Shortest Path Algorithms

CS483

Jana Kosecka
Department of Computer Science
ST II, 417, 3-1876
Variants:

Single shortest path – single destination t (single source)
Given pair of vertices – what is the shortest path from u to v
All pairs shortest path

BFS – shortest path when all edges were labeled 1, now we have costs

Example:
Optimal Substructure

Cost of the path – sum of the weight of all edges

**Observation 1:** Shortest path has a optimal substructure problem (we will see greedy and dynamic programming techniques)

If $p$ is the shortest path from $v_1$ to $v_k$, $v_1$, $v_2$, ... $v_i$, .... $v_j$, ... $v_k$
The for any $i$, $j$ the shortest path from $v_i$ to $v_j$ is contained in it.

**Proof:** if some subpath is not a shortest path then we can substitute it and obtain shorter path then the original (contradiction)

Notion of the shortest path is well defined only when there are No negative weight cycles
Graphs with negative weights

One path from $s$ to $a$
One path from $s$ to $b$
- Infinitely many paths from $s$ to $c$ (cycle with negative weight)
- We can find path with arbitrarily negative costs
Relaxation

**Idea:** originally assign the path some upper bound and then keep on decreasing it as until you reach the cost of shortest path, $d[u]$ – attribute keeps upper bound on the cost of shortest path

Dijkstra’s Algorithm (G,w,s)  \%(with adjacency lists)

Initialize-Single-Source(G,s)

$D[s] := 0; S:=0$

$Q:= V[G]$  \% priority queue

**while** $Q \neq 0$ **do**

$u := extract\_min(Q)$

$S := S U \{u\}$

**for each** vertex $v$ in $Adj[u]$ **do**

**if** $d[u] + w(u,v) < d[v]$ **then**

$d[v] := d[u] + w(u,v)$

**end**

Relaxation
Running time analysis

The values of the array are kept in priority queue (min-heap)
• Updating the heap structure takes at most $O(lg V)$ and there are $|V|$ of such operations. Building the heap takes $O(V)$.
• Extract min, we need to fix heap structure $|E|$ times, $E lg V$
• Also relaxation needs to Decrease_key for some elements in The heap and still fix the heap structure (that can be done in $lg n$)
• The total running time:
$O ((V+E) lg V)$ for sparse graphs $O (E lg V)$
$O (V lg V + E)$ with Fibonacci heap decrease key amortized cost is $O(1)$

In general $T(n) : |V| * T_{extract\_min} + |E| * T_{decrease\_key}$

If the priority queue is maintained as linear array then Extract_min takes $O(V)$ and there are $|V|$ of such operations then Extract_min will take total $O(V^2) + $ each edge in the Adj. List is examined, then total running time $O(V^2 + E)$
Hence $O(V^2)$ since $E$ is $O(V^2)$.
Bellman-Ford Algorithm

• Can deal with negative weights
  - the problem is when we have negative weight cycles reachable from the source, the solution does not exist
  - Bellman-Ford reports if it finds negative weight cycle otherwise
  - Proceeds as normal shortest path algorithm

1. Initialize d
2. Relaxation |V|-1 times, do relaxation for each edge
3. Test whether you have a solution
Bellman-Ford Algorithm

Bellman-Ford Algorithm \((G, w, s)\) \%(with adjacency lists\)

\[O(V)\] Initialize-Single-Source\((G, s)\)
\[\text{for } i := 1 \text{ to } |V(G)| - 1\]

\[O(VE)\] do for each edge \((u, v)\) in \(E(G)\) \% relaxation

\[\text{if } d[u] + w(u, v) < d[v] \text{ then } d[v] := d[u] + w(u, v)\]

done

\[O(E)\] for each edge \((u, v)\) in \(E(G)\)

\[\text{if } d[u] + w(u, v) < d[v] \text{ then return \(FALSE\)}\]

done

return(\(TRUE\))
Algorithm returns true if there are not negative weight cycles Reachable from the source.

- Running time of the algorithm is $O(VE)$
All pairs shortest path

- Given graph $G = (V,E)$ find the shortest path between all pairs of vertices $v_i, v_j$
- Output the result in matrix $D_{ij}$
- Dynamic programming approach
- Running Bellman-Ford once for each vertex $V. O(VE) = O(V^2E) = O(V^4)$ for dense graphs