ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown

CSCI 3302
lab3.2

Fall 2017, CU-Boulder

1. **Implement a version of Dijkstra's algorithm on Sparki. Replace UI commands from the example with your data structures. In particular, replace the cost matrix with your distance function. Use a simple 4x4 map as in the last exercise to validate that Dijkstra's returns the correct distances.**

   Explanation:

   Dijkstras algorithm is a way to find the shortest path from the node we are currently at to any other node in our data structure. In this case, we have our data structure defined as coordinates of squares that the robot could be at. Dijkstras starts with a distance array that is an overestimate of the distance to all points (this could be infinity for all nodes, and 0 for the source node). As Dijkstras runs, it checks the adjacent nodes to the source node, and tries to get a better estimate for the distance to them. We replaced the array it uses to calculate these estimates with a more space efficient method. Since our graph is pretty uniform, we were able to use a function to determine the costs on the fly. If the checked nodes are marked as unvisited, it continues to that nodes adjacent nodes as well. It does a few iterations of this and eventually finds the shortest path to all nodes.

2. **Modify the code so that Dijkstra's stores the id of the parent node, that is the node which comes next on the shortest path in a separate data structure.**

   Explanation:

   This algorithm could be modified to also keep track of how we got to each node. A matrix that keeps the parent nodes in mind would be very useful. When dijkstras runs, whenever it updates a shortest path from a node (A) to a node (B) it can store that information as parents[B] = A. This information is almost free to keep track of, and very useful for pathing.

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

3. **Write code that returns a sequence of vertices from a given location on the map to the source.**

   Explanation:

   Once the parent matrix is populated, a simple script could be use to backtrack through it to find the path from the robots current node to any other. This script would look at the parents array entry for the target node, and get its parent. Then repeat with that as a new target. This would result in a backwards path which can then be flipped to get the shortest path to the target node.

   Example:
   Node index definition:

   $$\begin{bmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{bmatrix}$$ ex: (3,2) would equal the index of 6

   A sample map might look like:

   $$\begin{bmatrix} S & 4 & 8 & 12 \\ X & X & X & 13 \\ X & T & 10 & 14 \\ X & X & X & 15 \end{bmatrix}$$

   T represents the target square (index 14)
   X squares are impassable walls
   S represents sparkis location. (starts at index 1)

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

Dijkstras would yield:

$$\begin{bmatrix} 0 & 4 & 8 & 12 \\ 99 & 99 & 99 & 13 \\ 99 & 6 & 10 & 14 \\ 99 & 99 & 99 & 15 \end{bmatrix}$$ 99 squares are walls, and there is no route to these squares.

Parents[6] = 10
Parents[10] = 14
Parents[14] = 13
Parents[13]= 12
Parents[12]= 8
Parents[8]= 4
Parents[4]= 0

0 is the start square, reverse order,

$0 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 13 \rightarrow 14 \rightarrow 10 \rightarrow 6$

Next page is the code:

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

```
#include <Sparki.h>

const int NUMNODES = 16;
const int LARGE_INT = 100;
const int BLOCK_WEIGHT = 50;

int path[16];

//map
int blockedSquares[4][4] = {{1,1,1,1},
                            {0,0,0,1},
                            {0,1,1,1},
                            {0,0,0,1}};

void dijkstra(int startIndex, int goalIndex);
int minNotChecked(int checked[], int arr[], int size);
int isEmpty(int arr[], int size);
int inBounds(int neighbor, int cur);
int pathWeight(int current, int neighbor);
int isPath(int current, int neighbor);

void setup()
{
  sparki.beep();
}

void loop() {
  dijkstra(0,6);
  delay(5000);
}

void dijkstra(int startIndex, int goalIndex)
{
  int cur = startIndex;
  int checked[16] = {0};
  int parents[16] = {0};

  int dist[16];
```

ChenHao Cheng
Adam Siefkas
Charles Davies
Kyle Hartland Brown
Fall 2017, CU-Boulder

```
for (int i = 0; i < NUMNODES; i++)
{
  dist[i] = LARGE_INT;
}

dist[cur] = 0;

while (!isEmpty(checked, NUMNODES))
{

  cur = minNotChecked(checked, dist, NUMNODES);
  checked[cur] = 1;

  int neighbors[4] = {cur-1,
                      cur+4,
                      cur+1,
                      cur-4};

  for (int i = 0; i < 4; i++)
  {
    if (inBounds(neighbors[i], cur) && isPath(cur, neighbors[i]))
    {
      int altDist = dist[cur] + 1;
      if (altDist < dist[neighbors[i]])
      {
        dist[neighbors[i]] = altDist;
        parents[neighbors[i]] = cur;
      }
    }
  }

  int curPath = goalIndex;

  int i = 0;
  while (curPath != startIndex)
  {
    path[i] = curPath;
```

```
      curPath = parents[curPath];
      i++;
    }

    for (int i = 0; i < NUMNODES; i++)
    {
      Serial.println(path[i]);
    }
  }
}

int minNotChecked(int checked[], int arr[], int size)
{
  //ret index of minimum value
  int min = 999;
  int minIndex;

  for (int i = 0; i < size; i++)
  {
    if ((arr[i] < min) && (!checked[i]))
    {
      min = arr[i];
      minIndex = i;
    }
  }

  return minIndex;
}

int isEmpty(int arr[], int size)
{
  //zero in arr is not checked, one is checked
  for (int i = 0; i < size; i++)
  {
    if (arr[i] == 0)
    {
      return 0;
    }
```

```
  }
  return 1;
}

int inBounds(int neighbor, int current)
{
  int currentX = current%4;
  int currentY = current/4;

  int neighborX = neighbor%4;
  int neighborY = neighbor/4;

  //valid coords would be (+1,0), (-1,0), (0,+1), (0,-1)
  int diffX = abs(neighborX-currentX);
  int diffY = abs(neighborY-currentY);

  if ((neighbor > 15) || (neighbor < 0))
  {
    return 0;
  }

  if ((diffX == 1) && (diffY == 0))
  {
    return 1;
  }

  else if ((diffX == 0) && (diffY == 1))
  {
    return 1;
  }

  return 0;
}

int isPath(int current, int neighbor)
{
  int currentX = current%4;
  int currentY = current/4;
```

```
    int neighborX = neighbor%4;
    int neighborY = neighbor/4;

    if ((blockedSquares[currentX][currentY] == 0) ||
    (blockedSquares[neighborX][neighborY] == 0))
    {
      return 0;
    }

    return 1;

  }
```

Output: