Contents

Chapter 1

Otimização de Knuth-Yao

1.1 Avisos (Temporário)

Eu estou usando algumas notações que eu não tenho certeza se são razoáveis.

- $[i..j] := \{k \in \mathbb{N} \mid i \leq k \leq j\}$ (um intervalo)
- $[n] := [1..n]$ (o intervalo de um até n)
- $[[n]] := \{[i..j] \mid i, j \in \mathbb{N} \text{ e } 1 \leq i \leq j \leq n\}$ (todos os subintervalos de $[1..n]$)

Além disso, eu defino funções sobre intervalos (exemplo: $f : [[n]] \to \mathbb{R}$) e depois uso elas como se fossem funções sobre $\mathbb{N}^2$ (exemplo: $f(i, j)$ ao invés de $f[i, j]$). Tudo bem?

1.2 Concatenação de Custo Mínimo

Para apresentar a técnica da otimização de Knuth-Yao, vamos introduzir o problema da *Concatenação de Custo Mínimo*. O problema consiste em um inteiro n, um vetor $v \in \mathbb{R}_+^n$ e duas operações:

1. Criar um novo vetor unitário $x \in \mathbb{R}_+$. Esta operação tem custo x.
2. Concatenar dois vetores $a \in \mathbb{R}^p$ e $b \in \mathbb{R}^q$ já existentes. Esta operação tem custo $\sum_{i=1}^{p} a_i + \sum_{i=1}^{q} b_i$.

Queremos realizar uma sequência destas operações de forma a obter um vetor idêntico a v. Dentre todas as possíveis, queremos a sequência de menor custo possível.

Precisamos de duas observações, qualquer vetor de tamanho 1 deve ser gerado com a primeira operação e qualquer vetor de tamanho maior do que 1 deve ser obtido pela concatenação de dois outros vetores um prefixo dele e outro sufixo dele. Isso nos diz que todos os vetores intermediários necessários para gerar v de maneira ótima são subvetores[1] de v, já que se um vetor não é subvetor de v ele nunca vai ajudar a gerar v. Com isso, concluímos uma recorrência que nos dá o custo mínimo necessário para gerar v.

$$f(i, j) = \begin{cases} v_i & \text{se } i = j, \\ \min\limits_{i \leq k < j} \Big\{ f(i, k) + f(k+1, j) \Big\} + \sum\limits_{k=i}^{j} v_k & \text{c.c.} \end{cases}$$

[1]Um vetor v é dito subvetor de um vetor u se existem índices i,j tais que $u[i..j] = v$

A recorrência acima pode ser resolvida facilmente com programação dinâmica em tempo $O(n^3)$.

Algorithm 1 Concatenação de Custo Mínimo $O(n^3)$

```
função MINCOSTCONCAT(v, n)
    devolve resposta
```

Vamos apresentar uma técnica que nos ajuda a resolver este problema em tempo $O(n^2)$ e pode ser adaptada para vários outros problemas.

1.3 A Desigualdade Quadrangular

Para ajudar nisso, vamos escrever f de uma maneira mais genérica.

Definição 1.1 (Recorrência de Intervalos). *Dizemos que uma recorrência $f : [[n]] \to \mathbb{R}$ é de intervalos se existe, para todo $k \in [1..n]$, uma função $f_k : \{[i..j] \in \mathbb{N}^2 \mid 1 \leq i \leq k < j \leq n\} \to \mathbb{R}$ que depende apenas de $f_{k'}(i', j')$ onde $[i', j'] \subset [i, j]$ e $i' \leq k' < j'$ e uma função $f_\bullet : [[n]] \to \mathbb{R}$ tais que*

$$f(i,j) = \begin{cases} f_\bullet(i,j) & \text{se } i = j, \\ \min\limits_{i \leq k < j} \left\{ f_k(i,j) \right\} + f_\bullet(i,j) & \text{c.c.} \end{cases}$$

Além disso, chamamos $f_k(i,j)$ de corte do estado (i,j) no ponto k e $f_\bullet(i,j)$ de constante do estado (i,j).

Se definirmos $f_\bullet : [i,j] \in [[n]] \mapsto \sum\limits_{k=i}^{j} v_k$ e, para todo $k \in [n]$, $f_k : [i,j] \in \{\mathbb{N}^2 \mid 1 \leq i \leq k < j \leq n\} \mapsto f(i,k) + f(k+1,j)$, temos f escrita como recorrência de intervalos.

Definição 1.2 (Monótono nos Intervalos). *Dizemos que uma função $w : [[n]] \to \mathbb{R}$ é monótona nos intervalos se, para todo $[i', j'] \subseteq [i,j] \in [[n]]$,*

$$w(i', j') \leq w(i,j)$$

Proposição 1.3. *f é monótona nos intervalos.*

Proof. Prova vazia. □

Definição 1.4 (Desigualdade Quadrangular). *Dizemos que uma função $w : [[n]] \to \mathbb{R}$ respeita a desigualdade quadrangular se para todo $i, i', j, j' \in \mathbb{N}$ tais que $1 \leq i \leq i' \leq j \leq j' \leq n$ vale*

$$w(i,j) + w(i', j') \leq w(i, j') + w(i', j)$$

Proposição 1.5. *f respeita a desigualdade quadrangular.*

Proof. Prova vazia. □

Definimos $f_* : [i,j] \in [[n]] \mapsto \min\limits_{i \leq k < j} \left\{ k \mid f(i,j) = f_k(i,j) \right\}$, ou seja, o ponto de corte do estado (i,j) que gera uma resposta ótima e tem o menor índice. Para poder otimizar o código que usamos para calcular f, queremos rovar e explorar a seguinte propriedade sobre esta recorrência

$$f_*(i, j-1) \leq f_*(i,j) \leq f_*(i+1, j), \text{ para todo } [i,j] \in [[n]] \tag{1.6}$$

Proposição 1.7. *Se uma recorrência f é uma recorrência de intervalos (1.1) que respeita 1.4 e é monótona nos intervalos (1.2), vale 1.6.*

A proposição 1.7 (apresentada de forma diferente) foi provada em Yao, 1980. Vamos apresentar aqui uma versão adaptada da prova.

Proof. Prova vazia. □

1.4 Otimização

Sabendo que para a função f vale 1.6, podemos alterar o algoritmo 1 gerando um novo algoritmo de complexidade $O(n^2)$ para calcular f. Se calcularmos, junto com f, os valores de f_*, podemos limitar os testes feitos para calcular cada estado de f. É importante notar que para todos estados (i, j) com $i < j$, precisamos conhecer o valor de $f_*(i+1, j)$ e $f_*(i, j-1)$ antes de calcular $f(i, j)$, ou seja, os estados $(i+1, j)$ e $(i, j-1)$ devem ser calculados antes de (i, j). Essa restrição não ocorria na ultima versão deste algoritmo. Para resolver isso, basta iterar pelos estados em ordem de tamanho, ou seja, primeiro calculamos todos os estados (i, j) onde $j - i = 1$, depois aqueles onde $j - i = 2$ e assim por diante.

Algorithm 2 Concatenação de Custo Mínimo $O(n^2)$

1: **devolve** resposta

Proposição 1.8. *O algoritmo 2 tem tempo $O(n^2)$.*

Bibliography

Yao, F. Frances (1980). “Efficient Dynamic Programming Using Quadrangle Inequalities”. In: *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*. STOC ’80. Los Angeles, California, USA: ACM, pp. 429–435. ISBN: 0-89791-017-6. DOI: 10.1145/800141.804691. URL: http://doi.acm.org/10.1145/800141.804691.