INSTITUTO FEDERAL
Mato Grosso
Campus Cuiabá
Cel. Octayde Jorge da Silva

INSTITUTO FEDERAL DE MATO GROSSO – IFMT
DEPARTAMENTO DA ÁREA DE INFORMÁTICA
ENGENHARIA DA COMPUTAÇÃO

LEANDRO KLEIN
WENDERSON PINHEIRO FARIAS

GRÁFICO DE ESPALHAMENTO NA TABELA HASH

Cuiabá - MT
Julho/2021

INSTITUTO FEDERAL DE MATO GROSSO – IFMT
DEPARTAMENTO DA ÁREA DE INFORMÁTICA
ENGENHARIA DA COMPUTAÇÃO

LEANDRO KLEIN
WENDERSON PINHEIRO FARIAS

DISCIPLINA: ESTRUTURA DE DADOS II
PROF. DR. JOÃO PAULO DELGADO PRETI

Cuiabá - MT
Julho/2021

RESUMO

Implementação do trabalho final da disciplina de Estrutura de Dados 2 (2021/1), referente ao “Gráfico de espalhamento na Tabela Hash”, cujo gráfico deverá ser entregue no formato PPM, com o propósito de melhor visualizar as diferentes estratégias quanto ao preenchimento da tabela hash de forma mais eficiente e, concomitantemente, tratar das possíveis colisões no decorrer desta distribuição.

Para alimentação da tabela, foi sugerido um dicionário da Língua Portuguesa, com cerca de 28 mil palavras, no formato de texto (txt) para ser distribuído na tabela.

O trabalho deverá ser apresentado em três tamanhos de tabela e duas funções de hash para posterior análise.

ESPALHAMENTO NA TABELA HASH

Algoritmos de hash permitem a transformação de um bloco de dados em uma representação resumida, normalmente de poucos bytes de comprimento. Após passar pela função, o valor obtido representa um resumo dos dados que será colocado dentro de um intervalo sobre um bloco de dados e o número de chaves possíveis, o resultado da função retorna um determinado ponto dentro desse intervalo e com certa freqüência um espaço já mapeado pode vir a ser ocupado novamente, o que é conhecido por colisão.

A colisão implica num maior esforço computacional. Com isso em vista, um bom espalhamento nessa tabela hash envolve a questão da uniformidade (não haver aglomeração de dados num ponto e espaços vazios em outro) e a eficiência (as instruções devem ser reduzidas, vez que não deve ocupar tanto espaço como os dados originais).

IMPLEMENTAÇÃO

Para a construção presente código, utilizamos as bibliotecas implementadas em sala de aula, no semestre 2021/1, pelo Prof. Dr. João Paulo Delgado Preti, as quais são (Hash.c, Hash.h, DoublyLinkedList.c e DoublyLinkedList.h), que tratam justamente de uma forma eficiente para se navegar e preencher uma lista duplamente ligada.

Para implementação, foi utilizada a linguagem C, adicionando as bibliotecas acima mencionadas, adaptando algumas funções como a de leitura do arquivo de texto e algumas opções estatísticas quanto ao preenchimento e a ocorrência das colisões.

Ao final, o usuário tem a opção de gerar um “Mapa de Espalhamento”, onde é possível verificar por meio de um gráfico os pontos de concentração (colisões) e até espaços reservados na memória que não foram preenchidos.

FUNCIONAMENTO DO PROGRAMA

O programa é apresentado na forma de um simples menu, onde o usuário tem as opções de:

1. Exibir toda a lista;
2. Verificar a porcentagem de ocupação da tabela;
3. Exibir as posições de hash que ocorreram colisões;
4. Gerar o mapa de espalhamento em formato PPM a ser lido no Gimp.

Após iniciada a tabela e carregamento do arquivo de texto (ações executados antes da apresentação do menu) a tabela já se encontra preenchida de acordo com o tamanho predefinido na biblioteca Hash.h. É necessário habilitar ou desabilitar #define MAX conforme a situação desejada, podendo ser utilizados outros. Em nosso teste foram elencados os tamanhos de hashes em 3025, 4900 e 10000.

Para o cálculo da função Hash, implementamos duas funções. Uma delas fornecidas pelo professor em sala de aula e outra encontramos numa dissertação apresentada pelo mestrando Chayner Cordeiro Barros, apresentada na Universidade Federal de Goiás em 2020.

Para geração do mapa de espalhamento tivemos que utilizar os comandos para manipular arquivo. Foi definida uma variável que recebia o valor cheio da cor, no caso 255 para cor verde, dividido pelo tamanho das hashes maior ou igual a 1. As posições vazias receberam uma cor em tom amarelo-claro, para distinguir das posições próximas a 1. Isso é feito dentro de um laço que vai escrevendo vários fprint no arquivo. Não havendo mais hashes a serem buscadas o laço é finalizado e o arquivo imagem.ppm é fechado.

Na visualização do mapa, a ser feita via software Gimp ou semelhante, as faixas ou conjunto de pontos próximos indicam que houve colisões muito próximas uma da outra. De outro modo, faixas claras ou aglomerações de pontos claros indicam que nessa parte houve uma boa distribuição.

Já a presença de pontos em tom de amarelo-claro indica que naquele espaço não houve qualquer preenchimento, ou seja, se nessa tabela há concentração de pontos escuros e ocorrência de pontos amarelo-claro, indica que a função hash não funcionou muito bem para aquele caso. Não quer dizer que sempre vai ser ruim, mas para aquele determinado pode haver uma função que melhor atenda a situação.

MODO DE TESTE

Para realização dos testes sugeridos (3 tamanhos de hashes e 2 funções), implementamos seis situações de acordo com as possibilidades acima. Dessa forma, por meio de comentários habilitamos/desabilitamos essas opções e compilamos seis programas diferentes com essas pequenas alterações.

Assim, estabelecemos seis casos diferentes divididos em seis pastas:

Caso 1: Função hash tipo 1 com tamanho MAX de 3025 (Arquivo PPM 55 x 55).
Caso 2: Função hash tipo 2 com tamanho MAX de 3025 (Arquivo PPM 55 x 55).
Caso 3: Função hash tipo 1 com tamanho MAX de 4900 (Arquivo PPM 70 x 70).
Caso 4: Função hash tipo 2 com tamanho MAX de 4900 (Arquivo PPM 70 x 70).
Caso 5: Função hash tipo 1 com tamanho MAX de 10000 (Arquivo PPM 100 x 100).
Caso 6: Função hash tipo 2 com tamanho MAX de 4900 (Arquivo PPM 70 x 70).

Para compilação, utilizamos o seguinte comando via terminal (gcc):

gcc main.c DoublyLinkedList.c DoublyLinkedList.h Hash.c Hash.h -o teste

CONSIDERAÇÕES FINAIS

O presente trabalho contribuiu bastante para um melhor entendimento acerca da tabela hash. Após a implementação foi possível visualizar a eficiência e uniformidade do espalhamento dos dados. Constatamos que a depender do tamanho da tabela e da função de hash utilizada para cálculo da chave podemos obter resultados muito variados. Logo, uma função que atende satisfatoriamente determinada situação pode, num outro contexto, ter um resultado totalmente adverso e isso significa desperdício de memória alocada ou um custo computacional maior.

OPÇÕES DE TESTE DO PROGRAMA(requer habilitar/desabilitar via comentário)

Função 1:

```
int hash(char *key) {
    int sum = 0;
    //Percorre todos os caracteres da string passada
    for (int i = 0; key[i]!=0;i++) {
        //acumulamos os códigos ascii de cada letra com um peso
        sum+=key[i]*(i+1);
    }
    return sum%MAX;  //retorna o resto da divisão*/
}
```

Função 2:

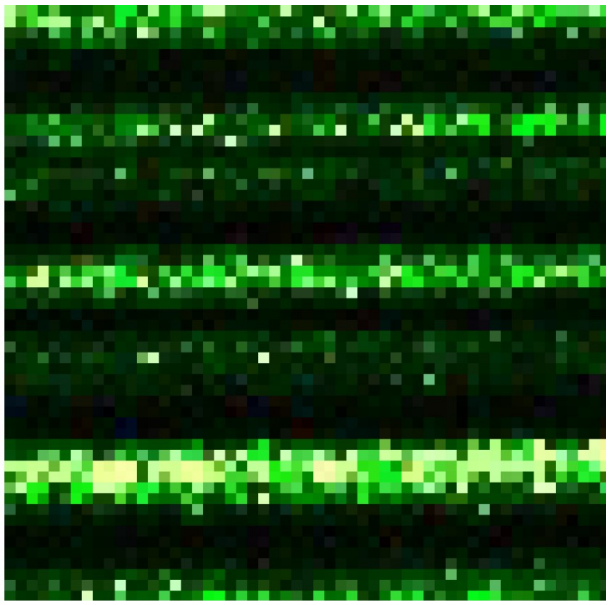
```
int hash(char *key) {
    unsigned long hash = 5381;
    int c;
    while ((c = *key++))
        hash = c + (hash << 6) + (hash << 16) - hash;
        return hash % MAX;
}
```

Tamanhos de tabela pré-definidos (MAX)

MAX 3025 hashes → Gera um arquivo PPM com distribuição de 55 x 55.
MAX 4900 hashes → Gera um arquivo PPM com distribuição de 70 x 70.
MAX 10000 hases → Gera um arquivo PPM com distribuição de 100 x 100.

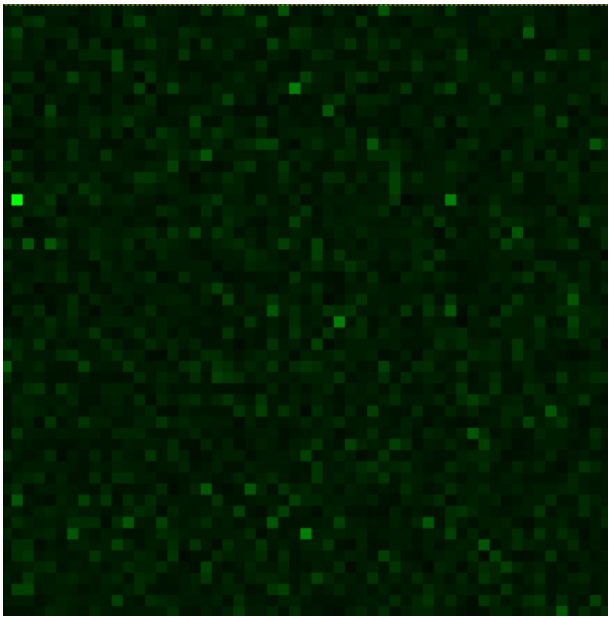
TESTES REALIZADOS E RESULTADOS OBTIDOS

Caso 1: Função hash tipo 1 com tamanho MAX de 3025 (Arquivo PPM 55 x 55).



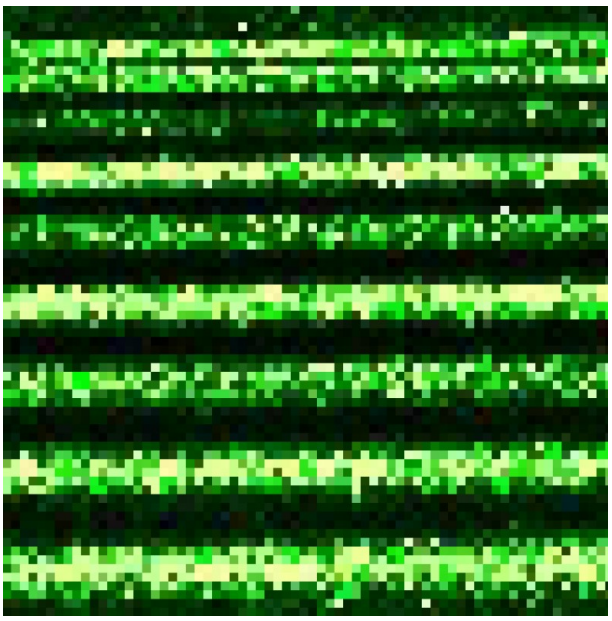
- Porcentagem de Ocupação: 94,94%.
- Quantidade de colisões: 2649.
- Presença de faixas de aglomeração, sendo as mais escuras pontos onde ocorreram muitas colisões.
- Presença de faixas verdes em tons mais claros indicando que houve poucas colisões.
- Presença de espaços em tom amarelo claro indicando que estes espaços não foram utilizados.

Caso 2: Função hash tipo 2 com tamanho MAX de 3025 (Arquivo PPM 55 x 55).



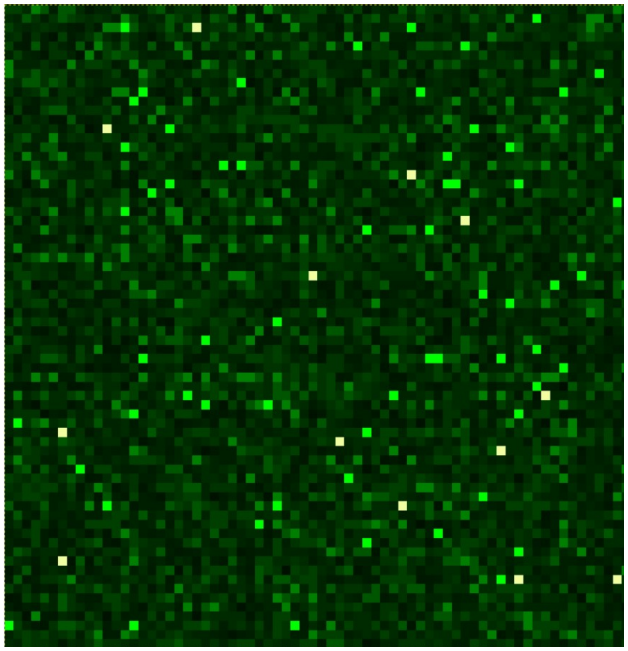
- Porcentagem de Ocupação: 100%.
- Quantidade de colisões: 3024.
- Não há presença de faixas, nem claras nem escuras.
- Há colisões. Contudo, os tons ficaram mais claros do que a anterior, indicando uma melhor distribuição.
- Não há presença de pontos amarelo-claro, indicando que não há espaços sem ocupação.

Caso 3: Função hash tipo 1 com tamanho MAX de 4900 (Arquivo PPM 70 x 70).



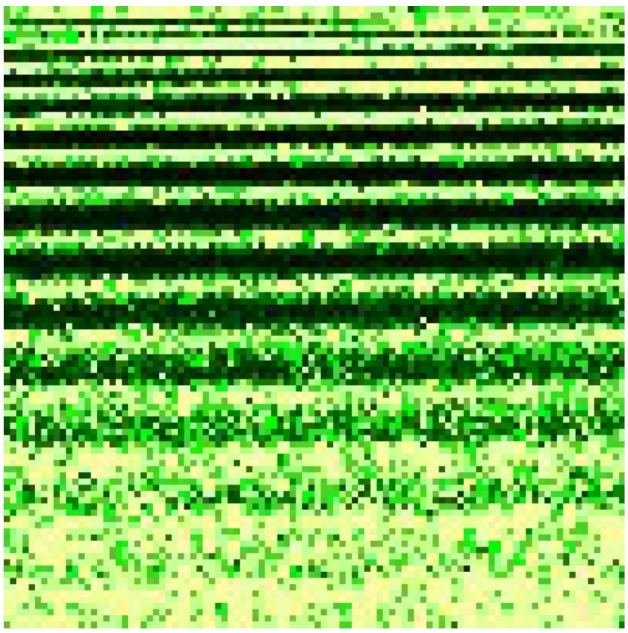
- Porcentagem de Ocupação: 83.86%.
- Quantidade de colisões: 3445.
- Presença de faixas de aglomeração, sendo as mais escuras pontos onde ocorreram muitas colisões. Com o aumento do tamanho ficaram mais espaçadas que a 55x55.
- Presença de faixas verdes em tons mais claros indicando que houve poucas colisões. Com o aumento do tamanho ficaram mais espaçadas que a 55x55.
- Presença de espaços em tom amarelo claro indicando que estes espaços não foram utilizados. Com o aumento do tamanho apareceram mais desses pontos do que a 55x55.

Caso 4: Função hash tipo 2 com tamanho MAX de 4900 (Arquivo PPM 70 x 70).



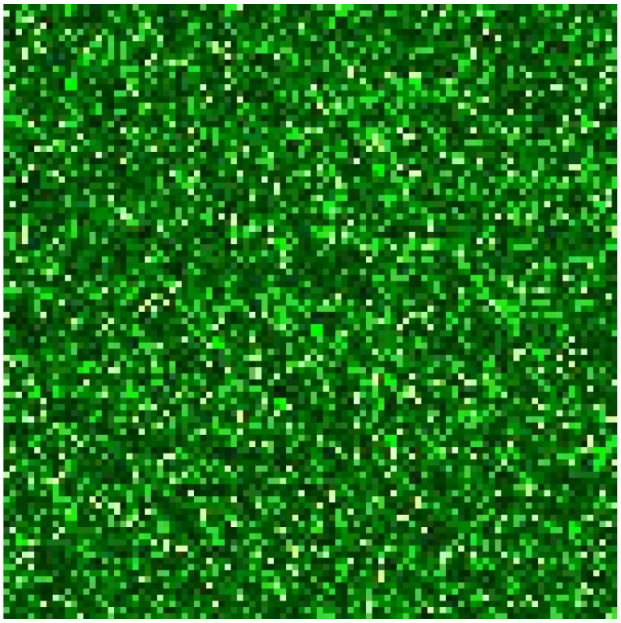
- Porcentagem de Ocupação: 99.73%.
- Quantidade de colisões: 4826.
- Não há presença de faixas, nem claras nem escuras.
- Há colisões. Contudo, os tons ficaram mais claros do que a anterior, indicando uma melhor distribuição.
- Há presença de pontos amarelo-claro, indicando que há espaços sem ocupação. Isso se deve ao aumento de espaço na memória.

Caso 5: Função hash tipo 1 com tamanho MAX de 10000 (Arquivo PPM 100 x 100).



- Porcentagem de Ocupação: 55.60%.
- Quantidade de colisões: 3851.
- Faixas escuras indicando colisões concentradas no início da memória reservada.
- Na segunda metade da imagem é possível observar a presença de faixas verdes em tons mais claros indicando que houve poucas colisões, mais elas vão ficando cada vez mais esparsas.
- Presença de muitos espaços em tom amarelo claro indicando que estes espaços não foram utilizados e até houve concentração destes. Com o aumento do tamanho apareceram mais pontos do que a 70x70.

Caso 6: Função hash tipo 2 com tamanho MAX de 4900 (Arquivo PPM 70 x 70).



- Porcentagem de Ocupação: 95.24%.
- Quantidade de colisões: 8004.
- Não há presença de faixas, nem claras nem escuras.
- Há colisões. Contudo, os tons ficaram mais claros do que a anterior, indicando uma melhor distribuição.
- Há presença de pontos amarelo-claro, indicando que há espaços sem ocupação. Isso se deve ao aumento de espaço na memória.

Conclusão: para este caso em específico de espalhamento a segunda função de hash é mais eficiente que a primeira. Contudo, é possível verificar que com o aumento do espaço disponível a porcentagem de ocupação vai diminuindo e surgindo mais pontos amarelo-claro, indicando uma contínua perda de eficiência sendo necessárias alterações na função para aumentar essa eficiência e uniformidade no espalhamento, caso essa tabela aumente consideravelmente.