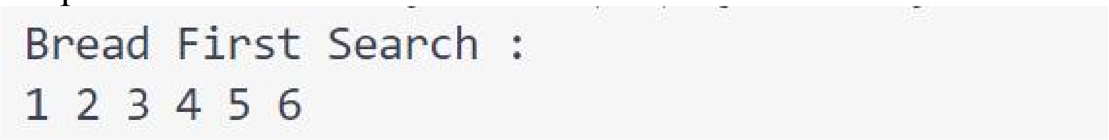
Code

```c
#include <stdio.h>
#define n 6

int queue[100];
int front=0, rear=0;

void push(int num) {
    queue[rear] = num;
    rear++;
}

void pop() {
    front++;
}

void bfs(int graph[n][n], int vis[]) {
    push(1);
    vis[0] = 1;
    while (front != rear) {
        int curr = queue[front];
        printf("%d ", curr);
        pop();

        for (int i=0; i<n; i++) {
            if (graph[curr-1][i]==1 && vis[i]==0) {
                vis[i] = 1;
                push(i+1);
            }
        }
    }
}

int main() {
    int graph[n][n] = {
        {0, 1, 1, 0, 0, 0},
        {1, 0, 1, 1, 0, 0},
        {1, 1, 0, 0, 1, 0},
        {0, 1, 0, 0, 1, 1},
        {0, 0, 1, 1, 0, 1},
        {0, 0, 0, 1, 1, 0}
    };

    int vis[n] = {0};
    bfs(graph, vis);
    return 0;
}
```

Output

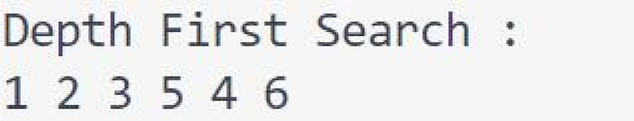


```
Bread First Search :
1 2 3 4 5 6
```

Code

```
#include <stdio.h>
#define n 6

void dfs(int graph[n][n], int curr, int vis[]) {
    printf("%d ", curr);
    vis[curr-1] = 1;

    for (int i=0; i<n; i++){
        if (graph[curr-1][i]==1 && vis[i]==0) {
            dfs(graph, i+1, vis);
        }
    }
}

int main() {
    int graph[n][n] = {
        {0, 1, 1, 0, 0, 0},
        {1, 0, 1, 1, 0, 0},
        {1, 1, 0, 0, 1, 0},
        {0, 1, 0, 0, 1, 1},
        {0, 0, 1, 1, 0, 1},
        {0, 0, 0, 1, 1, 0}
    };
    int vis[n] = {0};

    printf("Depth First Search : \n");
    dfs(graph, 1, vis);
    return 0;
}
```

Output



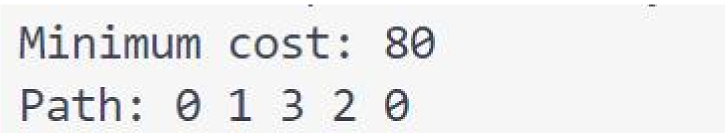
```
Depth First Search :
1 2 3 5 4 6
```

Code

```c
#include <stdio.h>
#include <limits.h>

#define n 4

int min(int a, int b) {
    return a < b ? a : b;
}

int findMinEdge(int dist[n][n], int i) {
    int minEdge = INT_MAX;
    for (int j = 0; j < n; j++) {
        if (i != j) {
            minEdge = min(minEdge, dist[i][j]);
        }
    }
    return minEdge;
}

int calc(int dist[n][n]) {
    int bound = 0;
    for (int i = 0; i < n; i++) {
        bound += findMinEdge(dist, i);
    }
    return bound / 2;
}

void tsp(int dist[n][n], int vis[n], int currCost, int bound, int level, int currPath[], int
res[], int* minCost) {
    if (level == n) {
        int total = currCost + dist[currPath[level-1]][currPath[0]];
        if (total < *minCost) {
            *minCost = total;
            for (int i = 0; i < n; i++) {
                res[i] = currPath[i];
            }
        }
        return;
    }

    for (int i = 0; i < n; i++) {
        if (!vis[i]) {
            int tempBound = bound;
            vis[i] = 1;
            currPath[level] = i;

            int newCost = currCost + dist[currPath[level - 1]][i];
            tempBound -= findMinEdge(dist, currPath[level - 1]);

            if (newCost + tempBound < *minCost) {
                tsp(dist, vis, newCost, tempBound, level + 1, currPath, res, minCost);
            }

            vis[i] = 0;
        }
    }
}
```

```
int main() {
    int dist[n][n] = {
        {0, 3, 6, 7},
        {3, 0, 2, 5},
        {6, 2, 0, 4},
        {7, 5, 4, 0}
    };

    int vis[n] = {0};
    int currPath[n + 1];
    int res[n];
    int minCost = INT_MAX;

    int initialBound = calc(dist);

    vis[0] = 1;
    currPath[0] = 0;

    tsp(dist, vis, 0, initialBound, 1, currPath, res, &minCost);

    printf("Minimum cost: %d\n", minCost);
    printf("Path: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", res[i]);
    }
    printf("0\n");

    return 0;
}
```

Output



```
Minimum cost: 80
Path: 0 1 3 2 0
```