FUNDAÇÃO GETÚLIO VARGAS

ESCOLA DE MATEMÁTICA APLICADA

MESTRADO 2015.1

ESTRUTURA DE DADOS E SEUS ALGORITMOS

Prof Alexandre Rademaker

Apresentação e discussão dos problemas selecionados

ALUNOS:

KIZZY TERRA

OTTO TAVARES

RIO DE JANEIRO

JUNHO DE 2015

1 Introdução

Este relatório reúne comentários e implementações dos exercícios selecionados cujos enunciados foram previamente apresentados em sala de aula. O objetivo deste documento é, acima de tudo, apresentar as interessantes discussões que emergiram a medida que os exercícios foram sendo analisados e solucionados, ultrapassando a mera exposição da resposta final encontrada para cada questão escolhida. Cabe ressaltar que a maioria dos exercícios selecionados foram utilizados apenas como ponto de partida para discussões mais enriquecedoras a fim de proporcionar maior aprendizado e possibilitando os conhecimentos adquiridos ao longo do curso fossem, de fato, colocados em prática.

2 Exercícios selecionados: comentários e discussões

2.1 Subsequência de soma máxima

O problema da subsequência de soma máxima consiste em encontrar a subsequência de números com maior soma dentro de lista de números unidimensional dada. Este problema possui duas principais variações unidimensionais: na primeira, consideram-se apenas subsequências contíguas da lista de números; na segunda, por sua vez, as subsequências podem envolver números que não são contíguos.

O problema da subsequência contígua de soma máxima foi proposto pela primeira vez em 1977 por Ulf Grenander da *Brown University,* como um modelo simplificado do problema de estimativa de padrões por máxima verossimilhança em imagem digitalizadas. O problema original do estimador de máxima verossimilhança é na verdade um problema de subsequência de soma máxima bidimensional, mas foi proposto por Grenander em sua forma simplificada (unidimensional) visto que o problema bidimensional exigia mais tempo para ser resolvido. Uma solução de tempo linear para o problema unidimensional foi encontrada pouco tempo depois que o problema foi proposto por Jay Kadane da *Carnegie-Mellon University.[Bentley, 1984 —]*

2.1.1 A primeira versão

Problema: Subsequência contígua de soma máxima

Entrada: Uma lista de números a_1, a_2, a_3, $\ldots$, a_n

Saída: A subsequência contígua de soma máxima (uma subsequência de tamanho zero possui soma zero)

Discussão

A solução mais ingênua que se pode dar para esta questão consiste de um algoritmo força bruta que calcula todas as possíveis combinações de subsequências, calcula a soma para cada uma delas e retorna a subsequência correspondente a maior soma encontrada. A seguir é apresentada a implementação deste algoritmo em Python.

O método implementado possui três laços *for* encadeados e cada um deles irá iterar no máximo N vezes, em que N é o número de elemntos da lista de números dada como entrada, portanto a complexidade deste algoritmo é $O(N^3)$.

```python
def cubic(L):
    sizeL = len(L)
    if sizeL == 0:
        return L
    max_sum = L[0]
    max_subsequence = L[:1]
    for j in range(1, sizeL):
        for i in range(j):
            current_sum = 0
            for k in range(i,j):
                current_sum += L[k]
            if current_sum > max_sum:
                max_sum = current_sum
                max_subsequence = L[i:j]
    return max_subsequence
```

A partir da complexidade do algoritmo já é possível observar que esta solução é pouco eficiente, apenas para ilustrar, em um notebook comum, para uma entrada de tamanho $N = 1000$ este método levou aproximadamente 13 segundos para terminar de executar e para $N = 10000$ o tempo de execução foi X horas.

A solução cúbica apresentada é claramente ineficiente e por essa razão, basta pensar um pouco para identicar que podemos modificá-la facilmente e torná-la ligeiramente melhor. É possível tornar o algortimo anteriormente proposto mais eficiente apenas eliminando o laço *for* mais interno, de forma a obtermos a implementação a seguir:

```python
def quadratic(L):
    sizeL = len(L)
    if sizeL == 0:
        return L
    max_sum = L[0]
    max_subsequence = L[:1]
    for j in range(sizeL):
        current_sum = 0
        for i in range(j, sizeL):
            current_sum += L[i]
            if current_sum > max_sum:
                max_sum = current_sum
                max_subsequence = L[i:j+1]
    return max_subsequence
```

Esta implementação possui dois laços *for* encadeados e cada um será executado N vezes, no pior caso, logo a complexidade do algoritmo apresentado é $O(N^2)$. Para esta implementação o tempo de execução de uma entrada de tamanho $N = 1000$ em um notebook com capacidade de processamento regular foi de 154 milésimos de segundo e para $N = 10000$ o tempo de execução foi 6 segundos, aproximadamente.

Embora o algoritmo quadrático seja mais eficiente do que o algoritmo cúbico, é possível propor soluções ainda melhores. A seguir será discutida uma solução linear similiar à solução proposta por Kadane na ocasião em que o problema foi proposto.

A solução linear

A idéia para esta implementação é utilizar a técnica de programação dinâmica, isto é, construir a solução de um problema através da solução de subproblemas associados. Neste contexto, a primcipal idéia é considerar que para cada elemento I da lista de números a solução procurada (subsequência contígua de soma máxima) é dada pela solução encontrada para o elemento anteiror I - 1(se existir) mais o atual elemento I, ou seja, $S[I-1] \cup L[I]$ ou a solução procurada é uma subsequência que começa no elemento I, nesse caso $S[I] = \{L[I]\}$. Para implementar esta idéia é necessário utilizar variáveis auxiliares para armazenar a melhor solução encontrada, bem como a solução para cada iteração. Assim, o método inicia-se através da inicialização das variáveis como segue:

```
startSoFar = 0
startPrevious = 0
endSoFar = 0
maxSoFar = L[0]
maxPrevious = L[0]
```

A variável *startSoFar* armazena o índice do elemento que inicia a subsequência de maior soma, a variável *startPrevious* armazena o índice do elemento que inicia a subsequência de maior soma encontrada na iteração anterior; a variável endSoFar armazena o índice do elemento que encerra a subsequência de maior soma, a variável maxSoFar armazena o valor da subsoma máxima e a variável maxPrevious armazena o valor da subsoma máxima encontrada na iteração anterior.

Em seguida implementa-se um laço *for* para calcular a solução do problema para cada elemento I (em cada iteração calcula-se a resposta para o problema como se a lista de números terminasse no elemento I), para isso verifica-se qual subsequência possui maior soma: a subsequência composta por todos os elementos até I ($S[I-1] \cup L[I]$) ou a subsequência que se inicia em I($S[I] = \{L[I]\}$). Uma vez encontrada a melhor solução para a iteração as variáveis auxiliares são devidamente atualizadas: caso a subsoma encotrada ao final de uma determinada iteração (*maxPrevious*) seja maior que a subsoma máxima encontrada em todas as iterações anteriores (*maxSoFar*), então as variáveis *maxSoFar*, *startSoFar* e *endSoFar* são modificadas.

```python
if maxPrevious + L[i] >= L[i]:
    maxPrevious = maxPrevious + L[i]
else:
    startPrevious = i
    maxPrevious = L[i]
if maxPrevious >= maxSoFar:
    maxSoFar = maxPrevious
    endSoFar = i
    startSoFar = startPrevious
```

Ao final de todas as iterações a subsequência de soma máxima é retornada. O método completo é apresentado a seguir:

```python
def linear(L):
    startSoFar = 0
    startPrevious = 0
    endSoFar = 0
    maxSoFar = L[0]
    maxPrevious = L[0]
    for i in range(1,len(L)):
        if maxPrevious + L[i] >= L[i]:
            maxPrevious = maxPrevious + L[i]
        else:
            startPrevious = i
            maxPrevious = L[i]
        if maxPrevious >= maxSoFar:
            maxSoFar = maxPrevious
            endSoFar = i
            startSoFar = startPrevious
    return L[startSoFar:endSoFar+1]
```

No método apresentado o laço *for* é o único bloco da implementação que depende do tamanho da entrada e este laço irá executar N vezes, e para cada iteração irá executar um número constante de operações O(1), portanto a complexidade desta solução é $O(N)$. O tempo de execução deste método para uma entrada de tamanho N=1000 foi de 25 milésimos, para N=10000 foi de 88 milésimos.

2.1.2 A segunda versão

Problema: Subsequência de soma máxima

Entrada: Uma lista de números a_1, a_2, a_3, ... , a_n
Saída: A subsequência (contígua ou não) de maior soma

Discussão

2.1.3 A versão bidimensional

Problema: Subsequência de soma máxima - bidimensional

Entrada: Duas listas de números
Saída: A subsequência (contígua ou não) de maior soma

Discussão

Complexidade das soluções encontradas até hoje...