

# 实验四：文本索引

张俊华 16030199025

## 一、实验内容

编写一个构建大块文本索引的程序，然后进行快速搜索，来查找某个字符串在该文本中的出现位置。

## 二、实验环境

IntelliJ IDEA 2018.2.5 (Ultimate Edition)

JRE: 1.8.0\_152-release-1248-b19 amd64

JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

Windows 10 10.0

## 三、实验步骤

### 1. 构建后缀数组

使用 c++ 的流操作运算，从 txt 文件中读取待查找文本。将文件内容保存在 `str` 字符串中。

```
1  std::ifstream ifstream("D:\\alice29.txt");
2  std::stringstream stream;
3  ifstream.seekg(0, std::ios::end);    // go to the end
4  int length = ifstream.tellg();      // report location (this is the length)
5  std::cout << "Input File length: " << length << std::endl;
6  ifstream.seekg(0, std::ios::beg);    // go back to the beginning
7  ifstream.read(str, length);          // read the whole file into the buffer
8  ifstream.close();                   // close file handle
```

### 2. 编写 suffixSort() 函数，实现后缀数组的排序

后缀数组保存在 pos[] 数组中，后缀数组的逆保存在 rank 数组中。使用 MSD 算法进行实现

```
1  void suffixSort(int n){
2      //sort suffixes according to their first characters
3      for (int i=0; i<n; ++i){
4          pos[i] = i;
5      }
6      std::sort(pos, pos + n, smaller_first_char);
7      //{pos contains the list of suffixes sorted by their first character}
8
9      for (int i=0; i<n; ++i){
10         bh[i] = i == 0 || str[pos[i]] != str[pos[i-1]];
11         b2h[i] = false;
```

```

12     }
13
14     for (int h = 1; h < n; h <= 1){
15         //{bh[i] == false if the first h characters of pos[i-1] == the first h
characters of pos[i]}
16         int buckets = 0;
17         for (int i=0, j; i < n; i = j){
18             j = i + 1;
19             while (j < n && !bh[j]) j++;
20             next[i] = j;
21             buckets++;
22         }
23         if (buckets == n) break; // We are done! Lucky bastards!
24         //{suffixes are separted in buckets containing strings starting with the same h
characters}
25
26         for (int i = 0; i < n; i = next[i]){
27             cnt[i] = 0;
28             for (int j = i; j < next[i]; ++j){
29                 rank[pos[j]] = i;
30             }
31         }
32
33         cnt[rank[n - h]]++;
34         b2h[rank[n - h]] = true;
35         for (int i = 0; i < n; i = next[i]){
36             for (int j = i; j < next[i]; ++j){
37                 int s = pos[j] - h;
38                 if (s >= 0){
39                     int head = rank[s];
40                     rank[s] = head + cnt[head]++;
41                     b2h[rank[s]] = true;
42                 }
43             }
44             for (int j = i; j < next[i]; ++j){
45                 int s = pos[j] - h;
46                 if (s >= 0 && b2h[rank[s]]){
47                     for (int k = rank[s]+1; !bh[k] && b2h[k]; k++) b2h[k] = false;
48                 }
49             }
50         }
51         for (int i=0; i<n; ++i){
52             pos[rank[i]] = i;
53             bh[i] |= b2h[i];
54         }
55     }
56     for (int i=0; i<n; ++i){
57         rank[pos[i]] = i;
58     }
59 }
60 void getHeight(int n){
61     for (int i=0; i<n; ++i) rank[pos[i]] = i;
62     height[0] = 0;

```

```

63     for (int i=0, h=0; i<n; ++i){
64         if (rank[i] > 0){
65             int j = pos[rank[i]-1];
66             while (i + h < n && j + h < n && str[i+h] == str[j+h]) h++;
67             height[rank[i]] = h;
68             if (h > 0) h--;
69         }
70     }
71 }

```

### 3. 编写二分查找函数

编写 `binarychop` 函数，利用二分查找，实现对输入的 key 关键字的查找匹配

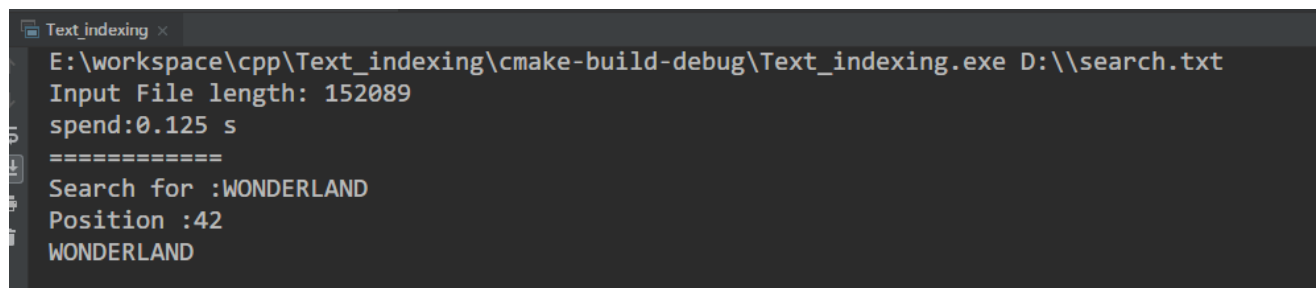
```

1  int binarychop(char* key, int key_length, int left, int right){
2      if (left > right){
3          return -1;
4      }
5      int mid = (right-left)/2+left;
6      int p = pos[mid];
7      for (int i = 0; i < key_length; i++){
8          if (key[i]<str[p+i]){
9              return binarychop(key,key_length,left,mid-1);
10         } else if (key[i] > str[p+i]){
11             return binarychop(key,key_length,mid+1,right);
12         }
13     }
14     return p;
15 }

```

## 四、实验结果

使用 `alice29.txt` 文本数据进行测试（长度：152089）可以在 0.125s 实现对整个后缀数组排序



```

Text_indexing x
E:\workspace\cpp\Text_indexing\cmake-build-debug\Text_indexing.exe D:\\search.txt
Input File length: 152089
spend:0.125 s
=====
Search for :WONDERLAND
Position :42
WONDERLAND

```

```
Text_indexing x
=====
Search for :chain
Position :697
chain would be worth the trouble
of getting up and picking t
=====
Search for :would
Position :106466
would cost them their lives.

    All the time they were playi
=====
Search for :be
Position :57817
believe it,' said the Pigeon; `but if they do, why
then they
=====
Search for :of
Position :40577
of the window, I only wish they COULD! I'm sure I
don't wan
=====
Search for :the
```

## 附：实验完整代码

```
1  #include <iostream>
2  #include <algorithm>
3  #include <sstream>
4  #include <cstring>
5  #include <fstream>
6  #include <ctime>
7
8  const int N = 2000000;
9
10 char str[N]; //input
11 int rank[N], pos[N]; //output
12 int cnt[N], next[N]; //internal
13 bool bh[N], b2h[N];
14
```

```

15 // Compares two suffixes according to their first characters
16 bool smaller_first_char(int a, int b){
17     return str[a] < str[b];
18 }
19
20 void suffixSort(int n){
21     //sort suffixes according to their first characters
22     for (int i=0; i<n; ++i){
23         pos[i] = i;
24     }
25     std::sort(pos, pos + n, smaller_first_char);
26     //{pos contains the list of suffixes sorted by their first character}
27
28     for (int i=0; i<n; ++i){
29         bh[i] = i == 0 || str[pos[i]] != str[pos[i-1]];
30         b2h[i] = false;
31     }
32
33     for (int h = 1; h < n; h <= 1){
34         //{bh[i] == false if the first h characters of pos[i-1] == the first h
characters of pos[i]}
35         int buckets = 0;
36         for (int i=0, j; i < n; i = j){
37             j = i + 1;
38             while (j < n && !bh[j]) j++;
39             next[i] = j;
40             buckets++;
41         }
42         if (buckets == n) break;
43         //{suffixes are parted in buckets containing strings starting with the same
h characters}
44
45         for (int i = 0; i < n; i = next[i]){
46             cnt[i] = 0;
47             for (int j = i; j < next[i]; ++j){
48                 rank[pos[j]] = i;
49             }
50         }
51
52         cnt[rank[n - h]]++;
53         b2h[rank[n - h]] = true;
54         for (int i = 0; i < n; i = next[i]){
55             for (int j = i; j < next[i]; ++j){
56                 int s = pos[j] - h;
57                 if (s >= 0){
58                     int head = rank[s];
59                     rank[s] = head + cnt[head]++;
60                     b2h[rank[s]] = true;
61                 }
62             }
63             for (int j = i; j < next[i]; ++j){
64                 int s = pos[j] - h;
65                 if (s >= 0 && b2h[rank[s]]){

```

```

66         for (int k = rank[s]+1; !bh[k] && b2h[k]; k++) b2h[k] = false;
67     }
68 }
69 }
70 for (int i=0; i<n; ++i){
71     pos[rank[i]] = i;
72     bh[i] |= b2h[i];
73 }
74 }
75 for (int i=0; i<n; ++i){
76     rank[pos[i]] = i;
77 }
78 }
79 // End of suffix array algorithm
80
81
82
83 int height[N];
84
85 void getHeight(int n){
86     for (int i=0; i<n; ++i) rank[pos[i]] = i;
87     height[0] = 0;
88     for (int i=0, h=0; i<n; ++i){
89         if (rank[i] > 0){
90             int j = pos[rank[i]-1];
91             while (i + h < n && j + h < n && str[i+h] == str[j+h]) h++;
92             height[rank[i]] = h;
93             if (h > 0) h--;
94         }
95     }
96 }
97 // End of longest common prefixes algorithmd
98
99 int binarychop(char* key, int key_length, int left, int right){
100     if (left > right){
101         return -1;
102     }
103     int mid = (right-left)/2+left;
104     int p = pos[mid];
105     for (int i = 0; i < key_length; i++){
106         if (key[i]<str[p+i]){
107             return binarychop(key,key_length,left,mid-1);
108         } else if (key[i] > str[p+i]){
109             return binarychop(key,key_length,mid+1,right);
110         }
111     }
112     return p;
113 }
114
115 int main(int argc, char ** argv) {
116     std::ifstream ifstream("D:\\alice29.txt");
117     std::stringstream stream;
118     ifstream.seekg(0, std::ios::end);    // go to the end

```

```

119     int length = ifstream.tellg();           // report location (this is the
length)
120     std::cout << "Input File length: " << length << std::endl;
121     ifstream.seekg(0, std::ios::beg);       // go back to the beginning
122     ifstream.read(str, length);             // read the whole file into the buffer
123     ifstream.close();                       // close file handle
124     clock_t start,end;
125     start = clock();
126     suffixSort(strlen(str));
127     end = clock();
128     std::cout<<"spend:"<< (double)(end-start)/CLOCKS_PER_SEC << " s" << std::endl;
129
130     std::ifstream search(argv[1]);
131     while (search.peek() != EOF){
132         std::cout << "======" << std::endl;
133         char key[1000];
134         search >> key;
135         search.get();
136         std::cout << "Search for :" << key << std::endl;
137         int p = binarychop(key, strlen(key),0,strlen(str));
138         std::cout << "Position :" << p << "    " << std::endl;
139         for (int i = 0 ; i < 60 ; i++){
140             std::cout << str[p+i] ;
141         }
142         std::cout << std::endl;
143     }
144
145     return 0;
146 }

```