec`. Ou seja, para o arquivo `abacaxi.xxx.cod`, o arquivo gerado com a opção `-x` seria `abacaxi.xxx.dec`. Repare que seu programa não deve destruir os arquivos dados como entrada.

Exemplo:

O comando `EP2 carta.tex` deve gerar um arquivo de nome `carta.tex.cod`, enquanto que o comando `EP2 -x carta.tex.cod` deve gerar um arquivo chamado `carta.tex.dec`.

Instruções adicionais

Submeta seu EP até às 23:30 do dia 22 de outubro através do site edisciplinas.usp.br (o sistema aceitará submissões até às 23:55, mas caso você tente submeter em um horário muito próximo ao encerramento e tiver problemas, fica por sua conta e risco...). O arquivo submetido deve estar no formato zip e deve conter apenas os códigos-fonte (somente os arquivos .cpp e .h, não adicione os executáveis). Não será permitido utilizar nenhuma API que implemente as estruturas de dados necessárias. Caso haja casos de plágio, todos os envolvidos serão convocados a prestar contas e, dependendo da gravidade do caso, podem ser sumariamente reprovados na disciplina.

Bom Trabalho!