Pr 5

```
#include <iostream>
#include <map>
using namespace std;
map<string, string> dictionary;
void AddKeyword() {
    string word, meaning;
    cout << "Enter keyword: ";
    cin >> word;
    cout << "Enter meaning: ";
    cin >> meaning;
    dictionary[word] = meaning;
    cout << "Keyword added successfully.\n";
}
void DeleteKeyword() {
    string word;
    cout << "Enter keyword to delete: ";
    cin >> word;
    if (dictionary.erase(word) == 1) {
        cout << "Keyword deleted successfully.\n";
    } else {
        cout << "Keyword not found.\n";
    }
}
void UpdateKeyword() {
    string word, meaning;
    cout << "Enter keyword to update: ";
    cin >> word;
    if (dictionary.find(word) != dictionary.end()) {
        cout << "Enter new meaning: ";
        cin >> meaning;

`
```

```
        dictionary[word] = meaning;
        cout << "Meaning updated successfully.\n";
    } else {
        cout << "Keyword not found.\n";
    }
}
void DisplayAscending() {
    cout << "Dictionary in Ascending Order:\n";
    for (auto it = dictionary.begin(); it != dictionary.end(); ++it) {
        cout << it->first << ": " << it->second << endl;
    }
}
void DisplayDescending() {
    cout << "Dictionary in Descending Order:\n";
    for (auto it = dictionary.rbegin(); it != dictionary.rend(); ++it) {
        cout << it->first << ": " << it->second << endl;
    }
}
void FindMaxComparisons() {
    string word;
    cout << "Enter keyword to find maximum comparisons: ";
    cin >> word;
    auto it = dictionary.find(word);
    if (it != dictionary.end()) {
        cout << "Maximum comparisons required: " << distance(dictionary.begin(), it) + 1 << endl;
    } else {
        cout << "Keyword not found.\n";
    }
}

int main() {

`
```

```
int choice;
do {
   cout << "\nDictionary Operations:\n";
   cout << "1. Add Keyword\n";
   cout << "2. Delete Keyword\n";
   cout << "3. Update Meaning\n";
   cout << "4. Display in Ascending Order\n";
   cout << "5. Display in Descending Order\n";
   cout << "6. Find Maximum Comparisons\n";
   cout << "7. Exit\n";
   cout << "Enter your choice: ";
   cin >> choice;
   switch (choice) {
      case 1:
         AddKeyword();
        break;
      case 2:
         DeleteKeyword();
         break;
      case 3:
         UpdateKeyword();
         break;
      case 4:
         DisplayAscending();
         break;
      case 5:
         DisplayDescending();
         break;
      case 6:
         FindMaxComparisons();
         break;
```

`

```
            case 7:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while (choice != 7);
    return 0;
}
```

Output

```
Dictionary Operations:
1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Display in Ascending Order
5. Display in Descending Order
6. Find Maximum Comparisons
7. Exit
Enter your choice: 1
Enter keyword: A
Enter meaning: Apple
Keyword added successfully.

Dictionary Operations:
1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Display in Ascending Order
5. Display in Descending Order
6. Find Maximum Comparisons
7. Exit
```

`

Enter your choice: 1

Enter keyword: B

Enter meaning: Board

Keyword added successfully.

Dictionary Operations:

1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Display in Ascending Order
5. Display in Descending Order
6. Find Maximum Comparisons
7. Exit

Enter your choice: 1

Enter keyword: C

Enter meaning: Chair

Keyword added successfully.

Dictionary Operations:

1. Add Keyword
2. Delete Keyword
3. Update Meaning
4. Display in Ascending Order
5. Display in Descending Order
6. Find Maximum Comparisons
7. Exit

Enter your choice: 4

Dictionary in Ascending Order:

A: Abcd

B: BDDE

Dictionary Operations:

`

1. Add Keyword

2. Delete Keyword

3. Update Meaning

4. Display in Ascending Order

5. Display in Descending Order

6. Find Maximum Comparisons

7. Exit

Enter your choice: 5

Dictionary in Descending Order:

B: BDDE

A: Abcd

Dictionary Operations:

1. Add Keyword

2. Delete Keyword

3. Update Meaning

4. Display in Ascending Order

5. Display in Descending Order

6. Find Maximum Comparisons

7. Exit

Enter your choice: 6

Enter keyword to find maximum comparisons: A

Maximum comparisons required: 1

`

PR 6

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <tuple>
#include <algorithm> // for find()
using namespace std;
struct Edge {
    int destination;
    int cost
};
class Graph {
public:
    vector<string> cities;
    unordered_map<int, vector<Edge>> adjList;
    void addFlight(int source, int destination, int cost) {
        adjList[source].push_back({destination, cost});
        adjList[destination].push_back({source, cost}); // Assuming flights are bidirectional
    }
    bool isConnected() {
        vector<bool> visited(cities.size(), false);
        dfs(0, visited); // Start from any city (assume city at index 0)
        for (bool v : visited) {
            if (!v) {return false; }
        return true;
    }
private:
    void dfs(int city, vector<bool>& visited) {
        visited[city] = true;
        for (const auto& edge : adjList[city])
            if (!visited[edge.destination])
```

`

```
            dfs(edge.destination, visited);
        }
    };
    int main() {
        vector<string> cities = {"Chicago", "New York", "Los Angeles", "Seattle"};
        vector<tuple<string, string, int>> flights = {{"Chicago", "New York", 2}, {"New York", "Los Angeles", 5}, {"Los Angeles", "Seattle", 3}};
        Graph graph;
        for (const auto& city : cities)
            graph.cities.push_back(city);
        for (const auto& flight : flights) {
            string source, destination;
            int cost;
            tie(source, destination, cost) = flightl
            int sourceIndex = find(cities.begin(), cities.end(), source) - cities.begin();
            int destinationIndex = find(cities.begin(), cities.end(), destination) - cities.begin();
            graph.addFlight(sourceIndex, destinationIndex, cost);
        }
        if (graph.isConnected()) cout << "The flight network is connected!" << endl;
        else cout << "The flight network is not connected." << endl;
        return 0;
    }
```

Output:

```
The flight network is connected.
=== Code Execution Successful ===
```

`

PR 7

```
#include <iostream>
using namespace std;
int main()
{
   int n, i, j, k, row, col, mincost = 0, min;
   char op;
   cout << "Enter no. of vertices: ";
   cin >> n;
   int cost[n][n];
   int visit[n];
   for (i = 0; i < n; i++)
      visit[i] = 0;
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         cost[i][j] = -1;
      }}
   for (i = 0; i < n; i++)
   {
      for (j = i + 1; j < n; j++)
      {
         cout << "Do you want an edge between " << i + 1 << " and " << j + 1 << ": ";
         cin >> op;
         if (op == 'y' || op == 'Y')
         {
            cout << "Enter weight: ";
            cin >> cost[i][j];
            cost[j][i] = cost[i][j];
         }}}
```

`

```
   visit[0] = 1;
   for (k = 0; k < n - 1; k++)
   {
      min = 999;
      for (i = 0; i < n; i++)
      {
         for (j = 0; j < n; j++)
         {
            if (visit[i] == 1 && visit[j] == 0)
            {
               if (cost[i][j] != -1 && min > cost[i][j])
               {
                  min = cost[i][j];
                  row = i;
                  col = j;
               }}}}
      mincost += min;
      visit[col] = 1;
      cost[row][col] = cost[col][row] = -1;
      cout << row + 1 << "->" << col + 1 << endl;    }
   cout << "\nMin. Cost: " << mincost;
   return 0;
}
```

Output:

```
Enter no. of vertices: 5
Do you want an edge between 1 and 2: Y
Enter weight: 4
Do you want an edge between 1 and 3: Y
Enter weight: 8
Do you want an edge between 1 and 4: Y
Enter weight: 3
```

`

Do you want an edge between 1 and 5: N

Do you want an edge between 2 and 3: N

Do you want an edge between 2 and 4: Y

Enter weight: 7

Do you want an edge between 2 and 5: Y

Enter weight: 7

Do you want an edge between 3 and 4: Y

Enter weight: 2

Do you want an edge between 3 and 5: Y

Enter weight: 1

Do you want an edge between 4 and 5: Y

Enter weight: 9

1->4

4->3

3->5

1->2

Min. Cost: 10

=== Code Execution Successful ===

`