

Tabela Hash

Prof. Flavio B. Gonzaga
flavio.gonzaga@unifal-mg.edu.br
Universidade Federal de Alfenas
UNIFAL-MG

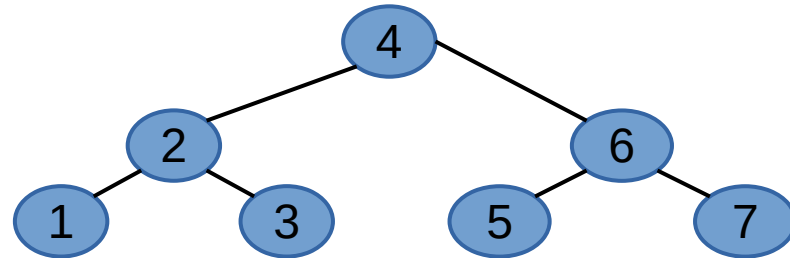
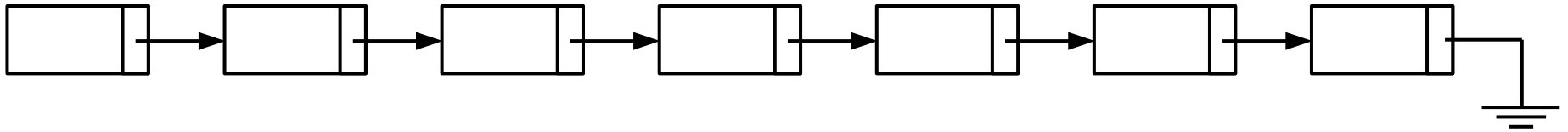
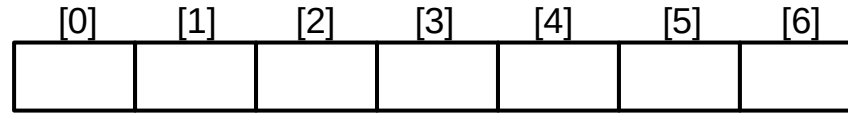
Sumário

- Tabela Hash
 - Introdução
 - Funções de Hashing (alguns exemplos)
 - Hashing Universal
 - Tratamento de colisões:
 - Endereçamento aberto
 - Sondagem Linear
 - Sondagem Quadrática
 - Duplo Hash
 - Encadeamento separado

Introdução

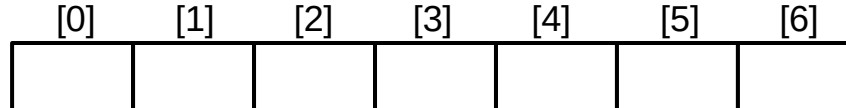
- Vimos até aqui aplicações para a busca de informações em vetores (arrays), listas encadeadas, árvores...
- Independente da estrutura escolhida, a tarefa de buscar pela informação consiste em percorrer a estrutura aonde os dados estão gravados, checando se o dado desejado está presente ou não.

Introdução



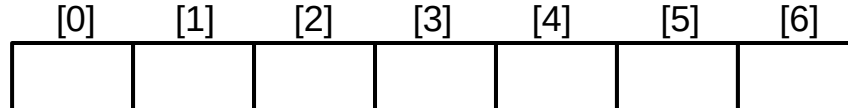
Introdução

- Arrays são estruturas que utilizam índices para armazenar informações
- O tempo de acesso a qualquer posição possui complexidade $O(1)$, mas...
 - Uma pessoa que deseje saber se uma informação está ou não presente em um array, ainda terá que percorrê-lo procurando
 - Isso porque, não é possível saber, com base na informação que se busca, em que posição do array a mesma seria gravada (caso já tenha sido gravada)
 - Assim, a busca em um array não é $O(1)$, mas sim $O(n)$ no pior caso

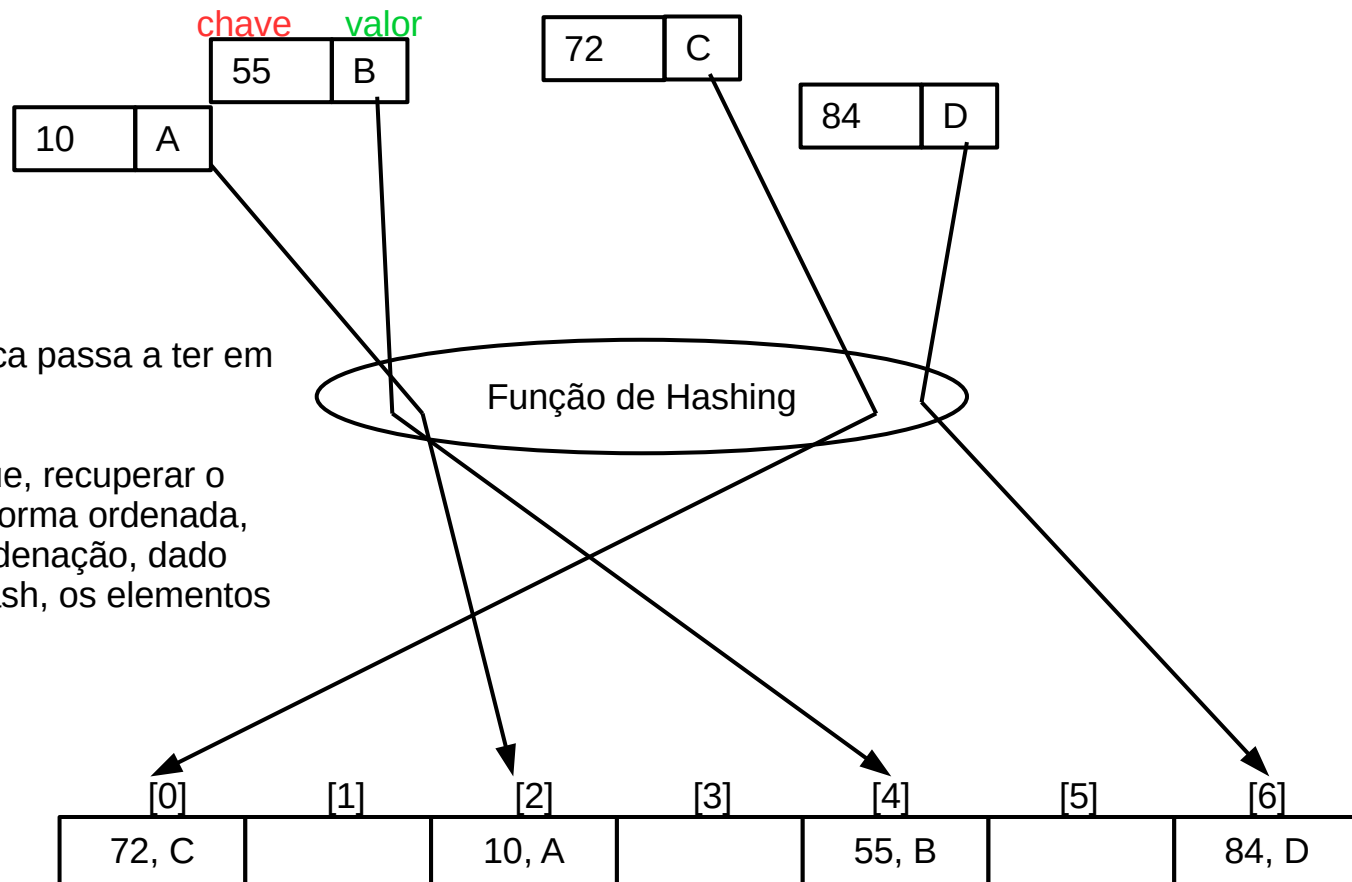


Introdução

- A proposta que “resolve” essa questão, pode ser implementada através da chamada Tabela Hash, também conhecida como Tabela de Espalhamento
- A ideia é você espalhar os dados em um array, tomando como base o dado em si
- Esse espalhamento, definição da posição onde o dado será gravado, é dada por uma função de hashing



Introdução



Posteriormente, a busca passa a ter em média, $O(1)$.

Observe no entanto que, recuperar o conteúdo do array de forma ordenada, possuirá o custo de ordenação, dado que em uma tabela Hash, os elementos não estão ordenados.

Funções de Hashing

- Uma função de hashing deve possuir as seguintes propriedades:
 - Ser simples e barata computacionalmente de se calcular
 - Garantir que valores diferentes produzam posições diferentes
 - Gerar uma distribuição equilibrada dos dados na tabela
- Importante: a implementação da função de hashing depende do conhecimento prévio da natureza dos dados a serem gravados

Funções de Hashing

- Método da divisão, ou congruência linear
 - Consiste em calcular o resto da divisão entre o valor, que representa o elemento a ser inserido, e o tamanho do array

```
int chave_divisao(int chave, int TABLE_SIZE){  
    return chave % TABLE_SIZE;  
}
```

Funções de Hashing

- Método da multiplicação, ou congruência linear multiplicativo
 - Usa uma constante $0 < A < 1$
 - A constante é multiplicada pelo valor, que representa o elemento, em seguida, a parte fracionária resultante é multiplicada pelo tamanho da tabela, resultando na posição do elemento

```
int chave_multiplicacao(int chave, int TABLE_SIZE) {  
    double A = 0.7834729723;  
    double val = chave * A;  
    val = val - (int) val;  
    return (int) (val * TABLE_SIZE);  
}
```

Funções de Hashing

```
int chave_multiplicacao(int chave, int TABLE_SIZE) {  
    double A = 0.7834729723;  
    double val = chave * A;  
    val = val - (int) val;  
    return (int) (val * TABLE_SIZE);  
}
```

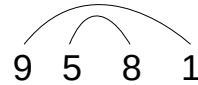
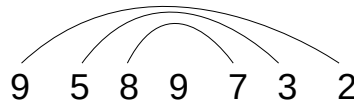
```
chave = 578  
TABLE_SIZE = 1500  
 $0 < A < 1$   
=====  
val = 452,847377989  
val = 0,847377989  
return 0,847377989 * 1500  
  
1271
```

Funções de Hashing

- Método da dobra;
 - Consiste em “dobrar” e somar os dígitos resultantes na dobra, para calcular a posição (desprezando-se as dezenas)
 - Enquanto o valor obtido for maior que o tamanho da tabela, segue-se dobrando...

Funções de Hashing

- Método da dobra:
 - Tamanho da tabela: 1000
 - Chave: 9589732



3 0

Hashing Universal

- A função de hashing está sujeita ao problema de gerar posições iguais para chaves diferentes
- Conhecendo a função de hashing, é possível gerar chaves de maneira tal que todas colidam, diminuindo o desempenho da tabela na busca para $O(n)$

Hashing Universal

- Hashing universal é uma estratégia que busca minimizar o problema de colisões
- A proposta original de hashing universal foi feita por Carter e Wegman, e consistia no seguinte:
 - Escolha um número primo p que seja maior do que qualquer chave k que será inserida na hash, assim: $0 \leq k < p$
 - p também é maior do que o tamanho da tabela
 - Escolha aleatoriamente duas constantes inteiras a e b , de modo que:
 - $0 < a < p; 0 \leq b < p$
- A função de hashing universal será então:
 - $h(k)_{\{a,b\}} = ((ak + b) \% p) \% m$
 - Onde m é o tamanho da tabela

Hashing Universal

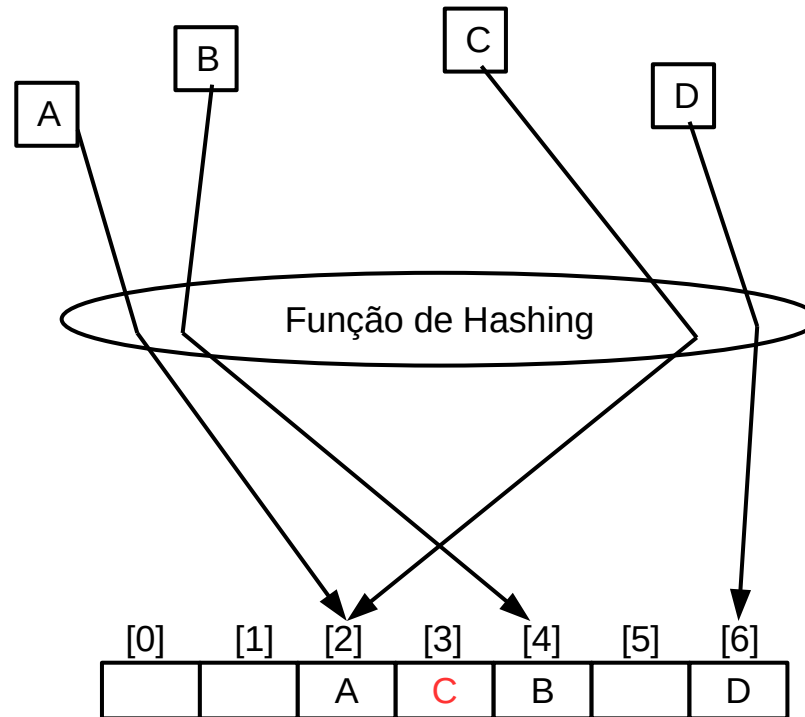
- Com a hashing universal, garante-se matematicamente que a probabilidade de que dois valores diferentes caiam na mesma posição será menor ou igual a $1/m$, para quaisquer valores escolhidos

Tratamento de Colisões

- Endereçamento aberto (ou rehash):
 - No caso de haver colisão, percorre-se a tabela procurando por outra posição disponível
- Encadeamento separado:
 - Dentro de cada posição da tabela existe uma lista encadeada, de modo que, caso haja colisão, o elemento é simplesmente inserido na lista

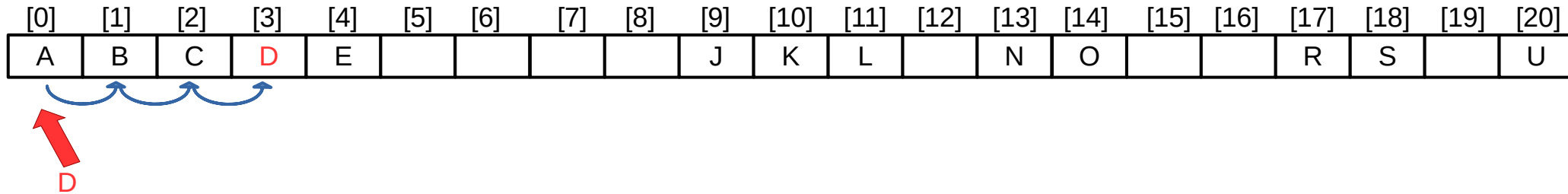
Tratamento de Colisões

- Endereçamento aberto (ou rehash):



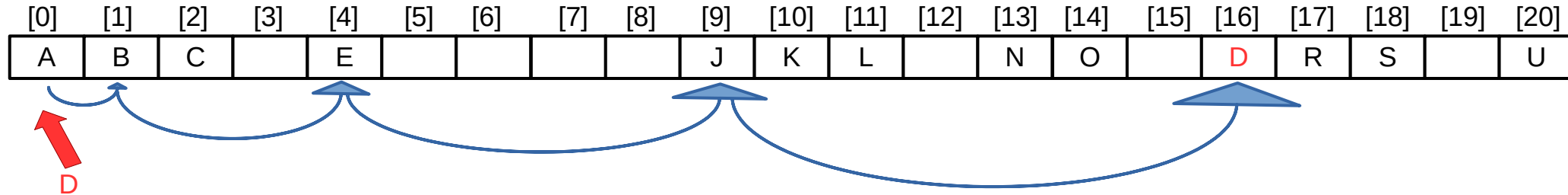
Tratamento de Colisões

- Sondagem Linear: procura-se pela próxima posição livre na tabela, saltando-se de 1 em 1



Tratamento de Colisões

- Sondagem Quadrática: procura-se pela próxima posição livre na tabela, a ser definida através de alguma função quadrática



Tratamento de Colisões

- Duplo Hash: Tenta espalhar os elementos usando duas funções de hash:
 - $H1 + i * H2$
 - Onde, usa-se a primeira função de hashing (H1) para calcular a posição inicial do elemento
 - Caso haja colisão, H2 é usada para calcular o deslocamento, em função da posição inicialmente definida por H1

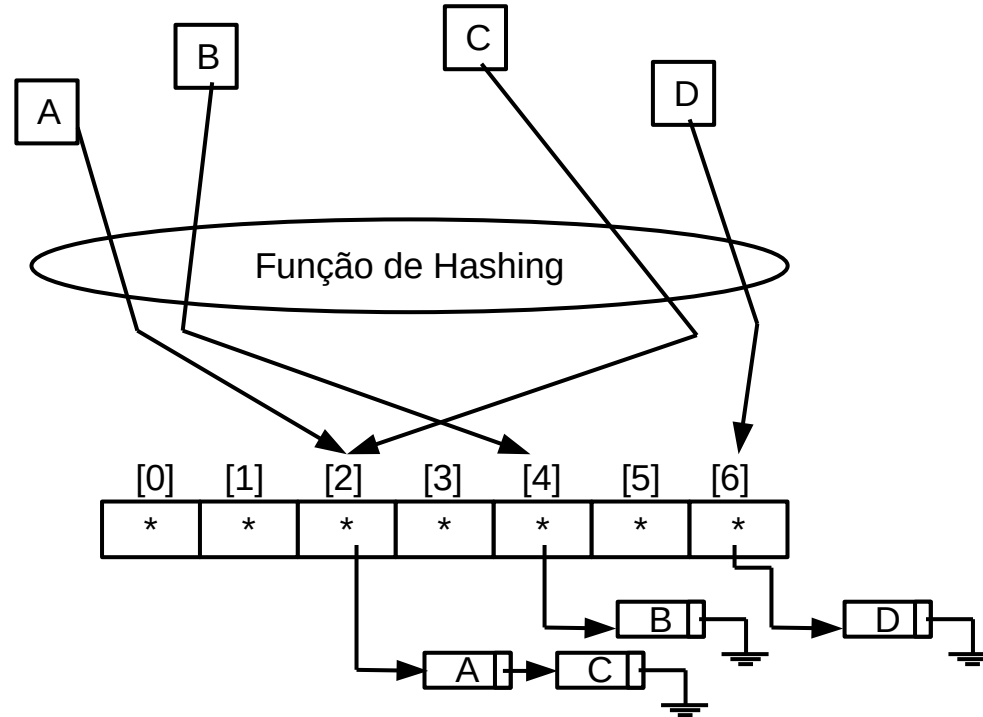
Tratamento de Colisões

- Duplo Hash: Tenta espalhar os elementos usando duas funções de hash:
 - $H1 + i * H2$
 - Onde, usa-se a primeira função de hashing (H1) para calcular a posição inicial do elemento
 - Caso haja colisão, H2 é usada para calcular o deslocamento, em função da posição inicialmente definida por H1

```
int duplo_hash(int H1, int chave, int i, int TABLE_SIZE) {  
    //garantindo que a funcao chave_divisao nao retorne 0  
    int H2 = chave_divisao(chave, TABLE_SIZE - 1) + 1;  
    return ((H1 + i * H2) % TABLE_SIZE);  
}
```

Tratamento de Colisões

- Encadeamento separado:



Referências Bibliográficas

- Estruturas de Dados e Seus Algoritmos. Szwarcfiter J. L.; Markenzon L.. 3a Edição. Editora LTC. 2010.
- Estruturas De Dados Usando C. Tenenbaum A. M.; Langsam Y.; Augenstein M. J.. 1a Edição. Editora Pearson. 1995.
- Introdução a Estruturas de Dados: Com Técnicas de Programação em C. Celes W.; Cerqueira R.; Rangel J.. 2a Edição. Editora Elsevier. 2017.
- https://en.wikipedia.org/wiki/Universal_hashing , acesso em 30/03/2022.
- <https://www.youtube.com/channel/UCUc6UwvpQfOLDE7e52-OCMw> , acesso em 30/03/2022.