Trabalho Prático 3 - Busca de Passagens Aéreas

1 Descrição do problema

A Xulambs Tour deseja implementar um sistema eficiente e flexível para buscar passagens aéreas. A proposta é que cada usuário possa especificar filtros através de expressões lógicas e definir o critério de ordenação dos vôos filtrados.
O sistema deve ser construído a partir de uma lista de vôos que é atualizada frequentemente por outro componente do sistema da Xulambs Tour. Você pode assumir que sempre vai ter uma base atualizada para executar as consultas[1]. e essa base contem as seguintes informações sobre cada vôo:

1. Origem
2. Destino
3. Preço
4. Assentos disponíveis
5. Data-hora de partida
6. Data-hora de chegada
7. Número de paradas

A partir dessas informações, deseja-se implementar a busca de vôos utilizando consultas. Uma consulta tem três componentes:

1. número máximo de vôos a serem retornados
2. critério de ordenação das respostas, que é sempre um trigrama composto pelas letras `p` (preço), `d` (duração) e `s` (paradas – *stops*). Exemplos de trigramas são `pds`, `dsp` e `psd`.
3. expressão lógica da filtragem, que é uma expressão lógica conjuntiva delimitada por parênteses, onde cada predicado especifica os valores aceitos para um atributo.

A seguir apresentamos um exemplo de uma consulta:

```
10 pds (((org==DEN)&&(dst==ORD))&&((prc<=500)&&(dur<=8400)))
```

onde `10` é o número máximo de vôos a serem retornados, `pds` indica que os vôos retornados devem ser ordenados por preço, duração e número de paradas, e estamos interessados em vôos que partem de `DEN` para `ORD`, custando menos que 500 e durando menos que 2 horas e 20 minutos (8400 segundos).
Para fins deste trabalho, você vai receber um arquivo único, contendo inicialmente a lista de vôos, seguida das consultas a serem respondidas. A primeira linha do arquivo informa n, que é o número de vôos constantes da lista. As próximas n linhas do arquivo são a lista propriamente dita, que não tem, necessariamente, qualquer ordenação garantida, mas pode, por exemplo, ser ordenada por data-hora de partida. A seguir temos c, que é o

```
1000
DEN ORD  68.60 9 2022-11-01T17:54:00.000-06:00 2022-11-01T21:12:00.000-05:00 0
CLT ATL 273.99 4 2022-04-17T00:30:00.000-06:00 2022-04-17T05:22:00.000-04:00 0
DFW LGA 579.01 4 2022-04-17T00:30:00.000-06:00 2022-04-17T10:00:00.000-04:00 1
EWR PHL 614.01 4 2022-04-17T00:30:00.000-06:00 2022-04-17T10:18:00.000-04:00 1
DEN ATL 296.61 9 2022-04-17T00:35:00.000-06:00 2022-04-17T05:15:00.000-04:00 0
DEN ORD  68.60 9 2022-11-02T17:54:00.000-06:00 2022-11-02T21:12:00.000-05:00 0
DTW BOS 322.60 9 2022-04-17T00:35:00.000-06:00 2022-04-17T09:00:00.000-04:00 1
ATL CLT 305.61 9 2022-04-17T00:35:00.000-06:00 2022-04-17T08:34:00.000-04:00 1
LGA DFW 211.60 9 2022-04-17T00:35:00.000-06:00 2022-04-17T09:43:00.000-05:00 1
DEN DTW 305.61 8 2022-04-17T00:35:00.000-06:00 2022-04-17T11:02:00.000-04:00 1
BOS EWR 417.60 9 2022-04-17T00:35:00.000-06:00 2022-04-17T09:21:00.000-04:00 1
...
1
10 pds (((org==DEN)&&(dst==ORD))&&((prc<=500)&&(dur<=8400)))
```

Figura 1: Exemplo de arquivo de entrada

```
10 pds (((org==DEN)&&(dst==ORD))&&((prc<=500)&&(dur<=8400)))
DEN ORD 68.60 9 2022-11-01T17:54:00.000-06:00 2022-11-01T21:12:00.000-05:00 0
DEN ORD 68.60 9 2022-11-01T17:55:00.000-06:00 2022-11-01T21:13:00.000-05:00 0
DEN ORD 68.60 9 2022-11-02T17:54:00.000-06:00 2022-11-02T21:12:00.000-05:00 0
DEN ORD 68.60 9 2022-11-02T17:55:00.000-06:00 2022-11-02T21:13:00.000-05:00 0
DEN ORD 68.60 9 2022-11-08T17:55:00.000-07:00 2022-11-08T21:13:00.000-06:00 0
DEN ORD 68.60 9 2022-11-09T17:55:00.000-07:00 2022-11-09T21:13:00.000-06:00 0
DEN ORD 68.60 9 2022-11-15T17:55:00.000-07:00 2022-11-15T21:13:00.000-06:00 0
DEN ORD 68.60 9 2022-11-16T17:55:00.000-07:00 2022-11-16T21:13:00.000-06:00 0
DEN ORD 78.60 9 2022-11-04T17:54:00.000-06:00 2022-11-04T21:12:00.000-05:00 0
DEN ORD 78.60 9 2022-11-04T17:55:00.000-06:00 2022-11-04T21:13:00.000-05:00 0
```

Figura 2: Exemplo de arquivo de saída

número de consultas, seguido por c linhas, cada linha contendo uma consulta no formato indicado. A Figura 1 apresenta uma amostra do arquivo de entrada.
A saída do sistema de busca de passagens é, para cada consulta do arquivo de entrada, a lista contendo o número especificados de vôos filtrados, ordenados pelo critério escolhido, um por linha, em formato idêntico à entrada. Por exemplo, o arquivo de entrada da Figura 1 geraria como saída o arquivo apresentado na Figura 2.

2 Arquitetura

Nesta seção vamos detalhar a especificação do sistema a ser implementado neste trabalho prático.

2.1 Entrada e Saída

O sistema a ser implementado recebe um arquivo de entrada, pela entrada padrão (`stdin`), no formato apresentado na Figura 1. A partir dessa entrada, que deve ser lida uma única

[1] Atualizar a base não faz parte do trabalho, mas as estruturas de dados a serem utilizadas tem que ser compatíveis com atualizações periódicas.

Var	Descrição
`org`	Origem do vôo
`dst`	Destino do vôo
`prc`	Preço de uma passagem
`sea`	Número de assentos disponíveis
`dep`	Data-hora de partida
`arr`	Data-hora de chegada
`sto`	Número de paradas
`dur`	Duração total do vôo

Tabela 1: Variáveis que podem ser usadas nos predicados

vez, ele gera as respostas para as consultas na saída padrão (`stdout`), seguindo o formato apresentado na Figura 2.

2.2 Carregamento da lista de vôos

A lista de vôos deve ser armazenada em memória interna em um vetor de estruturas, uma por vôo, que contém, além dos sete atributos do arquivo de entrada, a duração total do vôo. A partir do armazenamento, o vôo passa a ser referenciado pelo índice do vetor, que, para n vôos, vai estar na faixa $0 \ldots n-1$. Para fins de armazenamento, você pode converter data-hora em segundos.
Armazenada a entrada, você deve construir 8 índices, um por critério de consulta. Os critérios de consulta são apresentados na Tabela 1. Cada índice é uma árvore à sua escolha, onde a chave é uma variável que pode ocorrer na filtragem e possui uma lista de vôos que estão associados aos atributos. Por exemplo, os índices dos atributos `org` e `dst` são apresentados nas Figuras 3 e 4, onde cada nó da árvore contem a chave e os índices dos vôos.
Recomenda-se a utilização de árvores balanceadas ou de mecanismos que maximizem o balanceamento, de forma que não haja risco do índice degradar em termos de desempenho por uma inserção de valores de atributos ordenados.

2.3 Análise de Consultas

A análise da consulta compreende duas ações: construção da representação interna da consulta e planejamento da resposta.
A construção da representação interna da consulta pressupõe a definição da representação interna, que é uma atividade que faz parte do trabalho. Uma representação típica é uma árvore sintática, que é uma árvore binária onde os operadores lógicos e relacionais são nós internos e as variáveis e os seus valores são folhas. Uma vantagem dessa representação é que ela nativamente dispensa o uso dos parênteses. A avaliação de uma expressão lógico-relacional pode ser realizada por um caminhamento pós-ordem na árvore sintática, armazenando resultados parciais numa pilha.
Para fins de simplificação, a versão base do trabalho vai considerar apenas consultas conjuntivas (apenas `&&`), ou seja, não há disjunções nas expressões, e sempre delimitado por parênteses em todos os níveis.
À semelhança dos índices, cada nó interno da árvore binária vai, eventualmente, conter a lista dos vôos que satisfazem o predicado. Em particular, cada predicado relacional

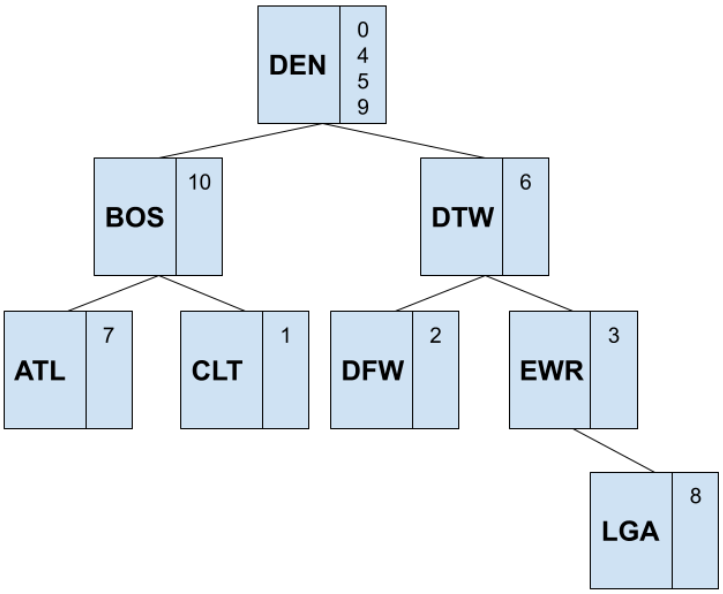

Figura 3: Índice para atributo Org

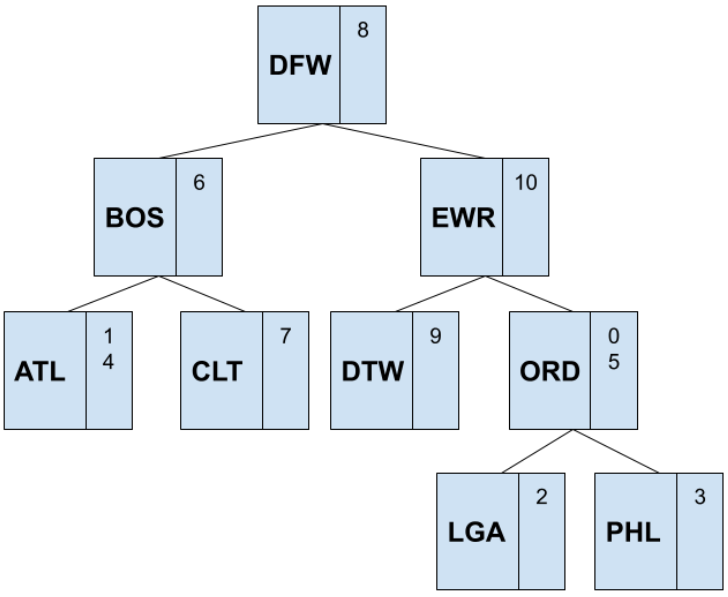

Figura 4: Índice para atributo DST

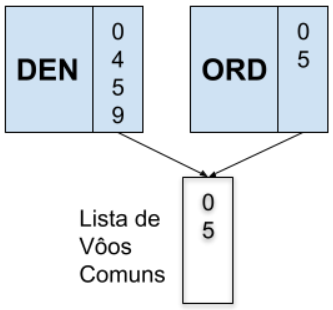

Figura 5: Exemplo de junção de listas de vôos

(p.ex., `dst==ORD`) vai demandar acesso ao índice da variável para recuperar a lista de vôos que satisfaz o predicado. Se o predicado relacional for uma faixa (p.ex., `dur<=8400`), é necessária a identificação de todas as listas de vôos que satisfazem os predicados, ou seja, todas as listas cujo valor de duração seja menor ou igual a 8400 segundos. Uma vez identificadas, essas listas tem que ser intercaladas para que tenhamos a lista dos vôos que satisfazem ao predicado. A seguir, para cada operador lógico (que é apenas `&&` no caso base do TP) temos que realizar a junção das listas constantes da conjunção, o que é ilustrado na Figura 5. O resultado do processo de avaliação é uma lista de vôos que satisfazem à consulta como um todo.

2.4 Gerador de respostas

Identificados todos os vôos que satisfazem à filtragem, eles são ordenados de acordo como critério de ordenação e, finalmente, selecionados os vôos a serem retornados. A Figura 2 apresenta um exemplo de saída, como mencionado.

3 Pontos Extra

Os pontos extra tem por objetivo aumentar a flexibilidade e poder expressivo das consultas, por exemplo:

- Tornar as consultas mais genéricas, permitindo os operadores lógicos ou (`||`) e negação (`!`).
- Auxiliar o usuário de forma pró-ativa, fazendo recomendações de aperfeiçoamento da consulta, de forma a torná-la mais específica e retornar um número menor de vôos.
- Permitir que vôos sejam removidos e acrescentados durante a execução, ou seja, a partir da entrada inicial e de um lote de consultas, podem ser informadas no arquivo de entrada alterações na lista de vôos, após o que podemos receber mais consultas e avançar alternando alterações na lista de vôos e consultas.

4 Como será feita a entrega

4.1 Submissão

A entrega do TP3 compreende duas submissões:

VPL TP3: Submissão do código a ser submetido até **03/02/25, 7:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). **Não serão aceitas submissões em atraso.** Detalhes sobre a submissão do código são apresentados na Seção 4.3.

Relatório TP3: Arquivo PDF contendo a documentação do TP, assim como a avaliação experimental, conforme instruções, a ser submetido até **03/02/25, 7:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). **Não serão aceitas submissões em atraso.** Detalhes sobre a submissão de relatório são apresentados na Seção 4.2.

4.2 Documentação

A documentação do trabalho deve ser entregue em formato **PDF** e também **DEVE** seguir o modelo de relatório que será postado no `minha.ufmg`. Além disso, a documentação deve conter **TODOS** os itens descritos a seguir **NA ORDEM** em que são apresentados:

1. **Capa:** Título, nome, e matrícula.
2. **Introdução:** Contém a apresentação do contexto, problema, e qual solução será empregada.
3. **Método:** Descrição da implementação, detalhando as estruturas de dados, tipos abstratos de dados (ou classes) e funções (ou métodos) implementados.
4. **Análise de Complexidade:** Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
5. **Estratégias de Robustez:** Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
6. **Análise Experimental:** Apresenta os experimentos realizados em termos de desempenho computacional[2], assim como as análises dos resultados.
7. **Conclusões:** A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.
8. **Bibliografia:** Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.
9. Número máximo de páginas incluindo a capa: 10

A documentação deve conter a descrição do seu trabalho em termos funcionais, dando foco nos algoritmos, estruturas de dados e decisões de implementação importantes durante o desenvolvimento.

[2]Para este trabalho não é necessário analisar a localidade de referência.

Evite a descrição literal do código-fonte na documentação do trabalho.

Dica: Sua documentação deve ser clara o suficiente para que uma pessoa (da área de Computação ou não) consiga ler, entender o problema tratado e como foi feita a solução.

A documentação deverá ser entregue como uma atividade separada designada para tal no minha.ufmg. A entrega deve ser um arquivo `.pdf`, nomeado `nome_sobrenome_matricula.pdf`, onde nome, sobrenome e matrícula devem ser substituídos por suas informações pessoais.

4.3 Código

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é terminantemente vetado. Você DEVE utilizar a estrutura de projeto abaixo junto ao Makefile:

```
– TP
    |– src
    |– bin
    |– obj
    |– include
    Makefile
```

A pasta **TP** é a raiz do projeto; **src** deve armazenar arquivos de código (*.c, *.cpp, ou *.cc); a pasta **include**, os cabeçalhos (headers) do projeto, com extensão *.h, por fim as pastas **bin** e **obj** devem estar vazias. O Makefile deve estar na raiz do projeto. A execução do Makefile deve gerar os códigos objeto *.o no diretório **obj** e o executável do TP no diretório **bin**. O arquivo executável **DEVE** se chamar **tp3.out** e deve estar localizado na pasta **bin**. O código será compilado com o comando:

```
make all
```

O seu código será avaliado através de uma **VPL** que será disponibilizada no moodle. Você também terá à disposição uma VPL de testes para verificar se a formatação da sua saída está de acordo com a requisitada. A VPL de testes não vale pontos e não conta como trabalho entregue. Um pdf com instruções de como enviar seu trabalho para que ele seja compilado corretamente estará disponível no Moodle.

5 Avaliação

- Corretude na execução dos casos de teste - (20% da nota total)
- Indentação, comentários do código fonte e uso de boas práticas - (10% da nota total)
- Conteúdo segundo modelo proposto na seção **Documentação**, com as seções detalhadas corretamente - (20% da nota total)
- Definição e implementação das estruturas de dados e funções - (10% da nota total)
- Apresentação da análise de complexidade das implementações - (10% da nota total)
- Análise experimental - (25% da nota total)
- Aderência completa às instruções de entrega - (5% da nota total)

Se o programa submetido **não compilar**[3], seu trabalho não será avaliado e sua nota será **0**. Trabalhos não poderão ser entregues com atraso.

6 Considerações finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega está tão distante quanto jamais estará.
2. Leia atentamente o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
4. Plágio é crime. Trabalhos onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas (em relação a todos os envolvidos). Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na seção de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos.**

7 FAQ (*Frequently asked Questions*)

1. Posso utilizar qualquer versão do C++? NÃO, o corretor da VPL utiliza C++11.
2. Posso fazer o trabalho no Windows, Linux, ou MacOS? SIM, porém lembre-se que a correção é feita sob o sistema Linux, então certifique-se que seu trabalho está funcional em Linux.
3. Posso utilizar alguma estrutura de dados do C++ do tipo Queue, Stack, Vector, List, etc? NÃO.
4. Posso utilizar smart pointers? NÃO.
5. Posso utilizar o tipo String? SIM.
6. Posso utilizar o tipo String para simular minhas estruturas de dados? NÃO.
7. Posso utilizar alguma biblioteca para tratar exceções? SIM.
8. Posso utilizar alguma biblioteca para gerenciar memória? SIM.
9. As análises e apresentação dos resultados são importantes na documentação? SIM.
10. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? NÃO.
11. Posso fazer o trabalho em dupla ou em grupo? NÃO.
12. Posso trocar informações com os colegas sobre os fundamentos teóricos do trabalho? SIM.
13. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? SIM.

[3]Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.