DSAP期末報告

英文字彙小幫手

B11705024謝友毅

DSAP期末報告

英文字彙小幫手

B11705024 資管一 謝友毅

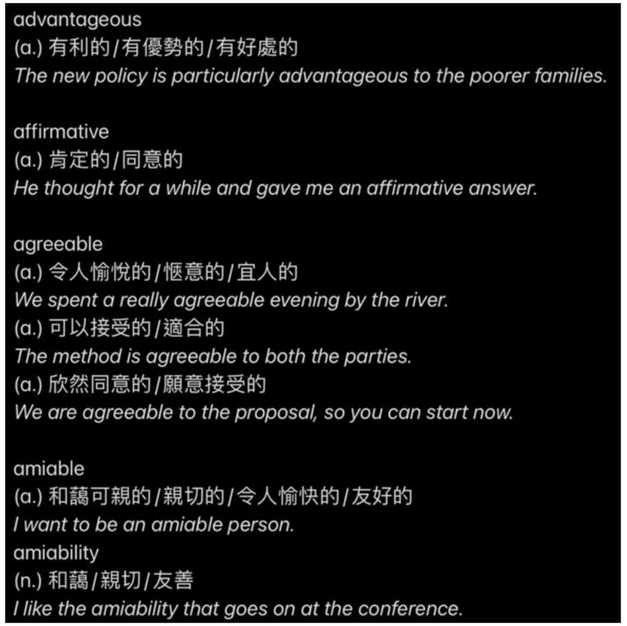
情境說明

比利有在手機記事本上作英文筆記的習慣，他會記下單字、解釋和例句，並按照單字的字典順序找尋插入位置，但隨著紀錄的單字越來越多，問題也開始變多

1.要尋找新單字的插入位置變得越來越麻煩費時

2.偶爾眼花會將新單字插入錯誤的位置，而沒有按照字典順序

3.查找已經記錄過的單字實屬不容易



解決方案

比利打算利用上課所學實作「 英文字彙小幫手」，協助把先前的筆記整理好，匯入程式中，並使用有效率的資料結構來執行新增單字、刪除單字、查找單字的功能

整理筆記格式

新增單字　　刪除單字　　查找單字

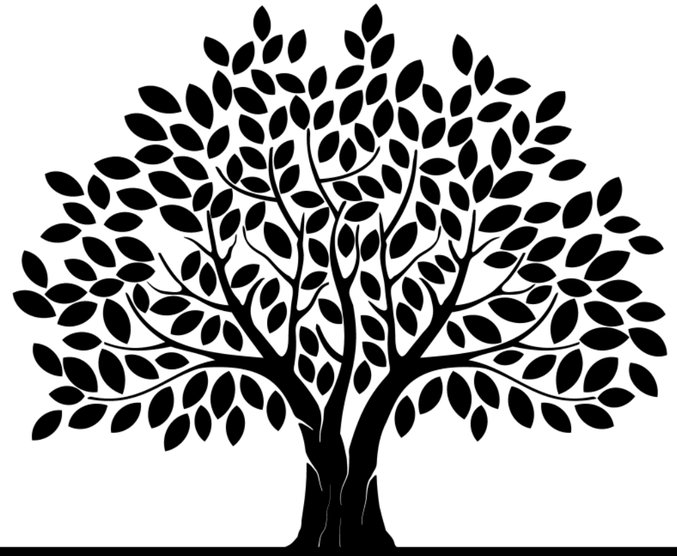
資料結構

[自行實作]

Trie(字首樹、字典樹)：

每個節點都儲存一個字母(a~z, A~Z），且任一個節點的子孫都有相同的字首

時間複雜度：新增O(k)、刪除O(k)、查找O(k)（k為單字大小）



[C++標準函式庫]

Set(集合)：

實作通常為紅黑樹

上課有提到，不多作敘述

A

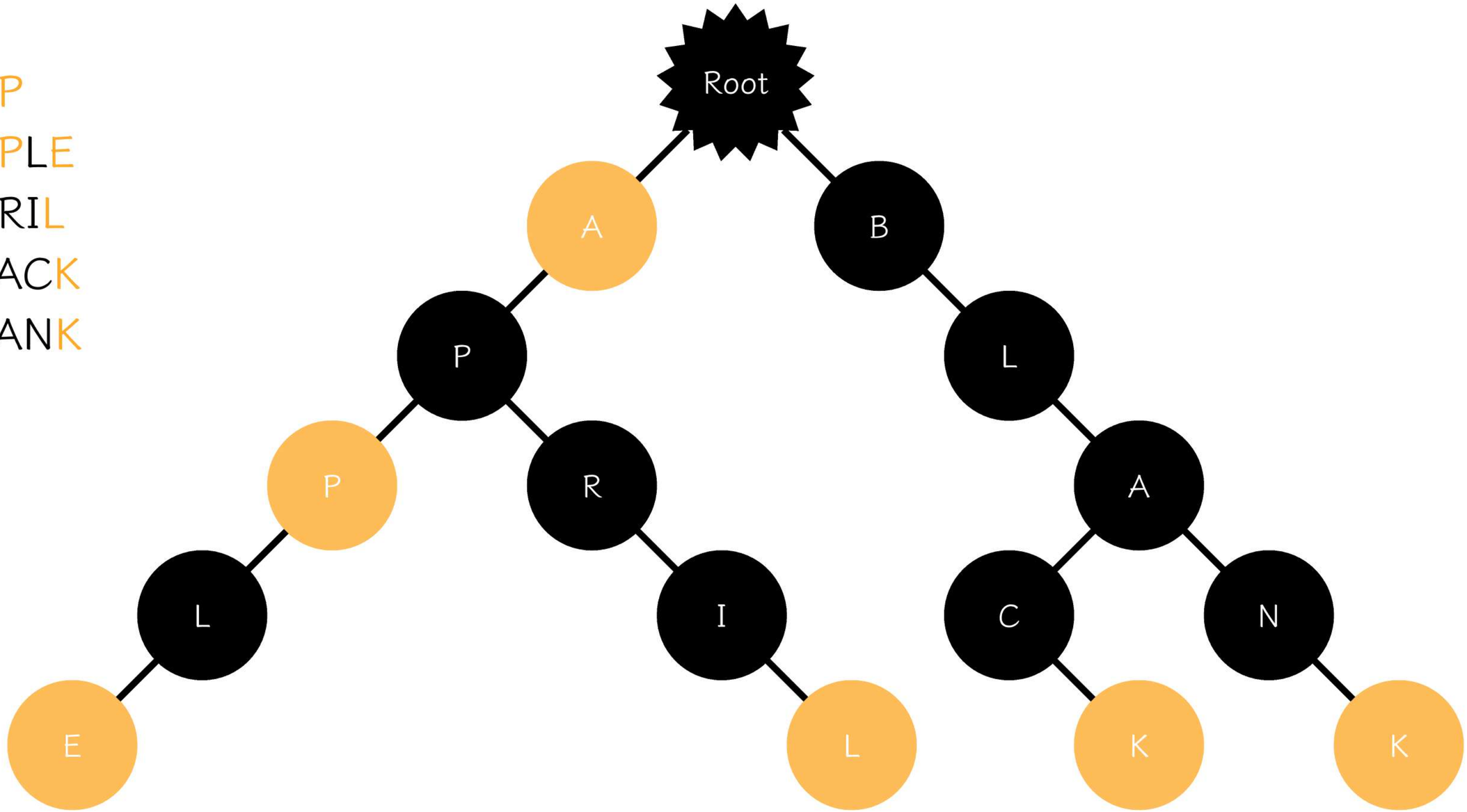
APP

APPLE

APRIL

BLACK

BLANK



預期結果

[時間]

新增：

Trie較快 ，因為不用比較單字字典順序，且字首已存在時會更快，加上不用平衡

刪除：

Trie較快，因為不用比較單字字典順序，且最好情況可以只將節點取消標示為單字結尾，加上不用平衡

查找：

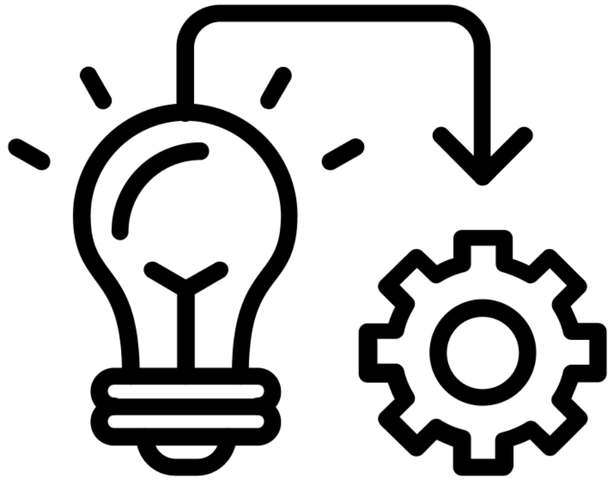
Trie較快，因為知道字首不存在時即可結束

[空間]

因為Trie每次新增節點需要new出全部字母數量的陣列，因此可能較佔空間

程式實作

1. 資料結構&功能實作
2. 整理先前的筆記或單字庫
3. 匯入整理好的筆記或單字庫
4. 進行個別功能測試以及比較效能指標



Trie 節點

```
24  const int CHARACTERS_SIZE = 52;
25
26  class TrieNode {
27  public:
28      bool is_end_of_word;
29      TrieNode* children[CHARACTERS_SIZE];
30      std::string examples = "";
31
32      TrieNode() {
33          is_end_of_word = false;
34          for (int i = 0; i < CHARACTERS_SIZE; i++) {
35              children[i] = nullptr;
36          }
37      }
38  };
```

可支援的字元數量(a~z, A~Z)

標記是否為某單字的結尾

Trie

```
40    class Trie {
41
42    private:
43        TrieNode* root;
44
45    public:
46        Trie() {
47            root = new TrieNode();
48        }
```

新增單字

```
52      void Add(const std::string& word, const std::string& example) {
53          TrieNode* current = root;
54          for (int i = 0; i < word.size(); i++) {
55              char ch = word[i];
56              int index;
57              if (std::isupper(ch)) {
58                  index = ch - 'A';
59              } else if (std::islower(ch)) {
60                  index = ch - 'a' + 26;
61              }
62
63              if (current->children[index] == nullptr) {
64                  current->children[index] = new TrieNode();
65              }
66              current = current->children[index];
67          }
68          current->is_end_of_word = true;
69
70          if (example != "") {
71              current->examples = example;
72          }
73      }
```

已經有存在的字母節點時，直接前往，否則新增節點

到達最後一個字母時，將節點標示為單字字尾

查詢單字

```
181     TrieNode* SearchVocabularyPos(const std::string& word) const {
182         TrieNode* current = root;
183         for (int i = 0; i < word.size(); i++) {
184             char ch = word[i];
185             int index;
186             if (std::isupper(ch)) {
187                 index = ch - 'A';
188             } else if (std::islower(ch)) {
189                 index = ch - 'a' + 26;
190             }
191 
192             current = current->children[index];
193             if (current == nullptr) {
194                 return nullptr;
195             }
196         }
197         if (current->is_end_of_word) {
198             return current;
199         } else {
200             return nullptr;
201         }
202     }
```

沒有此單字

```
std::string& SearchVocabulary(const std::string& word) const {
    TrieNode* pos = SearchVocabularyPos(word);

    return pos->examples;
}
```

有此單字字首，但不是已記錄的單字

刪除單字(快速)

```
76      void Remove_t(const std::string& word) {
77          TrieNode* current = root;
78          for (int i = 0; i < word.size(); i++) {
79              char ch = word[i];
80              int index;
81              if (std::isupper(ch)) {
82                  index = ch - 'A';
83              } else if (std::islower(ch)) {
84                  index = ch - 'a' + 26;
85              }
86
87              if (current->children[index] == nullptr) {
88                  return;
89              }
90              current = current->children[index];
91          }
92          current->is_end_of_word = false;
93          current->examples = "";
94      }
```

如果指向下一個字母節點的指標為nullptr，代表此單字不存在

到達最後一個字母時，將節點取消標示為單字字尾

刪除單字(實際)

```cpp
105 TrieNode* Remove_m(TrieNode* root, std::string word, int depth = 0) {
106     if (root == nullptr) {
107         return nullptr;
108     }
109
110     if (depth == word.size()) {
111         if (root->is_end_of_word) {
112             root->is_end_of_word = false;
113         }
114         if (isEmpty(root)) {
115             delete (root);
116             root = nullptr;
117         }
118         return root;
119     }
120
121     char ch = word[depth];
122     int index = 0;
123     if (std::isupper(ch)) {
124         index = ch - 'A';
125     } else if (std::islower(ch)) {
126         index = ch - 'a' + 26;
127     }
128     root->children[index] = Remove_m(root->children[index], word, depth + 1);
129
130     if (isEmpty(root) && root->is_end_of_word == false) {
131         delete (root);
132         root = nullptr;
133     }
134     return root;
```

如果傳入之root為空，回傳nullptr

到達最後一個字母時，將節點取消標示為單字字尾，若此節點無子孫，即可刪除此節點

```cpp
bool isEmpty(TrieNode* root) {
    for (int i = 0; i < CHARACTERS_SIZE; i++)
        if (root->children[i] != nullptr) {
            return false;
        }
    return true;
}
```

遞迴函式前往下一個字母

若此節點無子孫且不是某單字之字尾，即可刪除此節點

參考資料：

確認單字是否存在

```
158     bool CheckWordExistence (const std::string& word) const {
159         TrieNode* current = root;
160         for (int i = 0; i < word.size(); i++) {
161             char ch = word[i];
162             int index;
163             if (std::isupper(ch)) {
164                 index = ch - 'A';
165             } else if (std::islower(ch)) {
166                 index = ch - 'a' + 26;
167             }
168 
169             current = current->children[index];
170             if (current == nullptr) {
171                 return false;
172             }
173         }
174         if (current->is_end_of_word) {
175             return true;
176         } else {
177             return false;
178         }
179     }
```

Trie

讀檔&
建立Trie

```cpp
void BuildTrieFromVocabularyFile(const std::string& filename, Trie& trie) {
    std::ifstream inputFile(filename);
    if (!inputFile) {
        std::cerr << "Failed to open vocabulary file: " << filename << std::er
        return;
    }

    std::string word = "";
    std::string example = "";
    while (std::getline(inputFile, word)) {
        // Skip empty or malformed lines
        if(word == "") {
            continue;
        }

        // Insert the vocabulary into the Trie
        trie.Add(word, example);
    }

    inputFile.close();
}
```

有例句時要再加一個
getline(inputfile, example)

查詢單字測試

```
430  void SearchTrieDemo(const std::string& filename, Trie& trie) {
431      std::ifstream inputFile(filename);
432      if (!inputFile) {
433          std::cerr << "Failed to open vocabulary file: " << file
434          return;
435      }
436
437      std::string word = "";
438      std::string example = "";
439      while (std::getline(inputFile, word)) {
440          // Skip empty or malformed lines
441          if(word == "") {
442              continue;
443          }
444          trie.SearchVocabularyPos(word);
445      }
446
447      inputFile.close();
448  }
```

刪除單字測試

```
void RemoveTrieDemo(const std::string& filename, Trie& trie) {
    std::ifstream inputFile(filename);
    if (!inputFile) {
        std::cerr << "Failed to open vocabulary file: " << filename
        return;
    }

    std::string word = "";
    std::string example = "";
    while (std::getline(inputFile, word)) {
        // Skip empty or malformed lines
        if(word == "") {
            continue;
        }
        trie.Remove_t(word);
    }

    inputFile.close();
}
```

自訂Set元素

```
458  class SetElement {
459  public:
460      std::string vocabulary = "";
461      std::string examples = "";
462                                  如此才能夠使用內建函式
463      // Overloading the equality operator
464      bool operator==(const SetElement& other) const {
465          return vocabulary == other.vocabulary;
466      }
467      bool operator>(const SetElement& other) const {
468          return vocabulary > other.vocabulary;
469      }
470      bool operator<(const SetElement& other) const {
471          return vocabulary < other.vocabulary;
472      }
473  };
```

讀檔&
建立Set

```
void BuildSetFromVocabularyFile(const std::string& filename, std::set<S
    std::ifstream inputFile(filename);
    if (!inputFile) {
        std::cerr << "Failed to open vocabulary file: " << filename <<
        return;
    }

    SetElement word;
    while (std::getline(inputFile, word.vocabulary)) {
        // Skip empty or malformed lines
        if(word.vocabulary == "") {
            continue;
        }
        // Insert the vocabulary into the Trie
        set.insert(word);
    }

    inputFile.close();
}
```

Set

查詢單字測試

```
void SearchSetDemo(const std::string& filename, std::set<SetEle
    std::ifstream inputFile(filename);
    if (!inputFile) {
        std::cerr << "Failed to open vocabulary file: " << file
        return;
    }

    SetElement word;
    while (std::getline(inputFile, word.vocabulary)) {
        // Skip empty or malformed lines
        if(word.vocabulary == "") {
            continue;
        }
        auto tem = set.find(word);
    }

    inputFile.close();
}
```

刪除單字測試

```
void RemoveSetDemo(const std::string& filename, std::set<SetElemer
    std::ifstream inputFile(filename);
    if (!inputFile) {
        std::cerr << "Failed to open vocabulary file: " << filenar
        return;
    }

    SetElement word;
    while (std::getline(inputFile, word.vocabulary)) {
        // Skip empty or malformed lines
        if(word.vocabulary == "") {
            continue;
        }
        set.erase(word);
    }

    inputFile.close();
}
```

互動式操作

```
823     std::string choice = "";
824     while (true) {
825         std::cout << "What do you want to do?(0:Exit 1:Add 2:Remove 3:Search)" <<
826         std::cin >> choice;
827 
828         if (choice == "0") {
829             break;
830         } else if (choice == "1") {
831             AddService(trie, set);
832         } else if (choice == "2") {
833             RemoveService(trie, set) ;
834         } else if (choice == "3") {
835             SearchService(trie, set);
836         }
837     }
838     return 0;
839 }
```

新增單字服務

```
void AddService(Trie& trie, std::set<SetElement>& set) {
    std::cout << "-----------------------------------" << std::endl;

    bool want_to_stop = false;
    std::string want_to_stop_input = "";

    while (!want_to_stop) {
        SetElement target;
        std::cout << "What vocabulary do you want to add?" << std::endl;
        std::cin >> target.vocabulary;

        if (trie.CheckWordExistence(target.vocabulary) == true) {
            std::cout << "Word already exists!!!(trie)" << std::endl;
            break;
        }

        if (set.find(target) != set.end()) {
            std::cout << "Word already exists!!!(set)" << std::endl;
            break;
        }

        while (true) {
            //Allow to add one example now.
            std::cout << "Do you want to add examples?(y/n)" << std::endl;
            std::string answer;
            std::cin >> answer;
            if (answer == "y") {
                std::cout << "Please type the example here..." << std::endl;
```

如果單字已存在，則結束服務

比較效能指標

```
751  int main() {
752      Trie trie;
753      std::set<SetElement> set;
754
755      // Read a vocabulary file and convert it to a trie
756      auto start1 = std::chrono::high_resolution_clock::now();
757
758      BuildTrieFromVocabularyFile("dictionary_inorder_all.txt", trie);
759
760      auto stop1 = std::chrono::high_resolution_clock::now();
761
762      // Read a vocabulary file and convert it to a set
763      auto start2 = std::chrono::high_resolution_clock::now();
764
765      BuildSetFromVocabularyFile("dictionary_inorder_all.txt", set);
766
767      auto stop2 = std::chrono::high_resolution_clock::now();
768
769      auto durationSearch1 = std::chrono::duration_cast<std::chrono::microseconds>(stop1 - start1);
770      std::cout << "BuildTrieFromVocabularyFile: " << durationSearch1.count() << "ms" << std::endl;
771
772      auto durationSearch2 = std::chrono::duration_cast<std::chrono::microseconds>(stop2 - start2);
773      std::cout << "BuildSetFromVocabularyFile: " << durationSearch2.count() << "ms" << std::endl;
```

測資說明

- 隨機生成(8個字元)[370105筆]
- 按照順序、全部單字(模擬之前作好的筆記)[370105筆]
- 打亂順序、全部單字(模擬之後加入新單字)[370105筆]
- 按照順序、短單字(8個字元以內)[149188筆]
- 按照順序、長單字(9個字元以上)[220917筆]

隨機生成單字

```
8   // Random word generator
9   std::string generateRandomWord(int length) {
10      static const char alphabet[] = "abcdefghijklmnopqrstuvwxyz";
11      static const int alphabetSize = sizeof(alphabet) - 1;
12      static std::random_device rd;
13      static std::mt19937 gen(rd());
14      std::string word;
15      std::uniform_int_distribution<> dis(0, alphabetSize - 1);
16
17      for (int i = 0; i < length; ++i) {
18          word += alphabet[dis(gen)];
19      }
20
21      return word;
22  }
```

過濾特定長度的單字

```
10  void FilterVocobularyByLength(const std::string& filename) {
11      std::ifstream inputFile(filename);
12      std::ofstream outputFile("dictionary_inorder_short.txt");  // Create a new output file
13
14      if (!inputFile) {
15          std::cerr << "Failed to open input file: " << filename << std::endl;
16          return;
17      }
18
19      std::string line;
20      while (std::getline(inputFile, line)) {
21          if (line.length() <= 8) {
22              outputFile << line << std::endl;  // Write the line to the output file
23          }
24      }
25
26      inputFile.close();
27      outputFile.close();
28  }
```

選擇想要的單字長度

打亂單字順序

```cpp
void randomizeVocabulary(const std::string& inputFile, const std::string& outputFile) {
    // Read the input file containing vocabulary in alphabetical order
    std::ifstream input(inputFile);
    if (!input) {
        std::cerr << "Failed to open input file: " << inputFile << std::endl;
        return;
    }

    std::vector<std::string> vocabulary;
    std::string word;
    while (std::getline(input, word)) {
        vocabulary.push_back(word);
    }
    input.close();

    // Randomize the order of the vocabulary words
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::shuffle(vocabulary.begin(), vocabulary.end(), std::default_random_engine(seed));

    // Create the output file
    std::ofstream output(outputFile);
    if (!output) {
        std::cerr << "Failed to create output file: " << outputFile << std::endl;
        return;
    }

    // Write the vocabulary words in random order
    for (const auto& word : vocabulary) {
        output << word << std::endl;
    }
    output.close();

    std::cout << "Vocabulary randomized and written to " << outputFile << std::endl;
}
```

1	osjxapwn
2	yzerinsu
3	gboiipga
4	eyzaqumi
5	biakvxhq
6	xowaohxs
7	asglifyw
8	mszxtisr
9	epmbqtrv
10	wdnahmic
11	ksoqstom
12	iuztpzke
13	kldvqojk
14	ducmdovt
15	qdvddtul
16	gfbxjimv
17	vyjulihm
18	fdilsofw
19	msijzyup
20	debjnton
21	lwljbtdf
22	ylefkkld
23	atopsvmt
24	lzrxaykw
25	fuyxxjzj
26	fbfvarlr
27	fruhqnxh
28	vucvbojc
29	qffekhlh
30	xglihjeo
31	pvqumfzi
32	bzkiojjf
33	hskwklqm
34	eotjcaoc
35	sftwvzeo

隨機生成

1	a
2	aa
3	aaa
4	aah
5	aahed
6	aahing
7	aahs
8	aal
9	aalii
10	aaliis
11	aals
12	aam
13	aani
14	aardvark
15	aardvarks
16	aardwolf
17	aardwolves
18	aargh
19	aaron
20	aaronic
21	aaronical
22	aaronite
23	aaronitic
24	aarrgh
25	aarrghh
26	aaru
27	aas
28	aasvogel
29	aasvogels
30	ab
31	aba
32	ababdeh
33	ababua
34	abac
35	abaca

按順序、全部單字

1	prelatism
2	ophidious
3	siricoidea
4	antiae
5	cruises
6	strictest
7	sulphoselenide
8	pharmacognostical
9	apparats
10	recovered
11	wastel
12	membranocalcareous
13	lachenalia
14	unnitrogenous
15	shoddydom
16	archipallial
17	wandsman
18	pumicated
19	cow
20	genotypicity
21	orseille
22	mycetozoa
23	mismeets
24	ouabains
25	drownded
26	lombardian
27	loveably
28	ideas
29	sitology
30	dermatodynia
31	protelidae
32	unbeautifully
33	superinquisitively
34	exorbitate
35	leguleian

打亂順序、全部單字

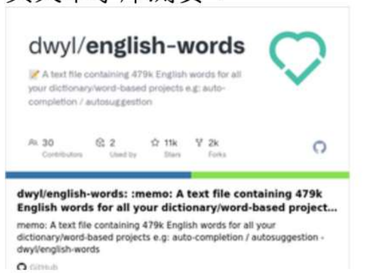
1	a
2	aa
3	aaa
4	aah
5	aahed
6	aahing
7	aahs
8	aal
9	aalii
10	aaliis
11	aals
12	aam
13	aani
14	aardvark
15	aardwolf
16	aargh
17	aaron
18	aaronic
19	aaronite
20	aarrgh
21	aarrghh
22	aaru
23	aas
24	aasvogel
25	ab
26	aba
27	ababdeh
28	ababua
29	abac
30	abaca
31	abacay
32	abacas
33	abacate
34	abacaxi
35	abaci

按照順序、短單字

1	aardvark
2	aardvarks
3	aardwolf
4	aardwolves
5	aaronical
6	aaronite
7	aaronitic
8	aasvogel
9	aasvogels
10	abacinate
11	abacination
12	abacisci
13	abaciscus
14	abacterial
15	abactinal
16	abactinally
17	abaction
18	abaculus
19	abacuses
20	abadengo
21	abaisance
22	abaissed
23	abalation
24	abalienate
25	abalienated
26	abalienating
27	abalienation
28	abalones
29	abampere
30	abamperes
31	abandonable
32	abandoned
33	abandonedly
34	abandonee
35	abandoner

按照順序、長單字

英文單字庫測資：



結果分析

功能 \ 測資	隨機生成	按照順序 全部	打亂順序 全部	按照順序 短單字	按照順序 長單字
新增	Set	Trie	Trie	Trie	Set
查詢	Trie	Trie	Trie	Trie	Trie
刪除(快速)	Trie	Trie	Trie	Trie	Trie
刪除(實際)	Set	Set	Set	Set	Set

檢討反思

- Trie的刪除(實際)節點時似乎有問題(搞不好這是比Set慢的原因?!
- Set在刪除測試時一行一行讀取會卡住，所以改成Trie和Set的刪除測試都隔行讀取
- 記憶體用量沒有測試到
- 提供更多功能 e.g.字首查詢、最長前綴匹配
- Trie好像有其他更好的實作方法
- 建立好的Trie和Set能夠儲存起來以便下次使用

感謝聆聽