Nejdelší palindrom

Tomáš Krejčí
tomas789@gmail.com

22. října 2012

1 Zadání

Úkolem je vyšetřit délku nejdelšího palindromického podřetězce vstupního řetězce v čase lineárním vzhledem k počtu znaků vstupního řetězce.

2 Definice

O řetězci řekneme, že je **palindromický**, pokud se čte z obou stran stejně. To znamená, že je symetrický podle svého středu. Pokud je řetězec liché délky, rozumíme středem prostřední písmeno. V případě sudého řetězce je středem mezera mezi znaky na pozici $\lfloor \frac{n}{2} \rfloor$ a $\lfloor \frac{n}{2} \rfloor + 1$, kde n je délka vstupního řetězce.

V následujícím textu budeme jako *content* označovat vstupní řetězec a *len* bude jeho délka.

Pokud budeme indexovat pole, budeme tak dělat od nuly a to i v textu. To znamená, že v řetězci $ABCD$ je prvek na nulté pozici písmeno A.

3 Řešení

3.1 Naivní algoritmus

Princip tohoto řešení je vzít všechny možné podřetězce vstupního řetězce a otestovat, zda-li jsou to palindromy.

Vstupní řetězec má celkem $\binom{n}{2}$ podřetězců. Vybraný řetězec umíme ověřit, zda je palindromem v lineárním čase vzhledem k délce testovaného podřetězce. Dohromady algoritmus pracuje v čase $O(n^3)$.

Implementace tohoto algoritmu je ukázána v příloze A na straně 5.

3.2 Kvadratické vylepšení

Algoritmus popsaný v sekci 3.1 (str. 1) není příliš vhodný zejména z důvodu, že vybírá všechny možné podřetězce a těch je kvadraticky mnoho.

Možným vylepšením je namísto přes podřetězce iterovat hlavní cyklus algoritmu přes jednotlivé středy palindromů kterých je celkem $2n + 1$ a expandovat zkoumanou oblast na obě strany dokud je u řetězce zachována vlastnost býti palindromem.

Jak již bylo řečeno výše, všech možných středů je lineárně mnoho a pro každé ověření nejdelšího palindromu okolo daného středu je potřeba až lineárně mnoho práce v případě překrývajících se palindromů (např. delší posloupnost tvořená jedním znakem). To znamená celkovou složitost $O(n^2)$. Poukažme ještě na skutečnost, že v případě nepřekrývajících se palindromů je složitost lineární.

Implementace algoritmu je opět k nalezení v příloze B na straně 6.

3.2.1 Poznámky k implementaci.

Vnější *for* cyklus iteruje přes jednotlivé potenciální středy palindromu. Pokud je proměnná i přes kterou se cyklus iteruje sudá, značí to, že aktuálně ukazuje na písmeno. V opačném případě na mezeru mezi písmeny.

Algoritmus pomocí proměnných *ukLevy* a *ukPravy* ukazuje na konce palindromu.

Před zehájením iterací *while* cyklu je třeba nastavit vhodně počáteční proměnné. Počáteční nastavení ukazuje Obrázek 1. Proměnná *curr* obsahuje velikost aktuálního palindromu.

	curr	*ukLevy*	*ukPravy*
Středem je písmeno	1	$\frac{i}{2} - 1$	$ukLevy + 2$
Středem je mezera	0	$\frac{i-1}{2}$	$ukLevy + 1$

Obrázek 1: Inicializace proměnných

3.3 Lineární algoritmus (úprava)

3.3.1 Algoritmus

```
nastav stred aktualniho superpalindromu na prvni pismenko;
nastav velikost palindromu se stredem v prvnim pismenku na 1;
for i := 1 to 2*N+1 do begin
        if stred i je uvnitr superpalindromu then begin
                if leva hrana symetrickeho palindromu presahuje
                   levou hranu superpalindromu then begin
                        nastav velikost palindromu se stredem v i na velikost
                        symetrickeho palindromu nalezici do superpalindromu;
                end else begin
                        nastav velikost palindromu se stredem v i na velikost
                        symetrickeho palindromu;
                        continue;
                end;
        end;

        while rozsireni palindromu se stredem i je stale palindrom do
                inkrementuj velikost palindromu se stredem v i;
                if je to prvni palindrom, ktery se podival na
                   novy znak vstupu then begin
                        oznac tento palindrom jako novy superpalindrom;
                end;
        begin
end;

return max(d[1], d[2], ..., d[2*N+1]);
```

3.3.2 Důkaz správnosti

Předpokládejme, že výše zmíněný kvadratický algoritmus je správný (triviální - ověřuje všechny možnosti). Lineární algoritmus vznikne tak, že předtím, než začneme expandovat okolo středu, podíváme se, zda-li je uvnitř nějakého většího palindromu. Pakliže uvnitř většího palindromu je, podíváne se na symetrický protějšek, který musí být nutně shodný resp. jeho část uvnitř většího palindromu. V případě, že ho přesahuje, tak pokračujeme expanzi. Odebrali jsme z algoritmu tedy pouze kroky, které již byly jednou počítány a správnost zůstane zachována.

3.3.3 Důkaz časové složitosti

Algoritmus je lineární vzhledem k počtu znaků vstupního řetězce, protože naždý znak ze vstupu ne nahlédnut maximálně konstantně krát (dvakrát) a každý z prvků pole d je doplněn právě jednou.

A Naivní algoritmus

```
#include <stdio.h>
#include <string.h>

#define MAX_INPUT 500

int main(int argc, const char * argv[])
{
    char content[MAX_INPUT];
    FILE * pFile;
    size_t len;
    int max = 0;
    int i, j, ukLevy, ukPravy;

    if (2 != argc)
    {
        printf("Usage: palindrome <filename>");
        return 1;
    }

    pFile = fopen(argv[1], "r");
    fscanf(pFile, "%s", content);

    len = strlen(content);

    for (i = 0; i < len; i++)
    {
        for (j = i; j < len; j++)
        {
            ukLevy = i;
            ukPravy = j;

            while (content[i] == content[j])
            {
                if (ukPravy - ukPravy <= 1)
                {
                    if (j-i > max) max = j-i;
                    break;
                }
                else ++ukLevy, ++ukPravy;
            }
        }
    }

    printf("%d\n", max);

    return 0;
}
```

B Kvadratický algoritmus

```c
#include <stdio.h>
#include <string.h>

#define MAX_INPUT 500

int isLetter(int i);

int main(int argc, const char * argv[])
{
    char content[MAX_INPUT];
    FILE * pFile;
    size_t len;
    int max;
    int i, curr, ukLevy, ukPravy;

    if (2 != argc)
    {
        printf("Usage: palindrome <filename>");
        return 1;
    }

    pFile = fopen(argv[1], "r");
    fscanf(pFile, "%s", content);

    len = strlen(content);
    max = 0;

    /* i%2 == 0 -> je to pismeno */
    /* i%2 == 1 -> je to mezera */

    for (i = 0; i < 2*len + 1; i++)
    {
        if (isLetter(i))
        {
            ukLevy = i / 2 - 1;
            ukPravy = ukLevy + 2;
            curr = 1;
        }
        else
        {
            ukLevy = (i - 1) / 2;
            ukPravy = ukLevy + 1;
            curr = 0;
        }

        if (curr > max) max = curr;

        while (ukLevy >= 0 && ukPravy < len)
        {
            if (content[ukLevy] == content[ukPravy])
            {
                curr += 2;
                --ukLevy;
                ++ukPravy;
                if (curr > max) max = curr;
            }
            else
            {
                break;
```

```
                }
            }
        }

        printf("%d\n", max);

        return 0;
}

/* Is it letter or space? */
inline int isLetter(int i)
{
        return i % 2 == 0;
}
```