Report-01: Title :Heuristic Function

CSE-0408 Summer 2021

Safayet tanvir shiddiki
ID: ug02-37-14-016
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email address: safayettanvirshiddiki99@gmail.com

***Abstract*—We are given an initial state and we have to reach the goal state which is also specified. In this project, we have used various informed search methods like a*algorithm, ida* algorithm to solve the puzzle. Various heuristic involved in the informed search like number of misplaced tiles, Manhattan distance were analyzed; Manhattan distance being one of the most popular ones. Drawbacks of the heuristics are mentioned and an improvement in Manhattan distance heuristic is implemented.**

***Index Terms*——About Heuristic Function ,the 8-puzzle problem in C++.**

I. Introduction

The 8-puzzle is a sliding tile puzzle that is made up of a square structured frame area containing tiles in random/irregular order with one tile missing. It is a smaller version of the 15-puzzle (also called Gem Puzzle, Boss Puzzle, Game of Fifteen, Mystic Square and numerous other names) . 8-puzzle is basically a frame area separated into 3x3 grids containing 8 tiles and one void grid. The tiles are marked in some way so as they can be identified. The tiles are mostly numbered from 1 to 8. We are given with an initial configuration of the tiles. A desired final configuration is also given. We have to reach the final state by sliding the tiles using the empty grid present.

II. Literature Review

AIn 2012 Farhad S. et. al. [4] proposed new resolution for solving N-queens by using combination of DFS (Depth First Search) and BFS (Breadth First Search) techniques. The proposed algorithm act based on placing queens on chess board directly. The results report the performance and run time of this approach.

III. Proposed Methodology

Heuristics are methods for solving problems in a quick way that delivers a result that is sufficient enough to be useful given time constraints. Investors and financial professionals use a heuristic approach to speed up analysis and investment decisions. Current state [[8,1,2],[3,6,4],[0,7,5]] and the goal state is [[1,2,3],[8,0,4],[7,6,5]

IV. Rules for Solving the Puzzle

As we have already seen that an informed search makes use of heuristic functions in order to reach the goal node in a more prominent way. Therefore, there are several pathways in a search tree to reach the goal node from the current node. The selection of a good heuristic function matters certainly. A good heuristic function is determined by its efficiency. More is the information about the problem, more is the processing time. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive. Admissibility of the heuristic function is given as: h(n) ¡= h*(n) 1.UP 2.Down 3.Right 4.Left

V. Conclusion and Future Work

An approach for solving the 8-puzzle problem has been proposed which minimizes the memory required while achieving optimum results. All the heuristics have been applied to a* algorithm to bring uniformity and the results were shown. Comparison of memory used makes sense only in a star algorithm. At last memory use of all the heuristics are compared. Ida* algorithm was implemented but only with one heuristic i.e. number of misplaced tiles.

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] Doran, 1. and Michie, D. “Experiments with the Graph-Traverser Algorithm” Proceedings of the Royal Society, 294 (A), 1966, pp. 235-259..

[2] peter e. hart,Nils j. nillson, member IEEE, “A formal basis for the heuristic determination of Minimum cost paths”. Conference on Automated Planning and Scheduling (ICAPS 2008

[3] Bellmore, M. and Nemhauser, G.L. ”The Traveling Salesman Problem: A Survey”. Operations Research 16 (1968), 538-558.

[4] Sudip Roy. ”Artificial intelligence approach to test vector reordering for dynamic power reduction during VLSI testing”, TENCON 2008 IEEE Region 10 Conference, 11/2008

```
#include<bits/stdc++.h>
using namespace std;
#define D(x) cerr<<__LINE__<<" : "<<#x<<" -> "<<x<<endl
#define rep(i,j) for(int i = 0; i < 3; i++) for(int j = 0; j < 3; j++)
#define PII pair < int, int >
typedef vector<vector<int>> vec2D;
const int MAX = 1e5+7;
int t=1, n, m, l, k, tc;
int dx[4] = {0, 0, 1, -1};
int dy[4] = {1, -1, 0, 0};
vec2D init{
 {8, 1, 2},
 {3, 6, 4},
 {0, 7, 5}
};
vec2D goal{
 {1, 3, 2},
 {8, 0, 4},
 {7, 6, 5}
};
//vec2D init{// {1, 2, 3},
// {8, 6, 0},
// {7, 5, 4}
//};
//vec2D goal{
// {1, 2, 3},
// {8, 0, 4},
// {7, 6, 5}
//};
```

Fig. 1. Example of a figure caption.

```
//vec2D init{
// {1, 3, 2},
// {4, 0, 7},
// {6, 5, 8}
//};
//vec2D goal{
// {0, 2, 4},
// {1, 3, 8},
// {6, 5, 7}
//};
struct Box {
 vec2D mat{ { 0,0,0 },{ 0,0,0},{ 0,0,0} };
 int diff, level;
 int x, y; int lastx, lasty;
 Box(vec2D a,int b = 0, int c = 0, PII p = {0,0}, PII q = {0,0}) {
 rep(i,j) mat[i][j] = a[i][j];
 diff = b;
 level = c;
 x = p.first;
 y = p.second;
 lastx = q.first;
 lasty = q.second;
 }
};
bool operator < (Box A, Box B) {
 if(A.diff == B.diff) return A.level < B.level;
 return A.diff < B.diff;
}
int isEqual(vec2D a, vec2D b) {
```

Fig. 2. Example of a figure caption.

```
int ret(0);
rep(i,j) if (a[i][j] != b[i][j]) ret--;
return ret;
}
bool check(int i, int j) { return i>=0 and i<3 and j>=0 and j<3;
}
void print(Box a) {
rep(i,j)
cout << a.mat[i][j] << (j == 2 ? "\n" : " ");
D(-a.diff);
D(-a.level);
cout << "(" << a.x << "," << a.y <<")\n\n";
}
void dijkstra(int x, int y) {
map < vec2D, bool > mp;
priority_queue < Box > PQ;
int nD = isEqual(init, goal);
Box src = {init, nD, 0, {x,y}, {-1,-1}};
PQ.push(src);
int state = 0;
while(!PQ.empty()) {
state++;
Box now = PQ.top();
PQ.pop(); print(now);
if(!now.diff) {
puts("Goal state has been discovered");
cout << "level : " << -now.level << "\n";
D(state);
break;
```

Fig. 3. Example of a figure caption.

```
}
if(mp[now.mat]) continue;
mp[now.mat] = true;
for(int i = 0; i < 4; i++) {
int xx = now.x + dx[i];
int yy = now.y + dy[i];
if(check(xx, yy)) {
if(now.lastx == xx and now.lasty == yy) continue;
Box temp = now;
swap(temp.mat[temp.x][temp.y], temp.mat[xx][yy]);
temp.diff = isEqual(temp.mat, goal);
temp.level = now.level - 1;
temp.x = xx;
temp.y = yy;
temp.lastx = now.x;
temp.lasty = now.y;
PQ.push(temp);
} }
}
}
signed main() {
puts("Current State:");
rep(i,j) cout << init[i][j] << (j == 2 ? "\n" : " ");
puts("");
puts("Goal State:");
rep(i,j) cout << goal[i][j] << (j == 2 ? "\n" : " ");
puts("\n.....Search Started.....\n");
rep(i,j) if(!init[i][j]) dijkstra(i,j);
return 0;
```

Fig. 4. Example of a figure caption.

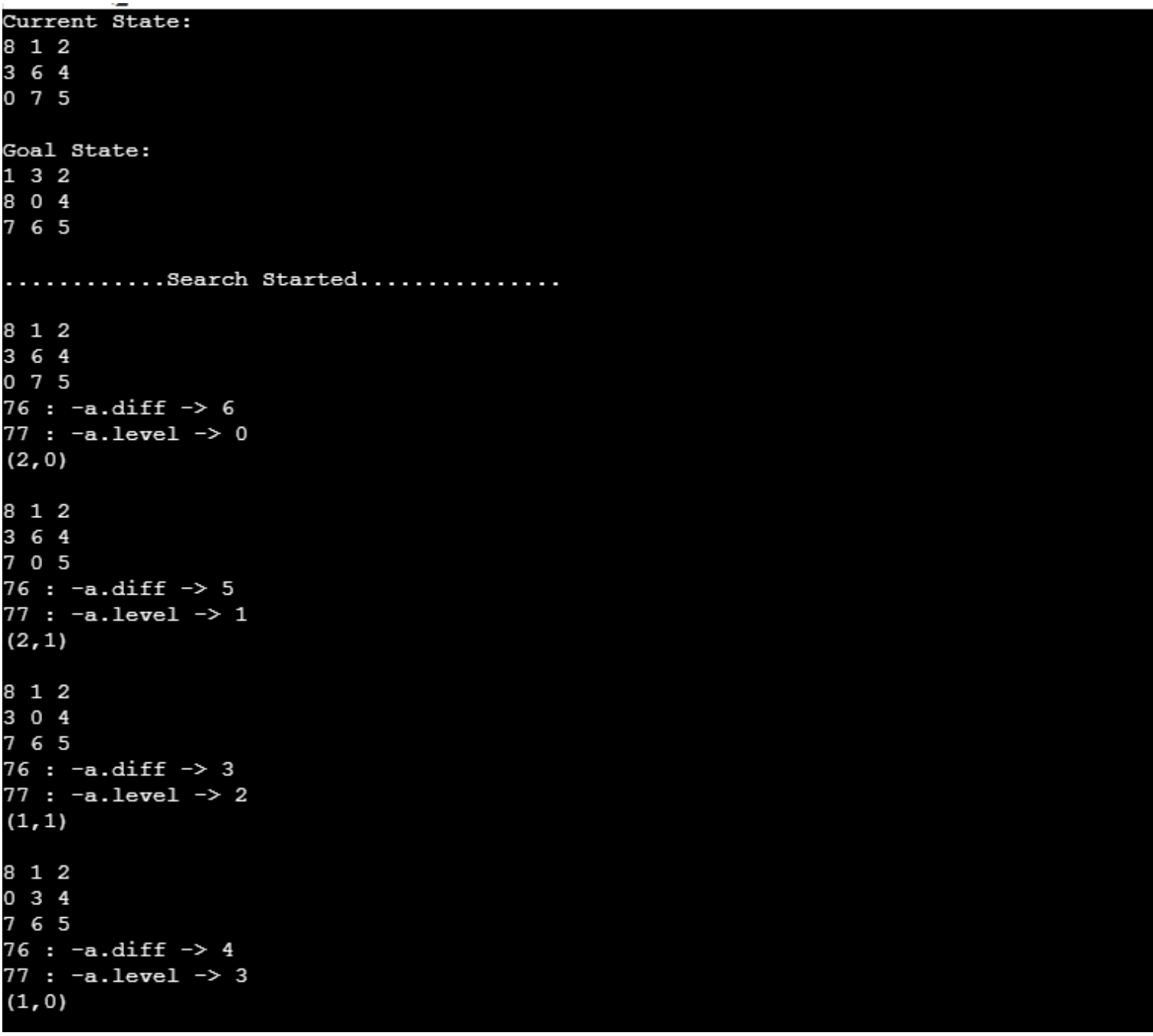

Fig. 5. Example of a figure caption.

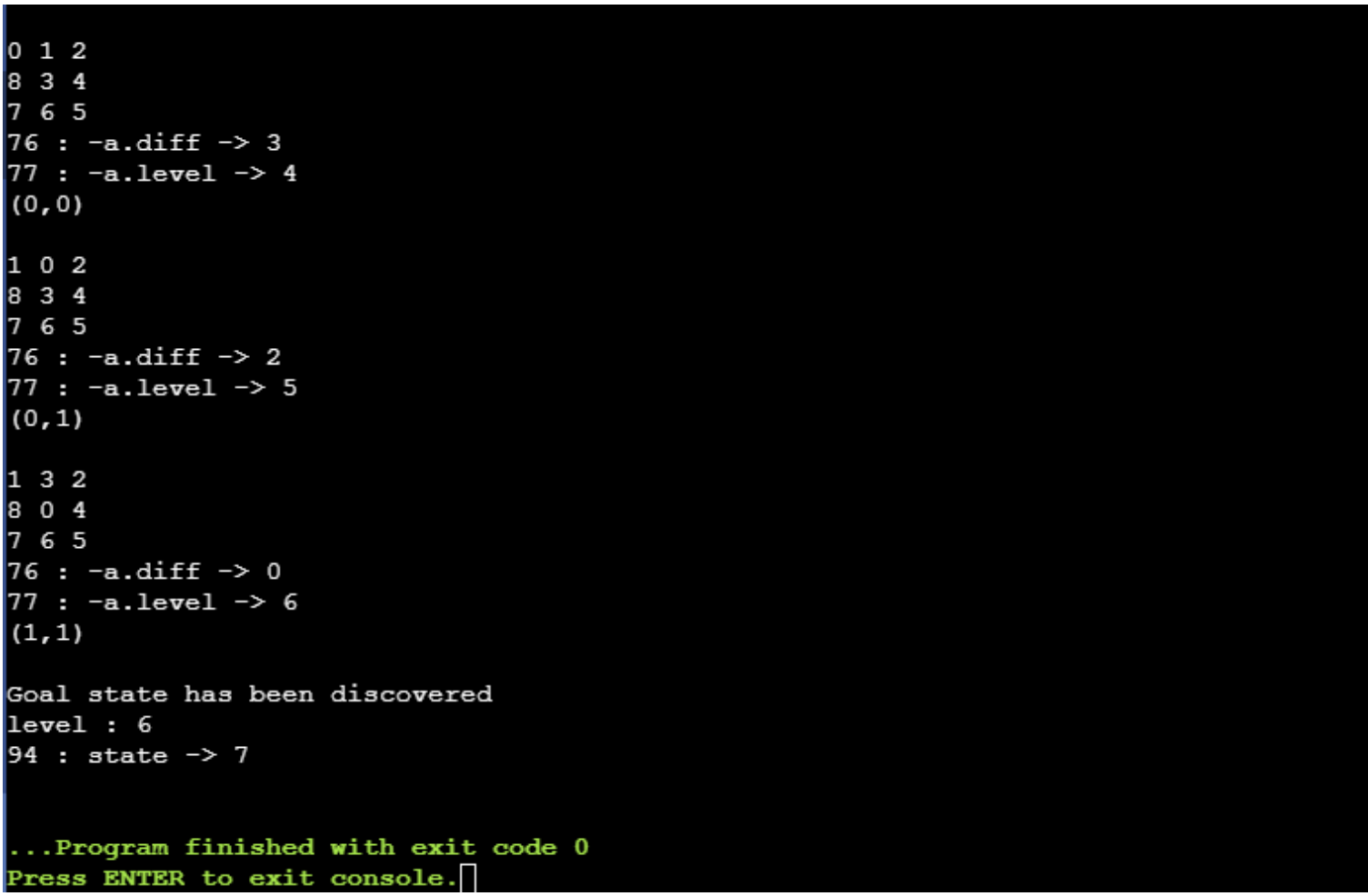

Fig. 6. Example of a figure caption.

Report-02: Title Breadth-First Search

CSE-0408 Summer 2021

Safayet tanvir shiddiki
ID: ug02-37-14-016
Department of Computer Science and Engineering
State University of Bangladesh (SUB)
Dhaka, Bangladesh
email address: safayettanvirshiddiki99@gmail.com

***Abstract*—Solving problem and learn about breadth first search algorithm using c++ language. Breadth First Traversal or Breadth First Search is a recursive algorithm for searching all the vertices ofa graph or tree data structure**

***Index Terms*——About—Nodes,Edges,path;distance,source,graph, visited, IDE www.onlinegdb.com in C++.**

I. Introduction

The breadth-first algorithm is a particular graph-search algorithm that can be applied to solve a variety of problems such as finding all the vertices reachable from a given vertex, finding if an undirected graph is connected, finding (in an unweighted graph) the shortest path from a given vertex to all other vertices, determining if a graph is bipartite, bounding the diameter of an undirected graph, partitioning graphs, and as a subroutine for finding the maximum flow in a flow network (using Ford-Fulkerson's algorithm). As with the other graph searches, BFS can be applied to both directed and undirected graphs.

II. Proposed Methodology

Breadth-First Search is an important kernel used by many graph-processing applications. In many of these emerging applications of BFS, such as analyzing social networks, the input graphs are low-diameter and scale-free. We propose a hybrid approach that is advantageous for low-diameter graphs, which combines a conventional top-down algorithm along with a novel bottom-up algorithm. The bottom-up algorithm can dramatically reduce the number of edges examined, which in turn accelerates the search as a whole. On a multi-socket server, our hybrid approach demonstrates speedups of 3.3 – 7.8 on a range of standard synthetic graphs and speedups of 2.4 – 4.6 on graphs from real social networks when compared to a strong baseline. We also typically double the performance of prior leading shared memory (multicore and GPU) implementations.

III. Here i Discuss BFS Algorithm

1. for each u in V s
2. do color[u] ← WHITE
3. d[u] ← infinity
4. [u] ← NIL
5. color[s] ← GRAY
6. d[s] ← 0
7. [s] ← NIL
8. Q ←
9. ENQUEUE(Q, s)
10 while Q is non-empty
11. do u ← DEQUEUE(Q)
12. for each v adjacent to u
13. do if color[v] ← WHITE
14. then color[v] ← GRAY
15. d[v] ← d[u] + 1
16. [v] ← u
17. ENQUEUE(Q, v)
18. DEQUEUE(Q)
19. color[u] ← BLACK

IV. Conclusion and Future Work

In this paper we evaluated a best-first AND/OR search algorithm which extends the classic AO algorithm and traverses a context-minimal AND/OR search graph for solving the MPE task in belief networks. The algorithm is guided by mini-bucket heuristics which can be either pre-compiled or assembled dynamically during search. The efficiency of the best-first AND/OR search approach compared to the depth-first AND/OR Branch-and-Bound search is demonstrated empirically on various random and real-world benchmarks, including the very challenging ones that arise in the field of genetic linkage analysis. Our approach leaves room for further improvements. The space required by AOBF can be enormous, due to the fact that all the nodes generated by the algorithm have to be saved prior to termination. Therefore, AOBF can be extended to incorporate a memory bounding scheme similar to the one suggested in [

Acknowledgment

I would like to thank my honourable**Khan Md. Hasib Sir** for his time, generosity and critical insights into this project.

References

[1] R. Marinescu and R. Dechter. Best-first and/or search for graphical models. In AAAI, 2007
[2] pM. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

```
#include<bits/stdc++.h>
using namespace std;

#define MX 110

vector < int > graph[MX];
bool vis[MX];
int dist[MX];
int parent[MX];

void bfs(int source){
   queue < int > Q;
   // initialization
   vis[source] = 1;
   dist[source] = 0;
   Q.push(source);

   while(!Q.empty()){
      int node = Q.front();
      Q.pop();
```

Fig. 1. Example of a figure caption.

```
      for (int i = 0; i < graph[node].size(); i++){
        int next = graph[node][i];
        if (vis[next] == 0){
          vis[next] = 1; // visit
          dist[next] = dist[node] + 1; // update
          Q.push(next); // push to queue

          // set parent
          parent[next] = node;
        }
      }
    }
}

/*
   input:
   7 9
   1 2
   1 3
   1 7
   2 3
```

Fig. 2. Example of a figure caption.

```
   3 7
   2 4
   4 5
   3 6
   5 6
   1
*/

// path printing functions

// recursive function
void printPathRecursive(int source, int node){
   if (node == source){
      cout << node << " "; // print from source
      return;
   }
   printPathRecursive(source, parent[node]);
   cout << node << " ";
}

// iterative function
```

Fig. 3. Example of a figure caption.

```
void printPathIterative(int source, int node){
   vector<int> path_vector;

   while(node != source){
      path_vector.push_back(node);
      node = parent[node];
   }
   path_vector.push_back(source); // inserting source

   for (int i = path_vector.size() - 1; i >= 0; i--){
      cout << path_vector[i] << " ";
   }
}

int main()
{
   int nodes, edges;
   cin >> nodes >> edges;

   for (int i = 1; i <= edges; i++){
      int u, v;
```

Fig. 4. Example of a figure caption.

```
    cin >> u >> v;
    graph[u].push_back(v);
    graph[v].push_back(u);
  }

  int source;
  cin >> source;

  bfs(source);

  cout << "From node " << source << endl;
  for (int i = 1; i <= nodes; i++){
    cout << "Distance of " << i << " is : " << dist[i] <<
endl;
  }
  cout << endl;

  // path printing example

  // recursive version
  for (int i = 1; i <= nodes; i++){
```

Fig. 5. Example of a figure caption.

```
        cout << "Path from " << i << " to source: ";
        printPathRecursive(source, i);
        cout << endl;
    }


    return 0;
}
```

Fig. 6. Example of a figure caption.

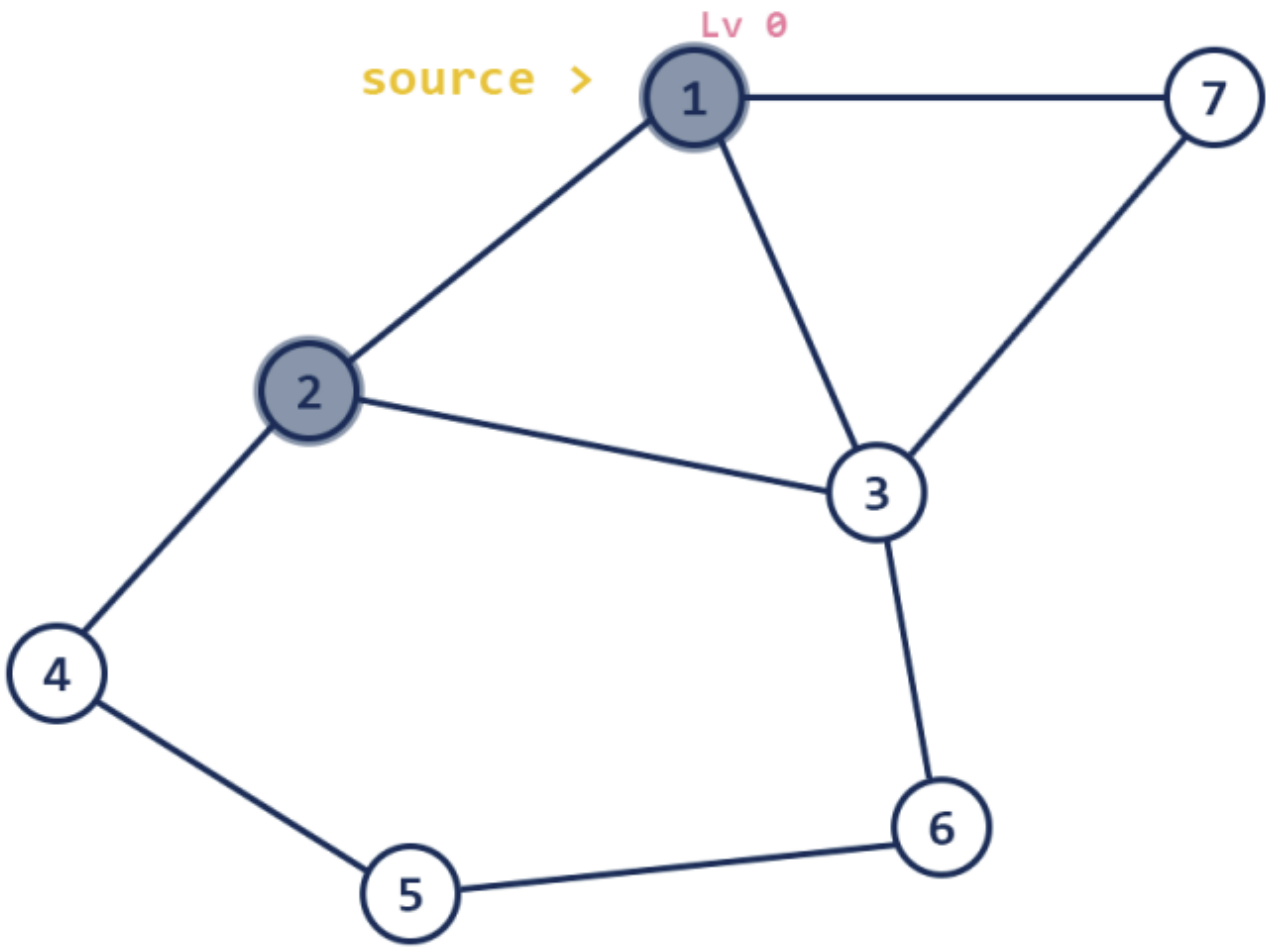

Fig. 7. Example of a figure caption.

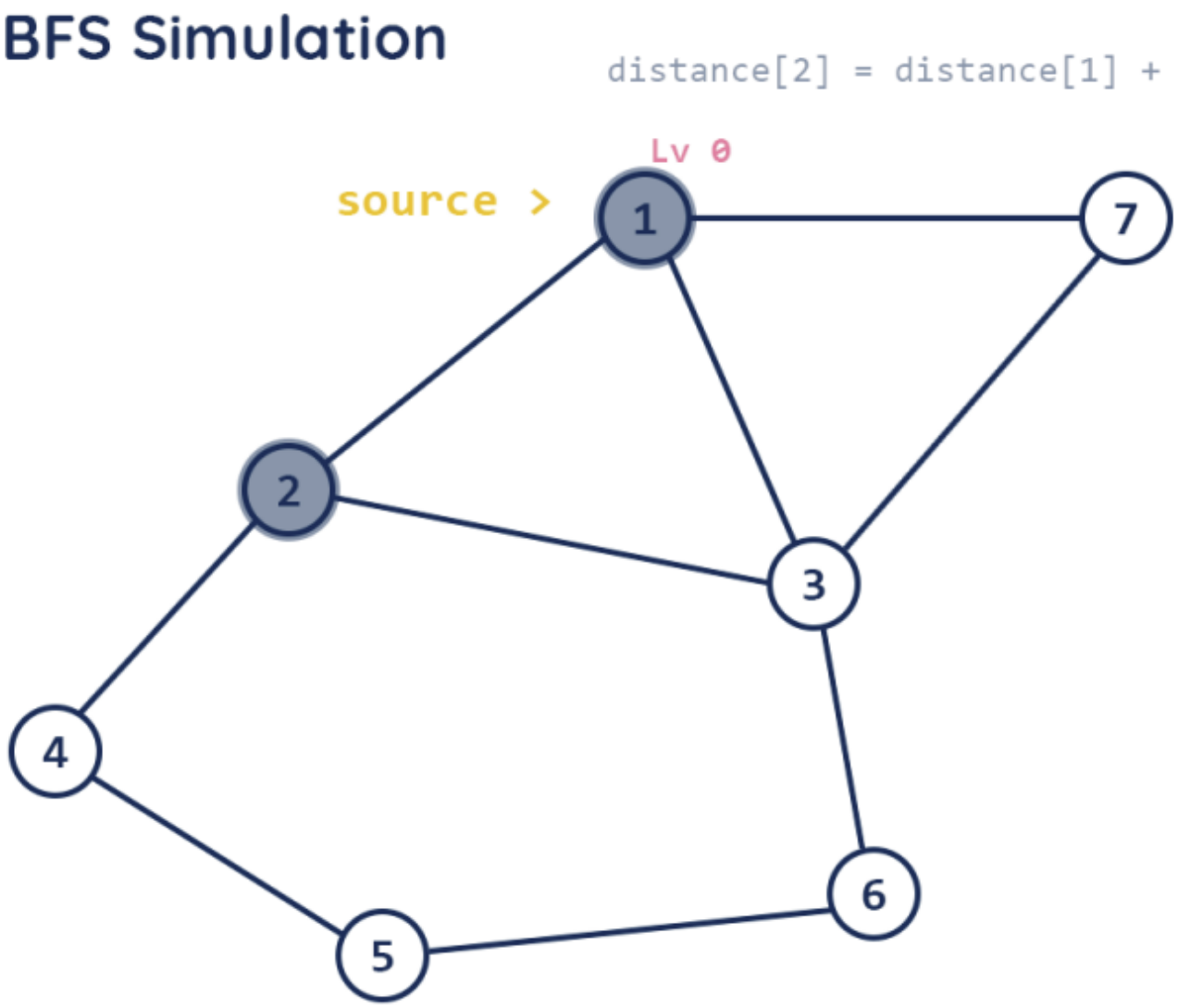

Fig. 8. Example of a figure caption.

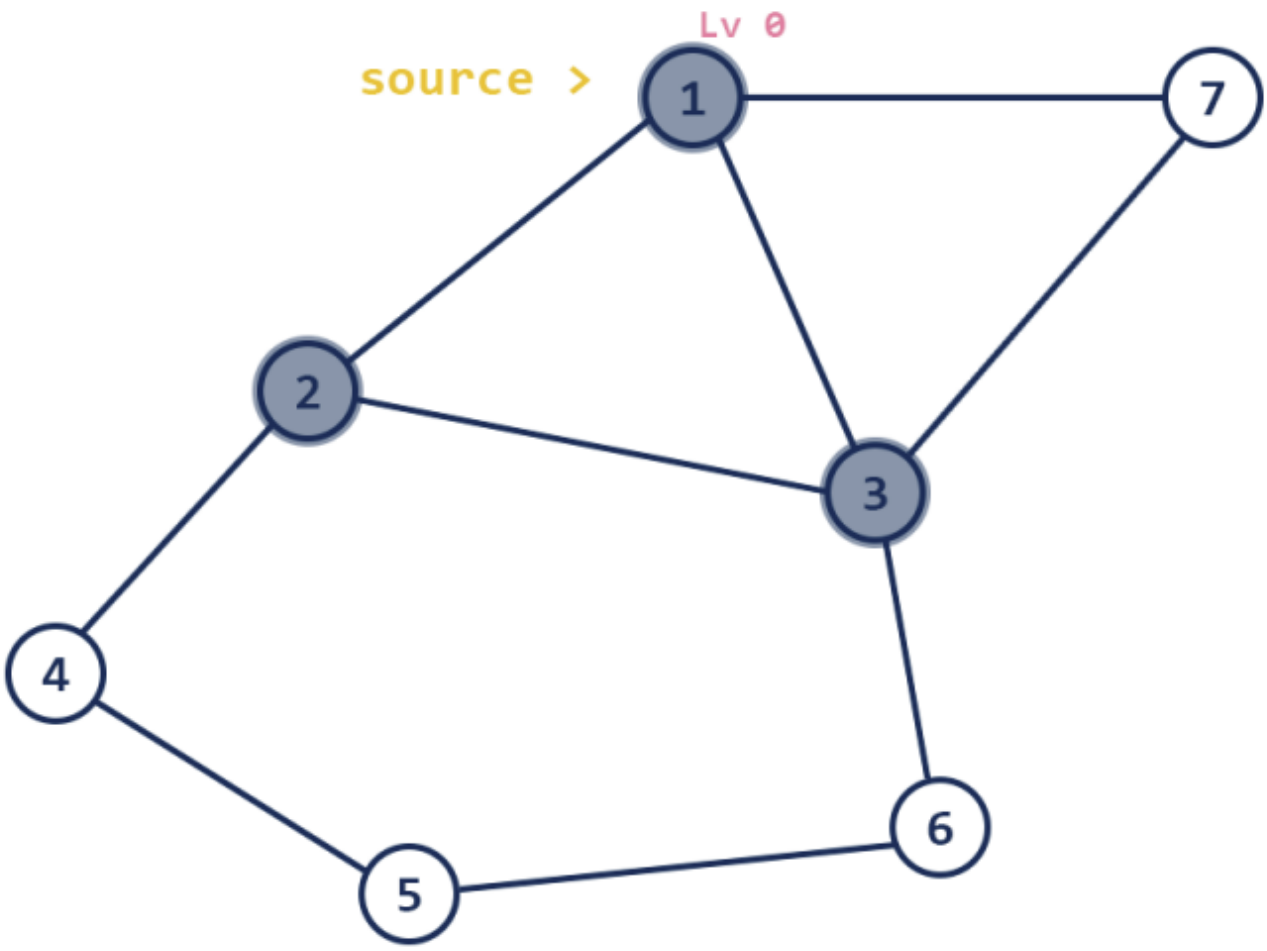

Fig. 9. Example of a figure caption.

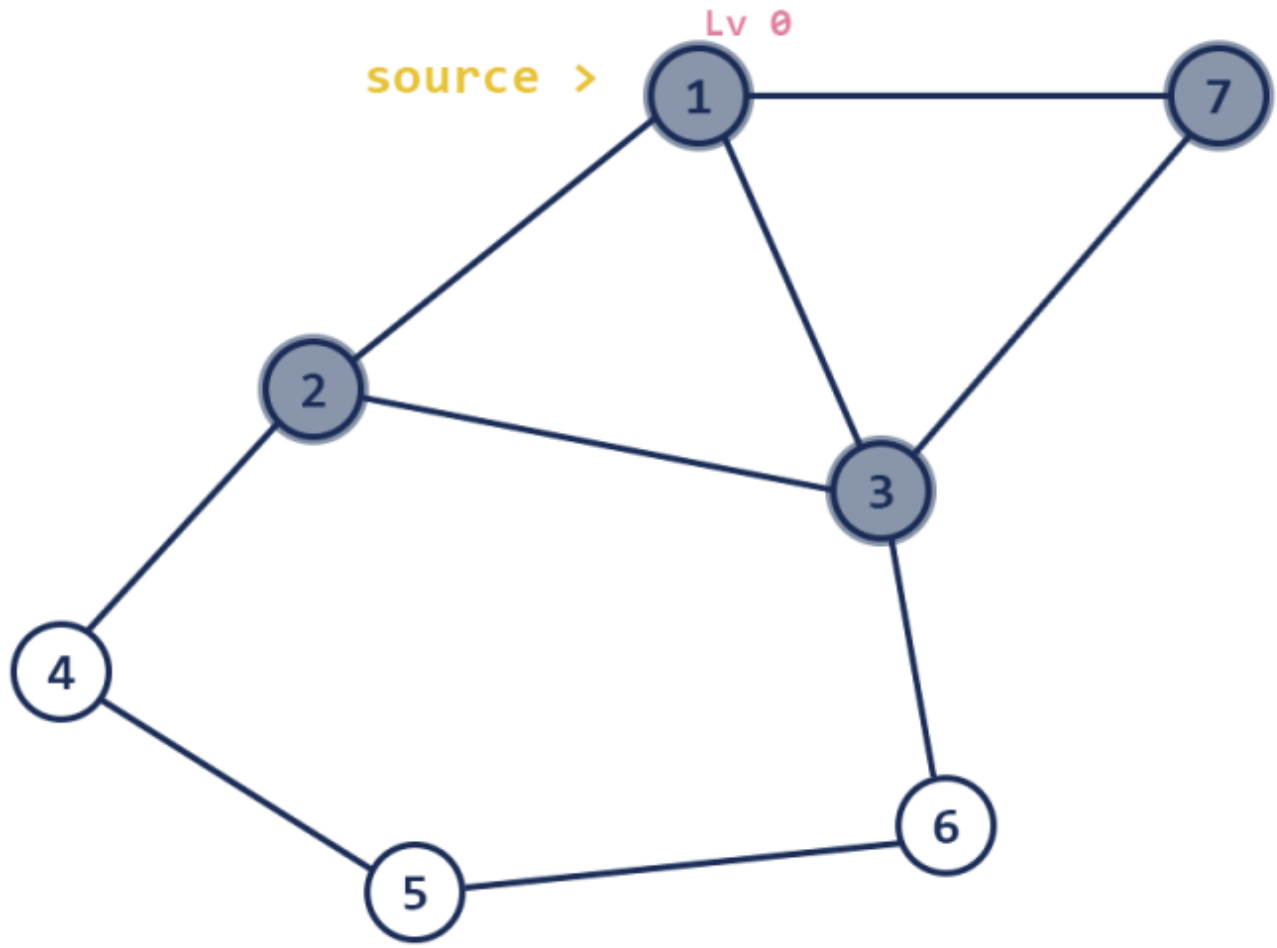

Fig. 10. Example of a figure caption.

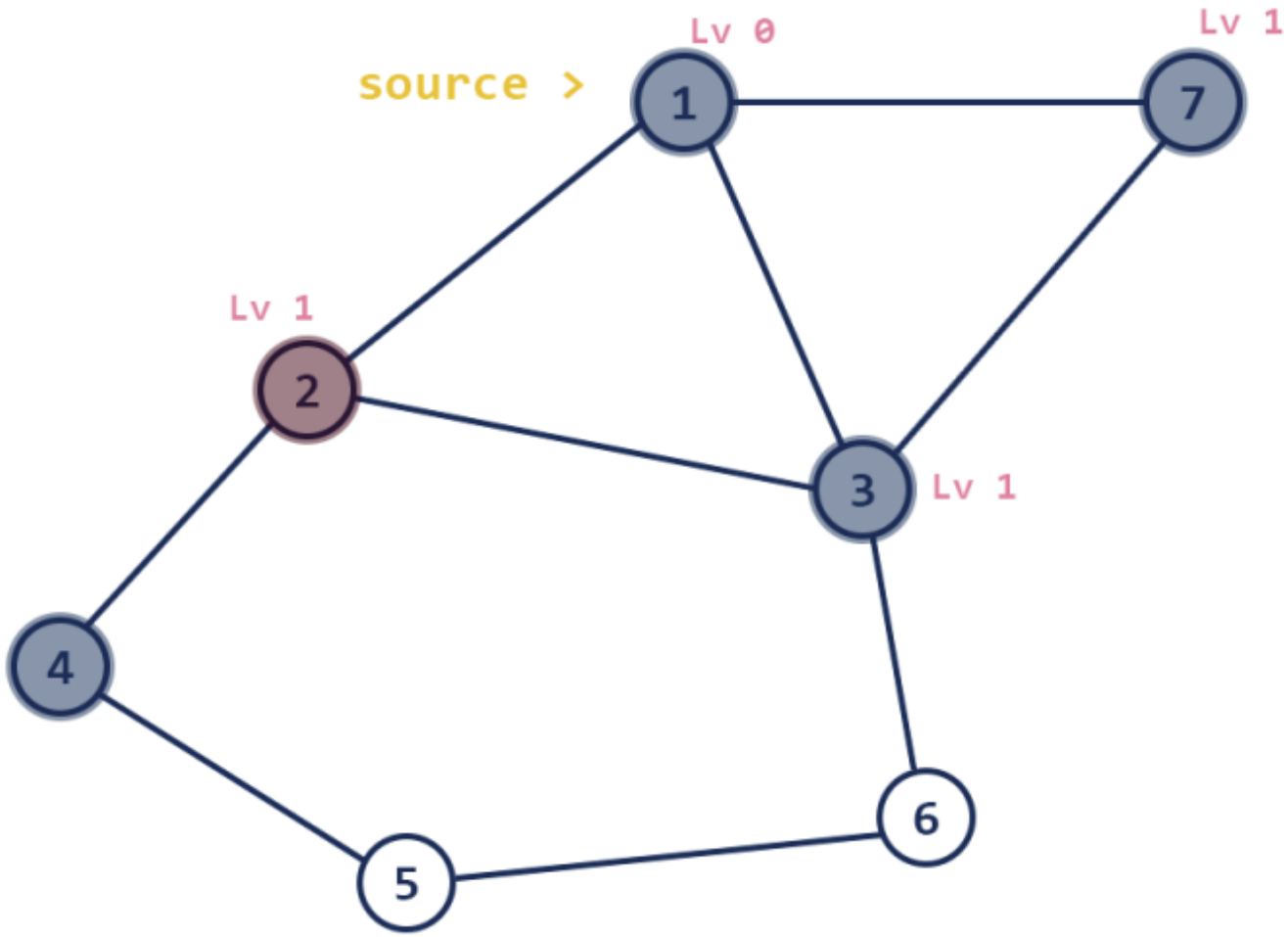

Fig. 11. Example of a figure caption.

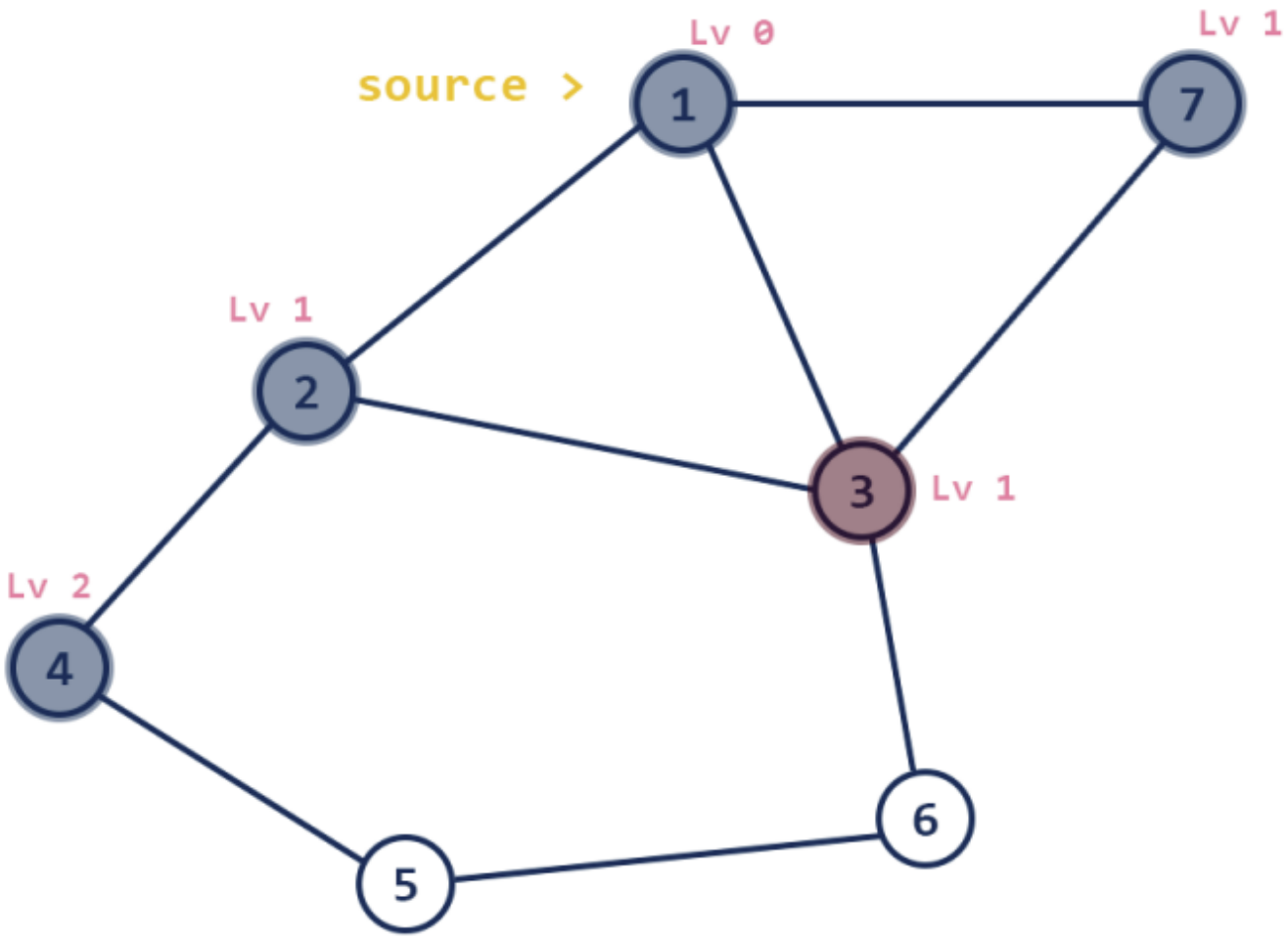

Fig. 12. Example of a figure caption.

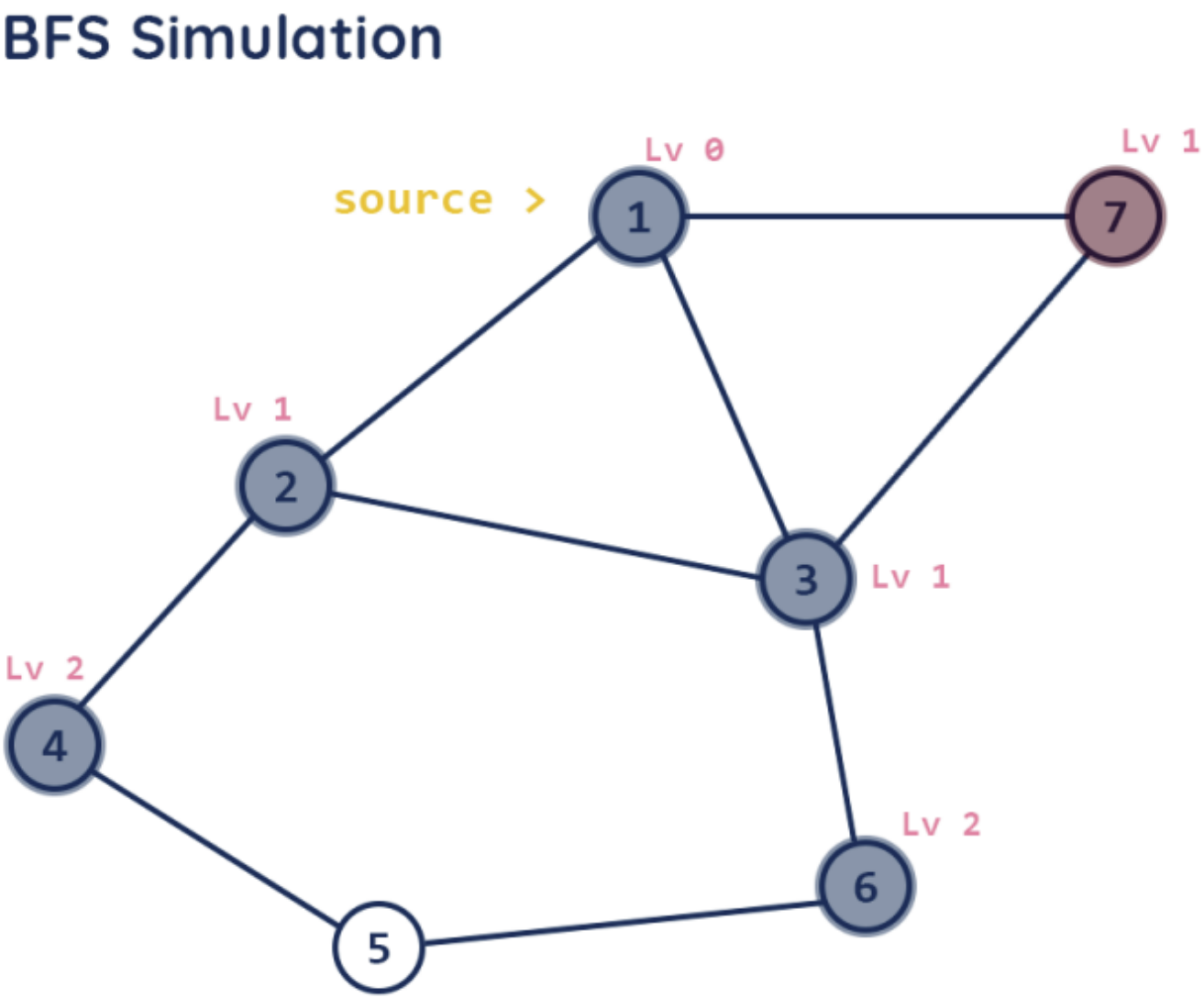

Fig. 13. Example of a figure caption.

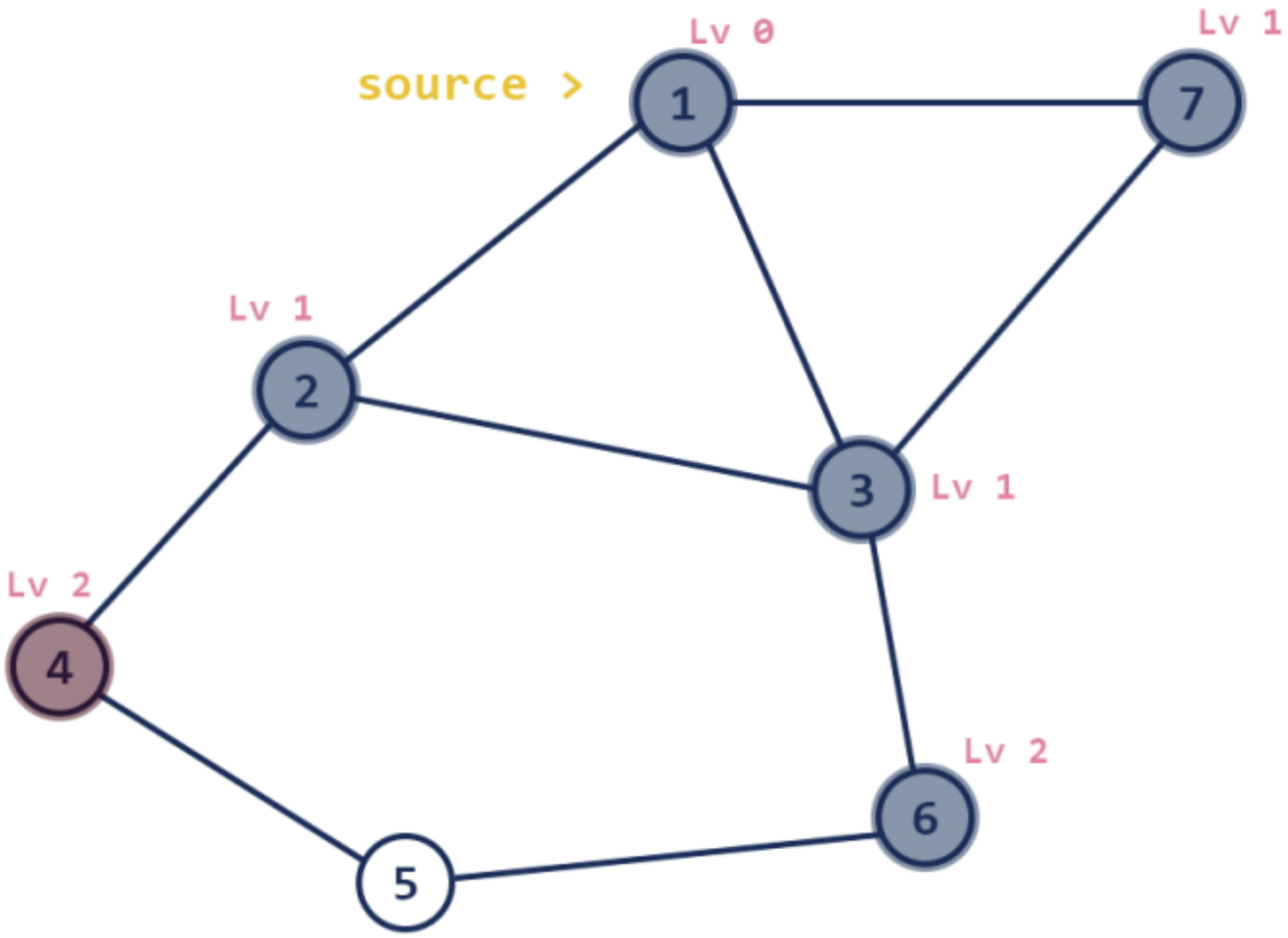

Fig. 14. Example of a figure caption.

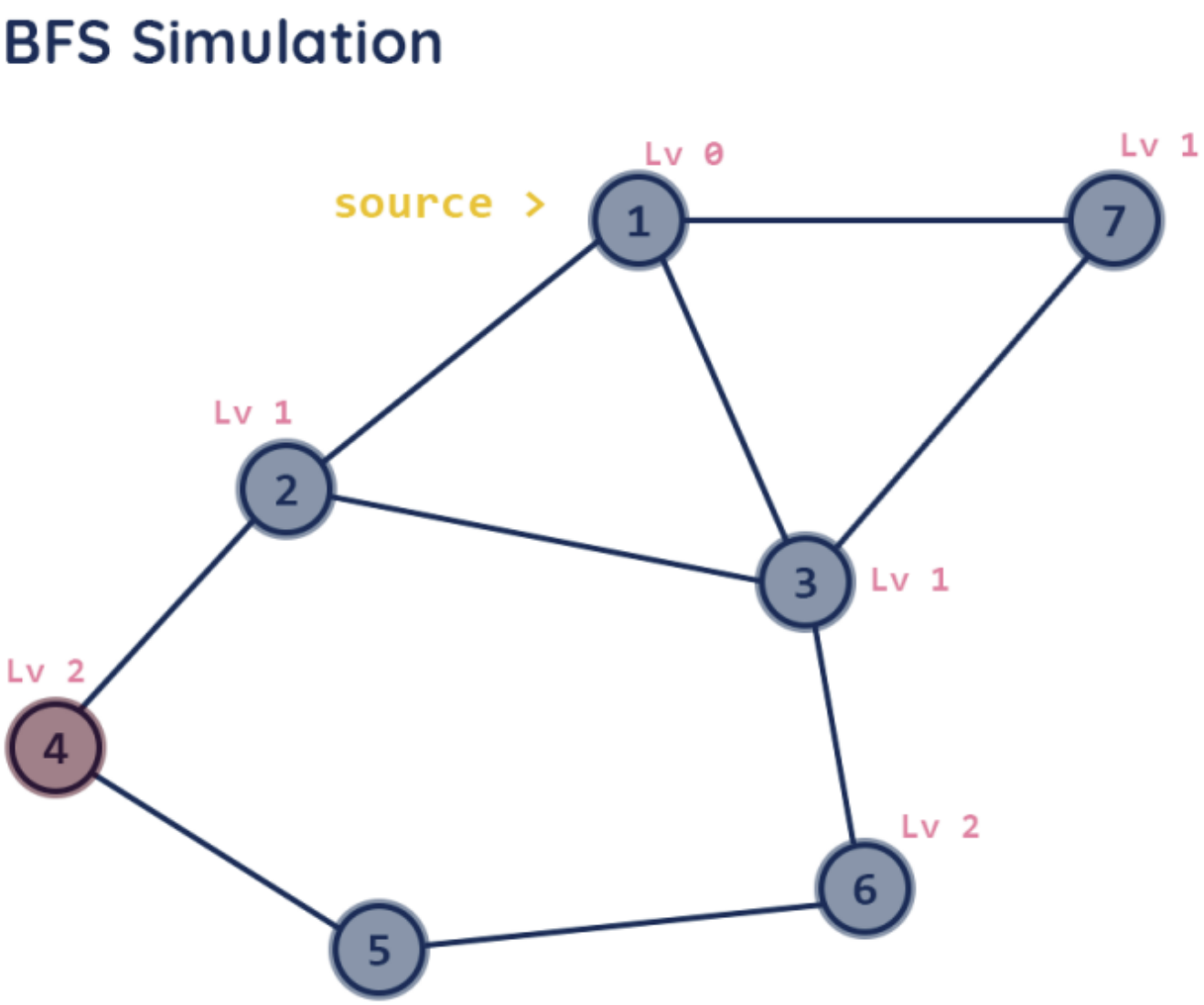

Fig. 15. Example of a figure caption.

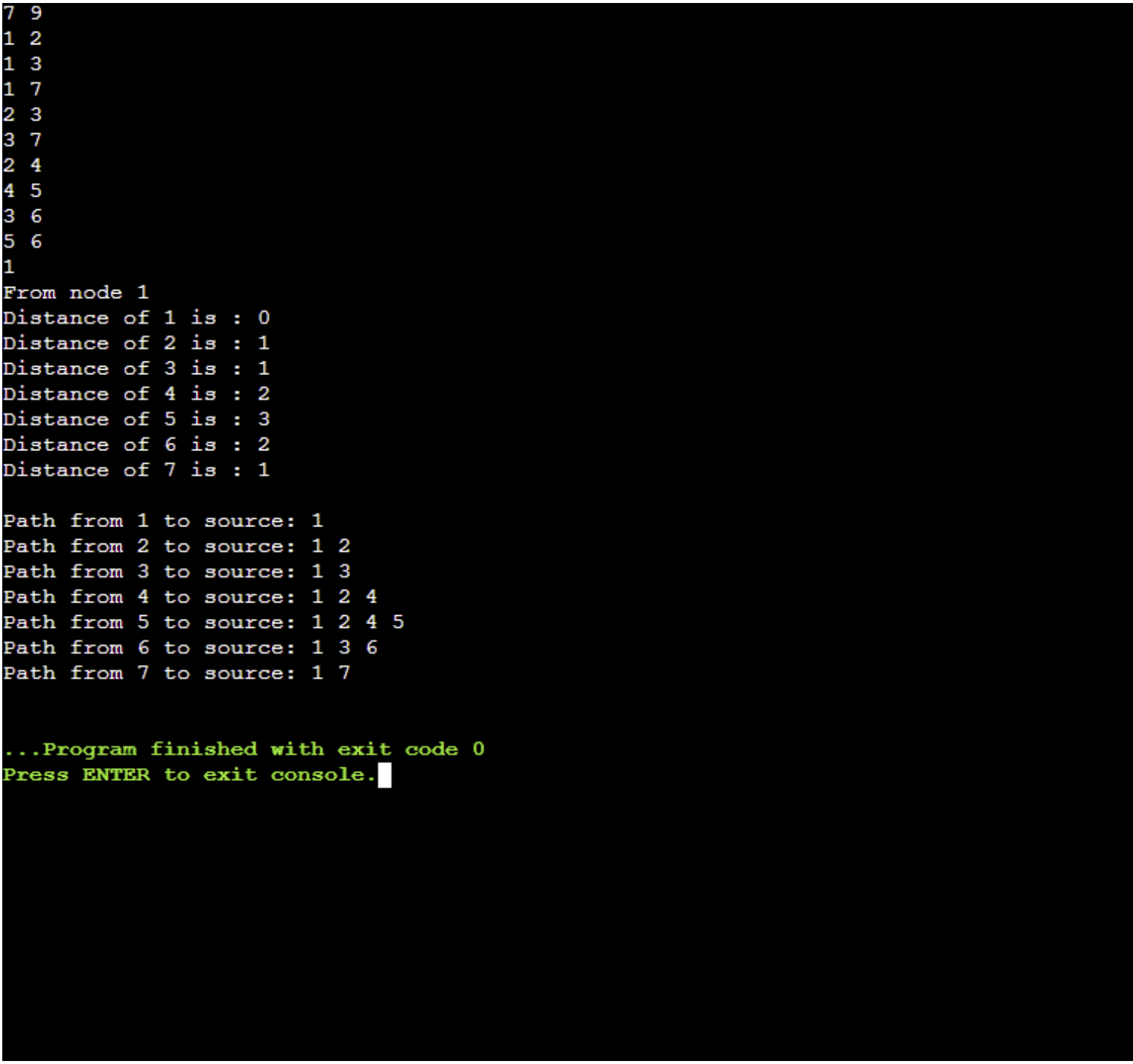

Fig. 16. Example of a figure caption.