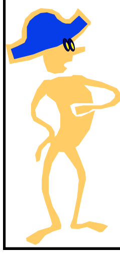
Algorithm Design and Analysis

Victor Adamchik — CS 15-451 — Spring 2014

Lecture 13 — Feb 12, 2014 — Carnegie Mellon University

Graph Algorithms - 3

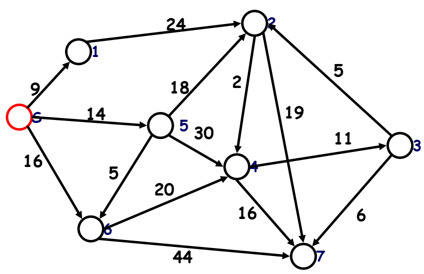
The Shortest Path Problem



Edsger Dijkstra
(1930-2002)

The Shortest Path Problem

Given a positively weighted graph G with a source vertex s, find the shortest path from s to all other vertices in the graph.



Greedy approach

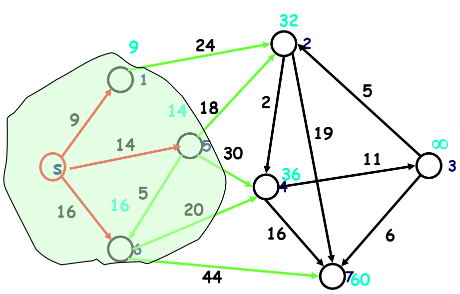
When algorithm proceeds all vertices are divided into two groups

- vertices whose shortest path from the source is known
- vertices whose shortest path from the source is NOT known

Move vertices (shortest distance) one at a time from the unknown set to the known set.

Maintain a PQ of distances from the source to a vertex.

The Shortest Path Problem



Complexity

O (V Log V + E log V)

Let D(v) denote a length from the source s to vertex v. We store distances D(v) in a PQ.

O(V)

INIT: D(s) = 0; D(v)= ∞ — PQ has V vertices

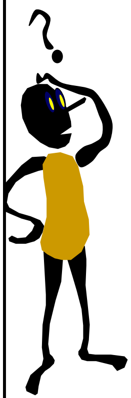
LOOP: — O(log V)

Delete a node v from PQ using deleteMin()

Update D(w) for all w in adj(v) using decreaseKey()

O(log V)

D(w) = min[D(w), D(v) + c(v, w)] — We do O(E) updates



Assume that a unsorted array is used instead of a priority queue.

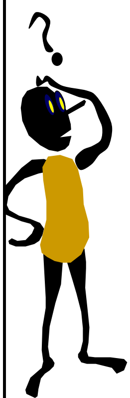
What would the algorithm's running time in this case?

PQ is a linear array

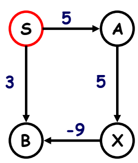
findMin takes $O(V)$ - for one vertex
findMin takes $O(V^2)$ - for all vertices

Updating takes $O(1)$ - for one edge
total edge adjustment $O(E)$

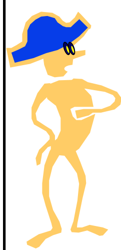
the algorithm running time $O(E + V^2)$



Why Dijkstra's algorithm does not work on graphs with negative weights?



The Bellman-Ford algorithm (1958)



```
repeat V - 1 times:
for all e in E:
  update(e)
```

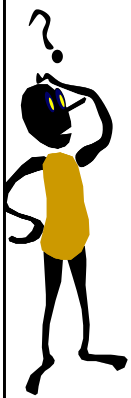
The Bellman-Ford Algorithm

```
for (k = 0; k < V; k++)  dist[k] = INFINITY;

Queue q = new Queue();
dist[s] = 0; q.enqueue(s);
while (!q.isEmpty())
 {
   v = q.dequeue();
   for each w in adj(v) do
     if (dist[w] > dist[v] + weight[v,w]) {
          dist[w] = dist[v] + weight[v,w];
          if (!q.isInQueue(w)) q.enqueue(w);
    }}
```

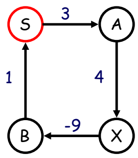
What is the worst-case complexity of the Bellman-Ford algorithm?

```
for (k = 0; k < V; k++)
          dist[k] = INFINITY;

Queue q = new Queue();                       V
dist[s] = 0; q.enqueue(s);
while (!q.isEmpty())  {
   v = q.dequeue();                               E
   for each w in adj(v) do
     if (dist[w] > dist[v] + weight[v,w]) {
          dist[w] = dist[v] + weight[v,w];
          if (!q.isInQueue(w)) q.enqueue(w);
     }}
```

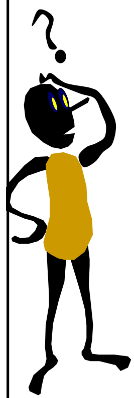
$O(V\ E)$



Graph with a negative cycle?



How would you apply the Bellman-Ford algorithm to find out if a graph has a negative cycle?



How would you apply the Bellman-Ford algorithm to find out if a graph has a negative cycle?

Do not stop after V-1 iteration, perform one more round. If there is such a cycle, then some distance will be reduced...

Bellman-Ford

Dynamic programming approach

We will be counting the number of edges in the shortest path

Dynamic programming approach

For each node, find the length of the shortest path to t that uses at most 1 edge, or write down ∞ if there is no such path.

Suppose for all v we have solved for length of the shortest path to t that uses k – 1 or fewer edges. How can we use this to solve for the shortest path that uses k or fewer edges?

We go to some neighbor x of v, and then take the shortest path from x to t that uses k–1 or fewer edges.

All-Pairs Shortest Paths (APSP)

Given a weighted graph, find a shortest path from any vertex to any other vertex.

Note, no distinguished vertex

All-Pairs Shortest Paths

One approach: run Dijkstra's algorithm using every vertex as a source.

Complexity: O(V E Log V)

sparse: $O(V^2 \text{ Log } V)$

dense: $O(V^3 \text{ Log } V)$

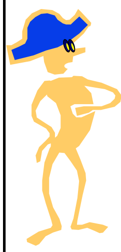
But what about negative weights...

APSP: Bellman-Ford's

Complexity : $O(V^2 E)$

Note, for a dense graph we have $O(V^4)$.

APSP : Dynamic programming approach



Floyd-Warshall, $O(V^3)$

We won't discuss it…

APSP: Johnson's algorithm

Complexity: $O(V E + V E \log V)$

for a dense graph -- $O(V^3 \log V)$.

for a sparse graph -- $O(V^2 \log V)$.

Johnson's Algorithm

It improves the runtime only when a graph has negative weights.

A bird's view:
- Reweight the graph, so all weights are nonnegative (by running Bellman-Ford's)
- Run Dijkstra's on all vertices

Complexity: $O(V E + V E \log V)$

Johnson's Algorithm: intuition

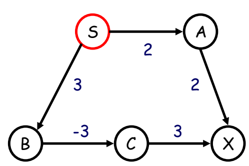
The way to improve the runtime is to run Dijkstra's from each vertex.

But Dijkstra's does not work on negative edges.

So what about if we change the edge weight to be nonnegative?

We have to be careful on changing the edge weight… to preserve the shortest path

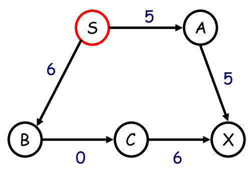
Wrong reweighting (adding the fix amount)



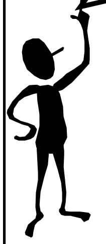
The actual shortest path to X is S-B-C-X

Let us add 3 to all edges

Wrong reweighting
(adding the fix amount)



Shortest path to X is S-A-X



Adding the fix amount does not work, since every shortest path has a different number of edges

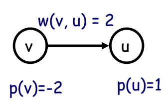
Johnson's Algorithm: reweighting

Every edge (v, u) with the cost w(v, u) is replaced by

$$w^*(v, u) = w(v, u) + p(v) - p(u)$$

where p(v) will be decided later.

$w^*(v, u) = 2 + (-2) - 1$
$= -1$



Johnson's Algorithm: reweighting

Theorem. All paths between the same two vertices are reweighted by the same amount.

Proof.

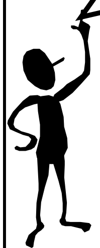
Consider path $v = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n = u$

Then we have

$w^*(v,u) = w^*(v_1, v_2) + \ldots + w^*(v_{n-1}, v_n)$
$= w(v_1, v_2) + p(v_1) - p(v_2) +$
$w(v_2, v_3) + p(v_2) - p(v_3) + \ldots$

Telescoping sum

$\boxed{w^*(v, u) = w(v, u) + p(v) - p(u)}$

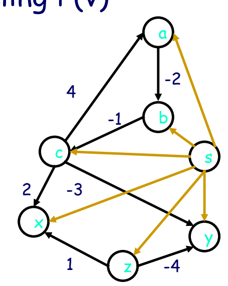


Johnson's reweighting changes any path between u and v by the same amount and therefore preserves the shortest path unchanged

$\boxed{w^*(v, u) = w(v, u) + p(v) - p(u)}$

Find vertex labeling P(v)

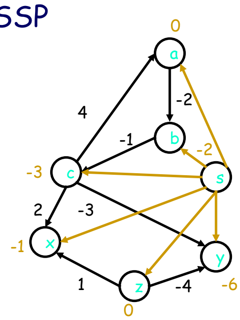
First we need to create a new vertex and connect it to all other vertices with zero weight.



Note this change in the graph won't change the shortest distances between vertices.

Running SSSP

Next we run Bellman-Ford's starting at vertex s.



Shortest Path s-a is 0

s-b is -2

s-c is -3

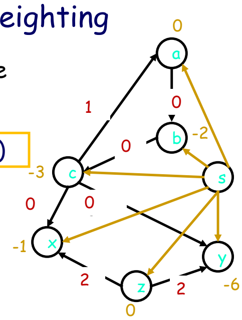
and so on…

Now we define p(v) as the shortest distance s-v.

Johnson's Reweighting

Here we redraw the example by using

$\boxed{w^*(v, u) = w(v, u) + p(v) - p(u)}$



Edge (a,b): -2+0-(-2) = 0

Edge (b,c): -1+(-2)-(-3) = 0

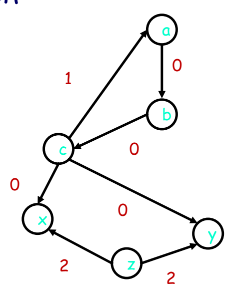
Edge (z,x): 1+0-(-1) = 2

New graph

After Johnson's reweighting we get a new graph with <u>non-negative weights</u>.

Remember, Johnson's reweighting preserves the shortest path.

Now we can use Dijkstra's



Johnson's Algorithm: reweighting

$\boxed{w^*(v, u) = w(v, u) + p(v) - p(u)}$

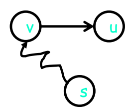
<u>Theorem.</u> After reweighting every edge has a nonnegative cost.

Proof. Consider edge (v, u)

p(v) is the shortest distance from s to v

p(u) is the shortest distance from s to u

$p(u) \le p(v) + w(v, u)$



since the shortest path s-u cannot be longer then p(v) + (v, u).

QED

Johnson's Algorithm

1. Add a new vertex s and connect it with all other vertices.
2. Run Bellman-Ford's algorithm from s to compute p(v).

   Note that Bellman-Ford's algorithm will correctly report if the original graph has a negative cost cycle.
3. Reweight all edges: $w^*(v,u) = w(v,u)+p(v)-p(u)$
4. Run Dijkstra's algorithm from all vertices
5. Compute the actual distances by subtracting p(v)-p(u)

Complexity

1. Add a new vertex s and connect it with all other vertices. O(V)
2. Run Bellman-Ford's algorithm from s to compute p(v). O(V E)
3. Reweight all edges: $w^*(v,u) = w(v,u)+p(v)-p(u)$ O(E)
4. Run Dijkstra's algorithm from all vertices O(V E log V)
5. Compute the actual distances by subtracting p(v)-p(u) O(E)

Total: O(V E log V)

Johnson's Algorithm

It shines for sparse graphs with negative edges

$O(V^2 \log V)$

Better than Floyd-Warshall's, which is $O(V^3)$