CSCI 230 PA 12 Submission

Due Date: 05/31/2022 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 A and B) -- need to submit source code and I/O

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

Source code below:
exercise_2_option_B.cpp:

```
/*  Program: PA_12_exercise_2
    Author: Nero Li
    Class: CSCI 230
    Date: 05/31/2022
    Description:
        Option B: Perform file I/O of 100,000 random integers with
        three options below and collect times (3 different times
        for input and 3 different times for output).

    I certify that the code below is my own work.

      Exception(s): N/A

*/

#include <iostream>
#include <fstream>
#include <vector>
```

```
#include <ctime>

using namespace std;

const int SIZE{100000};

int generateNumber()
{
    return (rand() % INT_MAX);
}

void txtOneVal(string str)
{
    ofstream file;

    file.open(str, ios::binary);
    for (int i = 0; i < SIZE; ++i)
        file << generateNumber() << ' ';

    file.close();
}

void binOneVal(string str)
{
    ofstream file;

    file.open(str, ios::binary);
    int value;
    for (int i = 0; i < SIZE; ++i)
    {
        value = generateNumber();
        file.write(reinterpret_cast<char*>(&value), sizeof(int));
    }

    file.close();
}

void bin256Val(string str)
{
    ofstream file;

    file.open(str, ios::binary);
    int value[256];

    for (int i = 0; i < SIZE / 256; ++i)
    {
        for (int j = 0; j < 256; ++j)
            value[j] = generateNumber();
        file.write(reinterpret_cast<char*>(&value), sizeof(int) * 256);
    }

    file.close();
```

```
}

void readFile(string str, bool isTxt = false)
{
    int vec[SIZE];
    ifstream file;
    file.open(str, ios::binary);

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

    if (isTxt)
    {
        int i = 0;
        while (!file.eof())
        {
            int n;
            file >> n;
            vec[i++] = n;
        }

    }
    else
        file.read(reinterpret_cast<char*>(vec), sizeof(int) * SIZE);

    cout << "In file " << str << ":\n";
    for (int i = 0; i < 5; ++i)
        cout << "vec[" << i << "] = " << vec[i] << endl;
    cout << endl;
}

int main()
{
    srand(time(NULL));
    vector<string> str;
    str.push_back("randomInTxt.txt");
    str.push_back("randomInBin.bin");
    str.push_back("randomIn256Bin.bin");

    txtOneVal(str[0]);
    binOneVal(str[1]);
    bin256Val(str[2]);

    readFile(str[0], true);
    readFile(str[1]);
    readFile(str[2]);

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

```
    return 0;
}
```

Input/output below:

```
In file randomInTxt.txt:
vec[0] = 11700
vec[1] = 11144
vec[2] = 26341
vec[3] = 24305
vec[4] = 24154

In file randomInBin.bin:
vec[0] = 31230
vec[1] = 2444
vec[2] = 11827
vec[3] = 26365
vec[4] = 9810

In file randomIn256Bin.bin:
vec[0] = 31562
vec[1] = 4555
vec[2] = 6924
vec[3] = 4518
vec[4] = 19635

Author: Nero Li
```

exercise_2_option_A.cpp:

```
/*  Program: PA_12_exercise_2
    Author: Nero Li
    Class: CSCI 230
    Date: 05/31/2022
    Description:
        Option A: Perform Project P-14.1 on page 687 of C++ book
        in C++ or Java. We will limit to only two of the four
        algorithms.
        Write a C++ class that simulates the best-fit, worst-fit, first-
fit, and nextfit algorithms for memory management. Determine
experimentally which
method is the best under various sequences of memory requests.

    I certify that the code below is my own work.

      Exception(s): N/A

*/

#include <iostream>
#include <fstream>
#include <vector>
```

```
#include <map>
#include <ctime>

using namespace std;

const int SIZE = 1024;

int generateNumber()
{
    return (rand() % 15 + 5);
}

int randomSpace()
{
    return (rand() % SIZE);
}

map<int, int> checkAvailableHoles(bool blocks[], bool withOutput = false)
{
    map<int, int> holes;
    bool countingHoleSize = false;
    int holeStartPoint = -1;
    int size = 1;

    int countHoles = 0;
    int countSize = 0;

    for (int i = 0; i < SIZE; ++i)
    {
        if (blocks[i])
        {
            if (!countingHoleSize)
            {
                ++countHoles;
                countingHoleSize = true;
                holeStartPoint = i;
                size = 1;
            }
            else if (i == SIZE - 1)
            {
                ++size;
                countSize += size;
            }
            else
            {
                ++size;
            }
        }
        else
        {
            if (countingHoleSize)
            {
```

```
                countSize += size;
                countingHoleSize = false;
                holes[holeStartPoint] = size;
            }
        }
    }

    if (withOutput)
        cout << " - " << countHoles << " block(s) are still available.\n"
<< " - " << countSize * 4 << " byte(s) of memory still available.\n" <<
endl;


    return holes;
}

void doFit(bool isFirstFit)
{
    bool blocks[SIZE];
    for (int i = 0; i < SIZE; ++i)
        blocks[i] = false;
    int totalAllocation = 0;
    while (totalAllocation <= (SIZE) / 2)
    {
        int allocationTime = generateNumber();
        int startBlock = randomSpace();
        totalAllocation += allocationTime;
        for (int i = 0; i < allocationTime && i + startBlock < SIZE; ++i)
            blocks[startBlock + i] = true;
    }

    bool allocationFail = false;
    while (!allocationFail)
    {
        int allocationTime = generateNumber();
        map<int, int> holes = checkAvailableHoles(blocks);
        int holeStartPoint = -1;

        if (isFirstFit)
        {
            // main part for first-fit
            for (auto i : holes)
            {
                if (i.second >= allocationTime)
                {
                    holeStartPoint = i.first;
                    break;
                }
            }
        }
        else
        {
```

```
            // main part for best-fit
            int minGap = SIZE;
            for (auto i : holes)
            {
                int curGap = i.second - allocationTime;
                if (curGap >= 0 && curGap < minGap)
                {
                    minGap = curGap;
                    holeStartPoint = i.first;
                }
            }
        }


        if (holeStartPoint = -1)
        {
            allocationFail = true;
        }
        else
        {
            for (int j = 0; j < allocationTime; ++j)
                blocks[holeStartPoint + j] = true;
            allocationFail = false;
        }
    }

    if (isFirstFit)
        cout << "For first-fit:\n";
    else
        cout << "For best-fit:\n";
    checkAvailableHoles(blocks, true);
}

int main()
{
    srand(time(NULL));

    doFit(false);
    doFit(true);

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

    return 0;
}
```

Input/output below:

```
For best-fit:
 - 26 block(s) are still available.
 - 1668 byte(s) of memory still available.

For first-fit:
```

```
 - 24 block(s) are still available.
 - 1604 byte(s) of memory still available.

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.