Exercícios de Ordenação Comentados

Daniel Saad Nogueira Nunes

7 de outubro de 2024

Aviso

Os comentários a respeito das soluções das listas de exercício buscam explicar resumidamente a ideia para resolução dos problemas, mas sem "entregar o ouro". O objetivo é apenas fornecer uma direção para aqueles alunos que estejam com dificuldades. Para instruções mais detalhadas, sugiro procurar o professor por e-mail, sala de aula virtual da disciplina ou presencialmente nos horários de atendimento.

O objeto de discussão em questão se trata da lista de exercícios sobre ordenação publicada aqui.

Ordenando Palavras

Para produzir o resultado correto, basta utilizar um método de ordenação **estável** eficiente, como o Mergesort, que leve em consideração apenas o tamanho das palavras. Para agilizar as comparações, o tamanho das palavras pode ser pré-computado, para evitar chamadas à função `strlen`.

Código feito em sala de aula:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct string{
    char* str;
    int len;
}string;

void merge(string *v, string *v1, string *v2, size_t size) {

    size_t size_v1 = size / 2;
    size_t size_v2 = size - size_v1;
    size_t i = 0;
    size_t j = 0;
```

```
    size_t k = 0;

    /** Enquanto não chegar ao fim da primeira
     *  e da segunda metade **/
    for (i = 0; j < size_v1 && k < size_v2; i++) {
        /* Se o elemento da primeira metade
         * é menor ou igual ao da segunda metade,
         * insira-o no vetor resultado
         */
        if (v1[j].len <= v2[k].len) {
            strcpy(v[i].str, v1[j].str);
            v[i].len = v1[j].len;
            j++;
        }
        /* Caso contrário, insira o elemento da
         * segunda metade no vetor resultado */
        else {
            strcpy(v[i].str,v2[k].str);
            v[i].len = v2[k].len;
            k++;
        }
    }

    /** Se ainda  restam elementos na primeira partição **/
    while (j < size_v1) {
        /* Copiamos os elementos para o vetor resultado */
        strcpy(v[i].str, v1[j].str);
        v[i].len = v1[j].len;
        j++;
        i++;
    }
    /** Se ainda  restam elementos na segunda partição **/
    while (k < size_v2) {
        /* Copiamos os elementos para o vetor resultado */
        strcpy(v[i].str, v2[k].str);
        v[i].len = v2[k].len;
        k++;
        i++;
    }
}

void merge_sort(string *v, size_t size) {
    size_t mid;
    if (size > 1) {
        mid = size / 2;
        /* aloca espaço para os subvetores */
```

```
        string *v1 = malloc(sizeof(string) * mid);
        string *v2 = malloc(sizeof(string) * size - mid);
        /* Copia os elementos de v para os subvetores */
        int i;
        for (i = 0; i < mid; i++) {
            v1[i].str = malloc(11*sizeof(char));
            strcpy(v1[i].str,v[i].str);
            v1[i].len = v[i].len;
        }
        for (i = mid; i < size; i++) {
            v2[i-mid].str = malloc(11*sizeof(char));
            strcpy(v2[i-mid].str,v[i].str);
            v2[i-mid].len = v[i].len;
        }
        /* Ordena recursivamente a primeira metade */
        merge_sort(v1, mid);
        /* Ordena recursivamente a segunda metade */
        merge_sort(v2, size - mid);
        /* Faz a junção das duas metades */
        merge(v, v1, v2, size);
        /* Libera o espaço alocado */
        for (i = 0; i < mid; i++) {
            free(v1[i].str);
        }
        for (i = mid; i < size; i++) {
            free(v2[i-mid].str);
        }
        free(v1);
        free(v2);
    }
}

void sort(string* v,int n){
    merge_sort(v,n);
}


int main(void){
    int n;
    string* v;
    scanf("%d",&n);
    v = malloc(sizeof(string)*n);
    for(int i=0;i<n;i++){
        v[i].str = malloc(sizeof(char)*11);
    }
    for(int i=0;i<n;i++){
```

```
        scanf("%s",v[i].str);
        v[i].len = strlen(v[i].str);
    }
    sort(v,n);
    for(int i=0;i<n;i++){
        printf("%s\n",v[i].str);
        free(v[i].str);
    }
    free(v);
    return 0;
}
```

Fantástica Fábrica de Fibonaccis

Primeiramente computamos a sequência dos números de Fibonacci até o valor de 10^{18}, o que é feito rapidamente, uma vez que a função cresce exponencialmente. Certifique-se de utilizar o tipo de dados correto para representar essa sequência. Uma vez que a sequência esteja pronta (e naturalmente ordenada), percorra a sequência, da direita para a esquerda, subtraindo do número de entrada o maior número de Fibonacci que seja menor ou igual a ele. O processo é repetido até chegar a zero. A resposta é o número de iteraçoes necessárias alcançar a condição de parada. Contudo, existe uma pequena observação a ser feita: dois números consecutivos da sequência nunca devem ser utilizados.

Para mais detalhes, consulte o Teorema de Zeckendorf.

Números Distintos

O problema é trivial. Ordene a sequência de entrada usando um método eficiente de ordenação e depois a percorra. Sempre que `v[i+1]!=v[i]` o contador de números distintos deve ser incrementado.

Espetinho do Barbosinha

É interessante converter todos os tempos para segundos, facilitando as comparações posteriores. Construimos dois vetores: `inicio` e `fim`, com os tempos de início e fim de cada cliente. Em seguida, os vetores são ordenados. Os vetores são percorridos da seguinte forma: sempre que `inicio[i]<=fim[j]` existe uma sobreposição de intervalos, logo uma variável acumuladora deve ser incrementada, bem como o índice `i`. Quando não é o caso, incrementamos o índice `j` e a variável acumuladora é decrementada. A resposta é o valor máximo atingido pela variável acumuladora. O processo termina ao atingir o fim de um dos vetores.

F1

É recomendável tratar todos os tempos como inteiros em uma única unidade (milissegundos), para evitar erros de cálculo em ponto-flutuante. Uma vez que o tempo total de volta seja calculado, basta ordenar os pilotos em ordem crescente e realizar a conversão de milissegundos para o formato original ao imprimir a classificação.

Índice-h

É necessário usar duas etapas de ordenação para resolver este problema. Primeiro, ordenamos, para cada autor, o número de citações de cada publicação para cálcular o índice-h de cada autor. O índice-h é simplesmente o maior índice do vetor de citações ordenado cujo valor é maior ou igual ao próprio índice. Em seguida, basta ordenar os autores em ordem decrescente de índice-h. Não se esqueça que, em caso de empate, o desempate é realizado por uma comparação lexicográfica dos nomes.

Observe o Equilíbrio

Como $v_i|v_j$ ou $v_j|v_i$ em uma sequência equilibrada, se tomarmos o valor absoluto de cada elemento em ordem crescente, temos que ter como propriedade de que $v_k|v_{k+1}$. Tomar cuidado no caso específico em que v_k é 0 para evitar uma divisão por zero.