Pontifícia Universidade Católica de Minas Gerais - campus Betim
Sistemas de Informação - Fundamentos de Projeto e Análise de Algoritmos
Profa. Amália Vasconcelos

PUC Minas

Trabalho prático - 20 pontos

Data limite de entrega: 25 de junho de 2025 Data da apresentação: 26 de junho de 2025

Regras:

- Não é permitido fazer o trabalho de forma individual;
- Deve ser desenvolvido em grupo de 7 ou 8 alunos(as);
- Não serão aceitos trabalhos atrasados;
- Não serão aceitos trabalhos enviados por e-mail;
- Cópias, de qualquer natureza, serão penalizadas com nota zero.

Entrega: apenas um(a) dos componentes do grupo deverá submeter no Canvas um arquivo **.zip** contendo os arquivos relacionados ao código, as respostas às perguntas da última página deste roteiro e o arquivo da apresentação (.pdf, .pptx).

Critérios de avaliação:

- A correta interpretação do roteiro faz parte da avaliação;
- Comentários no código:
  - Cabeçalho com as informações sobre os autores, versão, data, etc;
  - Explicar o que foi feito em cada parte do código. Obs: a linguagem de programação pode ser escolhida pelo grupo.
- Validação das entradas;
- Legibilidade do código;
- Interface gráfica:
  - Os grupos que fizerem interface gráfica com usuário (GUI) serão bonificados com 3 pontos extras.
- Apresentação do programa:
  - Limitada a 15 minutos por grupo;
  - O formato de apresentação é livre, mas deve ter slides que contenham os requisitos mínimos de um trabalho acadêmico (introdução, metodologia, considerações finais, referências, etc.);
  - Durante a apresentação é indispensável demonstrar o programa funcionando. Caso o desenvolvimento tenha sido feito de forma incompleta, apontar os requisitos do roteiro não cumpridos e explicitar as dificuldades encontradas pelo grupo;
  - Um(a) aluno(a) será sorteado(a) no dia da apresentação para apresentar o trabalho;
  - Será avaliado o entendimento individual dos componentes do grupo durante a fase de perguntas.

Roteiro:

Descobrindo padrões — a jornada para sincronizar dados complexos

Problema

Em ambientes de análise de dados, é comum que diferentes equipes coletem registros de eventos ao longo do tempo, cada uma com suas próprias fontes e formatos. Helena e Marcus são analistas responsáveis por interpretar grandes volumes de dados gerados por sistemas complexos — como logs de servidores, transações financeiras ou registros de sensores em uma fábrica inteligente. Cada um possui uma sequência cronológica desses eventos, que podem variar em detalhes e quantidade, mas que, em essência, refletem a mesma operação ou fenômeno.

O desafio surge quando eles precisam sincronizar esses dados, ou seja, encontrar as subsequências comuns que se repetem nas duas fontes, respeitando a ordem dos eventos, para garantir que estão analisando os mesmos padrões subjacentes. Essas subsequências comuns são essenciais para validar hipóteses, detectar inconsistências, melhorar a qualidade dos dados e, finalmente, extrair insights confiáveis que podem impactar decisões estratégicas.

Assim, encontrar todas as maiores subsequências comuns entre as sequências de Helena e Marcus é uma tarefa fundamental para garantir a integridade e a utilidade da análise conjunta dos dados. As letras usadas vão de 'a' a 'z', representando até 26 tipos diferentes de eventos.

Entrada

A primeira linha da entrada contém um número D (em que $D \leqslant 10$), que indica quantos conjuntos de dados serão processados. Cada conjunto de dados possui duas linhas:

- A primeira linha traz a sequência de eventos de Helena;
- A segunda linha traz a sequência de eventos de Marcus.

Cada sequência tem entre 1 e 80 letras minúsculas.

Saída

Para cada conjunto de dados, o programa deve imprimir todas as subsequências comuns mais longas, sem repetir nenhuma delas. As subsequências devem estar em ordem alfabética. É garantido que haverá pelo menos uma subsequência válida, e o número total de subsequências distintas não ultrapassará 1000. Entre os blocos de saída de diferentes conjuntos de dados, o programa deve imprimir uma linha em branco.

Exemplo

Entrada:
1
ijkijkii
ikjikji

Saída:
ijiji
ijiki
ijkji
ikiji
ikiki
ikjii
ikjki

Entrega

Para resolver o problema, o grupo deverá utilizar a técnica de programação dinâmica em conjunto com a de *backtracking*. Apenas um(a) integrante do grupo deverá realizar a entrega de um arquivo .zip contendo os quatro arquivos a seguir:

1. O primeiro arquivo contendo uma solução do problema utilizando somente programação dinâmica;
2. Um outro arquivo com a solução utilizando programação dinâmica e *backtracking*;
3. Um arquivo README com uma breve descrição da solução utilizada e respondendo às perguntas da próxima seção;
4. Por fim, o arquivo da apresentação de slides.

Perguntas e problemas para o README

O arquivo README deve possuir respostas para as seguintes perguntas:

1. Como a programação dinâmica foi aplicada na solução?
2. Por que o uso de *backtracking* é necessário neste problema?
3. Houve desafios na implementação? Quais? Como foram superados?
4. Qual é a complexidade da solução proposta? Faça o cálculo da ordem de complexidade passo a passo para:
    - a versão utilizando apenas programação dinâmica;
    - a versão que combina programação dinâmica com *backtracking*.
5. O que o grupo aprendeu ao resolver esse problema?