Dynamic Programming

Lecture 9

Timothy Kim
GWU CSCI 6212

All-Pairs Shortest Path

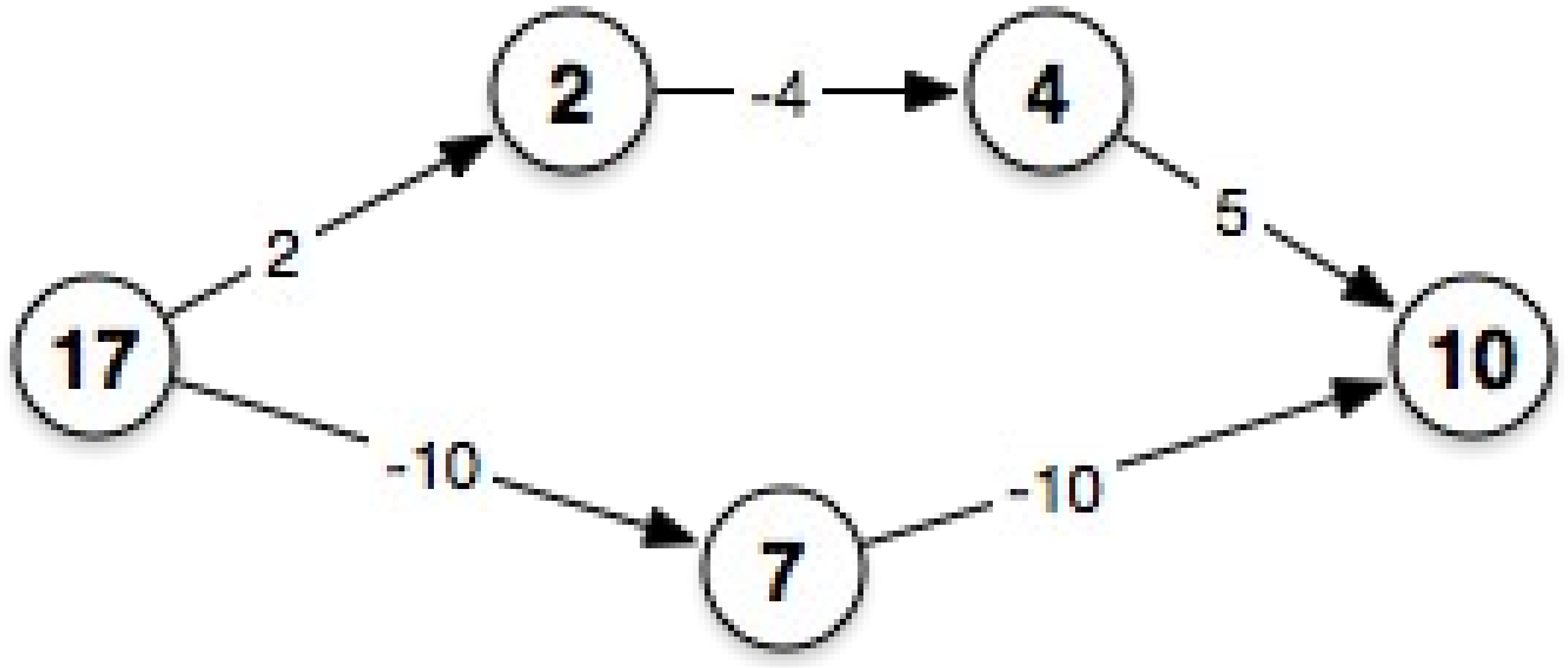
- Input: Directed Graph
- Output: Shortest distances between all possible source-destination pair vertices
- Reduce the problem to n·dijkstra's
  - O(n·m log n)
    - If G is sparse: $O(n^2 \log n)$
    - If G is dense: $O(n^3 \log n)$
  - What about negative edges?

Possible solutions

- For negative edges reduce the problem to n·Bellman-Ford
  - $O(n \cdot m \cdot n)$
    - If G is sparse: $O(n^3)$
    - If G is dense: $O(n^4)$
- Can we do better?

Floyd-Warshall Algorithm

- Principle of optimality
  - Key point: order the vertices V = {1, 2, ... ,n} (similar to Knapsack)
- Notation
  - $V^k = \{1, 2, \ldots, k\}$
- Suppose G has no negative cycle
  - Fix a source, $i \in V$,
  - Fix a destination, $j \in V$,
  - Fix a value, $k \in \{1, 2, \ldots, n\}$
- Let P = shortest i–j path with internal nodes in V^k



Principle of optimality

- Case 1: if k is not internal to P
  - P is the shortest cycle free i—j path using V^{k-1}
- Case 2: if k is internal to P
  - P_1 = shortest i→k path, with V_{k-1}
  - P_2 = shortest k→j path, with V_{k-1}

Recurrence

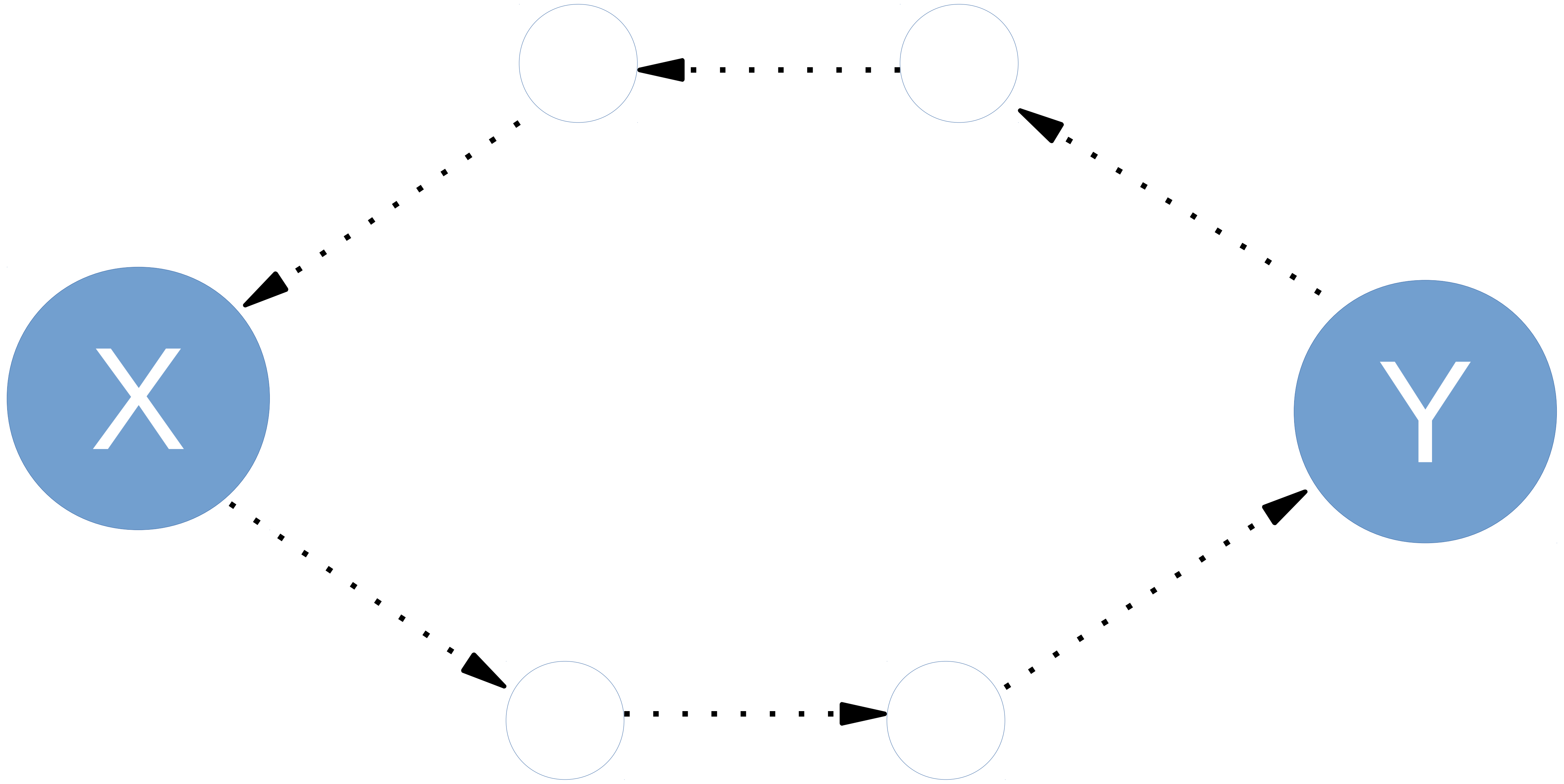
- $A_{i,j,k}$ = length of the shortest $i \rightarrow j$ path with internal node V^k
- $A_{i,j,k} = \min \{ A_{i,j,k-1} , A_{i,k,k-1} + A_{k,j,k-1} \}$
- For all possible values of i,j,k

Algorithm

```
A = 3d-Array (i, j, k)
A[i,j,0] = if (i == j) → 0
           if (i,j) ∈ E → Length[i,j]
           else → ∞
for k = 1 to n:
  for i = 1 to n:
    for j = 1 to n:
      A[i,j,k] = min( A[i,j,k-1],
                      A[i,k,k-1] + A[k,j,k-1] )
```

Negative cycle?

- If A$[i,i,n] < 0$ for at least one $i \in$ V then there must be a negative cycle.
- When calculating A$[i,i,n]$, the second case must've produced a negative value.



Reconstruction?

- We need to keep track how many vertices we used to generate the shortest path in order to back track.

- $B[i,j]$ = max label of an internal node on a shortest i – j path.

```
A = 3d-Array (i, j, k)
B = 2d-Array
A[i,j,0] = if (i == j) → 0
           if (i,j) ∈ E → Length[i,j]
           else → ∞
for k = 1 to n:
  for i = 1 to n:
    for j = 1 to n:
      case1 = A[i,j,k-1]
      case2 = A[i,k,k-1] + A[k,j,k-1]
      if (case1 < case):
        A[i,j,k] = case1
      else:
        A[i,j,k] = case2
        B[i,j] = k
```

APSP Comparison

	Sparse	Dense	Note
n·Dijkstra's	$O(n^2 \log n)$	$O(n^3 \log n)$	Non-negative edges
n·Bellman-Ford	$O(n^3)$	$O(n^4)$	Negative Edges
Floyd-Warshall	$O(n^3)$	$O(n^3)$	Negative Edges

Johnson's Algorithm

- We just want to use Dijkstra's because it's so fast. But we have to deal with negative edges.
- Can we get rid of them?
- Idea 1: shift the value to eliminate the negatives.
  - It works if two paths have same # of edges

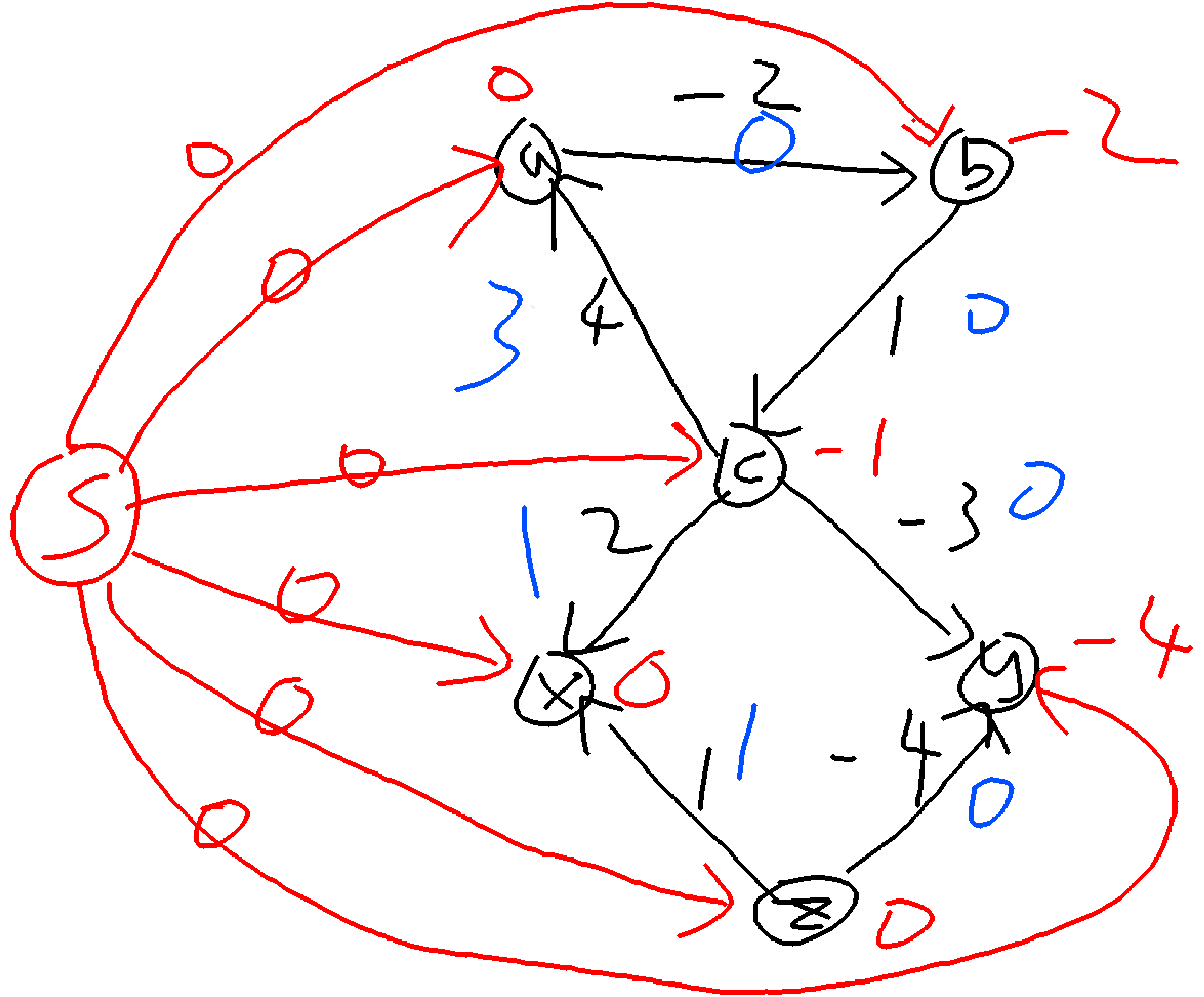
Getting Rid of negative edges

- Idea 2:
  - Fix a P_v for each $v \in V$.
  - For every e = (u,v) of G,

    $c'_e = c_e + P_u - P_v$
  - Given a path, P: s $\rightarrow$ t, $L = \Sigma c_e$
  - Note, $\Sigma c'_e = \Sigma c_e + P_u - P_v$
  - Then $L' = L + P_s - P_t$
  - This preserves shortest path!
- Only if we can find {Pv} that will make every path length non-negative...

Johnson's Algorithm

- Algorithm Reduction
  - 1 invocation of Bellman-Ford to get rid of negative edges – O(mn)
  - n invocation of Dijkstra to find all shortest paths – O(n m log n)
- Bellman-Ford to rid of negative edges:
  - Add a starting vertex S and create n edges between S and all other vertices with edge length of 0.
  - Call Bellman-Ford to find the shortest paths using S as the source.
  - P_v = length of the shortest path from Bellman-Ford
- But does it work?


s
a
b
c
x
y
z
-2
4
1
2
-3
1
-4
0
-2
-1
-4
3
1

Proof

- Claim: for every edge $e \in G$, $c'_e = c_e + P_u + P_v$ is non-negative
- Proof:
  - Fix an edge (u,v)

    P_u = length of shortest path $S \to u$

    P_v = length of shortest path $S \to v$
  - Let P = a shortest $S \to u$ path in G'

    $P + (u,v) = S \to v$ path with length $P_u + c_{u \to v}$

    $P_v \leq P_u + c_{u \to v}$

    $c'_{u \to v} = c_{u \to v} + P_u - P_v \geq 0$

Johnson's Algorithm

1) G' = {S + V, n edges with 0 length from S to v + E} O(n)

2) Run Bellman-Ford on G' with S as source, if negative cycle is detected then halt O(mn)

3) For each v ∈ G, P_v = length of shortest path S → v in G', and for all e ∈ G, calculate the new length: $c'_e = c_e + P_u - P_v$ O(m)

4) For each vertex u of G, run Dijkstra in G with $\{c'_e\}$ using u as source: d'(u,v)   O(n m log n)

5) For each pair of u,v ∈ G, $d(u,v) = d'(u,v) - P_u + P_v$ $O(n^2)$

APSP Comparison

	Raw	Sparse	Dense	Note
n·Dijkstra's	$O(n\ m \log n)$	$O(n^2 \log n)$	$O(n^3 \log n)$	Non-negative edges
n·Bellman-Ford	$O(n\ m\ n)$	$O(n^3)$	$O(n^4)$	Negative Edges
Floyd-Warshall	$O(n^3)$	$O(n^3)$	$O(n^3)$	Negative Edges
Johnson's	$O(n\ m \log n)$	$O(n^2 \log n)$	$O(n^3 \log n)$	Negative Edges