BFSPath in C++

```
#include <iostream>
#include <vector>
#include <deque>

using namespace std;

// Edge structure representing an edge between two vertices
struct Edge {
    int src;
    int nbr;

    Edge(int src, int nbr) {
        this->src = src;
        this->nbr = nbr;
    }
};

// Pair structure to store vertex and path so far
struct Pair {
    int v;
    string psf;

    Pair(int v, string psf) : v(v), psf(psf) {}
};

// Function to add an edge between two vertices
void addEdge(vector<Edge>* graph, int v1, int v2) {
    graph[v1].push_back(Edge(v1, v2));
    graph[v2].push_back(Edge(v2, v1));
}

int main() {
    int vtces = 7; // Number of vertices
    vector<Edge>* graph = new vector<Edge>[vtces]; // Adjacency list of edges

    // Adding edges to the graph
    addEdge(graph, 0, 1);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 3);
    addEdge(graph, 0, 3);
    addEdge(graph, 3, 4);
    addEdge(graph, 4, 5);
    addEdge(graph, 5, 6);
    addEdge(graph, 4, 6);

    int src = 0; // Source vertex for BFS

    deque<Pair> q; // Queue for BFS
    vector<bool> visited(vtces, false); // Array to mark visited vertices

    q.push_back(Pair(src, to_string(src))); // Pushing source vertex with path so far
```

Graph Structure:

Edges (undirected):

0 -- 1
1 -- 2
2 -- 3
0 -- 3
3 -- 4
4 -- 5
5 -- 6
4 -- 6

This gives us the following adjacency list:

Vertex	Neighbors
0	1, 3
1	0, 2
2	1, 3
3	2, 0, 4
4	3, 5, 6
5	4, 6
6	5, 4

🧠 BFS Behavior:

- Queue type: deque
- Visited is marked **only when popped** (standard BFS behavior)
- Pair stores (vertex, path-so-far)
- Queue allows tracking of the shortest path from source

📜 Dry Run Table:

Step	Queue (Front → Back)	Visited	Output
1	(0, "0")	{}	
2	—	{0}	0 0
	Enqueue: (1, "01"), (3, "03")		
3	(1, "01"), (3, "03")	{0}	
4	—	{0, 1}	1 01
	Enqueue: (0, "010"), (2, "012")		
5	(3, "03"), (0, "010"), (2, "012")	{0, 1}	
6	—	{0, 1, 3}	3 03
	Enqueue: (2, "032"), (0, "030"), (4, "034")		
7	(0, "010"), (2, "012"), (2, "032"), (0, "030"), (4, "034")	{0, 1, 3}	
8	— 0 already visited → skip	{0, 1, 3}	
9	—	{0, 1, 2, 3}	2 012
	Enqueue: (1, "0121"), (3, "0123")		
10	— 2 already visited → skip		
11	— 0 already visited → skip		

```
    while (!q.empty()) {
        Pair rem = q.front();
        q.pop_front();

        if (visited[rem.v]) {
            continue;
        }
        visited[rem.v] = true;

        cout << rem.v << " " << rem.psf << endl; // Printing vertex and path so far

        // Iterating through all adjacent vertices
        for (Edge e : graph[rem.v]) {
            q.push_back(Pair(e.nbr, rem.psf + to_string(e.nbr))); // Adding adjacent vertices to queue
        }
    }

    delete[] graph; // Freeing dynamically allocated memory for graph
    return 0;
}
```

Step	Queue (Front → Back)	Visited	Output
12	—	{0,1,2,3,4}	4 034
	Enqueue: (3, "0343"), (5, "0345"), (6, "0346")		
13	— 1 already visited → skip		
14	— 3 already visited → skip		
15	—	{..., 5}	5 0345
	Enqueue: (4, "03454"), (6, "03456")		
16	—	{..., 6}	6 0346
	Enqueue: (5, "03465"), (4, "03464")		
...	All remaining vertices already visited → skip		

✅ **Final Output:**

(printed in order of **first encounter** in BFS)

```
0 0
1 01
3 03
2 012
4 034
5 0345
6 0346
```

Output:-

```
0 0
1 01
3 03
2 012
4 034
5 0345
6 0346
```