MC458A - Laboratório 3:
Ordenação de contas bancárias

1 Introdução

O Internet Banking do famoso banco Tamojunto realiza milhares de operações diariamente. Alguns clientes realizam mais operações que outros. Assim, algumas contas aparecem mais vezes em uma lista de operações. O banco Tamojunto contratou você para fazer um programa que ordena os números das contas correntes em ordem crescente. Ao lado de cada conta corrente, você deve imprimir o número de vezes que ela aparece na lista (o número da conta deve aparecer apenas uma vez). O número de cada conta corrente utiliza o sistema **hexadecimal** e tem o seguinte formato: dígitos de controle, um código de 8 dígitos do banco, 16 dígitos identificadores do titular da conta (escritos em grupos de quatro dígitos).
Exemplo: $2A$ $47F84112$ $2A45$ $32C4$ $6E98$ $118D$.

O banco quer que a sua implementação seja **eficiente** e assim ele exige que você use o Radix sort para resolver o problema.

Observação: você deve implementar o Radix Sort, caso contrário o banco não lhe pagará nada e sua nota do Lab será 0 **(ZERO). Além disso, você NÃO pode usar qualquer tipo de função ou estrutura de ordenação pronta como** `qsort()`, `stl::map`, `insort`, `heap`, **etc.**

2 Especificação de entrada e saída

A primeira linha da entrada contém um inteiro n ($5 \leq n \leq 30000$) que representa o número de operações realizadas. Então, se segue n linhas, com a i-ésima linha contendo o número da conta c_i que realizou a i-ésima operação. Cada conta c_i está no formato:

XX XXXXXXXX XXXX XXXX XXXX XXXX,

onde cada X representa um dígito **hexadecimal**, ou seja, $0 \leq X \leq$ F.

Na saída deverá ser impresso um inteiro k, que representa o total de contas distintas da entrada. Em seguida, deverá ser impresso k linhas com as contas distintas ordenadas em ordem crescente e suas respectivas frequências. Ou seja, a j-ésima linha deverá conter o número da j-ésima menor conta c_j, seguida da quantidade de operações realizadas por c_j. Utilize um espaço

para separar esses dois números. Além disso, a frequência da conta c_j deve ser dada no sistema **decimal**.

Observação: Note que os dígitos de 0 até 9 precedem as letras de A até F na tabela ASCII, logo podemos ordenar números hexadecimais com mesma quantidade de dígitos por meio da ordenação lexicográfica.

Exemplo:

Entrada	Saída
5 00 00000000 0000 0000 0000 0000 A0 00000000 0000 0000 0000 0000 00 20000000 0000 0000 0000 0000 00 00000000 0000 0000 0000 0000 A0 00000000 0000 0000 0000 0000	3 00 00000000 0000 0000 0000 0000 2 00 20000000 0000 0000 0000 0000 1 A0 00000000 0000 0000 0000 0000 2

Entrada	Saída
5 90 00000000 0000 0000 0000 0000 AB 90000000 0000 0000 0000 0000 AB 00000000 0000 0000 0000 0200 FA 00000000 0000 0000 0000 0000 10 00000000 0000 0000 0000 0000	5 10 00000000 0000 0000 0000 0000 1 90 00000000 0000 0000 0000 0000 1 AB 00000000 0000 0000 0000 0200 1 AB 90000000 0000 0000 0000 0000 1 FA 00000000 0000 0000 0000 0000 1

3 Implementação e Submissão

- A solução deverá ser implementada em C, C++ ou Python 3. Só é permitido o uso de bibliotecas padrão e de flags/diretivas de otimização.
- O programa deve ser submetido no SuSy, com o nome principal **t3** (por exemplo, t3.c).
- O número máximo de submissões é 20.
- A tarefa contém 10 testes abertos e 10 testes fechados. A nota será proporcional ao número de acertos nos testes fechados.

A solução pode ser submetida até o dia 15/05/24 às 8h.