Hashing
Endereçamento Aberto

Professora:

Fátima L. S. Nunes

Introdução

- Aula passada:
  - Hashing
  - Endereçamento Direto
  - Tabelas Hash

Endereçamento aberto

- Todos os elementos estão armazenados na própria tabela *hash:*
  - cada entrada da tabela contém um elemento do conjunto dinâmico ou NULO;
  - não existe nenhuma lista e nenhum elemento armazenado fora da tabela;
  - a tabela pode ficar cheia – impossibilitando inserções adicionais;
  - fator de carga α (número médio de elementos armazenados) não pode exceder 1.
- Vantagem?

Endereçamento aberto

- Todos os elementos estão armazenados na própria tabela *hash.*
  - cada entrada da tabela contém um elemento do conjunto dinâmico ou NULO;
  - não existe nenhuma lista e nenhum elemento armazenado fora da tabela;
  - a tabela pode ficar cheia – impossibilitando inserções adicionais;
  - fator de carga α (número médio de elementos armazenados) não pode exceder 1.
- Vantagem: evita o uso de ponteiros das listas encadeadas.

Endereçamento aberto

- Vantagens:
  - evita o uso de ponteiros das listas encadeadas;
  - em vez de seguir ponteiros, calcula-se a sequência de posições a serem examinadas;
  - espaço da tabela *hash* é melhor aproveitado: espaço não alocado para as listas é usado para aumentar tamanho da tabela ⇒ número menor de colisões.

Endereçamento aberto

- Inserção:
  - examinar sucessivamente a tabela *hash* até encontrar posição vazia para inserir chave: processo chamado de **sondagem.**
  - Como examinar sucessivamente? Começa na posição 0 e vai até *m-1*?

Endereçamento aberto

- Inserção:
  - examinar sucessivamente a tabela *hash* até encontrar posição vazia para inserir chave: processo chamado de **sondagem.**
  - Como examinar sucessivamente? Começa na posição 0 e vai até *m-1*?
  - Isso daria um tempo de pesquisa O(n).
  - Então: sequência de posições verificadas depende *da chave que está sendo inserida*.

Endereçamento aberto

- Inserção:
  - para determinar as posições a serem sondadas, estende-se a função *hash,* incluindo-se o número de sondagens a partir de 0 como uma segunda entrada.

$$h: U \times \{0, 1, m-1\} \rightarrow \{0, 1, m-1\}$$

  - Para toda chave *k,* a sequência de sondagem <*h(k,0), h(k,1), ..., h(k,m-1)*> é uma permutação de <0, 1, ..., m-1>.
    - isto significa que toda posição da tabela *hash* será considerada como posição possível para alocar uma nova chave.

Endereçamento aberto

- Algoritmo para Inserção:

```
HashInsertion(T,k)
i ← 0
repita
   j ← h(k,i)
   se T[j]= NULO
      T[j] ← k
      return j
   else
      i ← i + 1
   fim se
até i = m
erro “estourou a tabela hash”
```

Algoritmo considera:

- elementos da tabela *hash* são chaves sem dados adicionais;
- chave *k* é idêntica ao elemento que contém a chave k;
- cada posição contém uma chave ou NULO se estiver vazia.

Endereçamento aberto

- Busca:
  - faz a sondagem da mesma sequência de posições examinadas pelo algoritmo de inserção;
  - pesquisa termina sem sucesso ao encontrar uma posição vazia.

Endereçamento aberto

- Algoritmo para Busca:

```
HashSearch(T,k)
i ← 0
repita
    j ← h(k,i)
    se T[j]= k
        return j
    fim se
    i ← i + 1
até T[j] = NULO ou i = m
return NULO
```

Algoritmo considera:

- elementos da tabela *hash* são chaves sem dados adicionais;
- chave *k* é idêntica ao elemento que contém a chave k;
- cada posição contém uma chave ou NULO se estiver vazia.

Endereçamento aberto

- Remoção:
  - é mais complexa;
  - não se pode simplesmente assinalar a posição eliminada como NULO. Por quê?

Endereçamento aberto

- Remoção:
  - é mais complexa;
  - não se pode simplesmente assinalar a posição eliminada como NULO. Por quê?
    - problemas na busca – algoritmo não encontraria chaves incluídas depois da chave eliminada.
  - Solução???

Endereçamento aberto

- Remoção:
  - é mais complexa;
  - não se pode simplesmente assinalar a posição eliminada como NULO. Por quê?
    - problemas na busca – algoritmo não encontraria chaves incluídas depois da chave eliminada.
  - Solução???
    - marcar a posição como ELIMINADA – assim o algoritmo de busca saberá que há outras chaves depois dela.

Endereçamento aberto

- Algoritmo de Remoção:
  - marcando eliminado na posição...

```
HashDelete(T,k)
i ← 0
j ← h(k,i)
enquanto (i!=m) e (T[j]!=NULO)repita
   se T[j]= k
      T[j]=ELIMINADO
      return j
   fim se
   i ← i + 1
   j ← h(k,i)
fim enquanto
return NULO
```

Endereçamento aberto

- Que mudança é necessária na inserção para contemplar as alterações na tabela feitas pela remoção?

```
HashInsertion(T,k)
i ← 0
repita
   j ← h(k,i)
   se T[j]= NULO
      T[j] ← k
      return j
   else
      i ← i + 1
   fim se
até i = m
erro “estourou a tabela hash”
```

Endereçamento aberto

- Que mudança é necessária na inserção para contemplar as alterações na tabela feitas pela remoção?

```
HashInsertion(T,k)
i ← 0
repita
   j ← h(k,i)
   se T[j]= NULO ou T[j]=ELIMINADO
      T[j] ← k
      return j
   else
      i ← i + 1
   fim se
até i = m
erro “estourou a tabela hash”
```

Endereçamento aberto

- Que mudança é necessária na busca?

```
HashSearch(T,k)
i ← 0
repita
    j ← h(k,i)
    se T[j]= k
        return j
    fim se
    i ← i + 1
até T[j] = NULO ou i = m
return NULO
```

Endereçamento aberto

- Que mudança é necessária na busca?

```
HashSearch(T,k)
i ← 0
repita
    j ← h(k,i)
    se T[j]= k
        return j
    fim se
    i ← i + 1
até T[j] = NULO ou i = m
return NULO
```

NENHUMA!!!

Mas teremos problemas em relação ao tempo de busca. O tempo de pesquisa não dependerá mais somente do fator de carga α (n/m). Isto significa que não dependerá mais somente do número de elementos presentes na tabela, mas também do número de elementos eliminados.

Endereçamento aberto

- Até agora consideramos *hash uniforme*. O que é?

Endereçamento aberto

- Até agora consideramos *hash uniforme*. O que é?
  - cada chave tem probabilidade igual de atingir qualquer uma da m! permutações <0 ,1, ..., m> como sua sequência de sondagem.
  - não é trivial de obter e implementar: na prática, são usadas aproximações.
- Três técnicas são usadas comumente para calcular as sequências de sondagem exigidas para o endereçamento aberto:
  - sondagem linear
  - sondagem quadrática
  - hash duplo

Sondagem linear

- Dada uma função *hash* comum $h': U \rightarrow \{0,1,\ldots, m-1\}$, chamada de **função *hash* auxiliar**, define-se a função *hash*:

$$h(k,i) = (h'(k) + i) \bmod m \qquad \text{para } i=0,1,\ldots,m-1$$

  - *mod* é o operador de resto

Qual o comportamento dessa sondagem???

Sondagem linear

- Dada uma função *hash* comum *h'*: $U \rightarrow \{0,1,\ldots, m-1\}$, chamada de **função *hash* auxiliar**, define-se a função *hash*:

$$h(k,i) = (h'(k) + i) \bmod m \qquad \text{para } i=0,1,\ldots,m-1$$

  - *mod* é o operador de resto

i	h(k,i)
0	T[h'(k)]
1	T[h'(k) + 1]
...	...
...	T[m-1]
...	T[0]
...	T[1]
....	...
m-1	T[h'(k)-1]

Sondagem linear

- Fácil de implementar.
- Posição *h'(k)* determina a sequência posterior
  - assim, existem somente *m* sequências de sondagem distintas;
- Problema: **agrupamento primário**

Sondagem linear

- Agrupamento primário:
  - agrupamentos surgem porque uma posição vazia precedida por *i* posições completas é preenchida em seguida com probabilidade (i+1)/m.
  - sequências de posições ocupadas ficam mais longas → aumento do tempo médio de pesquisa.
  - Gera no máximo *m* sequências distintas, ou seja, número possível de sequências é Θ(m).

Sondagem quadrática

- Usa função *hash* na forma:

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

onde:

  - $h'(k)$ *é uma* função *hash* auxiliar
  - c_1 *e* $c_2 \neq 0$ são constantes auxiliares
  - $i = 0, 1, \ldots, m-1$

- Posição inicial sondada: T[h'(k)]

- Posições posteriores são deslocadas por quantidade que dependem de forma quadrática do número da sondagem *i*.

- Exemplo: $h(k, i) = (h'(k) + i + 3i^2) \bmod 11$

h'(k)=k mod 11
c1=1
c2=3
m=11

Sondagem quadrática

- Método funciona melhor que sondagem linear, mas para usar tabela *hash* plenamente, valores de c_1, c_2 e m devem ser limitados.
- Problema desta abordagem:
  - se duas chaves têm a mesma posição de sondagem inicial, suas sequências de sondagem serão iguais, pois $h(k_1,0) = h(k_2,0)$ implica $h(k_1,i) = h(k_2,i)$.
  - essa propriedade conduz a uma forma mais interessante de agrupamento, chamada **agrupamento secundário**.
  - Da mesma forma que sondagem linear, gera no máximo m sequências distintas, ou seja, número possível de sequências é $\Theta(m)$.

Hash duplo

- Um dos melhores métodos para endereçamento aberto → permutações produzidas têm muitas características de permutações escolhidas aleatoriamente.
- Usa função *hash* na forma:

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

onde:

- h_1 *e* h_2 *são* funções *hash* auxiliares

- Posição inicial sondada:$T[h_1(k)]$
- Posições de sondagem sucessivas são deslocadas a partir de posições anteriores pela quantidade $h_2(k)$, módulo m.

Hash duplo

- $h(k,i) = (h_1(k) + ih_2(k)) \bmod m$

onde:

- h_1 e h_2 são funções *hash* auxiliares

i	h(k,i)
0	$T[h_1(k)]$
1	$T[h_1(k)+h_2(k)] \bmod m$
2	$T[h_1(k)+2h_2(k)] \bmod m$
....	...
m-1	$T[h_1(k)+(m-1)h_2(k)] \bmod m$

Hash duplo

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

- Diferentemente das sondagens linear e quadrática, no *hash duplo* a sequência de sondagem depende da chave k de duas maneiras:
  - posição de sondagem inicial, o deslocamento, ou ambos, podem variar.

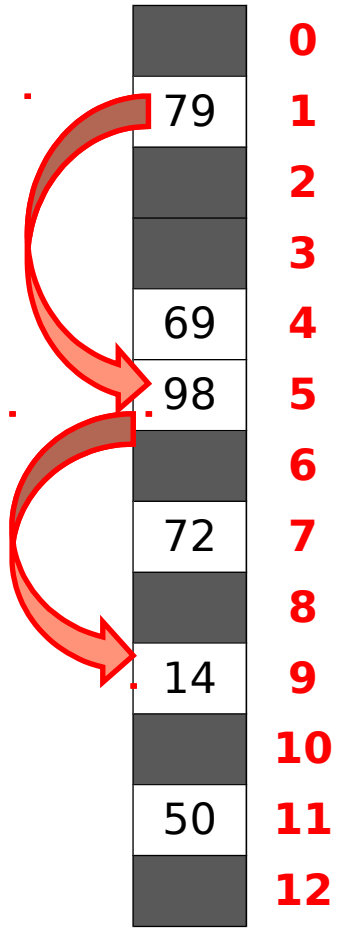

Tabela *hash* com 13 posições:

$h_1(k) = k \bmod 13$

$h_2(k) = 1 + (k \bmod 11)$

Como $14 \equiv 1 \bmod 13$ e $14 \equiv 3 \bmod 11$, a chave 14 será inserida na posição vazia 9, depois que as posições 1 e 5 tiverem sido examinadas e se descobrir que já estão ocupadas.

Hash duplo

- Como escolher h_1 e h_2?
  - O valor $h_2(k)$ e o tamanho m da tabela *hash* devem ser primos entre si para que a tabela *hash* inteira possa ser pesquisada.
  - Formas de garantir esta condição:
    1. fazer m uma potência de 2 e projetar h_2 para retornar sempre um número ímpar
    2. fazer m um número primo e projetar h_2 para retornar sempre um inteiro positivo menor que m.

Exemplo:

- escolher m primo e fazer:

$$h_1(k) = k \bmod m,$$

$$h_2(k) = 1 + (k \bmod m')$$

m' é escolhido como valor ligeiramente menor que *m* *(Exemplo:m-1)*

Hash duplo

Exemplo:

- escolher *m* primo e fazer:

  $h_1(k) = k \bmod m,$

  $h_2(k) = 1 + (k \bmod m')$

m' é escolhido como valor ligeiramente menor que *m* *(Exemplo:m-1)*

- Usando o exemplo:

  k = 123456

  m = 701

  m'= 700

  $h_1(k)$=80

  $h_2(k)$=257

  - primeira sondagem: posição 80
  - depois cada 257-ésima posição (módulo m) é examinada até a chave ser encontrada ou todas as posições serem examinadas.

Hash duplo

- *Hash* duplo é um aperfeiçoamento em relação à sondagem linear ou quadrática:
  - são usadas $\Theta(m^2)$ sequências de sondagem, em lugar de $\Theta(m)$, visto que cada par ($h_1(k)$ e $h_2(k)$) gera uma sequência de sondagem distinta.
- Resultado: desempenho do *hash* duplo é muito próximo do desempenho do esquema ideal de *hash uniforme*.

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002.
- Nota de aulas do professor Delano Beder (EACH-USP).

Hashing
Endereçamento Aberto

Professora:

Fátima L. S. Nunes