Jeuring’s algorithm on palindromes

An implement with literate programming and C++ programming language

July 20, 2012

1 Introduction

All the techniques used in this article is well-introduced in Don Knuth’s magnificent book [1], which is strongly-suggested in Computer Science. This is a very efficient algorithm, designed by Johan Jeuring, which is used to determine the longest palindrome in $O(n)$ time. You can read his original article [3] and his pretty Haskell code. Now, I want to show how his algorithm works and why it’s right.

As usual, a program (especially C/C++ code) is made up of three parts: preprocessors, global variables and routines.

1a ⟨*pldrm.cc* 1a⟩≡

```
⟨preprocessors 1b⟩
⟨globals 2c⟩
⟨main 2a⟩
```

It’s essential to include the standard libraries, such as cstdio, cstdlib and cstring.

1b ⟨*preprocessors* 1b⟩≡ (1a) 2b ▷

```
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <algorithm>
using namespace std;
```

2 Main routine

Here's the main routine.

2a ⟨*main* 2a⟩≡ (1a)
```
int main()
{
  ⟨mainvar 3b⟩
  ⟨input 2d⟩
  ⟨main loop 3c⟩
  ⟨output 2e⟩
  return 0;
}
```

Given that $s[1\,..\,n]$ is the string inputted, and n is the length of s, where $s[0] = 1, s[n+1] = 0$.

2b ⟨*preprocessors* 1b⟩+≡ (1a) ◁1b
```
#define MAX_LEN 100000
```

2c ⟨*globals* 2c⟩≡ (1a) 3a▷
```
int n;
char s[MAX_LEN+2];
```

2d ⟨*input* 2d⟩≡ (2a)
```
s[0] = 1;
scanf("%s", &s[1]);
n = strlen(&s[1]);
```

Definition 1. *We say $l + r$ is the center of substring $s[l\,..\,r]$. For example, 4 is the center of $s[2\,..\,2]$ or $s[1\,..\,3]$.*

$a[k]$ is the length of the longest palindrome whose center is k.

2e ⟨*output* 2e⟩≡ (2a)
```
for (int k=1; k<=2*n+1; k++) {
  printf("%d\n", a[k]);
}
```

3 Main loop

Definition 2. *We call some string A is a tail palindrome of the other string B if and only if A is a palindromic tail substring of B. For example, $A = aba$ and $B = aaaababa$, where A is palindromic and A is a tail substring of B.*

The main loop calculate array a in the increasing order of the index. We will see that the invariant of main loop is assertion 1.

3a ⟨*globals* 2c⟩+≡ (1a) ◁2c

```
int a[2*MAX_LEN+4];
```

3b ⟨*mainvar* 3b⟩≡ (2a) 4a▷

```
int j, l;
```

3c ⟨*main loop* 3c⟩≡ (2a)

```
j = 1;
l = 1;
a[0] = 1;
a[1] = 0;
for (int k=2; k<=n+1; k++) {
  ⟨process 3d⟩
  advance:
  ;
}
```

Process is made up of an infinite loop, which is used to find the longest tail palindrome of $s[0\,..\,k]$. There are two exits of it. One is in extension subroutine, while the other one is in move loop. The way of exit is *goto advance;*. The loop invariant for process is assertion 3.

3d ⟨*process* 3d⟩≡ (3c)

```
for (;;) {
  ⟨check 3e⟩
  ⟨move loop 4b⟩
}
```

The check subroutine checks whether a tail palindrome of $s[0\,..\,k-1]$ could be extended to that of $s[0\,..\,k]$. If so, exit from the process loop and advance k, otherwise start the move loop.

3e ⟨*check* 3e⟩≡ (3d)

```
if (s[k] == s[k-l-1]) {
  l += 2;
  goto advance;
}
```

Here's the move loop, which looks short and easy. It's used to find a shorter tail palindrome of $s[0\,..\,k-1]$.

4a ⟨*mainvar* 3b⟩+≡ (2a) ◁3b

```
int p;
```

4b ⟨*move loop* 4b⟩≡ (3d)

```
a[++j] = l;
for (p=j-1; --l>=0&&l!=a[p]; p--) {
  a[++j] = min(a[p], l);
}
if (l < 0) {
  l = 1;
  goto advance;
}
```

4 The outline of the proof

Lemma 1. *An arbitrary tail palindrome of* $s[0\,..\,m]$ *is also a tail palindrome of* $s[1\,..\,m]$ *for all* $m > 0$.
Lemma 2. *The substring of* s *whose center is* c *and length is* l *is* $s[(c-l+1)/2\,..\,(c+l-1)/2]$.

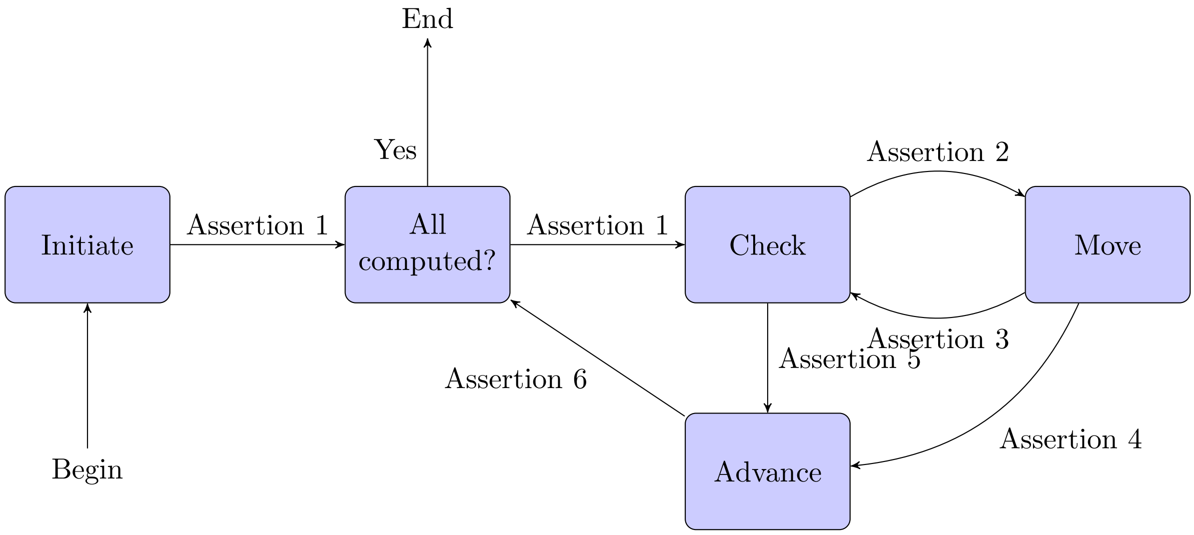

Figure 1: Flowchart of the main loop

Now let's make some assertions on the flow chart of the main loop. They're not very difficult to verify.
Assertion 1. $2 \le k \le n+1$, *and the length and the center of the longest tail palindrome of* $s[0\,..\,k-1]$ *is* l *and* $j+1$, *while* $a[0\,..\,j]$ *is computed.*
Assertion 2. $2 \le k \le n+1$, $s[k-l\,..\,k-1]$ *is a palindrome or empty string, whose center is* $j+1$, $l \ge 0$, *the length of the longest tail palindrome of* $s[0\,..\,k]$ *is less than* $l+2$, *and* $a[0\,..\,j]$ *is calculated.*
Assertion 3. $2 \le k \le n+1$, $s[k-l\,..\,k-1]$ *is a palindrome or empty string, whose center is* $j+1$, $l \ge 0$, *the length of the longest tail palindrome of* $s[0\,..\,k]$ *is not more than* $l+2$, *and* $a[0\,..\,j]$ *is assigned.*
Assertion 4. $2 \le k \le n+1$, *the length and the center of the longest tail palindrome of* $s[0\,..\,k]$ *is* $l = 1$ *and* $j+1$, *where* $a[0\,..\,j]$ *is determined.*
Assertion 5. $2 \le k \le n+1$, *the length and the center of the longest tail palindrome of* $s[0\,..\,k-1]$ *is* l *and* $j+1$, *and* $a[0\,..\,j]$ *is calculated.*
Assertion 6. $3 \le k \le n+2$, *the length and the center of the longest tail palindrome of* $s[0\,..\,k-1]$ *is* l *and* $j+1$, *and* $a[0\,..\,j]$ *is computed.*

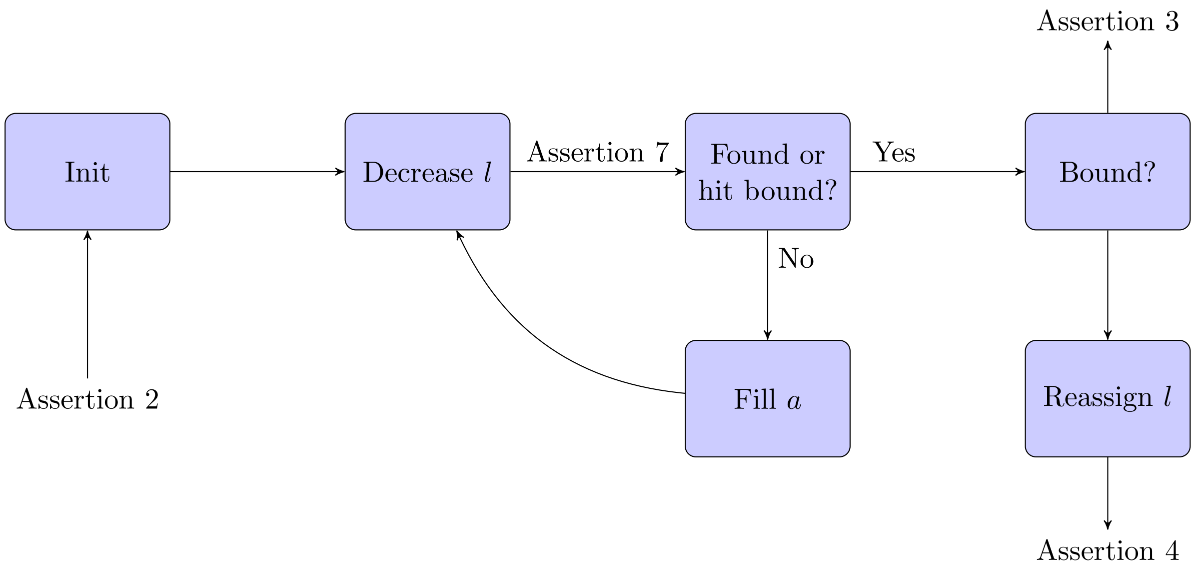

Figure 2: Flowchart of the move loop

Just like the flowchart of the main loop, we can make some assertions on the flow chart of the move loop. I will figure out the critical part, so that others could be discovered mechanically.
Assertion 7.

5 Analysis of the algorithm

References

[1] Donald E. Knuth: *The Art of Computer Programming*, Volume 1, Third edition.

[2] *One-Page Guide to Using Noweb with LaTeX*

[3] Johan Jeuring: *Finding palindromes efficiently*

[4] T Oetiker: *The Not So Short Introduction to LaTeX 2ε* (1995)

[5] Kjell Magne Fauske: *Example: Simple flow chart*

[6] The TikZ and PGF manual: *Example: State machine*