

## Exercício 6.3 (Papadimitriou)

Alice Duarte Scarpa, Bruno Lucian Costa

2015-06-23

### 1 Enunciado

O Yuckdonald's está considerando abrir uma cadeia de restaurantes em Quaint Valley Highway (QVH). Os  $n$  locais possíveis estão em uma linha reta, e as distâncias desses locais até o começo da QVH são, em milhas e em ordem crescente,  $m_1, m_2, \dots, m_n$ . As restrições são as seguintes:

- Em cada local, o Yuckdonald's pode abrir no máximo um restaurante. O lucro esperado ao abrir um restaurante no local  $i$  é  $p_i$ , onde  $p_i > 0$  e  $i = 1, 2, \dots, n$ .
- Quaisquer dois restaurantes devem estar a pelo menos  $k$  milhas de distância, onde  $k$  é um inteiro positivo.

Dê um algoritmo eficiente para computar o maior lucro total esperado, sujeito às restrições acima.

### 2 Introdução

Com este exercício vamos abordar uma técnica chamada de programação dinâmica que tem como característica que a solução ótima pode ser calculada de soluções de subproblemas.

Antes porém, vai ser apresentado uma solução utilizando um algoritmo guloso.

## 3 Soluções para o problema

### 3.1 Algoritmo guloso

Esse algoritmo recebe duas lista de tamanho  $n$ , uma com as distancias dos locais até o ponto inicial e a outra com os respectivos lucros esperados, e um inteiro  $k$  que é a distancias em milhas desejada. E começamos nosso algoritmo saindo do ponto inicial, em direção ao fim da QVH

```
def restaurante(distancias,k,lucros):

    lmax = max(distancias) # Valor maximo das distancias
    rest = []
    lucro = []
    lucro.append(0) # iniciando lucro de 0 loja, como 0
    j = 0 # posicao da loja
    for i in range(lmax):# percorrendo toda a QVH
        if j in distancias: #Quando local estiver disponivel
            rest.append(distancias[distancias.index(j)]) # guarda posicao escolhida
            lucro.append(lucros[distancias.index(j)]) # guarda lucro escolhido
            j=j+k # pula k milhas a frente
        else:
            j += 1 # anda 1 milha a frente
    return sum(lucro) #retorna soma dos lucros escolhidos
```

Vamos reproduzir um exemplo no qual esse algoritmo não retonar o valor máximo possível e vamos tentar entender.

Chamada da funcao:

```
restaurante([3, 8, 9, 15],3,[5, 6, 10, 8])
```

Resultado:

19

O resultado obtido utilizando desse algoritmo não foi o resultado ótimo pois nesse exemplo é fácil perceber que o valor máximo que se pode ter respeitando as restrições é de 23, no qual a escolha seria feita pelos locais [3, 9, 15], porém o algoritmo guloso está instruído sempre a escolher o primeiro local vago respeitando as restrições, ou seja nesse exemplo ele escolhe os locais [3, 8, 15] totalizando o lucro de 19 que é inferior ao valor ótimo. O algoritmo guloso funcionaria bem para o caso que todos os locais tem o mesmo lucro esperado.

Vamos resolver esse problema com utilizando um algoritmo baseado no paradigma de programação dinâmica.

### 3.2 Algoritmo utilizando programação dinâmica

Esse técnica de programação utiliza as soluções dos sub-problemas para calcular a solução do problema.

Então vamos definir o nosso sub-problema: Suponha  $L(i)$  como o lucro máximo que podemos obter com os locais de 1 até  $i$

Esse algoritmo recebe duas lista de tamanho  $n$ , uma com as distancias dos locais até o ponto inicial e a outra com os respectivos lucros esperados, e um inteiro  $k$  que é a distancias em milhas desejada.

```
def restaurante(distancias,k,pay):
    lucro = [0 for a in range(len(pay)+1)]
    for i in range(len(distancias)):
        m_novo = distancias[i]-k
        i_est=[b for b in distancias if b <= m_novo]
        if len(i_est) > 0:
            i_est=i_est[-1]
            d_est= distancias.index(i_est)
            d_novo=lucro[d_est+1]
        else:
            d_novo=0
        lucro[i+1]=max(lucro[i],pay[i]+d_novo)

    return lucro[-1]

print restaurante([3, 8, 9, 15],3,[5, 6, 10, 8])
```

## 4 Complexidade

Para o problema proposto foi apresentado quatro possiveis soluções. Duas opções “naive”, uma força bruta e outra utilizando o método de algoritmo guloso.

A primeira solução “naive” é linear em relação ao tamanho da rua, ou seja, tem complexidade  $O(m)$ , onde  $m$  é a distancia máxima que temos uma casa.

A segunda solução “naive” é linear em relação ao tamanho do vetor de distancias, ou seja, tem complexidade  $O(n)$ , onde  $n$  é o número de casas na

rua.

A terceira solução é uma força bruta, escolhendo aleatoriamente uma posição para colocar a antena o que com valor de  $n$  muito grande torna-se algo exponencial.

A quarta solução é a única solução ótima e possui complexidade também linear, dependendo apenas do tamanho da rua.