Group 02:
+ Lê Dương Công Đức
+ Nguyễn Hoàng Gia Bảo
+ Trương Hữu Đức

CS163 Final Project Proposal

Data structure:

In this project, we you tries to store data. Tries is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree defines the key with which it is associated. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are not necessarily associated with every node. Rather, values tend only to be associated with leaves, and with some inner nodes that correspond to keys of interest.

Thus we have only one trie, e.i. all words in these files will be stored in the same trie.

```
struct Trie {

	Node *root = new Node;
	~Trie();
	void Destruct(details*);
	void Destruct(Node*);
	void Insert(const char*,int,int); // word, file_id, pos_in_content
	Node *Find(char*);
	void Show_trie(Node *cur,string s ="");

};
```

Trie has nodes. If a certain word is a complete word, in the “insert to trie” process, the linked-list of struct details *info contain:

- “int file_id” : a specific number represent for filename.
- details *next : is the pointer to the next file that contain keyword;
- vector<int> pos : is an array contain the position of that word in the current file.

```
struct details {
    int file_id;
    details *next = NULL;
    vector<int> pos;

};
```

Thus, with 1 Node we will know: Which file contain that word, where they are and the number of occurrences (by pos.size() method).

```
class Node {
public:
    Node() {for (int i=0;i<256;++i) next[i] = NULL;}
    Node *next[256];
    details *info = NULL; // info is a linkedlist
    int point = 0; // the times this word appeared in the whole trie

};
```

Group 02:
+ Lê Dương Công Đức
+ Nguyễn Hoàng Gia Bảo
+ Trương Hữu Đức

```
struct Data {
    char filename[20]; // show file name ex. Data1005.txt
    char *title = NULL;
    char *content = NULL; // pointer point to the string contain file
content.

};
```

struct Data use to show the paragraph that contain keyword in the console window.

Trie-manipulate:

- ### Insertion:

  The inserted word is lead to the suitable place, update the information (file, position, point).

```
void Trie::Insert(const char* s,int file_id,int pos) {
    Node *cur = root;
    for (int i=0;i<strlen(s);++i) {
        if (cur->next[s[i]] == NULL)
            cur->next[s[i]] = new Node;
        cur = cur->next[s[i]];
    }
    cur->point++;
    if (cur->info == NULL) {
        cur->info = new details;
        cur->info->file_id = file_id;
        cur->info->pos.push_back(pos);
    }
    else {
        details *info_cur = cur->info, *info_prev;
        while (info_cur != NULL && info_cur->file_id != file_id)
            info_prev = info_cur, info_cur = info_cur->next;
        if (info_cur == NULL) {
            info_prev->next = new details;
            info_cur = info_prev->next;
            info_cur->file_id = file_id;
        }
        info_cur->pos.push_back(pos);
    }

}
```

Group 02:
+ Lê Dương Công Đức
+ Nguyễn Hoàng Gia Bảo
+ Trương Hữu Đức

- ## Searching:

  The char* word point to the keyword, while char c is the prefix (such as #,$ and blank-space " " for nothing.

  When Node found, It is returned, thus we got the linked list contain information about the keyword (where it is, position, number of occurrence), otherwise NULL be returned.

```
Node* Trie::Find(char c, char* word){

if (word == "\0") return NULL;
Node* cur = root;
int cursor = 0;
for (;cursor < strlen(word);cursor++)
      if (cur->next[int(word[cursor])] != NULL)
            cur = cur->next[int(word[cursor])];
      else break;
if (cursor < strlen(word)) return NULL;
if (c == ' ') return cur;
if (c == '#' || c=='$'){
      Node* result = new Node;
      result->info = new detail;
      details *cur_res=result->info ,*cur_word = cur->info;

      for (;cur_word != NULL;cur_word = cur_word->next){

      for (details *tmp = root[int(c)].info, tmp != NULL;tmp = tmp->next){

            if (tmp->file_id == cur_word->file_id){

            for (vector<int>::iterator i = tmp->pos.begin();i < tmp->pos.end();i++)
            for (vector<int>::iterator j = cur_word->pos.begin();j < cur_word->pos.end();j++)
                  if (*i + 2 == *j){
                  cur_res->next = new details;
                  cur_res = cur_res->next;
                  cur_res->file_id = tmp->file_id;
                  cur_res->pos.push_back(*i);
                  cur_res->next = NULL;
                  }
            }
      }
      }
            cur_res = result->info;
            result->info = result->info->next;
            delete cur_res;
            return result;
}
      return NULL;

}
```

Group 02:
+ Lê Dương Công Đức
+ Nguyễn Hoàng Gia Bảo
+ Trương Hữu Đức

Solution for query process:

1. **AND** (A AND B)
   We will search each term independently. One file is fine if it contains all input operand of AND operator.
2. **OR** (A OR B)
   We will search each term independently. One file is fine if it contains only one term of them.
3. **– (e.g. Manchester -united)**
   For example: A -B.
   We just need to search for **A** in this trie. If a file has **A** we'll continue to search for **B** in this trie. One file is fine if it contains **A** and doesn't contain **B**
4. ***+ (e.g. Peanut Butter +"and Jam")***
   For example: A +B.
   If a file has **A** we'll continue to search for **B** in this trie. One file is fine if it contains **A** and be ranking up if contain **B**
5. **In title:**
   For example: intitle:debate
   One file is fine if in its title contain keyword.
   For example: intitle:"debate in NewYork"
   One file is fine if in its title contain exact the whole phrase "debate in NewYork"
6. **Filetype:**
   For example: intitle:txt
   First its point (pos.size() method) must bigger than zero, e.i. It contains keywords in query. Then we check the file extension, matched ones will be ranking and show in console.
7. **Hashtag #**
   Thank to vector<int> pos we can determine the position of query. Thus, one file is fine if it contain all keyword and position of sharp(#) are ordered in font of keyword.
8. **Dollar $**
   Thank to vector<int> pos we can determine the position of query. Thus, one file is fine if it contain all keyword and position of Dollar($) are ordered in font of keyword.
9. **Exact search with speech mark " ":**
   Thank to vector<int> pos we can determine the position of query. Thus, one file is fine if it contain all keyword and position of them are ordered continuously.