CS2010 – Data Structures and Algorithms II
Lecture 08 – Finding Shortest Way from Here to There, Part I
stevenhalim@gmail.com
Outline

Single-Source Shortest Paths (SSSP) Problem
• Motivating example
• Some more definitions
• Discussion of negative weight edges and cycles

Algorithms to Solve SSSP Problem (CP3 Section 4.4)
• BFS algorithm cannot be used for the general SSSP problem
• Bellman Ford’s algorithm
  – Pseudo code, example animation, and later: Java implementation
  – Theorem, proof, and corollary about Bellman Ford’s algorithm
Motivating Example
Review: Definitions that you know (1)

- **Vertex set** \( V \) (e.g. street intersections, houses, etc)
- **Edge set** \( E \) (e.g. streets, roads, avenues, etc)
  - **Directed** (e.g. one way road, etc)
    - Note that we can simply use 2 edges (bi-directional) to model 1 undirected edge (e.g. two ways road, etc)
    - Recall that for the MST problem discussed in the previous lecture, we generally deal with a **connected undirected weighted graph**
  - **Weighted** (e.g. distance, time, toll, etc)
    - Weight function \( w(a, b) : E \rightarrow R \), sets the weight of edge from \( a \) to \( b \)
- **Weighted Graph**: \( G(V, E), w(a, b) : E \rightarrow R \)
Review: Definitions that you know (2)

• **(Simple) Path** $p = \langle v_0, v_1, v_2, \ldots, v_k \rangle$
  
  - Where $(v_i, v_{i+1}) \in E$, $\forall 0 \leq i \leq (k-1)$
  
  - Simple = No repeated vertex!

• **Shortcut notation**: $v_0 \xrightarrow{p} v_k$
  
  - Means that $p$ is a path from $v_0$ to $v_k$

• **Path weight**: $PW(p) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$
More Definitions (1)

• **Shortest Path weight** from vertex $a$ to $b$: $\delta(a, b)$
  
  – $\delta$ is pronounced as ‘delta’
  
  $\delta(a, b) = \begin{cases} 
  \min(PW(p)) & \text{if there exists such path} \\
  \infty & \text{if } b \text{ is unreachable from } a
  \end{cases}$

• **Single-Source Shortest Paths** (SSSP) Problem:
  
  – Given $G(V, E)$, $w(a, b): E \rightarrow \mathbb{R}$, and a **source vertex** $s$
  
  – Find $\delta(s, b)$ (+best paths) from vertex $s$ to each vertex $b \in V$
    
    • i.e. From one source **to the rest**
More Definitions (2)

- **Additional Data Structures** to solve the SSSP Problem:
  - An array/Vector $D$ of size $V$ ($D$ stands for ‘distance’)
    - Initially, $D[v] = 0$ if $v = s$; otherwise $D[v] = \infty$ (a large number)
    - $D[v]$ decreases as we find better paths
    - $D[v] \geq \delta(s, v)$ throughout the execution of SSSP algorithm
    - $D[v] = \delta(s, v)$ at the end of SSSP algorithm
  - An array/Vector $p$ of size $V$
    - $p[v]$ = the predecessor on best path from source $s$ to $v$
    - $p[s]$ = NULL (not defined, we can use a value like -1 for this)
    - Recall: The usage of this array/Vector $p$ is already discussed in BFS/DFS Spanning Tree (and also in PS4, Min Spanning Tree)
s = 0
Initially:
D[s] = D[0] = 0
D[v] = ∞ for the rest
Denoted as values in **red font/vertex**
p[s] = -1 (to say ‘no predecessor’)
p[v] = -1 for the rest
Denoted as **orange edges (none initially)**

---

s = 0
At the end of algorithm:
D[s] = D[0] = 0 (unchanged)
D[v] = δ(s, v) for the rest
p[s] = -1 (source has no predecessor)
p[v] = the origin of **orange edges** for the rest
e.g. p[0] = 2, p[4] = 0
Negative Weight Edges and Cycles

They exist in some applications

• Fictional application: Suppose you can travel back in time by passing through time tunnel (edges with negative weight)

• Shortest paths from 0 to \{1, 2, 3\} are undefined
  – 1→2→1 is a negative cycle as it has negative total path (cycle) weight
  – One can take 0→1→2→1→2→1→... indefinitely to get -∞

• Shortest path from 0 to 4 is ok, with \(\delta(0, 4) = -99\)
SSSP Algorithms

This SSSP problem is another well-known CS problem

We will discuss three algorithms in this lecture:

1. $O(V+E)$ BFS fails on general case of SSSP problem
   - Introducing the “initSSSP” and “Relax” operations

2. $O(VE)$ Bellman Ford’s SSSP algorithm
   - General idea of SSSP algorithm
   - Trick to ensure termination of the algorithm
   - Bonus: Detecting negative weight cycle
Initialization Step

We will use this initialization step for all our SSSP algorithms

\[
\text{initSSSP}(s) \\
\text{for each } v \in V \quad \text{// initialization phase} \\
\quad D[v] \leftarrow 1000000000 \quad \text{// use 1B to represent INF} \\
\quad p[v] \leftarrow -1 \quad \text{// use -1 to represent NULL} \\
\quad D[s] \leftarrow 0 \quad \text{// this is what we know so far}
\]
“Relax” Operation
(abbreviated name of these actions)

relax(u, v, w_{u,v})

if D[v] > D[u]+w_{u,v} // if SP can be shortened
  D[v] ← D[u]+w_{u,v} // relax this edge
  p[v] ← u // remember/update the predecessor
// if necessary, update some data structure
Review: BFS

When the graph is **unweighted***, the SSSP can be viewed as a problem of finding the **least number of edges** traversed from source $s$ to other vertices

* We can view each edge as having weight 1 or constant weight

The $O(V+E)$ Breadth First Search (BFS) traversal algorithm precisely measures such thing

- BFS Spanning Tree = Shortest Paths Spanning Tree
Modified BFS

Do these **three** simple modifications:

1. Rename `visited` to `D` 😊
2. At the start of BFS, set `D[v] = INF` (say, `1B`) for all `v` in `G`, except `D[s] = 0` 😊
3. Change this part (in the BFS loop) from:
   ```java
   if visited[v] = 0 // if v is not visited before
     visited[v] = 1; // set v as reachable from u
   ```
   into:
   ```java
   if D[v] = INF // if v is not visited before
     D[v] = D[u]+1; // v is 1 step away from u 😊
   ```
Modified BFS Pseudo Code (1)

for all v in V
    D[v] \leftarrow \text{INF}
    p[v] \leftarrow -1
Q \leftarrow \{s\} \quad \text{// start from s}
D[s] \leftarrow 0

while Q is not empty
    u \leftarrow Q\text{.dequeue()}
    for all v adjacent to u \quad \text{// order of neighbor}
        if D[v] = \text{INF} \quad \text{// influences BFS}
            D[v] \leftarrow D[u]+1 \quad \text{// visitation sequence}
            p[v] \leftarrow u
            Q\text{.enqueue}(v)

// we can then use information stored in D/p
Modified BFS Pseudo Code (2)
simpler form

initSSSP(s)
Q ← {s} // start from s

while Q is not empty
    u ← Q.dequeue()
    for all v adjacent to u // order of neighbor
        relax(u, v, 1); // the weight is 1

// we can then use information stored in D/p
SSSP: BFS on Unweighted Graph

Ask VisuAlgo to perform BFS *from various sources* on the sample Graph (CP3 4.3)

In the screen shot below, we show the start of BFS from source vertex 5 (the same example as in Lecture 06, *it just looks messier due to bidirectional edges*)
But BFS will not work on general cases

The shortest path from 0 to 2 is not path 0→2 with weight 9, but a “detour” path 0→1→3→4→2 with weight 2+3+2+1= 8

- BFS cannot detect this and will only report path 0→2 (wrong answer)
- You can draw this graph @ VisuAlgo and try it for yourself

**Rule of Thumb:**
If you know for sure that your graph is unweighted (all edges have weight 1 or all edges have the same constant weight), then solve SSSP problem on it using the more efficient $O(V+E)$ BFS algorithm
Reference: CP3 Section 4.4 (especially Section 4.4.4)

visualgo.net/sssp.html

BELLMAN FORD’S SSSP ALGORITHM
Bellman Ford’s Algorithm

initSSSP(s)

// Simple Bellman Ford's algorithm runs in $O(VE)$
for $i = 1$ to $|V|-1$ // $O(V)$ here
    for each edge $(u, v) \in E$ // $O(E)$ here
        relax$(u, v, w_{u,v})$ // $O(1)$ here

// At the end of Bellman Ford's algorithm,
// $D[v] = \delta(s, v)$ if no negative weight cycle exist

// Q: Why "relaxing all edges $V-1$ times" works?
SSSP: Bellman Ford’s

Ask VisuAlgo to perform Bellman Ford’s algorithm *from various sources* on the sample Graph (CP3 4.17)

The screen shot below is *the first pass* of all E edges of *BellmanFord(0)*
Theorem: If $G = (V, E)$ contains no negative weight cycle, then the shortest path $p$ from $s$ to $v$ is a simple path.

Let’s do a Proof by Contradiction!

1. Suppose the shortest path $p$ is not a simple path.
2. Then $p$ contains one (or more) cycle(s).
3. Suppose there is a cycle $c$ in $p$ with positive weight.
4. If we remove $c$ from $p$, then we have a shorter ‘shortest path’ than $p$.
5. This contradicts the fact that $p$ is a shortest path.
Theorem: If $G = (V, E)$ contains no negative weight cycle, then the shortest path $p$ from $s$ to $v$ is a simple path

6. Even if $c$ is a cycle with zero total weight (it is possible!), we can still remove $c$ from $p$ without increasing the shortest path weight of $p$

7. So, $p$ is a simple path (from point 5) or can always be made into a simple path (from point 6)

In another word, path $p$ has at most $|V| - 1$ edges from the source $s$ to the “furthest possible” vertex $v$ in $G$ (in terms of number of edges in the shortest path)
Theorem: If $G = (V, E)$ contains no negative weight cycle, then after Bellman Ford’s terminates $D[v] = \delta(s, v), \forall v \in V$

Let’s do a **Proof by Induction**!

1. Consider the shortest path $p$ from $s$ to $v_i$  
   $(p$ will have minimum number of edges$)$
   - $v_i$ is defined as a vertex which shortest path requires $i$ hops (number of edges) from $s$

2. Initially $D[v_0] = \delta(s, v_0) = 0$, as $v_0$ is just $s$

3. After 1 pass through $E$, we have $D[v_1] = \delta(s, v_1)$
Theorem: If $G = (V, E)$ contains no negative weight cycle, then after Bellman Ford’s terminates $D[v] = \delta(s, v), \forall v \in V$

4. After 2 passes through $E$, we have $D[v_2] = \delta(s, v2), ...$
5. After $k$ passes through $E$, we have $D[v_k] = \delta(s, v_k)$
6. When there is no negative weight cycle, the shortest path $p$ will be simple (see the previous proof)
7. Thus, after $|V|-1$ iterations, the “furthest” vertex $v_{|V|-1}$ from $s$ has $D[v_{|V|-1}] = \delta(s, v_{|V|-1})$
   – Even if edges in $E$ are in worst possible order
“Side Effect” of Bellman Ford’s

Corollary: If a value $D[v]$ fails to converge after $|V|-1$ passes, then there exists a negative-weight cycle reachable from $s$

Additional check after running Bellman Ford’s:

for each edge $(u, v) \in E$
    if $D[v] > D[u] + w(u, v)$
        report negative weight cycle exists in G
Java Implementation (2)

See BellmanFordDemo.java

• Now implemented using AdjacencyList 😊
  – AdjacencyList or EdgeList can be used to have an $O(VE)$ Bellman Ford’s

Show performance on:

• Small [graph](#) without negative weight cycle $\rightarrow$ OK, in $O(VE)$
• Small [graph](#) with negative weight cycle $\rightarrow$ terminate in $O(VE)$
  – Plus we can report that negative weight cycle exists
• Small [graph](#); some negative edges; no negative cycle $\rightarrow$ OK
Summary

Introducing the SSSP problem

Revisiting BFS algorithm for unweighted SSSP problem
• But it fails on general case

Introducing Bellman Ford’s algorithm
• This one solves SSSP for general weighted graph in $O(VE)$
• Can also be used to detect the presence of -ve weight cycle
PS5* should now be doable 😊

* The first Subtask of PS5...
(but I will only open it on Saturday, 17 Oct 2015, 8am)

Subtask B (easy), Subtask C (medium-hard), and Subtask E (R-option, also medium-hard) require something else 😊

Train first to check basic understanding of the past two lectures on graph algorithms:
http://visualgo.net/training.html?diff=Medium&n=5&tl=0&module=mst,sssp