

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 subvetores¹ 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 .

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

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 $O(1)$.

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

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 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 $k \in [n]$, $f_k : [i..j] \in \{[i..j] \in \mathbb{N}^2 \mid 1 \leq i \leq k < j \leq n\} \mapsto f(i, k) + f(k + 1, j)$, temos f escrita como recorrência de intervalos.

Definição 3 (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 4 (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 5 (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 $1 \leq i \leq i' \leq j \leq j' \leq n$ vale*

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

Foram apresentados em [1] alguns fatos sobre a Desigualdade Quadrangular que serão enunciados e provados aqui.

Teorema 6. *Se uma recorrência f é uma recorrência de intervalos que respeita a Desigualdade Quadrangular, vale 4.*

Demonstração. Seja f uma recorrência de intervalos que respeita a Desigualdade Quadrangular. Sejam $[i', j'] \subseteq [i, j] \in [[n]]$.

Vamos provar que vale $f_*(i, j') \leq f_*(i, j)$. Suponha que vale $j' > c = f_*(i, j')$, temos

$$f_c(i, j') = f(i, c) + f(c + 1, j') < f(i, b) + f(b + 1, j') = f_b(i, j') \text{ e} \\ f(i, b) + f(b + 1, j) \leq f(i, c) + f(c + 1, j).$$

Somando as duas equações, obtemos

$$f(b + 1, j) + f(c + 1, j') < f(b + 1, j') + f(c + 1, j),$$

porém, já que $[c + 1, j'] \subseteq [b + 1, j] \in [[n]]$, esta afirmação contradiz a Desigualdade Quadrangular. Portanto, por absurdo, vale $f_*(i, j') \leq f_*(i, j)$. De forma parecida, vamos provar que vale $f_*(i, j) \leq f_*(i', j)$. Suponha que vale $j > c = f_*(i, j) > f_*(i', j) = b \geq i'$, temos

$$f_c(i, j) = f(i, c) + f(c + 1, j) < f(i, b) + f(b + 1, j) = f_b(i, j) \text{ e} \\ f_b(i', j) = f(i', b) + f(b + 1, j) \leq f(i', c) + f(c + 1, j) = f_c(i', j).$$

Somamos as duas, obtendo

$$f(i', b) + f(i, c) < f(i, b) + f(i', c),$$

mas $[i', b] \subseteq [i, c] \in [[n]]$, portanto, esta afirmação contradiz a Desigualdade Quadrangular. Por absurdo, vale $f_*(i, j) \leq f_*(i', j)$. \square

Definição 7 (Monótono nos Intervalos). *Dizemos que uma função $w : [[n]] \rightarrow \mathbb{R}$ é monótona nos intervalos se, para todo $[i'..j'] \subseteq [i..j] \in [[n]]$,*

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

Teorema 8. *Se f é uma recorrência de intervalos com função de custo f_\bullet , f é monótona nos intervalos e f_\bullet respeita a Desigualdade Quadrangular, então, f respeita a Desigualdade Quadrangular.*

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

Demonstração. Prova vazia. \square

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

Proposição 9. f é monótona nos intervalos.

Demonstração. 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 ultima 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

Proposição 10. f é Knuth-Yao otimizável.

Demonstração. Primeiramente, vamos mostrar que vale a desigualdade quadrangular para f_\bullet . Sejam $[i', j'] \subseteq [i, j] \in [[n]]$,

$$f_\bullet(i, j') + f_\bullet(i', j) = \sum_{k=i}^{j'} v_k + \sum_{k=i'}^j v_k = \sum_{k=i}^j v_k + \sum_{k=i'}^{j'} v_k = f_\bullet(i, j) + f_\bullet(i', j').$$

Pelo Teorema 8, a desigualdade quadrangular vale para f . Pelo Teorema 6, 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 1 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 reccorência de intervalos e respeite 4 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 1 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 13. *O algoritmo 1.12 faz $O(n^2)$ chamadas a f_\bullet e qualquer das f_k .*

Demonstração. As chamdas a f_\bullet estão todas na linha 13. Podemos expressar a quantidade de ocorrências desta linha da seguinte maneira

$$\sum_{d=1}^{n-1} \sum_{i=1}^{n-d} 1 = n(n-1) - \sum_{d=1}^{n-1} d = O(n^2).$$

As chamdas a f_k , para todo k , estão todas nas linhas 10, 11. Podemos expressar a quantidade máxima de ocorrências destas linhas da seguinte maneira:

$$\sum_{d=1}^{n-1} \sum_{i=1}^{n-d} \sum_{k=f_*(i,i+d-1)}^{f_*(i+1,i+d)} 2 = \sum_{d=1}^{n-1} \sum_{i=1}^{n-d} 2(f_*(i+1, i+d) - f_*(i, i+d-1) + 1) \leq \sum_{d=1}^{n-1} 4n = O(n^2)$$

□

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.

REFERÊNCIAS

- [1] F. Frances Yao. Efficient dynamic programming using quadrangle inequalities. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC '80, pages 429–435, New York, NY, USA, 1980. ACM.