Busca e Tabelas Hash

Prof. Lilian Berton

São José dos Campos, 2018

Busca sequencial em vetores

- Procurar um valor numa lista ou vetor:
  pesquisa 5 em [8,2,1,5,2,6] → True

  pesquisa 5 em [8,2,1,3,2,6] → False

  x é comparado sucessivamente com

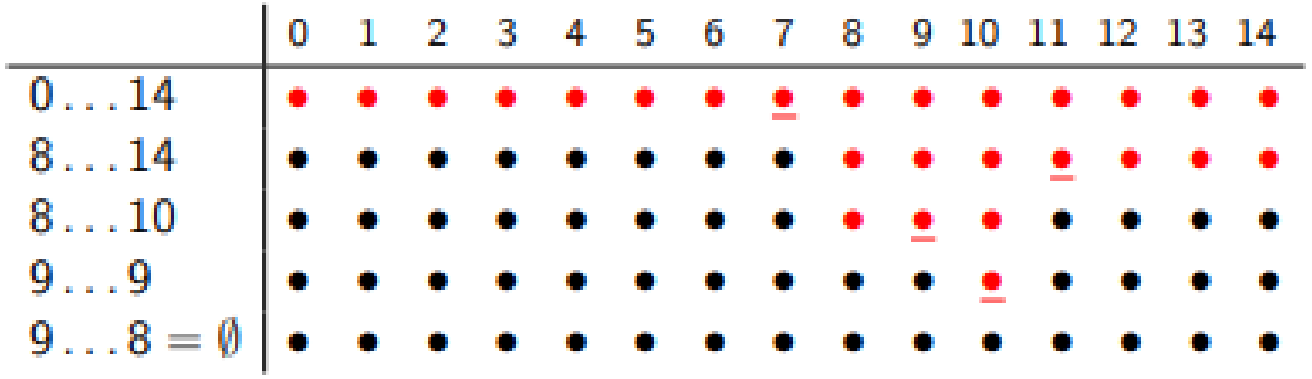
  a[0], a[1],. . . , a[n-1].

  Se x for igual a algum dos a[i], retorna-se True. Senão, retorna-se False.

```c
#include<stdio.h>
bool buscaSequencial (int x, int v[]);

int main ()
{
  int v[10], i, x;
  bool encontra;

    for(i = 0; i < 10; i++) {
        scanf("%d",&v[i]);
    }
    printf("Insira o numero que deseja procurar no vetor.\n");
    scanf("%d",&x);
    encontra = buscaSequencial(x,v);

    printf(encontra ? "true" : "false");

  return 0;
}

bool buscaSequencial (int x, int v[]) {
    int i;
    for(i = 0; i < 10; i++) {
        if(v[i] == x) {
            return true;
            break;
        }
    }
    return false;
}
```

Busca binária em vetores

- Busca o elemento 10 num vetor de 0 a 14:



- O número de divisões ao “meio” é igual ao número de comparações, neste caso, igual a 4 = log(n + 1).

Busca binária em vetores

- **Se a lista a está ordenada** – suponhamos por ordem crescente – há algoritmos muito mais eficientes de procurar x:

- x é procurado entre os índices i1 e i2.
- Inicialmente faz-se i1=0 e i2=n-1.
- Repetidamente, calcula-se o ponto médio m=(i1+i2)/2 (divisão inteira)
  - Se x < V[m] → i2=m-1;
  - Se x > V[m] → i1=m+1
  - Se x == V[m] → retorna true

- Se o intervalo [i1,i2] acaba por ficar vazio (i1 > i2)
  - x não está na lista: → retorna false.

```c
int main (){
  int V[10], i, X;
  bool encontra;

    for(i = 0; i < 10; i++) {
        scanf("%d",&V[i]);
    }
    printf("Insira o numero que deseja procurar no vetor.\n");
    scanf("%d",&X);
    ordenaVetor(V);
    encontra = buscaBinaria(X,V);
        printf(encontra ? "true" : "false");

  return 0;
}

bool buscaBinaria (int X, int V[]) {
    int i1 = 0, i2 = 9, m;
    while (i1 <= i2) {
        m = (i1+i2)/2;
        if (X < V[m]) {
            i2 = m-1;
        } else if (X > V[m]) {
            i1 = m+1;
        } else {
            return true;
        }
    }
    return false;
}
```

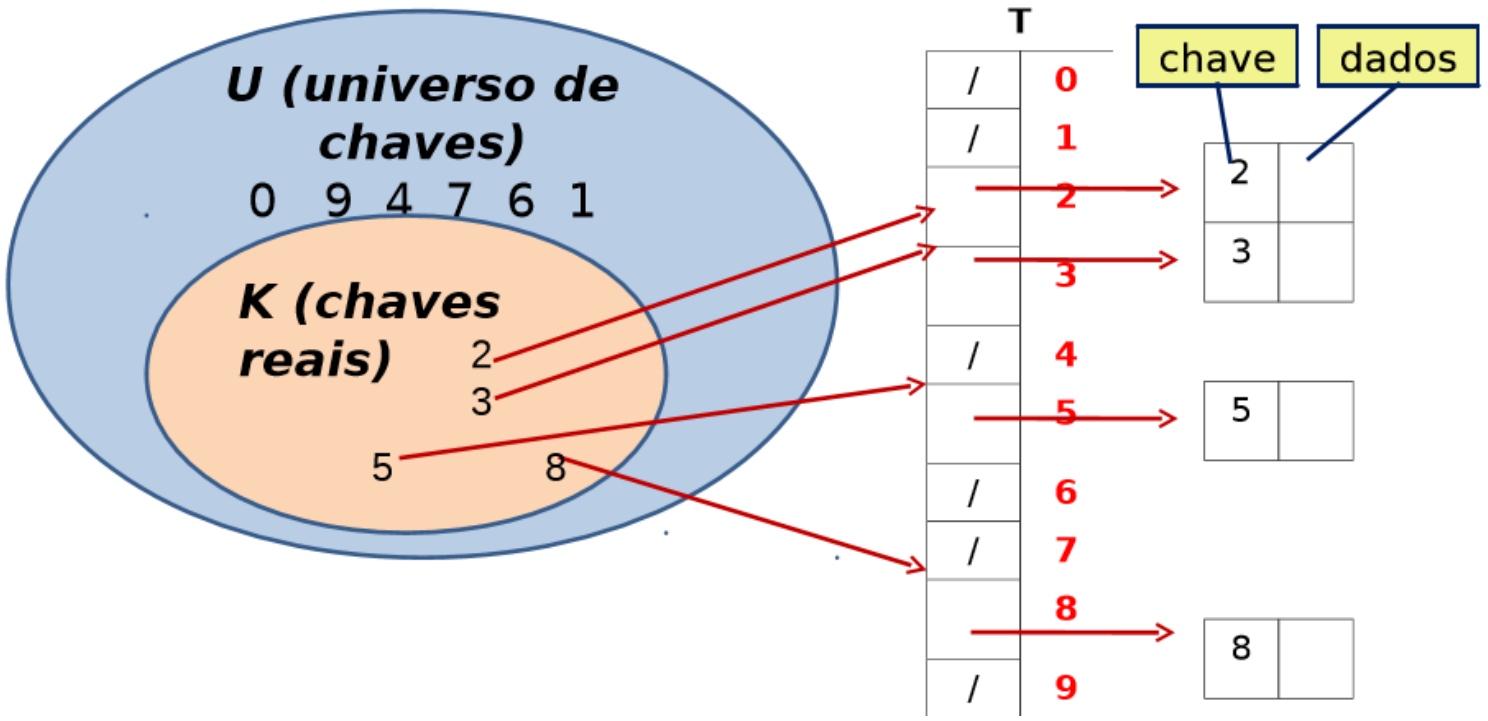
Tabelas Hash

- É uma estrutura de dados eficiente para busca de dados.
  - No pior caso pode demorar O(n);
  - No melhor caso é O(1).

- É uma **generalização de um arranjo comum usando endereçamento direto**.

- O endereçamento direto é aplicável quando temos condições de alocar um arranjo que tem uma única posição para cada chave possível.

Endereçamento direto




```
ED-search(T,k)
 return T[k]
```

```
ED-insert(T,x)
 T[chave[x]] ← x
```

```
ED-delete(T,x)
 T[chave[x]] ← NULL
```

Ex: armazenar alunos

Código	Ingresso	Curso	Nome	
8956495	2014/1	97001	Afonso Celso Penze Nunes da Cunha	Matriculado
9075205	2014/1	55051	Alair Jose de Souza Junior	Matriculado
9364819	2015/1	55051	Allan Ribeiro da Costa	Matriculado
9391831	2015/1	55051	Altair Fernando Pereira Junior	Matriculado
9313197	2015/1	18045	Anderson Hiroshi de Siqueira	Matriculado
8936694	2015/1	55051	Andre Merino Jorge	Matriculado
9292816	2015/1	55051	Andreia de Barros Carpi	Matriculado
8937510	2014/1	55041	Arnaldo Lopes Stanzani	Matriculado
7992894	2014/1	18045	Arthur Demarchi	Matriculado
8624525	2014/1	55051	Artur Artimonte	Matriculado
9292858	2015/1	55051	Bruno Bacelar Abe	Matriculado
9378996	2015/1	55051	Bruno Felipe Barbosa Pereira	Matriculado
9292910	2015/1	55051	Bruno Henrique Rasteiro	Matriculado
7656533	2011/1	55041	Bruno Molina Rosaboni	Matriculado
8957012	2014/1	18045	Cainã de Oliveira Figares	Matriculado

Valor máximo código: 9391831 Valor mínimo: 3144931
Diferença: 6246900

Ex: armazenar alunos

- Se eu tiver apenas 71 alunos na turma.
- Como mapear o código para um intervalo pequeno de endereços?
  - usar os dois últimos algarismos do código , mapeamos no intervalo 0-99.

O paradoxo do aniversário (Feller,1968), diz que em um grupo de 23 ou mais pessoas, juntas ao acaso, existe uma chance maior do que 50% de que 2 pessoas comemorem aniversário no mesmo dia.

71 alunos 23 colisões

5	9075205
10	8937510
16	9292816
19	9364819
25	8624525
31	9391831
94	8936694
95	8956495
97	9313197

8956495
9075205
9364819
9391831
9313197
8936694
9292816
8937510
7992894
8624525

Ex: armazenar alunos

- Como podemos reduzir as colisões?
- Podemos, por exemplo, usar os 3 últimos dígitos e aumentar o tamanho da tabela.

Temos uma única colisão
Mas uma tabela de tamanho 1000

Código	Hash
6908006	6
6767010	10
8957012	12
8124062	62
8936120	120
8083126	126
8084127	127
8955132	132
8957155	155
9313197	197
9075205	205
8626207	207
8626228	228
8957263	263
6792263	263
8532280	280
6608332	332
8602357	357

Endereçamento direto

- Quando o **endereçamento direto funciona bem**?

1. Quantidade de chaves do universo **U é razoavelmente pequeno** e facilmente mapeadas;
2. Não há dois elementos com a mesma chave.

- **Problemas**:

1. se o universo **U é grande**: armazenamento de uma tabela T de tamanho [U] pode ser inviável e até mesmo impossível.
2. conjunto k de chaves realmente usada é muito menor que o conjunto possível de chaves do universo U. Maior parte do espaço alocado para T seria desperdiçada.

- **Solução**:
- Estender conceito de endereçamento direto nas **tabelas hash**.

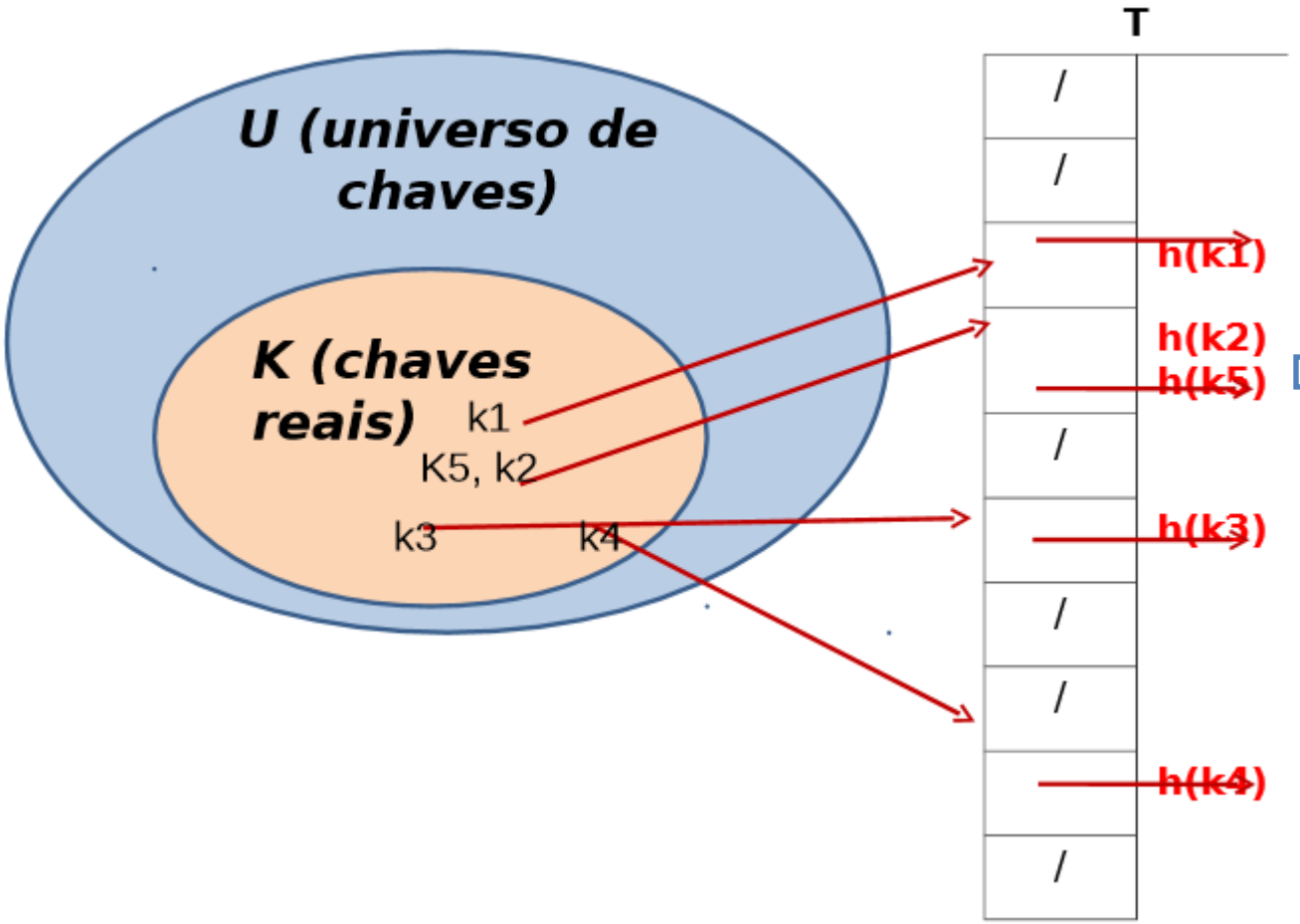
Tabelas Hash

- **Endereçamento direto**: elemento com chave **k** é armazenado na posição **k.**

- **Tabela hash**: elemento com chave **k** é armazenado na posição **h(k)**.
- **h(k) é chamada da função hash** e é usada para calcular a posição da chave k;
- h mapeia o universo U de chaves nas posições de uma tabela hash T[0..m-1].

- **Finalidade da função hash**: reduzir o intervalo de índices de arranjos que precisam ser tratados.

Tabelas Hash



Duas chaves podem ter o hash na mesma posição: colisão!

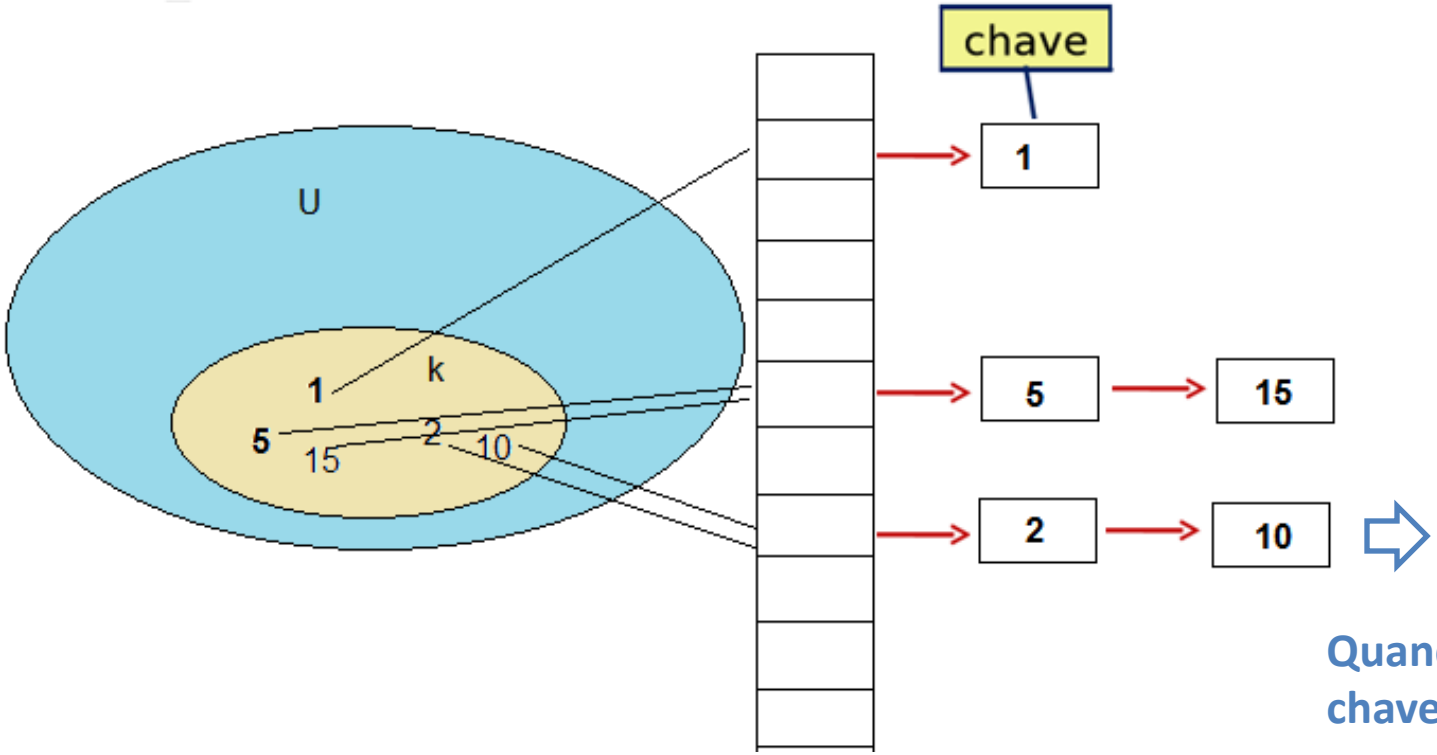
Resolução de colisões por encadeamento

- **Todos os elementos que têm hash para a mesma posição são colocados em uma lista linear ligada.**

- Lista linear:

1. estrutura de dados que implementa operações de inserir, eliminar e buscar;
2. estruturas dinâmicas e flexíveis: podem aumentar e diminuir de tamanho durante execução do programa;
3. estrutura muito útil para alocação dinâmica de memória- quando não é possível prever a quantidade necessária de memória para uma determinada aplica ção.

Resolução de colisões por encadeamento



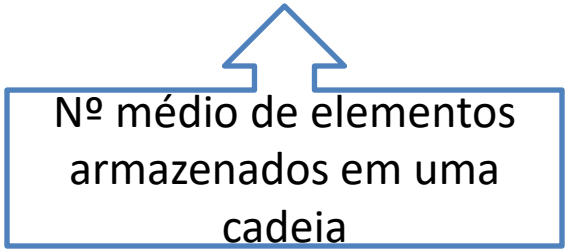
Quando duas chaves tem o hash na mesma posição

Complexidade

- Quanto tempo leva para procurar um elemento com uma determinada chave?

- Depende da função h:
- **Pior caso**: todas as chaves executam o hash na mesma posição: O(n) + tempo para calcular função hash.

- **Caso médio**: depende de como a função h distribui o conjunto de chaves k a serem armazenadas entre as m posições, em média (k/m) = α -> fator de carga.



Complexidade

- Inserção = O(1).
- A Pesquisa sem sucesso demora $O(1 + \alpha) \approx O(1)$.
- A Pesquisa com sucesso demora em média $O(1 + \alpha) \approx O(1)$.
- A Remoção em lista simplesmente ligada tem o mesmo tempo de execução que a Pesquisa. Em lista duplamente ligada pode levar tempo O(1).

Funções Hash

- **Função hash de boa qualidade**: cada chave tem igual probabilidade de efetuar o hash para qualquer das m posições.

- A maior parte das funções hash supõe que o universo de chaves é o conjunto N = { 0, 1, 2 , ...} de números naturais. Se as chaves não são números naturais, deve-se encontrar um modo de interpretá-las como tal.

- Ex: uma cadeia de caracteres (pt) pode ser interpretada como um par de inteiros decimais (112,116) pois p = 112 e t = 116 na tabela ASCII.

Ex: Dicionário

- Construir um dicionário dinâmico que pode conter pares do seguinte tipo: (string nome, string significado), onde a **chave é nome**.

- nome pode ter no máximo 8 letras.

- O dicionário dinâmico, apesar de possuir como chave qualquer string de 8 letras, na prática vai armazenar em média uns 500 elementos.

- Este dicionário deve ter as seguintes funções: insere, elimina e busca.

Ex: Dicionário

- Encontrar uma função hash h(s) que mapeia qualquer String s para um número entre 0 e 500.

- Esta função deve ter complexidade assintótica (1), como o endereçamento direto.

- Tratar as possíveis colisões... porque o número de 500 elementos é uma média. Se houver, mais de 500 inevitavelmente haverá colisões.

Ex: Dicionário

- Função Hash

```
int hash(char nome[]) {
  int soma = 0;
  int i, len = strlen(nome);

  // Soma os valores dos caracteres
 for (i = 0; i < len; i++)
   // nome contem somente letras e nome.length <= 8
   soma += nome[i];

  return (soma % 501);
}
```

soma armazena o somatório dos valores ASCII de cada letra

O método de divisão

- No método de divisão, mapeamos uma chave k para uma de m posições, tomando o resto de k dividido por m:
- **h(k) = k mod m**

- Ex: se a tabela hash tem tamanho m = 12 e a chave é k = 100, então h(k) = 4.

- Deve-se evitar certos valores de m, ex. **potências de 2** -> m = 2^p, **números pares**.

- **Um primo não muito próximo a uma potência de 2 costuma ser uma boa escolha para m.**
- **Ex: n = 2000, média de 3 consultas,**
- **m = primo próximo de 2000/3= 701**

O método de multiplicação

- O método de multiplicação, opera em duas etapas:

1) multiplicamos a chave k por uma constante A no intervalo $0 < A < 1$ e extraímos a parte fracionária de k.

2) multiplicamos esse valor por m e tomamos o piso do resultado.

$$b(k) = \lfloor m(k\,A \bmod 1) \rfloor$$

- Onde (k A mod 1) significa a parte fracionária de kA, i.e., $kA - \lfloor kA \rfloor$.

- Embora possa ser usado qualquer valor para a constante A, ele funciona melhor com alguns valores do que com outros. Knuth sugere que:

$$A \approx (\sqrt{5} - 1)/2 = 0{,}6180339887\ldots$$

O método da multiplicação

- Ex: determine os resultados de h(k) para:
- k = {1100,1101,1102,1103,1104,1105}

$$h(k) = \lfloor m(k\,A \bmod 1) \rfloor$$

- m = 1000 A = 0.5

h(1100) = 0
h(1101) = 500
h(1102) = 0
h(1103) = 500
h(1104) = 0
h(1105) = 500
h(1106) = 0
h(1107) = 500

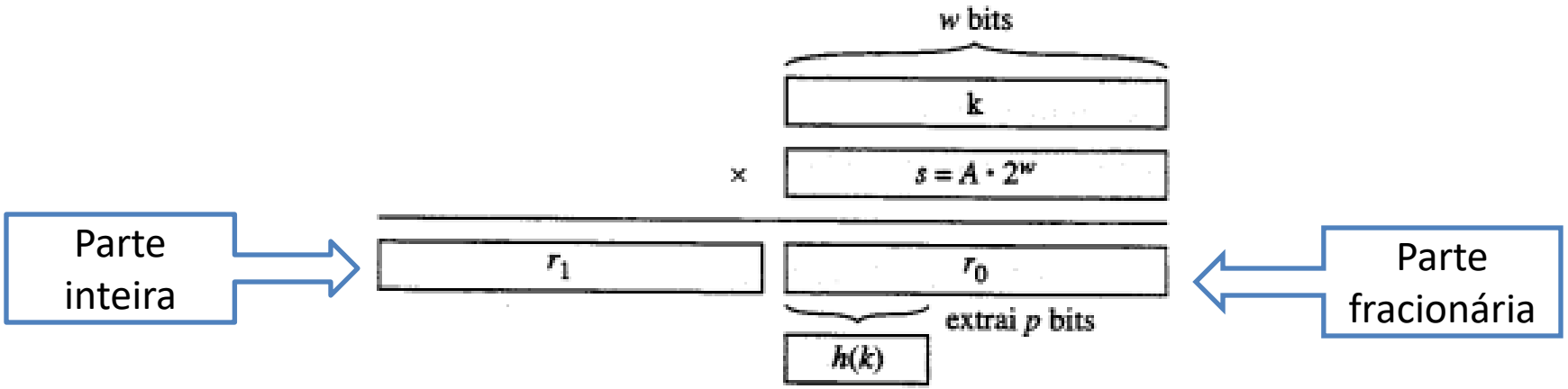
m = 1000 A = 0.6180339887

h(1100) = 837
h(1101) = 455
h(1102) = 73
h(1103) = 691
h(1104) = 309
h(1105) = 927
h(1106) = 545
h(1107) = 163

O método da multiplicação

- **A escolha de m pode ser uma potência de 2 ($m = 2^p$).**
- Suponha que o tamanho da palavra da máquina seja w bits = k (chave).
- $A = s/2^w$, onde s é um inteiro no intervalo $0 < s < 2^w$.
- Multiplicando k por s obtém-se 2wbits $r_1 2^w + r_0$.
- r_1 é a palavra de alta ordem do produto e r_0 é de baixa ordem.
- O valor hash de p bits consiste nos p bits mais significativos de r_0.

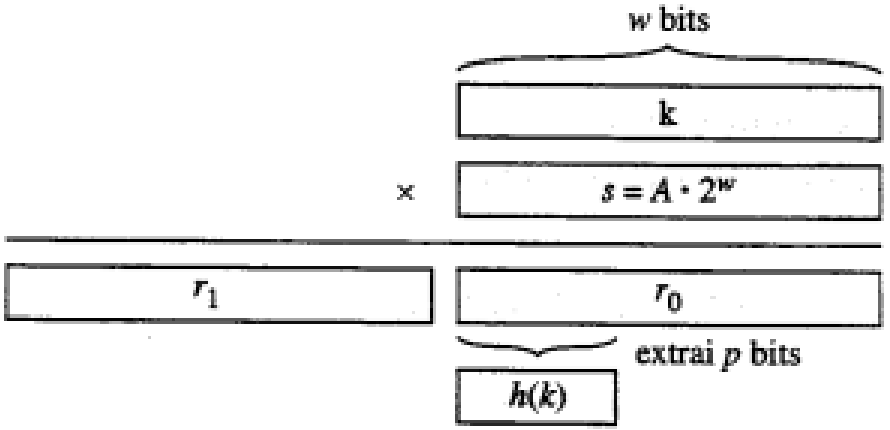
Nº max em w bits



O método da multiplicação

- **Exemplo:** $h(k) = \lfloor m(k\,A \bmod 1) \rfloor$
- k (chave) = 123456, p = 14, **m = 2^{14} = 16384** , w = 32.

- A = s/2^{32} mais próxima a 0,6180..., A = 2654435769/2^{32}
- Multiplicando k por s obtém-se 327706022297664 = (76300 * 2^{32}) + 17612864.
- r_1 = 76300 e r_0 = 17612864.
- Os 14 bits mais significativos de r_0 formam o valor de h(k) = 67.
- 0000 0001 0000 1100 1100 0000 0100 0000



Hash universal

- A única maneira de evitar totalmente o pior caso (número de colisões é n) é escolher a **função hash aleatoriamente**, independente das chaves armazenadas. Essa abordagem é chamada hash univesal.

- **Seleciona-se uma função hash ao acaso a partir de uma classe de funções projetadas no início da execução.**

- **Seja H uma coleção finita de funções hash que mapeiam um universo U de chaves no intervalo {0,1 ..., m-1}. Essa coleção é dita universal se para cada par de chaves distintas k e l o número de funções hash H para as quais h(k) = h(l) é no máximo H.**

- Em virtude da aleatoriedade, o algoritmo poderá se comportar de modo diferente em cada execução, ainda que para a mesma entrada, garantindo um bom desempenho.

Endereçamento aberto

- No endereçamento aberto, todos os elementos estão armazenados na própria tabela. Não existe uma lista ou elemento fora da tabela como no encadeamento.

- **Vantagens**:
- Evita o uso de ponteiros, sendo necessário calcular a sequência de posições a ser examinadas.

- Busca gerar menos colisões e recuperação mais rápida.

- **Desvantagens**:
- A tabela hash pode ficar cheia, de tal forma que não podem ser feitas inserções adicionais.

Endereçamento aberto

- Para inserir um elemento, sondamos a tabela hash até encontrarmos uma posição vazia na qual seja possível inserir chave.

- **Sondagem linear**

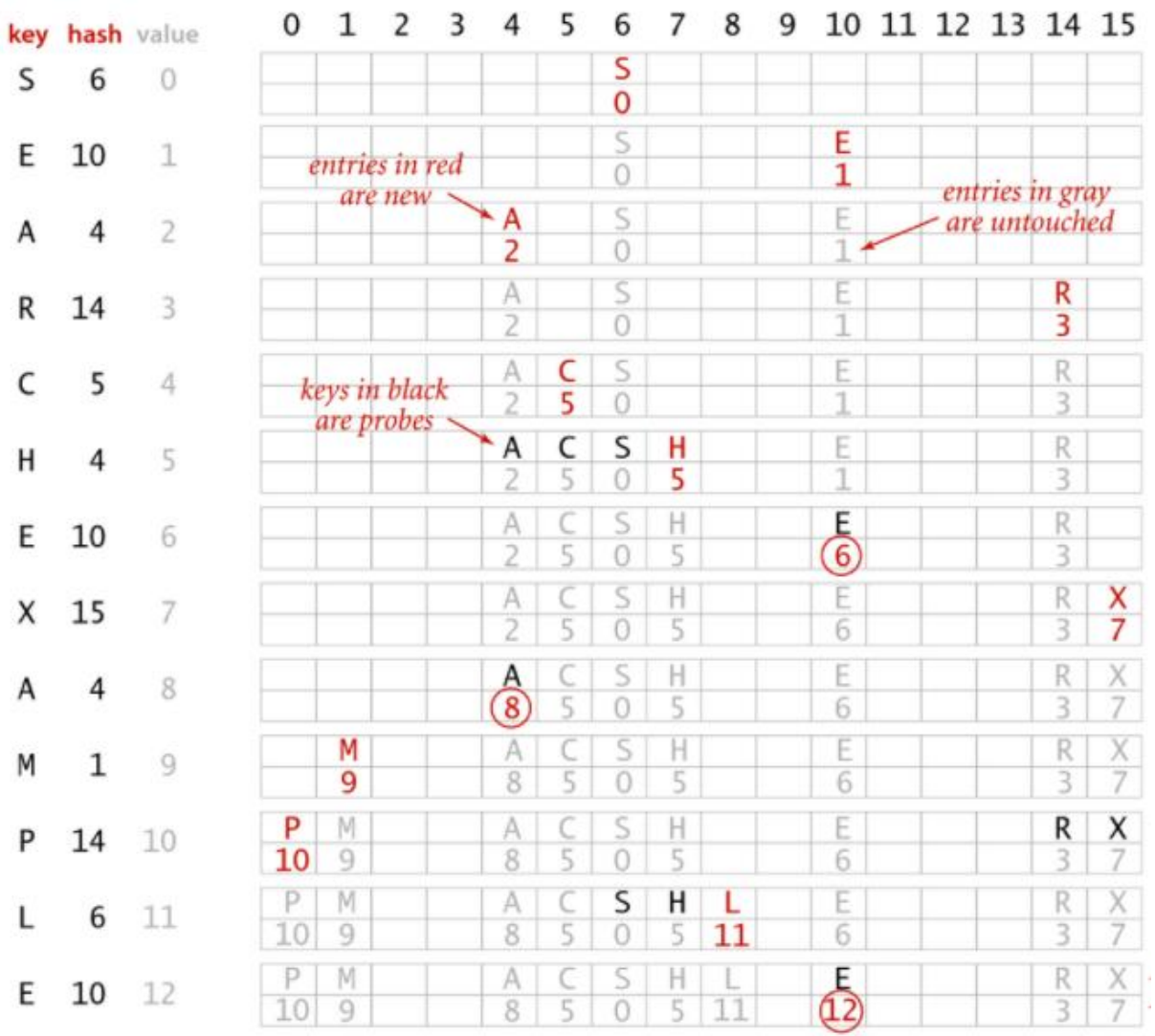
- Usa a função hash:

- **h(k,i) = (h´(k) + i) mod m**

Valor de i	Posição sondada
0	T[$h'(k)$]
1	T[$h'(k)$+1]
...	...
...	T[m-1]
...	T[0]
...	T[1]
...	...
m-1	T[$h'(k)$-1]

- Fácil de implementar, sofre de agrupamento primário. Gera sequencias longas de posições ocupadas, aumentando o tempo médio de pesquisa.

Exemplo sondagem linear



Algoritmos inserção e busca

```
insereHash(T,k)
i = 0
j = h(k,i)
while (i != m) {
    if (T[j] == NULL)
            T[j] = k
            return j
     else
            i = i + 1
     j = h(k,i)
}
return -1 // Estouro da tabela hash
```

```
buscaHash(T,k)
i = 0
j = h(k,i)
while (i != m) and (T[j] != NULL){
      if (T[j] == k)
          return j
       i = i + 1
       j = h(k,i)
}
 return NUL // Não foi
encontrada a chave k
```

Eliminação

- A eliminação é feita colocando uma marca "eliminado"e não o valor NULL.
- **O problema é que se colocarmos o valor NULL na posição eliminada o algoritmo de busca não irá encontrar as demais chaves incluídas depois da chave eliminada.**
- Colocando a marca sabemos que a posição está livre, mas há outras chaves que vêm depois dela e devem ser verificadas durante uma busca.

- O algoritmo de inserção pode ser modificado para incluir quando encontrar uma posição marcada como "eliminado".
- Problema: tempo de pesquisa não depende mais somente do número elementos presentes na tabela, mas também do número de elementos eliminados.

Algoritmo de remoção

```
eliminaHash(T,k)
i = 0
j = h(k,i)
while (i != m) and (T[j] != NULL)
    if (T[j] == k)
        // Marca a posição j como eliminada
         T[j] = “eliminado”
         return j
    i = i + 1
    j = h(k,i)
 // A chave k não está presente na tabela hash
return NULL
```

Endereçamento aberto

- **Sondagem quadrática**
- Utiliza uma função hash da forma:

- **$h(k, i) = (h'(k) + c1i + c2i^2) \bmod m$**

onde c1 e c2 são constantes auxiliares.

- A posição inicial sondada é T[h'(k)]
- Posições posteriores são deslocadas por quantidades que dependem de forma quadrática do número da sondagem i.

Exemplo

- $h(k, i) = (h'(k) + 0.5i + 0.5i^2) \bmod 11$
- Inserção das chaves 100 e 12:

- k = 100 -> (100 mod 11) = 1
- k = 12 -> (12 mod 11) = 1 (colisão)

$((12 \bmod 11) + 0.5*1 + 0.5*1^2) \bmod 11 = 2$

Endereçamento aberto

- **Hash duplo**
- É um dos melhores métodos disponíveis para endereçamento aberto, porque as permutações produzidas têm muitas características de permutações escolhidas aleatoriamente.

- O hash duplo usa uma função hash da forma:

- **h(k, i) = (h1(k) + ih2(k)) mod m**
- onde h1 e h2 são funções hash auxiliares.

Valor de i	Posição sondada
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	$T[(h_1(k) + (m-1)h_2(k)) \bmod m]$

Exemplo

- h1(k) = k mod m,
- h2(k) = 1 + (k mod m-1),

- Exemplo:
- Para k = 123456, m = 701, tem-se
  h1(123456) = 80 e h2(123456)= 257.

- Portanto, a primeira posição sondada é de número 80; as demais estão separadas por 257 posições.
- Ou seja: 80, 337, 594, 150, ...

Resumo

- Endereçamento aberto é uma alternativa ao tratamento de colisões utilizando encadeamento.

- **Vantagens**:
- Não utiliza apontadores/listas;
- Uso mais eficiente do espaço alocado para a tabela hash.

- **Desvantagens**:
- Eliminação mais complicada;
- A tabela hash pode ficar cheia, com todos os seus elementos ocupados.

Exercícios

1. Mostre a inserção das chaves 5, 28, 19, 15, 20, 33, 12, 17, 10 em uma tabela hash com colisões resolvidas por encadeamento. Seja a tabela com 9 posições, e seja a função hash **h(k) = k mod 9.**

2. Considere uma tabela hash de tamanho m = 1000 e a função hash correspondente h(k) igual a

$$h(k) = \lfloor m(k\,A \bmod 1) \rfloor$$

para A = $(\sqrt{5}-1)/2$. . Calcule as localizações para as quais as chaves 61, 62, 63, 64 e 35 estão mapeadas.

Exercícios

3. Considere a inserção das chaves 10, 22, 31, 4, 15, 28, 17, 88, 59 em uma tabela hash de comprimento m = 11 usando o endereçamento aberto com a função hash primário h(k) = k mod m.

   Ilustre o resultado da inserção dessas chaves com:

a) o uso da sondagem linear

b) o uso da sondagem quadrática com c1 = 1 e c2 = 3

c) o uso do hash duplo com h2(k) = 1 + (k mod (m − 1))

Exercícios

4. Pesquise outros tipos de hash (Ex. hash perfeito, etc) ou uma aplicação com hash (Ex. Blockchain, criptografia, etc) para apresentar na próxima aula.

5. Faça um resumo da aula sobre hash.

Se você não quer receber críticas, não tente nada novo.

Jeff Bezos