On List Update with Locality of Reference

Autor: Mário César San Felice

Fonte: Artigo de Albers e Lauer.

12 de novembro de 2010, IC-Unicamp

Problemas *Online*

- Problemas de otimização.
- Entrada recebida em partes.
- Cada parte é processada antes da seguinte.

Problema de Acesso à Lista

- Aparece na gerência de dicionários e na compactação de dados.
- Análogo ao problema de encontrar pastas num arquivo de escritório.
- Temos uma lista não ordenada de itens.
- Recebemos operações de acesso, inserção e remoção de itens.

Problema de Acesso à Lista

- A entrada é uma sequência de requisições σ.
- Cada parte da entrada, chamada requisição, é uma operação.
- Para processar uma operação deve-se encontrar o item requisitado por ela.

Modelo de Custos

- Custos associados a uma lista ligada.
- Operações no item da i-ésima posição custam i.
- Decisões do algoritmo referem-se a como reordenar a lista.
- Transposições livres aproveitam os custos das operações.
- Transposições pagas de itens adjacentes custam 1.

Análise Competitiva

- Técnica de análise de algoritmos *online*.
- Apresenta como garantia uma razão de competitividade (c) que satisfaz:

$$ALG(I) \leq c \times OPT(I) + \alpha,$$

para toda entrada I.

- Análise de pior caso.

Algoritmo *Move-to-Front*

- *Move-to-Front* (MTF): Depois de acessar ou ao inserir um item coloca-o no início da lista.
- O clássico resultado de Sleator-Tarjan mostra que:

$$MTF(\sigma) \leq 2 \times OPT(\sigma).$$

Limite Inferior de Competitividade

- Consideramos o problema de Acesso à Lista Estático.
- O conjunto de itens da lista é L.
- A razão de competitividade (c) de qualquer algoritmo determinístico respeita:

$$c \geq 2 - \frac{2}{|L| + 1}.$$

Conceito de Localidade de Referência

- A qualquer momento apenas um pequeno subconjunto de itens é referenciado.
- Um item requisitado provavelmente será requisitado novamente num futuro próximo.
- Sugere o conceito de *runs*.
- No entanto *runs* puras são raras.

Objetivos

- Analisar a qualidade do algoritmo MTF.
- Considerar o efeito da localidade de referência na análise.
- Obter uma razão de competitividade como medida de qualidade.

Estrutura do Restante da Apresentação

- Modelo para Localidade de Referência.
- Análise Básica de Custos.
- Custo do algoritmo *Offline* Ótimo (OPT).
- Custo do algoritmo *Move-to-Front* (MTF).
- Considerações Finais.

Modelo de Localidade de Referências

- Vamos analisar a localidade de referência considerando pares de itens.
- σ_{xy} é a sequência derivada de σ restrita a $\{x, y\}$.
- $r(\sigma_{xy}) =$ número de *runs* em σ_{xy}.
- $s(\sigma_{xy}) =$ número de *short runs* em σ_{xy}.
- $l(\sigma_{xy}) =$ número de *long runs* em σ_{xy}.
- $r(\sigma_{xy}) = s(\sigma_{xy}) + l(\sigma_{xy})$.

Modelo de Localidade de Referências

- Sejam ρ e ρ' *long runs*.
- ρ' precede ρ se não existem *long runs* entre o final de ρ' e o início de ρ.
- $l_c(\sigma_{xy}) =$ número de *long run changes* em σ_{xy}.
- Para $v \in \{r, s, l, l_c\}$ temos:

$$v(\sigma) = \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} v(\sigma_{xy}).$$

Modelo de Localidade de Referências

- Intuitivamente, uma sequência de requisições σ apresenta alto grau de localidade se possui muitas *long runs*.
- No entanto, utilizaremos uma definição de localidade que utiliza o número de *long run changes*.
- Uma sequência σ apresenta λ-localidade se:

$$\lambda \leq \frac{l_c(\sigma)}{r(\sigma)}.$$

- Se σ só possui *long runs* então $\lambda = 1$.

Análise Básica de Custos

- Vamos mostrar que o custo pago por um algoritmo A para atender uma sequência σ pode ser calculado considerando apenas pares de itens.
- Sendo $m = |\sigma|$ temos que $\sigma(t)$ é o item de σ requisitado no tempo t, $1 \leq t \leq m$.
- $A_x(t, \sigma) = 1$ se x precede $\sigma(t)$ na lista de A no tempo t. Caso contrário $A_x(t, \sigma) = 0$.
- Note que se $x = \sigma(t)$ então $A_x(t, \sigma) = 0$.
- $A_p(t, \sigma)$ é o número de transposiçõe pagas por A no tempo t.

Análise Básica de Custos

- O custo total pago por A para atender σ é:

$$A(\sigma) = \sum_{t=1}^{m} \left(\sum_{x \in L} A_x(t, \sigma) + A_p(t, \sigma) + 1 \right).$$

- Sendo $A_{p,xy}$ o custo de transposições pagas para inverter a posição relativa de x e y, temos:

$$\begin{aligned} A(\sigma) &= \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} \left(\sum_{t:\sigma(t) \in \{x,y\}} (A_x(t, \sigma) + A_y(t, \sigma)) \right) \\ &\quad + \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} A_{p,xy}(\sigma) + m. \end{aligned}$$

Análise Básica de Custos

- Sendo

$$A_{xy}(\sigma) = \sum_{t:\sigma(t)\in\{x,y\}} (A_x(t,\sigma) + A_y(t,\sigma)) + A_{p,xy}(\sigma)$$

temos:

$$A(\sigma) = \sum_{\substack{\{x,y\}\subseteq L \\ x\neq y}} A_{xy}(\sigma) + m. \qquad (1)$$

Análise Básica de Custos

- Na análise dividimos σ_{xy} em p_{xy} fases.
- Cada fase $\pi(i)$, $1 \leq i \leq p_{xy}$, termina com uma *long run* ou com o final de σ_{xy}.
- Sem perda de generalidade vamos considerar que $\pi(i)$ começa com x. Assim ela terá uma das seguintes estruturas:

  (a) $(xy)^k x^l$ com $k \geq 0$ e $l \geq 1$.

  (b) $(xy)^k y^l$ com $k \geq 1$ e $l \geq 0$.

Análise Básica de Custos

- $f_b(\sigma_{xy}) = 1$ se o primeiro item requisitado em σ_{xy} precede o outro. Caso contrário $f_b(\sigma_{xy}) = 0$.

- $f_e(\sigma_{xy}) = 1$ se a última fase tem apenas uma requisição e esta referencía o item que não inicia a lista. Caso contrário $f_e(\sigma_{xy}) = 0$.

Custo do Algoritmo *Offline* Ótimo

- Vamos mostrar que o custo do algoritmo *Offline* Ótimo (OPT) para atender σ respeita:

$$OPT(\sigma) \geq \frac{1}{2}(r(\sigma) + l_c(\sigma) + f_e(\sigma)) - f_b(\sigma) + m.$$

- Como $OPT_{xy}(\sigma) \geq OPT(\sigma_{xy})$ usando (1) temos:

$$OPT(\sigma) \geq \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} OPT(\sigma_{xy}) + m.$$

Custo do Algoritmo *Offline* Ótimo

- Assim para provar o resultado basta mostrar que para todo x e y distintos temos:

$$OPT(\sigma_{xy}) \geq \frac{1}{2}(r(\sigma_{xy}) + l_c(\sigma_{xy}) + f_e(\sigma_{xy})) - f_b(\sigma_{xy}). \quad (2)$$

- Para uma lista com apenas dois itens um algoritmo *offline* ótimo move para o início um item quando começa uma *long run* que o requisita.

- Quando $|\sigma_{xy}| = 1$ o resultado vale. Assim, a partir de agora consideramos que $|\sigma_{xy} > 1|$.

- Dividimos σ_{xy} em fases e, sem perda de generalidade, supomos que a fase $\pi(i)$, para $1 \leq i \leq p_{xy}$, começa com uma requisição por x.

Custo do Algoritmo *Offline* Ótimo

- Assim, para mostrar que (2) vale temos de mostrar que, para $1 < i < p_{xy}$, vale:

$$OPT(\pi(i)) \geq \frac{1}{2}(r(\pi(i)) + l_c(\pi(i))), \tag{3}$$

bem como

$$OPT(\pi(1)) \geq \frac{1}{2}(r(\pi(1)) + l_c(\pi(1))) - f_b(\sigma_{xy}), \tag{4}$$

e, se $p_{xy} > 1$,

$$OPT(\pi(p_{xy})) \geq \frac{1}{2}(r(\pi(p_{xy})) + l_c(\pi(p_{xy})) - f_e(\sigma_{xy})). \tag{5}$$

Custo do Algoritmo *Offline* Ótimo

- Primeiro vamos analisar $\pi(i)$, para $1 < i \leq p_{xy}$.
- Neste caso sabemos que a fase anterior terminou com uma *long run* por y. Logo y está na frente na lista de OPT.
- Se $f_e(\sigma_{xy}) = 1$ então $\pi(p_{xy})$ termina com uma única requisição por x.
- Logo $r(\pi(p_{xy})) = 1$, $l_c(\pi(p_{xy})) = 0$ e

$$OPT(\pi(p_{xy})) = 1 \geq \frac{1}{2}(1 + 0 + 1).$$

- Neste caso (5) vale.

Custo do Algoritmo *Offline* Ótimo

- Para mostrarmos que (3) e (5) valem basta mostrar que, para $1 < i < p_{xy}$ ou $i = p_{xy}$ com $f_e(\sigma_{xy}) = 0$, temos:

$$OPT(\pi(i)) \geq \frac{1}{2}(r(\pi(i)) + l_c(\pi(i))).$$

- Se $\pi(i)$ tem a estrutura (a) $(xy)^k x^l$ então:
  - $OPT(\pi(i)) = k + 1$.
  - $r(\pi(i)) = 2k + 1$.
  - $l_c(\pi(i) \leq 1$.
- Logo:

$$OPT(\pi(i)) = k+1 = \frac{1}{2}((2k+1)+1) \geq \frac{1}{2}(r(\pi(i))+l_c(\pi(i))).$$

Custo do Algoritmo *Offline* Ótimo

- Se $\pi(i)$ tem a estrutura (b) $(xy)^k y^l$ então:
  - $OPT(\pi(i)) = k$.
  - $r(\pi(i)) = 2k$.
  - $l_c(\pi(i) = 0$.
- Logo:

$$OPT(\pi(i)) = k = \frac{1}{2}(2k + 0) \geq \frac{1}{2}(r(\pi(i)) + l_c(\pi(i))).$$

- Assim (3) e (5) valem.

Custo do Algoritmo *Offline* Ótimo

- Resta mostrar que (4) vale.
- Se x está depois de y na lista inicial então $f_b(\sigma_{xy}) = 0$. Neste caso valem os mesmos argumentos do caso anterior.
- Se x está antes de y então $f_b(\sigma_{xy}) = 1$.

Custo do Algoritmo *Offline* Ótimo

- Se $\pi(i)$ tem a estrutura (a) $(xy)^k x^l$ então:
  - $OPT(\pi(i)) = k$.
  - $r(\pi(i)) = 2k + 1$.
  - $l_c(\pi(i) \leq 1$.
- Logo:

$$\begin{aligned} OPT(\pi(i)) &= k \\ &= \frac{1}{2}((2k+1)+1) - 1 \\ &\geq \frac{1}{2}(r(\pi(1)) + l_c(\pi(1))) - f_b(\pi(1)). \end{aligned}$$

Custo do Algoritmo *Offline* Ótimo

- Se $\pi(i)$ tem a estrutura (b) $(xy)^k y^l$ então:
  - $OPT(\pi(i)) = k$.
  - $r(\pi(i)) = 2k$.
  - $l_c(\pi(i) = 0$.
- Logo:

$$\begin{aligned} OPT(\pi(i)) &= k \\ &\geq \frac{1}{2}(2k + 0) - 1 \\ &= \frac{1}{2}(r(\pi(1)) + l_c(\pi(1))) - f_b(\pi(1)). \end{aligned}$$

- Assim (4) vale.

Custo do *Move-to-Front*

- Vamos mostrar que o custo do algoritmo *Move-to-Front* (MTF) para atender σ respeita:

$$MTF(\sigma) = r(\sigma) - f_b(\sigma) + m.$$

- Como $MTF_{xy}(\sigma) = MTF(\sigma_{xy})$ usando (1) temos:

$$MTF(\sigma) = \sum_{\substack{\{x,y\} \subseteq L \\ x \neq y}} MTF(\sigma_{xy}) + m.$$

Custo do *Move-to-Front*

- Assim, para provar o resultado basta mostrar que para todo x e y distintos temos:

$$MTF(\sigma_{xy}) = r(\sigma_{xy}) - f_b(\sigma_{xy}).$$

- Isto vale porque MTF paga 1 para atender cada *run*.

- A única possível exceção é a primeira requisição se o item estiver no início da lista. Mas neste caso $f_b(\sigma_{xy}) = 1$.

Competitividade do *Move-to-Front*

- Sendo $\alpha(\sigma) = (m - f_b(\sigma))/r(\sigma)$ e $\beta(\sigma) = l_c(\sigma)/r(\sigma)$ temos que:

$$
\begin{aligned}
c &= MTF(\sigma)/OPT(\sigma) \\
&\leq \frac{r(\sigma) - f_b(\sigma) + m}{\frac{1}{2}(r(\sigma) + l_c(\sigma) + f_e(\sigma)) - f_b(\sigma) + m} \\
&\leq \frac{2 + 2(m - f_b(\sigma))/r(\sigma)}{1 + l_c(\sigma)/r(\sigma) + 2(m - f_b(\sigma))/r(\sigma)} \\
&\leq \frac{2 + 2\alpha(\sigma)}{1 + \beta(\sigma) + 2\alpha(\sigma)} \\
&\leq \frac{2}{1 + \lambda}
\end{aligned}
$$

Consideração Final

- Temos a razão de competitividade $\frac{2}{1+\lambda}$ para o algoritmo *Move-to-Front* que considera a localidade de referência da sequência de requisições.
- Esta razão de competitividade mostrou-se justa quando comparada com resultados experimentais.

Referências


Susanne Albers and Sonja Lauer.
On List Update with Locality of Reference.
Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I, 96–107, 2008.


Allan Borodin and Ran El-Yaniv.
Online Computation and Competitive Analysis.
Cambridge University, 1998.