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。

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

```
void suffixSort(int n){
  //sort suffixes according to their first characters
  for (int i=0; i<n; ++i){
    pos[i] = i;
  }
  std::sort(pos, pos + n, smaller_first_char);
  //{pos contains the list of suffixes sorted by their first character}

  for (int i=0; i<n; ++i){
    bh[i] = i == 0 || str[pos[i]] != str[pos[i-1]];
    b2h[i] = false;
```

```
  }

  for (int h = 1; h < n; h <<= 1){
    //{bh[i] == false if the first h characters of pos[i-1] == the first h characters of pos[i]}
    int buckets = 0;
    for (int i=0, j; i < n; i = j){
      j = i + 1;
      while (j < n && !bh[j]) j++;
      next[i] = j;
      buckets++;
    }
    if (buckets == n) break; // We are done! Lucky bastards!
    //{suffixes are separted in buckets containing strings starting with the same h characters}

    for (int i = 0; i < n; i = next[i]){
      cnt[i] = 0;
      for (int j = i; j < next[i]; ++j){
        rank[pos[j]] = i;
      }
    }

    cnt[rank[n - h]]++;
    b2h[rank[n - h]] = true;
    for (int i = 0; i < n; i = next[i]){
      for (int j = i; j < next[i]; ++j){
        int s = pos[j] - h;
        if (s >= 0){
          int head = rank[s];
          rank[s] = head + cnt[head]++;
          b2h[rank[s]] = true;
        }
      }
      for (int j = i; j < next[i]; ++j){
        int s = pos[j] - h;
        if (s >= 0 && b2h[rank[s]]){
          for (int k = rank[s]+1; !bh[k] && b2h[k]; k++) b2h[k] = false;
        }
      }
    }
    for (int i=0; i<n; ++i){
      pos[rank[i]] = i;
      bh[i] |= b2h[i];
    }
  }
  for (int i=0; i<n; ++i){
    rank[pos[i]] = i;
  }
}
void getHeight(int n){
  for (int i=0; i<n; ++i) rank[pos[i]] = i;
  height[0] = 0;
```

```
  for (int i=0, h=0; i<n; ++i){
    if (rank[i] > 0){
      int j = pos[rank[i]-1];
      while (i + h < n && j + h < n && str[i+h] == str[j+h]) h++;
      height[rank[i]] = h;
      if (h > 0) h--;
    }
  }
}
```

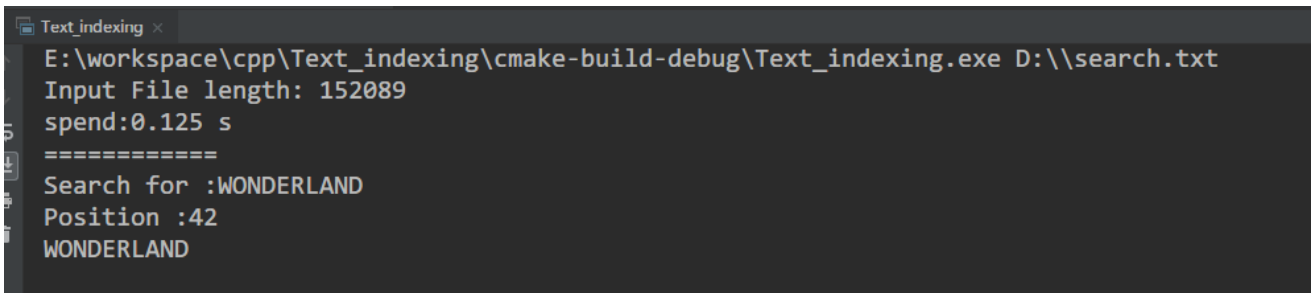
3. 编写二分查找函数

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

```
int binarychop(char* key, int key_lengh, int left, int right){
    if (left > right){
        return -1;
    }
    int mid = (right-left)/2+left;
    int p = pos[mid];
    for (int i = 0; i < key_lengh; i++){
        if (key[i]<str[p+i]){
            return binarychop(key,key_lengh,left,mid-1);
        } else if (key[i] > str[p+i]){
            return binarychop(key,key_lengh,mid+1,right);
        }
    }
    return p;
}
```

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

============
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

```cpp
#include <iostream>
#include <algorithm>
#include <sstream>
#include <cstring>
#include <fstream>
#include <ctime>

const int N = 2000000;

char str[N]; //input
int rank[N], pos[N]; //output
int cnt[N], next[N]; //internal
bool bh[N], b2h[N];

```

```
// Compares two suffixes according to their first characters
bool smaller_first_char(int a, int b){
  return str[a] < str[b];
}

void suffixSort(int n){
  //sort suffixes according to their first characters
  for (int i=0; i<n; ++i){
    pos[i] = i;
  }
  std::sort(pos, pos + n, smaller_first_char);
  //{pos contains the list of suffixes sorted by their first character}

  for (int i=0; i<n; ++i){
    bh[i] = i == 0 || str[pos[i]] != str[pos[i-1]];
    b2h[i] = false;
  }

  for (int h = 1; h < n; h <<= 1){
    //{bh[i] == false if the first h characters of pos[i-1] == the first h characters of pos[i]}
    int buckets = 0;
    for (int i=0, j; i < n; i = j){
      j = i + 1;
      while (j < n && !bh[j]) j++;
      next[i] = j;
      buckets++;
    }
    if (buckets == n) break;
    //{suffixes are separted in buckets containing strings starting with the same h characters}

    for (int i = 0; i < n; i = next[i]){
      cnt[i] = 0;
      for (int j = i; j < next[i]; ++j){
        rank[pos[j]] = i;
      }
    }

    cnt[rank[n - h]]++;
    b2h[rank[n - h]] = true;
    for (int i = 0; i < n; i = next[i]){
      for (int j = i; j < next[i]; ++j){
        int s = pos[j] - h;
        if (s >= 0){
          int head = rank[s];
          rank[s] = head + cnt[head]++;
          b2h[rank[s]] = true;
        }
      }
      for (int j = i; j < next[i]; ++j){
        int s = pos[j] - h;
        if (s >= 0 && b2h[rank[s]]){
```

```
          for (int k = rank[s]+1; !bh[k] && b2h[k]; k++) b2h[k] = false;
        }
      }
    }
    for (int i=0; i<n; ++i){
      pos[rank[i]] = i;
      bh[i] |= b2h[i];
    }
  }
  for (int i=0; i<n; ++i){
    rank[pos[i]] = i;
  }
}
// End of suffix array algorithm



int height[N];

void getHeight(int n){
  for (int i=0; i<n; ++i) rank[pos[i]] = i;
  height[0] = 0;
  for (int i=0, h=0; i<n; ++i){
    if (rank[i] > 0){
      int j = pos[rank[i]-1];
      while (i + h < n && j + h < n && str[i+h] == str[j+h]) h++;
      height[rank[i]] = h;
      if (h > 0) h--;
    }
  }
}
// End of longest common prefixes algorithmd

int binarychop(char* key, int key_lengh, int left, int right){
    if (left > right){
        return -1;
    }
    int mid = (right-left)/2+left;
    int p = pos[mid];
    for (int i = 0; i < key_lengh; i++){
        if (key[i]<str[p+i]){
            return binarychop(key,key_lengh,left,mid-1);
        } else if (key[i] > str[p+i]){
            return binarychop(key,key_lengh,mid+1,right);
        }
    }
    return p;
}

int main(int argc, char ** argv) {
    std::ifstream ifstream("D:\\alice29.txt");
    std::stringstream stream;
    ifstream.seekg(0, std::ios::end);    // go to the end
```

```
    int length = ifstream.tellg();           // report location (this is the length)
    std::cout << "Input File length: " << length << std::endl;
    ifstream.seekg(0, std::ios::beg);    // go back to the beginning
    ifstream.read(str, length);       // read the whole file into the buffer
    ifstream.close();                    // close file handle
    clock_t start,end;
    start = clock();
    suffixSort(strlen(str));
    end = clock();
    std::cout<<"spend:"<< (double)(end-start)/CLOCKS_PER_SEC << " s" << std::endl;

    std::ifstream search(argv[1]);
    while (search.peek()!=EOF){
        std::cout << "============" << std::endl;
        char key[1000];
        search >> key;
        search.get();
        std::cout << "Search for :" << key << std::endl;
        int p = binarychop(key, strlen(key),0,strlen(str));
        std::cout << "Position :" << p << "   " << std::endl;
        for (int i = 0 ;i < 60 ; i++){
            std::cout << str[p+i] ;
        }
        std::cout << std::endl;
    }

    return 0;
}
```