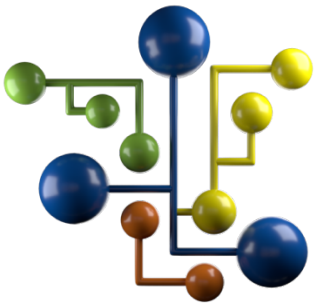

Data Science Academy

www.datascienceacademy.com.br

Introdução à Lógica de Programação

O Problema da Mochila

Suponha que você seja um colecionador e esteja com sua mochila em um evento de itens antigos que só acontece a cada 2 anos. No evento existem diversos itens em promoção que você deseja comprar. Porém você só pode levar aquilo que caiba na sua mochila, que só suporta 15 Kg.

Você está tentando maximizar o valor dos itens que colocará na sua mochila. Para isso, que algoritmo você usa? Que tal esse:

1. Pegue o item mais caro que caiba na sua mochila.
2. Pegue o próximo item mais caro que caiba na sua mochila, e assim por diante.

Por exemplo, suponha que existam três itens que você deseja comprar:

1- Radio Toca-Fita-Cassete do Fusca 1967 – Valor R$3.000 e Peso 13 Kg
2- Notebook Compaq de 1986 – Valor R$2.000 e Peso 9 Kg
3- Violão do John Lennon de 1972 – Valor R$1.500 e Peso 6 Kg

Quais itens você deveria comprar para maximizar o valor comprado?

Sua mochila suporta 15 quilos. O aparelho de som é o item mais caro e você pode comprá-lo. Mas agora não há espaço para mais nada.

Você comprou R$ 3.000 em bens, mas espere um pouco! Caso tivesse comprado o notebook e o violão, você poderia ter R$ 3.500! E os 2 itens caberiam na sua mochila!

Claramente, a estratégia gulosa estudada na Aula 11 não oferece a melhor solução aqui, mas fornece um valor bem próximo. Vou explicar agora como calcular a solução correta, mas se você é um colecionador voraz, talvez não se importe com a melhor solução. “Muito bom” é bom o suficiente.

Moral da história para este exemplo: às vezes, o melhor é inimigo do bom. Em alguns casos, tudo o que você precisa é de um algoritmo que resolva o problema de uma maneira muito boa. E é aí que os algoritmos gulosos entram, pois eles são simples de escrever e normalmente chegam bem perto da solução perfeita.

Mas ainda temos um problema a resolver: Quais itens você deveria comprar para maximizar o valor comprado?

O algoritmo mais simples é o seguinte: você deve testar todos os conjuntos de itens possíveis e descobrir qual conjunto maximizará o valor comprado e caiba na sua mochila.

Isto funciona, mas é uma solução muito lenta, pois para três itens você deverá calcular oito conjuntos possíveis. Para quatro itens, são 16 conjuntos. Cada item adicionado dobrará o número de cálculos. Este algoritmo tem tempo de execução O(2^n); é muito, muito lento. Esta solução não é prática para qualquer número razoável de itens.

Então, como calcularemos a solução ideal?

Resposta: Usando a Programação Dinâmica! Vamos observar como o algoritmo da Programação Dinâmica funciona.

Ele começa com a resolução de subproblemas e vai escalando-os até resolver o problema geral. No problema da mochila, você começaria resolvendo o problema para mochilas menores (ou "submochilas") e iria escalando estes problemas até resolver o problema original.

A Programação Dinâmica (PD) é uma das técnicas disponíveis para resolver problemas de auto-aprendizagem. É amplamente utilizado em áreas como pesquisa operacional, economia e sistemas de controle automático, entre outras. A Inteligência Artificial é a principal aplicação da PD, uma vez que lida principalmente com informações de aprendizado em um ambiente altamente incerto. Alguns algoritmos de PD são estudados no curso Deep Learning II e Processamento de Linguagem Natural.

Os algoritmos de PD podem ser classificados em três subclasses.

Value iteration algorithms
Policy iteration algorithms
Policy search algorithms

Todos esses algoritmos têm suas respectivas vantagens em várias aplicações.

A PD também é muito útil para problemas de espaço grande e contínuo em tempo real. Ele fornece informações de representações de políticas complexas por meio de uma técnica chamada "aproximação". Isso significa fornecer uma estimativa do grande número de possíveis valores alcançáveis por meio de iterações de valor ou política na PD. A amostragem de dados grandes pode ter um grande efeito no aspecto de aprendizado em qualquer um dos algoritmos.

Aqui você encontra uma lista com algumas centenas de aplicações da PD:

https://www.geeksforgeeks.org/dynamic-programming/

Programação Dinâmica é um assunto complexo e poderíamos facilmente ter um curso inteiro dedicado ao tema! Mas veremos um exemplo bem interessante em Python!

O diferencial da Programação Dinâmica está em construir uma “memória” das soluções encontradas, até encontrar a melhor solução. Essa “memória” nada mais é do que uma estrutura de dados em Python, por exemplo, como Dicionários, Listas ou Tuplas. A Programação Dinâmica aplica conceitos de diversos algoritmos que estudamos até aqui.

Imagine que você receba uma string como esta:

marceloachaqueoclimapodemudar

E seu trabalho é produzir uma saída como esta:

marcelo acha que o clima pode mudar

Como você resolveria esse problema? A Programação Dinâmica pode nos ajudar! Cuidado com a identação e leia atentamente os comentários que foram colocados para você. Compreenda cada linha de código e o que e porque está sendo feito! Não há qualquer técnica avançada no código, apenas o arranjo necessário para resolver o problema.

```
# Programação Dinâmica - Partição de Strings

# Input: marceloachaqueoclimapodemudar
# Output: marcelo acha que o clima pode mudar

# Dicionário com as palavras disponíveis
dicionario = {
"acha": True,
"que": True,
"o": True,
"clima": True,
"pode": True,
"mudar": True
}

# Função de custo
def calculaCusto(palavra):
--"""Avalia o custo de uma determinada palavra.
--Retorna 0 se a palavra estiver no dicionário, ou o número de caracteres caso contrário.

--Argumentos:
----palavra (string): uma palavra cujo custo precisa ser avaliado.

--Retorno:
----O custo da palavra (int).
----"""
```

```
---if palavra in dicionario:
----return 0
--else:
----return len(palavra)

# O cache para memorizar soluções parciais
# Aqui está a grande diferença da Programação Dinâmica, uma espécie de "memória" para armazenar soluções parciais
cache = {}

# Função para dividir a string de input
def divideString(input_string, inicio_index):
--"""Esta função recursiva, tenta dividir a substring começando em 'inicio_index' em partições, ou seja, palavras!

--Argumentos:
----input_string (string): a string inicial que precisa ser dividida.
----inicio_index (int): queremos dividir a substring de input_string começando nesse índice

--Retorno:
----Uma forma de tupla da solução parcial e seu custo
"""

--# Já calculamos a solução ideal a partir do ponto
--if inicio_index in cache:
----return cache[inicio_index]

--# A substring para dividir
--substring = input_string[inicio_index:]

--# Estas são as condições de contorno
--# Se a substring estiver vazia, retorne uma string vazia sem nenhum custo
--if not len(substring):
----return '', 0

--min_cost = None
--min_string = None

--# Colocamos nossa próxima partição em algum lugar entre o início + 1 e o final do input_string
--for i in range(1, len(substring) + 1):

----# Dividimos o resto da string recursivamente
----rest_string, rest_cost = divideString(input_string, inicio_index + i)
```

```
----current_string = substring[:i]
----current_cost = calculaCusto(current_string) + rest_cost

----# Atualiza o custo mínimo e string, se for o melhor até agora
----if min_cost is None or current_cost < min_cost:

------# Se as duas partes não estiverem vazias, junte-as com espaço
------if current_string and rest_string:
--------min_string = current_string + ' ' + rest_string

--------# Adicionamos um custo ao espaço em branco para evitar a divisão de palavras desconhecidas em pequenos pedaços
--------current_cost += 1
------else:
--------min_string = current_string + rest_string

------min_cost = current_cost

--cache[inicio_index] = min_string, min_cost
--return min_string, min_cost


# Executa a função começando do índice 0
print(divideString("marceloachaqueoclimapodemudar", 0))
```

Saída: ('marcelo acha que o clima pode mudar', 13)

```
# Executa a função começando do índice 7
print(divideString("marceloachaqueoclimapodemudar", 7))
```

Saída: ('acha que o clima pode mudar', 5)

Observe que aplicamos recursão e fomos dividindo a string de input, comparando com as palavras no dicionário e gravando os resultados intermediários na "memória". O exemplo acima é aplicado em algumas aulas do curso de Processamento de Linguagem Natural da Formação IA.

Quem quiser conhecer mais sobre o famoso Problema da Mochila, encontra alguns detalhes aqui:

https://pt.wikipedia.org/wiki/Problema_da_mochila
#aula12