Programação Dinâmica

André, Salles

Departamento de Informática
Universidade Federal de Viçosa

INF 333 - 2024/1

Max 1D Range Sum

- Entrada: uma sequência de inteiros quaisquer
- Objetivo: encontrar a subsequência consecutiva de soma máxima
- Exemplo: 3 -6 5 4 -3 5 -7 -2 3 -8 7 2

Max 1D Range Sum

- Seja $A[0..n-1]$ os n valores da entrada
- Seja $\textsc{RangeSum}(i, j)$ a soma $A[i] + \cdots + A[j]$

- Busca exaustiva em todos os $O(n^2)$ pares (i, j), calculando $\textsc{RangeSum}(i, j)$ em $O(n)$ para cada uma deles $\Rightarrow O(n^3)$

- Pré-computar $\textsc{RangeSum}(0, j)$ para cada j em $O(n)$ (basta acumular)
- com isso, $\textsc{RangeSum}(i, j)$ fica $O(1)$, pois vale $\textsc{RangeSum}(0, j) - \textsc{RangeSum}(0, i-1)$
- assim, calcular para todos os pares $(i, j) \Rightarrow O(n^2)$

Max 1D Range Sum

É possível em $O(n)$, com o algoritmo de Kadane

- Adicionar os elementos um por um, mantendo a soma acumulada
- Resetar a soma para zero, quando for negativa
  (pois 0 é melhor que qualquer soma negativa)
- Retornar a maior soma encontrada

Max 2D Range Sum

- Entrada: uma matriz de inteiros quaisquer
- Objetivo: encontrar a submatriz de soma máxima
- Exemplo:

0	−2	−7	0
9	2	−6	2
−4	1	−4	1
−1	8	0	−2

Max 2D Range Sum

- Seja $A[0..n-1, 0..m-1]$ os $n \times m$ valores da entrada
- $\textsc{RangeSum}(a, b, i, j)$: soma da submatriz de cantos (a, b) e (i, j)
- Busca exaustiva em todos as $O(n^4)$ submatrizes, calculando em $O(n^2)$ a soma de cada uma delas $\Rightarrow O(n^6)$
- Pré-computar $\textsc{RangeSum}(0, 0, i, j)$ para cada para i, j em $O(n^2)$ (acumulando as anteriores, vide matriz B abaixo)
- com isso, $\textsc{RangeSum}(a, b, i, j)$ fica $O(1)$ ($RangeSum(1, 2, 3, 3) = -3 - (13) - (-9) + (-2)$, vide matriz C)

B

0	-2	-9	-9
9	9	-4	2
5	6	-11	-8
4	13	-4	-3

C

0	-2	-9	-9
9	9	-4	2
5	6	-11	-8
4	13	-4	-3

- assim, calcular para todos os casos $(a, b, i, j) \Rightarrow O(n^4)$

Obs.: é possível em $O(n^3)$ com ideia semelhante ao algoritmo de Kadane

LIS – Longest Increasing Subsequence

- Dada uma sequência A de n números, encontrar a maior subsequência (não necessariamente consecutiva) crescente
- Exemplo: -7 10 9 2 3 8 8 1 – tamanho 4

- Seja LIS(i) a maior subsequência que termina na posição i
  - caso base: $\text{LIS}(0) = 1$
  - caso recursivo: $\text{LIS}(i) = 1 + \max(\text{LIS}(j)), \forall j < i$ com $A[j] < A[i]$
  - assim não se perde tempo com sobreposição $\Rightarrow O(n^2)$
  - o tamanho da maior LIS é o maior valor de LIS
  - a LIS em si pode ser recuperada guardando-se o predecessor de i na LIS(i)

Index	0	1	2	3	4	5	6	7
X	-7	10	9	2	3	8	8	1
LIS(i)	1	2	2	2	3	4	4	2

LCS – Longest Common Subsequence

- Dadas duas sequências, encontrar a maior subsequência comum
- Exemplo: `republicano`, `democrata` – tamanho 3: `eca`

- $\mathrm{LCS}(i,j)$: tamanho da LCS até posição i da 1ª sequência e j da 2ª
- caso base: $\mathrm{LCS}(0,j) = 0$ e $\mathrm{LCS}(i,0) = 0$
- caso recursivo:
  - $\mathrm{LCS}(i,j) = 1 + \mathrm{LCS}(i-1, j-1)$, se $A[i] = B[j]$
  - $\mathrm{LCS}(i,j) = \max\{\mathrm{LCS}(i-1,j), \mathrm{LCS}(i,j-1)\}$, senão

Edit distance

- Dadas duas strings, determinar a distância de edição entre elas
- A distância é o mínimo de edições para transformar uma na outra
- Cada edição pode ser inserir, deletar ou substituir um caractere
- Exemplo: `intention`, `execution` – distância 5

```
inte*ntion
*execution
dss=is====
```

deletar `i`, substituir `n` por `e`, substituir `t` por `x`, inserir `c` e substituir `n` por `u`

Edit distance

- Duas strings, A de tamanho n e B de tamanho m
- $\textsc{Dist}(i, j)$ é a distância de $A[1..i]$ a $B[1..j]$
  isto é, dos primeiros i caracteres de A e j de B
- Queremos encontrar $\textsc{Dist}(n, m)$

- caso base: $\textsc{Dist}(0, j) = j$ e $\textsc{Dist}(i, 0) = i$
- casos recursivos:
  - Se $A[i] = B[j]$, $\textsc{Dist}(i, j) = \textsc{Dist}(i-1, j-1)$
  - Senão, $\textsc{Dist}(i, j) = \min \begin{cases} \textsc{Dist}(i, j-1) + 1, & \text{inserir } B[j] \\ \textsc{Dist}(i-1, j) + 1, & \text{deletar } A[i] \\ \textsc{Dist}(i-1, j-1) + 1, & \text{subst. } A[i] \to B[j] \end{cases}$

Edit distance

- Facilmente implementado *bottom-up*
- `dist` é uma matriz $(n+1) \times (m+1)$
  (strings alinhados a partir da posição 1; posição 0 para string vazio)
-
```
for i = 0 to n
   for j = 0 to m
      dist[i,j] = ... #conforme definicao anterior
```
- A solução estará em `dist[n,m]`

Edit distance

- Fácil adaptar se custos de inserção, deleção e substituição são $\neq$
- Em particular, resolve LCS se custo substituir $>$ inserir $+$ deletar

Mais exemplos de aplicação

UVa 10943: How do you add? (link)

- Dado um inteiro N, de quantas maneiras K números $\leq N$ somam N?
- Restrições: $1 \leq N, K \leq 100$
- Exemplo: para $N = 20$ e $K = 2$, há 21 maneiras: 0+20, 1+19, ..., 20+0

- Existe uma fórmula fechada, mas pode ser resolvido por *PD*
- Que parâmetro usar para os estágios? N ou K? precisamos dos dois...
  - usando apenas N, não saberemos quantos números já usados
  - usando apenas K, não saberemos qual soma devemos atingir
- Caso base: $K = 1$, não interessa N, só há uma forma, usar o próprio N
- Caso geral: $K > 1$, podemos dividir N em X e $N - X$, com $X \in [0 \ldots N]$
  - Ficamos então com os subproblemas $N - X, K - 1$
  - Basta somar seus valores: $DP(N, K) = \sum_{X=0}^{N} DP(N - X, K - 1)$

Mais exemplos de aplicação

UVa 10003: Cutting Sticks (link)

- Dada uma barra de tamanho L e n pontos de corte (coordenadas [1..L-1]), determinar o menor custo para cortar nesses pontos
- O custo de cortar é o tamanho do pedaço sendo cortado
- Restrições: $1 \leq L < 1000$, $1 \leq n \leq 50$
- Exemplo: $L = 100, n = 3 : 25, 50, 75$
  - Cortando na ordem $25, 50, 75$: custo $100 + 75 + 50 = 225$
  - Solução: cortar na ordem $50, 25, 75$, custo $100 + 50 + 50 = 200$

- Força bruta: são $n!$ possíveis ordens de corte, TLE
- PD: os pedaços podem ser identificados pelos seus extremos
  - $1 \ldots n$: pontos de corte; 0 e $n + 1$: extremid. inicial e final da barra
  - $\textsc{Custo}(i, j)$ menor custo para cortar o pedaço $[i, j]$
    - Base: $\textsc{Custo}(i, i + 1) = 0$ *(pedaço que não precisa mais ser cortado)*
    - Rec.: $\textsc{Custo}(i, j) = \min_{k=i+1}^{j-1}\{\textsc{Custo}(i, k) + \textsc{Custo}(k, j) + A[j]\text{-}A[i]\}$ *(corte no ponto k: custo de cortar pedaços resultantes + custo desse corte)*

Mais exemplos de aplicação

Elevator Optimization (*do livro Programming Challenges*)

Elevador faz P paradas a partir do térreo; quem vai para andar que ele não para deve descer ou subir andando a partir da parada mais próxima; minimizar o "custo" (total de andares que devem subir/descer andando).

- Ex: 10 andares, pessoas no térreo (0), uma para cada andar
  - 1 parada: 7 – custo 18
  - 2 paradas: 3, 8 – custo 11
  - 3 paradas: 3, 6, 9 – custo 7
- $M(i, j)$: custo mínimo de j paradas com a última no andar i
- Como acrescentar mais uma parada de forma ótima?
- A parada $j + 1$ deve estar depois da parada j, que foi no andar i
- Ela não é interessante para ninguém que desce em i ou abaixo
- Quem quer descer depois de i pode descer em i ou na nova
- $M(i, j + 1) = \min_{k=0}^{i}\{M(k, j) - anda(k, \infty) + anda(k, i) + anda(i, \infty)\}$[1]

[1] *anda*(a, b) é o custo total de todos com destino entre as paradas nos andares a e b