Review

Topological Sorting of a DAG

Depth-First Search

Breadth-First Search

Algorithm DFS(G, v)
setLabel(v, VISITED)
for all e ∈ G.incidentEdges(v)
if getLabel(e) = UNEXPLORED
w ← opposite(v,e)
if getLabel(w) = UNEXPLORED
setLabel(e, DISCOVERY)
DFS(G, w)
else
setLabel(e, BACK)
Breadth-First Search

Algorithm BFS(G, s)
Q ← new queue
Q.enqueue(s)
while ¬Q.isEmpty
v ← Q.dequeue()
for all e ∈ G.incidentEdges(v)
if getLabel(e) = UNEXPLORED
w ← opposite(v,e)
if getLabel(w) = UNEXPLORED
setLabel(e, DISCOVERY)
setLabel(w, VISITED)
Q.enqueue(w)
else
setLabel(e, CROSS)

Shortest Paths

Minimum Hopping Flight

Minimum Hopping Flight

Minimum Hopping Flight

Minimum Hopping Flight
Algorithm ShortestPath \((G, s)\)

for all \(v \in G.\)vertices()
    if \(v = s\)
        \(s.\)distance ← 0
    else \(s.\)distance ← ∞
    \(v.\)parent ← null

\(Q \leftarrow \) new queue
\(Q.\)enQueue\((s)\)
while \(!Q.\)isEmpty()
    \(v \leftarrow Q.\)deQueue()
    for all \(e \in G.\)incidentEdges\((v)\)
        \(w \leftarrow \) opposite\((v,e)\)
        if \(w.\)distance = ∞
            \(w.\)distance ← \(v.\)distance + 1
        \(w.\)parent ← \(v\)
        \(Q.\)enQueue\((w)\)

Weighted Graphs
- In a weighted graph, each edge has an associated numerical value, called the weight of the edge.
- Edge weights may represent distances, costs, etc.
- Example: In a flight route graph, the weight of an edge represents the distance in miles between the endpoint airports.

Shortest Path Problem
- Given a weighted graph and two vertices \(u\) and \(v\), we want to find a path of minimum total weight between \(u\) and \(v\).
  - Length of a path is the sum of the weights of its edges.
- Example:
  - Shortest path between Orlando and San Francisco

BFS strategy may not work!!

Dijkstra’s Algorithm
for Single source Shortest Path:
- i.e. shortest path between one vertex and all other vertices.
- Extension of Breadth First Search
- uses a Greedy algorithm.
Dijkstra's Algorithm

HNL 2555

LAX 1233

DFW 1120

MCO

Dijkstra's Algorithm

HNL 2555

LAX 1233

DFW 1120

MCO

Dijkstra's Algorithm

HNL 2555

LAX 1233

DFW 1120

MCO

Dijkstra's Algorithm

HNL 2555

LAX 1233

DFW 1120

MCO

Dijkstra's Algorithm

HNL 2555

LAX 1233

DFW 1120

MCO

Dijkstra's Algorithm

HNL 2555

LAX 1233

DFW 1120

MCO
Dijkstra's Algorithm

• How to find shortest path and shortest path length?
Dijkstra’s Algorithm

Dijkstra’s Algorithm

Dijkstra’s Algorithm

Dijkstra’s Algorithm

Analysis

- Graph operations
  - Method incidentEdges is called once for each vertex
- Label operations
  - We set/get the distance and locator labels of vertex \( v \) \( \Theta(1) \) times
  - Setting/getting a label takes \( \Theta(1) \) time
- Priority queue operations
  - Each vertex is inserted once into and removed once from the priority queue, where each insertion or removal takes \( \Theta(\log n) \) time
  - The key of a vertex in the priority queue is modified at most \( \deg(v) \) times, where each key change takes \( \Theta(\log n) \) time
- Dijkstra’s algorithm runs in \( \Theta((n + m) \log n) \) time provided the graph is represented by the adjacency list structure
  - Recall that \( \sum_{v} \deg(v) = 2m \)
- The running time can also be expressed as \( \Theta(m \log n) \) since the graph is connected

Shortest Path Properties

Property 1: A subpath of a shortest path is itself a shortest path

Property 2: There is a tree of shortest paths from a start vertex to all the other vertices.

Example: Tree of shortest paths from Orlando
Single Source Shortest Path

Property 1: A subpath of a shortest path is itself a shortest path. **Greedy Algorithm**

Property 2: There is a tree of shortest paths from a start vertex to all the other vertices. **Single source shortest path.**

Example: Tree of shortest paths from Orlando

Why It Doesn’t Work for Negative-Weight Edges

- Dijkstra’s algorithm is based on the greedy method. It adds vertices by increasing distance.

- If a node with a negative incident edge were to be added late to the vertex list for which decisions have been made, it could mess up distances for vertices already in the list.