```cpp
#include <algorithm>
#include <cstdio>
#include <vector>
#include <queue>
using namespace std;

typedef pair<int, int> ii;      // In this chapter, we will frequently use these
typedef vector<ii> vii;      // three data type shortcuts. They may look cryptic
typedef vector<int> vi;   // but shortcuts are useful in competitive programming

int V, E, a, b, s;
vector<vii> AdjList;
vi p;                                 // addition: the predecessor/parent vector

void printPath(int u) {    // simple function to extract information from `vi p'
  if (u == s) { printf("%d", u); return; }
  printPath(p[u]);   // recursive call: to make the output format: s -> ... -> t
  printf(" %d", u); }

int main() {
  /*
  // Graph in Figure 4.3, format: list of unweighted edges
  // This example shows another form of reading graph input
  13 16
  0 1    1 2    2  3   0  4   1  5   2  6    3  7   5  6
  4 8    8 9    5 10   6 11   7 12   9 10   10 11  11 12
  */

  freopen("in_04.txt", "r", stdin);

  scanf("%d %d", &V, &E);

  AdjList.assign(V, vii()); // assign blank vectors of pair<int, int>s to AdjList
  for (int i = 0; i < E; i++) {
    scanf("%d %d", &a, &b);
    AdjList[a].push_back(ii(b, 0));
    AdjList[b].push_back(ii(a, 0));
  }

  // as an example, we start from this source, see Figure 4.3
  s = 5;

  // BFS routine
  // inside int main() -- we do not use recursion, thus we do not need to create
separate function!
  vi dist(V, 1000000000); dist[s] = 0;      // distance to source is 0 (default)
  queue<int> q; q.push(s);                                  // start from source
  p.assign(V, -1); // to store parent information (p must be a global variable!)
  int layer = -1;                             // for our output printing purpose
  bool isBipartite = true;       // addition of one boolean flag, initially true

  while (!q.empty()) {
    int u = q.front(); q.pop();                          // queue: layer by layer!
    if (dist[u] != layer) printf("\nLayer %d: ", dist[u]);
    layer = dist[u];
    printf("visit %d, ", u);
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
      ii v = AdjList[u][j];                             // for each neighbors of u
      if (dist[v.first] == 1000000000) {
        dist[v.first] = dist[u] + 1;                    // v unvisited + reachable
        p[v.first] = u;          // addition: the parent of vertex v->first is u
        q.push(v.first);                                // enqueue v for next step
      }
      else if ((dist[v.first] % 2) == (dist[u] % 2))              // same parity
        isBipartite = false;
  } }

  printf("\nShortest path: ");
  printPath(7), printf("\n");
```

```
  printf("isBipartite? %d\n", isBipartite);

  return 0;
}
```