CSE 551 Programming Assignment #1

Name: Sang-Hun Sim

Topic: Ford-Fulkerson Algorithm for the maximum flow

Language / IDE: C++ / Visual Studio 2019

Pseudo Code

1. Initialize "Flight" type vector to store NAS information

2. Read and store the input data to the vector initialized in step 1.

3. Convert the name of airport to numbers in accordance with their names and times

4. Set capacity large number(INF) to avoid flight between the same airports

5. Ford-Fulkerson method

While there is an augmented path from source to destination

a) Implement BFS method to find the path

b) Initialize "path_flow = INF" to find the minimum flow of the path

c) Search augmented path in backward, update "path_flow"

d) Add the minimum flow

i) Subtract "path_flow" to capacity

ii) Add "path_flow" to residual value

e) Terminate the loop if there is not augment path

f) Return "result"

Output: 8727

```
8727
C:₩Users₩ssh90₩source₩repos₩CSE551₩Debug₩CSE551.exe (process 18552) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automat
ically close the console when debugging stops.
Press any key to close this window . . .
```

Source Code

```cpp
#define _CRT_SECURE_NO_WARNINGS
#include <vector>
#include <iostream>
#include <queue>
#define INF 1000
#define n 194

class Flight
{
private:
	char start[4];
	char sink[4];
	int depart;
	int arrival;
	int capacity;

public:
	Flight(char* a, char* b, int c, int d, int e) {
		strcpy(start, a);
		strcpy(sink, b);
		depart = c;
		arrival = d;
		capacity = e;
	}
	// To check if txt.file is read correclty
	void getString() {
		std::cout << start << " " << sink << " " << depart << " " <<
			arrival << " " << capacity << std::endl;
	}
	char* getStart() {
		return start;
	}
	char* getDestination() {
		return sink;
	}
	int getDepart() {
		return depart;
	}
	int getArrival() {
		return arrival;
	}
	int getCapacity() {
		return capacity;
	}
};

// To read and store data
void openFiles(char* name, std::vector<Flight>& arr)
{
	FILE* file = fopen(name, "r");
	char line[1024];
	while (1) {
		char* pChar = fgets(line, sizeof(line), file);

		char start[4];
		char sink[4];
		int depart;
		int arrival;
		int capacity;

		char* cur = strtok(pChar, " \t");
		strcpy(start, cur);
		cur = strtok(NULL, " \t");
		strcpy(sink, cur);
```

```
			cur = strtok(NULL, " \t");
			depart = atoi(cur);
			cur = strtok(NULL, " \t");
			arrival = atoi(cur);
			cur = strtok(NULL, " \t");
			capacity = atoi(cur);

			Flight flight = Flight(start, sink, depart, arrival, capacity);

			arr.push_back(flight);

			if (feof(file) == 1) break;
		}
		fclose(file);
}

// To assign the number to each vertices
// in accordance with the airport and their time
inline constexpr int convert(const char* x) {
		int temp = 0;
		temp = temp << 8;
		temp += x[0];
		temp = temp << 8;
		temp += x[1];
		temp = temp << 8;
		temp += x[2];
		return temp;
}
int convert(const char* Name, int t)
{
		switch (convert(Name)) {
		case convert("LAX"):
			return 0;
		case convert("SFO"):
			return 1 + t;
		case convert("PHX"):
			return 25 + t;
		case convert("SEA"):
			return 49 + t;
		case convert("DEN"):
			return 73 + t;
		case convert("ATL"):
			return 97 + t;
		case convert("ORD"):
			return 121 + t;
		case convert("BOS"):
			return 145 + t;
		case convert("IAD"):
			return 169 + t;
		}
		return 193;
}

//Ford-Fulkerson method with BFS
int fordFulkerson(int flow[n][n], int start, int sink)
{
		int d[n]; // To check if visited
		int x, y;
		int parent[n];
		int result = 0;
		while (1)
		{
			std::fill(d, d+n, -1); // Assuming all vertices are not visited
			std::queue<int> q; // To track the path
			q.push(start);
			while (!q.empty()) {
				int x = q.front();
```

```
				q.pop();
				for (int i = 0; i < n; i++) // To check adjacent airport
				{
					if (flow[x][i] > 0 && d[i] == -1) // if flowable or not 
visited
					{
						q.push(i);
						parent[i] = x;
						d[i] = x; // mark as visited
					}
				}
			}
			if (d[sink] == -1) break; // terminate if there is no augmenting path

			int path_flow = INF; // To find the minimum value
			// Search the flow backward
			for (y = sink; y != start; y = d[y])
			{
				x = parent[y];
				path_flow = std::min(path_flow, flow[x][y]);
			}
			// Add the minimum flow
			for (y = sink; y != start; y = d[y])
			{
				x = parent[y];
				flow[x][y] -= path_flow;
				flow[y][x] += path_flow;
			}
			result += path_flow;
		}
		return result;
}

int main(void) {
		std::vector<Flight> flight;
		char name[] = "flights.txt";
		openFiles(name, flight);

		int direct = 0; // non-stop flights
		int c[n][n] = { 0, }; // multi-hop flights
		int result; // final result
		int x;
		int y;
		for (int i = 0; i < flight.size(); i++)
		{
			x = convert(flight[i].getStart(), flight[i].getDepart());
			y = convert(flight[i].getDestination(), flight[i].getArrival());
			if (x == 0 && y == n-1)
			{
				direct = direct + flight[i].getCapacity();
			}
			else
			{
				c[x][y] = c[x][y] + flight[i].getCapacity();
			}
		}

		// To avoid traveling between same airport
		for (int i = 1; i < n - 1; i++)
		{
			for (int j = 1; j < n - 1; j++)
			{
				if (i < j)
				{
					if (((i > 0 && j > 0) && (i < 25 && j < 25)) || ((i > 24 
&& j > 24) && (i < 49 && j < 49))
```

```
                                        || ((i > 48 && j > 48) && (i < 73 && j < 73)) ||
((i > 72 && j > 72) && (i < 97 && j < 97))
                                        || ((i > 96 && j > 96) && (i < 121 && j < 121)) ||
((i > 120 && j > 120) && (i < 145 && j < 145))
                                        || ((i > 144 && j > 144) && (i < 169 && j < 169))
|| ((i > 168 && j > 168)))
                                        c[i][j] = INF;
                        }
                }
        }
        result = fordFulkerson(c, 0, n-1) + direct;
        //for (int i = 0; i < n; i++) {
        //      for (int j = 0; j < n; j++) {
        //              if (c[i][j] != 0){
        //              cout << "c" << i << " " << j << " " << c[i][j] << endl;
        //              }
        //      }
        //}
        printf("%d", result);
        return 0;
}
```