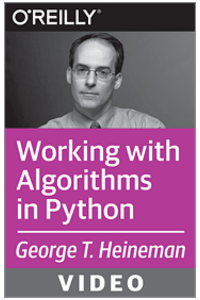
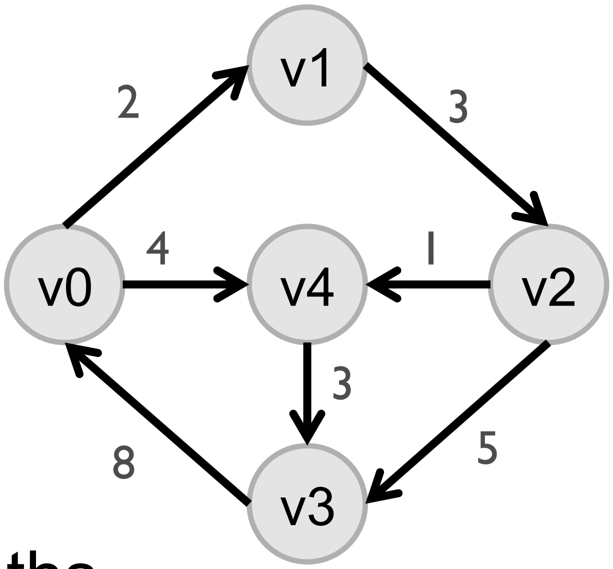
O'REILLY®

ALL PAIRS SHORTEST PATH



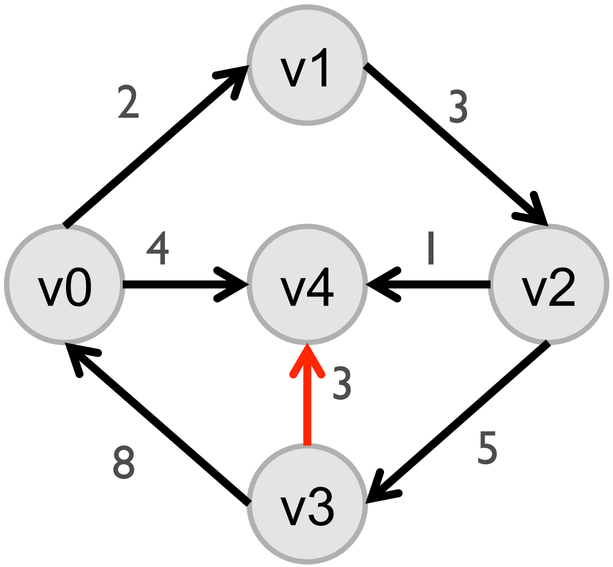
Consider Following Graph

- Consider some questions
  - What is shortest distance from v1 to v3 when considering edge weights?
  - In fact, what is shortest distance between any two vertices?
- Must avoid generating all possible paths



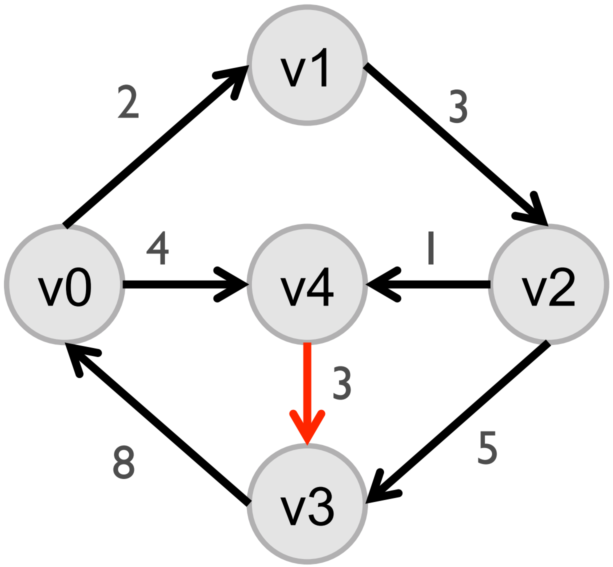
Consider Following Graph

- Observation
  - Edge direction matters!
  - Flip direction of edge (v4,v3) and there would be no way to get from v4 any other vertex



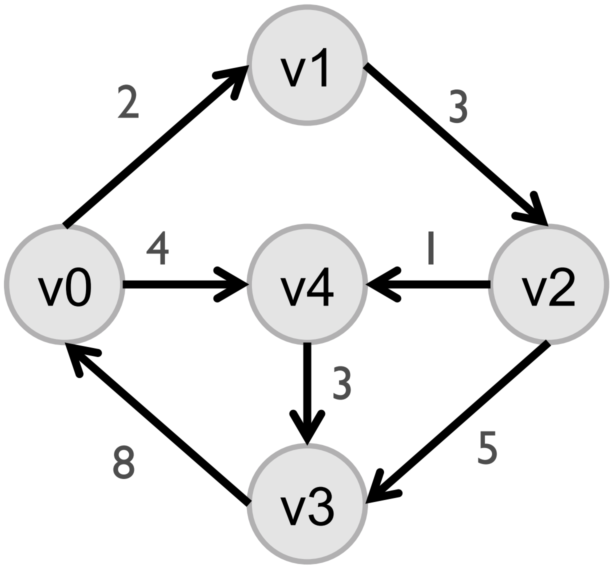
Consider Following Graph

- Observation
  - There is only one edge exiting v4
  - If you can find the shortest path from v3 to the other vertices, just add 3 to find the shortest path from v4 to these vertices
  - Suggests reuse of sub-problems



Consider Following Graph

- Observation
  - Is 5 shortest distance from v2 to v3?
  - That is the direct edge
  - But shortest distance is really 4 because you can go v2 → v4 →v3

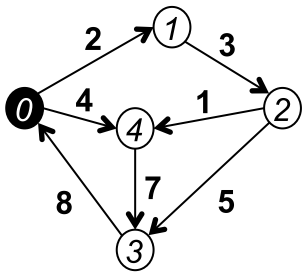


Dynamic Programming Algorithm

- Instead of finding shortest path from single source
  - Generate all pairs shortest paths
- Dynamic Programming
  - Solves small, constrained versions of problems
  - Systematically relax constraints until final answer computed

Dynamic Programming Algorithm

- Given following graph
  - Compute dist[u][v] which represents best estimate of shortest path between vertices
- Starting point
  - Only include original edges in graph
  - “Smallest constrained version of problem”

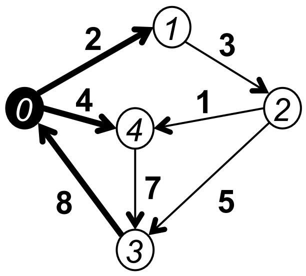


	0	1	2	3	4
0	0	2	∞	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	∞	∞	0	∞
4	∞	∞	∞	7	0

dist[u][v]

Dynamic Programming Algorithm

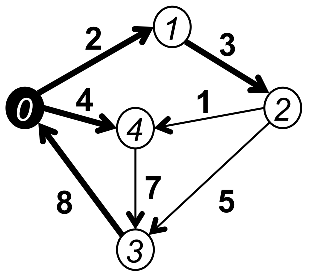
- Relax constraints
  - Allow paths to include vertex v0
  - Observe improvements (v3 → v1, v3 → v4)



	0	1	2	3	4
0	0	2	∞	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	∞	0	12
4	∞	∞	∞	7	0

dist[u][v]

Dynamic Programming Algorithm



- Relax constraints
  - Allow paths to include vertex v0 and v1
  - Two more improvements (v0 → v2, v3 → v2)
- Continue this process until all vertices are allowed in all paths
  - Record separate pred[u][v] array to be able to recover actual shortest paths

	0	1	2	3	4
0	0	2	5	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	13	0	12
4	∞	∞	∞	7	0

dist[u][v]

Dynamic Programming

Initial setup defines dist[][][] and pred[][][] assuming only original edges can be used. Can be completed in $O(n^2)$

Key Step

is dist[u][t]+dist[t][v] < dist[u][v]

Relax constraints by allowing algorithm to consider vertices *t* with each successive pass

Floyd-Warshall

Best	Average	Worst
$O(V^3)$	$O(V^3)$	$O(V^3)$

Weighted Directed Graph

Overflow

Dynamic Programming

2D Array

```
def allPairsShortestPath (G)
 foreach u∈V do
  foreach v∈V do
   if (u = v) then
    dist[u][u] = 0
    pred[u][u] = –1
   else if (exists edge (u,v)) then
    dist[u][v] = weight of edge (u,v)
    pred[u][v] = u
   else
    dist[u][v] = ∞
    pred[u][u] = –1

foreach t∈V do
 foreach u∈V do
  foreach v∈V do
   newLen = dist[u][t]+dist[t][v]
   if (newLen < dist[u][v]) then
    dist[u][v] = newLen
    pred[u][v] = pred[t][v]
```

Dynamic Programming Project

- Minimum Edit Distance between two strings
  – Convert “Grates” to “Create”
- Three possible operations
  – Replace character (i.e., “G” with “C”)
  – Remove character (i.e., delete “s”)
  – Insert character (i.e., “e” after the “r”)
  – Edit distance = 3

Dynamic Programming Project

- Identify Matrix for recording solutions
  - Determine computation that relates past solutions to individual steps
  - When new minimum is found, choose that cost
  - Let’s go to code

		C	R	E	A	T	E
	0	1	2	3	4	5	6
G	1	1	2	3	4	5	6
R	2	2	1	2	3	4	5
A	3	3	2	2	2	3	4
T	4	4	3	2	3	4	4
E	5	5	4	3	2	3	4
S	6	6	5	4	3	2	3

Dynamic Programming Summary

- Polynomial order of growth
  - Applicable even when problems become large
- Technique used in broad range of disciplines
  - Bioinformatics
  - Economics
  - Industrial Engineering