Universidade Federal da Bahia
Instituto de Computação

Caio Costa Sa da Nova e Matheus Costa Pinto Marçal

Estrutura de dados e Algoritmos II
Trabalho 1

Salvador 2023

Apresentação:

Neste trabalho foi feita a análise do método de Hash Linear. Foram considerados os seguintes tamanhos de páginas (PAGE_SIZE) e fatores de carga máximo (MAX_LOAD_FACTOR).

$$\text{PAGE\_SIZE} \in \{1, 5, 10, 20, 50\}$$
$$\text{MAX\_LOAD\_FACTOR} \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$$

O experimento é repetido para cada combinação {1, 5, 10, 20, 50}X {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}. Além disso, para cada combinação de PAGE_SIZE e MAX_LOAD_FACTOR se repete o experimento 10 vezes.
O método de Hash Linear implementado faz uso de uma estrutura de dados auxiliar chamada de Page e uma estrutura de dados principal chamada Hash Table. A Hash Table apresenta uma lista de páginas e cada página apresenta uma lista de chaves e um ponteiro para a próxima página encadeada

Rodando o projeto:

Para rodar o projeto de Hash Linear, primeiramente baixe NodeJS em: https://nodejs.org/en. Após instalado, basta ir com um terminal na pasta do projeto e rodar **npm start**

Implementação:

Estrutura de dados HashTable

```
export default class HashTable {
  level;
  N;
  pages;
  pageSize;
  maxLoadFactor;
```

A estrutura possui 5 atributos principais:

1. level => Indica o nível que a tabela hash se encontra, o nível é utilizado para fazer o cálculo do hash.

2. N => Um ponteiro que aponta para a próxima página a ser redistribuída ao se exceder o fator de carga máximo.
3. pages => Uma lista de páginas que irão guardar as chaves.
4. pageSize => O tamanho máximo de cada página, ao se exceder esse valor uma página é encadeada a anterior.
5. maxLoadFactor => Fator de carga máximo, ao se exceder esse valor uma nova página é criada e se atualiza o valor de N.

Inicialização

```
constructor(pageSize, maxLoadFactor) {
  this.level = 0;
  this.N = 0;
  this.pages = [new Page(pageSize), new Page(pageSize)];
  this.pageSize = pageSize;
  this.maxLoadFactor = maxLoadFactor;
}
```

Ao se inicializar a estrutura de HashTable, aceita-se dois parâmetros de configuração, o pageSize (tamanho máximo das páginas) e o maxLoadFactor (fator de carga máximo). A estrutura é inicializada com nível 0, N apontando para a página 0 e duas páginas iniciais.

Hash

```
hash(key, l) {
  return key % (2 * Math.pow(2, l));
}
```

O hash de uma chave é calculado em função de dois parâmetros, a própria chave e o nível da tabela hash.

Cálculo do fator de carga

```
calculateLoadFactor() {
  let numberOfElements = 0;
  let numberOfPages = 0;
  for (let page of this.pages) {
    numberOfElements += page.getNumberOfElements();
    numberOfPages += page.getNumberOfPages();
  }

  return numberOfElements / (numberOfPages * this.pageSize);
}
```

Para cada inserção o fator de carga atual é calculado para identificar se e hora de fazer uma redistribuição. O fator de carga é calculado obtendo-se o número de elementos presentes na tabela hash e a capacidade de armazenamento da tabela hash atual. O cálculo é feito fazendo uma divisão entre estes dois valores.

Inserção

```
insert(key) {
  let hash = this.hash(key, this.level);
  if (hash < this.N) {
    hash = this.hash(key, this.level + 1);
  }
  this.pages[hash].insert(key);
  if (this.calculateLoadFactor() >= this.maxLoadFactor) {
    this.split();
  }
}
```

Para se inserir na tabela hash primeiro se calcula o valor hash da chave. Em seguida se compara o valor hash com o valor atual de N, se o primeiro for menor, isso indica que precisamos utilizar um nível acima do atual, pois a página apontada pelo primeiro hash já pode ter sofrido uma redistribuição anteriormente. Após se inserir a chave, avalia o fator de carga atual comparando-o com o fator de carga máximo, caso o primeiro seja maior ou igual ao fator de carga máximo, a tabela hash realiza um split, redistribuindo a página apontada por N.

Redistribuição

```
split() {
  this.pages.push(new Page(this.pageSize));
  const pageToSplit = this.pages[this.N];
  this.pages[this.N] = new Page(this.pageSize);
  let traverser = pageToSplit;
  while (traverser) {
    for (let key of traverser.keys) {
      const hash = this.hash(key, this.level + 1);
      this.pages[hash].insert(key);
    }
    traverser = traverser.nextPage;
  }
  this.pages[this.N].clearEmptyLinkedPages();

  this.N = (this.N + 1) % (2 * Math.pow(2, this.level));
  if (this.N === 0) {
    this.level = this.level + 1;
  }
}
```

Ao se fazer uma redistribuição, primeiramente se cria uma nova página e a adiciona a tabela Hash. Em seguida, se identifica a página apontada por N para ser redistribuída. Após a identificação, cada chave da página é visitada e seu hash calculado novamente, porém dessa vez um nível acima é utilizado. Após a redistribuição, atualiza-se o valor de N para a próxima página a ser redistribuída. Caso N retorne a 0, o nível é atualizado.

Busca

```
search(key) {
  let hash = this.hash(key, this.level);
  if (hash < this.N) {
    hash = this.hash(key, this.level + 1);
  }
  let traverser = this.pages[hash];
  while (traverser) {
    for (let travKey of traverser.keys) {
      if (key === travKey) {
        return travKey;
      }
    }
    traverser = traverser.nextPage;
  }
}
```

Para se buscar uma chave, primeiro é calculado o hash da chave com o nível atual, caso esse hash aponte para uma página anterior a página apontada por N, calcula-se o hash novamente utilizando um nível acima, pois a página apontada já pode ter sido redistribuída. Em seguida uma navegação é feita na lista encadeada de páginas comparando cada chave com a chave que se procura, caso a chave seja encontrada ela é retornada.

Estrutura de dados Page

```
export default class Page {
  keys;
  size;
  nextPage;
```

A estrutura possui 3 atributos principais:

1. keys => Chaves primárias armazenadas
2. size => Tamanho da página (quantas chaves a página pode armazenar)
3. nextPage => Ponteiro que aponta para a proxima pagina na lista encadeada

Inserção

```
insert(key) {
  if (this.keys.length < this.size) {
    this.keys.push(key);
  } else {
    if (this.nextPage == null) {
      this.nextPage = new Page(this.size);
    }
    let traverser = this.nextPage;
    while (traverser.nextPage) {
      traverser = traverser.nextPage;
    }
    if (traverser.keys.length >= this.size) {
      traverser.nextPage = new Page(this.size);
      traverser = traverser.nextPage;
    }
    traverser.keys.push(key);
  }
}
```

Para inserir uma chave em uma página, primeiramente verifica-se se a pagina ja nao esta com carga máxima. Caso não esteja, simplesmente é feita a inserção da chave. Caso contrário, uma nova página é adicionada à lista encadeada caso nextPage não exista, em seguida é feita uma navegação até a última página na lista encadeada. Ao chegar na última página, verifica-se se esta não está com carga máxima, uma nova página é adicionada caso sim e atualiza-se o navegador. Finalmente, a chave é inserida.

Análise Experimental:

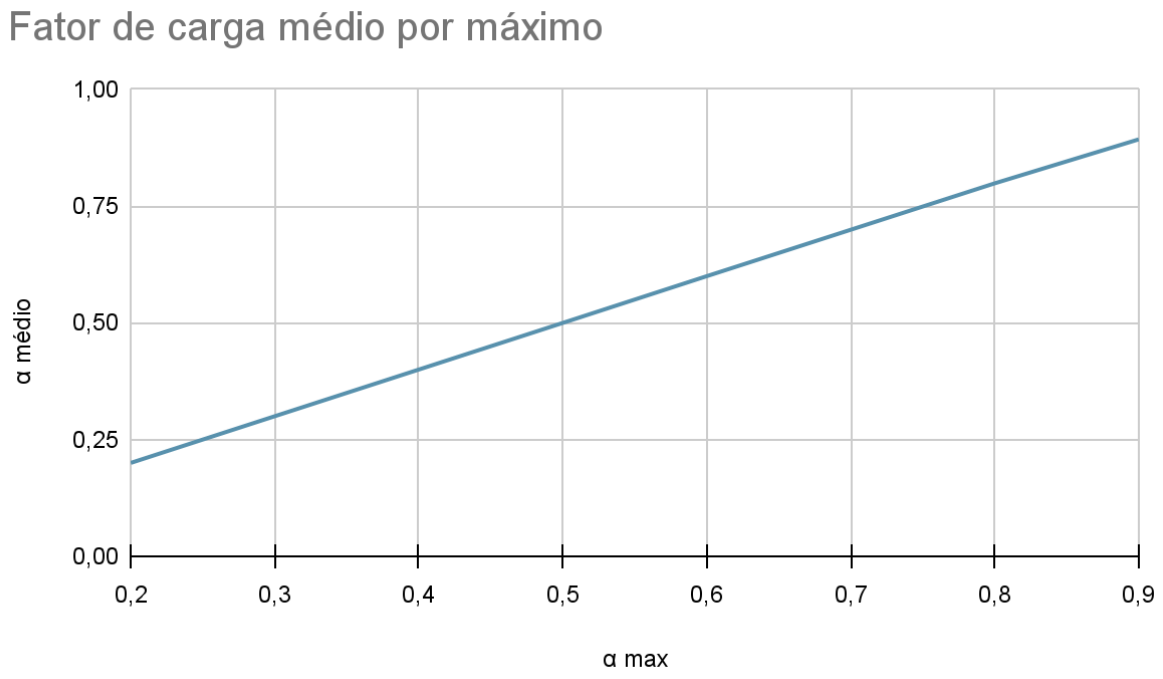
Desempenho quanto ao espaço

Para analisar o desempenho quanto ao espaço, utilizaremos 4 comparações, variando o tamanho de cada página(p) ou o fator de carga máximo(α^{max}) para verificar a variação no fator de carga médio($\alpha^{médio}$) e o número de páginas adicionais(p*) . Para todos os valores de

$\alpha^{médio}$ e p*, o programa foi rodado 10 vezes, cada uma obtendo um valor para eles, e foi feita a média.

Comparação 1: fator de carga médio($\alpha^{médio}$) por fator de carga máximo(α^{max}). (p = 5).

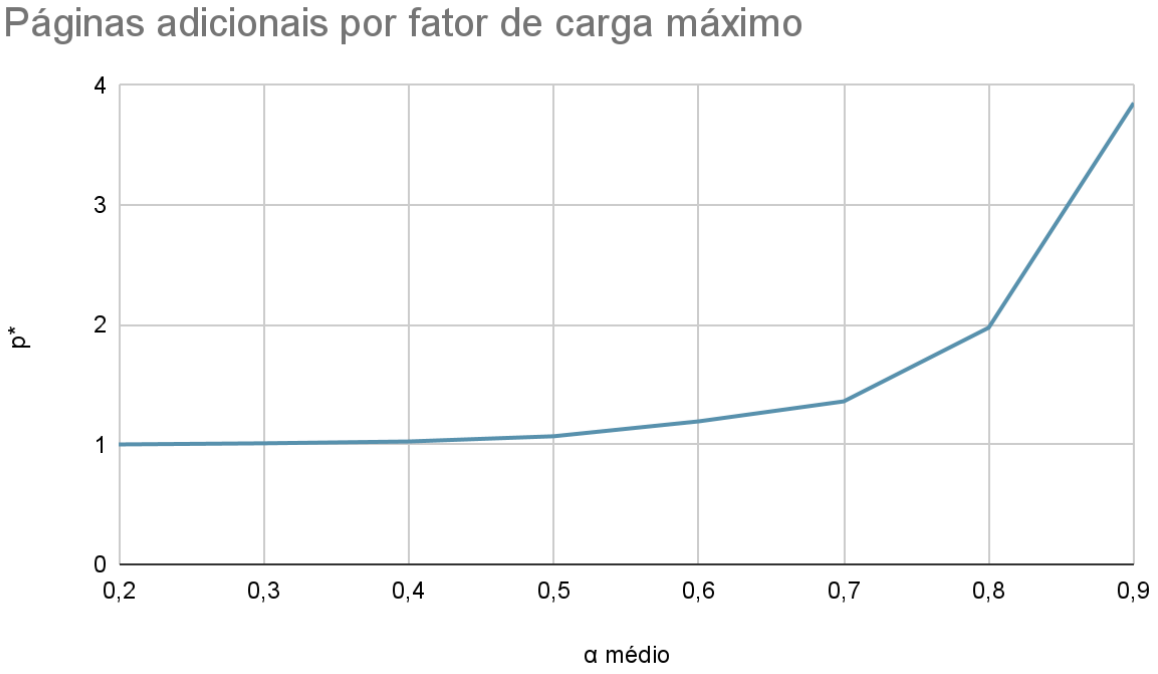
α^{max}	$\alpha^{médio}$
0.2	0.199960008
0.3	0.299940012
0.4	0.399840064
0.5	0.4997501249
0.6	0.599880024
0.7	0.6993006993
0.8	0.7987220447
0.9	0.8928571429



Observa-se que os valores de $\alpha^{médio}$ são bem próximos dos de α^{max} e, com isso, crescem linearmente com eles. Isso ocorre devido ao grande número de chaves na tabela hash, garantindo que, mesmo logo após o incremento do l do hashing linear o $\alpha^{médio}$ continua bem próximo do α^{max}.

Comparação 2: Número de páginas adicionais(p*) por fator de carga máximo(α^{max}). (p = 5).

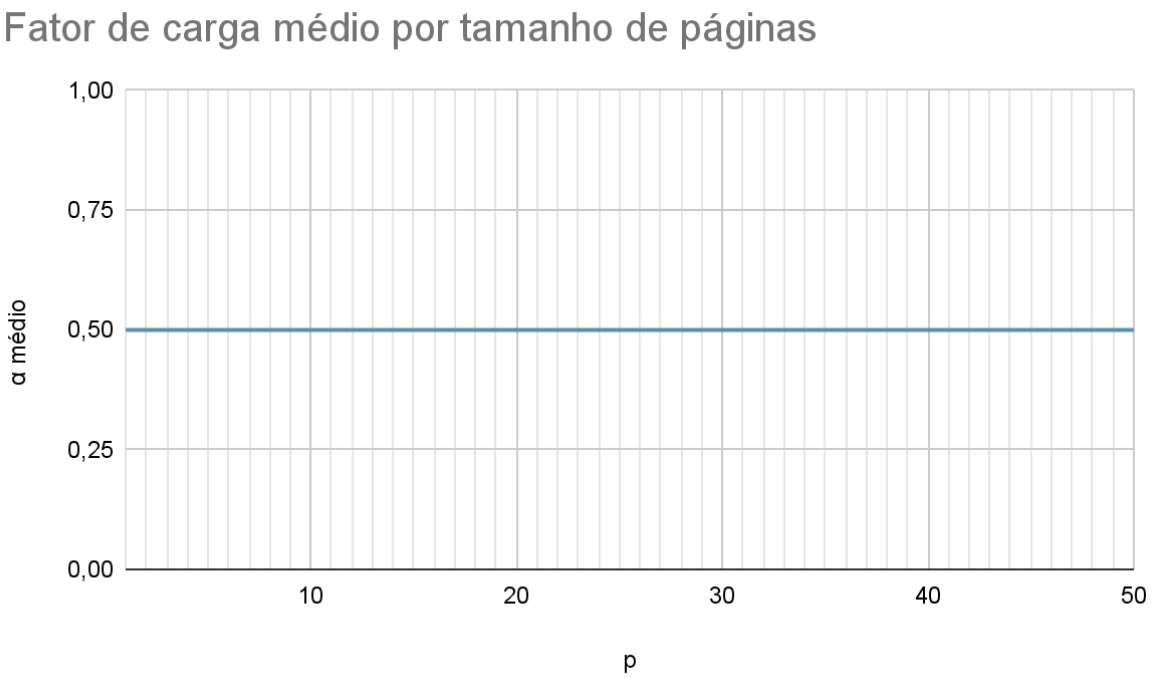
α^{max}	p*
0.2	1.001181523
0.3	1.010672587
0.4	1.025804620
0.5	1.069510959
0.6	1.193936085
0.7	1.360703535
0.8	1.976306969
0.9	3.852511716



Observa-se que os valores de p* crescem exponencialmente com o aumento de α^{max}. Quando maior o fator de carga máximo, mais se cria páginas adicionais dentro de cada hash antes que seja necessário aumentar o número de valores de hash.

Comparação 3: fator de carga médio($\alpha^{médio}$) por tamanho de cada página(p). ($\alpha^{max} = 0.5$).

p	$\alpha^{médio}$
1	0.4997501249
5	0.4997501249
10	0.4997501249
20	0.4997501249
50	0.4997501249

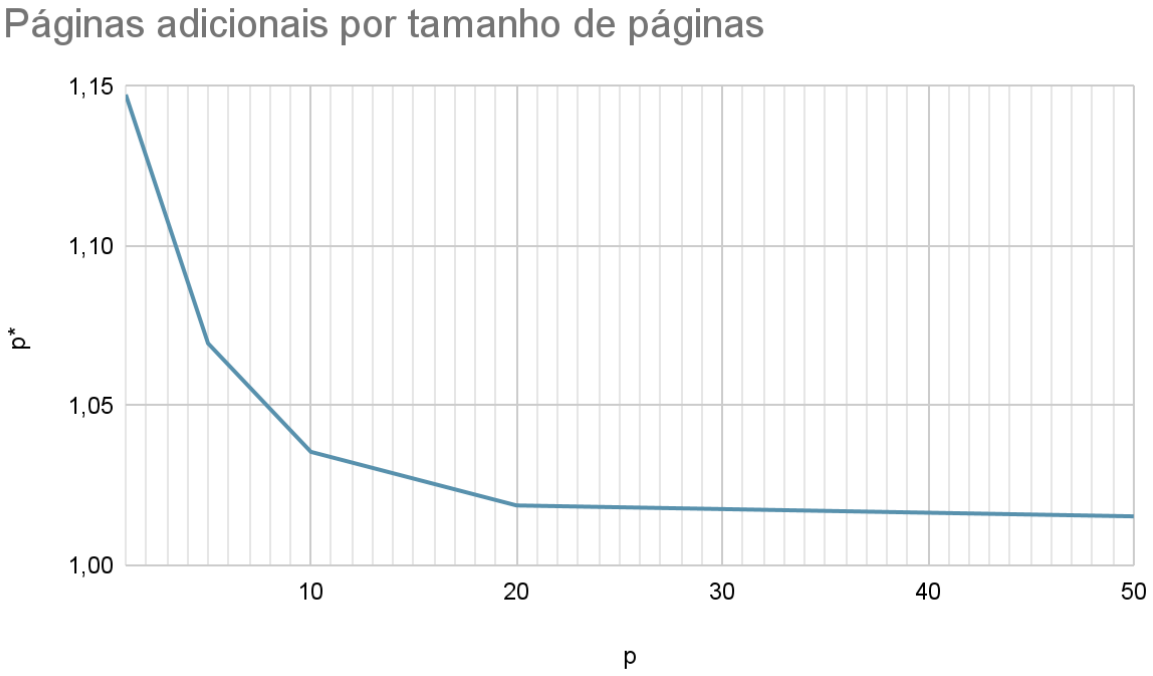


Observa-se que o valor de $\alpha^{médio}$ não se altera com a mudança do tamanho das páginas. O tamanho das páginas não é relevante para o fator de carga, pois este depende realmente da quantidade de chaves e da quantidade de valores hash no hashing linear.

Comparação 4: fator de carga médio($\alpha^{médio}$) por tamanho de cada página(p). ($\alpha^{max} = 0.5$).

p	p*
1	1.147397505
5	1.069510959
10	1.035532506
20	1.018751394

50	1.015337005



Observa-se que o valor de p * diminui exponencialmente com o tamanho das páginas. Quanto maior a página, mais chaves serão necessárias para que páginas adicionais sejam necessárias.

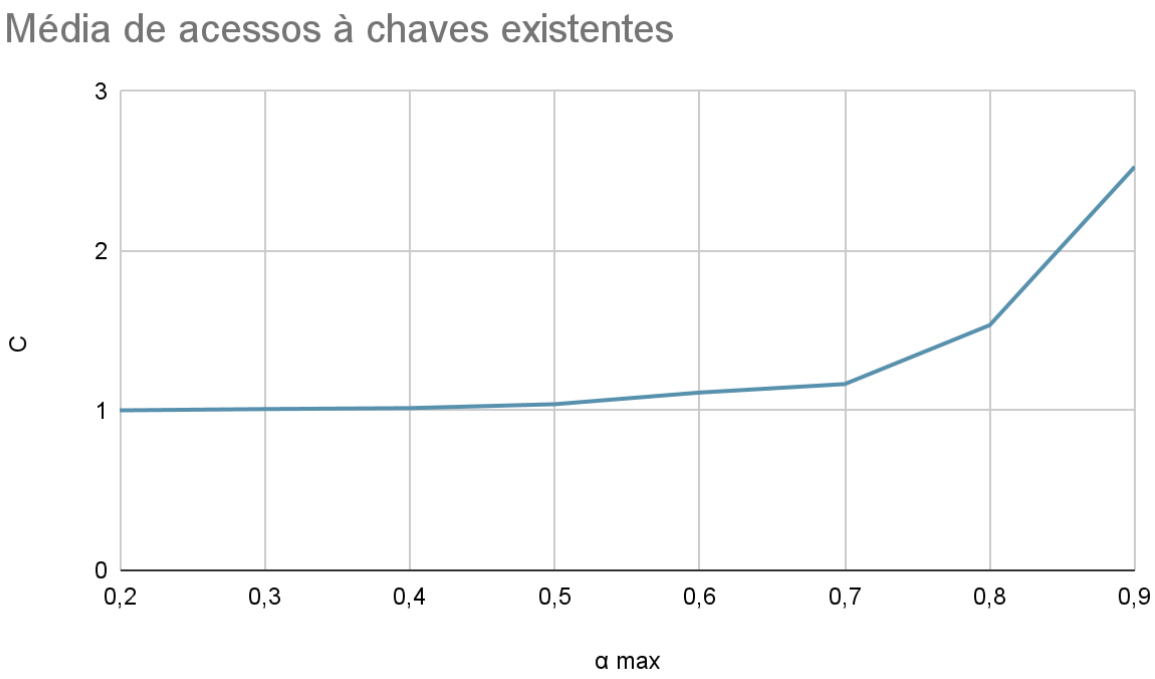
Desempenho quanto ao espaço

Para analisar o desempenho quanto ao espaço, iremos variar o tamanho de cada página(p) ou o fator de carga máximo(α^{max}) para verificar como se comportam a média da quantidade de acessos buscando chaves que se encontram na tabela hash(C) e a média da quantidade de acessos buscando chaves que não se encontram na tabela hash(S) . Para todos os valores de C e S, o programa fez a média buscando k = (0,2*chaves totais na tabela) chaves diferentes dentro das condições de C e S, usando a quantidade de acessos para encontrar cada uma no caso de C ou determinar que ela não se encontra na tabela no caso de S . O programa foi rodado 2 vezes para se obter um valor médio de C e S, tendo em vista que como já tinha sido feita média com grande quantidade de valores, o erro seria mínimo.

Variando α^{max} (p = 5):

Chaves existentes:

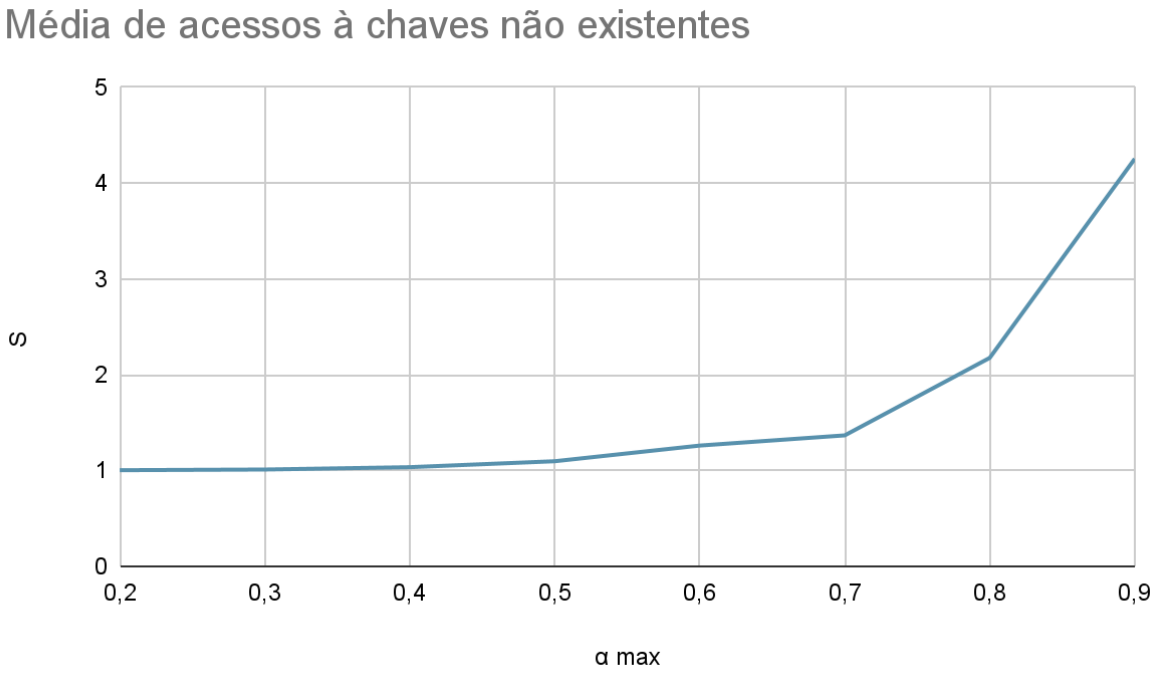
α^{max}	C
0.2	1.002
0.3	1.01
0.4	1.016
0.5	1.0405
0.6	1.1135
0.7	1.1665
0.8	1.5355
0.9	2.5275



Chaves não existentes:

α^{max}	S
0.2	1.0035
0.3	1.0105
0.4	1.035
0.5	1.097
0.6	1.2605
0.7	1.367

0.8	2.176
0.9	4.254

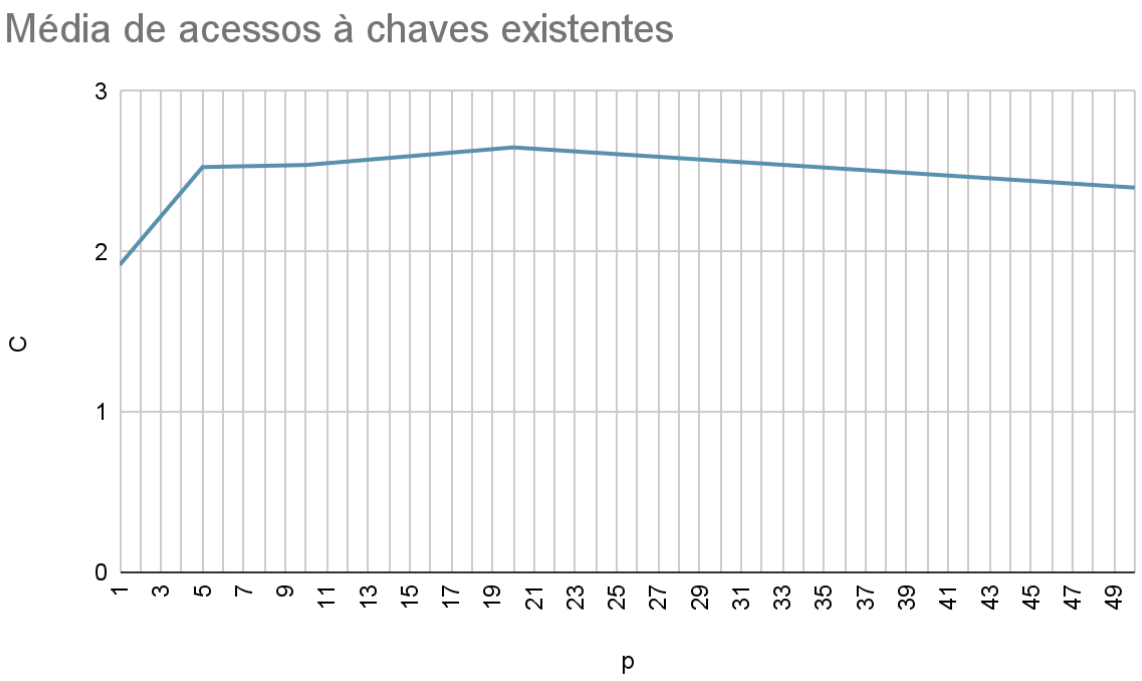


Nota-se que a quantidade de acessos aumenta exponencialmente com o aumento do fator de carga máximo. Também é notado que o crescimento é maior para as chaves que não existem na tabela hash. Vemos então como se diferem o caso médio e o pior caso para a busca de chaves em uma tabela hash usando hash linear, sendo eles muito próximos para pequenos valores de fator de carga máximo e mais destoantes à medida que esse fator de carga máximo aumenta. Isso ocorre pois quanto maior for o fator de carga máximo, mais páginas adicionais serão geradas dentro de cada valor de hash até que mais valores de hash devam ser adicionados. Dessa forma, as cadeias de páginas serão maiores e mais acessos serão necessários para percorrê-las. E como as chaves que existem na tabela em média estarão no meio da cadeia de páginas, a média de acessos cresce mais ou menos na metade da quantidade da média de acessos de chaves que não se encontram na tabela.

Variando p ($\alpha^{max} = 0.5$):

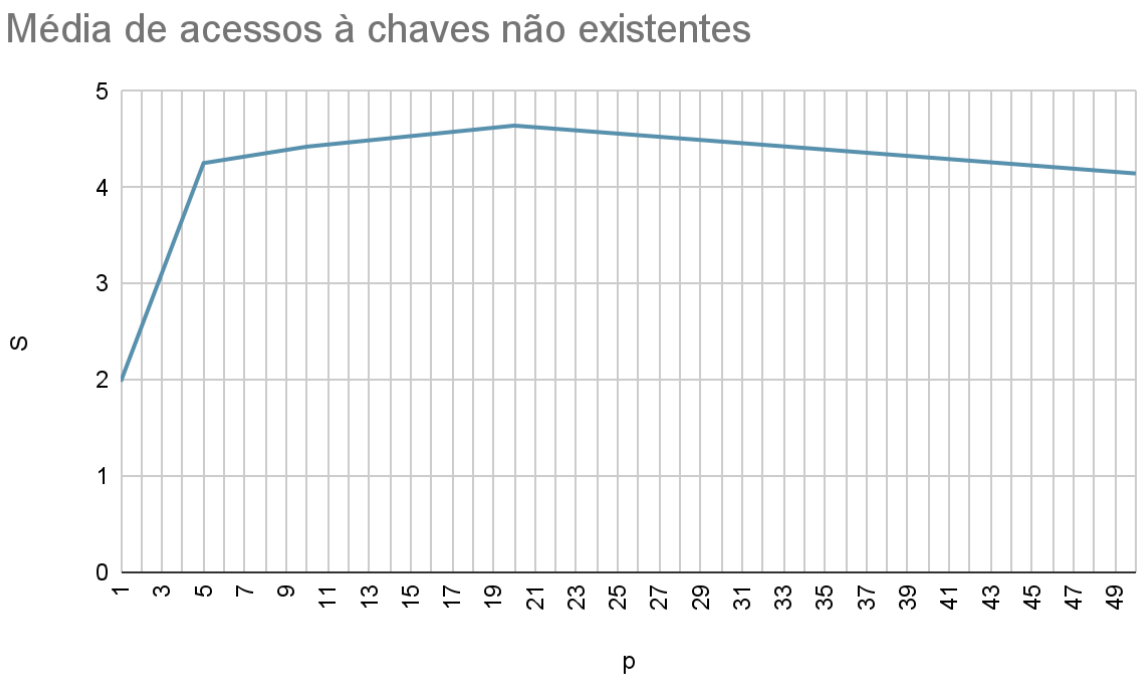
Chaves existentes:

p	C
1	1.9175
5	2.5275
10	2.54075
20	2.65025
50	2.3994



Chaves não existentes:

p	S
1	1.9825
5	4.254
10	4.42525
20	4.643575
50	4.1462

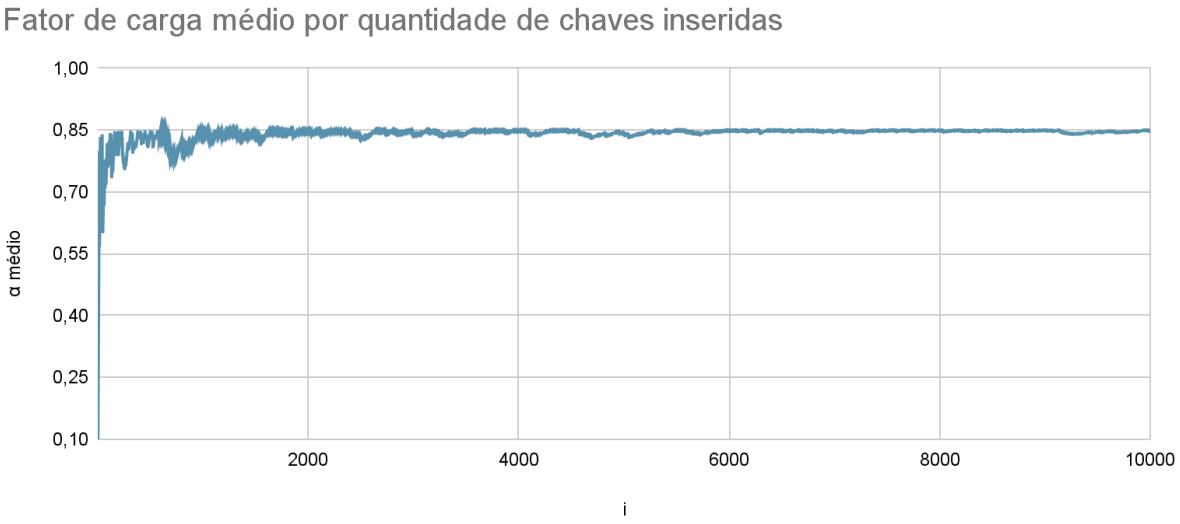


Assim como no fator de carga, a média de acesso para chaves que existem e que não existem na tabela hash é bem próxima para tamanhos de páginas menores, e com o tempo vão se tornando mais distintas. Isso também porque o crescimento da média de acessos para as chaves que não estão na tabela hash é maior. Porém, chega a um certo valor de tamanho de página em que as médias de acesso voltam a diminuir. Isso ocorre no momento em que o hash alcança o hash máximo e muitos outros valores de hash são criados.

Desempenho durante a inclusão dos n registros

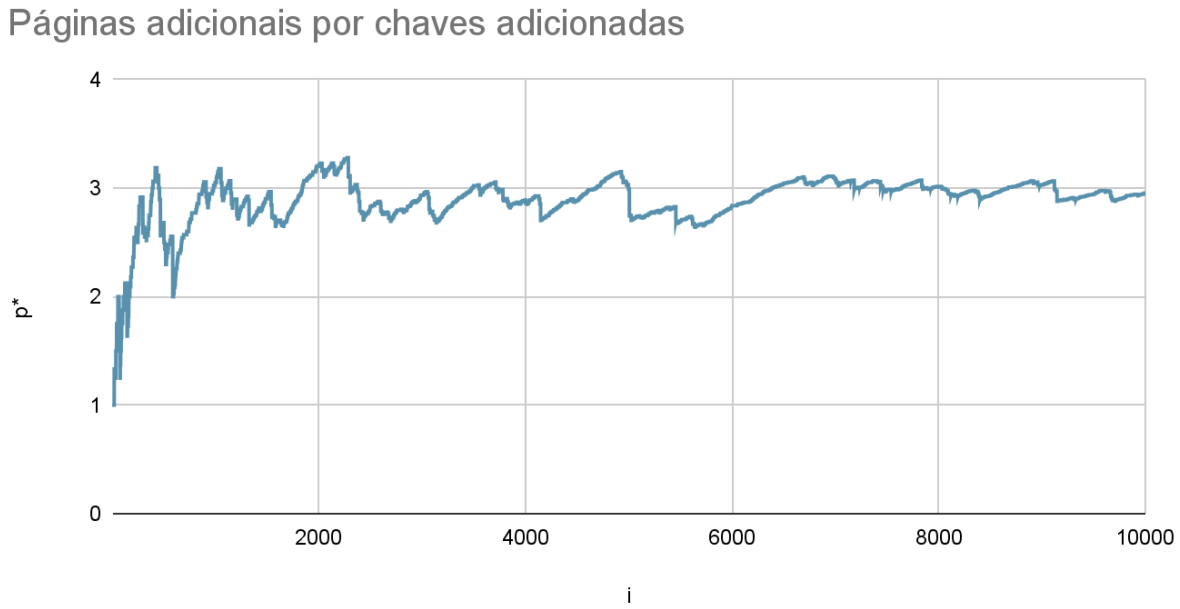
Para essa análise o programa foi alterado para ser iterativo, rodando 10 mil vezes e incrementando a quantidade de chaves que são adicionadas à tabela hash de 1 até 10 mil. cada vez os prints do programa seriam reduzidos apenas aos valores desejados de se obter, primeiramente o $\alpha^{médio}$, depois o p* e por fim o L^{max}.

Variando $\alpha^{médio}$ (α^{max} = 0.85, p = 10):



O fator de carga médio começa baixo, pois poucos espaços são preenchidos por chaves, como poucas ainda foram inseridas. Com a adição de novas chaves esse valor vai se aproximando do valor de carga máximo até chegar a estar a uma adição de chave para ultrapassá-lo. Nesse momento acontece o incremento do valor para cálculo do hash, aumentando o número de listas que os valores podem ser inseridos. Quando há a queda do valor do fator de carga médio é isso que acontece. E o processo se repete de inserir e o fator de carga médio se aproxima do máximo até reduzir de novo.

Variando p* (α^{max}= 0.85, p = 10):



A quantidade de páginas adicionais geradas varia bastante pelos valores aleatórios das chaves que são adicionadas, mesmo assim percebe-se um padrão. No começo, quanto mais chaves se adicionar, mais páginas extras serão geradas até o momento em que o fator de carga alcança o

máximo e o valor médio reduz. Dessa mesma forma continua ocorrendo, mas quanto mais chaves já existem na tabela hash menor é a redução das páginas adicionais no momento da redistribuição.

Variando L^{max} (α^{max}= 0.85, p = 10):

Quanto mais chaves são adicionadas, mais as listas vão aumentando e gerando páginas adicionais, até o momento que o fator de carga ultrapassa o máximo, e então a redistribuição das chaves faz com que a maior lista seja menor do que antes. E cada vez que há a redistribuição nos hashes aumenta-se a quantidade de listas e portanto, demora mais para se aumentar as listas com a adição de chaves aleatórias.