Técnicas e Análise de Algoritmos

Hash e Colisões

Professor: **Jeremias Moreira Gomes**
E-mail: jeremias.gomes@idp.edu.br

Introdução

Motivação

- Estruturas lineares permitem armazenar elementos sem que o valor de N seja conhecido a priori, em tempo de compilação
- Estas estruturas, porém, não são eficientes na busca do elementos
  - Complexidade linear (O(N))

Motivação

- As árvores de busca, como as estruturas lineares, também permitem o armazenamento de um número arbitrário de elementos
  - Limitado somente pela memória disponível
- Em árvores de busca perfeitamente balanceadas, a ordem de complexidade da busca é O(log N)
- Porém a organização de memória das árvores não é contígua, levando à perda de eficiência em relação à posicionamentos de memória (cache*)

*Brincadeira para os vocês do 4º semestre

Motivação

- A idéia da **hash** é deduzir o índice de um elemento em um vetor a partir apenas da informação armazenada pelo elemento, reduzindo a ordem de complexidade busca para O(1)
    - **Utilizar o próprio valor ou conteúdo do elemento, para determinar a sua posição no vetor**

Hashes

Hash - Definição

Uma função h é uma função de hash se ela transforma uma chave K no índice do elemento que contém K na tabela

Hashes - Exemplo de Função de Hash

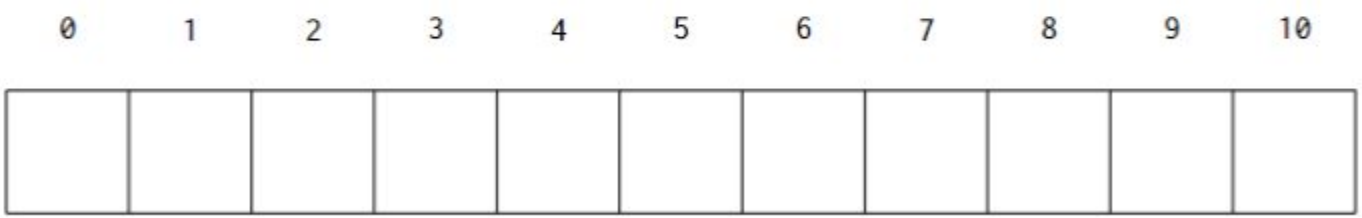
- Nome: Resto da Divisão
- Chave: Um inteiro K
- Algoritmo: Obtêm-se o resto da divisão da chave K pelo tamanho T da tabela

```
unsigned long hash(unsigned long k, unsigned long t) {
    return k % t;
}
```

Hashes - Exemplo de Função de Hash

Algoritmo: Resto da divisão - $T = 11$

Elemento a ser inserido: 51

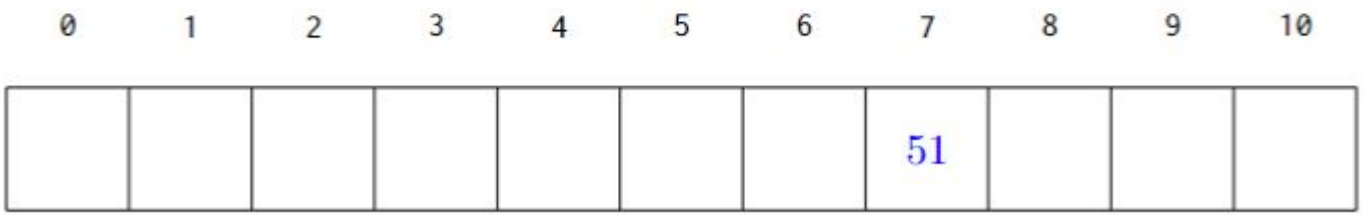


Hashes - Exemplo de Função de Hash

Algoritmo: Resto da divisão - $T = 11$

Elemento a ser inserido: 51

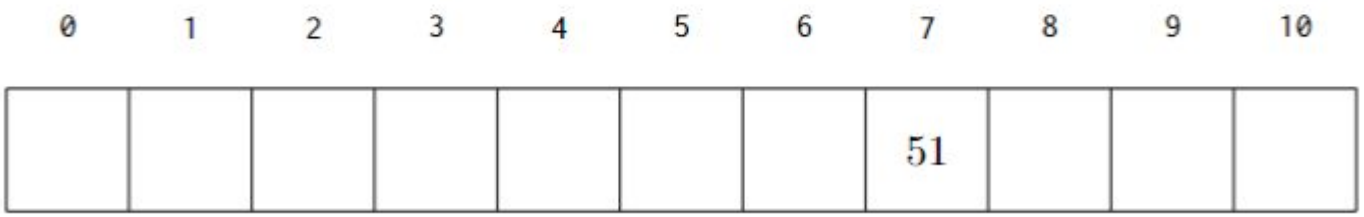
$$h(51) = 51 \pmod{11} = 7$$



Hashes - Exemplo de Função de Hash

Algoritmo: Resto da divisão - $T = 11$

Elemento a ser inserido: 16

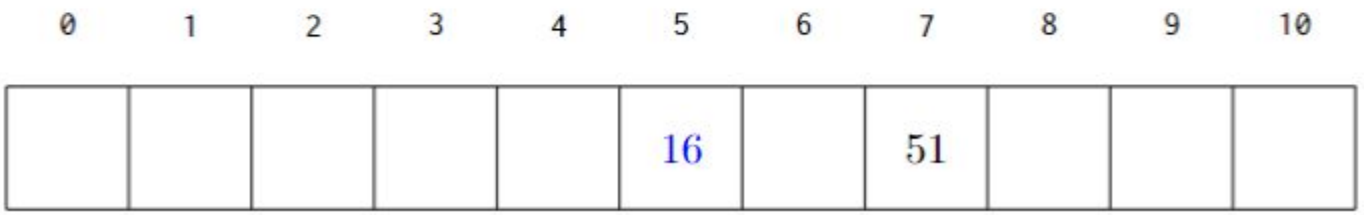


Hashes - Exemplo de Função de Hash

Algoritmo: Resto da divisão - $T = 11$

Elemento a ser inserido: 16

$$h(16) = 16 \pmod{11} = 5$$



Hashes - Colisão

- Qual o problema com a função de hash anterior (resto da divisão) ???
  - O que acontece se duas inserções resultarem no mesmo resto?
    - Inserção na mesma posição?
- **Quando duas chaves distintas geram o mesmo índice, é dito que ocorreu uma colisão**
- Formalmente, uma colisão ocorre se duas chaves distintas K1 e K2 gerarem o mesmo índice, isto é, se h(K1) = h(K2) com K1 ≠ K2

Hashes - Colisão

- Se uma função de hash não for bem escolhida, podem ocorrer muitas colisões ou até espaços de memória acabarem subutilizados
- A idéia é **encontrar uma função h que gere o mínimo de colisões**, mas que não seja sofisticada ao ponto de seu cálculo interferir na performance do programa

Hashes - Outros Exemplos de Função de Hash

- Chave: Uma string K
- Algoritmo: Código numérico que é a soma do valor dos caracteres
- Nível de colisão: alto

```
unsigned int h(string k) {
    unsigned int total = 0;
    for (auto c : k) {
        total += c;
    }
    return total;
}
```

Hashes - Outros Exemplos de Função de Hash

- Chave: Um inteiro K
- Algoritmo: Eleva-se K ao quadrado e pega-se o centro do resultado
- Nível de colisão: médio

```
unsigned int h(int k) {
    auto res = k * k;
    res = (res & 0x00FFF00) >> 8;
    return res;
}
```

Hashes - Outros Exemplos de Função de Hash

- Chave: Uma string K com N caracteres
- Algoritmo: g(x) é um polinômio de grau N − 1 com coeficientes a_i = K[i] e h(K) = g(p) (mod T), onde T é o tamanho da tabela e p ≠ T é primo
- Nível de colisão: baixo

```
unsigned long h(const string& K, size_t p, size_t T) {
    unsigned long res = 0;
    for (int i = K.size() - 1; i >= 0; i--) {
        res = (res * p) % T;
        res = (res + K[i]) % T;
    }
    return res;
}
```

Hashes

- Funções de hash tendem a ficar cada vez mais complexas, à medida que se busca um baixo nível de colisões
- Porém é difícil evitar que colisões ocorram
- Além disso, é de interesse da estrutura de dados manter o conteúdo mesmo com a colisão, então como proceder?
- Existem, principalmente, duas maneiras de se lidar com colisões
    - **Endereçamento aberto** ou **Encadeamento**

Endereçamento Aberto

Hashes - Endereçamento Aberto

- Endereçamento Aberto (Open Addressing ou Closed Hashing) é um método de resolução de colisão que utiliza sondagem (*probing*) ou pesquisa em locais alternativos no vetor até que uma posição livre seja encontrada

Hashes - Endereçamento Aberto - Definição

- Se a chave K for mapeada para uma posição já ocupada da tabela, o endereçamento aberto utiliza a sequência de sondagem
  - N (h(K) + p(1)), N (h(K) + p(2)), . . . , N (h(K) + p(i)), . . .

onde p é a função de sondagem, i é o índice de sondagem e N a função de normalização, até que:

  - Tenta encontrar uma posição disponível por
    - N (h(K) + p(j)) = N (h(K))
  - Ou verifica se a tabela está cheia

Hashes - Endereçamento Aberto

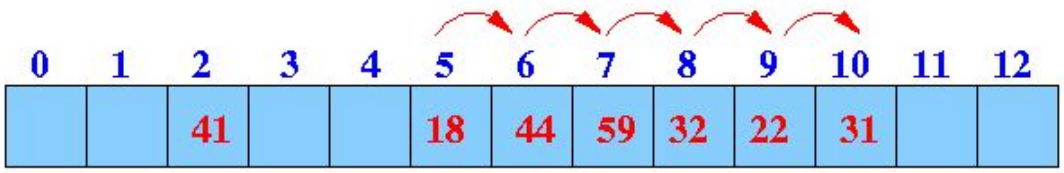
- A função de normalização faz com que o índice resultante esteja dentro dos limites da tabela, usando o resto da divisão:
  - N (K) = K(mod T), onde T é o tamanho da tabela
- Se uma posição N (h(K) + p(i)) já estiver ocupada, tenta-se o próximo índice de sondagem (i + 1) até que se encontre um espaço vago ou ocorra uma das outras condições

Hashes - Endereçamento Aberto - Tipos de Sondagem

- A sondagem é comumente feita de três formas diferentes:
  - Sondagem Linear
  - Sondagem Quadráditca
  - Sondagem por Hash Duplo (double hashing)

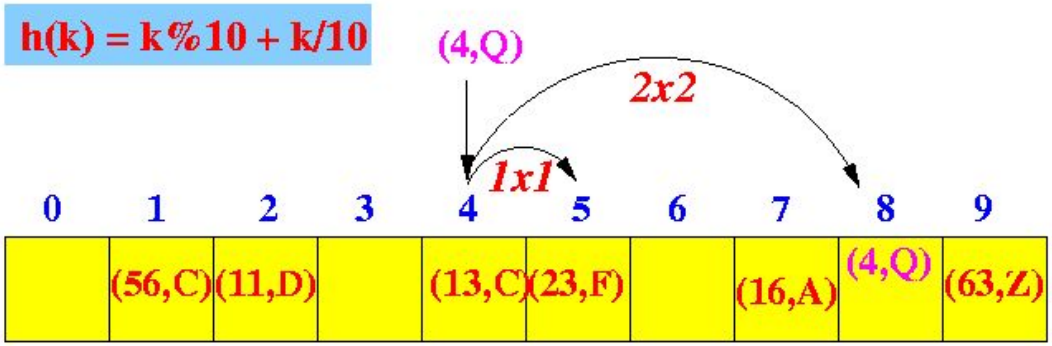
Hashes - Endereçamento Aberto - Tipos de Sondagem

- **Sondagem Linear**
  - Na sondagem linear, ao ocorrer uma colisão, verifica-se linearmente no vetor, pela próxima posição disponível



Hashes - Endereçamento Aberto - Tipos de Sondagem

- **Sondagem Quadrática**
  - Para evitar agrupamentos de valores, pode-se utilizar a sondagem quadrática, que realiza saltos quadráticos na busca pela próxima posição disponível, após ocorrer uma colisão



Hashes - Endereçamento Aberto - Tipos de Sondagem

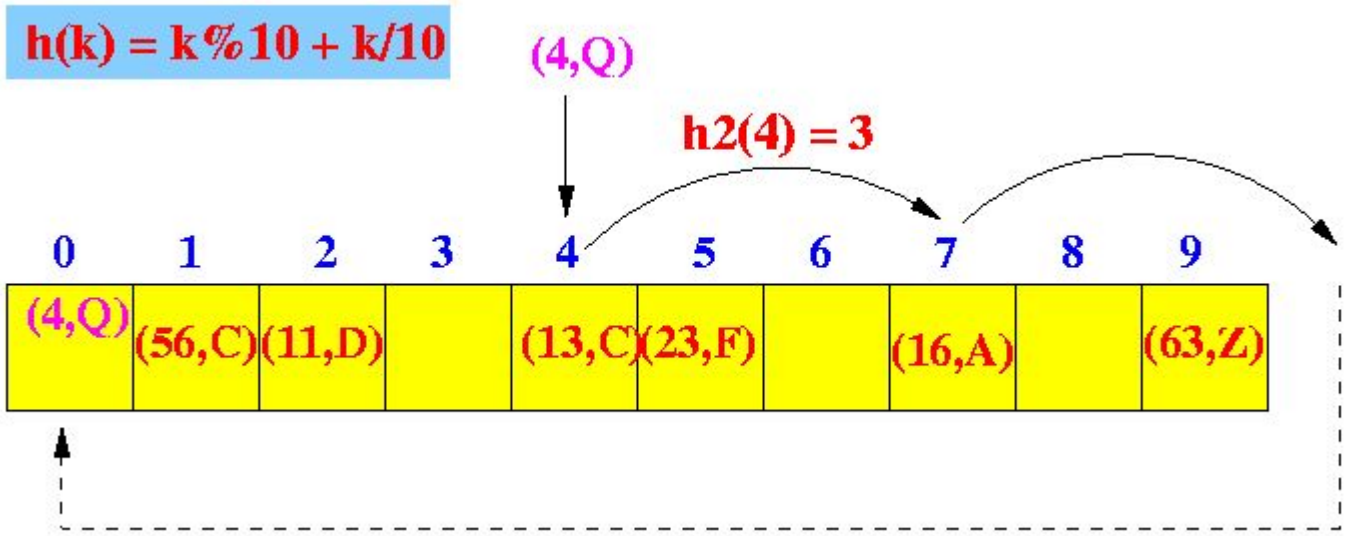
- **Hash Duplo**
  - O hash duplo é uma das melhores estratégias de endereçamento
  - Isto porque a sequência de sondagem gerada tem muitas das características das sequências aleatórias
  - No hash duplo a sequência de sondagem tem a forma

    $h(K) = (h1(K) + ih2(K)) \pmod{T}, \quad i = 0, 1, 2, \ldots, T - 1$

  - onde h1 (K), h2 (K) são duas funções de hash auxiliares, i é o índice de sondagem e T é o tamanho da tabela

Hashes - Endereçamento Aberto - Tipos de Sondagem

- Hash Duplo



Encadeamento (Chaining)

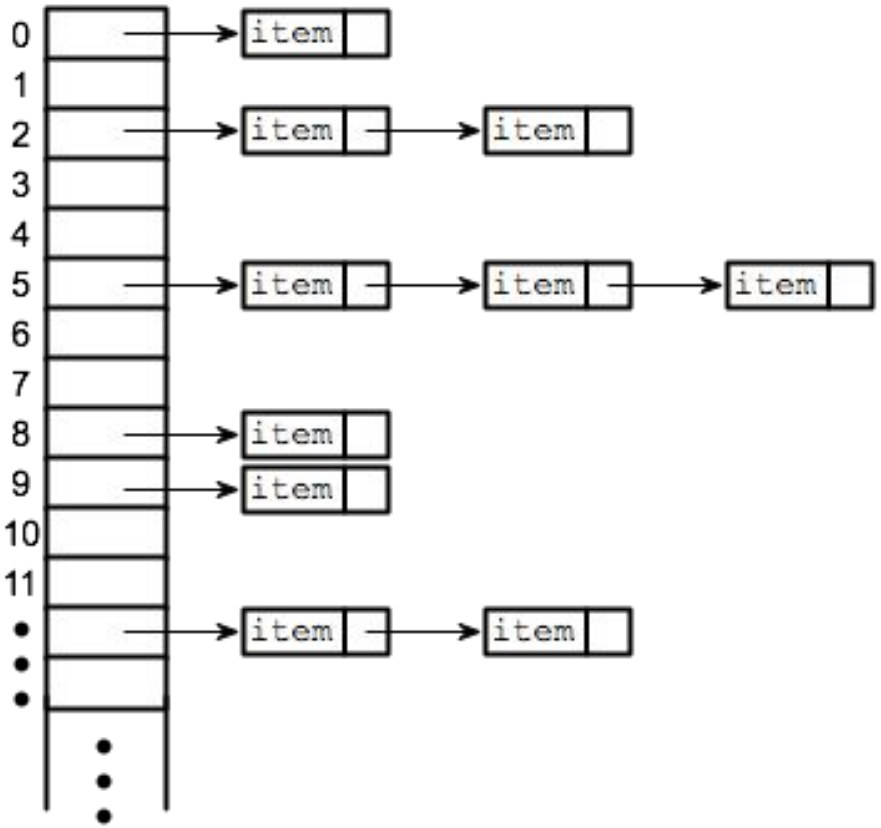
Hashes - Encadeamento

- Encadeamento (Chaining ou Open Hashing) é um método de resolução de colisão que utiliza listas duplamente encadeadas para cada posição do vetor
- Cada chave K tal que h(K) = j, onde j = 0, 1, . . . , T − 1, é adicionada à lista que ocupa a posição j
- O uso de listas duplamente encadeadas permite uma remoção mais eficiente

Hashes - Encadeamento

- Se a opção de remoção não for implementada, uma lista simplesmente encadeada ou um vector também podem ser utilizados
- Deve-se tomar cuidado, porém, porque no pior caso todas as chaves colidem em uma mesma posição, de modo que a busca e a inserção passam a ter complexidade O(N), porém ocupando mais espaço em memória que uma única lista

Hashes - Encadeamento



Hashes em C++

Hashes em C++

- A STL da linguagem C++ oferece dois contêiners que utilizam hashes para obter complexidade médio O(1) para as operações de inserção, remoção e busca:
  - unordered_set
  - unordered_map

Hashes em C++

- unordered_set é um contêiner que abstrai a ideia de conjuntos, armazenando elementos únicos
  - A versão que admite repetições é a unordered_multiset
- unordered_map e unordered_multimap são as abstrações equivalentes para dicionários
- Ambas armazenam pares de chaves e valores
- Estas estruturas têm interface semelhantes às que utilizam BSTs

unordered_set

- unordered_set mantém um conjunto de elementos únicos
  - O tipo destes elementos é indicado na declaração do conjunto
- Também podem ser indicados na declaração a função de hash a ser utilizada, a função de comparação de igualdade entre os elementos e o alocador de memória
- Este contêiner foi introduzido no C++11

unordered_set

- Ele compartilha com o set a mesma interface para inserção, remoção e consulta
- O número de elementos armazenados é dado pelo método size()
- Novos elementos podem ser inseridos por meio dos métodos insert() e emplace()
- Os elementos podem ser removidos através do método erase()

unordered_set

- O método count() retorna o número de ocorrências de um dado elemento (zero ou um)
- O método find() localiza um elemento e retorna o iterador para sua posição
  - Ou end(), caso o elemento não se encontre no conjunto
- Todos estes métodos tem complexidade **média** O(1)
  - A base de implementação é o encadeamento

unordered_set

```
unordered_set<int> s {27, 21, 7, 12, 7, 7, 33};
cout << "Quantidade de elementos: " << s.size() << endl;

auto it = s.insert(7);   // par <iterator, bool> onde bool indica se inseriu ou nao
if (it.second) cout << "Insercao do valor " << *it.first << " bem sucedida" << endl;

it = s.insert(8);
if (it.second) cout << "Insercao do valor " << *it.first << " bem sucedida" << endl;

s.erase(12);            // {7, 8, 21, 27, 33}
for (auto x: s) {
    cout << x << " ";
}
cout << endl;
```

unordered_set

- Existem algumas funções relativas, que permitem obter informações sobre posicionamento de elementos e controle de colisões de acordo com a função de hash
  - bucket_count() - número de posições do “vetor”
  - bucket() - permite verificar em qual célula um elemento se encontra

unordered_set

```
struct custom_hash {
    size_t operator()(int k) const {
        return k % 10;
    }
};

int main() {
    unordered_set<int, custom_hash> s;

    for (int i = 0; i < 25; i++) s.insert(i);

    cout << "Numero de slots: " << s.bucket_count() << endl;
    for (auto x: s) {
        cout << x << " encontra-se na celula " << s.bucket(x) << endl;
    }
    cout << endl;
}
```

unordered_map

- unordered_map mantém um conjunto de pares, compostos por uma chave e um valor
- Os tipos Key, T das chaves e dos valores são indicados na declaração do dicionário
- Também podem ser indicados na declaração a função de hash a ser utilizada, a função de comparação de igualdade entre os elementos e o alocador de memória

unordered_map

```
unordered_map<string, double> m {
    {"Maria", 1.68}, {"Lucas", 1.59}, {"Pedro", 1.74},
    {"Joana", 1.72}, {"Alberto", 1.78}
};

cout << "Quantidade de elementos: " << m.size() << endl;
m.insert({"Carlos", 1.78});

cout << "Numero de slots: " << m.bucket_count() << endl;

for (auto x: m)
    cout << x.first << " tem " << x.second << "m de altura" << endl;

for (int i = 0; i < m.bucket_count(); i++)
    cout << "Celula " << i << " tem " << m.bucket_size(i) << " elementos" << endl;
```

Conclusão