Graphs IV

Wednesday, March 28, 2018 2:06 PM

Single-source shortest paths

Weighted graphs

- Weight = distance, $w(e)\colon E \to \mathbb{R}$

Shortest Paths

- Between two nodes
- BFS = minimum number of hops, not distance
- Triangle inequality
  - Maintain estimate for each distance
  - Reduce estimate
  - Invariant points: estimate >= distance

```
relax(int u, int v){
    if (dist[v] > dist[u] + weight(u,v))
      dist[v] = dist[u] + weight(u,v);
}
```

Bellman-Ford Algorithm

- Works by proof of induction
- O(VE)

```
n = V.length
for i = 1 to n-1
  for Edge e in Graph
     relax(e)
```

- Can only terminate early when an entire sequence of |E| relax operations have no effect (no faster way to get to any node)

Negative weight cycle

- After V-1 iterations, should be done
- If the Vth iteration changes the estimate, then there is a negative weight cycle
  - Infinitely negative to follow this cycle
  - Bellman-Ford does not work

Same Weights

- Use BFS

Condition	Algorithm	Time Complexity
No Negative Weight Cycles	Bellman-Ford Algorithm	$O(VE)$
On Unweighted Graph (or equal weights)	BFS	$O(V + E)$
No Negative Weights	Dijkstra's Algorithm	$O((V + E) \log V)$
On Tree	BFS / DFS	$O(V)$
On DAG	Dynamic Programming	$O(V + E)$

Searching for Maximum with a Stack

- Add additional data structures to record more information

Using a Heap

- Push = add to heap O(log n)
- Pop = remove from heap (have to use indirect heap, hash table) = O(log n)
    - Swap with last item, bubble last item down (to maintain completeness)
- Find max = O(1)
- Additional O(n) space complexity to store heap

maxVal Variable

- Use a variable to keep track of the maximum value
- Push = update max = O(1)
- Pop = search again for max, O(n)

maxStack

- Push = check for max and push onto max stack
- Pop = pop stack and maxStack
- For 2nd max: the stack holds both max and 2nd max (object)
- Additional O(n) space