CS101-Quiz9-Review

Suting Chen

CS101-Quiz9-Review

Key Points

1. Dijkstra’s algorithm
2. Bellman-Ford algorithm
3. A* Search Algorithm

Dijkstra's algorithm

1. Solves single-source shortest path problem.

2. Choose the vertex "nearest" to the source each time.

3. Does not work for graphs with negative weight edges.

4. Time complexity of Dijkstra's algorithm is $O\left((V + E)\log V\right)$

Dijkstra’s algorithm

Algorithm analysis

1. We need to initialize the distance array and the binary heap with all **vertices**.
2. Each **edge** is processed once when relaxing the distances.
3. **Binary heap** is used to find the vertex with the minimum distance.

Dijkstra’s algorithm

Algorithm analysis — Time

1. Inserting all the vertices into the binary heap: $O\left(V \log V\right)$

2. Every edge visit could update distances in binary heap: $O\left(E \log V\right)$

3. Total time complexity: $O\left((V + E) \log V\right)$

Dijkstra's algorithm

Algorithm analysis — Better **time** complexity with fib heap

1. Inserting all the vertices into the fib heap: $O\left(V\log V\right)$

2. Every edge visit could update distances in fib heap: $O\left(E\right)$

3. Total time complexity: $O\left(V\log V+E\right)$

CS101-Quiz9-Review

Key Points

1. Dijkstra's algorithm
2. **Bellman-Ford algorithm**
3. A* Search Algorithm

Bellman-Ford algorithm

1. Solves single-source shortest path problem.
2. Update cost of all vertices in each iteration.
3. Work for graphs with negative weight edges.
4. Time complexity is $O(VE)$

Bellman-Ford algorithm

Algorithm analysis — Time

1. In each iteration, the algorithm goes through all the edges in the graph: $O(E)$

2. Relaxation process is run for $O(V)$ times.

3. Total time complexity: $O(VE)$

Comparison

	Dijkstra's	Bellman-Ford
Time complexity	$O\left((V+E)\log V\right)$ $O\left(V\log V+E\right)$	$O\left(VE\right)$
Negative weights	✗	✓
Negative cycle	✗	**Detect**

CS101-Quiz9-Review

Key Points

1. Dijkstra's algorithm

2. Bellman-Ford algorithm

3. **A* Search Algorithm**

A* Search Algorithm

1. Heuristic-based Pathfinding algorithm

2. Admissible and consistent

3. We don’t care the time complexity

A* Search Algorithm

Admissible and consistent

1. Admissible: never overestimates.

2. Consistent: triangle inequality.

3. Consistency implies admissibility (See post @173)

A* Search Algorithm

Tree search and Graph search

1. Tree search is the algorithm that **believes** it is running on a tree, so it will not check whether a vertex is visited

2. By using tree search:

   2.1. You could fall into an infinite loop

   2.2. "Scores" or "Costs" of vertices can be updated multiple times

3. By using graph search:

   3.1. Your algorithm will terminate

   3.2. Avoid visiting the same vertices a second time

A* Search Algorithm

Tree search and Graph search

	None	Admissible	Consistent
Graph	✗	✗	✓
Tree	✗	✓	✓