AULA 23

TABELAS DE ESPALHAMENTO

Donald E. Knuth, em seu livro *The Art of Computer Programming*, observou que o cientista da computação Hans P. Luhn parece ter sido o primeiro a usar o conceito de espalhamento[1] em janeiro de 1953. Também observou que Robert H. Morris, outro cientista da computação, usou-o em um artigo da *Communications of the ACM* de 1968, tornando o termo espalhamento formal e conhecido na academia. O termo com o sentido que enxergamos na Computação vem da analogia com a mesma palavra da língua inglesa, um termo não-técnico que significa "corte e misture" ou "pique e misture".

Muitos problemas podem ser resolvidos através do uso das operações básicas de busca, inserção e remoção sobre as suas entradas. Por exemplo, um compilador para uma linguagem de programação mantém uma tabela de símbolos em memória onde as chaves são cadeias de caracteres que correspondem às palavras-chaves da linguagem. Outro exemplo menos abstrato é o de armazenamento das correspondências em papel dos funcionários de uma empresa. Se a empresa é grande, em geral não é possível manter um compartimento para cada funcionário(a), sendo mais razoável manter, por exemplo, 26 compartimentos rotulados com as letras do alfabeto. As correspondências de um dado funcionário(a), cujo nome inicia com uma dada letra, estão armazenadas no compartimento rotulado com aquela letra. A implementação de uma tabela de espalhamento para tratar das operações básicas de busca, inserção e remoção é uma maneira eficiente de solucionar problemas como esses. De fato, uma tabela de espalhamento é uma generalização da noção de vetor, como veremos.

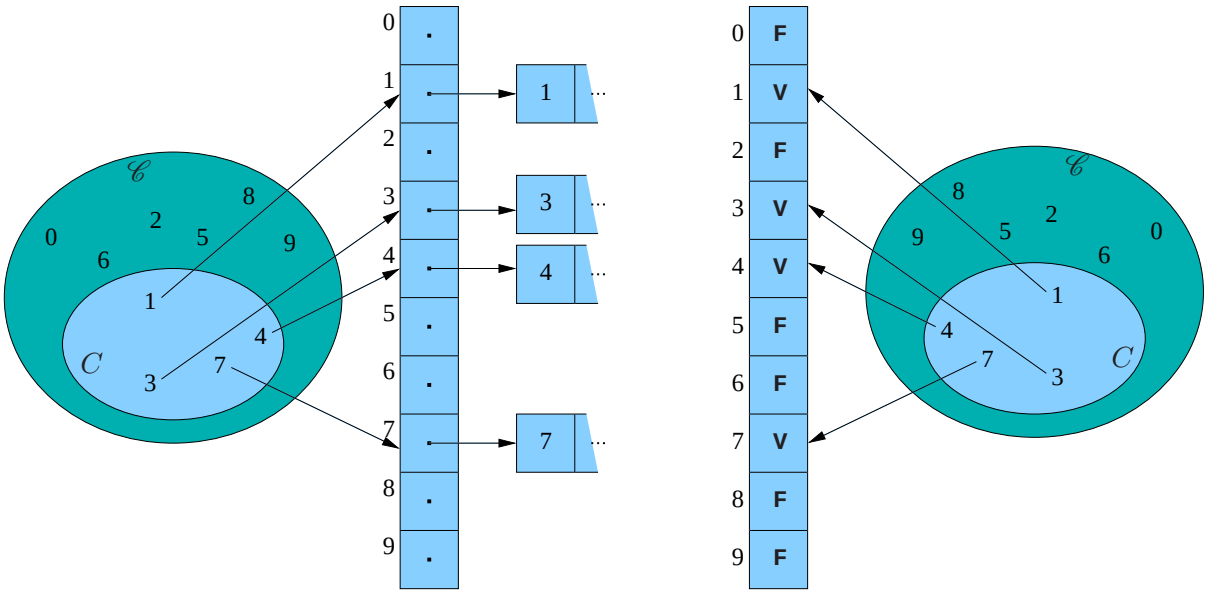
Nesta aula, baseada nas referências [1, 13], estudaremos essas estruturas de dados.

23.1 Tabelas de acesso direto

Suponha que temos uma aplicação em que cada informação seja identificada por uma chave e que as operações essenciais realizadas sobre essas informações sejam a busca, inserção e remoção. Podemos supor, sem perda de generalidade, que as chaves sejam apenas chaves numéricas, já que é sempre fácil associar um número inteiro a cada informação. Como as outras informações associadas às chaves são irrelevantes para as operações básicas necessárias à aplicação, podemos descartá-las e trabalhar apenas com as chaves. Suponha que todas as chaves possíveis na aplicação pertençam ao conjunto $\mathscr{C} = \{0, 1, \ldots, m-1\}$, onde m é um número inteiro relativamente pequeno.

[1] Do inglês *hash* ou *hashing*. Optamos pela tradução "espalhamento" e "tabela de espalhamento" usada pelo professor Tomasz Kowaltowsky, do IC-UNICAMP. O professor Jayme Luiz Szwarcfiter, da COPPE-UFRJ, usa os termos "dispersão" e "tabela de dispersão".

Então, usamos uma **tabela de acesso direto** para representar esse conjunto de informações, que nada mais é que um vetor $T[0..m-1]$ do tipo inteiro, onde o índice de cada compartimento de T corresponde a uma chave do conjunto $\mathscr{C}$. Em um dado momento durante a execução da aplicação, um determinado conjunto de chaves $C \subseteq \mathscr{C}$ está representado na tabela T. Veja a figura 23.1 para uma ilustração.



(a) (b)

Figura 23.1: Dois exemplos de tabela de acesso direto. (a) Cada compartimento da tabela armazena um ponteiro para um registro contendo a chave e outras informações relevantes. (b) Cada compartimento da tabela armazena um bit indicando se a chave representada no índice do vetor está presente.

É fácil ver que as operações básicas de busca, remoção e inserção são realizadas em tempo constante em uma tabela de acesso direto, considerando que as chaves são todas distintas. O problema com esta estratégia é a impossibilidade evidente de alocação de espaço na memória quando o conjunto de todas as possíveis chaves $\mathscr{C}$ contém alguma chave que é um número inteiro muito grande. Ademais, se há poucas chaves representadas no conjunto C em um dado instante da aplicação, a tabela de acesso direto terá, nesse instante, muito espaço alocado mas não usado.

23.2 Introdução às tabelas de espalhamento

Para tentar solucionar eficientemente o problema conseqüente do acesso direto, usamos uma tabela de espalhamento. Em uma tabela de acesso direto, a chave k é armazenada no compartimento de índice k. Em uma tabela de espalhamento, uma chave x é armazenada no compartimento $h(x)$, onde a função $h\colon \mathscr{C} \to \{0, 1, \ldots, m-1\}$ é chamada uma **função de espalhamento**. A função h mapeia as chaves de $\mathscr{C}$ nos compartimentos de uma **tabela de espalhamento** $T[0..m-1]$. O principal objetivo de uma função de espalhamento é reduzir o intervalo de índices da tabela de espalhamento.

A figura 23.2 mostra um exemplo de uma tabela de espalhamento associada a uma função de espalhamento.

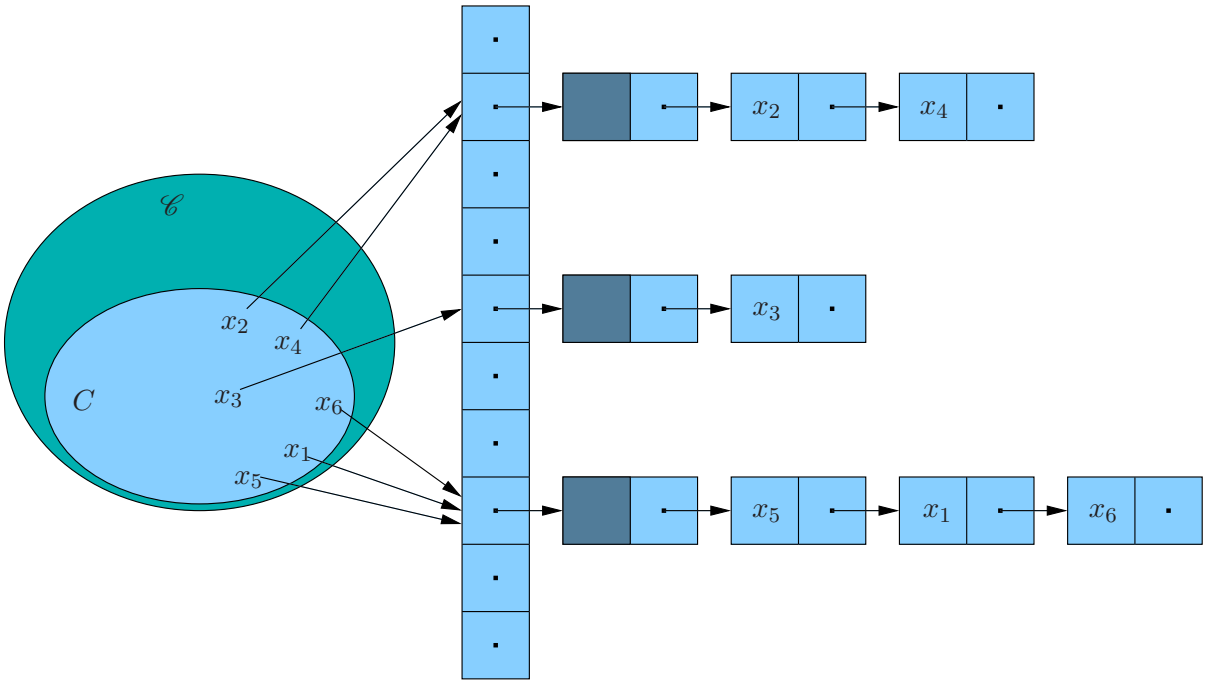


Figura 23.2: Exemplo de uma tabela de espalhamento T associada a uma função de espalhamento h. Observe que ocorrem as colisões $h(x_5) = h(x_1) = h(x_6)$ e $h(x_2) = h(x_4)$ e que tais colisões são resolvidas com listas lineares encadeadas com cabeça.

A principal estratégia de uma tabela de espalhamento é fazer uso de uma boa função de espalhamento que permita distribuir bem as chaves pela tabela. Em geral, como $|\mathscr{C}| > m$, isto é, como o número de elementos do conjunto de chaves possíveis é maior que o tamanho da tabela T, duas ou mais chaves certamente terão o mesmo índice. Ou seja, devem existir pelo menos duas chaves, digamos x_i e x_j, no conjunto das chaves possíveis $\mathscr{C}$ tais que $h(x_i) = h(x_j)$, como vimos na figura 23.2. Quando duas chaves x_i e x_j possuem essa propriedade, dizemos que há, ou pode haver, uma **colisão** entre x_i e x_j na tabela de espalhamento T. Uma maneira bastante evidente de solucionar o problema das colisões é manter uma lista linear encadeada com cabeça para cada subconjunto de colisões.

Observe que se a função de espalhamento é muito ruim, podemos ter um caso degenerado em que todas as chaves têm o mesmo índice, isto é, $h(x_1) = h(x_2) = \cdots = h(x_n)$ para toda chave $x_i \in C$, com $1 \leqslant i \leqslant n$ e $|C| = n$. Isso significa que temos, na verdade, uma tabela implementada como lista linear encadeada. Observe que esse caso é muito ruim especialmente porque todas as operações básicas, de busca, remoção e inserção, gastam tempo proporcional ao número de chaves contidas na lista, isto é, tempo proporcional a n. Veja a figura 23.3 para um exemplo desse caso.

A situação ideal quando projetamos uma tabela de espalhamento é usar uma função de espalhamento que distribua bem as chaves por todos os m compartimentos da tabela T. Dessa forma, uma tal função maximiza o número de compartimentos usados em T e minimiza os comprimentos das listas lineares encadeadas que armazenam as colisões. Um exemplo de uma situação como essa é ilustrado na figura 23.4.

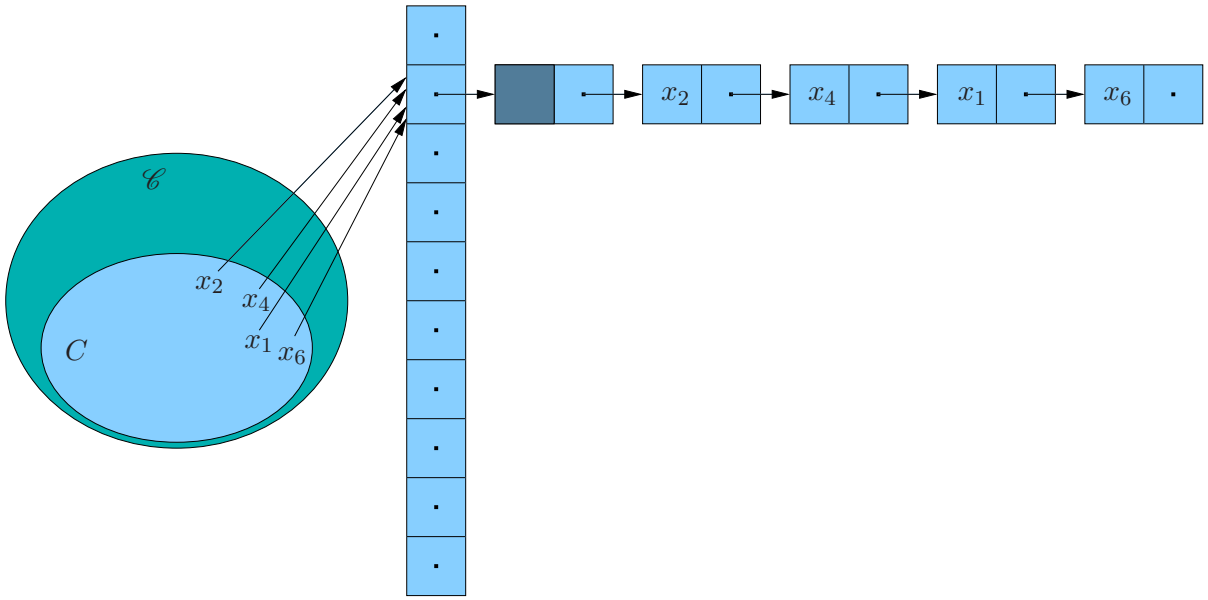


Figura 23.3: Exemplo de uma função de espalhamento "ruim", isto é, uma função que produz uma tabela de espalhamento degenerada.

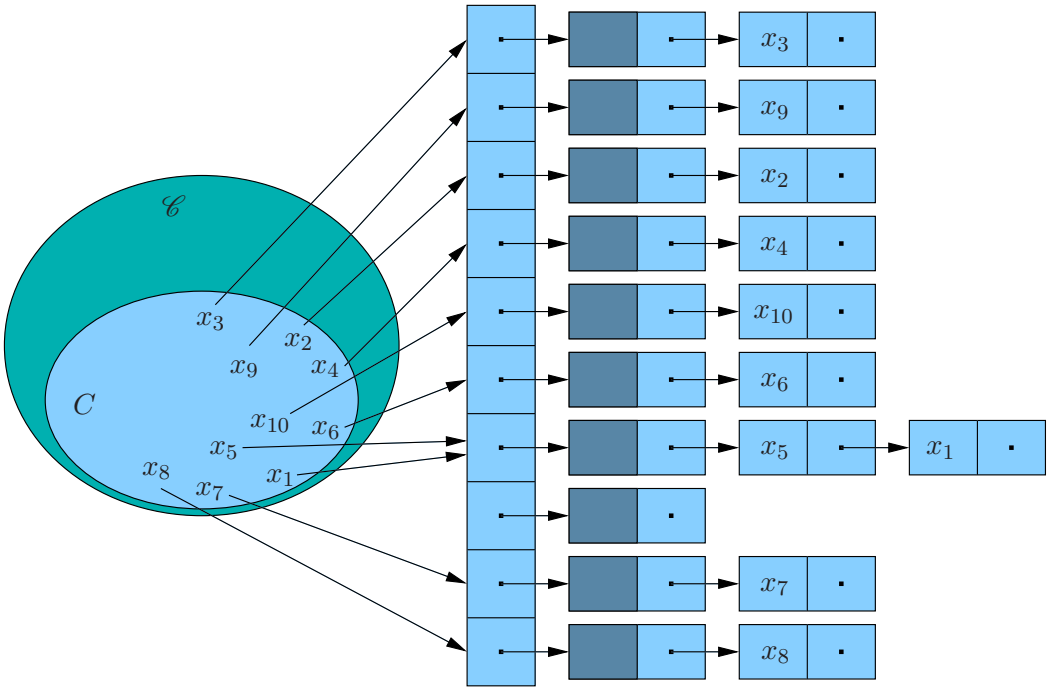


Figura 23.4: Exemplo de uma função de espalhamento próxima do ideal, que "espalha" bem as chaves pela tabela T, havendo poucas colisões e poucos compartimentos apontando para listas lineares vazias.

23.3 Tratamento de colisões com listas lineares encadeadas

Em uma tabela de espalhamento, o uso de listas lineares encadeadas é uma maneira intuitiva e eficiente de tratar colisões. Assim, as operações básicas de busca, remoção e inserção são operações de busca, remoção e inserção sobre listas lineares encadeadas, como vimos na aula 18. Apenas modificamos levemente a operação de inserção, que em uma tabela de espalhamento é sempre realizada no início da lista linear apropriada.

Se temos uma função de espalhamento, então a implementação de uma tabela de espalhamento com tratamento de colisões usando listas lineares encadeadas é um espelho da implementação de uma lista linear encadeada. Para analisar a eficiência dessa implementação, basta analisar o tempo de execução de cada uma das operações. Se considerarmos que todas elas necessitam de alguma forma da operação de busca, podemos fixar e analisar o tempo de execução apenas dessa operação.

O tempo de execução no pior caso da operação de busca é proporcional ao tamanho da lista linear encadeada associada. No pior caso, como já mencionamos, tal lista contém todas as n chaves do conjunto C e, assim, o tempo de execução de pior caso da operação de busca é proporcional a n. Dessa forma, à primeira vista parece que uma tabela de espalhamento é uma forma muito ruim de armazenar informações, já que o tempo de pior caso de cada uma das operações básicas é ruim.

No entanto, se estudamos o **tempo de execução no caso médio**, ou **tempo esperado de execução**, não o tempo de execução no pior caso, a situação se reverte em favor dessas estruturas. O tempo de execução no caso médio de uma busca em uma tabela de espalhamento é proporcional a $1 + \alpha$, onde α é chamado de **fator de carga** da tabela. Se a tabela de espalhamento T tem tamanho m e armazena n chaves, então α é definido como n/m. Dessa forma, se a quantidade de chaves n armazenada na tabela T é proporcional ao tamanho m de T, então as operações básicas são realizadas em tempo esperado constante.

23.3.1 Funções de espalhamento

Uma função de espalhamento deve, a princípio, satisfazer a suposição de que cada chave é igualmente provável de ser armazenada em qualquer um dos m compartimentos da tabela de espalhamento T, independentemente de onde qualquer outra chave foi armazenada antes. Uma função de espalhamento que tem essa propriedade é chamada de **função de espalhamento simples e uniforme**. Essa suposição implica na necessidade de conhecimento da distribuição de probabilidades da qual as chaves foram obtidas, o que em geral não sabemos previamente.

O uso de heurísticas para projetar funções de espalhamento é bastante freqüente. Isso significa que não há garantia alguma de que uma tal função seja simples e uniforme, mas muitas delas são usadas na prática e funcionam bem na maioria dos casos. O método da divisão e o método da multiplicação são heurísticas que fornecem funções de espalhamento com desempenho prático satisfatório. Algumas aplicações, no entanto, exigem que as funções de espalhamento tenham garantias de que são simples e uniformes. O método universal de espalhamento gera uma classe de funções de espalhamento que satisfazem essa propriedade.

É comum que uma função de espalhamento tenha como domínio o conjunto dos números naturais. Se as chaves de uma aplicação não são números naturais, podemos facilmente

transformá-las em números naturais. Por exemplo, se as chaves são cadeias de caracteres, podemos somar o valor de cada caractere que compõe a cadeia, obtendo assim um número natural que representa esta chave.

Nesta aula, estudamos as heurísticas mencionadas acima, deixando que os leitores interessados no método universal de espalhamento consultem as referências desta aula.

Método da divisão

O **método da divisão** projeta uma função de espalhamento mapeando o valor x da chave em um dos m compartimentos da tabela T. O método divide então x por m e toma o resto dessa divisão. Dessa forma, a função de espalhamento h é dada por:

$$h(x) = x \bmod m \, .$$

Para que a função obtida seja o mais efetiva possível, devemos evitar certos valores de m que podem tornar a função menos uniforme. Em geral, escolhemos um número primo grande não muito próximo a uma potência de 2.

Por exemplo, suponha que desejamos alocar uma tabela de espalhamento com colisões resolvidas por listas lineares encadeadas, que armazena aproximadamente $n = 1000$ cadeias de caracteres. Se não nos importa examinar em média 5 chaves em uma busca sem sucesso, então fazemos $m = 199$, já que 199 é um número primo próximo a $1000/5$ e distante de uma potência de 2.

Método da multiplicação

O **método da multiplicação** projeta funções de espalhamento da seguinte forma. Primeiro, multiplicamos a chave x por uma constante de ponto flutuante A, com $0 < A < 1$, e tomamos a parte fracionária de xA. Depois, multiplicamos esse valor por m e tomamos o piso do valor resultante. Ou seja, uma função de espalhamento h é dada por:

$$h(x) = \lfloor m \left(xA - \lfloor xA \rfloor \right) \rfloor \, .$$

Diferentemente do método da divisão, o valor de m no método da multiplicação não é o mais importante. Em geral, escolhemos $m = 2^p$ para algum número inteiro p, já que dessa forma podemos usar operações sobre bits para obter $h(x)$. No entanto, o método da multiplicação funciona melhor com alguns valores de A do que com outros. Donald E. Knuth sugere que $A \approx (\sqrt{5} - 1)/2 = 0{,}6180339887\ldots$ seja um valor que forneça boas funções de espalhamento pelo método da multiplicação.

Outros métodos

Algumas outras heurísticas para geração de funções de espalhamento podem ser citadas, como por exemplo, o método da dobra e o método da análise dos dígitos. Interessados devem procurar as referências desta aula. Além disso, o método universal escolhe ou gera funções aleatoriamente, de forma independente das chaves a serem armazenadas na tabela, garantindo assim um bom desempenho médio.

23.4 Endereçamento aberto

Em uma tabela de espalhamento com endereçamento aberto, as chaves são todas armazenadas na própria tabela. Por isso, muitas vezes uma tabela de espalhamento com colisões solucionadas por listas lineares encadeadas é chamada de tabela de espalhamento com endereçamento exterior.

Cada compartimento de uma tabela de espalhamento com endereçamento aberto ou contém uma chave ou um valor que indica que o compartimento está vazio. Na busca por uma chave, examinamos sistematicamente os compartimentos da tabela até que a chave desejada seja encontrada ou até que fique claro que a chave não consta na tabela. Ao invés de seguir ponteiros, como em uma tabela de espalhamento com endereçamento exterior, devemos computar a seqüência de compartimentos a serem examinados. Para realizar uma inserção em uma tabela de espalhamento com endereçamento aberto, examinamos sucessivamente a tabela até que um compartimento vazio seja encontrado. A seqüência de compartimentos examinada depende da chave a ser inserida e não da ordem dos índices $0, 1, \ldots, m-1$.

Em uma tabela com endereçamento aberto, a função de espalhamento tem domínio e contra-domínio definidos como:

$$h\colon \mathscr{C} \times \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}\ .$$

Além disso, é necessário que para toda chave x, a **seqüência de tentativas** ou **seqüência de exames** dada por $\langle h(x,0), h(x,1), \ldots, h(x,m-1)\rangle$ seja uma permutação de $\langle 0, 1, \ldots, m-1\rangle$.

Inicialmente, a tabela de espalhamento com endereçamento aberto T é inicializada com -1 em cada um de seus compartimentos, indicando que todos estão vazios. Veja a figura 23.5.

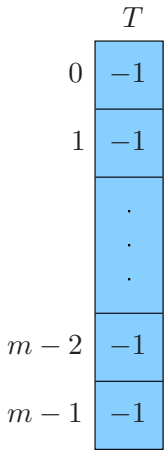


Figura 23.5: Uma tabela de espalhamento com endereçamento aberto vazia.

Na função `insere_aberto` a seguir, uma tabela de espalhamento T com endereçamento aberto, contendo m compartimentos, e uma chave x são fornecidas como entrada. Consideramos que as chaves estão armazenadas diretamente na tabela de espalhamento T e que um compartimento vazio de T contém o valor -1. Após o processamento, a função devolve um número inteiro entre 0 e $m-1$ indicando que a inserção obteve sucesso ou o valor m indicando o contrário.

```
/* Recebe uma tabela de espalhamento T de tamanho m  e uma chave x e
   insere a chave x na tabela T, devolvendo o índice da tabela onde a
   chave foi inserida, em caso de sucesso,  ou m em caso contrário */
int insere_aberto(int m, int T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j] == -1) {
         T[j] = x;
         return j;
      }
      else
         i++;
   } while (i != m);

   return m;
}
```

A busca em uma tabela com endereçamento aberto é muito semelhante à inserção que vimos acima. A busca também termina se a chave foi encontrada na tabela ou quando encontra um compartimento vazio.

```
/* Recebe uma tabela de espalhamento T de tamanho m e uma chave
   x e busca a chave x na tabela T, devolvendo o índice da tabe-
   la onde a chave foi encontrada ou o valor m em caso contrário */
int busca_aberto(int m, int T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j] == x)
         return j;
      else
         i++;
   } while (T[j] != -1 && i != m);

   return m;
}
```

A remoção de uma chave em um compartimento i da tabela de espalhamento com endereçamento aberto T não permite que o compartimento i seja marcado com -1, já que isso tem implicação direta em seqüências de tentativas posteriores, afetando negativamente operações básicas subseqüentes, especialmente a inserção. Uma solução possível é fazer com que cada compartimento da tabela seja uma células com dois campos: uma chave e um estado. O estado de uma célula pode ser um dos três: **VAZIA**, **OCUPADA** ou **REMOVIDA**. Dessa forma, uma tabela de espalhamento com endereçamento aberto T é um vetor do tipo **celula**, definido abaixo:

```
struct cel {
   int chave;
   int estado;
};

typedef struct cel celula;
```

Inicialmente, a tabela de espalhamento T é inicializada com seus estados todos preenchidos com o **VAZIA**, como mostra a figura 23.6.

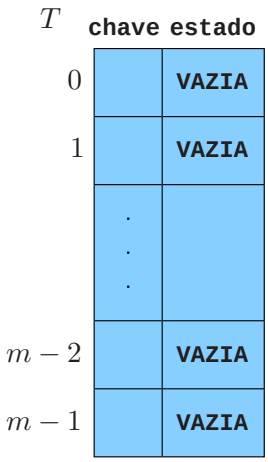


Figura 23.6: Uma tabela de espalhamento com endereçamento aberto vazia, onde cada compartimento contém uma chave e seu estado.

As três operações básicas sobre essa nova tabela de espalhamento com endereçamento aberto são implementadas nas funções a seguir.

```
/* Recebe uma tabela de espalhamento T de tamanho m e uma chave
   x e busca a chave x na tabela T, devolvendo o índice da tabe-
   la onde a chave foi encontrada ou o valor m em caso contrário */
int busca_aberto(int m, int T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j].chave == x && T[j].estado == OCUPADA)
         return j;
      else
         i++;
   } while (T[j].estado != VAZIA && i != m);

   return m;
}
```

```
/* Recebe uma tabela de espalhamento T de tamanho m  e uma chave x e
   insere a chave x na tabela T, devolvendo o índice da tabela onde a
   chave foi inserida, em caso de sucesso,  ou m em caso contrário */
int insere_aberto(int m, int T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j].estado != OCUPADA) {
         T[j].chave = x;
         T[j].estado = OCUPADA;
         return j;
      }
      else
         i++;
   } while (i != m);
   return m;
}
```

```
/* Recebe uma tabela de espalhamento T de tamanho m  e uma chave x e
   remove a chave x na tabela T, devolvendo o índice da tabela onde a
   chave foi inserida, em caso de sucesso,  ou m em caso contrário */
int remove_aberto(int m, int T[MAX], int x)
{
   int i, j;

   i = 0;
   do {
      j = h(x, i);
      if (T[j].estado == OCUPADA) {
         if (T[j].chave == x) {
            T[j].estado = REMOVIDA;
            return j;
         }
         else
            i++;
      }
      else
         i = m;
   } while (i != m);
   return m;
}
```

A figura 23.7 mostra um exemplo de uma tabela de espalhamento T com endereçamento aberto, alocada com 10 compartimentos e contendo 6 chaves: $41, 22, 104, 16, 37, 16$. A função de espalhamento usada h é dada pelo método da tentativa linear, que veremos na seção 23.4.1 a seguir, e é definida da seguinte forma:

$$h(x, i) = \big(h'(x) + i\big) \bmod 10\,,$$

onde h' é uma função de espalhamento ordinária dada pelo método da divisão apresentado na seção 23.3.1 e definida como:

$$h'(x) = x \bmod 10\,.$$

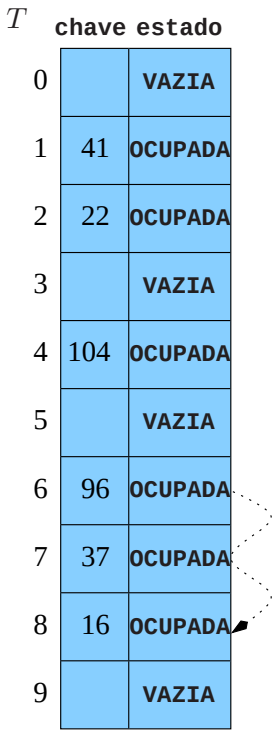


Figura 23.7: Um exemplo de uma tabela de espalhamento com endereçamento aberto. A busca da chave 16 usando a função de espalhamento h segue a seqüência de tentativas ilustrada pela linha pontilhada.

O papel da função de espalhamento h é gerar uma seqüência de compartimentos onde uma chave de interesse pode ocorrer. Observe que, para a função h particular que vimos acima, se a remoção da chave 37 é realizada na tabela T, a busca subseqüente pela chave 16 ocorre corretamente.

Em uma tabela de espalhamento com endereçamento aberto temos sempre no máximo uma chave por compartimento. Isso significa que $n \leqslant m$, onde n é o número de chaves armazenadas e m o total de compartimentos da tabela. Portanto, o fator de carga α da tabela sempre satisfaz $\alpha \leqslant 1$. Se consideramos a hipótese que a tabela de espalhamento é uniforme, isto é, que a seqüência de tentativas $\langle h(x,0), h(x,1), \ldots, h(x,m-1)\rangle$ usada em uma operação básica é igualmente provável a qualquer permutação de $\langle 0, 1, \ldots, m-1\rangle$, então temos duas conseqüências importantes:

- o número esperado de tentativas em uma busca sem sucesso é no máximo

$$\frac{1}{1-\alpha};$$

  isso significa, por exemplo, que se a tabela está preenchida pela metade, então o número médio de tentativas em uma busca sem sucesso é no máximo $1/(1-0.5) = 2$; se a tabela está 90% cheia, então o número médio de tentativas é no máximo $1/(1-0.9) = 10$;

- o número esperado de tentativas em uma busca com sucesso é no máximo

$$\left(\frac{1}{\alpha}\right) \ln\left(\frac{1}{1-\alpha}\right);$$

isso significa, por exemplo, que se a tabela está preenchida pela metade, então o número médio de tentativas em uma busca sem sucesso é menor que 1.387; se a tabela está 90% cheia, então o número médio de tentativas é menor que 2.559.

23.4.1 Funções de espalhamento

Idealmente, supomos que cada chave em uma tabela de espalhamento com endereçamento aberto tenha qualquer uma das $m!$ permutações igualmente prováveis de $\langle 0, 1, \ldots, m-1\rangle$ como sua seqüência de tentativas. Essa suposição sobre a função de espalhamento é chamada de **espalhamento uniforme** e generaliza a suposição de espalhamento simples e uniforme que vimos acima. Funções de espalhamento uniforme são difíceis de implementar e na prática são usadas algumas boas aproximações.

As técnicas mais comuns para computar seqüências de tentativas em tabelas de espalhamento com endereçamento aberto são as seguintes: tentativa linear, tentativa quadrática e espalhamento duplo. Todas essas técnicas garantem que a seqüência de tentativas produzida $\langle h(x,0), h(x,1), \ldots, h(x,m-1)\rangle$ é uma permutação da seqüência $\langle 0, 1, \ldots, m-1\rangle$ para cada chave x. Nenhuma delas satisfaz as suposições de espalhamento uniforme, sendo que a técnica do espalhamento duplo é a que apresenta melhores resultados.

Tentativa linear

Dada uma função de espalhamento auxiliar $h' : \mathscr{C} \to \{0, 1, \ldots, m-1\}$, o **método da tentativa linear** é dado pela função

$$h(x,i) = \big(h'(x) + i\big) \bmod m\,,$$

para $i = 0, 1, \ldots, m-1$. Para uma chave x, o primeiro compartimento examinado é $T[h'(x)]$, que é o compartimento obtido pela função de espalhamento auxiliar. Em seguida, o compartimento $T[h'(x)+1]$ é examinado e assim por diante, até o compartimento $T[m-1]$. Depois disso, a seqüência de tentativas continua a partir de $T[0]$, seguindo para $T[1]$ e assim por diante, até $T[h'(x)-1]$.

É fácil de implementar funções de espalhamento usando o método da tentativa linear, mas tais funções sofrem de um problema conhecido como **agrupamento primário**, que provoca o aumento do tempo médio das operações básicas.

Tentativa quadrática

Uma função de espalhamento projetada pelo **método da tentativa quadrática** é ligeiramente diferente de uma função dada pelo método da busca linear, já que usa uma função da forma:

$$h(x,i) = \big(h'(x) + c_1 i + c_2 i^2\big) \bmod m\,,$$

onde h' é uma função de espalhamento auxiliar, c_1 e c_2 são constantes auxiliares e $i = 0, 1, \ldots, m-1$. Para uma chave x, o primeiro compartimento examinado é $T[h'(x)]$. Em seguida, os compartimentos examinados são obtidos pelo deslocamento quadrático de valores que dependem de i. Este método é melhor que o anterior, mas a escolha dos valores c_1, c_2 e

m é restrita. Além disso, o método sofre do problema de **agrupamento secundário**, já que se $h(x_1, 0) = h(x_2, 0)$ então $h(x_1, i) = h(x_2, i)$ para todo i.

Espalhamento duplo

O **método do espalhamento duplo** usa uma função de espalhamento da seguinte forma:

$$h(x, i) = \big(h_1(x) + ih_2(x)\big) \bmod m\,,$$

onde h_1 e h_2 são funções de espalhamento auxiliares. O compartimento inicialmente examinado é $T[h_1(x)]$. Os compartimentos examinados em seguida são obtidos do deslocamento dos compartimentos anteriores da quantidade de $h_2(x)$ módulo m.

A recomendação é que o valor $h_2(x)$ seja um primo relativo ao tamanho da tabela m. Podemos assegurar essa condição fazendo m uma potência de 2 e projetando h_2 de tal forma que sempre produza um número ímpar. Outra forma é fazer m primo e projetar h_2 de tal forma que sempre devolva um número inteiro positivo menor que m.

O método do espalhamento duplo é um dos melhores métodos disponíveis para o endereçamento aberto já que as permutações produzidas têm muitas das características de permutações aleatórias. O desempenho do método do espalhamento duplo é, assim, próximo ao desempenho do esquema ideal de espalhamento universal.

Exercícios

23.1 Suponha que temos um conjunto de chaves C armazenado em uma tabela de acesso direto T de tamanho m. Escreva uma função que encontra a maior chave de C. Qual o tempo de execução de pior caso da sua função?

23.2 Suponha um conjunto de n chaves formado pelos n primeiros múltiplos de 7. Quantas colisões ocorrem mediante a aplicação das funções de espalhamento abaixo?

(a) $x \bmod 7$

(b) $x \bmod 14$

(c) $x \bmod 5$

23.3 Ilustre a inserção das chaves $5, 28, 19, 15, 20, 33, 12, 17, 10$ em uma tabela de espalhamento com colisões resolvidas por listas lineares encadeadas. Suponha que a tabela tenha 9 compartimentos e que a função de espalhamento seja $h(x) = x \bmod 9$.

23.4 Considere uma tabela de espalhamento de tamanho $m = 1000$ e uma função de espalhamento $h(x) = \lfloor m\,(Ax - \lfloor Ax \rfloor)\rfloor$ para $A = (\sqrt{5} - 1)/2$. Compute as posições para as quais as chaves $61, 62, 63, 64$ e 64 são mapeadas.

23.5 Considere a inserção das chaves $10, 22, 31, 4, 15, 28, 17, 88, 59$ em uma tabela de espalhamento com endereçamento aberto de tamanho $m = 11$ com função de espalhamento auxiliar $h'(x) = x \bmod m$. Ilustre o resultado da inserção dessas chaves usando tentativa linear, tentativa quadrática com $c_1 = 1$ e $c_2 = 3$ e espalhamento duplo com $h_2(x) = 1 + (x \bmod (m - 1))$.

23.6 A tabela abaixo é composta das palavras-chaves da linguagem C padrão.

`auto`	`double`	`int`	`long`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

Escreva um programa que leia um arquivo contendo um programa na linguagem C e identifique suas palavras-chaves.

23.7 Veja animações do funcionamento de tabelas de espalhamento nas páginas a seguir:

- Hashing Animation Tool, Catalyst Software, 2000
- Hash Table Animation, Woi Ang
- Hashing, Hang Thi Anh Pham, 2001