(I) What is the drawback of a data structure that stores the distances between every possible pairs of nodes, and how does the implementation from (5) address this problem?

The drawback of the data structure that stores distances between every possible pair of nodes is that is a massive amount of memory, particularly for small robots boards. Most of that data would be void/0/infinity because most nodes aren't connected to other nodes. Our implementation avoids this memory issue by only finding the distance from all nodes to a source node. In terms of mapping that allows the robot to know which path it needs to take to get to a singular destination node. That way the memory isn't scaling by n^2 but rather just n.

(II) Provide code snippets showing your data structure and functions from 4.

```
int xyToIndex(float objx, float objy) {
  int xIndex = ceil(objx / 15);
  int yIndex = ceil(objy / 10);

  nav[xIndex - 1][yIndex - 1] = 0;

  int index = 0;

  for (int i = 1; i < yIndex; i++)
  {
    index = xIndex + 4;
  }

  return index;
}
```

```
struct Coords indexToxy(int index) {
  struct Coords coords;

  if (index > 0 && index < 5) coords.y = 0;
  else if (index > 4 && index < 9) coords.y = 1;
  else if (index > 8 && index < 13) coords.y = 2;
  else coords.y = 3;

  if (index % 4 == 1) coords.x = 0;
  else if (index % 4 == 2) coords.x = 1;
  else if (index % 4 == 3) coords.x = 2;
  else if (index % 4 == 0) coords.x = 3;

  return coords;
}
```

```
int cost(int destIndex, int objIndex) {
  struct Coords destCoords = indexToxy(destIndex);


  // if we have detected an object at that index, explode
  if (nav[destCoords.x][destCoords.y] == 0) return 99;

  // else calc if two nodes are adjacent
  if (objIndex + 1 = destIndex or objIndex - 1 = destIndex or
  objIndex + 4 = destIndex or objIndex - 4 = destIndex)
    return 1;

  return 99;
}
```

We initialize our data structure with all 1's in the setup function indicating that no obstacles have been found yet.

```
for (int i = 0; i < 4; i++) {
  for (int k = 0; k < 4; k++) {
    nav[i][k] = 1; // no obstacles found yet
  }
}
```

We then update it as the following shows in the "xyToIndex" function screenshotted above.

```
nav[xIndex - 1][yIndex - 1] = 0;
```

Cost function simply checks if the next node is an adjacent node and returns 1 if that's the case, and a large number otherwise.