Chair for Algorithms
and Data Structures
Prof. Dr. Hannah Bast
Marjan Celikik

Search Engines WS 09/10

http://ad.informatik.uni-freiburg.de/teaching

Exercise Sheet 9 — Solutions

Exercise 1

Here is an algorithm that decides whether the UTF-8 multi-byte sequence starting at a given position in a string is valid or not:

1. If the first byte is of the form `0xxxxxxx`, the sequence is *valid* (all ASCII codes < 128 are valid UTF-8 codes).

2. If the first byte is of the form `10xxxxxx` or `11111xxx`, the sequence is invalid (the first byte must start with `0`, `110`, `1110`, or `11110`).

3. If the first byte is of the form `1100000x`, the sequence is invalid (2-byte UTF-8 sequence with code point < 128).

4. If the first byte is `11100000` and the second byte is of the form `100xxxxx`, the sequence is invalid (3-byte sequence with code point < 2048).

5. If the first byte is `11110000` and the second byte is of the form `1000xxxx`, the sequence is invalid (4-byte sequence with code point < 65536).

6. If the number of leading `1`s before the first `0` is m, and one of the next $m - 1$ bytes is not of the form `10xxxxxx`, the sequence is invalid (all bytes after the first must start with `10`).

7. All other sequences are valid.

Exercise 2

Here is a C++ implementation that implements the characterization from above in the most straightforward way (by Hannah).

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main(int argc, char** argv)
{
  unsigned int n = argc > 1 ? atoi(argv[1]) : 1000000;
  // Leave some space so that we don't have to check boundary conditions below.
  unsigned char* s = new unsigned char[n+3];
  for (int i = 0; i < n; ++i) s[i] = (unsigned char)(256 * drand48());
  clock_t t = clock();
  int i = 0;
```

```
  while (i < n)
  {
    // Count the number of leading 1s in the first byte.
    int l = s[i] < 128 ? 0 : (s[i] < 192 ? 1 : (s[i] < 224 ? 2 : (s[i] < 240 ? 3 : (s[i] < 248 ? 4 : 5)))
    // Should be 0, 2, 3, or 4, otherwise invalid first byte.
    if (l == 1 || l == 5) { s[i++] = 0; continue; }
    // Make sure that the next l - 1 bytes all start with 10.
    for (int j = i + 1; j < i + l; ++j) s[j] = s[j] & 63 + 128 ;
    // First byte must not be 1100000x.
    if (s[i] & 254 == 192) { s[i++] = 0; continue; }
    // First two bytes must not be 11100000 100xxxxx.
    if (s[i] == 224 && s[i+1] & 224 == 128) { s[i] = s[i+1] = 0; i += 2; continue; }
    // First two bytes must not be 11110000 1000xxxx.
    if (s[i] == 240 && s[i+1] & 240 == 128) { s[i] = s[i+1] = s[i+2] = 0; i += 3; continue; }
  }
  printf("Corrected random %d-byte string in %.2f seconds\n", n, (double)(clock() - t)/CLOCKS_PER_SEC);
}
```

Exercise 3

Here is Marjan's solution for JAVA.

```
import java.util.*;
public class Ex9
{
public static void main(String[] args) {
    int n = Integer.parseInt(args[0]);
    char[] seq = new char[n];
    Random rand = new Random();
    for(int i = 0; i < n; i++)
      seq[i] = (char)(1 + rand.nextInt(255));
    long startTime = System.nanoTime();
    for(int i = 0; i < n; i++) {
      if (seq[i] > 127 && seq[i] < 224) {  // start of a 2-byte UTF-8 sequence
        if (seq[i] < 194) {  // invalid first byte
          seq[i] = '0';
          if (i < n - 1)
            if (seq[i+1] > 127) seq[i+1] = '0';
        }
        else {  // first byte ok
          if (i < n - 1)
            seq[i] = '0';
          else
          if (seq[i+1] < 128) {    // invalid second byte
              seq[i+1] = '0'; seq[i] = '0';
            }
```

```
  }
}
else
if (seq[i] >= 224 && seq[i] < 240) { // start of a 3-byte UTF-8 sequence
  if (i < n - 1) {
    if (seq[i+1] < 160 || seq[i+1] > 191) {  // invalid second byte
      seq[i] = '0'; seq[i+1] = '0';
      if (i < n - 2)  // is there a third byte?
        if (seq[i+2] > 127)
          seq[i+2] = '0';
    }
    else {  // second byte ok
      if (i < n - 2)  // is there a third byte?
       if (seq[i+2] < 128 || seq[i+2] > 191) {  // invalid third byte
          seq[i] = '0'; seq[i+1] = '0'; seq[i+2] = '0';
        }
        else {  // no third byte!
        seq[i] = '0'; seq[i+1] = '0';
      }
    }
  }
  else  // no second byte
    seq[i] = '0';
}
else
if (seq[i] >= 240) {  // 4-byte UTF-8 sequence
  if (i == n - 1)
    seq[i] = '0';
  else
    if (i == n - 2) { seq[i] = '0'; seq[i+1] = '0'; }
    else
      if (i == n - 3) { seq[i] = '0'; seq[i+1] = '0'; seq[i+2] = '0'; }
      else
      if (i == n - 4) {
          seq[i] = '0'; seq[i+1] = '0'; seq[i+2] = '0'; seq[i+3] = '0';
        }
        else {
          if (seq[i+1] < 144 || seq[i+1] > 191) {  // invalid second
            seq[i] = '0'; seq[i+1] = '0';
            if (seq[i+2] > 127) {
              seq[i+2] = '0';
              if (seq[i+3] > 127) seq[i+3] = '0';
            }
          }
          else {  // second byte ok
            if (seq[i+2] < 128  || seq[i+2] > 191) { // invalid third
              seq[i] = '0'; seq[i+1] = '0'; seq[i+2] = '0';
```

```
                    if (seq[i+3] > 127) seq[i+3] = '0';
                  }
                  else { // third byte ok
                    if (seq[i+3] < 128  || seq[i+3] > 191) { // invalid fourth
                      seq[i] = '0'; seq[i+1] = '0'; seq[i+2] = '0'; seq[i+3] = '0';
                    }
                  }
                }
              }
      }
    }
    long stopTime = System.nanoTime() - startTime;
    System.out.println(" done. " + stopTime / 1000 + " usec.");
  }
}
```

Exercise 4

Here is Marjan's solution for Perl.

```
#!/usr/bin/perl/
use Time::HiRes;
use warnings;
$n = $ARGV[0];
@seq=();
for($i = 0; $i < $n; $i++)
{
  $seq[$i] = 1 + int(rand(255));
}
$start = [Time::HiRes::gettimeofday()];
for($i = 0; $i < $n; $i++)
{
  if ($seq[$i] > 127 && $seq[$i] < 224) {  # start of a 2-byte UTF-8 $sequence
    if ($seq[$i] < 194) {  # invalid first bytepr1
      $seq[$i] = 0;
      if ($i < $n - 1)
      {
        if ($seq[$i+1] > 127) {
          $seq[$i+1] = 0;
        }
      }
    }
    else  # first byte ok
    {
      if ($i < $n - 1) {
        $seq[$i] = 0;
```

```
    }
    else {
      if ($seq[$i+1] < 128) {  # invalid second byte
        $seq[$i+1] = 0; $seq[$i] = 0;
      }
    }
  }
}
else {
  if ($seq[$i] >= 224 && $seq[$i] < 240) { # start of a 3-byte UTF-8 $seq
    if ($i < $n - 1) {
      if ($seq[$i+1] < 160 || $seq[$i+1] > 191) {  # invalid second byte
        $seq[$i] = 0; $seq[$i+1] = 0;
        if ($i < $n - 2) {  # is there a third byte?
          if ($seq[$i+2] > 127) {
            $seq[$i+2] = 0;
          }
        }
      }
      else {  # second byte ok
        if ($i < $n - 2) {  # is there a third byte?
          if ($seq[$i+2] < 128 || $seq[$i+2] > 191) {  # invalid third byte
            $seq[$i] = 0; $seq[$i+1] = 0; $seq[$i+2] = 0;
          }
        }
        else {  # no third byte!
          $seq[$i] = 0; $seq[$i+1] = 0;
        }
      }
    }
    else {  # no second byte
      $seq[$i] = 0;
    }
  }
  else {
    if ($seq[$i] >= 240) {  # 4-byte UTF-8 $sequence
      if ($i == $n - 1) { $seq[$i] = 0; }
      else {
        if ($i == $n - 2) { $seq[$i] = 0; $seq[$i+1] = 0; }
        else {
          if ($i == $n - 3) {
            $seq[$i] = 0; $seq[$i+1] = 0; $seq[$i+2] = 0;
          }
          else {
            if ($i == $n - 4) {
              $seq[$i] = 0; $seq[$i+1] = 0; $seq[$i+2] = 0; $seq[$i+3] = 0;
            }
```

```
            else {
              if ($seq[$i+1] < 144 || $seq[$i+1] > 191) { # invalid second
                $seq[$i] = 0; $seq[$i+1] = 0;
                if ($seq[$i+2] > 127) {
                  $seq[$i+2] = 0;
                  if ($seq[$i+3] > 127) {
                    $seq[$i+3] = 0;
                  }
                }
              }
              else {  # second byte ok
                if ($seq[$i+2] < 128  || $seq[$i+2] > 191) { # invalid third
                  $seq[$i] = 0; $seq[$i+1] = 0; $seq[$i+2] = 0;
                  if ($seq[$i+3] > 127) {
                    $seq[$i+3] = 0;
                  }
                }
                else {  # third byte ok
                  if ($seq[$i+3] < 128  || $seq[$i+3] > 191) { # invalid fou.
                    $seq[$i] = 0; $seq[$i+1] = 0; $seq[$i+2] = 0;
                    $seq[$i+3] = 0;
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
$finish = 1000000 * Time::HiRes::tv_interval($start);
print "done. ".$finish." usec.\n";
```

Exercise 5

Input Size	*C++*	*JAVA*	*PERL*
10^3 B	0.01 ms	0.08 ms	0.86 ms
10^6 B	8.5 ms	15.2 ms	658 ms
10^9 B	8854 ms	8230 ms	10 min

Table 1: Average running times of the three impementations. All experiments were carried out on a single machine under low load (Quad-core Xeon X5560 @ 2.8 GHz)

As expected, the compiled or non-interpreted languages like JAVA and C++ perform significantly

faster than interpreted programming languages like Perl (here almost two orders of magnitute or 100 times faster). As for JAVA and C++, it is generally accepted that C++ compilers perform generally faster than JAVA (not much faster though, and not all C++ compilers). In our own example, C++ performs significantly faster than JAVA only when the size of the input is small, with very small difference for very large input.