Splash 2014

C8730: Find the Shortest Path

Varot Premtoon | varot@mit.edu

Introduction and Problems

- Shortest path algorithms aim at finding a path from one state to another, using the path that has the minimal 'cost.'
- A wide variety of applications.
    - Mazes
    - Transportation/logistics optimization on roads, rails, air, etc., and combinations of them.
    - Less obvious ones such as solving Rubik's Cube
- Topics today
    - Problem Representation
    - BFS: unit cost graph
    - Solving a maze
    - Dijkstra: positive cost graph
    - Solving a small map
    - Solving a weighted maze

Problem Representation

- Generic Representation: Graph (we will work with it for the rest of the class.)
    - $G(V, E)$
    - $V =$ set of nodes or vertices.
        - We can number them from $0 \dots n-1$, for n being the number of nodes.
        - Represented by an array of n items.
    - E = set of edges that connect vertices.
        - $e(v, w)$ represent an edge that points from node v to w.
        - Directional $e(v, w) \neq e(w, v)$ or
          Undirectional $e(v, w) = e(w, v)$.

    - Weighted $e(v, w) = c$ or
      Unweighted $e(v, w) = True/False$.
    - Represented by an $n \times n$ 2-dimentional array.

Breath First Search

- Note: the idea in **bold** means that you might want to think a bit harder on how to translate it into actual codes.
- Perform search level by level. (level = distance from the root.)
- As oppose to Depth First Search, which picks a branch and dives into it.
- Main Idea:
  1. The initial distances of all nodes are INF.
  2. Pick a root node. Set distance to be 0.
  3. **For each node adjacent the root**, **set the distance** to be 1.
  4. **Loop through** all the nodes of distance 1.
  5. For each node adjacent each 1-node, set the distance to be 2, but only if the distance of that node has not been set.
  6. Loop through all the 2-node. Do the same thing to label 3-nodes.
  7. Repeat until **all the nodes have their distances calculated**.
  8. Note that each item in the distance table needs to be set only once. For example, if BFS has determined that a node has distance of 2, that is the correct distance for that node and will not change.
- Observe how the algorithm proceeds level by level.
- Problem: what is a nice way to process the nodes by the order of their distance?
  - When trying to label d-nodes, just look at the entire graph and see which nodes have distance $(d - 1)$. But you will have to look at the entire graph for *each* distance, which is inefficient.
  - Solution: use **queue**. Basically, after you set the distance of a node, put that node in the queue. Then process the nodes in the queue order.
    - The root node is first in the queue.

        - Then, when you set the distances of the adjacent nodes (nodes of distance 1), you **put them in the queue**, too.
        - When you are done with the root node, you **move on** to the node next in line, which will be one of the 1-nodes.
        - From that 1-node, you will append to the queue all the **unvisited** nodes adjacent to it, which will be 2-nodes. Those 2-nodes will stand behind all the 1-nodes.
        - When you are done with that 1-node, move on to the node next in line, which might be another 1-node.
        - Once all the 1-nodes are processed, the 2-nodes will come up. The 3-nodes will be appended to the queue. So on and so forth.

Maze Representation

- V = set of cells.
    - We can number (index) each cell (r, c) in an $R \times C$ maze as follows:
        - The cell (0, 0) (top left) has index 0.
        - The next cell to the right (0, 1) has index 1.
        - Keep doing this from left to right, and then move to the next row.
        - Formula: cell (r, c) has index $ind = r \times C + c$
        - If we have the index, we can retrieve the coordinate (r, c):
            - $r = ind \,/\, C$
            - $c = ind \,\%\, C$
- E = set of borders between cells.
    - $e(ind1, ind2) = e(ind2, ind1) = 1$ if the wall between $ind1$ and $ind2$ is open,
    - 0 otherwise
    - You may find it inefficient. We will talk about this later.

Dijkstra Algorithm

- The problem now is that simple BFS won't work with weighted graph.
- Dijkstra is a modified-BFS.
- The main idea: establish the region where we know our answers are correct. We will call it **The Cloud.** It will be very small at first (like a region containing 1 node), and then you expand it systematically until the covers the entire graph.
- Main idea in more details:
    1. The initial distances of all nodes are INF.
    2. Pick a root node. Set distance to be 0.
    3. Initialize **the cloud** that has only one node: the root. (because we know that the root, and only the root, has the correct distance.)
    4. For each node adjacent to the root, **update** their distance table.
    5. Of all the nodes **outside of the cloud, pick one that has the minimal distance**.
    6. We can be sure that that node, *and only that node,* has the correct distance. There is no shorter way to get to that node.
    7. Because of that, we include it to the cloud.
    8. For each node adjacent to that node, update the distance.
    9. Again, pick the minimal distance node and keep doing the same procedure **until the cloud covers the entire graph** .
    10. * Note that, unlike BFS, this might involve updating the distance that has already been update once. For example, if Dijkstra has determined that a node has distance of 15, it IS possible that their might be a shorter way to that node discovered in the future.

Road Network Representation

- V = set of cities/junctions.
  - You may want to number your city from 0…n-1 for easy use.
- E = set of roads.
  - $e(ind1, ind2)$ is cost of travelling from $ind1$ to $ind2$. Could be distance, time, toll, etc.
  - 0 if there is no such road.

Weighted Maze Representation

- V = exactly the same as unweighted maze.
- E = set of borders between cells.
  - $e(ind1, ind2) = weight(ind2); e(ind2, ind1) = weight(ind1)$ if the wall between $ind1$ and $ind2$ is open
  - 0 otherwise

Limitation in the representation

- Takes lots of memory ($n \times n$ for edges). It is unlikely in many application that every node will have edges to all other nodes.
  - Solution: For each node, use an expandable list to store edges pointing from that node.
- Each pair of nodes can only have one edge.
  - Solution: same as above.

Limitation of BFS

- Allowed only finite/practically finite set of nodes
- Cannot find optimum cost shortest path

Limitation of Dijktra

- Allowed only finite/practically finite set of nodes.
- It explores all the nodes. Could be unnecessary.

- Doesn’t work with negative costs.
- Much slower than it could be
    - Solution: use the expandable edge lists + a more efficient algorithm to select the next node to update → **heap**~!!!

Conclusion

- Shortest path is a sub problem of state space search.
- A* star.
- Floyd-Warshall.