Shortest Paths and Dijkstra’s Algorithm

15-295 Competition Programming and Problem Solving

Dijkstra’s Algorithm

- Single-source shortest path (SSSP) problem: Given an edge-weighted graph and a source vertex s, find shortest paths to each other vertex from s
- Framework for SSSP:
  - Maintain “distance estimates” = the length of the shortest path found so far
  - Improve distance estimates by “relaxing” outgoing edges, i.e. if

    $$\operatorname{dist}(v) > \operatorname{dist}(u) + w(u, v)$$

    Then update dist(v).
- Bellman-Ford algorithm: Relax every edge n times. Complexity: $O(nm)$
- Dijkstra’s Algorithm:
  - If all edge weights are non-negative, each edge only needs to be relaxed once
  - Relax edges in order of the distance from u
  - Using an efficient priority queue, this can be done fast!
  - Complexity: $O(m \log(n))$

Dijkstra's Algorithm (In textbooks)

function DIJKSTRA($G = (V, E)$, s)
  $dist[1..n] = \infty$
  $pred[1..n] = 0$
  $dist[s] = 0$
  Q= priority_queue($V[1..n]$, key(v) = $dist[v]$)
  while Q **is not empty do**
    u = Q.pop_min()
    for each edge e that is adjacent to u **do**
      // Priority queue keys must be updated if relax improves a distance estimate!
      RELAX(e)
  return $dist[1..n]$, $pred[1..n]$

function RELAX($e = (u, v)$)
  if $dist[v] > dist[u] + w(u, v)$ **then**
    $dist[v] = dist[u] + w(u, v)$
    $pred[v] = u$

Need an efficient algorithm to update the key (distance) of a vertex in the priority queue every time a distance estimate is improved.

Dijkstra's Algorithm (In programming competitions)

- Instead of keeping a unique entry for every vertex in the priority queue, and updating the keys as the distances change, allow duplicate entries!
- Then ignore out-of-date entries whenever they are popped from the queue

```
function DIJKSTRA(G = (V, E), s)
    dist[1..n] = ∞
    pred[1..n] = 0
    dist[s] = 0
    Q = priority_queue()
    Q.push(s, key = 0)
    while Q is not empty do
        u, key = Q.pop_min()
        if dist[u] == key then
            for each edge e that is adjacent to u do
                if dist[v] > dist[u] + w(u, v) then
                    dist[v] = dist[u] + w(u, v)
                    pred[v] = u
                    Q.push(v, key = dist[v])
    return dist[1..n], pred[1..n]
```

Ignore duplicate (out-of-date) entries!