## CSCI 230 PA 8 Submission

## Due Date: 04/26/2022 Late (date and time):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Name(s): Nero Li

Exercise 1 (with extra credit) -- need to submit source code and I/O  
 -- check if completely done ✔️ ; otherwise, discuss issues below

Source code below:

**exercise\_1.cpp:**

/\* Program: PA\_8\_exercise\_1

Author: Nero Li

Class: CSCI 230

Date: 04/26/2022

Description:

Implement the MCP algorithm and print out resulting table as

well as the minimum number of operations. Try B x C x D with B

a 2x10 matrix, C a 10x50 matrix, and D a 50x20 matrix. Try

another test case with 10x5 (A), 5x2 (B), 2x20 (C), 20x12 (D),

12x4 (E), and 4x60 (F).

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include <iostream>

#include <vector>

using namespace std;

void MatrixChain(vector<int> d)

{

int n = d.size() - 1;

string order;

char c = 'A';

vector<vector<int>> N(n, vector<int>(n, 0));

vector<pair<int, int>> par(n, pair<int, int>(0, 0));

for (int b = 1; b < n; ++b)

for (int i = 0; i <= n - b - 1; ++i)

{

int j = i + b;

N[i][j] = INT\_MAX;

for (int k = i; k < j; ++k)

{

int a = N[i][j];

int b = N[i][k] + N[k + 1][j] + d[i] \* d[k + 1] \* d[j + 1];

N[i][j] = a < b ? a : b;

}

}

cout << "Matrix result:\n";

for (int i = 0; i < n; ++i)

{

for (int j = 0; j < n; ++j)

cout << N[i][j] << "\t";

cout << endl;

}

cout << "Order of evaluation:\n";

int ax = 0;

int bx = 0;

int ay = n - 1;

int by = n - 1;

++par[bx++].first;

++par[ay--].second;

for (int i = 0; i < n - 2; ++i)

{

if (N[ax][ay] < N[bx][by])

{

++par[ax].first;

++par[ay].second;

--ay;

--by;

}

else

{

++par[bx].first;

++par[by].second;

++ax;

++bx;

}

}

char ch = 'A';

int count = 0;

for (auto i : par)

{

while (i.first)

{

cout << "(";

--i.first;

}

cout << ch;

++ch;

while (i.second)

{

cout << ")";

--i.second;

}

if (count < n - 1)

{

cout << " \* ";

++count;

}

}

cout << endl;

}

int main()

{

// 2x10 (A), 10x50 (B), 50x20 (C)

vector<int> test1 = {2, 10, 50, 20};

// 10x5 (A), 5x2 (B), 2x20 (C), 20x12 (D), 12x4 (E), and 4x60 (F)

vector<int> test2 = {10, 5, 2, 20, 12, 4, 60};

MatrixChain(test1);

MatrixChain(test2);

cout << "Author: Nero Li\n";

return 0;

}

Input/output below:

Matrix result:

0 1000 3000

0 0 10000

0 0 0

Order of evaluation:

((A \* B) \* C)

Matrix result:

0 100 500 820 756 2356

0 0 200 600 616 1656

0 0 0 480 576 1056

0 0 0 0 960 5760

0 0 0 0 0 2880

0 0 0 0 0 0

Order of evaluation:

((A \* (B \* ((C \* D) \* E))) \* F)

Author: Nero Li

Exercise 2 -- need to submit source code and I/O  
 -- check if completely done ✔️ ; otherwise, discuss issues below

Source code below:

**exercise\_2.cpp:**

/\* Program: PA\_8\_exercise\_2

Author: Nero Li

Class: CSCI 230

Date: 04/26/2022

Description:

Implement a standard trie for a set of ASCII strings including

a terminating character $ for each word. You might want to look

at the trie in zyBook. Create a class that has a constructor

that accepts the name of an input file as a parameter (a string),

and the class should have an operation that test whether a given

string is stored in the trie. The driver should allow user to

specify the input data file, output number of words in the trie,

and then use a y/n loop to check for a few words (try the

following words: honor, honour, government, computer). Output

yes or no for each search word. Use the text file usdeclarPC.txt

as an input file and you should format the words to lowercase

and remove extra characters like comma, periods, etc.

I certify that the code below is my own work.

Exception(s): N/A

\*/

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <unordered\_set>

using namespace std;

class Trie

{

private:

class Node

{

public:

char elem;

vector<Node \*> child;

Node \*getChild(char c)

{

if (child.empty())

return NULL;

for (auto i : child)

if (i->elem == c)

return i;

return NULL;

}

} \*head;

int count;

public:

Trie(string file)

{

count = 0;

head = new Node;

head->elem = '\0';

ifstream fin;

fin.open(file, ios::binary);

if (!fin)

return;

while (!fin.eof())

{

string cur;

fin >> cur;

string::iterator itr = cur.begin();

while (itr != cur.end() && !cur.empty())

{

if (\*itr >= 'A' && \*itr <= 'Z')

\*itr = \*itr - 'A' + 'a';

if (!(\*itr >= 'a' && \*itr <= 'z'))

{

cur.erase(itr);

itr--;

}

itr++;

}

if (!cur.empty())

{

cur.push\_back('$');

bool newWord = false;

Node \*cursor = head;

for (char c : cur)

{

if (!cursor->getChild(c))

{

Node \*temp = new Node;

temp->elem = c;

cursor->child.push\_back(temp);

cursor = temp;

newWord = true;

}

else

cursor = cursor->getChild(c);

}

cur.pop\_back();

if (newWord)

++count;

}

}

}

int getNumOfWords()

{

return count;

}

bool checkWord(string word)

{

word.push\_back('$');

Node \*cursor = head;

for (char c : word)

{

if (!cursor->getChild(c))

return false;

else

cursor = cursor->getChild(c);

}

return true;

}

};

int main()

{

Trie test("usdeclarPC.txt");

vector<string> words = {"honor", "honour", "government", "computer"};

cout << "Number of words:\t" << test.getNumOfWords() << endl;

for (string w : words)

cout << "Check word " << w << ":\t" << (test.checkWord(w) ? "yes" : "no") << endl;

cout << "Author: Nero Li\n";

return 0;

}

Input/output below:

Number of words: 538

Check word honor: yes

Check word honour: no

Check word government: yes

Check word computer: no

Author: Nero Li

Answer for Question 1:

The main difference between standard tries and compressed tries is that compressed tries combine all the nodes that has only one child together. Due to this operation, we will have less node than standard tries, hence, we save the space that was created by our tries.

Answer for Question 2:

X = GTCCTA

Y = CGATA

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| L | -1 | 0 G | 1 T | 2 C | 3 C | 4 T | 5 A |
| -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 C | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 G | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 A | 0 | 1 | 1 | 1 | 1 | 1 | 2 |
| 3 T | 0 | 1 | 2 | 2 | 2 | 2 | 2 |
| 4 A | 0 | 1 | 2 | 2 | 2 | 2 | 3 |

Longest common sub-sequence: GTA