

Aula 26 – Hashing: Endereçamento Aberto

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2024

Tabelas de Hash

Endereçamento Aberto

- Vimos que a tabela de hash é uma estrutura em que inclusões, buscas e exclusões podem vir a ser feitas, em média, em $O(1)$

Tabelas de Hash

Endereçamento Aberto

- Vimos que a tabela de hash é uma estrutura em que inclusões, buscas e exclusões podem vir a ser feitas, em média, em $O(1)$
- Mais do que isso, é uma estrutura útil quando não sabemos o número de registros de antemão, e ainda assim queremos usar das vantagens dos arranjos em termos de velocidade

Tabelas de Hash

Endereçamento Aberto

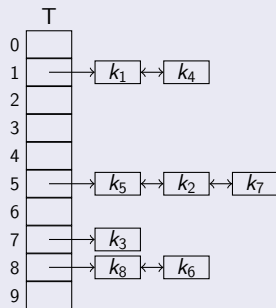
- Vimos que a tabela de hash é uma estrutura em que inclusões, buscas e exclusões podem vir a ser feitas, em média, em $O(1)$
- Mais do que isso, é uma estrutura útil quando não sabemos o número de registros de antemão, e ainda assim queremos usar das vantagens dos arranjos em termos de velocidade
 - Basicamente, usamos um arranjo

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

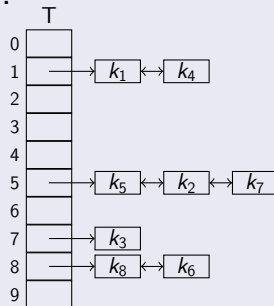
- Vimos que a tabela de hash é uma estrutura em que inclusões, buscas e exclusões podem vir a ser feitas, em média, em $O(1)$
- Mais do que isso, é uma estrutura útil quando não sabemos o número de registros de antemão, e ainda assim queremos usar das vantagens dos arranjos em termos de velocidade
 - Basicamente, usamos um arranjo encadeando colisões



Tabelas de Hash

Endereçamento Aberto

- Mas e quando temos como saber (ou estimar) o número de registros n previamente?



Tabelas de Hash

Endereçamento Aberto

- Mas e quando temos como saber (ou estimar) o número de registros n previamente?
- Nesse caso, não há necessidade de usarmos listas ligadas para tratar das colisões

	T
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Mas e quando temos como saber (ou estimar) o número de registros n previamente?
- Nesse caso, não há necessidade de usarmos listas ligadas para tratar das colisões
- Como isso?

	T
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Mas e quando temos como saber (ou estimar) o número de registros n previamente?
- Nesse caso, não há necessidade de usarmos listas ligadas para tratar das colisões
- Como isso?
 - Fazendo com que os elementos ocupem a própria tabela de hash

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Mas e quando temos como saber (ou estimar) o número de registros n previamente?
- Nesse caso, não há necessidade de usarmos listas ligadas para tratar das colisões
- Como isso?
 - Fazendo com que os elementos ocupem a própria tabela de hash
- Naturalmente, isso exige que $|T| = m \geq n$

T

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Métodos que usam lugares vazios na tabela para resolver colisões são chamados de **endereçoamento aberto**

	T
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Métodos que usam lugares vazios na tabela para resolver colisões são chamados de **endereçoamento aberto**
- Quando buscamos por um elemento, examinamos sistematicamente a tabela até encontrá-lo

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Métodos que usam lugares vazios na tabela para resolver colisões são chamados de **endereçoamento aberto**
- Quando buscamos por um elemento, examinamos sistematicamente a tabela até encontrá-lo
 - Ou até termos certeza de que não está lá

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Métodos que usam lugares vazios na tabela para resolver colisões são chamados de **endereçamento aberto**
- Quando buscamos por um elemento, examinamos sistematicamente a tabela até encontrá-lo
 - Ou até termos certeza de que não está lá
- Nenhuma lista é armazenada fora da tabela

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Isso, no entanto, pode fazer com que a tabela se encha e não possamos mais incluir nada

	T
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Isso, no entanto, pode fazer com que a tabela se encha e não possamos mais incluir nada
- E como agimos em caso de colisão?

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Isso, no entanto, pode fazer com que a tabela se encha e não possamos mais incluir nada
- E como agimos em caso de colisão?
 - **Calculamos** uma localização alternativa

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Isso, no entanto, pode fazer com que a tabela se encha e não possamos mais incluir nada
- E como agimos em caso de colisão?
 - **Calculamos** uma localização alternativa
 - Se tal cálculo não for possível, a tabela está cheia

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Isso, no entanto, pode fazer com que a tabela se encha e não possamos mais incluir nada
- E como agimos em caso de colisão?
 - **Calculamos** uma localização alternativa
 - Se tal cálculo não for possível, a tabela está cheia
- Poupa então o espaço usado pelos ponteiros da lista

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Tabelas de Hash

Endereçamento Aberto

- Isso, no entanto, pode fazer com que a tabela se encha e não possamos mais incluir nada
- E como agimos em caso de colisão?
 - **Calculamos** uma localização alternativa
 - Se tal cálculo não for possível, a tabela está cheia
- Poupa então o espaço usado pelos ponteiros da lista
 - Espaço este que pode ser aproveitado para aumentar a tabela, levando a menos colisões

	T
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Endereçamento Aberto

Inclusão de um Elemento

- Para incluir um elemento, devemos sondar a tabela até encontrarmos um espaço vazio onde armazenar esse dado

Endereçamento Aberto

Inclusão de um Elemento

- Para incluir um elemento, devemos sondar a tabela até encontrarmos um espaço vazio onde armazenar esse dado
- Mas isso não é $\Theta(m)$, onde m é o tamanho da tabela?

Endereçamento Aberto

Inclusão de um Elemento

- Para incluir um elemento, devemos sondar a tabela até encontrarmos um espaço vazio onde armazenar esse dado
- Mas isso não é $\Theta(m)$, onde m é o tamanho da tabela?
 - Se seguirmos uma ordem fixa $\langle 0, 1, \dots, m - 1 \rangle$, sim.

Endereçamento Aberto

Inclusão de um Elemento

- Para incluir um elemento, devemos sondar a tabela até encontrarmos um espaço vazio onde armazenar esse dado
- Mas isso não é $\Theta(m)$, onde m é o tamanho da tabela?
 - Se seguirmos uma ordem fixa $\langle 0, 1, \dots, m-1 \rangle$, sim.
- O truque é fazer com que as posições sondadas dependam do elemento (chave) a ser inserido

Endereçamento Aberto

Inclusão de um Elemento

- E como fazer isso?

Endereçamento Aberto

Inclusão de um Elemento

- E como fazer isso?
 - Fazendo com que a função de hash leve em conta o número da sondagem, quando gera uma posição na tabela

Endereçamento Aberto

Inclusão de um Elemento

- E como fazer isso?
 - Fazendo com que a função de hash leve em conta o número da sondagem, quando gera uma posição na tabela
- Geramos então uma sequência de sondagens $\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$

Endereçamento Aberto

Inclusão de um Elemento

- E como fazer isso?
 - Fazendo com que a função de hash leve em conta o número da sondagem, quando gera uma posição na tabela
- Geramos então uma sequência de sondagens $\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$
 - Onde $h(k, i)$ dá a posição no hash para a chave k em sua $(i + 1)$ -ésima sondagem

Endereçamento Aberto

Inclusão de um Elemento

- E como fazer isso?
 - Fazendo com que a função de hash leve em conta o número da sondagem, quando gera uma posição na tabela
- Geramos então uma sequência de sondagens $\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$
 - Onde $h(k, i)$ dá a posição no hash para a chave k em sua $(i + 1)$ -ésima sondagem
 - Ou seja, após i colisões

Endereçamento Aberto

Inclusão de um Elemento

- Note que $\langle h(k, 0), h(k, 1), \dots, h(k, m-1) \rangle$ é uma permutação das posições $\langle 0, 1, \dots, m-1 \rangle$ da tabela

Endereçamento Aberto

Inclusão de um Elemento

- Note que $\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$ é uma permutação das posições $\langle 0, 1, \dots, m - 1 \rangle$ da tabela
- Ou seja, no pior caso cada posição da tabela é verificada, caso a tabela esteja cheia

Endereçamento Aberto

Inclusão de um Elemento

- Note que $\langle h(k, 0), h(k, 1), \dots, h(k, m - 1) \rangle$ é uma permutação das posições $\langle 0, 1, \dots, m - 1 \rangle$ da tabela
- Ou seja, no pior caso cada posição da tabela é verificada, caso a tabela esteja cheia
- E como implementamos isso?

Endereçamento Aberto

Inclusão de um Elemento

- Note que $\langle h(k, 0), h(k, 1), \dots, h(k, m-1) \rangle$ é uma permutação das posições $\langle 0, 1, \dots, m-1 \rangle$ da tabela
- Ou seja, no pior caso cada posição da tabela é verificada, caso a tabela esteja cheia
- E como implementamos isso?
 - Vamos supor que as chaves são inteiros ≥ 0 , correspondendo aos CPFs das Pessoas a serem incluídas

Endereçamento Aberto

Inclusão de um Elemento

- Note que $\langle h(k, 0), h(k, 1), \dots, h(k, m-1) \rangle$ é uma permutação das posições $\langle 0, 1, \dots, m-1 \rangle$ da tabela
- Ou seja, no pior caso cada posição da tabela é verificada, caso a tabela esteja cheia
- E como implementamos isso?
 - Vamos supor que as chaves são inteiros ≥ 0 , correspondendo aos CPFs das Pessoas a serem incluídas
 - Vamos usar NULL para indicar que a posição na tabela está livre

Endereçamento Aberto

Inclusão de um Elemento

- Então...

Endereçamento Aberto

Inclusão de um Elemento

- Então...

```
typedef struct {
    Pessoa** tabela;
    int n;
    Pessoa* REMOVIDA;
} Dicionario;

int hash(int cpf, int i, int n) { ... }
int inserir(Dicionario d, Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```

Endereçamento Aberto

Inclusão de um Elemento

- Então...

A tabela agora guardará o endereço do próprio elemento, em vez de uma lista ligada

```
typedef struct {
    Pessoa** tabela;
    int n;
    Pessoa* REMOVIDA;
} Dicionario;

int hash(int cpf, int i, int n) { ... }
int inserir(Dicionario d, Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```

Endereçamento Aberto

Inclusão de um Elemento

- Então...

Por enquanto, deixemos
em aberto como funciona
o hash

```
typedef struct {
    Pessoa** tabela;
    int n;
    Pessoa* REMOVIDA;
} Dicionario;

int hash(int cpf, int i, int n) { ... }
int inserir(Dicionario d, Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```

Endereçamento Aberto

Inclusão de um Elemento

- Então...

Para inserir uma pessoa, começamos a sondagem

```
typedef struct {
    Pessoa** tabela;
    int n;
    Pessoa* REMOVIDA;
} Dicionario;

int hash(int cpf, int i, int n) { ... }
int inserir(Dicionario d, Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```

Endereçamento Aberto

Inclusão de um Elemento

- Então...

E a colocamos na primeira
posição vazia retornada
pela sondagem

```
typedef struct {
    Pessoa** tabela;
    int n;
    Pessoa* REMOVIDA;
} Dicionario;

int hash(int cpf, int i, int n) { ... }
int inserir(Dicionario d, Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```


Endereçamento Aberto

Inclusão de um Elemento

- Então...

Se já olhamos todos os espaços na tabela, ela está cheia (lembre que a sondagem gera uma permutação dos índices da tabela)

```
typedef struct {
    Pessoa** tabela;
    int n;
    Pessoa* REMOVIDA;
} Dicionario;

int hash(int cpf, int i, int n) { ... }
int inserir(Dicionario d, Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```

Endereçamento Aberto

Busca de um Elemento

- A busca deve sondar a mesma sequência de posições que a inserção

Endereçamento Aberto

Busca de um Elemento

- A busca deve sondar a mesma sequência de posições que a inserção

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] &&
           i < d.n) {
        if (d.tabela[pos] -> cpf == cpf)
            return d.tabela[pos];
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```

Endereçamento Aberto

Busca de um Elemento

- A busca deve sondar a mesma sequência de posições que a inserção
- Assim, ela pode terminar:

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] &&
           i < d.n) {
        if (d.tabela[pos]->cpf == cpf)
            return d.tabela[pos];
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```

Endereçamento Aberto

Busca de um Elemento

- A busca deve sondar a mesma sequência de posições que a inserção
- Assim, ela pode terminar:
 - Quando encontra o elemento

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] &&
           i < d.n) {
        if (d.tabela[pos]->cpf == cpf)
            return d.tabela[pos];
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```

Endereçamento Aberto

Busca de um Elemento

- A busca deve sondar a mesma sequência de posições que a inserção
- Assim, ela pode terminar:
 - Quando encontra o elemento
 - Quando encontra um NULL, indicando que não há mais elemento na sequência, ou a tabela termina

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] &&
           i < d.n) {
        if (d.tabela[pos]->cpf == cpf)
            return d.tabela[pos];
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?
 - Fácil, basta colocar um NULL em sua posição

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?
 - Fácil, basta colocar um NULL em sua posição
- Mas então o que acontecerá com a busca ?

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?
 - Fácil, basta colocar um NULL em sua posição
- Mas então o que acontecerá com a busca ?
 - Irá parar nesse NULL, ignorando os elementos posteriores

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?
 - Fácil, basta colocar um NULL em sua posição
- Mas então o que acontecerá com a busca ?
 - Irá parar nesse NULL, ignorando os elementos posteriores
 - Não é uma boa solução

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?
 - Fácil, basta colocar um NULL em sua posição
- Mas então o que acontecerá com a busca ?
 - Irá parar nesse NULL, ignorando os elementos posteriores
 - Não é uma boa solução
- A solução é marcar a posição como “apagada”, colocando um -1 no cpf, por exemplo

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?
 - Fácil, basta colocar um NULL em sua posição
- Mas então o que acontecerá com a busca ?
 - Irá parar nesse NULL, ignorando os elementos posteriores
 - Não é uma boa solução
- A solução é marcar a posição como “apagada”, colocando um -1 no cpf, por exemplo
 - Ou seja, não apagamos realmente o registro

Endereçamento Aberto

Remoção de um Elemento

- E como podemos fazer para remover um elemento?
 - Fácil, basta colocar um NULL em sua posição
- Mas então o que acontecerá com a busca ?
 - Irá parar nesse NULL, ignorando os elementos posteriores
 - Não é uma boa solução
- A solução é marcar a posição como “apagada”, colocando um -1 no cpf, por exemplo
 - Ou seja, não apagamos realmente o registro
 - Solução adotada pelos SOs, quando apagamos arquivos

Endereçamento Aberto

Remoção de um Elemento

- Então...

```
bool remover(Dicionario d, int cpf){  
    int i = 0;  
    int pos = hash(cpf, i, d.n);  
    while (d.tabela[pos] && i < d.n) {  
        if (d.tabela[pos]->cpf == cpf) {  
            d.tabela[pos] = d.REMOVIDA;  
            return true;  
        }  
        i++;  
        pos = hash(cpf, i, d.n);  
    }  
    return false;  
}
```

Endereçamento Aberto

Remoção de um Elemento

- Então...
- Usamos o código da busca

```
bool remover(Dicionario d, int cpf){  
    int i = 0;  
    int pos = hash(cpf, i, d.n);  
    while (d.tabela[pos] && i<d.n) {  
        if (d.tabela[pos]->cpf == cpf) {  
            d.tabela[pos] = d.REMOVIDA;  
            return true;  
        }  
        i++;  
        pos = hash(cpf, i, d.n);  
    }  
    return false;  
}
```


Endereçamento Aberto

Remoção de um Elemento

- Então...
- Usamos o código da busca
- Marcando o elemento quando o encontramos

```
bool remover(Dicionario d, int cpf){  
    int i = 0;  
    int pos = hash(cpf, i, d.n);  
    while (d.tabela[pos] && i<d.n) {  
        if (d.tabela[pos]->cpf == cpf) {  
            d.tabela[pos] = d.REMOVIDA;  
            return true;  
        }  
        i++;  
        pos = hash(cpf, i, d.n);  
    }  
    return false;  
}
```

Endereçamento Aberto

Remoção de um Elemento

- Mas isso não é suficiente

Endereçamento Aberto

Remoção de um Elemento

- Mas isso não é suficiente
- Temos que modificar inserir para que trate esses elementos como posições vazias

```
int inserir(Dicionario d,
            Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL ||
            d.tabela[j]->cpf == -1) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```

Endereçamento Aberto

Remoção de um Elemento

- Mas isso não é suficiente
- Temos que modificar inserir para que trate esses elementos como posições vazias
- Inserindo novos elementos lá, se necessário

```
int inserir(Dicionario d,
            Pessoa* pes){
    int j, i=0;
    while (i<d.n) {
        j = hash(pes->cpf,i, d.n);
        if (d.tabela[j] == NULL ||
            d.tabela[j]->cpf == -1) {
            d.tabela[j] = pes;
            return j;
        }
        i++;
    }
    return -1;
}
```

Endereçamento Aberto

Remoção de um Elemento

- Uma consequência de permitirmos remoção se dá na busca por elementos

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] && i < d.n) {
        if (d.tabela[pos]->cpf == cpf)
            return d.tabela[pos];
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```

Endereçamento Aberto

Remoção de um Elemento

- Uma consequência de permitirmos remoção se dá na busca por elementos
- Antes, ela dependia somente dos elementos presentes na tabela

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] && i < d.n) {
        if (d.tabela[pos]->cpf == cpf)
            return d.tabela[pos];
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```

Endereçamento Aberto

Remoção de um Elemento

- Uma consequência de permitirmos remoção se dá na busca por elementos
- Antes, ela dependia somente dos elementos presentes na tabela
- Agora, depende tanto dos presentes, quanto dos removidos

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] && i < d.n) {
        if (d.tabela[pos]->cpf == cpf)
            return d.tabela[pos];
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```

Endereçamento Aberto

Remoção de um Elemento

- Uma consequência de permitirmos remoção se dá na busca por elementos
- Antes, ela dependia somente dos elementos presentes na tabela
- Agora, depende tanto dos presentes, quanto dos removidos
- Pois estes serão analisados como se estivessem presentes

```
Pessoa* buscar(Dicionario d,
               int cpf){
    int i = 0;
    int pos = hash(cpf, i, d.n);
    while (d.tabela[pos] && i < d.n) {
        if (d.tabela[pos]->cpf == cpf) {
            return d.tabela[pos];
        }
        i++;
        pos = hash(cpf, i, d.n);
    }
    return NULL;
}
```


Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash
- Na medida em que elementos são armazenados, ele vai se preenchendo

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash
- Na medida em que elementos são armazenados, ele vai se preenchendo
 - $h(21, 0) = 1$

T	
0	
1	21
2	
3	
4	
5	
6	
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash
- Na medida em que elementos são armazenados, ele vai se preenchendo
 - $h(21, 0) = 1$
 - $h(34, 0) = 4$

T	
0	
1	21
2	
3	
4	34
5	
6	
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash
- Na medida em que elementos são armazenados, ele vai se preenchendo
 - $h(21, 0) = 1$
 - $h(34, 0) = 4$
 - $h(5, 0) = 5$

T	
0	
1	21
2	
3	
4	34
5	5
6	
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash
- Na medida em que elementos são armazenados, ele vai se preenchendo
 - $h(21, 0) = 1$
 - $h(34, 0) = 4$
 - $h(5, 0) = 5$
- E colisões podem ocorrer

T	
0	
1	21
2	
3	
4	34
5	5
6	
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash
- Na medida em que elementos são armazenados, ele vai se preenchendo
 - $h(21, 0) = 1$
 - $h(34, 0) = 4$
 - $h(5, 0) = 5$
- E colisões podem ocorrer
 - $h(35, 0) = 5$

T	
0	
1	21
2	
3	
4	34
5	5
6	
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Considere o seguinte hash
- Na medida em que elementos são armazenados, ele vai se preenchendo
 - $h(21, 0) = 1$
 - $h(34, 0) = 4$
 - $h(5, 0) = 5$
- E colisões podem ocorrer
 - $h(35, 0) = 5$
 - E uma localização alternativa é calculada ($h(35, 1) = 6$)

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Essa localização alternativa, por sua vez, pode gerar novos conflitos

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Essa localização alternativa, por sua vez, pode gerar novos conflitos
 - $h(26, 0) = 6$

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Essa localização alternativa, por sua vez, pode gerar novos conflitos
 - $h(26, 0) = 6$
 - Houve uma espécie de “cruzamento” de listas, em que elementos mapeados a uma posição colidem com elementos de outra lista de colisões que foram postos lá

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Essa localização alternativa, por sua vez, pode gerar novos conflitos
 - $h(26, 0) = 6$
 - Houve uma espécie de “cruzamento” de listas, em que elementos mapeados a uma posição colidem com elementos de outra lista de colisões que foram postos lá
- Calcula-se uma alternativa a esse então

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Essa localização alternativa, por sua vez, pode gerar novos conflitos
 - $h(26, 0) = 6$
 - Houve uma espécie de “cruzamento” de listas, em que elementos mapeados a uma posição colidem com elementos de outra lista de colisões que foram postos lá
- Calcula-se uma alternativa a esse então
 - $h(26, 1) = 7$

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Imagine agora que removemos o elemento 5

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Imagine agora que removemos o elemento 5
 - Ou seja, o marcamos como tal

T	
0	
1	21
2	
3	
4	34
5	-1
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Imagine agora que removemos o elemento 5
 - Ou seja, o marcamos como tal
- Se formos buscar o elemento 35, seguiremos a ordem de inserção

T	
0	
1	21
2	
3	
4	34
5	-1
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Imagine agora que removemos o elemento 5
 - Ou seja, o marcamos como tal
- Se formos buscar o elemento 35,
 - $h(35, 0) = 5$

T	
0	
1	21
2	
3	
4	34
5	-1
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Imagine agora que removemos o elemento 5
 - Ou seja, o marcamos como tal
- Se formos buscar o elemento 35, seguiremos a ordem de inserção
 - $h(35, 0) = 5$
 - $h(35, 1) = 6$

T	
0	
1	21
2	
3	
4	34
5	-1
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Imagine agora que removemos o elemento 5
 - Ou seja, o marcamos como tal
- Se formos buscar o elemento 35, seguiremos a ordem de inserção
 - $h(35, 0) = 5$
 - $h(35, 1) = 6$
- Ou seja, é como se o elemento nunca tivesse sido removido

T	
0	
1	21
2	
3	
4	34
5	-1
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Por isso endereçamento aberto é raramente utilizado quando precisamos permitir a remoção de elementos

T	
0	
1	21
2	
3	
4	34
5	-1
6	35
7	26
8	
9	

Endereçamento Aberto

Remoção de um Elemento – Exemplo

- Por isso endereçamento aberto é raramente utilizado quando precisamos permitir a remoção de elementos
- Nesse caso, encadeamento (com lista ligada, por exemplo) é mais adequado

T	
0	
1	21
2	
3	
4	34
5	-1
6	35
7	26
8	
9	

Endereçamento Aberto

Hashing Uniforme

- Vimos que, para garantir uma complexidade média de $\Theta(1)$:

Endereçamento Aberto

Hashing Uniforme

- Vimos que, para garantir uma complexidade média de $\Theta(1)$:
 - A função de hash precisaria fazer com que cada um dos n elementos tivesse a mesma chance de ser mapeado a qualquer uma das m posições da tabela

Endereçamento Aberto

Hashing Uniforme

- Vimos que, para garantir uma complexidade média de $\Theta(1)$:
 - A função de hash precisaria fazer com que cada um dos n elementos tivesse a mesma chance de ser mapeado a qualquer uma das m posições da tabela
 - O chamado **hashing uniforme simples**

Endereçamento Aberto

Hashing Uniforme

- Vimos que, para garantir uma complexidade média de $\Theta(1)$:
 - A função de hash precisaria fazer com que cada um dos n elementos tivesse a mesma chance de ser mapeado a qualquer uma das m posições da tabela
 - O chamado **hashing uniforme simples**
- Para o endereçamento aberto, essa suposição deve ser estendida para a sequência de sondagens

Endereçamento Aberto

Hashing Uniforme

- Vimos que, para garantir uma complexidade média de $\Theta(1)$:
 - A função de hash precisaria fazer com que cada um dos n elementos tivesse a mesma chance de ser mapeado a qualquer uma das m posições da tabela
 - O chamado **hashing uniforme simples**
- Para o endereçamento aberto, essa suposição deve ser estendida para a sequência de sondagens
 - Ou seja, a sequência de sondagens de cada chave deve ter a mesma probabilidade de qualquer outra das $m!$ permutações de $\langle 0, 1, \dots, m - 1 \rangle$

Endereçamento Aberto

Hashing Uniforme

- Essa condição é conhecida como **hashing uniforme**

Endereçamento Aberto

Hashing Uniforme

- Essa condição é conhecida como **hashing uniforme**
 - Trata-se, então de uma generalização do hashing uniforme simples para sequências de sondagens

Hashing Uniforme

- Essa condição é conhecida como **hashing uniforme**
 - Trata-se, então de uma generalização do hashing uniforme simples para sequências de sondagens
- Hashing uniforme, contudo, é difícil de implementar

Endereçamento Aberto

Hashing Uniforme

- Essa condição é conhecida como **hashing uniforme**
 - Trata-se, então de uma generalização do hashing uniforme simples para sequências de sondagens
- Hashing uniforme, contudo, é difícil de implementar
- Usamos então uma aproximação para a função de hash:

Hashing Uniforme

- Essa condição é conhecida como **hashing uniforme**
 - Trata-se, então de uma generalização do hashing uniforme simples para sequências de sondagens
- Hashing uniforme, contudo, é difícil de implementar
- Usamos então uma aproximação para a função de hash:
 - Sondagem linear

Endereçamento Aberto

Hashing Uniforme

- Essa condição é conhecida como **hashing uniforme**
 - Trata-se, então de uma generalização do hashing uniforme simples para sequências de sondagens
- Hashing uniforme, contudo, é difícil de implementar
- Usamos então uma aproximação para a função de hash:
 - Sondagem linear
 - Sondagem quadrática

Endereçamento Aberto

Hashing Uniforme

- Essa condição é conhecida como **hashing uniforme**
 - Trata-se, então de uma generalização do hashing uniforme simples para sequências de sondagens
- Hashing uniforme, contudo, é difícil de implementar
- Usamos então uma aproximação para a função de hash:
 - Sondagem linear
 - Sondagem quadrática
 - Hashing duplo

Endereçamento Aberto

Sondagem Linear

- Dada uma **função de hash auxiliar**
 $h' : U \rightarrow 0, 1, \dots, m - 1$, a
sondagem linear usa a função
de hash
 $h(k, i) = (h'(k) + i) \bmod m$,
com $i = 0, 1, \dots, m - 1$

Endereçamento Aberto

Sondagem Linear

- Dada uma **função de hash auxiliar**
 $h' : U \rightarrow 0, 1, \dots, m - 1$, a sondagem linear usa a função de hash
 $h(k, i) = (h'(k) + i) \bmod m$,
com $i = 0, 1, \dots, m - 1$
- Então, dada uma chave k , a sequência de sondagens será

Endereçamento Aberto

Sondagem Linear

- Dada uma **função de hash auxiliar**
 $h' : U \rightarrow 0, 1, \dots, m - 1$, a sondagem linear usa a função de hash
 $h(k, i) = (h'(k) + i) \bmod m$,
com $i = 0, 1, \dots, m - 1$
- Então, dada uma chave k , a sequência de sondagens será

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

Endereçamento Aberto

Sondagem Linear

- Uma vez que a sondagem inicial $h'(k)$ determina a sequência toda, há apenas m sequências de sondagem distintas

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear

- Uma vez que a sondagem inicial $h'(k)$ determina a sequência toda, há apenas m sequências de sondagem distintas
- Uma para cada posição inicial em T

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear

- Uma vez que a sondagem inicial $h'(k)$ determina a sequência toda, há apenas m sequências de sondagem distintas
- Uma para cada posição inicial em T
- Note que, pela condição do hashing uniforme, deveria haver $m!$

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear

- Uma vez que a sondagem inicial $h'(k)$ determina a sequência toda, há apenas m sequências de sondagem distintas
- Uma para cada posição inicial em T
- Note que, pela condição do hashing uniforme, deveria haver $m!$
- Método fácil de implementar

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear

- Uma vez que a sondagem inicial $h'(k)$ determina a sequência toda, há apenas m sequências de sondagem distintas
 - Uma para cada posição inicial em T
 - Note que, pela condição do hashing uniforme, deveria haver $m!$
- Método fácil de implementar
 - Porém sofre do problema do **agrupamento primário** (*primary clustering*)

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso significa que, se houver um agrupamento e a posição inicial de um novo elemento cair em qualquer ponto deste, o agrupamento aumentará

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso significa que, se houver um agrupamento e a posição inicial de um novo elemento cair em qualquer ponto deste, o agrupamento aumentará
- Ex: suponha que vamos inserir 16

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso significa que, se houver um agrupamento e a posição inicial de um novo elemento cair em qualquer ponto deste, o agrupamento aumentará
- Ex: suponha que vamos inserir 16
 - $h(16, 0) = 6$

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso significa que, se houver um agrupamento e a posição inicial de um novo elemento cair em qualquer ponto deste, o agrupamento aumentará
- Ex: suponha que vamos inserir 16
 - $h(16, 0) = 6$
 - Haverá uma colisão e a sondagem linear irá à posição seguinte ($h(16, 1) = 7$)

	T
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso significa que, se houver um agrupamento e a posição inicial de um novo elemento cair em qualquer ponto deste, o agrupamento aumentará
- Ex: suponha que vamos inserir 16
 - $h(16, 0) = 6$
 - Haverá uma colisão e a sondagem linear irá à posição seguinte ($h(16, 1) = 7$)
 - Lembre que $h(k, i) = (h'(k) + i) \bmod m$

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- ...
 - Haverá nova colisão e a sondagem linear irá à posição seguinte ($h(16, 2) = 8$)

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- ...
 - Haverá nova colisão e a sondagem linear irá à posição seguinte ($h(16, 2) = 8$)
 - Finalmente o elemento será incluído nessa posição

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	16
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- ...
 - Haverá nova colisão e a sondagem linear irá à posição seguinte ($h(16, 2) = 8$)
 - Finalmente o elemento será incluído nessa posição
 - Aumentando o agrupamento existente

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	16
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- ...
 - Haverá nova colisão e a sondagem linear irá à posição seguinte ($h(16, 2) = 8$)
 - Finalmente o elemento será incluído nessa posição
 - Aumentando o agrupamento existente
- Longas sequências de posições ocupadas são então construídas, aumentando o tempo médio de busca

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	16
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso ocorre porque uma posição vazia precedida de i cheias será ocupada na sequência com probabilidade $(i + 1)/m$

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	16
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso ocorre porque uma posição vazia precedida de i cheias será ocupada na sequência com probabilidade $(i + 1)/m$
- Probabilidade de colidir com cada uma das i anteriores, além de cair nela mesma

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	16
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso ocorre porque uma posição vazia precedida de i cheias será ocupada na sequência com probabilidade $(i + 1)/m$
- Probabilidade de colidir com cada uma das i anteriores, além de cair nela mesma
- Note que a probabilidade aumenta na medida em que aumenta i

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	16
9	

Endereçamento Aberto

Sondagem Linear – Agrupamento Primário

- Isso ocorre porque uma posição vazia precedida de i cheias será ocupada na sequência com probabilidade $(i + 1)/m$
- Probabilidade de colidir com cada uma das i anteriores, além de cair nela mesma
- Note que a probabilidade aumenta na medida em que aumenta i
 - Sequências longas tendem a ficar mais longas

T	
0	
1	21
2	
3	
4	34
5	5
6	35
7	26
8	16
9	

Endereçamento Aberto

Sondagem Quadrática

- Usa uma função de hash da forma
$$h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$$

Endereçamento Aberto

Sondagem Quadrática

- Usa uma função de hash da forma
$$h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$$
- onde h' é a função de hash auxiliar, $c_1, c_2 > 0$ são constantes auxiliares, e $i = 0, 1, \dots, m - 1$

Endereçamento Aberto

Sondagem Quadrática

- Usa uma função de hash da forma
$$h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$$
- onde h' é a função de hash auxiliar, $c_1, c_2 > 0$ são constantes auxiliares, e $i = 0, 1, \dots, m - 1$
- A primeira posição sondada é $T[h'(k)]$

Endereçamento Aberto

Sondagem Quadrática

- Usa uma função de hash da forma
$$h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$$
- onde h' é a função de hash auxiliar, $c_1, c_2 > 0$ são constantes auxiliares, e $i = 0, 1, \dots, m - 1$
- A primeira posição sondada é $T[h'(k)]$
- As demais são deslocadas de modo quadrático com relação ao passo i

Endereçamento Aberto

Sondagem Quadrática

- Usa uma função de hash da forma
$$h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$$
- onde h' é a função de hash auxiliar, $c_1, c_2 > 0$ são constantes auxiliares, e $i = 0, 1, \dots, m - 1$
- A primeira posição sondada é $T[h'(k)]$
 - As demais são deslocadas de modo quadrático com relação ao passo i
- Como na sondagem linear, a sondagem inicial determina a sequência inteira

Endereçamento Aberto

Sondagem Quadrática

- Usa uma função de hash da forma
$$h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$$
- onde h' é a função de hash auxiliar, $c_1, c_2 > 0$ são constantes auxiliares, e $i = 0, 1, \dots, m - 1$
- A primeira posição sondada é $T[h'(k)]$
 - As demais são deslocadas de modo quadrático com relação ao passo i
- Como na sondagem linear, a sondagem inicial determina a sequência inteira
 - Levando a um limite de m possíveis sequências

Endereçamento Aberto

Sondagem Quadrática

- Funciona melhor que a sondagem linear

Endereçamento Aberto

Sondagem Quadrática

- Funciona melhor que a sondagem linear
 - Mas precisamos ter cuidado na escolha de c_1 e c_2 , se quisermos poder usar a tabela toda

Endereçamento Aberto

Sondagem Quadrática

- Funciona melhor que a sondagem linear
 - Mas precisamos ter cuidado na escolha de c_1 e c_2 , se quisermos poder usar a tabela toda
- Além disso, pode levar à criação de **agrupamentos secundários** (*secondary clustering*)

Endereçamento Aberto

Sondagem Quadrática

- Funciona melhor que a sondagem linear
 - Mas precisamos ter cuidado na escolha de c_1 e c_2 , se quisermos poder usar a tabela toda
- Além disso, pode levar à criação de **agrupamentos secundários** (*secondary clustering*)
 - Caso em que 2 elementos somente terão a mesma sequência de colisões se sua posição inicial for a mesma

Endereçamento Aberto

Sondagem Quadrática

- Funciona melhor que a sondagem linear
 - Mas precisamos ter cuidado na escolha de c_1 e c_2 , se quisermos poder usar a tabela toda
- Além disso, pode levar à criação de **agrupamentos secundários** (*secondary clustering*)
 - Caso em que 2 elementos somente terão a mesma sequência de colisões se sua posição inicial for a mesma
 - Lembre que no agrupamento primário bastava que o 2º elemento começasse em algum ponto da sequência do 1º

Endereçamento Aberto

Sondagem Quadrática

- Funciona melhor que a sondagem linear
 - Mas precisamos ter cuidado na escolha de c_1 e c_2 , se quisermos poder usar a tabela toda
- Além disso, pode levar à criação de **agrupamentos secundários** (*secondary clustering*)
 - Caso em que 2 elementos somente terão a mesma sequência de colisões se sua posição inicial for a mesma
 - Lembre que no agrupamento primário bastava que o 2º elemento começasse em algum ponto da sequência do 1º
 - Trata-se então de uma forma mais leve de agrupamento

Endereçamento Aberto

Hashing Duplo

- Oferece um dos melhores métodos disponíveis para endereçamento aberto

Endereçamento Aberto

Hashing Duplo

- Oferece um dos melhores métodos disponíveis para endereçamento aberto
 - Possui o maior número de sequências de sondagem

Endereçamento Aberto

Hashing Duplo

- Oferece um dos melhores métodos disponíveis para endereçamento aberto
 - Possui o maior número de sequências de sondagem
 - Parece dar os melhores resultados

Endereçamento Aberto

Hashing Duplo

- Oferece um dos melhores métodos disponíveis para endereçamento aberto
 - Possui o maior número de sequências de sondagem
 - Parece dar os melhores resultados
- Usa uma função de hash da forma

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

Endereçamento Aberto

Hashing Duplo

- Oferece um dos melhores métodos disponíveis para endereçamento aberto
 - Possui o maior número de sequências de sondagem
 - Parece dar os melhores resultados
- Usa uma função de hash da forma

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

- Onde h_1, h_2 são funções de hash auxiliares

Endereçamento Aberto

Hashing Duplo

- Com hashing duplo, a sequência de sondagens depende de k de dois modos:

Hashing Duplo

- Com hashing duplo, a sequência de sondagens depende de k de dois modos:
 - Tanto a posição inicial quanto o deslocamento (i) podem variar

Endereçamento Aberto

Hashing Duplo

- Com hashing duplo, a sequência de sondagens depende de k de dois modos:
 - Tanto a posição inicial quanto o deslocamento (i) podem variar
- A sequência de sondagens é então:

i	<i>posição</i>
0	$T[h_1(k)]$
1	$T[(h_1(k) + h_2(k)) \bmod m]$
2	$T[(h_1(k) + 2h_2(k)) \bmod m]$
...	...
$m - 1$	$T[(h_1(k) + (m - 1)h_2(k)) \bmod m]$

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:

- $m = 13$
- $h_1(k) = k \bmod 13$
- $h_2(k) = 1 + (k \bmod 11)$

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$
- Suponha que queremos incluir o 14:

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$
- Suponha que queremos incluir o 14:
 - $h_1(14) = 1, h_2(14) = 4, h(14, 0) = 1$

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$
- Suponha que queremos incluir o 14:
 - $h_1(14) = 1, h_2(14) = 4, h(14, 0) = 1$
 - Colisão.

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$
- Suponha que queremos incluir o 14:
 - $h_1(14) = 1$, $h_2(14) = 4$, $h(14, 0) = 1$
 - Colisão. $h(14, 1) = 5$

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$
- Suponha que queremos incluir o 14:
 - $h_1(14) = 1, h_2(14) = 4, h(14, 0) = 1$
 - Colisão. $h(14, 1) = 5$
 - Colisão.

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$
- Suponha que queremos incluir o 14:
 - $h_1(14) = 1, h_2(14) = 4, h(14, 0) = 1$
 - Colisão. $h(14, 1) = 5$
 - Colisão. $h(14, 2) = 9$

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo – Exemplo

- Considere a seguinte tabela:
 - $m = 13$
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$
- Suponha que queremos incluir o 14:
 - $h_1(14) = 1, h_2(14) = 4, h(14, 0) = 1$
 - Colisão. $h(14, 1) = 5$
 - Colisão. $h(14, 2) = 9$
 - OK

	T
0	
1	79
2	
3	
4	69
5	96
6	
7	72
8	
9	14
10	
11	50
12	

Endereçamento Aberto

Hashing Duplo

- Para que a tabela inteira seja pesquisada, $h_2(k)$ e m devem ser primos entre si

Endereçamento Aberto

Hashing Duplo

- Para que a tabela inteira seja pesquisada, $h_2(k)$ e m devem ser primos entre si
- a e b são primos entre si se o máximo divisor comum entre eles for 1

Endereçamento Aberto

Hashing Duplo

- Para que a tabela inteira seja pesquisada, $h_2(k)$ e m devem ser primos entre si
 - a e b são primos entre si se o máximo divisor comum entre eles for 1
- E como conseguimos isso?

Endereçamento Aberto

Hashing Duplo

- Para que a tabela inteira seja pesquisada, $h_2(k)$ e m devem ser primos entre si
 - a e b são primos entre si se o máximo divisor comum entre eles for 1
- E como conseguimos isso?
 - Escolher m potência de 2 e fazer h_2 sempre produzir um número ímpar

Endereçamento Aberto

Hashing Duplo

- Para que a tabela inteira seja pesquisada, $h_2(k)$ e m devem ser primos entre si
 - a e b são primos entre si se o máximo divisor comum entre eles for 1
- E como conseguimos isso?
 - Escolher m potência de 2 e fazer h_2 sempre produzir um número ímpar
 - Ou escolher m primo e fazer h_2 sempre retornar um inteiro $0 < i < m$

Endereçamento Aberto

Hashing Duplo

- Para o segundo caso, podemos escolher, por exemplo, um m primo e fazer

Endereçamento Aberto

Hashing Duplo

- Para o segundo caso, podemos escolher, por exemplo, um m primo e fazer
 - $h_1(k) = k \bmod m$

Endereçamento Aberto

Hashing Duplo

- Para o segundo caso, podemos escolher, por exemplo, um m primo e fazer
 - $h_1(k) = k \bmod m$
 - $h_2(k) = 1 + (k \bmod m')$, onde escolhemos um m' ligeiramente abaixo de m (ex: $m - 1$)

Endereçamento Aberto

Hashing Duplo

- Para o segundo caso, podemos escolher, por exemplo, um m primo e fazer
 - $h_1(k) = k \bmod m$
 - $h_2(k) = 1 + (k \bmod m')$, onde escolhemos um m' ligeiramente abaixo de m (ex: $m - 1$)
- Note que, em nosso exemplo, usamos esse caso

Endereçamento Aberto

Hashing Duplo

- Para o segundo caso, podemos escolher, por exemplo, um m primo e fazer
 - $h_1(k) = k \bmod m$
 - $h_2(k) = 1 + (k \bmod m')$, onde escolhemos um m' ligeiramente abaixo de m (ex: $m - 1$)
- Note que, em nosso exemplo, usamos esse caso
 - $m = 13$ (primo)

Endereçamento Aberto

Hashing Duplo

- Para o segundo caso, podemos escolher, por exemplo, um m primo e fazer
 - $h_1(k) = k \bmod m$
 - $h_2(k) = 1 + (k \bmod m')$, onde escolhemos um m' ligeiramente abaixo de m (ex: $m - 1$)
- Note que, em nosso exemplo, usamos esse caso
 - $m = 13$ (primo)
 - $h_1(k) = k \bmod 13$

Endereçamento Aberto

Hashing Duplo

- Para o segundo caso, podemos escolher, por exemplo, um m primo e fazer
 - $h_1(k) = k \bmod m$
 - $h_2(k) = 1 + (k \bmod m')$, onde escolhemos um m' ligeiramente abaixo de m (ex: $m - 1$)
- Note que, em nosso exemplo, usamos esse caso
 - $m = 13$ (primo)
 - $h_1(k) = k \bmod 13$
 - $h_2(k) = 1 + (k \bmod 11)$ (ou seja, $m' = m - 2$)

Hashing Duplo

- Quando m é primo ou potência de 2, $\Theta(m^2)$ sequências de sondagem são possíveis

Hashing Duplo

- Quando m é primo ou potência de 2, $\Theta(m^2)$ sequências de sondagem são possíveis
 - Uma vez que h_1 e h_2 geram m possíveis sequências cada um

Hashing Duplo

- Quando m é primo ou potência de 2, $\Theta(m^2)$ sequências de sondagem são possíveis
 - Uma vez que h_1 e h_2 geram m possíveis sequências cada um
 - As mesmas m da sondagem linear e quadrática

Hashing Duplo

- Quando m é primo ou potência de 2, $\Theta(m^2)$ sequências de sondagem são possíveis
 - Uma vez que h_1 e h_2 geram m possíveis sequências cada um
 - As mesmas m da sondagem linear e quadrática
- Como resultado, o hashing duplo se aproxima mais dos $m!$ do hashing uniforme ideal

Referências

- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.
- Ziviani, Nivio. Projeto de Algoritmos: com implementações em Java e C++. Cengage. 2007.
- <https://stackoverflow.com/questions/27742285/what-is-primary-and-secondary-clustering-in-hash>
- https://en.wikipedia.org/wiki/Primary_clustering
- Slides dos professores Delano Beder e Marcos Chaim

Aula 26 – Hashing: Endereçamento Aberto

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2024