INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA CATARINENSE CAMPUS VIDEIRA

GUILHERME NADERER CHAVES

RELATÓRIO DO TRABALHO FINAL DE ESTRUTURA DE DADOS I

Manipulação de TXT com Hashing e QuickSort

Videira - SC
2022

Posso começar dizendo que, para quem já tinha desenvolvido um programa para lista encadeada dupla, pode reutilizar o código, a principal diferença, na minha opinião são duas: a inserção passa a ser sempre no tail (vira o novo tail) e a outra é que o valor do elemento agora é uma cadeia de caracteres (string).

Depois de ter feito todas as pequenas adaptações necessárias para funcionar com o que foi citado acima, eu aloquei memória para a lista de listas, e também já criei uma função para limpar todas as listas.

Então começo o que realmente importa, a função de hash ou *hashing*, nela eu recebo o nome, seja do TXT ou algum para inserir manualmente, onde eu separo caracter por caracter da palavra e multiplica cada um por sua posição, ao fim da palavra, eu retorno o resto da divisão do total da soma pelo total de listas.

```
int funcaoHash(char* name){
  int tamName = strlen(name);
   int hash = 0;

  for(int i=0; i<tamName; i++){
    hash += name[i] * (i+1);
  }
  hash %= TAM;
  return hash;
}
```

Depois monto um histograma, onde eu pego lista por lista, e o tamanho de cada dividido por 40 vai dar o total de riscos para o meu histograma, ou seja, cada risquinho vai ter um proporção de 1:40, cada um vale aproximadamente 40 nomes.

```
void histograma(Lista** lista){
  printf("\n%s  %s   %s\n\n", "Lista","Tamanho", "Histograma
(1:40)");
  for(int pos=0; pos<TAM; pos++){
    printf("%3u %9.2f    ", pos+1, (float)lista[pos]->size/40);
    for(int qtd=0; qtd<= (lista[pos]->size/40); qtd++){
      printf("-");
    }
    puts("");
  }
}
```

Praticamente junto com o histograma, mostro a média de nomes em cada lista, a com mais e a com menos nomes. Para isso só percorro a lista de listas, pegando o *size* de cada uma, somando em uma variável e ao mesmo tempo verificando as menores e maiores.

```c
void calculos(Lista** lista){
  int media=0,max=0,min=0;
  min = lista[0]-> size;
  //Percorrendo o Vetor de Listas e somando os tamanhos
  for(int i = 0; i < TAM; i++){
    media += lista[i]-> size;
    if(max<lista[i]->size){
      max=lista[i]->size;
    }
    if(min>lista[i]->size){
      min=lista[i]->size;
    }
  }
  media /= TAM;    //Calulando a média entre as listas

  printf("\nCálculos ---------- \n\nMédia: %d \nMaior Lista: %d
\nMenor Lista: %d\n\n", media, max, min);
}
```

Agora para o QuickSort vamos precisar ter uma função de troca de elementos, onde no meu caso, no QuickSort só funcionou trocando só os valores, então deixei assim.

```c
void trocar(Lista* lista, Elemento* um, Elemento* dois){
  char bkp[30];
  strcpy(bkp, um->name);
  strcpy(um->name, dois->name);
  strcpy(dois->name, bkp);
}
```

Para continuar o QuickSort, precisamos de uma função que faz mais ou menos uma sub-repartição, essa função acha a posição correta do elemento a ser ordenado, e deixa à esquerda os menores e a direta dele os maiores. No meu caso, o elemento que ele descobre a posição é sempre o último elemento da lista ou depois da sub-repartição.

```c
Elemento* partition(Lista* lista, Elemento* left, Elemento* last){
  char* pivo = last->name;
  Elemento* i = left->prev;
  for (Elemento* rodar = left; rodar != last; rodar = rodar->next){
    if (strcmp(rodar->name, pivo) <= 0){
      i = (i == NULL) ? left : i->next;
      trocar(lista, i, rodar);
    }
  }
  i = (i == NULL) ? left : i->next;
  trocar(lista, i, last);
  return i;
}
```

E então, a última função é o QuickSort em si, onde ele usa de recursividade para ordenar a(s) lista(s). A primeira vez sempre passando o primeiro e último elemento da lista, ou seja, o head e o tail. Depois que ele acha a posição correta do tail, ele se chama novamente, passando uma sub-repartição do começo até um antes do tail que agora já está na posição correta, e se chama mais uma vez para ordenar um depois dele até o fim da lista. Assim sucessivamente até terminar.

```c
void quicksort(Lista* lista, Elemento* esquerda, Elemento* direita)
{
  if(esquerda != NULL && direita != NULL && esquerda != direita &&
esquerda != direita->next){
    Elemento* aux = partition(lista, esquerda, direita);
    quicksort(lista, esquerda, aux->prev);
    quicksort(lista, aux->next, direita);
  }
}
```

Com tudo isso feito, é só criar um menuzinho trabalhando com as funções acima, e abrir o arquivo TXT, pegar os nomes, inserir eles na lista correta usando o hashing e depois ordenar com o QuickSort.

Para abrir o arquivo:

```c
//Upload e leitura do arquivo txt ---------------------------------------
FILE *file;                     //declarando o ponteiro com os nomes
file = fopen("nomes.txt","r");  //recebendo os nomes no ponteiro
if(file == NULL){               //verificando se o txt era vazio
  printf("Não foi possível ler o arquivo\n");
  return 0;
}
```

Para pegar os nomes:

```
char nome[30];                          //Variável que armazena cada nome
char* result; //Variável que vai armazenar se foi possível inserir ou não
while(fgets(nome,30,file) != NULL){//Método para pegar string por linha
  strcpy(nome, strtok(nome, "\n"));
  int indice = funcaoHash(nome);      //Recebendo o índice de armazenamento dada pela função de hashing
  result = inserir(VetorListas[indice], strcpy(nome, nome)); //Armazenando nome na lista
}
```

Obs: a função strtok tira as quebras de linha para não termos problemas depois manipulando as strings.