# Estrutura de Dados II - 2020/EARTE

Trabalho Prático T2

29 de outubro de 2020

Leia atentamente **todo** esse documento de especificação. Certifique-se de que você entendeu tudo que está escrito aqui. Havendo dúvidas ou problemas, fale com o professor o quanto antes. As dúvidas podem ser sanadas usando o fórum de dúvidas do AVA.

## 1 Objetivo

O objetivo deste trabalho é implementar um algoritmo junção utilizando ordenação externa.

## 2 O problema de Cruzamento de bases de dados

Uma tarefa comum no processo de coleta e análise de dados é o de cruzamento de bases de dados distintas. Mas o que isso significa? Vamos entender a tarefa por meio de um exemplo prático.
Suponha que você seja uma analista de dados do governo e que sua tarefa seja identificar fraudes em declarações de imposto de renda. Você tem em suas mão duas bases de dados (e esse cenário é 100% hipotético): a primeira contendo as declarações de imposto de renda dos contribuintes; e uma segunda, contendo dados de gastos com cartão de crédito de todos os cidadãos. Seu objetivo é encontrar discrepâncias entre as duas bases, i.e., pessoas físicas com renda e gastos no cartão incompatíveis. No entanto, antes de conseguir realizar tal tarefa, **você precisa mapear (usando por exemplo o CPF) os contribuintes da primeira base de dados com portadores de cartão da segunda**. Após realizar essa associação, você poderá, para cada CPF, verificar a compatibilidade entre declaração e gastos...
Uma vez que essas bases de dados podem ser gigantescas, surge o desafio: como podemos mapear os dados de uma base com os da outra? Neste trabalho, sua tarefa será resolver este desafio.

### 2.1 O problema

Em Banco de Dados, o desafio descrito acima está relacionado ao conceito de *junção* (mas não nos ateremos aos detalhes técnicos da definição aqui). Esse problema fica mais interessante (e complicado) quando os arquivos têm muitos campos, são muito grandes e temos interesse em realizar a junção com base em vários campos. O problema a ser resolvido é formalmente descrito a seguir.
Um arquivo é uma coleção de $N$ linhas (sendo $N$ possivelmente muito grande). Cada linha do arquivo possui $K$ *strings* separadas pelo caractere '`,`' (vírgula), as quais são indexadas a partir do 0. A memória principal do computador só é capaz de armazenar $M$ linhas do arquivo.
A junção de dois arquivos pode ser feita por um ou mais campos (no exemplo da seção anterior, foi escolhido um único campo, o CPF), os quais devem ser especificados por uma lista de números inteiros, $L_1$ (para o primeiro arquivo) e $L_2$ (para o segundo arquivo), onde o número de elementos em $L_1$ deve ser o mesmo de $L_2$.
A saída deve ser um novo arquivo. As linhas do novo arquivo são formadas pelas linhas dos dois arquivos de entrada, mas somente aquelas que tenham os campos dados por $L_1$ do primeiro arquivo iguais as que tenham os campos dados por $L_2$ do segundo arquivo.
Se o primeiro arquivo possuir $K_1$ campos, o segundo arquivo $K_2$ campos e a junção for feita com base em $l$ campos, então o novo arquivo terá $K_1$ + $K_2$ – $l$ campos, dados por:

- os $l$ campos usados na junção em ordem;
- os $K_1$–$l$ campos do primeiro arquivo que não estiverem na lista de junção, na mesma ordem que estavam originalmente no arquivo;

- os $K_2 - l$ campos do segundo arquivo que não estiverem na lista de junção, na mesma ordem que estavam originalmente no arquivo.

A ordenação das linhas da saída deve respeitar a ordem dos atributos especificados na lista de campos de junção. Em outras palavras, as linhas da saída estarão primeiramente ordenadas pelo primeiro atributo da lista, depois pelo segundo e assim sucessivamente.
Observação: todos os campos deverão ser tratados como *strings*, de forma que os algoritmos de ordenação devem trabalhar com *strings* (nesse caso, a *string* "`10`" é menor que a *string* "`2`").
Observação 2: as chaves definidas pelas listas $L_1$ e $L_2$ serão sempre únicas! Por exemplo, não há duas pessoas com o mesmo número de CPF.

## 2.2 Algoritmo

O problema de junção de dois arquivos pode ser resolvido de uma forma muito simples:

1. ordene o primeiro arquivo de acordo com os campos em $L_1$ (veja que ordenação externa pode ser necessária)
2. ordene o segundo arquivo de acordo com os campos em $L_2$ (novamente, você pode precisar de ordenação externa);
3. faça a intercalação do primeiro arquivo com o segundo arquivo, mantendo apenas as linhas que tenham chaves em $L_1$ e $L_2$, respectivamente, iguais.

## 2.3 Exemplo ilustrativo

A definição apresentada acima pode parecer abstrata, mas a tarefa a ser conduzida é simples. Considere o arquivo *arquivo1.txt* abaixo. Nesse caso, tem-se $N_1 = 10$ e $K_1 = 4$, uma vez que o arquivo tem 10 linhas e 4 colunas.

```
1,5,4,3
2,5,4,3
3,4,4,3
4,4,3,3
5,3,3,2
6,3,3,2
7,2,2,2
8,2,2,2
9,1,2,1
10,1,1,1
```

Similarmente, considere o arquivo *arquivo2.txt* abaixo. Nesse caso, tem-se $N_2 = 5$ e $K_2 = 3$, uma vez que o arquivo tem 5 linhas e 3 colunas.

```
1,a,10
1,b,1
4,b,3
4,b,5
5,5,2
```

Suponha agora que queiramos fazer a junção desses dois arquivos de acordo com dois campos, com $L_1 = [1, 0]$ e $L_2 = [0, 2]$. Mais especificamente:

1. queremos todas as linhas de *arquivo1.txt* e *arquivo2.txt* tais que o campo 1 do primeiro seja igual ao campo 0 do segundo **e** o campo 0 do primeiro seja igual ao campo 2 do segundo;
2. a saída deverá ter 5 colunas: As duas primeiras referentes aos atributos de junção; as próximas duas são referentes às colunas 2 e 3 do primeiro arquivo; e última será referente à coluna 1 do segundo arquivo.
3. a saída deve estar ordenada de acordo com os atributos de junção.

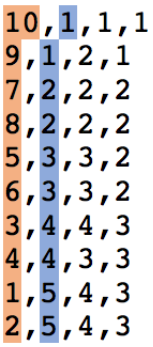


Figura 1: *arquivo1.txt* após ordenação de acordo com as colunas 1 (azul) e 0 (laranja).

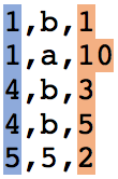


Figura 2: *arquivo2.txt* após ordenação de acordo com as colunas 0 (azul) e 2 (laranja).

Após ordenar o primeiro arquivo de acordo com os elementos das colunas 1 e 0, tem-se o resultado na Figura 1. Similarmente, após ordenar o segundo arquivo de acordo com os elementos das colunas 0 e 2, tem-se o resultado na Figura 2.
Uma vez que os dois arquivos estão ordenados, identificar os elementos comuns passa ser uma tarefa trivial, por meio de uma operação de intercalação, como exemplificado na Figure 3. Dessa forma, o conteúdo do arquivo de saída será o da Figura 4.

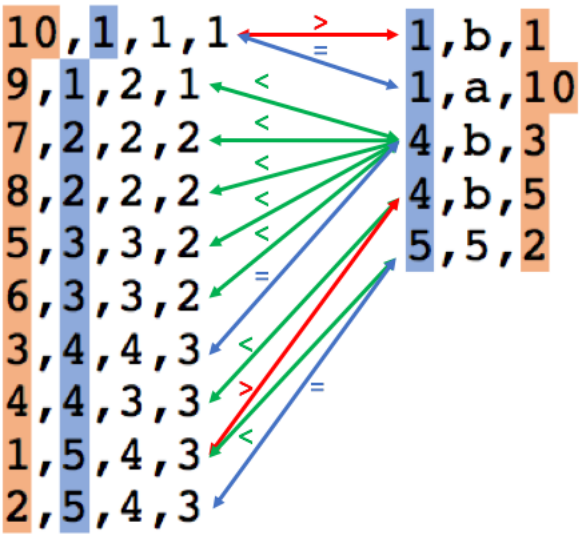


Figura 3: Uma vez que os arquivos estão ordenados, realizar a junção é uma tarefa simples. Uma a uma, linhas dos dois arquivos são lidas. Se elas forem iguais, de acordo com os atributos de junção, deve-se salvar o resultado e ler as próximas linhas dos dois arquivos. Se a linha do primeiro arquivo for "maior" que a linha do segundo, leia uma nova linha do segundo. Caso contrário, leia uma nova linha do primeiro. Quando um arquivo acabar, pare o processo.

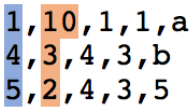


Figura 4: Conteúdo do arquivo de saída. Cores meramente ilustrativas.

### 2.4 Sobre a ordenação externa

A parte mais complicada deste trabalho é a tarefa de ordenação dos arquivos. Uma vez que eles podem ser muito grandes, um método de ordenação externa deve ser usado. Para esta tarefa, vocês devem implementar o método apresentado em sala, i.e., o método de **intercalação balanceada.**
Como a maioria de nós não possui um computador com vários dispositivos de armazenamento (e.g., discos e fitas), vamos simular a presença de tais dispositivos por meio de arquivos temporários. Dado o parâmetro $P$, vocês poderão criar $2P$ arquivos temporários para os auxiliar na ordenação.
Por exemplo, se $P = 2$, então vocês poderão usar os arquivos *0.txt*, *1.txt*, *2.txt* e *3.txt* como auxiliares. Na primeira passada sobre os dados, vocês deverão ler do arquivo de entrada e salvar os resultados em *2.txt* e *3.txt*. Na segunda passada, deverão ler de *2.txt* e *3.txt* e escrever resultados em *0.txt* e *1.txt*...

## 3 Entrada e Saída

Os padrões de entrada e saída devem seguir as definições e exemplos dados previamente neste texto.

### 3.1 Execução do trabalho

Para testar seu trabalho, o professor executará comandos seguindo o seguinte padrão.

```
tar -xzvf <nome_arquivo>.tar.gz
make
./trab2 P M i1,i2,...,ik j1,j2,...,jk <arquivo1> <arquivo2> <arquivo3>
```

Onde:

- `P` é o parâmetro relacionado ao número de dispositivos. Vocês poderão criar $2P$ arquivos auxiliares;

- `M` é o número máximo de registros (linhas) que cabem em memória principal;
- `i1,i2,...,ik` é a lista $L_1$, ou seja, os campos de junção do primeiro arquivo, separados por vírgula;
- `j1,j2,...,jk` é a lista $L_2$, ou seja, os campos de junção do segundo arquivo, separados por vírgula;
- `<arquivo1>` é o nome (ou caminho) do primeiro arquivo
- `<arquivo2>` é o nome (ou caminho) do segundo arquivo
- `<arquivo3>` é o nome (ou caminho) do arquivo de saída, com o resultado da junção.

Por exemplo,

```
./trab2 3 1000000 10,0 7,11 fileOne fileTwo fileOut
```

significa que queremos fazer a junção dos arquivos *fileOne* e *fileTwo*. Durante a ordenação externa, é permitido manter até um milhão de registros em memória principal. O resultado da junção deverá estar no arquivo *fieOut*. A junção será feita de forma que o campo 10 de *fileOne* seja igual ao campo 7 de *fileTwo* e o campo 0 de *fileOne* seja igual ao campo 11 de *fileTwo*. Por fim, as linhas do arquivo *fileOut* estarão ordenadas primeiramente pelo campo 10 e depois pelo campo 0 de *fileOne* (equivalentemente, pelos campos 7 e 11 de *fileTwo*).
Observação 1: assim como nos arquivos de entrada, os campos dos registros (linhas) do arquivo de saída devem estar separados por uma '`,`' (vírgula).
Observação 2: o caractere '`,`' será usado unicamente para separação de campos.
Observação 3: você pode assumir que a linha de comando estará sempre formatada de forma correta e seguindo o padrão acima; que os dois arquivos de entrada existem; que você tem permissão para criar o arquivo de saída; e que $M \geq P$.
É extremamente importante que vocês sigam esse padrão. Seu programa não deve solicitar a entrada de nenhum valor e também não deve imprimir nada na tela.

# 4 Detalhes de implementação

A seguir, alguns detalhes, comentários e dicas sobre a implementação. Muita atenção aos usuários do Sistema Operacional Windows.

- o trabalho deve ser implementado em C. A versão do C a ser considerada é a presente no Sistema Operacional Ubuntu 18.04.
- o caractere de nova linha será o `\n`.
- Seu programa deve ser, obrigatoriamente, compilado com o utilitário `make`. Crie um arquivo `Makefile` que gera como executável para o seu programa um arquivo de nome `trab2`.
- A linguagem C possui algumas funções que podem ser úteis na leitura dos arquivos. Em especial, sugere-se o estudo cuidadoso das funções `getline` e `strtok`.
- Ao longo do desenvolvimento do trabalho, certifique-se que o seu código não está vazando memória testando-o com o `valgrind`. Não espere terminar o código para usar o `valgrind`, incorpore-o no seu ciclo de desenvolvimento. Ele é uma ferramenta excelente para se detectar erros sutis de acesso à memória que são muito comuns em C. Idealmente o seu programa deve sempre executar sem nenhum erro no `valgrind`.
- Não é necessária nenhuma estrutura de dados muito elaborada para o desenvolvimento deste trabalho. Todas as estruturas que você vai precisar foram discutidas em aula ou no laboratório. Veja os códigos disponibilizados pelo professor para ter ideias. Prefira estruturas simples a coisas muito complexas. Pense bem sobre as suas estruturas e algoritmos antes de implementá-los: quanto mais tempo projetando adequadamente, menos tempo depurando o código depois.

# 5 Regras para desenvolvimento e entrega do trabalho

- **Data da Entrega**: O trabalho deve ser entregue até as 23:59h do dia 19/11/2020. Não serão aceitos trabalhos após essa data.
- **Grupo**: O trabalho pode ser feito em grupos de até três pessoas. Lembrando que duas pessoas que estavam no mesmo grupo no trabalho 1 não poderão estar no mesmo grupo no trabalho 2. Além disso, duas pessoas do mesmo grupo no trabalho 1 ou no trabalho 2 não poderão estar no mesmo grupo nos próximos trabalhos.
- **Como entregar**: Pela atividade criada no AVA. Envie um arquivo compactado, no formato `.tar.gz`, com todo o seu trabalho. A sua submissão deve incluir todos os arquivos de código e um `Makefile`, como especificado anteriormente. **Somente uma pessoa do grupo deve enviar o trabalho no AVA. Coloque a matrícula de todos integrantes do grupo (separadas por vírgula) no nome do arquivo do trabalho.**
- **Recomendações**: Modularize o seu código adequadamente. Crie códigos claros e organizados. Utilize um estilo de programação consistente. Comente o seu código extensivamente. Não deixe para começar o trabalho na última hora.

# 6 Relatório de resultados

Esse trabalho não demandará entrega de relatório.

# 7 Avaliação

- O uso da primitiva **goto** e variáveis globais não são permitidos.
- Assim como especificado no plano de ensino, o trabalho vale 10 pontos.
- A parte de implementação será avaliada de acordo com a fração e tipos de casos de teste que seu trabalho for capaz de resolver de forma correta. Casos *pequenos* e *médios* (5 pontos) serão utilizados para aferir se seu trabalho está correto. Casos *grandes* (3 pontos) serão utilizados para testar a eficiência do seu trabalho. Casos *muito grandes* (2 pontos) serão utilizados para testar se seu trabalho foi desenvolvido com muito cuidado e tendo eficiência máxima como objetivo. Todos os casos de teste serão projetados para serem executados em poucos minutos (no máximo 10) em uma máquina com 16GB de RAM.
- Trabalhos com erros de compilação receberão nota zero.
- Trabalhos que gerem *segmentation fault* para algum dos casos de teste disponibilizados no AVA serão severamente penalizados na nota.
- Trabalhos com *memory leak* (vazamento de memória) sofrerão desconto na nota.
- Organização do código e comentários valem nota. Trabalhos confusos e sem explicação sofrerão desconto na nota.
- Caso seja detectado plágio (entre alunos ou da Internet), todos os envolvidos receberão nota zero. Caso as pessoas envolvidas em suspeita de plágio discordem da nota, amplo direito de argumentação e defesa será concedido. Neste caso, as regras estabelecidas nas resoluções da UFES serão seguidas.
- A critério do professor, poderão ser realizadas entrevistas com os alunos, sobre o conteúdo do trabalho entregue. Caso algum aluno seja convocado para uma entrevista, a nota do trabalho será dependente do desempenho na entrevista. (Vide item sobre plágio, acima.)