Análise de Algoritmos
Descrição do Projeto
Reconhecimento Aproximado de Padrões em Palavras via Árvores de Sufixos

Prof. Mauricio Ayala-Rincón

27 de Agosto 2001

1 Introdução

1.1 A estrutura de dados

As árvores de sufixos são estruturas de dados de grande relevância pelas suas diversas aplicações práticas em problemas combinatórios em palavras, como por exemplo o problema de reconhecimento de padrões [McC76], busca de repetições em palavras [AP83], compressão de texto [Apo85], etc.. A árvore de sufixos de uma palavra dada sobre um certo alfabeto organiza a informação sobre a estrutura combinatória da palavra de maneira que se podem responder eficientemente questões sobre a própria estrutura da palavra em questão.

Seja Σ um alfabeto e Σ^* a linguagem das palavras sobre Σ. Dada uma palavra $s \in \Sigma^*$, constroi-se a árvore de sufixos de s, $\mathcal{T}_s$, compactando o *trie* (veja p. ex. cap. 17 de [Sed92]) dos sufixos da palavra $s\$$, onde $\$$ é um símbolo distinguido fora do alfabeto Σ. Assim, cada aresta de $\mathcal{T}_s$ estará marcada com uma subpalavra de $s\$$, existe exatamente um *passeio* da raiz até alguma folha de $\mathcal{T}_s$ correspondendo a cada sufixo da palavra $s\$$ e as arestas descendentes de qualquer vértice interno de $\mathcal{T}_s$ estarão marcadas com subpalavras de $s\$$ que iniciam com símbolo diferente. Além disso, cada vértice interno de $\mathcal{T}_s$ terá pelo menos dois descendentes. Já que o número de folhas de $\mathcal{T}_s$ é $|s|+1$, onde $|s|$ denota o comprimento da palavra s, e cada vértice interno tem pelo menos dois descendentes, pode-se provar facilmente que o número de vértices de $\mathcal{T}_s$ é de ordem linear no comprimento de s (i.e., está em $\Theta(|s|)$). Por outra parte a estrutura pode ser construída de maneira ótima em tempo $\Theta(|s|)$ por métodos bem conhecidos baseados

- na inserção iterativa dos sufixos de maior comprimento até inserir os de menor comprimento de s completando $\mathcal{T}_s$ (veja [Wei73, McC76]) ou

- na inserção iterativa dos prefixos de menor comprimento até inserir o de maior comprimento (veja [Ukk95]) ou

- na construção recursiva das árvores correspondentes aos sufixos nas posições pares e ímpares de $s\$$ e a combinação destas obtendo $\mathcal{T}_s$ (veja [Far97]).

Os métodos iterativos são equivalentes como demonstrado em [GK97]. A versão mais simples de construção é a apresentada em [Ukk95], que tem a grande vantagem de ser *on-line* (i.e., o texto vai sendo processado na medida em que vai sendo lido). O método recursivo é de grande interesse teórico, mas até hoje não se tem uma implementação deste que seja razoável.

O principal defeito das árvores de sufixos é sua falta de dinamicidade; i.e., a estrutura $\mathcal{T}_s$ resulta apropriada para problemas onde s se mantém fixa, mas não é possível, em geral, a adequar à resolução de problemas onde s sofre frequentes modificações. Isto é devido a que não existem algoritmos sublineares para atualização de árvores de sufixos e que a própria construção destas é linear [ARC98]. Mecanismos interessantes para dotar de dinamicidade dita estrutura podem ser criados a partir de [KR97] ou [FGM98].

1.2 O problema

O *reconhecimento aproximado de padrões em palavras* é uma generalização do problema de reconhecimento de padrões em palavras. O problema pode-se estabelecer da seguinte maneira:

> dadas palavras **padrão** e **texto**, p e t, respectivamente, sobre um alfabeto Σ, com $|p| = m > 0$ e $|t| = n > 0$ e dado um inteiro $k \geq 0$, encontrar subpalavras s de t tais que a **distância de edição** entre s e p, $d(s,p)$, é menor ou igual que k.

A distância de edição entre duas palavras x e y é o custo total mínimo possível de uma sequência de *passos de edição* para converter x em y. Denotando com ϵ a palavra vazia, em cada passo de edição aplica-se uma *regra de reescrita* da forma:

- $a \to \epsilon$, $a \in \Sigma$ (eliminação);
- $\epsilon \to a$, $a \in \Sigma$ (inserção);
- $a \to b$, $a, b \in \Sigma$ (substitução).

Diz-se que a palavra x é uma k-*aproximação* de y, se se pode converter x em y em k passos de edição. Por exemplo, existe uma ocorrência 1-aproximada do padrão *abb* na primeira posição do texto *abcdbdb* e uma ocorrência 2-aproximada na quarta posição, como se ilustra no seguinte esquema.

$$
\begin{array}{cccccccc}
a & b & b & & a & b & \epsilon & b \\
 & & \downarrow & & \downarrow & & \downarrow & \\
a & b & \epsilon & c & d & b & d & b
\end{array}
$$

Esta variação do problema de reconhecimento de padrões em palavras é relevante quando deven ser considerados *erros* no reconhecimento e é fundamental para razoar acerca do problema geral de reconhecimento de padrões (seja em imagens, sons, etc.). As soluções do problema em questão têm aplicações interessantes na biologia molecular [LVN85] e têm bom desempenho no montagem de fragmentos de DNA [Per95], [Ste92].
O projeto do curso consiste no estudo comparativo e implemantação dos algoritmos de reconhecimento aproximado de padrões desenvolvidos por Jokinen e Ukkonen em [Ukk93].

2 A solução dinâmica do problema

O mecanismo de solução mais "popular" do problema de reconhecimento aproximado de padrões é baseado em técnicas de *programação dinâmica* e tem complexidade $O(nm)$.
Sejam $p = p_1 \dots p_m$ e $t = t_1 \dots t_n$ um padrão e um texto, respectivamente. Basicamente o método consiste em construir uma $m + 1 \times n + 1$-tabela D tal que para $0 \leq i \leq m$, $0 \leq j \leq n$, $D[i,j]$ é a mínima distância de edição de $p_1 \dots p_i$ à subpalavra do texto que termina em t_j. Existem aproximações do padrão com distância de edição $\leq k$ terminando em t_j se e solamente se $D[m,j] \leq k$. A tabela se constroi segundo as seguintes regras:

- $D[0,j] = 0, \;\; 0 \leq j \leq n$;
- $D[i,0] = i, \;\; 0 \leq i \leq m$;

- $D[i,j]$ = mínimo entre:
  - $D[i-1,j]+1$,
  - $D[i-1,j-1]+$ se $p_i = t_j$ então 0 se não 1,
  - $D[i,j-1]+1$,

  $1 \leq j \leq n, 1 \leq i \leq m$.

Note que $D[i,j]$ depende unicamente de $D[i-1,j], D[i-1,j-1]$ e $D[i,j-1]$ o que permite construir a tabela coluna por coluna e fila por fila em ordem ascendente.
Considere, por exemplo, a tabela construida para $p = cab$, $t = abcabdab$:

		a	b	c	a	b	d	a	b
	0	0	0	0	0	0	0	0	0
c	1	1	1	0	1	1	1	1	1
a	2	1	2	1	0	1	2	1	2
b	3	2	1	2	1	0	1	2	1

3 As etapas do desenvolvimento do projeto

Os alunos deverão definir grupos de trabalho de **quatro** membros. O projeto será dividido em três etapas como segue:

- A primera etapa da realização do projeto é o estudo comparativo das soluções do problema e a definição das estruturas de dados apropriadas para o desenvolvimento da implementação.
  Para o estudo do problema e da solução os alunos deverão examinar detalhadamente a referência principal do projeto ([Ukk93]).
  Avaliação (peso 1.0): Cada grupo de trabalho entregará um Relatório Inicial no dia **8.10.2001** que conterá os seguintes itens:
  - Especificação do problema.
  - Explicação da solução de Ukkonen.
  - Descrição dos algoritmos a implementar.
  - Análise da complexidade em tempo e espaço dos algoritmos a implementar.
  - Definição das estruturas de dados e plano da implementação.
  - Referências.
- A segunda etapa do projeto e a de Verificação das Implementações. Os grupos deverão ter prontas as suas implementações até o dia **04.11.2001**. Na semana do 05-09 de Novembro realizar-se-á a verificação do trabalho para a qual os grupos deverão, em acordo com o professor, determinar um horário (dois horas) no qual os quatro membros do grupo deverão comparecer.
  Avaliação (peso 1.5):
  - Um dos membros, selecionado por sorteio, explicará os detalhes da implementação em máximo 20 minutos.
  - Os quatro membros do grupo poderão complementar a explicação inicial em máximo 10 minutos.
  - A implementação será testada em 45 minutos com um banco de palavras exemplo e palavras geradas aleatoriamente.

- A etapa final do projeto consiste da apresentação dos resultados finais e conclusões do estudo do problema e da implementação.
  Avaliação (peso 2.5): Cada grupo de trabalho deverá entregar um Relatório Final do projeto o dia **28.11.2001** com o seguinte conteúdo:
  - Introdução.
  - Especificação do problema e explicação do método de solução.
  - Descrição e análise da complexidade dos algoritmos implementados.
  - Definição das estruturas de dados da implementação.
  - Descrição da implementação.
  - Análise estatístico do processo de verificação incluendo tabelas e gráficos de comprimento das palavras e das distâncias de edição testadas (geradas aleatoriamente, do banco de exemplos) versus tempo de computação.
  - Conclusões.
  - Referências.

4 O estado da arte

As árvores de sufixos não aparecem nos textos clássicos de ánalise de algoritmos devido ao fato de que as apresentações originais desta estrutura de dados eram bastante complexas. Mais existem dois livros atuais em ánalise de algoritmos que explicam as árvores de sufixos são [CR94] e [Ste94].

Algumas soluções atuais, apropriadas para o caso de um texto fixo, e já implementadas com sucesso como projeto desta disciplina em semestres precedentes, podem ser consultadas em [JU91]. Nesta solução do problema, pré-procesa-se o texto construindo o seu *autômato de suffixos* para logo encontrar eficientemente todas as ocorrências $\leq k$-aproximadas do padrão. Trabalho adicional similar á solução proposto por Ukkonen foi desenvolvido em [Cob95].

Em [Per95] apresentam-se três métodos diferentes para o problema de encontar a maior subpalavra de um padrão em um texto que também podem ser aplicados ao problema de reconhecimento aproximado. Evidência da importância prática do problema proposto é a grande variedade de mecanismos de reconhecimento aproximado obtidos na literatura atual (veja p. ex. [APD99]).

Referências

[AP83] A. Apostolico and F. P. Preparata. Optimal Off-line Detection of Repetitions in a String. *Theoretical Computer Science*, 22:297–315, 1983.

[APD99] L. Allison, D. Powell, and T.I. Dix. Compression and Approximate Matching. *The Computer Journal*, 42(1):1–10, 1999.

[Apo85] A. Apostolico. The Myriad Virtues of Subword Trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*, pages 85–96. Springer Verlag, 1985.

[ARC98] M. Ayala-Rincón and P. D. Conejo. A Linear Time Lower Bound on Updating Algorithms for Suffix Trees. In *String Processing and Information Retrieval: A South American Symposium*, pages 1–6, September 1998. Santa Cruz de La Sierra, Bolivia.

[Cob95] A.L. Cobbs. Fast Approximate Matching Using Suffix Trees. volume 937 of *LNCS*, 1995.

[CR94] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[Far97] M. Farach. Optimal Suffix Tree Construction with Large Alphabets. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 137–143, Abril 1997.

[FGM98] P. Ferragina, R. Grossi, and M. Montangero. Note On Updating Suffix Tree Labels. *Theoretical Computer Science*, 201:249–262, 1998.

[GK97] R. Griegerich and S. Kurtz. From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction. *Algorithmica*, 19:331–353, 1997.

[JU91] P. Jokinen and E. Ukkonen. Two Algorithms for Approximate String Matching in Static Texts. In A. Tarkecki, editor, *Sixteenth International Symposium on Mathematical Foundations of Computer Science, Poland*, volume 520 of *LNCS*, pages 240–248. Springer, September 1991.

[KR97] G. Kucherov and M. Rusinowitch. Matching a Set of Strings with Variable Length don't Cares. *Theoretical Computer Science*, 178:129–154, 1997.

[LVN85] G. M. Landau, U. Vishkin, and R. Nussinov. An Efficient String Matching Algorithm with k Differences for Nucleotide and Amino Acid Sequences. Technical Report TR-37/85, Department of Computer Sciences, Tel Aviv University, 1985.

[McC76] E. M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. *Journal of the Association for Computing Machinery*, 23(2):262–272, April 1976.

[Per95] C. H. Perleberg. Three Longest Substring Algorithms. *Journal of the Brazilian Computer Society*, 1(3):12–22, April 1995.

[Sed92] R. Sedgewick. *Algorithms in* C^{++}. Addison-Wesley, 1992.

[Ste92] G. A. Stephen. String Search. Technical Report TR-92-gas-01, School of Electronic Engineering Science, University College of North Wales, Dean Street, Bangor, Gwynedd, UK LL57 1UT, October 1992.

[Ste94] G. A. Stephen. *String Searching Algorithms.* World Scientific, 1994. Paperback reprint 1998.

[Ukk93] E. Ukkonen. Approximate String-Matching over Suffix Trees. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Fourth Annual Symposium on Combinatorial Pattern Matching, Padova, Italy*, volume 684 of *LNCS*, pages 228–242. Springer, June 1993.

[Ukk95] E. Ukkonen. On-line Construction of Suffix-Trees. *Algorithmica*, 14:249–260, 1995.

[Wei73] P. Weiner. Linear Pattern Matching Algoritms. In *IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.