MAC e MVC em tempo $\langle O(n), O(1) \rangle$

Alair Pereira do Lago Imre Simon

Julho de 2003
Colóquio Brasiliero de Matemática

Apresentação

A combinatória das palavras é área bastante rica e diversas aplicações têm se desenvolvido ao longo dos anos auxiliando áreas da Matemática como

- teoria dos grupos, dos semigrupos,
- autômatos e linguagens formais
- desenvolvimento de algoritmos que possibilitem a análise, processamento e extração de informações em seqüências cada vez mais compridas,
  - ▷ textos disponíveis na internet
  - ▷ bibliotecas digitais
  - ▷ seqüências genômicas

algoritmos eficientes têm sido cada vez mais requisitados.

O problema do Menor Ancestral Comum em Árvores

Dado um vértice u de uma árvore existe um único caminho da raiz até u.

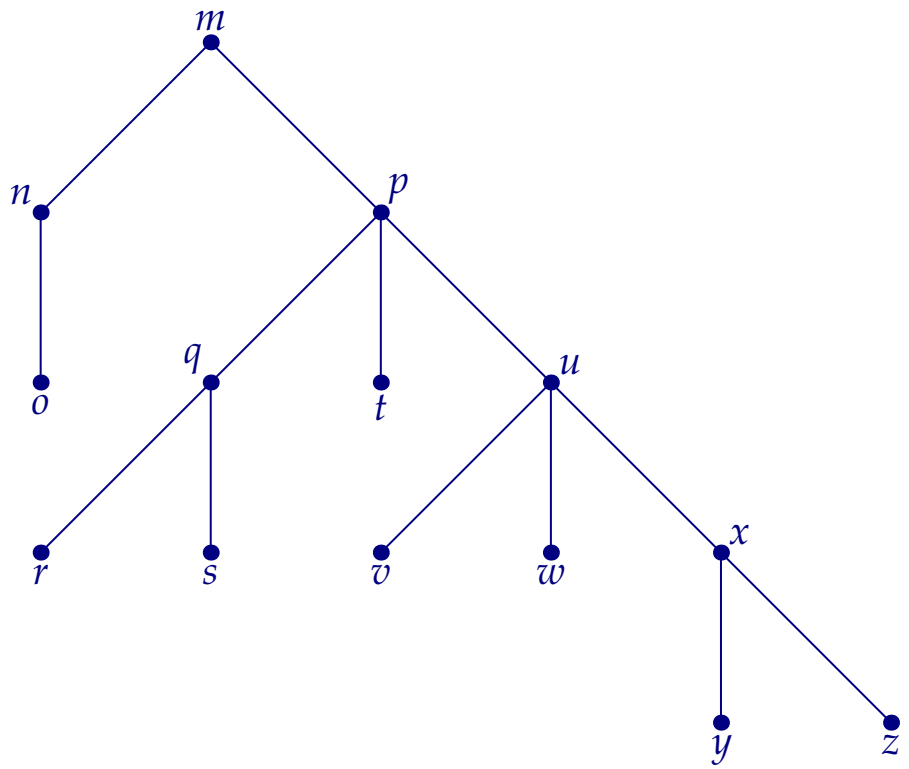
Qualquer vértice neste caminho é dito um ancestral de u.

Dados dois vértices, u e v numa árvore, eles sempre tem ancestrais comuns dado que a raiz é um exemplo de um ancestral comum.

Denominamos de menor ancestral comum de u e v o ancestral comum de u e de v que seja a mais distante da raiz.

- (O adjetivo menor refere-se ao ancestral comum "menos antigo" de u e de v).
- Denotaremos por $\mathsf{MAC}(u, v)$ o menor ancestral comum de u e v.

Exemplos de MACs



Por exemplo, na Figura :

$$\mathsf{MAC}(w, y) = u, \quad \mathsf{MAC}(s, z) = p, \quad \mathsf{MAC}(o, z) = m.$$

Notação de complexidade com pré-processamento

Os algoritmos do nosso interesse terão sempre duas partes bem distintas e sua complexidade será expressa em termos de um par ordenado de complexidades, como por exemplo em $\langle O(n), O(1) \rangle$.

A primeira complexidade, $O(n)$ no caso, refere-se ao tempo usado por um algoritmo de pré-processamento.

A segunda componente, $O(1)$ no caso, refere-se ao tempo usado para responder uma instância do problema, dado que o pré-processamento foi feito (e suas estruturas de dados estão disponíveis).

Desta forma, uma solução $\langle O(n), O(1) \rangle$ do problema de MAC implica que a árvore de n vértices sofre um pré-processamento linear. A partir deste pré-processamento qualquer instância do problema pode ser resolvida em tempo constante $(O(1))$, independente do tamanho da árvore.

Reflexão sobre solução $\langle O(n), O(1) \rangle$

Daremos as idéias principais que embasam uma solução $\langle O(n), O(1) \rangle$ do problema do Menor ancestral comum.

Vale a pena refletir sobre este resultado...

Pré-processamento que demora um tempo constante em cada vértice da árvore dada.

Faz-se anotações que poderão ser consultadas mais adiante. Tabelas de tamanho $O(n)$.

Feito o pré-processamento, qualquer instância do problema pode ser respondida em tempo constante, qualquer que tenha sido o tamanho da árvore envolvida.

Busca em profundidade de árvores

Dada uma árvore $\mathbf{T}$ com n vértices, calculamos um vetor com $m = 2n - 1$ vértices, que descreve, passo a passo, a busca em profundidade de $\mathbf{T}$.

Chamaremos este vetor de Busca, estando o grafo $\mathbf{T}$ subentendido.

O índice dos elementos serve também como um parâmetro de "cronometragem" do algoritmo da busca em profundidade: se Busca$[i] = u$ então dizemos que a busca visitou o vértice u no instante i.

Calcularemos, também, o vetor Prof, indexado de 1 a m, cujos elementos serão, respectivamente, as profundidades dos vértices em Busca.

Registraremos em Desc$[u]$ o instante da primeira visita ao vértice u e em Aban$[u]$ o instante da última visita a u. Estes instantes são, respectivamente, a descoberta e o abandono de u.

Exemplo de busca em profundidade

i	1	2	3	4	5	6	7	8
Busca$[i]$	m	n	o	n	m	p	q	r
Prof$[i]$	0	1	2	1	0	1	2	3

i	9	10	11	12	13	14	15	16
Busca$[i]$	q	s	q	p	t	p	u	v
Prof$[i]$	2	3	2	1	2	1	2	3

i	17	18	19	20	21	22	23	24	25	26	27
Busca$[i]$	u	w	u	x	y	x	z	x	u	p	m
Prof$[i]$	2	3	2	3	4	3	4	3	2	1	0

$\mathbf{v}$	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Desc$[\mathbf{v}]$	1	2	3	6	7	8	10	13	15	16	18	20	21	23
Aban$[\mathbf{v}]$	27	4	3	26	11	8	10	13	25	16	18	24	21	23

Um novo problema

Seja $V[1\,..\,m]$ um vetor de m números inteiros. Dizemos que o vetor V é contínuo se a diferença entre elementos consecutivos do vetor é ± 1, isto é, para cada i, $1 < i \leq m$, $|V[i] - V[i-1]| = 1$.

Dado um vetor contínuo $V[1\,..\,m]$, pré-processamos V para responder, em tempo constante, a pergunta: dados índices i e j, $1 \leq i \leq j \leq m$, qual é a posição do elemento mínimo do intervalo $V[i\,..\,j]$?

Denotamos por MVC(i, j) uma posição do mínimo procurado. MVC satisfaz

$$i \leq \mathsf{MVC}(i,j) \leq j,$$
$$\mathsf{V}[\mathsf{MVC}(i,j)] = \min\{\,\mathsf{V}[i], \mathsf{V}[i+1], \ldots, \mathsf{V}[j]\,\}.$$

Exemplo para MVC

O vetor de profundidades Prof$[i]$ da Tabela **??** é um exemplo de um vetor contínuo. Usaremos ele para exemplificar o novo problema: notamos que

$$\mathsf{MVC}(18, 21) = 19, \quad \mathsf{MVC}(10, 23) = 12, \quad \mathsf{MVC}(3, 23) = 5.$$

Os exemplos foram escolhidos para poder calcular

$$\mathsf{MAC}(w, y) = u, \quad \mathsf{MAC}(s, z) = p, \quad \mathsf{MAC}(o, z) = m.$$

Propriedade Fundamental

Sejam $[i\,..\,j]$ e $[i'\,..\,j']$ dois intervalos cuja união é o intervalo $[i\,..\,j']$.

$$(i < i' \leq j < j' \quad \text{ou} \quad i \leq j < j+1 = i' \leq j'.)$$

Os dois intervalos se sobrepõem ou são disjuntos.

Para todo vetor V de números reais (pode não ser um vetor contínuo)

$$\min V[i\,..\,j'] = \min\{\,\min V[i\,..\,j], \min V[i'\,..\,j']\,\}. \tag{1}$$

Mínimos de intervalos de comprimento 2^k

Para usar a Propriedade 1 pré-calcularemos a posição dos mínimos em intervalos de V cujo número de elementos são potências de dois, ou seja, intervalos com 1, 2, 4, 8 ... elementos.

- Para i, j e k tais que $1 \leq i \leq i + 2^k - 1 = j \leq m$ computaremos $\mathsf{PosMin}[i, k] = t \in [i\,..\,j]$ tal que $V[t] = \min V[i\,..\,j]$. Ou seja, teremos a identidade:

$$\mathsf{V}[\mathsf{PosMin}[i, k]] = \min V[i\,..\,j].$$

Computamos $\mathsf{MVC}(i, j)$ (sejam $k = \mathsf{Log}[j - i + 1]$ e $i' = j - 2^k + 1$) então

$$\mathsf{MVC}(i, j) = \begin{cases} \mathsf{PosMin}[i, k] & \text{se } \mathsf{V}[\mathsf{PosMin}[i, k]] \leq \mathsf{V}[\mathsf{PosMin}[i', k]], \\ \mathsf{PosMin}[i', k] & \text{caso contrário.} \end{cases}$$

Computamos PosMin pela recorrência (seja $i' = i + 2^{k-1}$)

$$\text{PosMin}[i,k] = \begin{cases} i & \text{se } k = 0, \\ \text{PosMin}[i, k-1] & \text{se } k > 1 \text{ e } \text{V}[\text{PosMin}[i,k]] \leq \text{V}[\text{PosMin}[i',k]], \\ \text{PosMin}[i', k-1] & \text{se } k > 1 \text{ e } \text{V}[\text{PosMin}[i,k]] > \text{V}[\text{PosMin}[i',k]]. \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$k=0$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$k=1$	1	2	4	5	5	6	7	9	9	11	12	12	14	14
$k=2$	1	5	5	5	5	6	7	9	12	12	12	12	14	14
$k=3$	1	5	5	5	5	6	12	12	12	12	12	12	14	14
$k=4$	1	5	5	5	5	6	12	12	12	12	12	27		

i	15	16	17	18	19	20	21	22	23	24	25	26	27
$k=0$	15	16	17	18	19	20	21	22	23	24	25	26	27
$k=1$	15	17	17	19	19	20	22	22	24	25	26	27	
$k=2$	15	17	17	19	19	20	22	25	26	27			
$k=3$	15	17	17	19	26	27							

Logaritmos e Potências

Os vetores Log e Exp tais que $\text{Log}[i] = \lfloor \log_2 i \rfloor$, para $1 \leq i \leq m$, e $\text{Exp}[k] = 2^k$, para $0 \leq k \leq \text{Log}[m]$, são calculados no Algoritmo em tempo $O(m)$.

LogaritmosEPotências(m)

```
1  k ← 0
2  Exp[k] ← 1
3  para i de 1 até m faça
4          se i ≥ Exp[k] então
5                  k ← k + 1
6                  Exp[k] ← 2 * Exp[k − 1]
7          Log[i] ← k − 1
8  devolva (Log, Exp)
```

MVC em tempo $\langle O(m \log m), O(1) \rangle$

PréIntermediário(V)

```
1  (Log, Exp) ← LogaritmosEPotências(m)
2  para i de 1 até m faça
3          PosMin[i, 0] ← i
4  para k de 1 até Log[m] faça
5          para i de 1 até m − Exp[k] faça
6                  PosMin[i, k] ← PapaLéguas(i, i + Exp[k] − 1)
7  devolva PosMin                                  ▷ Tabela m × Log[m]
```

MvcIntermediário(i, j)

```
1  devolva PapaLéguas(i, j)
```

MVC em tempo $\langle O(m), O(1) \rangle$

A idéia básica é esta: temos que diminuir o espaço ocupado pela tabela PosMin que usa $O(m \log m)$ posições. Uma forma de obter isto é particionar o vetor V em blocos de b elementos, com b proporcional a $\log m$, e construir a matriz PosMin apenas para auxiliar em consultas que começam e terminam em fronteiras entre blocos vizinhos, isto é, consultas que envolvem uniões de blocos inteiros. Naturalmente, com este PosMin restrito não poderemos responder muitas das consultas possíveis. Felizmente, estas respostas podem ser recuperadas através de uma técnica chamada dos "quatro russos". Esta técnica pode ser usada para levar em conta o que ocorre dentro dos blocos extremos envolvidos numa consulta.

A técnica dos quatro russos

A técnica dos "quatro russos" consiste, no caso, em pré-computar e estocar todos os MVC's internos aos blocos. Veremos que isto pode ser feito em tempo e espaço $O(m)$ para blocos de $\log m$ elementos. Usaremos fortemente a condição da continuidade do vetor dado para obter a complexidade linear.

Cálculo do MVC

Tendo o pré-processamento dos resultados internos aos blocos, qualquer consulta pode ser respondida encontrando, em geral, o mínimo de três mínimos parciais:

- o mínimo interno a um bloco inicial,
- o mínimo de uniões de blocos inteiros e
- o mínimo interno a um bloco final.

O caso que sobra é quando a consulta está inteiramente contida num só bloco: o pré-processamento fornece a resposta. Em qualquer caso, a consulta é respondida em tempo constante! Vejamos, agora, os detalhes.

Os crachás dos 7 blocos do exemplo

g	1	2	3	4	5	6	7
sd's	ssd	sss	sdd	dss	sds	dsd	dd
Crachá$[g]$	14	15	12	11	13	10	4

A decomposição de crachás

Quebra(x, c_1)

1 $c_2 \leftarrow \mathsf{Log}[x] - c_1 - 1$
2 $(y_1, y_2) \leftarrow x$ **div** $\mathrm{Exp}[c_2]$
3 $x_2 \leftarrow y_2 + \mathrm{Exp}[c_2]$
4 $(x_1, z) \leftarrow y_1$ **div** 2
5 **devolva** $(x_1, \mathsf{Letra}[z], x_2)$

Os russos dos 15 crachás de comprimento até 3

x	1	2	3	4	5	6	7
sd's	λ	d	s	dd	ds	sd	ss
Russo$[x]$	1	2	1	3	2	1	1

x	8	9	10	11	12	13	14	15
sd's	ddd	dds	dsd	dss	sdd	sds	ssd	sss
Russo$[x]$	4	3	2	2	4	1	1	1

A nova matriz PosMin

i	1	2	3	4	5	6
$k=0$	(1) 1	(2) 5	(3) 12	(4) 14	(5) 17	(6) 22
$k=1$	(2) 1	(3) 5	(4) 12	(5) 14	(6) 17	
$k=2$	(4) 1	(5) 5	(6) 12			

Pré-processamento linear para MVC

PréMvc(V)

1 (Log, Exp) ← LogaritmosEPotências(m)
2 b ← Log[m]
3 Russo ← Russifica(b)
4 Crachá ← FazCrachás(V, b)
5 PosMin ← Tabela(V, b)

O algoritmo de cálculo de MVC(i, j) fica simples mas não cabe numa só transparência. Tem umas 22 linhas (com comentários).

MVC(i, j)

```
 1  V[0] ← ∞                                  ▷ Anteparo para o cálculo dos mínimos
 2  r ← MvcInterno(i, j)                      ▷ i e j no mesmo grupo?
 3  se r ≠ 0 então
 4          devolva r
 5  ▷ Calculamos i' e j' satisfazendo i ≤ i' e j' ≤ j
 6  i' ← ⌈(i − 1)/b⌉ ∗ b + 1                  ▷ i' inicia o primeiro bloco depois de i
 7  r1 ← MvcInterno(i, i' − 1)
 8  j' ← ⌊j/b⌋ ∗ b                            ▷ j' termina o último bloco antes de j
 9  r3 ← MvcInterno(j' + 1, j)
10  ▷ g_i' e g_j' são os blocos contendo i' e j', respectivamente
11  g_i' ← 1 + (i' − 1)/b
12  g_j' ← j'/b
13  se g_i' ≤ g_j' então
14          r2 ← PapaLéguas(g_i', g_j')
15  senão   r2 ← 0
16  ▷ Calculamos, a seguir, a posição do mínimo de V[r1], V[r2] e V[r3]
17  ▷ Seis linhas de programa omitidos por falta de espaço
```

MAC em tempo $\langle O(n), O(1) \rangle$

PréMAC($\mathbf{T}$)

1 $(m, \text{Busca}, \text{Prof}, \text{Desc}, \text{Aban}) \leftarrow \text{BuscaEmProfundidade}(\mathbf{T})$
2 $\text{V} \leftarrow \text{Prof}$
3 PréMvc(V)

MAC(u, v)

1 **se** $\text{Desc}[u] \leq \text{Desc}[v]$ **então**
2 **devolva** $\text{Busca}[\text{MVC}(\text{Desc}[u], \text{Desc}[v])]$
3 **senão** **devolva** $\text{Busca}[\text{MVC}(\text{Desc}[v], \text{Desc}[u])]$

Notas bibliográficas

O problema do Menor Ancestral Comum em Árvores tem uma longa história da melhoras. A primeira solução não trivial aparece em 1973 e é publicada em [**?**]. A primeira solução de complexidade $\langle O(n), O(1)\rangle$ é de 1984 e aparece em Harel e Tarjan [**?**]. Estes algoritmos são muito complexos e foram simplificados em 1988 por Schieber e Vishkin em [**?**] e em 2000 por Bender e Farach-Colton em [**?**]. O problema do Mínimo de um Vetor aparece em 1984, num trabalho de Gabow, Bentley e Tarjan em [**?**]. Existe uma descrição detalhada da variante de Shieber e Vishkin no livro de Gusfield [**?**], que é um dos raros casos em que o algoritmo é descrito com alguns detalhes num livro. Existe uma ampla literatura sobre o problema, seus usos e suas generalizações. Recomendamos a consulta aos trabalhos [**?**, **?**, **?**].