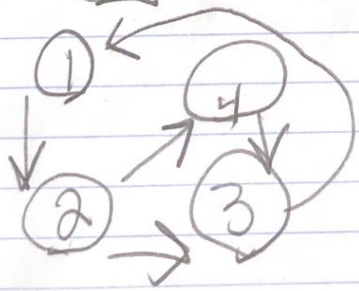
DESIGN

PURPOSE Assignment 4 demands that a user gives an input of a number of vertices and corresponding edges between them. From there, the program will iterate recursively through each and every possible path to ultimately return the shortest path that starts & ends at the origin (touching all other vertices).

Layout/Structure

DFS() This is a function that takes input graph, current path, shortest path, cities, a vertex, and an out file. We begin by marking the vertex as visited and iterate through all possible paths from that vertex. If we find an adjacent vertex that we can connect to, we progress and push that next vertex and call DFS on that vertex to calculate all possible paths from there. If no possible paths exist, we pop the vertex from the Path and try other possibilities.

Examine example use of DFS() on page 2 w/ a graph...



Total = 4 pv ② → ~~③~~ → ~~④~~ → ~~②~~ 12

① is visited

②Push = ~~False~~ True
① ~~cannot be~~ is already visited
② ~~can~~ is a ~~connection to ①~~ not visited
② is pushed Pysh = True
② is a ~~connection~~

---

② is visited Push = ~~False~~ True
① is already ② is already
③ is not visited ② ~~is pushed~~ Push = True

---

③ is visited Push = False
① ~~is also~~ ②, ③ already...
④ is not visited ④ is ~~not a~~ cannot be pushed
③ is unvisited
③ is popped

---

④ is not visited ④ is pushed Push = True

---

④ is visited Push = ~~False~~ True
① ② already...
③ is not visited ③ is pushed Push = True

---

③ is visited Push = False
① ② ③, ④ already
Hamiltnian Path!
② → ④ → ③ = 4 = 1
~~Long+ur~~
End! ③ unvisited ③ popped
End! ④ unvisited ④ popped
End! ② is unvisited ② popped

3

We know we found a hamiltonian path when the length of our path is equal to the (total vertices - 1) bc we are not pushing the origin.

Note: If verbose printing is enabled we print all hamiltonians. If not we copy the ~~shortest~~ shortest one into shortestPath.

PathCT · A structure w/ length and a stack "vertices".

· We use it to track the current trail of "footsteps"

· We push only when there is a valid edge connecting the last point and the ~~value~~ point being pushed. If first push then the value must have an edge w/ the origin ~~0~~ zero.

· Popping a vertex places popped value in specified variable

Stack · Used by path to hold all vertices
· Similar to asgn3 Stack except stack_peek which returns top value without tampering w/ stack.

Graph • Structure w/ attributes vertices, undirected, visited, matrix

- vertices is amount of points
- undirected is a boolean deciding if we mirror edges or not
- visited is ~~an~~ array that tracks where we visit
- Matrix is a 2D Array mapping edges to vertices.

- We add edge by setting value of specified coords. to specified value