Trabalho Prático 2: O problema da agenda de viagens de Rick Sanchez

Paulo Henrique Maciel Fraga 2018054451
Universidade Federal de Minas Gerais (UFMG)

Resumo. *Documentação do trabalho prático aplicado pela Professora Olga N. Goussevskaia para a turma de Estrutura de Dados 2019/2.*

1. Introdução

O problema proposto no trabalho recebe como entrada uma lista de tamanho pré-definido de planetas que contém um nome e um tempo de visitação. O objetivo do problema é, dado um tempo limite mensal, é necessário determinar o número de planetas que devem ser visitados em cada mês para que toda a lista de planetas seja visitada. Além disso, é necessário se maximizar o número de planetas que serão visitados nós meses iniciais escolhendo os planetas com menor tempo de visitação e agrupando-os no início. Por fim, o cronograma mensal deve ser apresentado em ordem alfabética. Para isso serão necessárias duas ordenações principais: o tempo dos planetas em ordem decrescente e o nome dos planetas em um mês em ordem alfabética. A ordenação temporal deve ser estável e possuir complexidade de tempo O(n log2 n) e a ordenação dos nomes dever ter complexidade de tempo de o O(n × k), dado que k é tamanho da cadeia de caracteres que identifica um planeta.

2. Implementação

Para a realização das ordenações propostas foram escolhidos algoritmos já conhecidos. Para a ordenação da lista de planetas pelo tempo de visitação uma variação do algoritmo mergeSort foi implementado, por se tratar de um algor timo estável, como pedido no enunciado, e com melhor, pior e caso médio equivalentes a O(n log n). Ademais, para a ordenação dos nomes do planeta uma variação do RadixSort que utiliza uma chamada do BucketSort para cada carácter do nome do planeta foi utilizada. A escolha se deu por essa variação do algoritmo que possuir complexidade O(n × k) com k representando o tamanho da cadeia de caráteres que representa o nome e foi viável uma vez que o tamanho máximo do vetor auxiliar gerado ser muito pequeno e não ultrapassar o tamanho 26, por se considerar que os nomes dos planetas possuíam apenas letras minusculas. Além disso, o algoritmo é estável.

O compilador utlizado foi G++ 5.4.0 e a pasta src contém os arquivos principais do programa, que serão explicados em detalhes a baixo. Eles são: main.cc, merge.h, merge.cc, radix.h e radix.cc.

A primeira parte da main.cc se é responsável por captar as entradas e salvá-las em uma struct chamado visitas que possui o int tempo e um ponteiro para um vetor de caracteres que representa o nome, alocando memória dinamicamente para isso. Além de receber o tempo total mensal, o número de planetas e o tamanho dos nomes.

Logo em seguida uma variação do mergeSort é chamada para ordenar a entrada baseada no tempo de visitação. A função está disponível nos arquivos merge.h e merge.cc e é composta por uma chamada recursiva que recebe como referência um vetor de visitas (struct definida acima) e ints que representam o início e o fim do vetor ou subvetor. A função se chama recursivamente diminuindo o vetor recebido na metade (o vetor não se altera, apenas os indicadores esquerda e direita) ate que chegue no caso base em que esquerda e direita são a mesma posição. Quando isso ocorre, a função junta que recebe como parâmetros o início do vetor a esquerda, o fim do vetor a direita e a divisória entre os dois vetores, ordena, em um vetor auxiliar, pelo método de inserção (método altamente adaptável), os dois subvetores parcialmente ordenados. Logo em seguida o vetor auxiliar é copiado no vetor principal nas posições em que representa.

De volta para a main, ordenada a lista de planetas por tempo de visitação, ocorre um looping que considera o número planetas na lista. Assim, em um while dentro dessa looping se determina o número máximo de planetas cuja soma dos tempos de visitação não ultrapassa o tempo total mensal estabelecido. Assim, conhecido o número de planetas máximos daquele mês, um vetor de visitas é alocado dinamicamente para representar as visitas desse mês. A variação do RadixSort é chamada para ordenar a lista em ordem alfabética (será explicado em breve) e os valores são imprimidos, juntamente com o número do mês, que é determinado pelo número da iteração realizada. Em seguida o vetor do mês é liberado da memória, as auxiliares utilizadas são atualizadas e o processo se repete para cada mês até que a lista seja toda percorrida.

Por fim, o Radix Sort. O algoritmo se encontra nas pastas radix.h e radix.cc e consiste basicamente em k chamadas do countingSort em que k representa o número de caracteres do nome de um planeta, ele começa do carácter menos relevante e termina no mais relevante. O countingSort implementado utiliza a struct contadores que possui um contador, inicialmente zerado, e um ponteiro para um vetor de visitas. Ele aloca memória dinamicamente baseada no maior caractere presente no nome de algum planeta. A letra ‘a’ (int 97) representa a posição zero desse vetor de contadores e sempre que um carácter é encontrado o vetor de contadores é atualizado na respectiva posição e o vetor de visitas é atualizado. No fim, o vetor auxiliar é descarregado no principal.

3. Instruções de compilação e execução

O trabalho pode ser compilado no linux utilizando-se o com o comando "make" enquanto dentro do domínio da pasta (paulo_fraga). O comando make gera o executável "tp2". Para rodar o executável basta clicar no .exec criado pelo comando make ou digitar "./tp2" no terminal enquanto no domínio que o executável se encontra.

4. Analise de complexidade

myMergeSort:

A função junta, dentro da merge, possui cerca de 2n comparações no pior e no melhor caso tendo complexidade O(n).

Assim, a complexidade do myMergeSort pode ser dada pela equação de recorrência

T(n) = 2T(n/2) + O(n), que representa o segundo caso do Teorema Mestre e, por isso, tem complexidade O(n log n), tanto no pior, quanto no melhor caso.

radix:

A função radix consiste em k chamadas da função countingSort que tem complexidade linear O(n) no melhor e no pior caso, logo a complexidade da função radix é dada por O(N x K) em que k é o tamanho do vetor de caracteres que compõem a entrada.

5. Conclusão

Com o trabalho que desenvolvi é possível se resolver todos os problemas propostos nas especificações. As implementações são autorais e não foram consultados códigos de terceiros, logo podem fugir um pouco (ou não, não sei), das implementações padrões dos algoritmos, desculpa a bagunça. Por fim, todos os cuidados foram tomados com a manipulação das memórias dinamicamente alocadas.

6. Bibliografia

Não foram consultados materiais adicionais para a confecção desse trabalho, apenas o que foi apresentado em sala de aula, ós slides postados no moodle.