Exercício-Programa 0: Conceitos básicos

Entrega: 10/03/2019

Objetivo

O objetivo desse EP é o de familiarizá-lo com o sistema de submissão e auto correção.

Conhecimentos envolvidos

A solução desse EP envolve o conhecimento dos seguintes conceitos de programação:

- Leitura e escrita de arquivos de texto;
- Manipulação de caracteres e strings;
- Utilização de estruturas de dados básicas (dicionários, filas, . . .);
- Aplicação dos conceitos de busca cega estudados em sala.

Organização do EP

Você deverá implementar sua solução no esqueleto de código fornecido no arquivo `ep0.py`. Você pode criar novas funções, mas não deve modificar o protótipo das funções existentes.

As diferentes funcionalidades/objetivos estão agrupadas em 4 partes descritas a seguir. Especialmente para esse EP, não há qualquer restrição/orientação com respeito a ordem de implementação, ou seja, você pode implementar e testar as funções na ordem que quiser. IMPORTANTE: Você só deve editar o arquivo *ep0.py*.

Cada função no esqueleto está inicialmente implementada com o seguinte comando:

```
raise NotImplementedError
```

Esse comando visa informar ao programa de teste que a função não foi implementada. A primeira coisa ao escrever sua implementação para uma determinada função é remover a linha acima.

Parte 00: Integridade

Para esse EP você deve ter recebido um único arquivo compactado denominado `EP0MAC0425.zip` contendo todos os arquivos necessários à realização deste EP. Assim, sua primeira tarefa é verificar se esse arquivo possui a seguinte estrutura:

```
EP0MAC0425.zip
├── ep0.py
├── autograder.py
├── war_and_peace.txt
├── test_parentheses01.txt
├── test_parentheses02.txt
├── simple_corpus.txt
├── simple_parentheses1.txt
└── simple_parentheses2.txt
```

Parte 1: Frequências

Dado um corpus textual podemos encontrar a frequência relativa de ocorrências de uma palavra, ou de uma "frase" composta de duas palavras. Por exemplo, na frase:

> eu sei eu gosto muito de estudar
> e eu gosto muito de fazer EP

Neste corpus a frequência da palavra freq("eu") = 3/14, da palavra freq("sei") = 1/14 e da palavra freq("prova") = 0/14.

De maneira análoga, se considerarmos como "frase" a composição de duas palavras contíguas, podemos computar sua frequência como sendo: freq("eu gosto") = 2/13,

freq("eu sei") = 1/13 e freq("hoje EP") = 0/13.

Seu objetivo nessa parte é implementar duas funções:

- `compute_word_freq`: função que recebe um nome de arquivo e retorna um dicionário com a frequência relativa de cada palavra com pelo menos cinco letras sem distinguir caixa alta e caixa baixa.

- `compute_phrase_freq`: função que recebe um nome de arquivo e retorna um dicionário com a frequência relativa de cada frase formada por duas palavras com pelo menos cinco letras sem distinguir caixa alta e caixa baixa.

Você pode supor que o arquivo do corpus contém apenas palavras em codificação ASCII.

Parte 2: Parentização

Em diferentes áreas do conhecimento (matemática, programação, lógica, etc.) existe uma estrutura semântica para verificação se uma expressão pode ser considerada como "bem formada". Para efeitos desse EP você pode considerar que uma expressão é "bem formada" se a ordem de abertura e fechamento de chaves, colchetes e parentes é respeitada e se todos esses simbolos estão casados. (Cada "abre" tem seu respectivo "fecha" e vice-versa)

Nessa parte do EP você deve implementar a função `is_well_formed` que verifica se a estrutura de um arquivo de texto qualquer é "bem formada".

Seu programa receberá um nome de arquivo e deverá retornar `True` se o arquivo é "bem formado" e `False` caso contrário.

Considere que os únicos símbolos relevantes são chaves, colchetes e parenteses, ou seja, a string `#include stdio.h>` é bem formada apesar de seu programa gerar um erro de compilação. Além disso, seu programa só deve se importar com a "parentização". Assim, se um parenteses é aberto e não fecha dentro de um comentário de um código em C (por exemplo), seu programa deve retornar False. Ou seja, especificidades de estrutura e do tipo de arquivo não importam.

Você pode ainda supor que o arquivo de entrada está codificado em ASCII.

Parte 3: UCS

Nesta parte você implementará uma função de busca de custo uniforme chamada `uniform_cost_search` que recebe um objeto da classe `Problem` representando um problema de busca e retorna um objeto da classe `Node` contendo uma solução para o problema ou `None` se o problema é insolúvel.

Essas classes estão especificadas em `ep0.py`. Você deve atentar para dois detalhes muito importantes: O primeiro é que o nó de busca inicial deve ter o atributo parent apontando para `None`, isso será importante pois a função responsável por construir a sequencia de ações realizada na sua solução (função `check_solution`) precisa dessa informação para funcionar.

O segundo é que para muitos domínios a função custo pode ser descrita dessa forma: $g(a_1, \ldots, a_n) = \sum_{i=1}^{n} C(s_{i-1}, a_i)$ onde $s_i = T(s_{i-1}, a_i)$, ou seja, tornando a função custo uma função do estado e da ação. Note que $g(a_1, \ldots, a_n) - g(a_1, \ldots, a_{n-1}) = C(s_{i-1}, a_n)$, o que nos permite calcular o custo de uma solução parcial para um nó de busca a partir do custo da solução parcial do nó pai e da função $C(s, a)$.

Desta forma o problema for ser descrito equivalentemente através da função $C(s, a)$ ao invés da função $g(a_1, \ldots, a_n)$. Essa é a abordagem adotada nesse EP.

Instruções para entrega

Você deve submeter via PACA apenas o arquivo `ep0.py` contendo a sua solução até às 23:55 do dia 10/03/2019. Para evitar que seu EP seja zerado, certifique-se que o arquivo foi submetido sem problemas (baixando e executando o arquivo do site) e que ele consiste em um script executável escrito em Python 3.

Além disso, não deixe de preencher o ca-

beçalho fornecido e atente para as questões de integridade acadêmica, ou seja, caso você utilize diretamente, ou se baseie fortemente em fontes que não sejam as fornecidas em sala, não se esqueça de inclui-las no arquivo de submissão (pode ser tanto no cabeçalho como na descrição do protótipo da função). Lembrem-se que casos de plágio serão tratados com o rigor acadêmico esperado.

Avaliação

O seu programa será avaliado pela implementação bem sucedida de cada função independentemente, assim como por questões subjetivas como documentação do código e simplicidade. Assim, você ainda pode conseguir uma boa nota mesmo que não consiga implementar todas as funções corretamente bem como pode obter uma nota baixa mesmo que implemente todas as funções. O critério detalhado de avaliação será divulgado posteriormente.

Conforme mencionado anteriormente, é **muito importante** que seu código passe em todos os testes. Entretanto, passar em todos os testes não é garantia de que seu código receberá nota máxima. É possível que os testes não verifiquem alguns casos peculiares onde seu programa pode vir a falhar.

Não deixe para a última hora e bom trabalho!