```
#include <iostream>
#include <vector>
#include <queue>
#include <unordered_map>
#include <algorithm>
using namespace std;

int adj_mat[50][50] = {0};
int visited[50] = {0};
unordered_map<int, vector<int>> adj_list;
void dfs(int s, int n, string arr[]) {
   visited[s] = 1;
   cout << arr[s] << " ";
   for (int i = 0; i < n; i++) {
      if (adj_mat[s][i] && !visited[i]) {
         dfs(i, n, arr);
      }
   }
}
void bfs(int s, int n, string arr[]) {
   vector<bool> visited(n, false);
   queue<int> bfsq;
   visited[s] = true;
   cout << arr[s] << " ";
   bfsq.push(s);
  while (!bfsq.empty()) {
      int v = bfsq.front();
      bfsq.pop();
      for (int neighbor : adj_list[v]) {
         if (!visited[neighbor]) {
            cout << arr[neighbor] << " ";
```

```
            visited[neighbor] = true;
            bfsq.push(neighbor);
        }
    }
  }
}

int main() {
  int n, u;

  cout << "Enter number of locations (nodes): ";
  cin >> n;

  string locations[n];
  for (int i = 0; i < n; i++) {
    cout << "Enter location #" << i << " (Landmark Name): ";
    cin >> locations[i];
  }

  cout << "\nYour locations are:\n";
  for (int i = 0; i < n; i++) {
    cout << "Location #" << i << ": " << locations[i] << endl;
  }

  cout << "\nEnter distances between connected locations (Enter 0 if not connected):\n";
  for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
      cout << "Enter distance between " << locations[i] << " and " << locations[j] << ": ";
      cin >> adj_mat[i][j];
      adj_mat[j][i] = adj_mat[i][j];
```

```
            if (adj_mat[i][j] > 0) {
                adj_list[i].push_back(j);
                adj_list[j].push_back(i);
            }
        }
    }

    cout << "\nAdjacency Matrix:\n\t";
    for (int i = 0; i < n; i++)
        cout << locations[i] << "\t";
    cout << endl;

    for (int i = 0; i < n; i++) {
        cout << locations[i] << "\t";
        for (int j = 0; j < n; j++) {
            cout << adj_mat[i][j] << "\t";
        }
        cout << endl;
    }

    cout << "\nEnter Starting Vertex (index): ";
    cin >> u;

    cout << "\nDFS Traversal: ";
    dfs(u, n, locations);
    fill_n(visited, 50, 0);  // Reset visited
    cout << "\nBFS Traversal: ";
    bfs(u, n, locations);
   return 0;
}
```

Output

Enter number of locations (nodes): 3

Enter location #0 (Landmark Name): A

Enter location #1 (Landmark Name): B

Enter location #2 (Landmark Name): C

Your locations are:

Location #0: A

Location #1: B

Location #2: C

Enter distances between connected locations (Enter 0 if not connected):

Enter distance between A and B: 1

Enter distance between A and C: 1

Enter distance between B and C: 0

Adjacency Matrix:

	A	B	C
A	0	1	1
B	1	0	0
C	1	0	0

Enter Starting Vertex (index): 0

DFS Traversal: A B C

BFS Traversal: A B C