CSCI 230 –Lab Final

Nero Li

Make sure to read all instructions before attempting this assignment. You cannot work with another student or communicate with anyone for this assignment. If you submitted an online solution, a solution on site such as Discord, or someone else solution, 0 will be given. If you posted/shared a solution and someone else uses it, you might get a 0 as well. You will only need to submit 2 out of 3 problems.

Pick 2 out of 3 problems and revise them as specified. Submit only 2 problems or the last problem will not be graded. Submit one PDF file with code and any input/output. It is your responsibility to confirm that you submitted your file correctly, so it is best to view your submission to make sure it was submitted correctly before you leave.

1. Given a large data file of positive integers up to 1 million values, your task is to quickly determine if an input value x is in the file and there is also a value y in the file that satisfies x + y = t. For the sake of simplicity, we will assume that the values in the file are unique. You need to do this look up very efficiently in O(1) so a hash map is needed. Design a solution that uses a hash map to solve this problem. *Hint: input values and store them in a hash map first*. Here are two test cases to help you with this problem:

```
Input file: large100k.txt from one PA (unique values from 1 to
100000)

Input x: 5<Enter>
Input t: 100<Enter>
Yes

Input x: 5<Enter>
Input t: 10000000<Enter>
No
```

New requirements:

- Store the key as string instead of int
- Given a valid input, determine if the reverse string is in the hash table.

Sample input and output:

```
Input x: 5<Enter>
Yes, both 5 and 5 are in the file

Input x: 1234<Enter>
Yes, both 1234 and 4321 are in the file

Input x: 1000<Enter>
No, 0001 is not in the file
```

Copy/paste source code and input/output below:

```
/*  Program: Lab_Final_1
    Author: Nero Li
    Class: CSCI 230
    Date: 06/09/2022
    Description:
        Given a large data file of positive integers up to 1 million values,
your task is to quickly
        determine if an input value x is in the file and there is also a
value y in the file that
        satisfies x + y = t. For the sake of simplicity, we will assume that
the values in the file
        are unique. You need to do this look up very efficiently in O(1) so a
hash map is needed.
        Design a solution that uses a hash map to solve this problem. Hint:
input values and
```

```
        store them in a hash map first. Here are two test cases to help you
with this problem:
            Input file: large100k.txt from project 2 (unique values from 1 to
100000)

            Input x: 5<Enter>
            Input t: 100<Enter>
            Yes

            Input x: 5<Enter>
            Input t: 10000000<Enter>
            No

        New requirements:

        -   Store the key as string instead of int
        -   Given a valid input, determine if the reverse string is in the
hash table.


    I certify that the code below is my own work.

    Exception(s): N/A

*/

#include <iostream>
#include <fstream>
#include <algorithm>
#include <unordered_set>

using namespace std;

void func(unordered_set<string> s)
{
    string x;

    cout << "Input x: ";
    cin >> x;

    string y = x;
    reverse(y.begin(), y.end());

    if (s.find(y) == s.end())
        cout << "No, " << y << " is not in the file" << endl;
    else if (s.find(x) == s.end())
        cout << "No, " << x << " is not in the file" << endl;
    else
```

```
        cout << "Yes, both " << x << " and " << y << " are in the file" <<
endl;

    cout << endl;
}

int main()
{
    unordered_set<string> s;
    string file = "large100k.txt";

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

    if (!fin)
    {
        cout << "err\n";
        exit(-1);
    }

    while (!fin.eof())
    {
        string n;
        fin >> n;
        s.insert(n);
    }

    func(s);
    func(s);
    func(s);

    fin.close();

    cout << "Author: Nero Li\n";
    return 0;
}
```

```
Input x: 5
Yes, both 5 and 5 are in the file

Input x: 1234
Yes, both 1234 and 4321 are in the file

Input x: 1000
No, 0001 is not in the file

Author: Nero Li
```

2. Given three sorted arrays/vectors of integers with lengths k, m, and n, provide the code to perform a three-way merge to merge them into one sorted array/vector in O(k + m + n). Here is one test case to try:

    - Array/Vector 1 – 5, 9, 12
    - Array/Vector 2 – 1, 4, 10, 16, 25
    - Array/Vector 3 – 2, 5, 8, 21

    Result:

    - Array/Vector 4 – 1, 2, 4, 5, 5, 8, 9, 10, 12, 16, 21, 25

    New requirements:

    - Given three unsorted arrays/vectors of names (strings)
    - Still need to merge three sorted arrays/vectors into one sorted array/vector in O(k + m + n); do not combine the 3 unsorted list and sort it

    Here is one test case to try:

    - Array/Vector 1 – “Adam”, “Kim”, “William”, “Bill”
    - Array 2/Vector – “Bob”, “Joann”, “Jane”, “John”, “Tim”
    - Array 3/Vector – “Michelle”, “Candace”, “Daniel”, “Eric”, “Tanya”

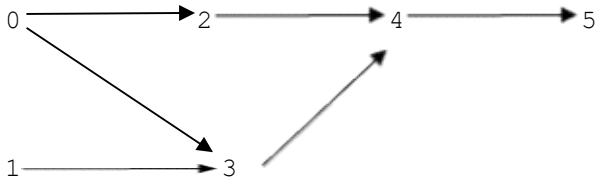
    Result:

    - Array/Vector 4 – “Adam”, “Bill”, “Bob”, “Candace”, “Daniel”, “Eric”, “Jane”, “Joann”, “John”, “Kim”, “Michelle”, “Tanya”, “Tim”, “William”

    Copy/paste source code and input/output below:

    (Skipped)

Given the following DAG, provide a simple matrix to represent it (each entry of the matrix would hold a value 0 or 1).



Provide code to print the above DAG using the following format (there are two edges from 0 to 2 and 0 to 3, …, and no edge from 5).

```
Vertex   Edges
0        2 3
…
5
```

Print one possible topological ordering for the above DAG such as:

```
0 2 1 3 4 5
```

New requirements:

- Compute the transitive closure and output the new graph using the same format.

Format for sample output:

```
Original graph:

Vertex   Edges
0        2 3
…
5


Topological ordering:
0 2 1 3 4 5

Transitive closure:

Vertex   Edges
0        2 3 4 5
…
5
```

Copy/paste source code and input/output below:

```
/*  Program: Lab_Final_3
    Author: Nero Li
    Class: CSCI 230
    Date: 06/09/2022
    Description:
        Given the following DAG, provide a simple matrix to represent it
(each entry of the
        matrix would hold a value 0 or 1).
```

```
        Provide code to print the above DAG using the following format (there
are two edges
        from 0 to 2 and 0 to 3, …, and no edge from 5).

        Vertex      Edges
        0           2 3
        …
        5

        Print one possible topological ordering for the above DAG such as:
            0 2 1 3 4 5

        New requirements:

        -   Compute the transitive closure and output the new graph using the
same format.


    I certify that the code below is my own work.

    Exception(s): N/A

*/

#include <iostream>
#include <vector>
#include <queue>

using namespace std;

class MatrixGraph
{
private:
    vector<vector<bool>> Matrix;
public:
    MatrixGraph(int n)
    {
        for (int i = 0; i < n; ++i)
        {
            vector<bool> cur;
            for (int j = 0; j < n; ++j)
                cur.push_back(false);
            Matrix.push_back(cur);
        }
    }

    void insert(int i, int j)
    {
```

```
        Matrix[i][j] = true;
    }

    void print()
    {
        cout << "Original graph:\nVertex\tEdges\n";

        for (int i = 0; i < Matrix.size(); ++i)
        {
            cout << i << "\t";
            for (int j = 0; j < Matrix.size(); ++j)
                if (Matrix[i][j])
                    cout << j << " ";
            cout << endl;
        }
    }

    void topologicalOrdering()
    {
        cout << "Topological ordering:\n";
        vector<bool> explored;

        for (int i = 0; i < Matrix.size(); ++i)
            explored.push_back(false);

        for (int i = 0; i < Matrix.size(); ++i)
        {
            if (!explored[i])
            {
                cout << i << " ";
                explored[i] = true;
            }

            for (int j = 0; j < Matrix.size(); ++j)
                if (Matrix[i][j])
                {
                    bool canExplore = true;
                    for (int k = 0; k < Matrix.size(); ++k)
                        if (Matrix[k][j])
                            if (!explored[k])
                                canExplore = false;

                    if (canExplore && !explored[j])
                    {
                        cout << j << " ";
                        explored[j] = true;
                    }
```

```
                }
        }

        cout << endl;
    }

    void transitiveClosure()
    {
        cout << "Transitive closure:\nVertex\tEdges\n";

        for (int i = 0; i < Matrix.size(); ++i)
        {
            queue<int> edges;
            vector<bool> explored;
            for (int j = 0; j < Matrix.size(); ++j)
                explored.push_back(false);

            cout << i << "\t";
            for (int j = 0; j < Matrix.size(); ++j)
                if (Matrix[i][j])
                    edges.push(j);

            while (!edges.empty())
            {
                int k = edges.front();
                if (!explored[k])
                {
                    cout << k << " ";
                    explored[k] = true;
                }
                edges.pop();
                for (int j = 0; j < Matrix.size(); ++j)
                    if (Matrix[k][j])
                        edges.push(j);
            }

            cout << endl;
        }
    }

};

int main()
{
    MatrixGraph G(6);
    G.insert(0, 2);
    G.insert(0, 3);
```

```
    G.insert(1, 3);
    G.insert(2, 4);
    G.insert(3, 4);
    G.insert(4, 5);
    G.print();
    cout << endl;
    G.topologicalOrdering();
    cout << endl;
    G.transitiveClosure();
    cout << endl;

    cout << "Author: Nero Li\n";
    return 0;
}
```

```
Original graph:
Vertex  Edges
0       2 3
1       3
2       4
3       4
4       5
5

Topological ordering:
0 2 1 3 4 5

Transitive closure:
Vertex  Edges
0       2 3 4 5
1       3 4 5
2       4 5
3       4 5
4       5
5

Author: Nero Li
```