

# Contents

<b>1</b>	<b>Otimização de Knuth-Yao</b>	<b>1</b>
1.1	Concatenação de Custo Mínimo . . . . .	1
1.2	A Desigualdade Quadrangular . . . . .	3
1.3	Otimização . . . . .	4
	<b>Bibliography</b>	<b>7</b>



## Chapter 1

# Otimização de Knuth-Yao

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

## 1.1 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<sup>1</sup> 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  $v$ .

$$f(i, j) = \begin{cases} v_i & \text{se } i = j, \\ \min_{i \leq k < j} \left\{ f(i, k) + f(k+1, j) \right\} + \sum_{k=i}^j v_k & \text{c.c.} \end{cases}$$

<sup>1</sup>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  $O(n^3)$ . Consideramos que os valores de  $\sum_{k=1}^j v_k$  estão pré calculados para todo  $j \leq n$ , assim, conseguimos calcular  $\sum_{k=i}^j v_k$  para todo par  $i, j$  onde  $1 \leq i \leq j \leq n$  em tempo  $O(1)$ .

---

**Algoritmo 1.1** Concatenação de Custo Mínimo  $O(n^3)$ 


---

```

1: função MINCOSTCONCAT( $v, n$ )
2:   para  $i$  de 1 até  $n$  faça
3:      $f(i, i) = v_i$ 
4:   para  $i$  de 1 até  $n$  faça
5:     para  $j$  de  $i + 1$  até  $n$  faça
6:        $f(i, j) = \inf$ 
7:         para  $k$  de  $i$  até  $j - 1$  faça
8:            $f(i, j) = \min\{f(i, j), f(i, k) + f(k + 1, j)\}$ 
9:        $f(i, j) += \sum_{k=i}^j v_k$ 
10:  devolve  $f(1, n)$ 

```

---

Vamos apresentar uma técnica que nos ajuda a resolver este problema em tempo  $O(n^2)$  e pode ser adaptada para vários outros problemas. Para ajudar nisso, vamos escrever  $f$  de uma maneira mais genérica.

**Definição 1.2** (Recorrência de Intervalos). Dizemos que uma recorrência  $f : [[n]] \rightarrow \mathbb{R}$  é de intervalos se existe, para todo  $k \in [1..n]$ , uma função  $f_k : \{[i..j] \in \mathbb{N}^2 \mid 1 \leq i \leq k < j \leq n\} \rightarrow \mathbb{R}$  que depende apenas de  $f_{k'}(i', j')$  onde  $[i'..j'] \subset [i..j]$  e  $i' \leq k' < j'$  e uma função  $f_\bullet : [[n]] \rightarrow \mathbb{R}$  tais que

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

Além disso, chamamos as funções  $f_k(i, j)$  de funções de corte da recorrência e a função  $f_\bullet$  de função custo da recorrência.

Se definirmos  $f_\bullet : [i..j] \in [[n]] \mapsto \sum_{k=i}^j v_k$  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 1.3** (Monótono nos Intervalos). Dizemos que uma função  $w : [[n]] \rightarrow \mathbb{R}$  é monótona nos intervalos se, para todo  $[i'..j'] \subseteq [i..j] \in [[n]]$ ,

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

**Proposição 1.4.**  $f$  é monótona nos intervalos.

*Proof.* É fácil ver que  $f_\bullet$  é monótona nos intervalos, pois todos os elementos de  $v$  são positivos.

Queremos provar que, para todo intervalo  $[i..j] \in [[n]]$  vale que se  $[i'..j'] \subseteq [i..j]$ , então  $f(i', j') \leq f(i, j)$ . Vamos usar indução no tamanho do intervalo, ou seja, indução

no valor de  $j - i + 1$ . Inicialmente, assumimos  $j - i + 1 = 1$ , isto é,  $j = i$ . Já que o único intervalo contido num intervalo de tamanho 1 é ele mesmo, vale a tese. Agora, assumamos que  $j - i + 1 = d > 1$ .

A última operação feita para gerar  $v[i..j]$  é do tipo 2, já que  $v[i..j]$  tem tamanho maior do que 1, ou seja, existe um  $k \in [i..j-1]$  tal que  $f(i, j) = f(i, k) + f(k+1, j) + f_\bullet(i, j)$ . Temos três casos. Se  $j' \leq k$ , então  $[i'..j'] \subseteq [i..k]$ , já que  $k - i + 1 < j - i + 1$ , aplicamos a hipótese de indução e podemos afirmar que  $f(i', j') \leq f(i, k)$ , portanto,  $f(i', j') \leq f(i', j') + f(k+1, j) + f_\bullet(i, j) \leq f(i, k) + f(k+1, j) + f_\bullet(i, j) = f(i, j)$ . O caso onde  $i' > k$  implica em  $[i'..j'] \subseteq [k+1..j]$  e, de maneira análoga, obtemos  $f(i', j') \leq f(k+1, j)$  e, portanto,  $f(i', j') \leq f(i, j)$ .

No último caso,  $i' \leq k < j'$ . Mas, então, uma forma válida de gerar o vetor  $v[i'..j']$  é concatenar os vetores  $v[i'..k]$  e  $v[k+1..j']$ . Isso quer dizer que  $f(i', j') \leq f(i', k) + f(k+1, j') + f_\bullet(i', j')$ . Podemos aplicar a hipótese de indução para afirmar que  $f(i', k) \leq f(i, k)$  e  $f(k+1, j') \leq f(k+1, j)$ . Além disso, já que  $f_\bullet$  é monótona nos intervalos,  $f_\bullet(i', j') \leq f_\bullet(i, j)$ , portanto,  $f(i', k) + f(k+1, j') + f_\bullet(i', j') \leq f(i, k) + f(k+1, j) + f_\bullet(i, j) = f(i, j)$ .  $\square$

**Definição 1.5** (Corte Ótimo). Se  $f : [[n]] \rightarrow \mathbb{R}$  é uma recorrência de intervalos, a função  $f_* : [i..j] \in [[n]] \mapsto \min_{i \leq k < j} \{k \mid f(i, j) = f_k(i, j)\}$  é chamada de corte ótimo de  $f$ . Ela representa, para cada estado  $[i..j]$ , o ponto de corte 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 os cortes ótimos da recorrência.

**Definição 1.6** (Knuth-Yao Otimizável). Uma recorrência de intervalos  $f$  é Knuth-Yao otimizável se a sua função de cortes ótimos  $f_*$  é tal que

$$f_*(i, j') \leq f_*(i, j) \leq f_*(i', j), \text{ para todo } [i'..j'] \subseteq [i..j] \in [[n]].$$

Queremos conseguir provar que  $f$  é Knuth-Yao Otimizável para podermos alterar o algoritmo 1.1 gerando um novo algoritmo de complexidade  $O(n^2)$  para calcular  $f$ . Para ajudar nesta prova, vamos apresentar a desigualdade quadrangular e alguns resultados sobre ela.

## 1.2 A Desigualdade Quadrangular

**Definição 1.7** (Desigualdade Quadrangular). Dizemos que uma função  $w : [[n]] \rightarrow \mathbb{R}$  respeita a desigualdade quadrangular se para todo  $[i', j'] \subseteq [i, j] \in [[n]]$  tais que  $1 < i < i' \leq j \leq j' \leq n$  vale

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

Foram apresentados em Yao, 1980 alguns fatos sobre a Desigualdade Quadrangular que serão enunciados e provados aqui.

**Teorema 1.8.** Se uma recorrência  $f$  é uma recorrência de intervalos que respeita a Desigualdade Quadrangular e é monótona nos intervalos, vale 1.6.

*Proof.* Prova vazia.  $\square$

**Teorema 1.9.** Se  $f$  é uma recorrência de intervalos com função de custo  $f_\bullet$  e  $f_\bullet$  respeita a Desigualdade Quadrangular, então,  $f$  respeita a Desigualdade Quadrangular.

*Proof.* Prova vazia.  $\square$

O Yao usa  $i \leq i' \leq j \leq j'$ , eu acho que em termos de intervalos fica bem menos confuso

### 1.3 Otimização

Agora, com os teoremas sobre desigualdade quadrangular, podemos provar que  $f$  é Knuth-Yao Otimizável.

**Proposição 1.10.**  $f$  é Knuth-Yao otimizável.

*Proof.* Primeiramente, vamos mostrar que vale a desigualdade quadrangular para  $f_\bullet$ . Sejam  $[i', j'] \subseteq [i, j] \in [[n]]$ . Se  $i < i' < j' < j$ , temos  $f_\bullet(i', j) + f_\bullet(i, j') = f_\bullet(i', j') + f_\bullet(j' + 1, j) + f_\bullet(i, i') + f_\bullet(i' + 1, j') = f_\bullet(i', j') + f_\bullet(i, j)$ . Se  $i < i' = j' < j$ , temos  $f_\bullet(i', j) + f_\bullet(i, j') = f_\bullet(i', j) + f_\bullet(i, i') = f_\bullet(i, j) + f_\bullet(i', j')$ . Os outros casos  $i = j, i = i'$  e  $j = j'$  valem trivialmente.

Pelo Teorema 1.9, a desigualdade quadrangular vale para  $f$ . Pelo Teorema 1.8,  $f$  é Knuth-Yao otimizável.  $\square$

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.

---

#### Algoritmo 1.11 Concatenação de Custo Mínimo $O(n^2)$

---

```

1: função MINCOSTCONCATOPT( $v, n$ )
2:   para  $i$  de 1 até  $n$  faça
3:      $f(i, i) = v_i$ 
4:      $f_*(i, i) = i$ 
5:   para  $d$  de 1 até  $n - 1$  faça
6:     para  $i$  de  $i$  até  $n - d$  faça
7:        $j = i + d$ 
8:        $f(i, j) = \inf$ 
9:         para  $k$  de  $f_*(i, j - 1)$  até  $\min\{f_*(i + 1, j), j - 1\}$  faça
10:           $l = f(i, k) + f(k + 1, j)$ 
11:          se  $l < f(i, j)$  então
12:             $f(i, j) = l$ 
13:             $f_*(i, j) = k$ 
14:           $f(i, j) + = \sum_{k=i}^j v_k$ 
15:   devolve  $f(1, n)$ 

```

---

Note que já que as observações feitas para o problema da Concatenação de Custo Mínimo foram feitas em termos de recorrências de intervalos, qualquer  $f$  que seja uma recorrência de intervalos e respeite 1.6 pode ser resolvida com uma versão adaptada do algoritmo 1.11. Consideramos então tal  $f$  com funções de corte  $f_k$  e contantes  $f_\bullet$ .

**Algoritmo 1.12** Otimização de Knuth-Yao

---

```

1: função KNUTHYAO( $f_{\bullet}, f_k \forall k \in [n], n$ )
2:   para  $i$  de 1 até  $n$  faça
3:      $f(i, i) = v_i$ 
4:      $f_*(i, i) = i$ 
5:   para  $d$  de 1 até  $n - 1$  faça
6:     para  $i$  de  $i$  até  $n - d$  faça
7:        $j = i + d$ 
8:        $f(i, j) = \inf$ 
9:         para  $k$  de  $f_*(i, j - 1)$  até  $\min\{f_*(i + 1, j), j - 1\}$  faça
10:          se  $f_k(i, j) < f(i, j)$  então
11:             $f(i, j) = f_k(i, j)$ 
12:             $f_*(i, j) = k$ 
13:        $f(i, j) + = f_{\bullet}(i, j)$ 
14:   devolve  $f(1, n)$ 

```

---

**Proposição 1.13.** O algoritmo 1.12 faz  $O(n^2)$  chamadas a  $f_{\bullet}$ ,  $f_*$  e  $f$ .

*Proof.* Prova vazia. □

Já que o algoritmo 1.11 é uma especificação de 1.12, ele tem complexidade  $O(n^2)$  já que cada chamada a  $f_{\bullet}$  tem custo  $O(1)$ , assim como as chamadas a  $f_*$  e  $f$  que são memorizadas.





# 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](https://doi.org/10.1145/800141.804691). URL: <http://doi.acm.org/10.1145/800141.804691>.