Relatório Trab 3 - INF1010

Alunos:
João Ricardo Malta - 2112714
Vinícius Machado - 2111343

Este relatório descreve os resultados de um experimento realizado para avaliar o desempenho de algoritmos de tabela hash com endereçamento aberto. Os algoritmos estudados incluem cálculo da hash, inserção, busca e exclusão de elementos em uma tabela hash de tamanho fixo. O objetivo é medir o tempo de execução de cada operação e analisar o comportamento dos algoritmos.

Para avaliar os algoritmos, utilizamos um conjunto de dados de placas de veículos contendo N elementos. Medimos o tempo necessário para executar as operações de inserção, busca e exclusão em diferentes tamanhos da entrada, variando N.
A tabela a seguir apresenta os tamanhos de entrada que foram considerados no experimento:

- Tamanho 1: 1000 elementos
- Tamanho 2: 512 elementos
- Tamanho 3: 256 elementos
- Tamanho 4: 128 elementos

Algoritmos Avaliados:

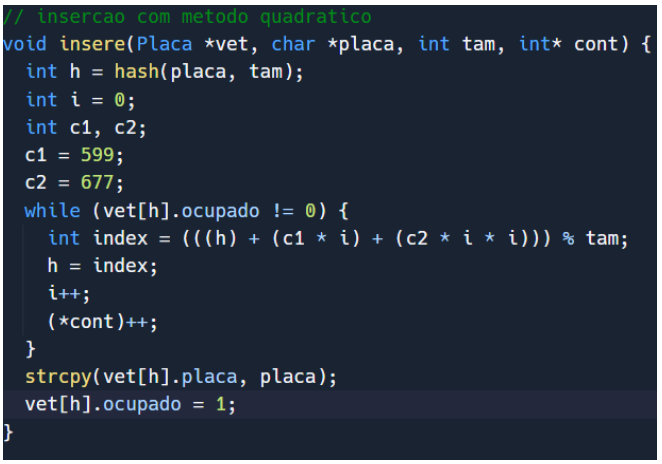
1. Cálculo do Hash (hash):
O Algoritmo começa inicializando uma constante chamada de hash e declarada com o valor 1. Que foi um valor escolhido após obtermos melhores resultados entre os diversos valores testados.
Então é criado uma variável C que irá armazenar o valor do caractere da placa e começamos um loop percorrendo todos os caracteres. Escolhemos percorrer todos os caracteres pois obteve um melhor resultado.
Dentro do loop, o algoritmo pega o próximo caractere da placa apontado por placa e o armazena em c. Em seguida, placa é incrementado para apontar para o próximo caractere na próxima iteração
O algoritmo calcula um novo valor para hash. Isso é feito em duas etapas:

- 1° : c é deslocado para a esquerda em 5 bits (<< 5) e adicionado a ele mesmo ((c << 5) + c). Isso é uma operação de bitshift que é usada para espalhar os bits do caractere c na nova posição do hash.
- 2°: hash é deslocado para a esquerda em 5 bits (<< 5) e adicionado a ele mesmo ((hash << 5) + hash). Isso é feito para espalhar os bits do valor atual de hash.

Após processar os 7 caracteres, o valor de hash é reduzido usando o operador de módulo (%), resultando em um valor final que é uma função do tamanho desejado (tam). Isso é feito para garantir que o valor final de hash seja um número dentro do intervalo desejado.

2. Inserção (insere):

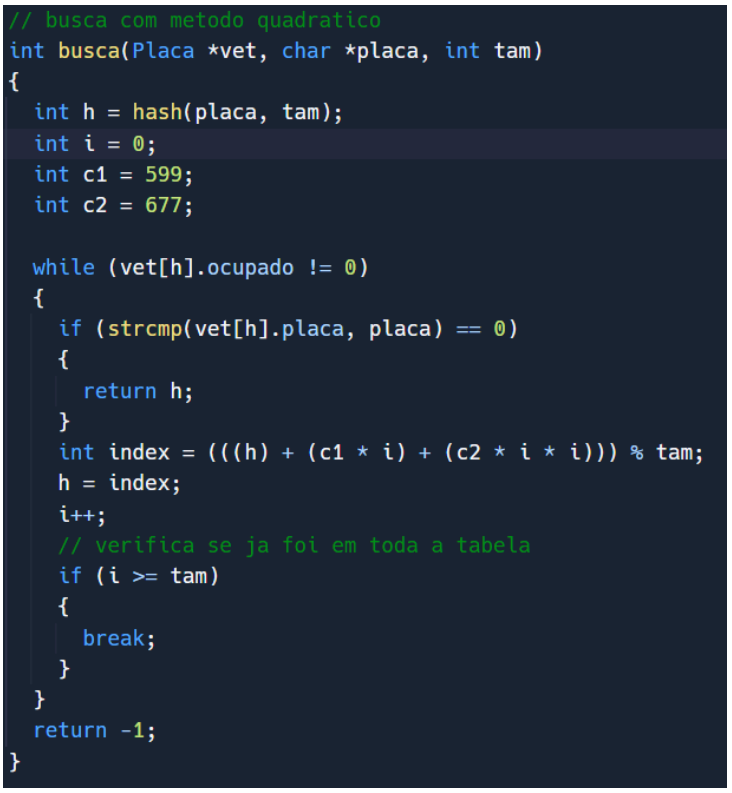
O algoritmo de inserção consiste em adicionar elementos à tabela hash com resolução de colisões por tentativa quadrática. Ele utiliza uma função de hash descrito acima para calcular o índice de inserção.
Primeiro, a função recebe como parâmetro a tabela hash (vet), a placa que deve ser inserida (placa), o tamanho da tabela (tam), e um contador de colisões (cont).
A função começa calculando o índice inicial, h, com base na placa e no tamanho da tabela. Isso é feito usando a função hash(placa, tam).
Em seguida, a função inicializa uma variável i com zero, variável essa que será usada para tratar colisões que possam ocorrer.
Também, são definidos os valores de c1(599) e c2 (677), que são fatores de sondagem usados para lidar com colisões no método quadrático.Os valores escolhidos foram arbitrários, os escolhemos pois são próximos de 1031 e primos.
A função entra em um loop while que continua enquanto a posição da tabela hash estiver ocupada, indicada pelo campo ocupado igual a 1.
Dentro do loop, a função calcula um novo índice, chamado de index, usando a fórmula do método de sondagem quadrática
O valor de h é atualizado para o valor de index, o que move o algoritmo para a próxima posição com base na fórmula de sondagem quadrática.
A cada iteração do loop, o contador cont é incrementado em 1, permitindo rastrear o número de colisões que ocorrem durante a inserção.
Quando o loop while termina, significa que uma posição vazia na tabela hash foi encontrada, e a placa é copiada para essa posição, vet[h].placa, na tabela.
Por fim, o campo ocupado na posição vet[h] é definido como 1, indicando que essa posição está agora ocupada.



```c
// insercao com metodo quadratico
void insere(Placa *vet, char *placa, int tam, int* cont) {
  int h = hash(placa, tam);
  int i = 0;
  int c1, c2;
  c1 = 599;
  c2 = 677;
  while (vet[h].ocupado != 0) {
    int index = (((h) + (c1 * i) + (c2 * i * i))) % tam;
    h = index;
    i++;
    (*cont)++;
  }
  strcpy(vet[h].placa, placa);
  vet[h].ocupado = 1;
}
```

3. Busca (busca)

O algoritmo de busca procura por elementos na tabela hash também utiliza a função de hash para calcular o índice de busca.
Inicialmente o mesmo processo para a inserção é executado, a diferença é que
dentro do loop, a função verifica se a placa na posição atual da tabela (vet[h]) é igual à placa que está sendo buscada (placa). Se a igualdade for verificada, a função retorna o índice atual "h", indicando que a placa foi encontrada com sucesso na tabela.
Em cada iteração do loop, a variável "i" é incrementada em 1, o que permite rastrear o número de colisões que ocorrem durante a busca.
A função também verifica se já percorreu toda a tabela, o que é feito verificando se "i" tornou-se maior ou igual ao tamanho da tabela (tam). Se essa condição for atendida, significa que a placa não foi encontrada na tabela, e a função retorna -1 para indicar que a busca não teve sucesso.
Por outro lado, se a placa for encontrada durante a busca, a função retorna o índice onde a placa está armazenada na tabela.



```c
// busca com metodo quadratico
int busca(Placa *vet, char *placa, int tam)
{
  int h = hash(placa, tam);
  int i = 0;
  int c1 = 599;
  int c2 = 677;

  while (vet[h].ocupado != 0)
  {
    if (strcmp(vet[h].placa, placa) == 0)
    {
      return h;
    }
    int index = (((h) + (c1 * i) + (c2 * i * i))) % tam;
    h = index;
    i++;
    // verifica se ja foi em toda a tabela
    if (i >= tam)
    {
      break;
    }
  }
  return -1;
}
```

4. Exclusão (exclui)

O algoritmo de exclusão remove elementos da tabela hash com base na chave. Ele identifica o elemento a ser excluído e o marca como vazio na tabela.
Inicialmente, a função chama a função de busca para encontrar o índice onde a placa se encontra na tabela. A busca é realizada com o auxílio da função "busca"
Se a busca não encontrar a placa na tabela,ou seja, a função "busca" retornar -1, a função de exclusão exibe uma mensagem de erro, indicando que o elemento não existe na tabela. Caso contrário, a função continua o processo de exclusão.
Na seção "else", a função procede com a exclusão efetiva da placa. Ela utiliza o índice retornado pela função de busca (que é o índice onde a placa se encontra na tabela). Nesse índice, a função atribui uma string vazia ("") ao campo "placa" na estrutura Placa, o que efetivamente remove a placa.
Além disso, a função também define o campo "ocupado" na estrutura Placa para 0, indicando que a posição na tabela agora está vazia e desocupada.
Por fim, a função exibe uma mensagem de sucesso, informando que o elemento foi excluído com êxito da tabela hash.

```c
// exclusao tabela hash
void exclui (Placa* vet , char *placa, int tam)
{
  int index = busca(vet, placa,tam);
  if (index==-1)
  {
    printf("Elemento nao existe na tabela\n");
  }
  else
  {
    strcpy(vet[index].placa, "");
    vet[index].ocupado =0;
    printf("Elemento excluido com sucesso\n");
  }
}
```

Coleta de Dados:

- Para cada tamanho de entrada, realizamos as seguintes etapas:
- Inicializamos a tabela hash.
- Abrimos o arquivo de entrada contendo as placas de veículos.
- Lemos o arquivo até o número de linhas desejado
- Inicializamos uma variável para contagem de colisões.
- Registramos o tempo de execução para as seguintes operações:
- Inserção de todas as placas.
- Busca de uma placa específica (por exemplo, "HFX5A59" que ocupa a linha 78 no arquivo e está dentro de todos os tamanhos)
- Funcionamento de todo o código

- Exclusão de uma placa específica. ("HFX5A59")
- Registramos o número de colisões durante a operação de inserção

**Resultados:
A tabela a seguir apresenta os tempos de execução das operações para diferentes tamanhos de entrada:

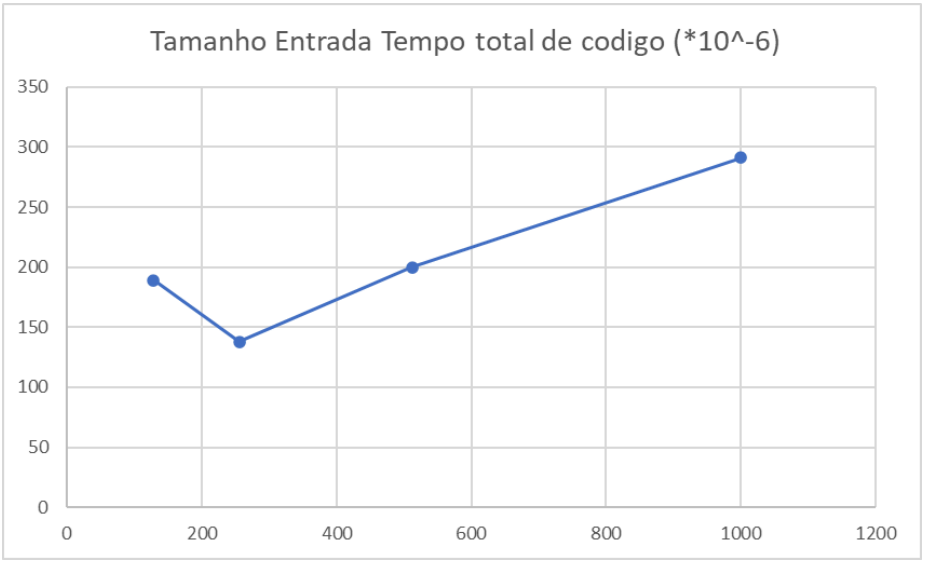
Tamanho de entrada	Tempo de inserção (s)	Tempo de busca (s)	Tempo de exclusão (s)	Número de colisões	Tempo total do código
1000	0.000187	0.000014	0.000010	3001	0.000291
512	0.000104	0.000007	0.000003	424	0.000200
256	0.000050	0.000005	0.000004	102	0.000138
128	0.000033	0.000032	0.000016	17	0.000189

Análise dos Resultados:
- Os tempos de inserção, busca e exclusão tendem a aumentar com o tamanho da entrada, indicando um comportamento esperado dos algoritmos.

- O número de colisões na operação de inserção também aumenta à medida que a tabela hash fica mais cheia. Isso é consistente com o método de sondagem quadrática.

- A busca por uma placa específica é realizada rapidamente, uma vez que a tabela hash já contém os elementos.

- A exclusão de uma placa específica também é eficiente, pois a localização do elemento é conhecida.



Conclusão

Os resultados deste experimento fornecem insights sobre o desempenho dos algoritmos de tabela hash com resolução de colisões por sondagem quadrática. Os tempos de execução variam de acordo com o tamanho da entrada, e a operação de busca é particularmente eficiente. A contagem de colisões é uma métrica importante para avaliar o comportamento dos algoritmos em tabelas hash cheias.