

Análise de Algoritmos

Paradigma Divisão e Conquista

Paradigma de Divisão-e-Conquista

- Este paradigma consiste em resolver problemas de forma recursiva, aplicando três passos em cada nível de recursão:
 - **Dividir:** o problema em um número de subproblemas que são instâncias menores do mesmo problema;
 - **Conquistar:** os subproblemas solucionando-os recursivamente; porém no caso de subproblemas muito pequenos apenas determine a solução; e
 - **Combinar:** as soluções dos subproblemas na solução para o problema original.

Paradigma de Divisão-e-Conquista

- Quando os subproblemas são suficientemente grandes para serem solucionados recursivamente, são chamados de **casos recursivos**; e
- Uma vez que, o subproblema tornando-se menor o suficiente para não necessitar de recursão para solução, então são chamados de **caso base**.
- Às vezes, além de subproblemas que são instâncias menores do mesmo problema, temos que resolver subproblemas que não são exatamente iguais ao problema original. Consideramos a resolução de tais subproblemas como parte da etapa de combinação.

Paradigma de Divisão-e-Conquista

- **Recorrências:**

- As recorrências andam de mãos dadas com o paradigma de divisão e conquista, porque nos dão uma maneira natural de caracterizar os tempos de execução desses algoritmos. **Uma recorrência é uma equação ou desigualdade que descreve uma função em termos de seu valor em entradas menores.**
- As recorrências podem assumir muitas formas. Por exemplo, um algoritmo recursivo pode dividir subproblemas em tamanhos desiguais, como uma divisão de $2/3$ - para $1/3$. Se as etapas de dividir e combinar levarem tempo linear, tal algoritmo daria origem à recorrência : $T(n) = T(2n/3) + T(n/3) + \Theta(n)$
- Os subproblemas não são necessariamente limitados a serem uma fração constante do tamanho original do problema, como em $T(n) = T(n-1) + \Theta(1)$.

Paradigma de Divisão-e-Conquista

Vamos apresentar a solução de dois problemas, onde comparamos, a complexidade de uma solução sem e outra com o uso de Divisão-e-Conquista.

As complexidades dos algoritmos recursivos estão determinadas a partir das recorrências.

Os problemas são:

- Determinar o Subvetor de Valor Máximo; e
- Multiplicação de Matrizes Quadradas.

Paradigma de Divisão-e-Conquista

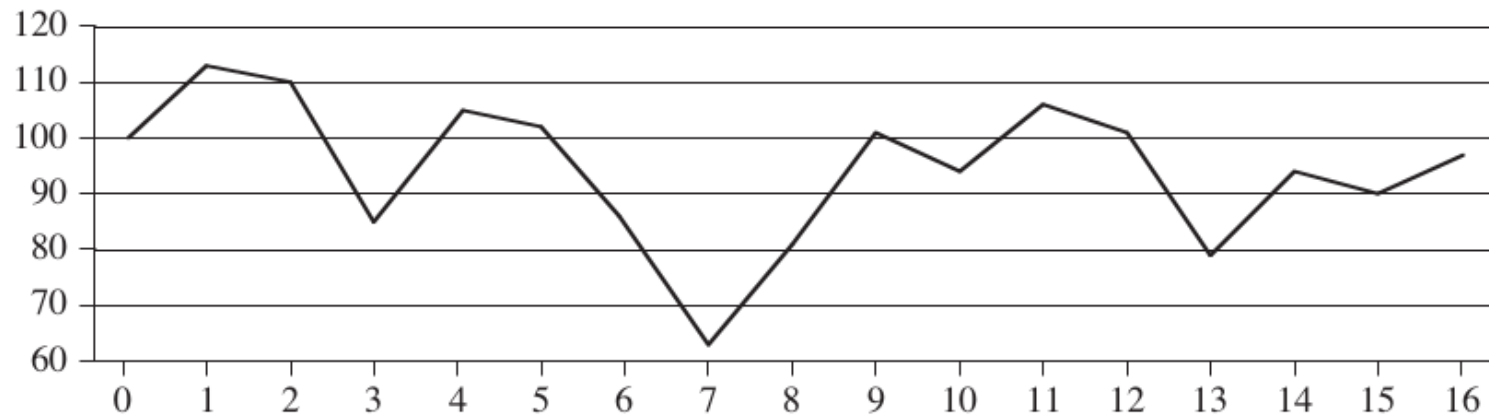
Problema:

Determina o Subvetor de Valor Máximo

Problema do subvetor máximo

- Suponha que lhe ofereceram a oportunidade de investir na Volatile Chemical Corporation. Assim como os produtos químicos que a empresa produz, o preço das ações da Volatile Chemical Corporation é bastante volátil.
- Você pode comprar uma unidade de ação apenas uma vez e depois vendê-la em uma data posterior, comprando e vendendo após o fechamento do pregão do dia. Para compensar essa restrição, você pode saber qual será o preço da ação no futuro.
- Seu objetivo é maximizar seu lucro. A Figura 1 mostra o preço da ação em um período de 17 dias. Você pode comprar a ação a qualquer momento, começando após o dia 0, quando o preço for de \$ 100 por ação.

Problema do Subvetor Máximo



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

Problema do Subvetor Máximo

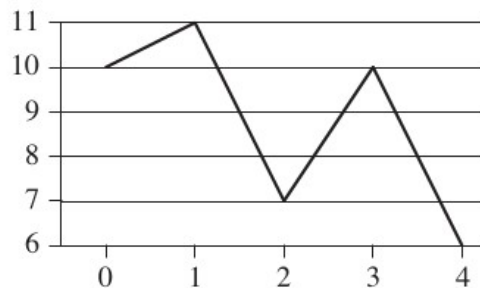
- É claro que você gostaria de “comprar na baixa, vender na alta” — comprar pelo preço mais baixo possível e depois vender pelo preço mais alto possível — para maximizar seu lucro.
- Infelizmente, você pode não conseguir comprar pelo preço mais baixo e depois vender pelo preço mais alto dentro de um determinado período. Na Figura 1, o preço mais baixo ocorre após o dia 7, que ocorre após o preço mais alto, após o dia 1.

Problema do Subvetor Máximo

- Você pode pensar que sempre pode maximizar o lucro comprando pelo preço mais baixo ou vendendo pelo preço mais alto.
- Por exemplo, na Figura 1, maximizaríamos o lucro comprando pelo preço mais baixo, após o dia 7. Se essa estratégia sempre funcionasse, seria fácil determinar como maximizar o lucro: encontre os preços mais altos e mais baixos e, em seguida, trabalhe para a esquerda do preço mais alto para encontrar o preço anterior mais baixo, trabalhe diretamente do preço mais baixo para encontrar o preço mais alto depois e pegue o par com a maior diferença.

Problema do Subvetor Máximo

- A Figura 2 mostra um contra-exemplo simples, demonstrando que o lucro máximo às vezes não vem da compra pelo preço mais baixo nem da venda pelo preço mais alto.



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	-4	3	-4

Problema do Subvetor Máximo

- FORÇA BRUTA

- Podemos facilmente conceber uma solução de força bruta para esse problema: tente todos os pares possíveis de datas de compra e venda em que a data de compra precede a data de venda.
- Qual A COMPLEXIDADE ??

Problema do Subvetor Máximo

- Um período de n dias tem $\binom{n}{2}$ pares de datas. Como $\binom{n}{2}$ é $\Theta(n^2)$, e o melhor que podemos esperar é avaliar cada par de datas em tempo constante, essa abordagem levaria um tempo $\Omega(n^2)$. Podemos fazer melhor?
- Para projetar um algoritmo com um tempo de execução $O(n^2)$, veremos a entrada de uma maneira ligeiramente diferente. Queremos encontrar uma sequência de dias em que a variação líquida do primeiro ao último dia seja máxima.

Problema do Subvetor Máximo

- Em vez de olhar para os preços diários, consideremos a variação diária do preço, onde a variação no dia i é a diferença entre os preços após o dia $i - 1$ e após o dia i . O vetor abaixo mostra essas mudanças diárias, e representa a linha inferior da Figura 1.

13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7
----	----	-----	----	----	-----	-----	----	----	----	----	----	-----	----	----	---

- Agora queremos encontrar o subvetor contíguo e não vazio de A cujos valores tenham a maior soma. Chamamos esse subarranjo contíguo de **subvetor máximo**.

Problema do Subvetor Máximo

- Por exemplo, no vetor anterior, o subvetor máximo de $A[1 \dots 16]$ é $A[8 \dots 11]$, com a soma 43. Assim, você gostaria de comprar a ação logo antes do dia 8 (que é, após o dia 7) e vendê-lo após o dia 11, obtendo um lucro de \$ 43 por ação.
- Como podemos solucionar agora o problema?
- Melhorou nossa complexidade ?

Problema do Subvetor Máximo

- Vamos pensar em como podemos resolver o problema do subarranjo máximo usando a técnica de dividir e conquistar. Suponha que queremos encontrar um subarranjo máximo do subarranjo $A[\text{low} .. \text{high}]$.
- Dividir e conquistar sugere que dividamos o subarranjo em dois subarranjos de tamanho tão igual quanto possível. Ou seja, encontramos o ponto médio, digamos, mid , do subarray, e consideramos os subarrays $A[\text{low} .. \text{mid}]$ e $A[\text{mid} + 1 .. \text{high}]$.

Problema do Subvetor Máximo

- Assim, qualquer subarranjo contíguo $A[i..j]$ de $A[\text{low}..\text{high}]$ deve estar exatamente em um dos seguintes locais:
 - inteiramente no subarranjo $A[\text{low}..\text{mid}]$, de modo que $\text{low} \leq i \leq j \leq \text{mid}$,
 - inteiramente no subarranjo $A[\text{mid} + 1 .. \text{high}]$, de modo que $\text{mid} < i \leq j \leq \text{high}$, ou
 - cruzando o ponto médio, de modo que $\text{low} \leq i \leq \text{mid} < j \leq \text{high}$.

Problema do Subvetor Máximo

- Portanto, um subarranjo máximo de $A[\text{low}..\text{high}]$ deve estar exatamente em um desses lugares. De fato, um subarray máximo de $A[\text{low}..\text{high}]$ deve ter a maior soma sobre todos os subarrays inteiramente em $A[\text{low}..\text{mid}]$, inteiramente em $A[\text{mid}+1..\text{high}]$, ou cruzando o ponto médio. Podemos encontrar subarranjos máximos de $A[\text{low}..\text{mid}]$ e $A[\text{mid}+1..\text{high}]$ recursivamente, porque esses dois subproblemas são instâncias menores do problema de encontrar um subarranjo máximo.

Problema do Subvetor Máximo

- Assim, tudo o que resta a fazer é encontrar um subvetor máximo que atravessa o ponto médio.
- Este problema não é uma instância menor do problema original e sim, é uma restrição que o subvetor escolhido pode cruzar o ponto médio.
- Podemos facilmente encontrar o subvetor máximo atravessando o ponto médio em tempo linear. O algoritmo a seguir resolve o problema.

Problema do Subvetor Máximo

algoritmo find_max_crossing_subarray(A,low,mid,high)

left_sum $\leftarrow -\infty$

sum $\leftarrow 0$

para i $\leftarrow$ mid até low passo -1

sum $\leftarrow$ sum + A[i]

se sum > left_sum então

left_sum $\leftarrow$ sum

left_max $\leftarrow$ i

right_sum $\leftarrow -\infty$

sum $\leftarrow 0$

para j $\leftarrow$ mid+1 até high

sum $\leftarrow$ sum + A[j]

se sum > right_sum então

right_sum $\leftarrow$ sum

right_max $\leftarrow$ j

Retorne(left_max, right_max, left_sum + right_sum)

Problema do Subvetor Máximo

algoritmo find_maximum_subarray(A, low, high)

se low==high então

retorne(low, high, A[low])

senão

 mid = (low+high)/2

 (left_low, left_high, left_sum) = find_maximum_subarray(A, low, mid)

 (right_low, right_high, right_sum) = find_maximum_subarray(A, mid+1, high)

 (cross_low, cross_high, cross_sum) = find_maximum_crossing_subarray(A, low, mid, high)

se left_sum ≥ right_sum e left_sum ≥ cross_sum então

Retorne(left_low, left_high, left_sum)

senão

se right_sum ≥ left_sum e right_sum ≥ cross_sum então

Retorne(right_low, right_high, right_sum)

senão

Retorne(cross_low, cross_high, cross_sum)

fim_algoritmo

Problema do Subvetor Máximo

- Analisando o algoritmo divisão e conquista.
 - Vamos construir uma equação, recorrência, que descreva o tempo de execução do algoritmo Find_Maximum_Subarray.
 - Para tal, vamos assumir que o tamanho da entrada, n , seja uma potência de 2 para simplificar o entendimento.
 - Denotamos por $T(n)$ o tempo de execução do algoritmo com uma instância de tamanho n .
 - O caso base do algoritmo ocorre quando $n=1$, e na segunda linha é dada a solução deste. Logo o custo é constante e temos que $T(1)=\Theta(1)$.

Problema do Subvetor Máximo

- O caso recursivo ocorre quando $n > 1$, e então temos:
 - As linhas 1 e 3 têm tempo constante;
 - Nas linhas 4 e 5 solucionam os subproblemas com subvetores contendo $n/2$ elementos cada e com tempo $T(n/2)$;
 - Na linha 6 é definido um subvetor máximo atravessando o ponto médio, que consome o tempo de $\Theta(n)$;
 - As linhas de 7 a 11 consomem um tempo constante; e assim
- A recorrência do caso recursivo é dado por
$$\begin{aligned}T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\ &= 2T(n/2) + \Theta(n)\end{aligned}$$

Problema do Subvetor Máximo

- Combinando caso base e caso recursivo temos a recorrência é dada por:

$$T(n) = \begin{array}{ll} 1 & \text{se } n=1 \\ 2T(n/2) + \Theta(n) & \text{se } n>1 \end{array}$$