

Exercício 4.5 (Tardos)

Alice Duarte Scarpa, Bruno Lucian Costa

2015-06-23

1 Enunciado

Vamos considerar uma rua campestre longa e quieta, com casas espalhadas bem esparsamente ao longo da mesma. (Podemos imaginar a rua como um grande segmento de reta, com um extremo leste e um extremo oeste.) Além disso, vamos assumir que, apesar do ambiente bucólico, os residentes de todas essas casas são ávidos usuários de telefonia celular.

Você quer colocar estações-base de celulares em certos pontos da rodovia, de modo que toda casa esteja a no máximo quatro milhas de uma das estações-base. Dê um algoritmo eficiente para alcançar esta meta, usando o menor número possível de bases.

2 Introdução

Com este exercício vamos abordar uma técnica chamada de algoritmos gulosos sempre realizando a escolha que parece ser a melhor no momento, fazendo uma escolha ótima local, com intuito de que esta escolha leve até a solução ótima global.

Antes porém, vai ser apresentado soluções utilizando algoritmos “naive” e um força bruta.

3 Soluções para o problema

3.1 Algoritmo naive

Esta primeira solução para o problema é uma das mais simples possíveis de se pensar quando confrontamos o problema. O problema diz que temos que colocar uma antena a no máximo 4 milhas de distancias, nesse algoritmo fizemos a solução baseado apenas nessa ideia, então com ele vamos colocar

uma antena a cada 4 milhas de distancia até que a casa mais distante esteja coberta pela nossas antenas.

```
def antena(lista):
    lmax = max(lista) # Valor maximo presente na lista de distancias
    ant = []
    j = 0
    for i in range(lmax): # Coloca uma antena a cada 4 milhas
        if j >= lmax: # Ver se a antena tem posicao maior que maximo da lista
            return ant
        j += 4
    ant.append(j)
```

Esse algoritmo bem simples nos retorna uma solução correta para o problema, mas ele ainda nos faz colocar muitas antenas de forma desnecessárias como podemos ver no exemplo a seguir.

Chamada da função:

```
print antena([3, 16, 11, 18, 5, 17, 24, 29, 1, 301])
```

Resultado: [4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100, 104, 108, 112, 116, 120, 124, 128, 132, 136, 140, 144, 148, 152, 156, 160, 164, 168, 172, 176, 180, 184, 188, 192, 196, 200, 204, 208, 212, 216, 220, 224, 228, 232, 236, 240, 244, 248, 252, 256, 260, 264, 268, 272, 276, 280, 284, 288, 292, 296, 300, 304]

Outro algoritmo “naive” que tem uma solução melhor do que anterior será apresentado a seguir.

```
def antena(lista):
    ant=[]
    for i in xrange(len(lista)): #Pecorre toda a lista
        ant.append(lista[i]) #Para cada item da lista coloca uma antena

    ant.sort()
    return ant
```

Neste algoritmo a ideia seria colocar uma antena para cada casa o que resolveria nosso problema.

Vamos rodar o novo algoritmo com o mesmo exemplo que usamos na solução anterior para compararmos as soluções.

```
print antena([3, 16, 11, 18, 5, 17, 24, 29, 1, 301])
```

Resultado:

1 3 5 11 16 17 18 24 29 301

Já conseguimos perceber uma diferença muito grande entre as soluções.

Esses dois algoritmos até agora apresentados não nos retorna a melhor solução, os proximos algoritmos tentaremos conseguir a solução ótima para resolução deste problema.

3.2 Força bruta

Esse algoritmo de força bem simples escolhe um ponto qualquer dentro dessa rua para coloca uma antena, depois disso ele percorre toda a lista para ver se tem alguma casa que é coberta por essa antena, se tiver retiramos essa casa da lista e efetuamos esse procedimento até que todas as casas tenham sido cobertas.

```
import math, numpy
def antena(lista):
    lmax = max(lista) # Valor maximo presente na lista de distancias
    ant = []

    while lista != []: # Realizar procedimento ate todas as casas cobertas
        torre = numpy.random.randint(1, lmax) #fixando uma torre em um ponto qualquer
        for j in lista: #Passando toda a lista
            if j >= torre-4 and j <= torre+4: # Verifica se tem casa esta coberta
                lista.remove(j) # remove a casa coberta
                ant.append(torre) # adiciona a torre a lista

    ant = list(set(ant)) # Remove as torres colocadas em duplicatas
    ant.sort() #Ordena as torres

    return ant
```

Vamos rodar o algoritmo com o mesmo exemplo usado com os algoritmos anteriores para vermos a diferença entre as soluções.

```
print antena([3, 16, 11, 18, 5, 17, 24, 29, 1, 301])
```

Resultado:

2 5 9 13 22 26 299

A solução do algoritmo para esse problema pode até ser a ótima eventualmente mas em suma ele demora mais a conseguir uma resposta para o problema devido a sua escolha aleatória do local a colocar a antena.

Em outras palavras esse algoritmo trabalha muito parecido com o jogo de batalha naval, ele escolhe aleatoriamente uma antena para colocar porém algumas vezes pode escolher em local vazio gerando retrabalho o algoritmo.

3.3 Algoritmo guloso

Esse algoritmo recebe uma lista com as distâncias das casas até o ponto inicial. Começamos nosso algoritmo saindo do ponto inicial, a oeste, em direção ao leste até que primeira casa esteja 4 milhas a oeste colocamos uma antena neste local e retiramos da lista todas as casas cobertas por essa antena. Depois continuamos com esse processo até todas as casas serem retiradas da lista.

```
def antena(lista):
    ant = []
    lista.sort()

    tamanho = len(lista)
    for i in range(tamanho):
        if len(ant) == 0:
            # 0 valor de -10 nao afeta a resposta, pois as posicoes
            # das casas sao positivas
            alcance = -10
        else:
            alcance = ant[-1] + 4

        if lista[i] > alcance:
            ant.append(lista[i] + 4)

    return ant
```

Vamos reproduzir o mesmo exemplo feito com o algoritmos anteriores para vermos a diferença entre as soluções.

```
print antena([3, 16, 11, 18, 5, 17, 24, 29, 1, 301])
```

Resultado:

Esse algoritmo sempre nos retorna a solução ótima e vamos mostrar isso a seguir.

Suponha $S = \{s_1, \dots, s_k\}$ sendo a solução com as posições das antenas que o nosso algoritmo retornou e $T = \{t_1, \dots, t_m\}$ sendo a solução ótima com as posições das antenas ordenadas de forma crescente. Queremos mostrar que $k = m$.

Vamos mostrar nosso algoritmo S “stays ahead” da solução T . Ou seja, $s_i \geq t_i$. Para $i = 1$ essa afirmação é verdade, já que vamos ao leste o máximo possível antes de colocar a antena. Iremos assumir também é verdade para $i \geq 1$, ou seja, $\{s_1 \dots s_i\}$ cobre as mesmas casas que $\{t_1 \dots t_i\}$, então se adicionarmos t_{i+1} para $\{s_1 \dots s_i\}$, não deixa nenhuma casa entre s_i e t_{i+1} descobertas. Mas no passo $(i + 1)$ do algoritmo guloso é escolhido o s_{i+1} para ser o maior possível com a condição cobrir as casas entre s_i e s_{i+1} e então $s_{i+1} > t_{i+1}$ o que prova o que queríamos.

Então, se $k > m$, a solução $\{s_1 \dots s_m\}$ falha ao cobrir todas as casas, mas $s_m \geq t_m$ logo $\{t_1 \dots t_m\} = T$ também falha ao cobrir todas as casas. O que é uma contradição, pois assumimos que T era uma solução ótima para o problema.

4 Complexidade

Para o problema proposto foi apresentado quatro possíveis 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 distância máxima que temos uma casa.

A segunda solução “naive” é linear em relação ao tamanho do vetor de distâncias, ou seja, tem complexidade $O(n)$, onde n é o número de casas na rua, a menos da ordenação do final. Com a ordenação, a complexidade é $O(n \log n)$.

A terceira solução é uma força bruta, escolhendo aleatoriamente uma posição para colocar a antena o que, com estradas muito grandes, pode demorar uma quantidade de tempo não-polinomial.

A quarta solução é a única solução ótima e possui complexidade linear a menos da chamada para a função de ordenação. Ou seja, seria linear se a entrada já viesse ordenada. Como este não é necessariamente o caso, o algoritmo demora tempo $O(n \log n)$.