ESPALHAMENTO (OU *HASHING*)

PAULO JOSÉ DA SILVA E SILVA

1. Introdução

Nas aulas anteriores nós estudamos árvores e mostramos que estas estruturas de dados podem ser usadas de forma muito eficiente para armazenar um grande volume de informações, garantido um acesso rápido a elas.

Por outro lado, é também muito comum a necessidade de guardar uma quantidade razoavelmente pequena de dados para os quais se quer acesso muito eficiente. É neste contexto onde se inserem as tabelas de espalhamento.

2. Espalhamento

Imagine que desejamos guardar uma quantidade pequena/média de dados que deve ser repetidamente recuperada. Consideramos também que esta informação é identificada por uma chave natural.

Se o conjunto das possíveis chaves fosse pequeno, digamos de $0, \ldots, m-1$, uma forma extremamente eficiente para resolver o problema proposto seria criação de um vetor de m posições e o armazenamento das informações indexadas pela chave.

Porém, muitas vezes, apesar de sabermos que a quantidade de dados não será grande, o conjunto das chaves possíveis pode ser enorme, impedindo o uso da estratégia inocente apresentada acima. Neste caso, lançamos mão de uma função de espalhamento.

Uma *função de espalhamento,* que denotaremos por h, mapeia o espaço das possíveis chaves em um conjunto na forma $\{0, 1, 2, \ldots, m-1\}$ para m "não muito grande". Assim, dada uma chave k, podemos usar $h(k)$ como índice para um vetor que servirá para armazenar as informações de forma semelhante à descrita anteriormente. Claramente a escolha da função de espalhamento é fundamental para que essa estratégia seja razoável.

Contudo, por melhor que seja a a função de espalhamento escolhida, como o espaço das chaves é em geral bem maior que m, deve-se esperar que ela provoque *colisões.* Isto é, devem existir chaves diferentes com mesmo valor de espalhamento. Para contornar este problema podemos usar uma lista ligada. A idéia é a seguinte: cria-se um vetor de listas ligadas inicialmente vazias e usa-se a função de espalhamento para decidir em qual lista a informação desejada deve ser inserida, buscada ou removida. Veja Figura 1

Mas como "projetar" boas funções de espalhamento?

3. Funções de espalhamento

A escolha da função de espalhamento é crucial, como vocês já devem ter notado. Elas devem ser rápidas de calcular e, ao mesmo tempo, devem associar as chaves

Date: 3 de junho de 2001.

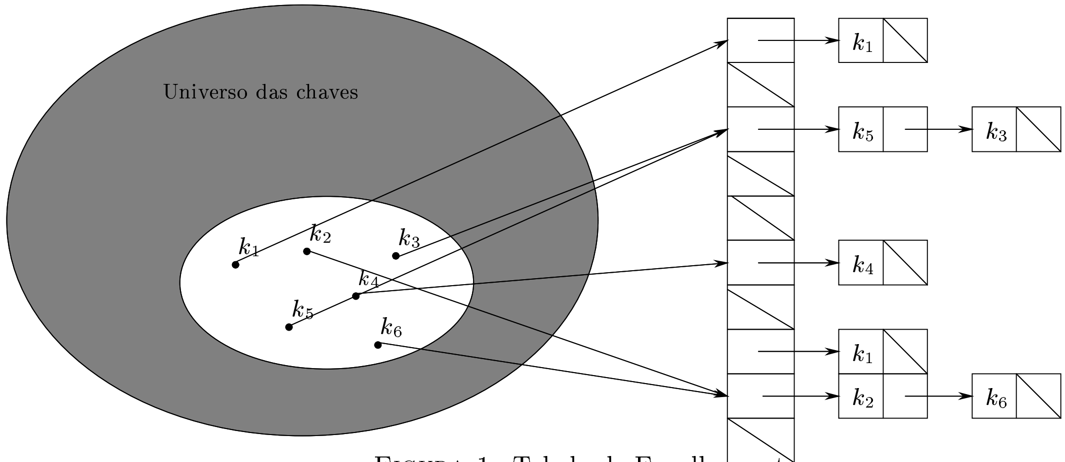

FIGURA 1. Tabela de Espalhamento

a valores “os mais diferentes” possíveis de modo a evitar colisões. Vamos ver agora dois tipos de funções de espalhamento.

3.1. **O método da divisão.** No método da divisão mapeamos uma chave k para um vetor de m posições tomando o resto da divisão de k por m.

$$h(k) = k \mod m.$$

Notem que esta função é muito simples e rápida de ser calculada, resta saber o quão bem ela “espalha”. Para garantir “bom espalhamento” precisamos tomar certos cuidados na escolha do m. Por exemplo, em geral não se deve usar uma potência de 2 como m. Se $m = 2^p$, o resto da divisão por m seria igual aos p bits menos significativos da chave e, ao menos que saibamos que estes bits estão bem “espalhados”, isso pode ser desastroso.

De uma maneira geral um bom m deve ser um número primo “longe” de uma potência de 2. Por exemplo imaginemos que queremos uma tabela de espalhamento para uns 2000 elementos e não nos importamos muito com a busca em listas de colisão curtas, digamos de 3 elementos. Neste caso, queremos uma tabela de aproximadamente 2000/3 elementos. Um primo próximo desse valor, mas “longe” de potências de 2, é 701. Deste modo, a função de hashing usada seria $h(k) = k \mod 701$.

3.2. **O método da multiplicação.** Neste caso, a função de espalhamento é calculada em duas etapas. Primeiro, multiplicamos a chave por uma constante $A \in (0, 1)$ e calculamos então a parte fracionária de kA. Em seguida, multiplicamos o resultado pelo tamanho da tabela m e o valor desejado é o chão deste resultado.[1] Usando “matematiquês”:

$$h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor .$$

A grande vantagem do método da multiplicação é que o tamanho da tabela, m, não é tão importante para garantir bom espalhamento. Inclusive, na página 229 do ótimo livro [1], mostra-se que a escolha de m como sendo uma potência de 2 pode ser aproveitada para faclitar a implementação do cálculo de h.

[1] Lembre-se, o chão de um número x, denotado por $\lfloor x \rfloor$, é o maior inteiro menor que o x.

Por fim, temos que ter cuidado na escolha da constante A. A escolha deste valor depende das propriedades das chaves que queremos espalhar. Para uma visão aprofundada, consulte [2]. Já os mais práticos podem tentar usar

$$A \approx (\sqrt{5}-1)/2 = 0,6180339887\ldots$$

que funciona razoavelmente bem na mior parte dos casos. Por exemplo, se $m = 10000$ e $k = 123456$ teremos:

$$\begin{aligned} h(k) &= \lfloor 10000(123456 \cdot 0618033\ldots - \lfloor 123456 \cdot 0618033\ldots \rfloor) \rfloor \\ &= \lfloor 10000 \cdot 0.0041151\ldots \rfloor \\ &= 41. \end{aligned}$$

4. Chaves como números naturais

Destacamos que toda a discussão anterior foi apresentada para o caso da chave ser um número natural. Se não for este o caso, ela deve ser convertida primeiro para um natural. Por exemplo, se a chave é uma cadeia de caracteres podemos interpretá-la como um natural em uma base conveniente (26, se levarmos em consideração apenas letras maiúsculas):

$$gato = 7 \cdot 26^3 + 1 \cdot 26^2 + 20 \cdot 26^1 + 15 = 124243.$$

Palavras mais longas representam um desafio maior, pois o valor calculado pode ser muito grande (e o cálculo pode levar muito tempo). Neste caso, podemos escolher algumas letras para representar a palavra, como a primeira, a do meio e a penúltima.

5. Exercícios

(1) Escreva um programa que, dado um tamanho desejado de tabela de espalhamento, calcula o primo "longe" de uma potência de dois mais próximo a esse tamanho. Explique o que fez e, em particular, como você definiu "longe".
(2) Implemente tabelas de espalhamento usando o método da divisão e o método da multiplicação para cadeias de caracteres de tamanho arbitrário. A tabela de espalhamento deve conter cerca de 700 posições. Compare os resultados usando o arquivo de dados fornecido. Quais critérios vocês escolheram para fazer a comparação? Defenda sua posição.

Este texto é baseado nas seções 12.1 a 12.3 do livro Introduction Algorithms [1].

Referências

[1] Cormen, T. H., Leiserson, C. E. e Rivest, R.L. *Introduction to Algorithms*, MIT Press, 1994.
[2] Knuth, D. *The Art of Computer Programming, volume 3: Sorting and Searching,* Addison Wesley, 1973.