Trabalho Prático 1
AEDS3

Pedro Araujo Pires
ppires@dcc.ufmg.br

Introdução

O objetivo deste trabalho prático foi criar um programa de pesquisa de padrões em cadeias de caracteres. A pesquisa em cadeias de caracteres é aplicada em diversas áreas, como edição de textos, recuperação de informações, e estudos de sequências de DNA. O programa desenvolvido recebe um padrão a ser pesquisado em uma cadeia de caracteres, e gera um novo programa que contém um autômato determinístico para pesquisar o padrão.

Um autômato determinístico é definido por uma tupla (Q, I, F, Σ, T), onde Q é um conjunto finito de estados, entre os quais existe um estdo inicial $I \in Q$, e alguns são estados finais $F \subseteq Q$. As transições entre estados são rotuladas por elementos de Σ, onde Σ é o alfabeto finito de entrada. As transições são formalmente definidas por uma função de transição T, que associa a cada estado $q \in Q$ um outro estado $p \in Q$, para cada $\alpha \in \Sigma$. Uma cadeia é reconhecida por um autômato se, para cada símbolo da cadeia, existe uma transição de um estado *p* para um estado *q*, e ao final do processamento dos símbolos, o autômato se encontra em um estado final.

Solução Proposta

A solução proposta para o problema do casamento de um padrão em uma cadeia de caracteres, utilizando um autômato finito determinístico, foi a utilização do algoritmo criado por Knuth-Morris-Pratt. Esse algoritmo executa um pré-processamento no padrão a ser encontrado, e gera um autômato determinístico capaz de reconhecer todas as ocorrências do padrão dentro de uma cadeia de caracteres.

Durante a fase de pré-processamento do padrão P, o algoritmo KMP busca pelo maior sufixo de P, que também é um prefixo. Dessa forma, são construídas transições no autômato que permitem que nenhum caractere seja lido mais de uma vez, fazendo com que o apontador do texto nunca seja decrementado.

O autômato gerado na fase de pré-processamento do padrão é representado por um *array* de tamanho *n+1*, onde *n* é o tamanho do padrão. Cada posição no *array* representa o comportamento do estado correspondente àquela posição. Por exemplo, para o padrão `ababaca`, o *array* gerado é:

```
{-1, 0, -1, 0, -1, 3, -1, 1}
```

Esse *array* deve ser interpretado da seguinte forma: se um estado possui o valor **-1**, esse estado possui somente uma transição para o estado seguinte, quando o símbolo adequado é lido. Caso o estado possua um valor diferente de **-1**, esse estado possui, além da transição para o estado seguinte, as mesmas transições que o estado correspondente ao número. A seguinte função de transição de estados (δ) pode ser consgtruída a partir do *array* acima:

```
δ(0, a) = 1
δ(1, b) = 2
δ(1, a) = δ(0, a) = 1
δ(2, a) = 3
δ(3, b) = 4
δ(3, a) = δ(0, a) = 1
δ(4, a) = 5
δ(5, c) = 6
δ(5, a) = δ(3, a) = 1
```

```
δ(5, b) = δ(3, b) = 4
δ(6, a) = 7
δ(7, b) = δ(1, b) = 2
```

Todas as outras transições que não aparecem na tabela acima, levam para o estado inicial (0). A partir da tabela com as transicões do autômato, é possível construir o diagrama de estados correspondente:

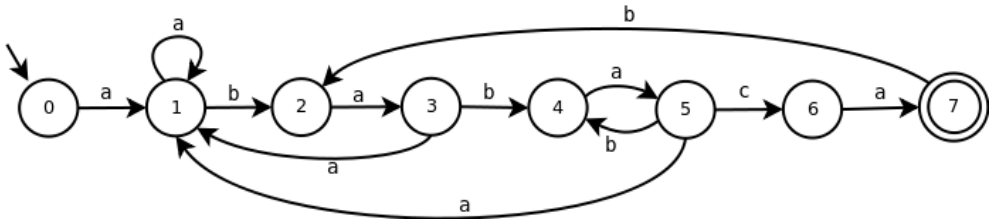

Diagrama de estados do autômato que reconhece o padrão `ababaca`, gerado pelo algoritmo KMP.

Implementação

O código do programa foi separado em dois módulos: `io` e `automata_constructor`. O módulo io possui somente uma função, `ParseParams`, que pega o primeiro parâmetro passado pela linha de comando, na hora da execução do programa. No caso do `tp1-1`, o parâmetro passado é o padrão a ser pesquisado na cadeia de caracteres.

O módulo automata_constructor possui duas funções: `preKMP` e `WriteFile`. A função `preKMP` é onde ocorre a construção do autômato. Essa função possui um *loop while*, onde o padrão a ser pesquisado é percorrido, e para cada caractere lido é verificado se o sufixo da *substring* lida até o momento é também um prefixo dessa *substring*. Para fazer essa verificação a *substring* é percorrida novamente, fazendo as comparações. No pior caso, o algoritmo de construção do autômato percorre o padrão 2n vezes, onde n é o tamanho do padrão. Logo, a construção do autômato possui complexidade O(n).

A função `WriteFile` é o responsável por escrever o programa `tp1-2`, que pesquisa o padrão em um texto. O programa `tp1-2` gerado possui uma estrutura de dados para armazenar os resultados da execução do autômato, assim como uma função para executar o mesmo.

A função KMP, presente no programa `tp1-2`, é a responsável por executar o autômato. Essa função lê uma cadeia de caracteres de um arquivos, e vai percorrendo o autômato. Sempre que o estado final é atingido, é impresso na tela a posição em que foi encontrado o padrão. Como a função lê cada caractere somente uma vez, ela possui complexidade O(k), onde k é o número de caracteres da cadeia de caracteres.

Como neste trabalho são feitos dois programas, são necessárias duas fases de compilação. Para compilar e executar o programa, para pesquisar o padrao `ababaca` em uma cadeia de caracteres que está em um arquivo chamado `entrada`, devem ser seguidos os seguintes passos:

```
make
./tp1-1 ababaca
make tp1-2
./tp1-2 entrada
```

Conclusão

Neste trabalho foi implementado um algoritmo de casamento de cadeias de caracteres. Para fazer o casamento foi utilizado um algoritmo baseado em autômatos determinísticos. Foi criado um programa que escreve outro programa, que contém o autômato para fazer a busca.

Apesar de existirem autômatos bem ineficientes para essa busca, foi utilizado o algoritmo Knuth-Morris-Pratt para gerar o autômato. O autômato gerado possui transições inteligentes que garantem que cada caractere do texto a ser pesquisado seja lido somente uma vez. A complexidade na hora de gerar o autômato foi compensada pela facilidade da análise de complexidade do algoritmo.