Aula 25 – Hashing: Endereçamento Direto e Resolução de Colisões

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

2024

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Imagine que queremos construir uma função que retorne a versão por extenso de um numeral de um único dígito

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Imagine que queremos construir uma função que retorne a versão por extenso de um numeral de um único dígito
  - Onde a chave de busca é o dígito e o valor resultante é sua versão por extenso

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Imagine que queremos construir uma função que retorne a versão por extenso de um numeral de um único dígito
  - Onde a chave de busca é o dígito e o valor resultante é sua versão por extenso
- Queremos então uma função que implemente a seguinte tabela:

Chave	*Valor*
0	"zero"
1	"um"
...	...
9	"nove"

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Mais do que isso, imagine que essa tabela não precisa conter todas as entradas pré-definidas

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Mais do que isso, imagine que essa tabela não precisa conter todas as entradas pré-definidas
  - Ela será atualizada dinamicamente

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Mais do que isso, imagine que essa tabela não precisa conter todas as entradas pré-definidas
  - Ela será atualizada dinamicamente
- Então, no início, não há elemento algum, sendo eles inseridos com o tempo

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Mais do que isso, imagine que essa tabela não precisa conter todas as entradas pré-definidas
  - Ela será atualizada dinamicamente
- Então, no início, não há elemento algum, sendo eles inseridos com o tempo
  - Mas sempre no intervalo entre 0 e 9

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Mais do que isso, imagine que essa tabela não precisa conter todas as entradas pré-definidas
  - Ela será atualizada dinamicamente
- Então, no início, não há elemento algum, sendo eles inseridos com o tempo
  - Mas sempre no intervalo entre 0 e 9
- Como podemos fazer?

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Mais do que isso, imagine que essa tabela não precisa conter todas as entradas pré-definidas
  - Ela será atualizada dinamicamente
- Então, no início, não há elemento algum, sendo eles inseridos com o tempo
  - Mas sempre no intervalo entre 0 e 9
- Como podemos fazer?
  - Criando um Dicionário Dinâmico

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Para isso, definimos o elemento a ser armazenado:

```
typedef struct {
  int chave;
  char* extenso;
} Elemento;

Elemento* novoElemento(int chave, char* ext){
  Elemento* el = (Elemento*) malloc(sizeof(Elemento));
  el->chave = chave;
  el->extenso = ext;
  return el;
}
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- E também uma estrutura para o dicionário:

```
typedef struct {
  Elemento** elementos;
  int n;
} Dicionario;

Dicionario novoDicionario(int n){
  int i;
  Dicionario d;
  d.n = n;
  d.elementos = (Elemento**) malloc(sizeof(Elemento*)*n);
  for (i=0; i<n; i++) d.elementos[i] = NULL;
  return d;
}
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Precisamos então criar as funções para manipulação do dicionário:

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Precisamos então criar as funções para manipulação do dicionário:
  - Inserir um elemento

```
void inserir(Dicionario d,
                    int chave, char* ext){
  d.elementos[chave] =
            novoElemento(chave, ext);
}
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Precisamos então criar as funções para manipulação do dicionário:
  - Inserir um elemento
  - Remover um elemento

```
void inserir(Dicionario d,
                    int chave, char* ext){
  d.elementos[chave] =
            novoElemento(chave, ext);
}
void remover(Dicionario d, int chave){
  if (d.elementos[chave])
    free(d.elementos[chave]);
  d.elementos[chave] = NULL;
}
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Precisamos então criar as funções para manipulação do dicionário:
  - Inserir um elemento
  - Remover um elemento
  - Buscar um elemento

```
void inserir(Dicionario d,
                    int chave, char* ext){
  d.elementos[chave] =
            novoElemento(chave, ext);
}
void remover(Dicionario d, int chave){
  if (d.elementos[chave])
    free(d.elementos[chave]);
  d.elementos[chave] = NULL;
}
Elemento* buscar(Dicionario d,
                              int chave) {
  return d.elementos[chave];
}
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

```
void inserir(Dicionario d, int chave, char* ext){
  d.elementos[chave] = novoElemento(chave, ext);
}

void remover(Dicionario d, int chave) {
  if (d.elementos[chave])
              free(d.elementos[chave]);
  d.elementos[chave] = NULL;
}

Elemento* buscar(Dicionario d, int chave) {
  return d.elementos[chave];
}
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

```
void inserir(Dicionario d, int chave, char* ext){
  d.elementos[chave] = novoElemento(chave, ext);
}

void remover(Dicionario d, int chave) {
  if (d.elementos[chave])
              free(d.elementos[chave]);
  d.elementos[chave] = NULL;
}

Elemento* buscar(Dicionario d, int chave) {
  return d.elementos[chave];
}
```

Note que fizemos um mapeamento direto da chave à posição do elemento no arranjo

Dicionários Dinâmicos

Tabela de Endereçamento Direto

```
void inserir(Dicionario d, int chave, char* ext){
  d.elementos[chave] = novoElemento(chave, ext);
}

void remover(Dicionario d, int chave) {
  if (d.elementos[chave])
              free(d.elementos[chave]);
  d.elementos[chave] = NULL;
}

Elemento* buscar(Dicionario d, int chave) {
  return d.elementos[chave];
}
```

E isso faz com que inserções, buscas e remoções, usando a chave, rodem em $\Theta(1)$

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O endereçamento direto é uma técnica simples que funciona bem quando o universo U de chaves é razoavelmente pequeno

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O endereçamento direto é uma técnica simples que funciona bem quando o universo U de chaves é razoavelmente pequeno
  - O conjunto dinâmico possui m chaves, obtidas do universo $U = \{0, 1, \ldots, m-1\}$, onde m não é muito grande e não há dois elementos com a mesma chave

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O endereçamento direto é uma técnica simples que funciona bem quando o universo U de chaves é razoavelmente pequeno
  - O conjunto dinâmico possui m chaves, obtidas do universo $U = \{0, 1, \ldots, m-1\}$, onde m não é muito grande e não há dois elementos com a mesma chave
- Sob essas condições, podemos representar o conjunto com uma Tabela de Endereçamento Direto

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O endereçamento direto é uma técnica simples que funciona bem quando o universo U de chaves é razoavelmente pequeno
  - O conjunto dinâmico possui m chaves, obtidas do universo $U = \{0, 1, \ldots, m-1\}$, onde m não é muito grande e não há dois elementos com a mesma chave
- Sob essas condições, podemos representar o conjunto com uma Tabela de Endereçamento Direto
  - Onde cada posição corresponde a uma chave no universo U

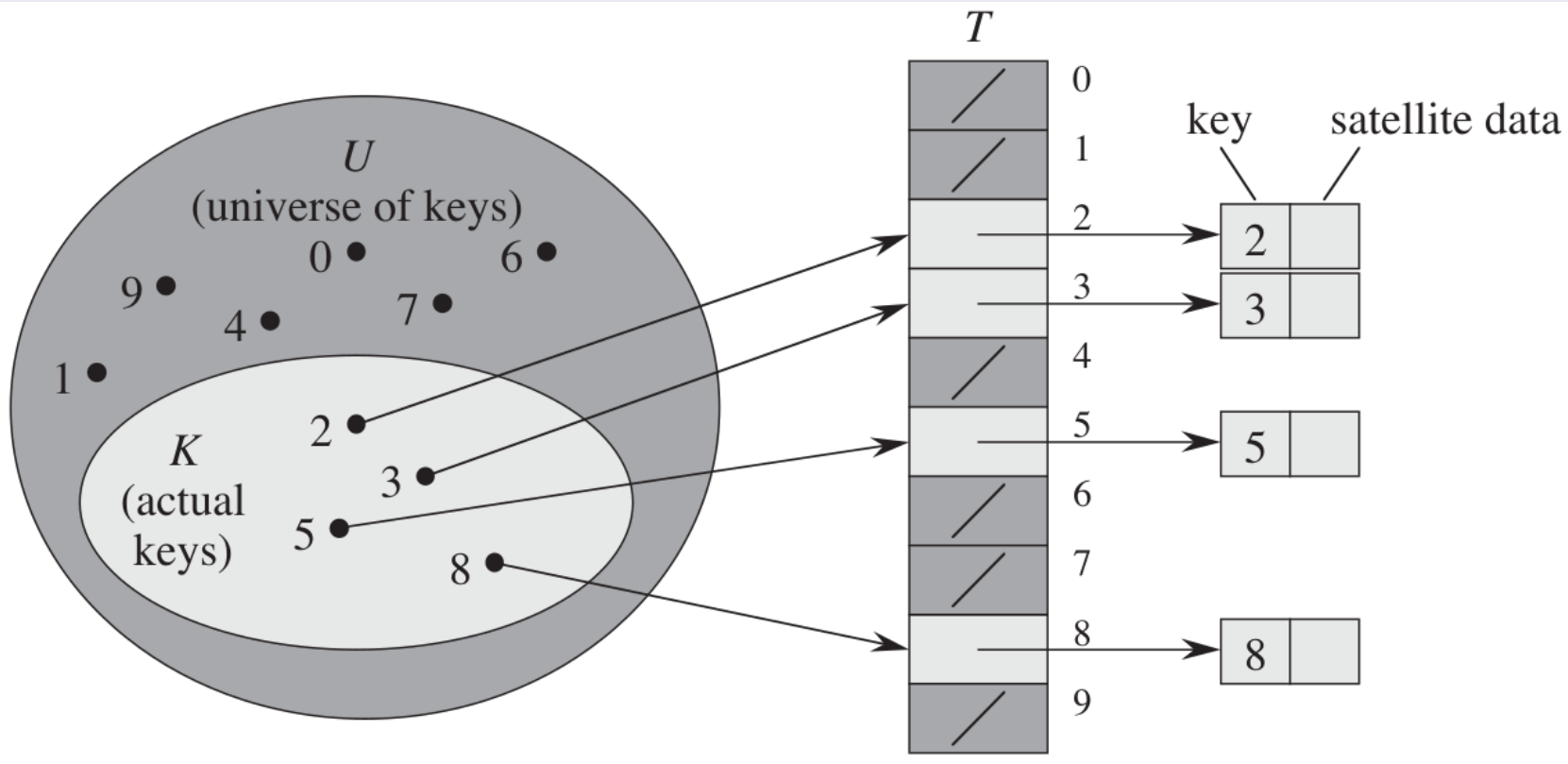
Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O endereçamento direto é uma técnica simples que funciona bem quando o universo U de chaves é razoavelmente pequeno
  - O conjunto dinâmico possui m chaves, obtidas do universo $U = \{0, 1, \ldots, m-1\}$, onde m não é muito grande e não há dois elementos com a mesma chave
- Sob essas condições, podemos representar o conjunto com uma Tabela de Endereçamento Direto
  - Onde cada posição corresponde a uma chave no universo U
  - A tabela será então do tamanho do universo

Dicionários Dinâmicos

Tabela de Endereçamento Direto



O endereçamento direto é aplicável quando podemos nos dar ao luxo de alocar um arranjo que tem uma posição para cada possível chave

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Suponha agora que queremos uma estrutura semelhante, mas para guardar nomes de pessoas

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Suponha agora que queremos uma estrutura semelhante, mas para guardar nomes de pessoas
- Como indexar?

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Suponha agora que queremos uma estrutura semelhante, mas para guardar nomes de pessoas
- Como indexar?
- Uma alternativa é fornecer um identificador a cada pessoa

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Suponha agora que queremos uma estrutura semelhante, mas para guardar nomes de pessoas
- Como indexar?
- Uma alternativa é fornecer um identificador a cada pessoa
  - Uma chave, que comece do 0 e vá até $n - 1$, sendo n o número de pessoas cadastradas

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Suponha agora que queremos uma estrutura semelhante, mas para guardar nomes de pessoas
- Como indexar?
- Uma alternativa é fornecer um identificador a cada pessoa
  - Uma chave, que comece do 0 e vá até $n - 1$, sendo n o número de pessoas cadastradas
- E se precisarmos remover alguém?

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Suponha agora que queremos uma estrutura semelhante, mas para guardar nomes de pessoas
- Como indexar?
- Uma alternativa é fornecer um identificador a cada pessoa
  - Uma chave, que comece do 0 e vá até $n - 1$, sendo n o número de pessoas cadastradas
- E se precisarmos remover alguém?
  - Basta removermos o registro, como fizemos antes

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O problema é que isso gera uma lacuna

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O problema é que isso gera uma lacuna
  - Que podemos preencher com a próxima pessoa a ser registrada

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O problema é que isso gera uma lacuna
  - Que podemos preencher com a próxima pessoa a ser registrada
- Mas então, a cada novo registro, precisamos:

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O problema é que isso gera uma lacuna
  - Que podemos preencher com a próxima pessoa a ser registrada
- Mas então, a cada novo registro, precisamos:
  - Correr a tabela buscando a primeira posição vazia

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O problema é que isso gera uma lacuna
  - Que podemos preencher com a próxima pessoa a ser registrada
- Mas então, a cada novo registro, precisamos:
  - Correr a tabela buscando a primeira posição vazia
  - Atualizar o identificador no registro, dando a ele o valor dessa posição

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O problema é que isso gera uma lacuna
  - Que podemos preencher com a próxima pessoa a ser registrada
- Mas então, a cada novo registro, precisamos:
  - Correr a tabela buscando a primeira posição vazia
  - Atualizar o identificador no registro, dando a ele o valor dessa posição
  - Armazenar o registro lá

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- O problema é que isso gera uma lacuna
  - Que podemos preencher com a próxima pessoa a ser registrada
- Mas então, a cada novo registro, precisamos:
  - Correr a tabela buscando a primeira posição vazia
  - Atualizar o identificador no registro, dando a ele o valor dessa posição
  - Armazenar o registro lá
  - $\Theta(n)$

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Podemos então usar o CPF da pessoa como identificador

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Podemos então usar o CPF da pessoa como identificador

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Podemos então usar o CPF da pessoa como identificador
  - Fazendo o mapeamento direto à tabela

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Podemos então usar o CPF da pessoa como identificador
  - Fazendo o mapeamento direto à tabela
- Volta a ser $\Theta(1)$ no tempo

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Dicionários Dinâmicos

Tabela de Endereçamento Direto

- Podemos então usar o CPF da pessoa como identificador
  - Fazendo o mapeamento direto à tabela
- Volta a ser $\Theta(1)$ no tempo
  - Mas a tabela ficará gigantesca!

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Dicionários Dinâmicos

Tabela de Hash

- O problema, nesse caso, é que o universo U de chaves é muito grande

Dicionários Dinâmicos

Tabela de Hash

- O problema, nesse caso, é que o universo U de chaves é muito grande
  - Armazenar a tabela T pode ser por vezes até impossível

Dicionários Dinâmicos

Tabela de Hash

- O problema, nesse caso, é que o universo U de chaves é muito grande
  - Armazenar a tabela T pode ser por vezes até impossível
  - Além disso, o conjunto de chaves K realmente armazenadas pode ser pequeno, desperdiçando muito espaço na tabela

Dicionários Dinâmicos

Tabela de Hash

- O problema, nesse caso, é que o universo U de chaves é muito grande
  - Armazenar a tabela T pode ser por vezes até impossível
  - Além disso, o conjunto de chaves K realmente armazenadas pode ser pequeno, desperdiçando muito espaço na tabela
- O que fazer então?

Dicionários Dinâmicos

Tabela de Hash

- O problema, nesse caso, é que o universo U de chaves é muito grande
  - Armazenar a tabela T pode ser por vezes até impossível
  - Além disso, o conjunto de chaves K realmente armazenadas pode ser pequeno, desperdiçando muito espaço na tabela
- O que fazer então?
  - Fixar o tamanho da tabela e mapear a chave de cada registro a uma de suas posições

Dicionários Dinâmicos

Tabela de Hash

- O problema, nesse caso, é que o universo U de chaves é muito grande
  - Armazenar a tabela T pode ser por vezes até impossível
  - Além disso, o conjunto de chaves K realmente armazenadas pode ser pequeno, desperdiçando muito espaço na tabela
- O que fazer então?
  - Fixar o tamanho da tabela e mapear a chave de cada registro a uma de suas posições
  - Criamos assim uma **Tabela de Espalhamento** ou **Tabela de Hash**

Tabela de Hash

- Estrutura de dados eficiente para implementar dicionários

Tabela de Hash

- Estrutura de dados eficiente para implementar dicionários
  - Embora a busca por um elemento possa, no pior caso, ser $\Theta(n)$, na prática, o tempo médio é $\Theta(1)$

Tabela de Hash

- Estrutura de dados eficiente para implementar dicionários
  - Embora a busca por um elemento possa, no pior caso, ser $\Theta(n)$, na prática, o tempo médio é $\Theta(1)$
- Trata-se de um mapeamento da chave a uma posição na tabela

Tabela de Hash

- Estrutura de dados eficiente para implementar dicionários
  - Embora a busca por um elemento possa, no pior caso, ser $\Theta(n)$, na prática, o tempo médio é $\Theta(1)$
- Trata-se de um mapeamento da chave a uma posição na tabela
  - Então, em vez de usarmos a chave como índice, calculamos o índice a partir da chave

Tabela de Hash

- Estrutura de dados eficiente para implementar dicionários
  - Embora a busca por um elemento possa, no pior caso, ser $\Theta(n)$, na prática, o tempo médio é $\Theta(1)$
- Trata-se de um mapeamento da chave a uma posição na tabela
  - Então, em vez de usarmos a chave como índice, calculamos o índice a partir da chave
- Tipicamente, a tabela tem tamanho proporcional ao número de chaves realmente armazenadas

Tabela de Hash

- Estrutura de dados eficiente para implementar dicionários
  - Embora a busca por um elemento possa, no pior caso, ser $\Theta(n)$, na prática, o tempo médio é $\Theta(1)$
- Trata-se de um mapeamento da chave a uma posição na tabela
  - Então, em vez de usarmos a chave como índice, calculamos o índice a partir da chave
- Tipicamente, a tabela tem tamanho proporcional ao número de chaves realmente armazenadas
  - Pouco desperdício de memória

Tabela de Hash

Função de Hash

- O mapeamento entre a chave e sua posição na tabela se dá via uma **função de hash**

Tabela de Hash

Função de Hash

- O mapeamento entre a chave e sua posição na tabela se dá via uma **função de hash**
- A função de hash mapeia então o universo de todas as possíveis chaves U às posições da tabela T

Tabela de Hash

Função de Hash

- O mapeamento entre a chave e sua posição na tabela se dá via uma **função de hash**
- A função de hash mapeia então o universo de todas as possíveis chaves U às posições da tabela T
  - $h : U \rightarrow \{0, 1, \ldots, m-1\}$, para $T[0...m-1]$

Tabela de Hash

Função de Hash

- O mapeamento entre a chave e sua posição na tabela se dá via uma **função de hash**
- A função de hash mapeia então o universo de todas as possíveis chaves U às posições da tabela T
  - $h : U \rightarrow \{0, 1, \ldots, m-1\}$, para $T[0...m-1]$
  - Tipicamente $m \ll |U|$

Tabela de Hash

Função de Hash

- O mapeamento entre a chave e sua posição na tabela se dá via uma **função de hash**
- A função de hash mapeia então o universo de todas as possíveis chaves U às posições da tabela T
  - $h : U \rightarrow \{0, 1, \ldots, m-1\}$, para $T[0...m-1]$
  - Tipicamente $m \ll |U|$
- Dizemos que $h(k)$ é o **valor de hash** da chave k

Tabela de Hash

Endereçamento Direto × Hash

- Direto
- Hash

Tabela de Hash

Endereçamento Direto × Hash

- Direto
  - O elemento com chave k é armazenado na posição k
- Hash

Tabela de Hash

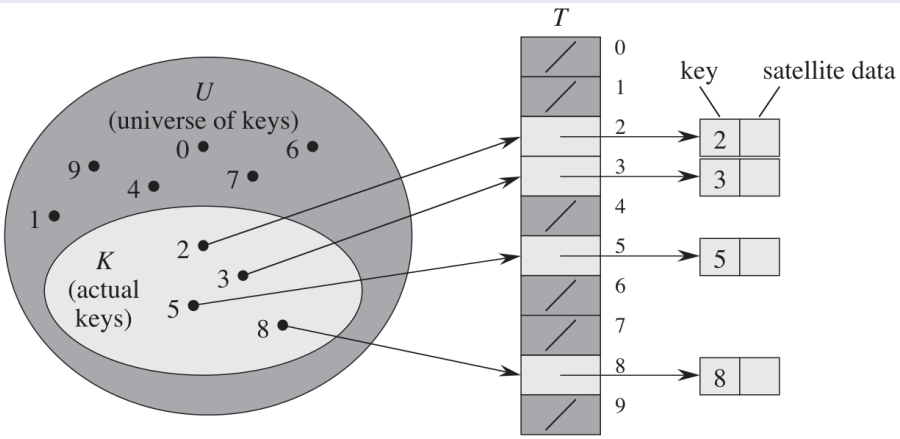
Endereçamento Direto × Hash

- Direto
  - O elemento com chave k é armazenado na posição k
  - $|T| = |U|$
- Hash

Tabela de Hash

Endereçamento Direto × Hash

- Direto
  - O elemento com chave k é armazenado na posição k
  - $|T| = |U|$

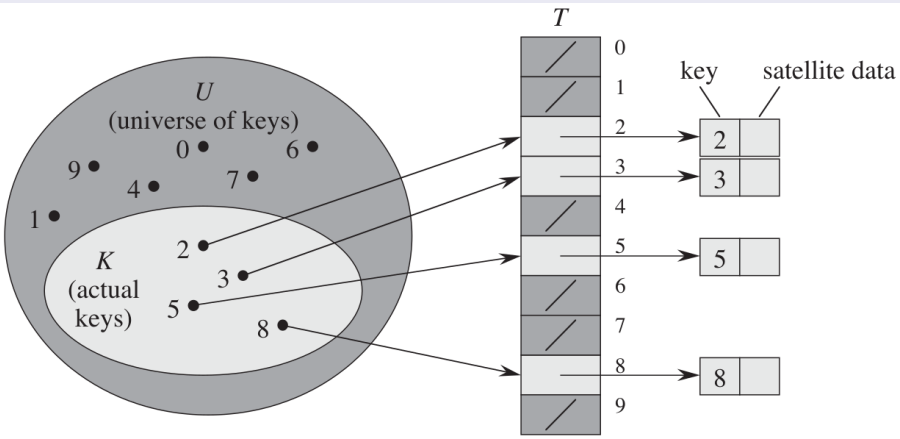


- Hash

Tabela de Hash

Endereçamento Direto × Hash

- Direto
  - O elemento com chave k é armazenado na posição k
  - $|T| = |U|$



- Hash
  - O elemento com chave k é armazenado em $h(k)$

Tabela de Hash

Endereçamento Direto × Hash

- Direto
  - O elemento com chave k é armazenado na posição k
  - $|T| = |U|$
- Hash
  - O elemento com chave k é armazenado em $h(k)$
  - $|T| = m \ll |U|$

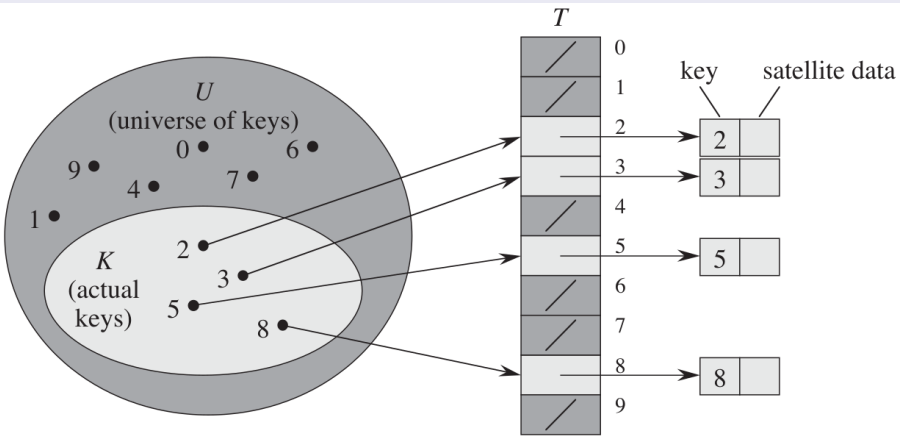
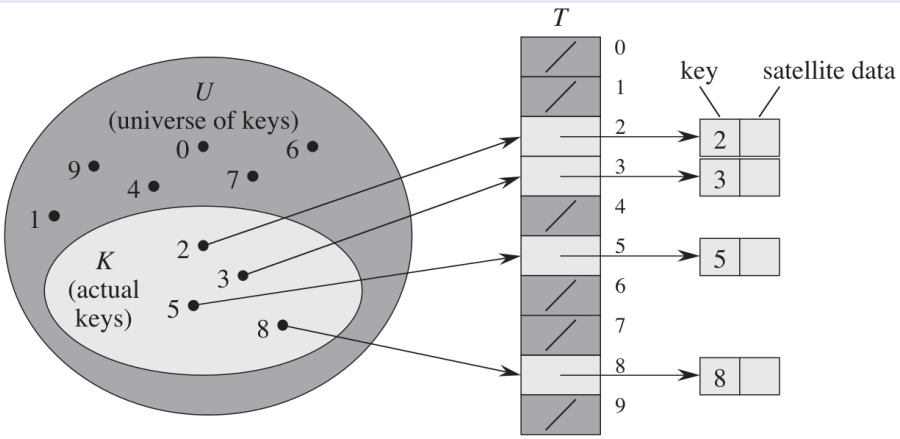


Tabela de Hash

Endereçamento Direto × Hash

- Direto
  - O elemento com chave k é armazenado na posição k
  - $|T| = |U|$
- Hash
  - O elemento com chave k é armazenado em $h(k)$
  - $|T| = m \ll |U|$



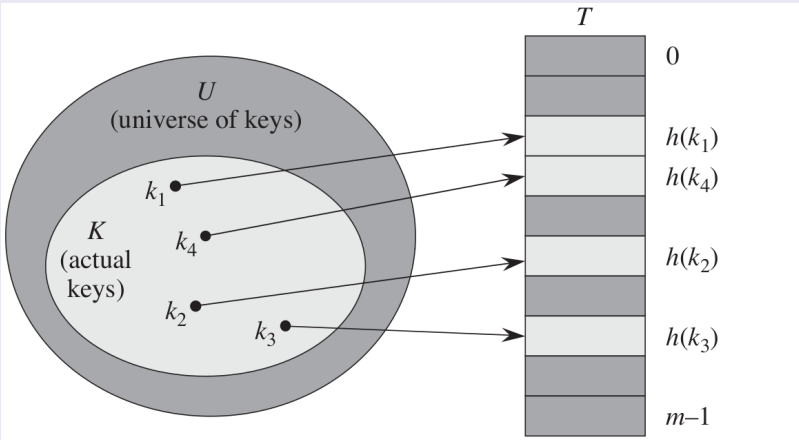


Tabela de Hash

Colisões

- Uma vez que o tamanho m da tabela não necessariamente abriga todas as chaves em uso, **colisões** podem existir

Tabela de Hash

Colisões

- Uma vez que o tamanho m da tabela não necessariamente abriga todas as chaves em uso, **colisões** podem existir
  - Duas ou mais chaves podem ser mapeadas à mesma posição

Tabela de Hash

Colisões

- Uma vez que o tamanho m da tabela não necessariamente abriga todas as chaves em uso, **colisões** podem existir
  - Duas ou mais chaves podem ser mapeadas à mesma posição

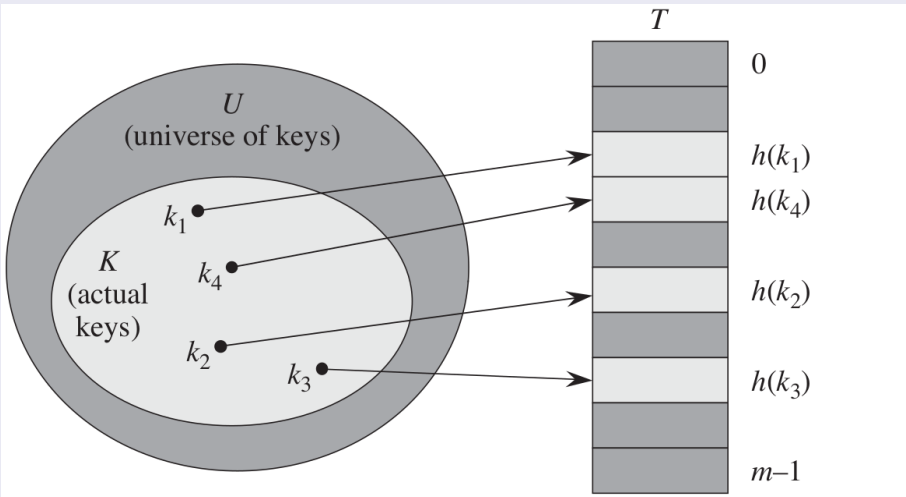
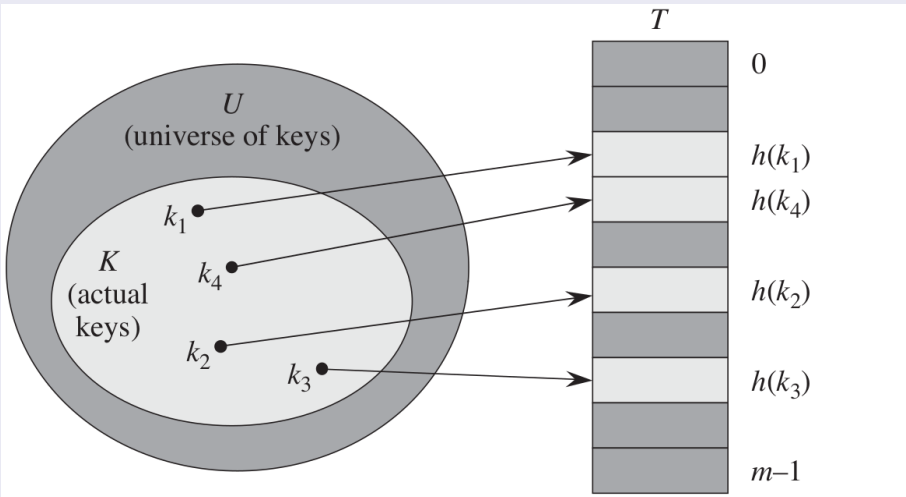


Tabela de Hash

Colisões

- Uma vez que o tamanho m da tabela não necessariamente abriga todas as chaves em uso, **colisões** podem existir
  - Duas ou mais chaves podem ser mapeadas à mesma posição



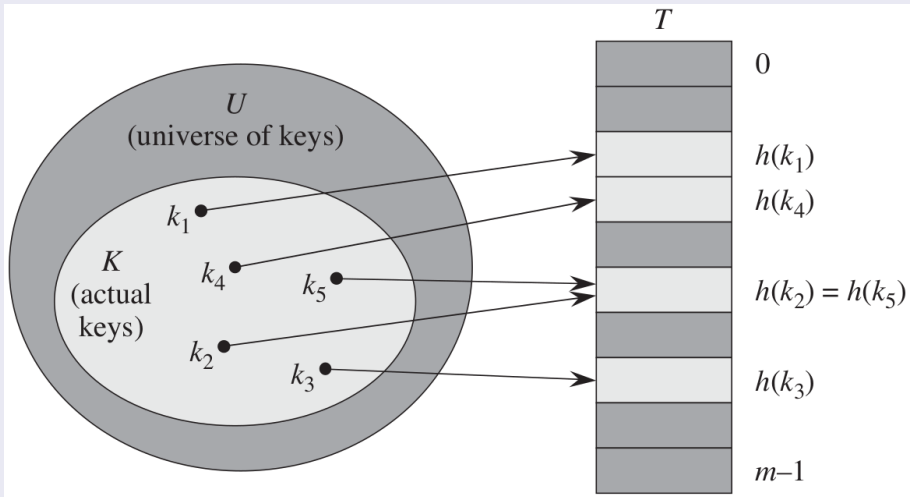
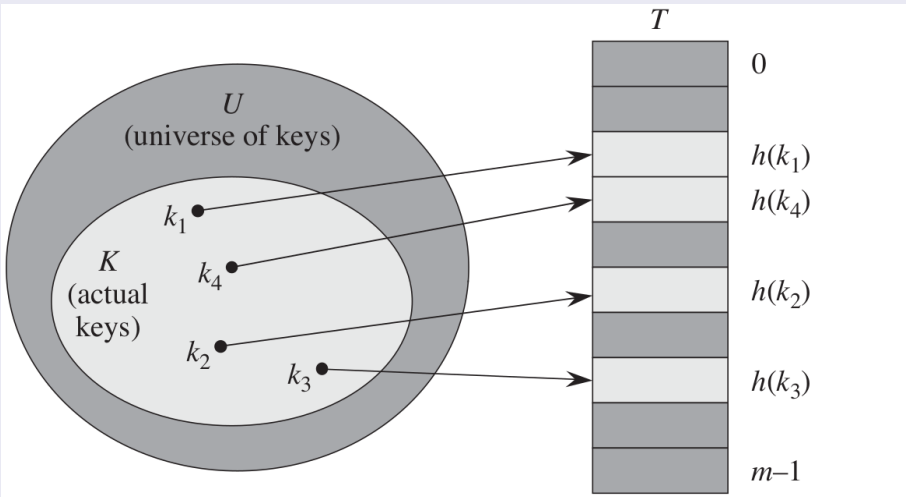
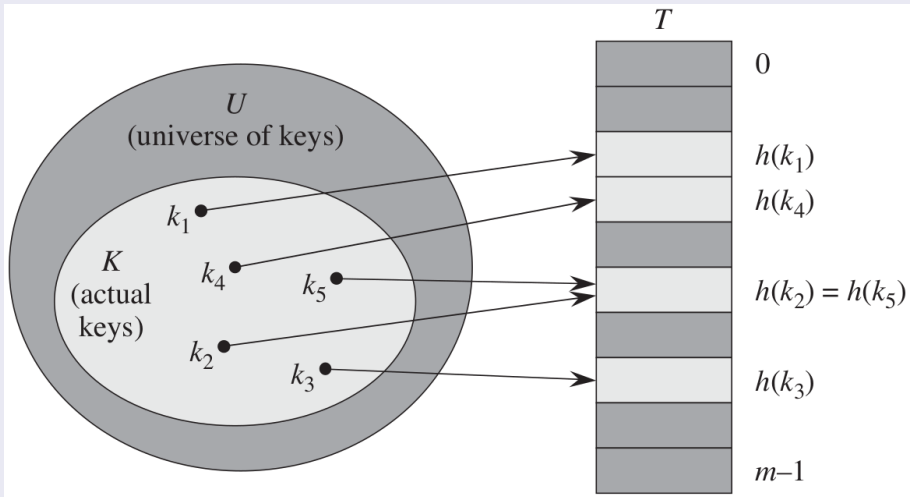


Tabela de Hash

Colisões

- Uma vez que o tamanho m da tabela não necessariamente abriga todas as chaves em uso, **colisões** podem existir
  - Duas ou mais chaves podem ser mapeadas à mesma posição





- Que fazer nesses casos?

Tabela de Hash

Colisões – Encadeamento

- Usamos **encadeamento**

Tabela de Hash

Colisões – Encadeamento

- Usamos **encadeamento**
  - Cada posição j na tabela de hash contém uma lista ligada de todas as chaves cujo valor de hash é $h(k) = j$

Tabela de Hash

Colisões – Encadeamento

- Usamos **encadeamento**
  - Cada posição j na tabela de hash contém uma lista ligada de todas as chaves cujo valor de hash é $h(k) = j$

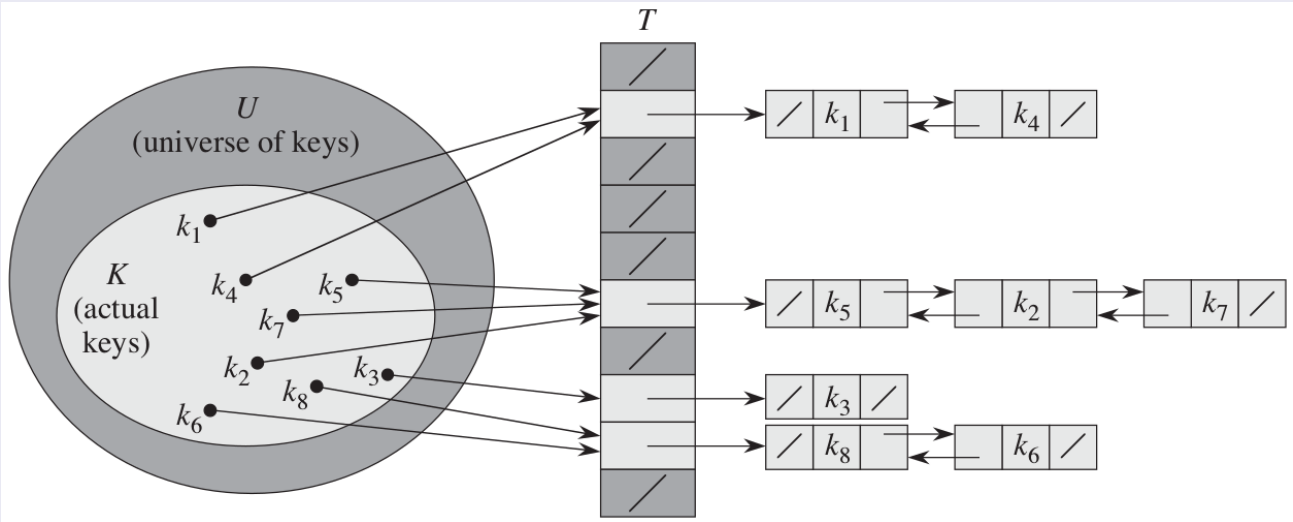
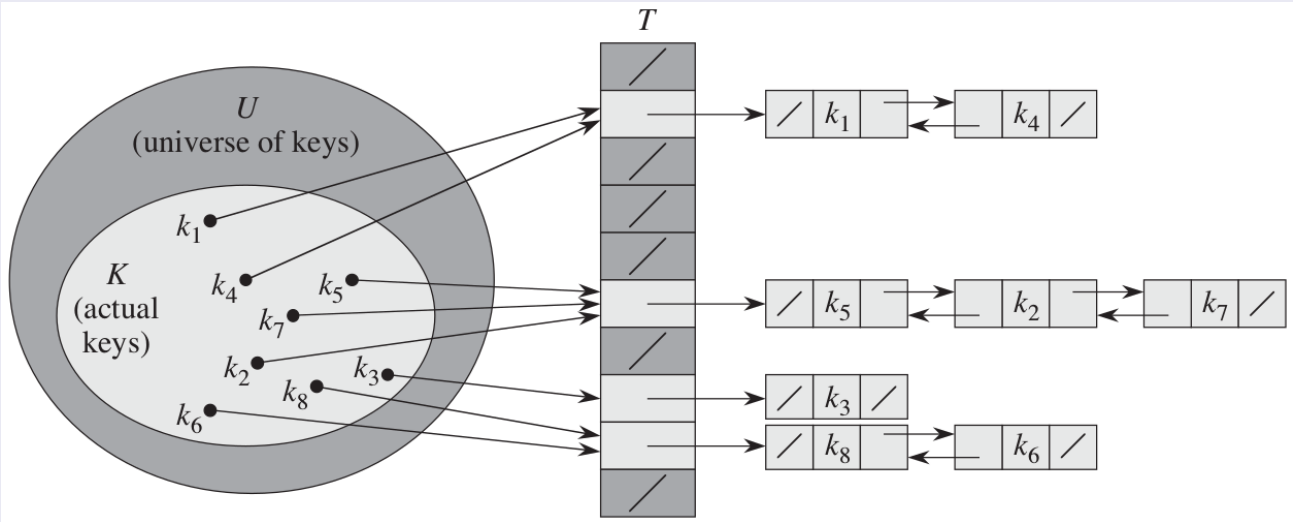


Tabela de Hash

Colisões – Encadeamento

- Usamos **encadeamento**
  - Cada posição j na tabela de hash contém uma lista ligada de todas as chaves cujo valor de hash é $h(k) = j$



  - Ou seja, encadeamos todos os elementos que são mapeados à mesma posição

Tabela de Hash

Encadeamento: Complexidade

- Quanto tempo levamos para incluir um elemento nessa estrutura?

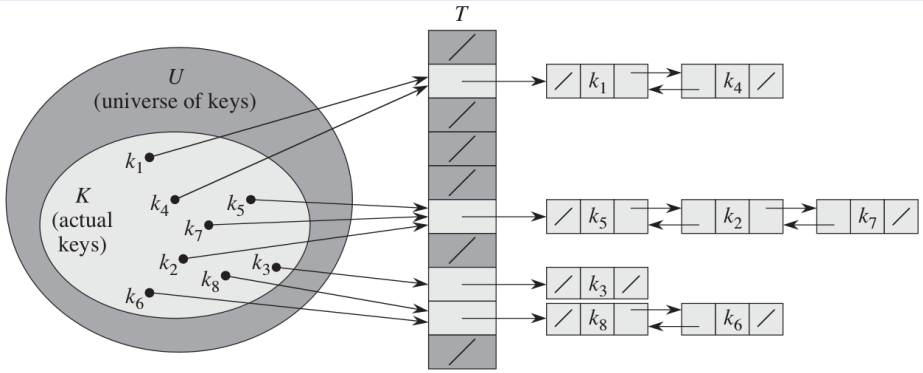


Tabela de Hash

Encadeamento: Complexidade

- Quanto tempo levamos para incluir um elemento nessa estrutura?
  - Hash + acesso à posição: $\Theta(1)$

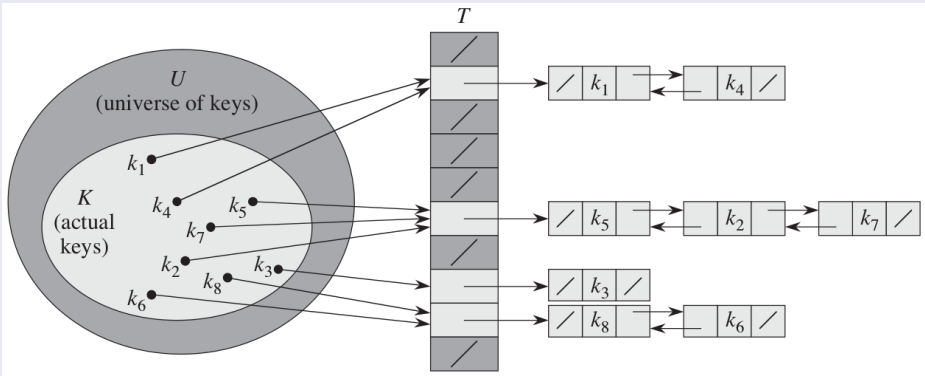


Tabela de Hash

Encadeamento: Complexidade

- Quanto tempo levamos para incluir um elemento nessa estrutura?
  - Hash + acesso à posição: $\Theta(1)$
  - Inclusão na lista: $\Theta(1)$ (para o hashing) $+\Theta(1)$ (para a inclusão), se incluirmos sempre no início da lista em $T[h(k_i)]$

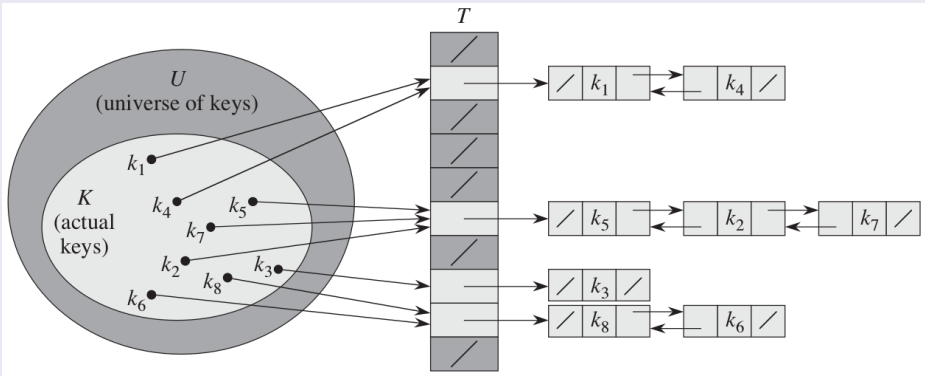
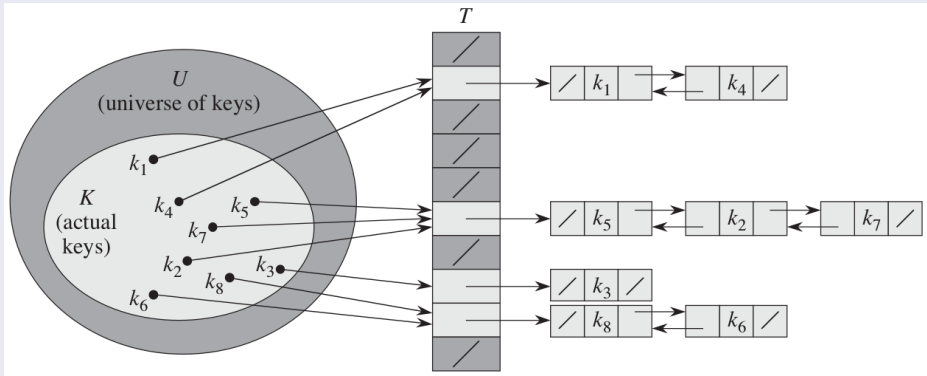


Tabela de Hash

Encadeamento: Complexidade

- Quanto tempo levamos para incluir um elemento nessa estrutura?



  - Hash + acesso à posição: $\Theta(1)$
  - Inclusão na lista: $\Theta(1)$ (para o hashing) $+\Theta(1)$ (para a inclusão), se incluirmos sempre no início da lista em $T[h(k_i)]$
  - Busca e exclusão: $\Theta(1)$ (hashing) $+\Theta(c)$ (busca), onde c é o número de elementos na lista em $T[h(k_i)]$

Tabela de Hash

Encadeamento: Complexidade

- No pior caso, $c = n$

Tabela de Hash

Encadeamento: Complexidade

- No pior caso, $c = n$
  - Todas as n chaves são mapeadas à mesma posição na tabela

Tabela de Hash

Encadeamento: Complexidade

- No pior caso, $c = n$
  - Todas as n chaves são mapeadas à mesma posição na tabela
  - E o custo total será $\Theta(n)$

Tabela de Hash

Encadeamento: Complexidade

- No pior caso, $c = n$
  - Todas as n chaves são mapeadas à mesma posição na tabela
  - E o custo total será $\Theta(n)$
  - Nada diferente de uma lista ligada com todos os elementos

Tabela de Hash

Encadeamento: Complexidade

- No pior caso, $c = n$
  - Todas as n chaves são mapeadas à mesma posição na tabela
  - E o custo total será $\Theta(n)$
  - Nada diferente de uma lista ligada com todos os elementos
- Mas e o caso médio?

Tabela de Hash

Encadeamento: Complexidade

- No pior caso, $c = n$
  - Todas as n chaves são mapeadas à mesma posição na tabela
  - E o custo total será $\Theta(n)$
  - Nada diferente de uma lista ligada com todos os elementos
- Mas e o caso médio?
  - Vai depender de quão boa é nossa função de hash

Tabela de Hash

Encadeamento: Complexidade

- No pior caso, $c = n$
  - Todas as n chaves são mapeadas à mesma posição na tabela
  - E o custo total será $\Theta(n)$
  - Nada diferente de uma lista ligada com todos os elementos
- Mas e o caso médio?
  - Vai depender de quão boa é nossa função de hash
  - Se ela mapear todas as chaves em algumas poucas posições, teremos novamente o mesmo custo da lista ligada

Tabela de Hash

Encadeamento: Função de Hash

- E o que seria, nesse caso, uma boa função de hash?

Tabela de Hash

Encadeamento: Função de Hash

- E o que seria, nesse caso, uma boa função de hash?
  - Uma que faça com que cada um dos n elementos tenha a mesma chance de ser mapeado a qualquer uma das m posições de T, independentemente dos demais elementos

Tabela de Hash

Encadeamento: Função de Hash

- E o que seria, nesse caso, uma boa função de hash?
  - Uma que faça com que cada um dos n elementos tenha a mesma chance de ser mapeado a qualquer uma das m posições de T, independentemente dos demais elementos
  - Essa suposição é chamada **hashing uniforme simples**

Tabela de Hash

Encadeamento: Função de Hash

- E o que seria, nesse caso, uma boa função de hash?
  - Uma que faça com que cada um dos n elementos tenha a mesma chance de ser mapeado a qualquer uma das m posições de T, independentemente dos demais elementos
  - Essa suposição é chamada **hashing uniforme simples**
- Nesse caso, o comprimento esperado de cada lista é de n/m

Tabela de Hash

Encadeamento: Função de Hash

- E o que seria, nesse caso, uma boa função de hash?
  - Uma que faça com que cada um dos n elementos tenha a mesma chance de ser mapeado a qualquer uma das m posições de T, independentemente dos demais elementos
  - Essa suposição é chamada **hashing uniforme simples**
- Nesse caso, o comprimento esperado de cada lista é de n/m
  - Nos levando à complexidade de $\Theta(n/m)$ para buscas e exclusões

Tabela de Hash

Encadeamento: Função de Hash

- E o que seria, nesse caso, uma boa função de hash?
  - Uma que faça com que cada um dos n elementos tenha a mesma chance de ser mapeado a qualquer uma das m posições de T, independentemente dos demais elementos
  - Essa suposição é chamada **hashing uniforme simples**
- Nesse caso, o comprimento esperado de cada lista é de n/m
  - Nos levando à complexidade de $\Theta(n/m)$ para buscas e exclusões
  - $\alpha = n/m$ é chamado de **fator de carga** do hash $\rightarrow$ o número médio de elementos armazenados nas listas

Tabela de Hash

Encadeamento: Função de Hash

- E o que $\Theta(n/m)$ nos diz?

Tabela de Hash

Encadeamento: Função de Hash

- E o que $\Theta(n/m)$ nos diz?
  - Que, se fizermos $m = \Theta(n)$, então $n = \Theta(m)$, e $\Theta(n/m) = \Theta(\Theta(m)/m) = \Theta(1)$

Tabela de Hash

Encadeamento: Função de Hash

- E o que $\Theta(n/m)$ nos diz?
  - Que, se fizermos $m = \Theta(n)$, então $n = \Theta(m)$, e $\Theta(n/m) = \Theta(\Theta(m)/m) = \Theta(1)$
- Ou seja, em média, e sob a condição de que $m = \Theta(n)$, buscas e remoções levam $\Theta(1)$

Tabela de Hash

Encadeamento: Função de Hash

- E o que $\Theta(n/m)$ nos diz?
  - Que, se fizermos $m = \Theta(n)$, então $n = \Theta(m)$, e $\Theta(n/m) = \Theta(\Theta(m)/m) = \Theta(1)$
- Ou seja, em média, e sob a condição de que $m = \Theta(n)$, buscas e remoções levam $\Theta(1)$
- E todas as operações (inserção, remoção e busca) em uma tabela de hash podem ser feitas em um tempo médio de $\Theta(1)$

Tabela de Hash

Encadeamento: Função de Hash

- E como podemos implementar a condição do hashing uniforme simples?

Tabela de Hash

Encadeamento: Função de Hash

- E como podemos implementar a condição do hashing uniforme simples?
  - Não podemos, pois raramente conhecemos a distribuição de probabilidade dos dados a serem armazenados

Tabela de Hash

Encadeamento: Função de Hash

- E como podemos implementar a condição do hashing uniforme simples?
  - Não podemos, pois raramente conhecemos a distribuição de probabilidade dos dados a serem armazenados
  - E mesmo que tivéssemos, pode ocorrer que elas não sejam obtidas de forma independente

Tabela de Hash

Encadeamento: Função de Hash

- E como podemos implementar a condição do hashing uniforme simples?
  - Não podemos, pois raramente conhecemos a distribuição de probabilidade dos dados a serem armazenados
  - E mesmo que tivéssemos, pode ocorrer que elas não sejam obtidas de forma independente
- Se soubéssemos, por exemplo, que as chaves estão uniformemente distribuídas entre 0 e 1, então a função $h(k) = \lfloor km \rfloor$ satisfaz essa condição

Tabela de Hash

Função de Hash: Método da Divisão

- O que fazer então?

Tabela de Hash

Função de Hash: Método da Divisão

- O que fazer então?
  - Usar alguma heurística que funcione em geral bem

Tabela de Hash

Função de Hash: Método da Divisão

- O que fazer então?
  - Usar alguma heurística que funcione em geral bem
- Uma possibilidade é o **método da divisão**:

Tabela de Hash

Função de Hash: Método da Divisão

- O que fazer então?
  - Usar alguma heurística que funcione em geral bem
- Uma possibilidade é o **método da divisão**:
  - Mapeamos uma chave k a uma das m posições fazendo $h(k) = k \ mod \ m$

Tabela de Hash

Função de Hash: Método da Divisão

- O que fazer então?
  - Usar alguma heurística que funcione em geral bem
- Uma possibilidade é o **método da divisão**:
  - Mapeamos uma chave k a uma das m posições fazendo $h(k) = k \; mod \; m$
  - Onde *mod* é o resto da divisão $\rightarrow$ equivalente ao operador % em C

Tabela de Hash

Função de Hash: Método da Divisão

- O que fazer então?
  - Usar alguma heurística que funcione em geral bem
- Uma possibilidade é o **método da divisão**:
  - Mapeamos uma chave k a uma das m posições fazendo $h(k) = k \; mod \; m$
  - Onde *mod* é o resto da divisão $\rightarrow$ equivalente ao operador % em C
  - Computar $h(k)$ torna-se $O(1)$

Tabela de Hash

Função de Hash: Método da Divisão

- O que fazer então?
  - Usar alguma heurística que funcione em geral bem
- Uma possibilidade é o **método da divisão**:
  - Mapeamos uma chave k a uma das m posições fazendo $h(k) = k \; mod \; m$
  - Onde *mod* é o resto da divisão $\rightarrow$ equivalente ao operador % em C
  - Computar $h(k)$ torna-se $O(1)$
  - Escolher, para m, um número primo não muito perto de uma potência de 2 é frequentemente uma boa escolha

Tabela de Hash

Função de Hash: Chaves não Numéricas

- Até agora, assumimos que a chave é um inteiro

Tabela de Hash

Função de Hash: Chaves não Numéricas

- Até agora, assumimos que a chave é um inteiro
- E se for o caso de ser um caractere?

Tabela de Hash

Função de Hash: Chaves não Numéricas

- Até agora, assumimos que a chave é um inteiro
- E se for o caso de ser um caractere?
  - Basta mapeá-lo a um inteiro (seu ASCII, por exemplo)

Tabela de Hash

Função de Hash: Chaves não Numéricas

- Até agora, assumimos que a chave é um inteiro
- E se for o caso de ser um caractere?
  - Basta mapeá-lo a um inteiro (seu ASCII, por exemplo)
- E se for um String?

Tabela de Hash

Função de Hash: Chaves não Numéricas

- Até agora, assumimos que a chave é um inteiro
- E se for o caso de ser um caractere?
  - Basta mapeá-lo a um inteiro (seu ASCII, por exemplo)
- E se for um String?
  - Podemos gerar uma chave inteira k fazendo $k = \sum_{i=0}^{n-1} c_i \times p_i$

    onde n é o número de caracteres da chave, c_i é o ASCII (ou Unicode) do i-ésimo caractere da chave, e p_i um peso de um conjunto gerado previamente, de forma aleatória

Tabela de Hash

Exemplo

- Voltemos agora ao nosso exemplo

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Tabela de Hash

Exemplo

- Voltemos agora ao nosso exemplo
- Quantas pessoas teremos, em média?

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Tabela de Hash

Exemplo

- Voltemos agora ao nosso exemplo
- Quantas pessoas teremos, em média?
  - Não sabemos, vamos estimar em 100

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Tabela de Hash

Exemplo

- Voltemos agora ao nosso exemplo
- Quantas pessoas teremos, em média?
  - Não sabemos, vamos estimar em 100
  - Então $|T| = m = 101 \rightarrow$ um primo não muito perto de potências de 2 ($2^6 = 64$ e $2^7 = 128$)

```
typedef struct {
  int cpf;
  char* nome;
} Pessoa;
```

Tabela de Hash

Exemplo – Passos:

Tabela de Hash

Exemplo – Passos:

1. Criar uma tabela de hash com $m = 101$ posições

Tabela de Hash

Exemplo – Passos:

1. Criar uma tabela de hash com $m = 101$ posições
2. Encontrar uma função de hash $h(k)$ que mapeie o cpf a uma posição i na tabela $(0 \leq i < 101)$

Tabela de Hash

Exemplo – Passos:

1. Criar uma tabela de hash com $m = 101$ posições
2. Encontrar uma função de hash $h(k)$ que mapeie o cpf a uma posição i na tabela $(0 \leq i < 101)$
   - Esta função deve ter complexidade assintótica $\Theta(1)$

Tabela de Hash

Exemplo – Passos:

1. Criar uma tabela de hash com $m = 101$ posições
2. Encontrar uma função de hash $h(k)$ que mapeie o cpf a uma posição i na tabela $(0 \leq i < 101)$
   - Esta função deve ter complexidade assintótica $\Theta(1)$
   - $h(k) = k \; mod \; m$

Tabela de Hash

Exemplo – Passos:

1. Criar uma tabela de hash com $m = 101$ posições
2. Encontrar uma função de hash $h(k)$ que mapeie o cpf a uma posição i na tabela $(0 \leq i < 101)$
   - Esta função deve ter complexidade assintótica $\Theta(1)$
   - $h(k) = k \; mod \; m$
3. Tratar as possíveis colisões

Tabela de Hash

Exemplo – Passos:

1. Criar uma tabela de hash com $m = 101$ posições
2. Encontrar uma função de hash $h(k)$ que mapeie o cpf a uma posição i na tabela $(0 \leq i < 101)$
   - Esta função deve ter complexidade assintótica $\Theta(1)$
   - $h(k) = k \; mod \; m$
3. Tratar as possíveis colisões
   - Uma vez que 101 é uma média, pode haver mais pessoas que isso, gerando colisões

Tabela de Hash

Exemplo

```
#include <stdio.h>
#include <stdlib.h>

#define true 1
#define false 0

typedef int bool;

typedef struct {
  int cpf;
  char* nome;
} Pessoa;

...
```

Tabela de Hash

Exemplo

```
...

typedef struct aux {
  Pessoa* pessoa;
  struct aux* prox;
} Elemento;

typedef struct {
  Elemento** tabela;
  int n;
} Dicionario;

  ...
```

Tabela de Hash

Exemplo

```
...

Dicionario novoDicionario(int n){
  int i;
  Dicionario d;
  d.n = n;
  d.tabela = (Elemento**) malloc(sizeof(Elemento*)*n);
  for (i=0; i<n; i++) d.tabela[i] = NULL;
  return d;
}

Elemento* novoElemento(Pessoa* pes) {
  Elemento* el = (Elemento*) malloc(sizeof(Elemento));
  el->pessoa = pes;
  el->prox = NULL;
  return el;
}
```

Tabela de Hash

Exemplo

```
...

Pessoa* novaPessoa(int cpf, char* nome){
  Pessoa* pes = (Pessoa*) malloc(sizeof(Pessoa));
  pes->cpf = cpf;
  pes->nome = nome;
  return pes;
}

int hash(int cpf, int n){
  return cpf % n;
}

...
```

Tabela de Hash

Exemplo

```
...

Pessoa* novaPessoa(int cpf, char* nome){
  Pessoa* pes = (Pessoa*) malloc(sizeof(Pessoa));
  pes->cpf = cpf;
  pes->nome = nome;
  return pes;
}

int hash(int cpf, int n){
  return cpf % n;
}

...
```

Tabela de Hash

Exemplo

```
...

void inserir(Dicionario d, Pessoa* pes){
  int pos = hash(pes->cpf,d.n);
  Elemento* el = novoElemento(pes);
  el->prox = d.tabela[pos];
  d.tabela[pos] = el;
}

...
```

Tabela de Hash

Exemplo

```
...

Pessoa* buscar(Dicionario d, int cpf) {
  Elemento* atual;
  int pos = hash(cpf, d.n);
  atual = d.tabela[pos];
  while (atual) {
      if (atual->pessoa->cpf == cpf) return atual->pessoa;
      atual = atual->prox;
  }
  return NULL;
}

...
```

Tabela de Hash

Exemplo

```
...
bool remover(Dicionario d, int cpf) {
  int pos = hash(cpf, d.n);
  Elemento* atual = d.tabela[pos];
  Elemento* ant = NULL;
  while (atual) {
    if (atual->pessoa->cpf == cpf){
      if (ant == NULL) d.tabela[pos] = d.tabela[pos]->prox;
      else ant->prox = atual->prox;
      return true;
    }
    ant = atual;
    atual = atual->prox;
  }
  return false;
}
```

Tabela de Hash

Observações

- Hashing é um modo eficiente de se implementar uma estrutura em que inclusões, buscas e exclusões são feitas, em média, em $O(1)$

Tabela de Hash

Observações

- Hashing é um modo eficiente de se implementar uma estrutura em que inclusões, buscas e exclusões são feitas, em média, em $O(1)$
  - Desde que tomados certos cuidados...

Tabela de Hash

Observações

- Hashing é um modo eficiente de se implementar uma estrutura em que inclusões, buscas e exclusões são feitas, em média, em $O(1)$
  - Desde que tomados certos cuidados...
- Construção da função de Hash:
  - Outros métodos existem, como o da multiplicação e hashing universal

Tabela de Hash

Observações

- Hashing é um modo eficiente de se implementar uma estrutura em que inclusões, buscas e exclusões são feitas, em média, em $O(1)$
  - Desde que tomados certos cuidados...
- Construção da função de Hash:
  - Outros métodos existem, como o da multiplicação e hashing universal
  - Consulte Cormen et al. (2001) para mais detalhes

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.

Aula 25 – Hashing: Endereçamento Direto e Resolução de Colisões

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

2024