CSCI 230 PA 12 Submission

Due Date: ##/##/2022 Late (date and time):_____

Name(s): Nero Li

Exercise 1 -- need to submit source code and I/O

-- check if completely done ✔; otherwise, discuss issues below

Source code below:
exercise_1.cpp:

```
/*  Program: PA_12_exercise_1
    Author: Nero Li
    Class: CSCI 230
    Date: 05/31/2022
    Description:
        Implement one MST algorithm -- either Prim-Jarnik Algorithm or
        Kruskal Algorithm. Try a small graph below and print out the MST
        and total cost.

    I certify that the code below is my own work.

      Exception(s): N/A

*/

#include <iostream>
#include <queue>
#include "AdjacencyListGraph.h"
#include "HeapPriorityQueue.h"

using namespace std;
typedef pair<int, Vertex *> PQP;

class isLess
{
public:
    bool operator()(PQP a, PQP b)
    {
        return a.first < b.first;
    }
};

void printMST(AdjacencyListGraph G, map<Vertex *, Edge *> parent, Vertex
*src, int level)
{
    cout << src->getElement() << endl;
```

```
    for (auto i : parent)
    {
        if (i.second != NULL)
        {
            if (G.opposite(i.first, i.second) == src)
            {
                for (int p = 0; p < level; ++p)
                {
                    if (p == level - 1)
                        cout << "-";
                    else
                        cout << " ";
                }

                printMST(G, parent, i.first, level + 1);
            }
        }
    }


}

void PrimJarnikMST(AdjacencyListGraph G, Vertex *s)
{
    HeapPriorityQueue<PQP, isLess> Q;
    map<Vertex *, int> distance;
    map<Vertex *, Edge *> parent;
    map<Vertex *, PQP> locator;
    map<Vertex *, bool> visited;
    for (auto v : G.getVertices())
    {
        if (v == s)
            distance[v] = 0;
        else
            distance[v] = INT_MAX;

        parent[v] = NULL;
        PQP l;
        l.first = distance[v];
        l.second = v;
        Q.insert(l);
        locator[v] = l;
        visited[v] = false;
    }

    while (!Q.empty())
    {
        PQP l = Q.min();
        Q.removeMin();
        Vertex *u = l.second;
        // cout << u->getElement() << endl;
```

```
        visited[u] = true;
        for (auto e : G.incomingEdges(u))
        {
            Vertex *z = G.opposite(u, e);
            int r = e->getElement();
            if (r < distance[z] && !visited[z])
            {
                distance[z] = r;
                parent[z] = e;
                Q.replace(locator[z], r);
                // cout << z->getElement() << ", " << r << ": " <<
Q.min().second->getElement() << endl;
            }
        }
    }

    printMST(G, parent, s, 1);

    int cost{0};
    for (auto i : parent)
        if (i.second != NULL)
            cost += i.second->getElement();
    cout << "Total cost: " << cost << endl;
}

int main()
{
    AdjacencyListGraph G;

    Vertex *A = G.insertVertex("A");
    Vertex *B = G.insertVertex("B");
    Vertex *C = G.insertVertex("C");
    Vertex *D = G.insertVertex("D");
    Vertex *E = G.insertVertex("E");
    G.insertEdge(A, B, 3);
    G.insertEdge(A, D, 5);
    G.insertEdge(A, E, 5);
    G.insertEdge(B, C, 4);
    G.insertEdge(C, D, 2);
    G.insertEdge(D, E, 5);
    G.insertEdge(C, E, 3);

    cout << "Original Graph:\n";
    G.print();
    cout << endl;

    cout << "MST:\n";
    PrimJarnikMST(G, A);
    cout << endl;

    cout << "Modified by: Nero Li\n";
```

```
    return 0;
}
```

Input/output below:

```
Original Graph:
Vertex A
 3 adjacencies:(B, 3)  (D, 5)  (E, 5)
Vertex B
 2 adjacencies:(A, 3)  (C, 4)
Vertex C
 3 adjacencies:(B, 4)  (D, 2)  (E, 3)
Vertex D
 3 adjacencies:(A, 5)  (C, 2)  (E, 5)
Vertex E
 3 adjacencies:(A, 5)  (D, 5)  (C, 3)

MST:
A
-B
 -C
  -D
  -E
Total cost: 12

Modified by: Nero Li
```

Exercise 2 (Option C) -- need to submit source code and I/O

-- check if completely done ✔; otherwise, discuss issues below

Note: This code might not be completely correct since I might misunderstand the requirement or condition that provided us even if the output result looks correct.

Source code below:
exercise_2.cpp:

```
/*  Program: PA_12_exercise_2
    Author: Nero Li
    Class: CSCI 230
    Date: 05/31/2022
    Description:
        Option C: Implement the simple external sorting using algorithm
        from the "external sorting" section of the Shaffer book (simple
        merge with no replacement selection). You will sort a binary
        file with 100,000 integers and assume a block size is 4KB.
        Output first 5 values and last 5 values when you are done.

    I certify that the code below is my own work.

      Exception(s): N/A
```

```
*/

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

const int BUFFER_SIZE{4000};

void seperateFile(string str)
{
    ifstream fin;
    fin.open(str, ios::binary);

    if (!fin)
    {
        cout << "File error\n";
        return;
    }

    fin.seekg(0, ios::end);
    int totalNum = fin.tellg() / sizeof(int);
    char fileBuffer[sizeof(int)];
    fin.close();

    fin.open(str, ios::binary);
    ofstream fout;
    fout.open("firstHalf.bin", ios::binary);
    for (int i = 0; i <= totalNum / 2; ++i)
    {
        fin.read(fileBuffer, sizeof(int));
        if (fin.gcount() != 0)
            fout.write(fileBuffer, sizeof(int));
    }
    fout.close();

    fout.open("secondHalf.bin", ios::binary);
    for (int i = 0; i <= totalNum / 2; ++i)
    {
        fin.read(fileBuffer, sizeof(int));
        if (fin.gcount() != 0)
            fout.write(fileBuffer, sizeof(int));
    }
    fout.close();
    fin.close();
}

void sortFile(string str)
```

```
{
    ifstream fin;
    ofstream fout;
    vector<int> vec;

    fin.open(str, ios::binary);
    int value = 0;
    fin.read(reinterpret_cast<char*>(&value), sizeof(int));
    while (fin.gcount() != 0)
    {
        vec.push_back(value);
        fin.read(reinterpret_cast<char*>(&value), sizeof(int));
    }
    fin.close();

    sort(vec.begin(), vec.end());

    fout.open(str, ios::binary);
    fout.clear();
    for (int i : vec)
        fout.write(reinterpret_cast<char*>(&i), sizeof(int));
    fout.close();
}

void mergeFile(string str1, string str2)
{
    ifstream fin1;
    ifstream fin2;
    ofstream fout;

    fin1.open(str1, ios::binary);
    fin2.open(str2, ios::binary);
    fout.open("result.bin", ios::binary);

    int i;
    int j;
    fin1.read(reinterpret_cast<char*>(&i), sizeof(int));
    fin2.read(reinterpret_cast<char*>(&j), sizeof(int));

    while (fin1.gcount() && fin2.gcount())
    {
        int k;
        if (i < j)
        {
            k = i;
            fin1.read(reinterpret_cast<char*>(&i), sizeof(int));
        }
        else
        {
            k = j;
            fin2.read(reinterpret_cast<char*>(&j), sizeof(int));
        }
```

```
        fout.write(reinterpret_cast<char*>(&k), sizeof(int));
    }

    while (fin1.gcount())
    {
        fout.write(reinterpret_cast<char*>(&i), sizeof(int));
        fin1.read(reinterpret_cast<char*>(&i), sizeof(int));
    }

    while (fin2.gcount())
    {
        fout.write(reinterpret_cast<char*>(&j), sizeof(int));
        fin2.read(reinterpret_cast<char*>(&j), sizeof(int));
    }

    fin1.close();
    fin2.close();
    fout.close();
}

void printFinalResult(string str)
{
    vector<int> vec;
    ifstream file;
    file.open(str, ios::binary);

    if (!file)
    {
        cout << "err\n";
        return;
    }
    file.seekg(0, ios::end);
    int totalNum = file.tellg() / sizeof(int);
    file.close();

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

    int value = 0;
    file.read(reinterpret_cast<char*>(&value), sizeof(int));
    while (file.gcount() != 0)
    {
        vec.push_back(value);
        file.read(reinterpret_cast<char*>(&value), sizeof(int));
    }

    for (int i = 1; i <= vec.size(); ++i)
        if (i <= 5 || i > vec.size() - 5)
            cout << "vec[" << i - 1 << "] = " << vec[i - 1] << endl;

}

int main()
```

```
{
    seperateFile("filetoSort.bin");
    sortFile("firstHalf.bin");
    sortFile("secondHalf.bin");
    mergeFile("firstHalf.bin", "secondHalf.bin");
    printFinalResult("result.bin");

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

    return 0;
}
```

Input/output below:

```
vec[0] = 0
vec[1] = 0
vec[2] = 0
vec[3] = 0
vec[4] = 1
vec[99995] = 32765
vec[99996] = 32765
vec[99997] = 32765
vec[99998] = 32765
vec[99999] = 32766
Author: Nero Li
```

Answer for Question 1:

A spanning tree S for a graph G is a graph contains all the vertex but the minimum edges that S need to be one connected component. Furthermore, a minimum spanning tree is a spanning tree that has the smallest total weight of edges.

There are many applications for us to use a MST. For example, if we want to find the connection network for each department that cost least, we can generate a graph that show all the possible connections as edges with weight as the cost and department as vertex, then find the MST to see the cheapest network.

Answer for Question 2:

The biggest difference between B-Tree and BST is the child that one node can have. BST can only have two child maximum for each node, but the B-Tree can have more than two and still keep the balance. It helps to read big chunk of data and reduce the time to find out where the data is.