|  |
| --- |
| **TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**  **KHOA ĐÀO TẠO QUỐC TẾ**    **BÀI TẬP LỚN MÔN HỌC**  **CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**  **Giảng viên hướng dẫn: TS.Hoàng Văn Thông**  **Sinh viên: Nguyễn Minh Quân**  **Lớp: CNTT VA 1 – Khóa 63**  **Hà Nội, tháng 12/2022** |

**BÀI 1 (BÀI SỐ 11 TRONG DANH SÁCH BÀI TẬP)**

Đề bài: Để kiểm tra lỗi chính tả trong các văn bản, người ta thực hiện kiểm tra từng từ của văn bản có trong từ điển của ngôn ngữ đó hay không? Việc kiểm tra lỗi chính tả được sử dụng trong nhiều ứng dụng như các phần mềm soạn thảo văn bản.

Hãy viết một chương trình kiểm tra lỗi chính tả văn bản tiếng anh theo các yêu cầu sau:

- Nạp một từ điển ngôn ngữ tiếng Anh vào một cấu trúc thích hợp từ file

- Viết chương trình nhập vào một đoạn văn, thực hiện kiểm tra xem trong đoạn văn đó có từ

nào sai lỗi chính tả hay không? Liệt kê các từ sai lỗi chính tả.

- Ghi từ điển vào file.

**Các lớp, các thuộc tính, các phương thức của lớp được sử dụng trong bài toán:**

**Lớp dictionary:**

**Thuộc tính:**

dict: Một tập hợp (set) các chuỗi để lưu trữ từ điển.

**Phương thức:**

* ChuanHoa(const string &word): Chuẩn hóa với kiểu trả về là chuỗi đã loại bỏ dấu câu ở cuối từ và chuyển đổi tất cả các ký tự thành chữ thường.

    string ChuanHoa(const string &word)

    {

        string res = word;

        while (!res.empty() && ispunct(res.back()))

        {

            res.pop\_back();

        }

        for (char &x : res)

        {

            x = tolower(x);

        }

        return res;

    }

* Vòng lặp while sẽ thực hiện khi biến res không phải là 1 xâu rỗng và phần tử cuối của xâu đó là 1 kí tự đặc biệt. Khi đó hàm ChuanHoa sẽ loại bỏ kí tự ở cuối (kí tự đặc biệt) và xét từng kí tự trong xâu và đặt nó thành kí tự in thường rồi trả về xâu đã được chuẩn hóa.
* readFile(const string &filename): Đọc dữ liệu từ file và thêm từng từ đã định dạng vào từ điển.

void readFile(const string &filename)

    {

        ifstream readFile(filename);

        if (!readFile.is\_open())

        {

            cout << "Mo file that bai!\n";

            return;

        }

        string word;

        while (getline(readFile, word))

        {

            word = transform(word);

            dict.insert(word);

        }

        readFile.close();

    }

* + Trong vòng lặp while, cứ đọc được 1 từ trong file thì sẽ dùng hàm transform để in thường và bỏ các ký tự đặc biệt để đồng bộ với đoạn văn nhập vào.
  + Sau đó, từ này sẽ được insert vào set.
* exportFile(const string filename): Xuất từ điển hiện tại ra file khác.

void exportFile(const string filename)

    {

        ofstream exportFile(filename);

        if (!exportFile.is\_open())

        {

            cout << "Mo file that bai!\n";

            return;

        }

        for (auto &it : dict)

        {

            exportFile << it << endl;

        }

        exportFile.close();

    }

* Trong hàm xuất file nếu không mở được file có tên được nhập vào để xuất ra thì sẽ thông báo mở file thất bại.
* Nếu truy cập thành công vào file cần xuất thì sẽ truy cập vào set và in ra các từ trong set.
* check(): Nhập một đoạn văn bản và kiểm tra chính tả của các từ trong đoạn văn bản so với từ điển.

void check()

    {

        cout << "Nhap doan van can kiem tra: \n";

        string text;

        getline(cin, text);

        stringstream ss(text);

        string word;

        int count = 0;

        while (ss >> word)

        {

            word = ChuanHoa(word);

            if (dict.find(word) == dict.end())

            {

                cout << "- " << word << " sai chinh ta !\n";

                count++;

            }

        }

        if (count == 0)

        {

            cout << "Van ban da nhap khong co tu nao sai chinh ta !\n";

        }

    }

* Trong hàm check, đầu tiên ta sẽ nhập vào 1 đoạn văn sau đó sử dụng stringstream để cắt các từ trong đoạn văn ra rồi chuẩn hóa nó, nếu có 1 từ trong đoạn văn không được tìm thấy trong set, có nghĩa là từ đó không có trong từ điển => sai chính tả thì ta sẽ tăng biến đếm count thêm 1. Nếu đã kiểm tra hết mà không thấy từ nào sai chính tả thì count vẫn = 0 => Thông báo rằng văn bản không có từ sai chính tả.

**Tính toán độ phức tạp của thuật toán theo big O:**

1. **Phương thức ChuanHoa(const string &word):**
   * Vì các thao tác trong phương thức này đều là các thao tác ngắn và cố định về độ phức tạp, thời gian chạy cho mỗi từ sẽ là O(n), trong đó n là độ dài của từ.
2. **Phương thức readFile(const string &filename):**
   * Độ phức tạp của phương thức này phụ thuộc vào số lượng từ trong file và độ dài của mỗi từ. Nếu n là số từ trong tệp và m là độ dài trung bình của từ, thì độ phức tạp thời gian chạy sẽ là O(n\*m).
3. **Phương thức exportFile(const string filename):**
   * Độ phức tạp của phương thức này phụ thuộc vào số lượng từ trong file và độ dài của mỗi từ. Nếu n là số từ trong tệp và m là độ dài trung bình của từ, thì độ phức tạp thời gian chạy sẽ là O(n\*m).
4. **Phương thức check():**
   * Độ phức tạp của phương thức này phụ thuộc vào số lượng từ cần kiểm tra và độ dài trung bình của mỗi từ. Độ phức tạp thời gian chạy là O(n\*m) với n là số lượng từ cần kiểm tra và m là độ dài trung bình của mỗi từ.

**BÀI 2 (BÀI SỐ 31 TRONG DANH SÁCH BÀI TẬP)**

Đề bài:

1. Xây dựng lớp cây nhị phân.

2. Ứng dụng cây nhị phân giải các bài toán sau:

a. Cho một cây nhị phân. Hãy liệt kê tất cả các đường đi từ gốc đến lá.

b. Cho một cây nhị phân chỉ gồm các số từ 0 đến 9, mỗi đường đi từ gốc đến lá có thể biểu diễn một

số. Ví dụ: đường đi từ gốc đến lá là 1->2->3 biểu diễn số 123. Tìm tổng cộng tất cả các số từ gốc đến

lá.

Ví dụ:

Đầu vào: [1,2,3]

1

/ \

2 3

Đầu ra: 12 + 13 = 25

c. Cho một cây nhị phân và một số a, xác định xem có đường đi từ gốc đến lá nào mà tổng giá trị của

nó bằng a.

Đầu vào: Với cây nhị phân bên dưới và a = 22

5

/ \

4 8

/ / \

11 13 4

/ \ \

2 7 1

Đầu ra: True, tồn tại đường đi 5->4->11->2 mà tổng là 22

**Các lớp được sử dụng trong bài toán:**

1. **Lớp node:**
   * **Thuộc tính:**
   * parent, left, right: các con trỏ tới các nút cha, nút con bên trái và nút con bên phải.
   * value: Giá trị của nút.
   * **Phương thức:**
   * Constructor: Hàm dựng không tham số khởi tạo giá trị mặc định (value = 0) hoặc hàm dựng có tham số khởi tạo giá trị (value) được được truyền vào.
   * Các hàm getter và setter cho giá trị, nút cha, nút con trái và nút con phải.
   * Các phương thức hasLeft, hasRight để kiểm tra sự tồn tại của nút con trái, nút con phải.
2. **Lớp BinaryTree:**
   * **Thuộc tính:**
   * root: Con trỏ tới nút gốc của cây.
   * sz: Kích thước của cây.
   * **Phương thức:**
   * Constructor: Hàm tạo để khởi tạo một cây trống với con trỏ root trỏ vào nullptr và sz = 0.
   * Getter getRoot() cho nút gốc và hàm isEmpty() kiểm tra xem cây có rỗng hay không.
   * Các phương thức như isRoot(), isInternal(), isExternal() để kiểm tra loại nút (gốc, nút trong, lá).
   * Các phương thức duyệt cây nhị phân (duyệt tiền tự, duyệt trung tự, duyệt hậu tự).
   * Phương thức insert: Thêm một nút mới vào cây theo quy tắc của cây nhị phân.
3. node \*newNode = new node(value);
4. newNode->setParent(parent);
   * + Đầu tiên sẽ cấp phát động 1 nút mới với giá trị = value.
     + Đặt con trỏ cha cua nút mới là con trỏ của nút cha đã được truyền vào trước đó
5. if (root == nullptr)
6. {
7. root = newNode;
8. }
   * + Nếu con trỏ root mà là con trỏ nullptr thì cây chưa có nút => đặt nút mới là gốc của cây.
9. else if (!parent->hasLeft())
10. {
11. parent->setLeft(newNode);
12. }
13. else if (!parent->hasRight())
14. {
15. parent->setRight(newNode);
16. }
    * + Trong trường hợp nút cha không có con trái/phải thì sẽ đặt nút mới là con trái/phải.
17. else
18. {
19. return nullptr;
20. }
    * + Nếu cây đã có nút và nút cha đã có con trái và con phải thì không thể thêm nút vào đó => trả về con trỏ nullptr.
21. sz++;
22. return newNode;
    * + Sau khi thêm 1 nút thì tăng biến sz thêm 1 và trả về nút mới.
    * Phương thức PathsList (Liệt kê tất cả các đường đi từ gốc đến lá):
23. if (node == nullptr)
24. {
25. return;
26. }
    * + Trong vòng lặp if này nếu node là 1 con trỏ nullptr hay cây rỗng thì hàm PathsList sẽ dừng ngay lập tức do không có đường đi nào để liệt kê.
27. paths.push\_back(node->getValue());
    * + Nếu cây có nút hay phần tử rồi thì giá trị các nút sẽ được đưa vào trong vector paths.
28. if (!node->hasLeft() && !node->hasRight())
29. {
30. cout << "Path: ";
31. for (int value : paths)
32. {
33. cout << value << " -> ";
34. }
35. cout << "NULL\n";
36. }
37. // neu chua duyet het duong
38. else
39. {
40. PathsList(node->getLeft(), paths);
41. PathsList(node->getRight(), paths);
42. }
    * + Trong vòng lặp if này, hàm PathsList sẽ kiểm tra điều kiện nếu nút hiện tại không có nút con trái và nút con phải tức là nút hiện tại là nút lá thì sẽ truy cập vào các phần tử trong vector qua vòng for và in ra quãng đường từ gốc đến lá.
      + Trong trường hợp chưa duyệt được đến nút lá, tức là nút hiện tại vẫn có con trái hoặc con phải thì sẽ thực hiện else. Hàm sẽ đệ quy với tham số là con trái của nút hiện tại và vector qua PathsList(node->getLeft(), paths); cũng như con phải và vector qua PathsList(node->getRight(), paths); lần lượt sẽ được thêm vào stack cho đến nút lá và bắt đầu thực hiện.
    * Phương thức SumOfPath (Tính tổng các số trên mỗi đường từ gốc đến lá):
43. void SumOfPath(node \*node, vector<int> path, int &sum)
    * Trong hàm SumOfPath ta sẽ truyền vào tham chiếu &sum để có thể in ra giá trị sum sau cuối mà không bị lặp lại do đệ quy.
44. if (node == nullptr)
45. {
46. return;
47. }
    * + Trong vòng lặp if này nếu node là 1 con trỏ nullptr hay cây rỗng thì hàm SumOfPath sẽ dừng ngay lập tức do không có đường đi nào để liệt kê.
48. path.push\_back(node->getValue());
    * + Nếu cây có nút hay phần tử rồi thì giá trị các nút sẽ được đưa vào trong vector path.
49. if (!node->getLeft() && !node->getRight())
50. {
51. int sumEach = 0;
52. for (int value : path)
53. {
54. sumEach = sumEach \* 10 + value;
55. }
56. sum += sumEach;
57. }
58. else
59. {
60. SumOfPath(node->getLeft(), path, sum);
61. SumOfPath(node->getRight(), path, sum);
62. }
    * + Trong vòng lặp if này, hàm SumOfPath sẽ kiểm tra điều kiện nếu nút hiện tại không có nút con trái và nút con phải tức là nút hiện tại là nút lá thì sẽ truy cập vào các phần tử trong vector qua vòng for và cho tổng đường đi = tổng hiện tại \* 10 + giá trị mỗi số duyệt qua trên đường đi đến lá. Sau khi kết thúc vòng lặp for có nghĩa là ta đã duyệt qua tất cả phần tử từ gốc đến lá và thu được giá trị sumEach bằng tổng các số trên 1 đường đi và sum là tổng các số trên tất cả đường đi
      + Trong trường hợp chưa duyệt được đến nút lá, tức là nút hiện tại vẫn có con trái hoặc con phải thì sẽ thực hiện else. Hàm sẽ đệ quy với tham số là con trái của nút hiện tại và vector qua SumOfPath(node->getLeft(), path, sum); cũng như con phải và vector qua SumOfPath(node->getRight(), path, sum); lần lượt sẽ được thêm vào stack cho đến nút lá và bắt đầu thực hiện.
    * Phương thức FindPathEqualToA (Tìm đường đi từ gốc đến lá có tổng các số bằng giá trị a được nhập vào từ bàn phím):
63. void FindPathEqualToA(node \*node, vector<int> path, int sum, int a, int &cnt)
    * + Trong hàm FindPathEqualToA ta sẽ truyền vào tham chiếu &cnt để sau cuối quyết định xem có in ra câu lệnh false hay không.
64. sum = 0;
65. // neu cay rong thi return
66. if (node == nullptr)
67. {
68. return;
69. }
    * + Đầu tiên, ta sẽ đặt lại giá trị của biến sum = 0 do hàm bên trên sử dụng tham chiếu thay đổi giá trị biến sum. Sau đó kiểm tra vòng lặp if này nếu node là 1 con trỏ nullptr hay cây rỗng thì hàm FindPathEqualToA sẽ dừng ngay lập tức do không có đường đi nào để liệt kê.
70. path.push\_back(node->getValue());
    * + Tiếp tục thêm giá trị các nút vào trong vector.
71. if (!node->hasLeft() && !node->hasRight())
72. {
73. for (int value : path)
74. {
75. sum += value;
76. }
77. if (sum == a)
78. {
79. cout << "True, ton tai duong di ";
80. for (int value : path)
81. {
82. cout << value << " -> ";
83. }
84. cout << "NULL co tong duong di = " << sum << endl;
85. cnt++;
86. }
87. }
88. else
89. {
90. FindPathEqualToA(node->getLeft(), path, sum, a, cnt);
91. FindPathEqualToA(node->getRight(), path, sum, a, cnt);
92. }
    * + Trong vòng lặp if này, hàm FindPathEqualToA sẽ kiểm tra điều kiện nếu nút hiện tại không có nút con trái và nút con phải tức là nút hiện tại là nút lá thì sẽ truy cập vào các phần tử trong vector qua vòng for và cho tổng đường đi = tổng hiện tại + giá trị mỗi số duyệt qua trên đường đi đến lá rồi kiểm tra vòng lặp if, nếu tổng đó = với giá trị a được nhập thì sẽ in ra câu lệnh: "True, ton tai duong di " và xét vòng for tiếp theo để in ra đường đi đó. Nếu tồn tại đường đi có tổng = a thì sẽ tăng biến cnt thêm 1 với mỗi đường đi tồn tại. Sau cuối, nếu biến cnt vẫn = 0 thì có nghĩa là không có đường đi nào tổng các số = a và sẽ in ra câu lệnh "False, khong ton tai duong di nao co tong = " << a << " !" .

**Phân tích thời gian chạy của từng phương thức có trong các lớp theo big O:**

**Lớp node:**

* **Phương thức:**
  + getValue(), getLeft(), getRight(), getParent(): Các phương thức này chỉ thực hiện các phép gán trả về giá trị, vì vậy độ phức của chúng là O(1) - hằng số.

**Lớp BinaryTree:**

* **Phương thức:**
  + insert(node \*parent, int value): Trong trường hợp tốt nhất khi cây cân bằng, thời gian chèn một nút mới vào cây có thể là O(log n), với n là số lượng nút hiện có trong cây. Trong trường hợp xấu nhất, nếu cây trở thành một danh sách liên kết, thời gian chèn có thể là O(n).
  + preOrder(node\*targetNode), inOrder(node\*currentNode), posOrder(node\*currentNode): Đối với một cây có n nút, thời gian duyệt các nút theo thứ tự tiền tự, trung tự và hậu tự là O(n) vì mọi nút sẽ được duyệt đúng một lần.
  + PathsList(node \*node, vector<int> paths), SumOfPath(node \*node, vector<int> path, int &sum), FindPathEqualToA(node \*node, vector<int> path, int sum, int a, int &cnt): Các phương thức này chạy qua tất cả các nút trong cây, do đó độ phức tạp thời gian của chúng là O(n).
* Đối với đối tượng tree trong bài toán được insert 10 nút thì thời gian chạy của các phương thức trong trường hợp tồi nhất có thể tính bằng O(log 10) cho các phương thức insert, và O(10) cho các phương thức còn lại. Tuy nhiên, trong trường hợp tồi nhất, khi cây là một cây nhị phân không cân bằng, thì độ phức tạp thời gian sẽ là O(10) cho các phương thức.