Graphs II - Shortest paths

Single Source Shortest Paths
All Sources Shortest Paths

some drawings and notes from prof. Tom Cormen
Single Source SP

- Context: directed graph $G=(V,E,w)$, weighted edges
- The shortest path (SP) between vertices $u$ and $v$ is the path that has minimum total weight
  - total weight is obtained by summing up path's edges weights
  $$\delta(u, v) = \begin{cases} \min \{w(p) : u \xrightarrow{p} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$
- Note: SP cannot contain cycles
  - positive cycles: a shortest path obtained by taking out the cycle
  - negative cycles: a shortest path obtained by iterating through the cycle few more times, minimum weight is $-\infty$. 
Negative edges and cycles

- Exercise: explain the following:
  - $SP(s, a) = 3$
  - $SP(s, b) = -1$
  - $SP(s, g) = 3$
  - $SP(s, e) = -\infty$

- negative weights possible
- negative cycles make some shortest paths $-\infty$
Task: Given a source vertex \( s \in V \), find the shortest path from \( s \) to all other vertices.

- Will write inside each vertex \( v \) the shortest path estimate \( \text{ESP}(s,v) \) weight from the source.
- These estimates change as the algorithm progresses.
- Highlight edges that give the SP-s.
- Highlighted edges form a tree with source as root.
- Tree not unique as (b) and (c) are both valid.
Relaxation

- If current (estimate) ESP(s,u) is 5 and edge (u,v) has weight w(u,v)=2, we can reach v with a path of 5+2=7
  - If current estimate ESP(s,v) is more than 7, we “relax edge (u,v)” by replacing the estimate ESP(s,v) =7.
  - If not (ESP(s,v) ≤7), we do nothing
Bellman Ford

- source is the SP tree root
- BF algorithm progresses in "waves", similar to BFS
- takes a maximum of $|V|-1$ waves to find SP
  - since there cannot be cycles
Bellman-Ford SSSP algorithm

- idea: relax all edges once (in any order) and we've got CORRECT all SP-s of one edge
  - relax again all edges (any order) and we obtained all SP-s of two edges
  - relax .... again, and get all SP-s of three edges
  - no SP can have more than |V|-1 edges, so repeat the relax-all-edges step |V|-1 times, to get all SP-s

Bellman-Ford

- init all SP: SP(s,v) = -∞ for all v
- for k=1:|V|-1
  - relax all edges
- check for negative cycles
SSSP exercise

- Discover SP by hand (start from source)
Bellman Ford

• discover \( SP(s,v) \) means having the current estimate equal with the actual (unknown) \( SP \)
  
  – discover \( SP : ESP(s,v) = SP(s,v) \)
  
  – ESP written "inside" each node, it may further decrease
  
  – once \( SP \) discovered, the ESP never decreases
Bellman Ford

- discover $SP(s,v)$ means having the current estimate equal with the actual (unknown) $SP$
  - discover $SP : ESP(s,v) = SP(s,v)$
  - $ESP$ written "inside" each node, it may further decrease
  - once $SP$ discovered, the $ESP$ never decreases

- init all $ESP = \infty$
Bellman Ford

- discover \( SP(s,v) \) means having the current estimate equal with the actual (unknown) \( SP \)
  - discover \( SP : ESP(s,v) = SP(s,v) \)
  - ESP written "inside" each node, it may further decrease
  - once \( SP \) discovered, the ESP never decreases
  - init all ESP = \( \infty \)
  - relax all edges (first time): discover all \( SP \)-s of one edge
Bellman Ford

- discover SP(s,v) means having the current estimate equal with the actual (unknown) SP
  - discover SP : ESP(s,v) = SP(s,v)
  - ESP written "inside" each node, it may further decrease
  - once SP discovered, the ESP never decreases

- init all ESP = \( \infty \)
- relax all edges (first time): discover all SP-s of one edge
- relax all edges (second time): discover all SP-s of two edges
Bellman Ford

• discover SP(s,v) means having the current estimate equal with the actual (unknown) SP
  – discover SP : ESP(s,v) = SP(s,v)
  – ESP written "inside" each node, it may further decrease
  – once SP discovered, the ESP never decreases

• init all ESP = ∞
• relax all edges (first time): discover all SP-s of one edge
• relax all edges (second time): discover all SP-s of two edges
• . . . repeat
  – how many times?
Bellman Ford

Essential mechanism (BF proof):

- \( SP(s,v) = [a_1, a_2, a_3, a_4] \)
- Relaxing \( a_1 \), then \( a_2 \), then \( a_3 \), then \( a_4 \) – you can do them over any amount of time, but it has to be in the right order
- \( SP(s,v) \) discovered
- for every \( SP=(edges \ a_1,a_2,a_3,…) \) there was a relaxation sequence of these edges, in this precise order: \( a_1 \) in the first round, \( a_2 \) in the second round, etc.
- overall quite a few more relaxations than necessary, in order to enforce correctness in all possible cases

Running time: \(|V|-1 \) iterations for the outer loop

inner loop: relax all edges \( O(E) \)
SSSP in a DAG

• Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

• in a DAG we have a way to relax all edges in path-order, without doing \(|V|-1\) rounds of relax-all-edges

• use topological sort, relax edges in topological order.

• Running time \(O(E)\) (if \(E>V\))
  - formally \(O(E+V)\)
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time $O(E)$ (if $E>V$)
  - formally $O(E+V)$
SSSP in a DAG

Essential mechanism:
- for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges
- use topological sort, relax edges in topological order.

- Running time \( O(E) \) (if \( E>V \))
  - formally \( O(E+V) \)
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time $O(E)$ (if $E>V$)
  - formally $O(E+V)$
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time \(O(E)\) (if \(E>V\))
  - formally \(O(E+V)\)
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time $O(E)$ (if $E>V$)
  - formally $O(E+V)$
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time $O(E)$ (if $E>V$)
  - formally $O(E+V)$
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time $O(E)$ (if $E>V$)
  - formally $O(E+V)$
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges $a_1,a_2,a_3,...$) there was a relaxation sequence of these edges, in this precise order: $a_1$ in the first round, $a_2$ in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing $|V|-1$ rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time $O(E)$ (if $E>V$)
  - formally $O(E+V)$
SSSP in a DAG

- Essential mechanism:
  - for every SP=(edges a1,a2,a3,...) there was a relaxation sequence of these edges, in this precise order: a1 in the first round, a2 in the second round, etc.

- in a DAG we have a way to relax all edges in path-order, without doing |V|-1 rounds of relax-all-edges

- use topological sort, relax edges in topological order.

- Running time $O(E)$ (if $E \geq V$)
  - formally $O(E+V)$
Dijkstra SSSP algorithm

- No negative weight edges allowed
- Instead of relaxing all edges (like Bellman Ford), keep track of a current "closest" vertex to the SP tree
  - "closest" = minimum ESP(s,v) of nodes not already part of SP tree
  - Add the current-closest to the partial SP tree, v
  - Relax the outing edges of v (all edges v->x)
- Repeat
- Similar to Prim's algorithm (conceptually)
We want to find the shortest path from $s$ to every node

diagram

graph $G$
source = $s$
After initialization, we have $v.\pi = NIL$ for all $v \in V, s.d = 0$, and $v.d = \infty$ for $v \in V - \{s\}$.
We are at node $s$

$s = \text{EXTRACT-MIN}(Q)$

$S = \{s\}$

$Q = \{t, x, y, z\}$
Test whether we can improve the shortest path to t found so far by going through s
Update $t.d = 10$ and $t.\pi = s$

**RELAX**($s, t, w$)

$S = \{s\}$

$Q = \{t, x, y, z\}$
Test whether we can improve the shortest path to \( y \) found so far by going through \( s \).
Update \( y.d = 5 \) and \( y.\pi = s \)

\[
\text{RELAX}(s, y, w)
S = \{s\}
Q = \{t, x, y, z\}
\]
All edges leaving s have been tested

$S = \{s\}$

$Q = \{t, x, y, z\}$
We are at node y

\[ y = \text{EXTRACT-MIN}(Q) \]
\[ S = \{ s, y \} \]
\[ Q = \{ t, x, z \} \]
Test whether we can improve the shortest path to t found so far by going through y.
Update $t.d = 8$ and $t.\pi = y$

RELAX($y$, $t$, $w$)

$S = \{s, y\}$

$Q = \{t, x, z\}$
Test whether we can improve the shortest path to $x$ found so far by going through $y$.

RELAX($y$, $x$, $w$)

$S = \{s, y\}$

$Q = \{t, x, z\}$
Update $x.d = 14$ and $x.\pi = y$

RELAX($y, x, w$)

$S = \{s, y\}$

$Q = \{t, x, z\}$
Test whether we can improve the shortest path to z found so far by going through y
Update $z.d = 7$ and $z.\pi = y$

\text{RELAX}(y, z, w)
$S = \{s, y\}$
$Q = \{t, x, z\}$
All edges leaving y have been tested.
We are at node $z$

$z = \text{EXTRACT-MIN}(Q)$

$S = \{s, y, z\}$

$Q = \{t, x\}$
Test whether we can improve the shortest path to $s$ found so far by going through $z$. 

RELAX($z, s, w$) 
$S = \{s, y, z\}$ 
$Q = \{t, x\}$
Test whether we can improve the shortest path to x found so far by going through z
Update $x.d = 13$ and $x.\pi = z$

RELAX($z, x, w$)
$S = \{s, y, z\}$
$Q = \{t, x\}$
All edges leaving $z$ have been tested.

$S = \{s, y, z\}$

$Q = \{t, x\}$
We are at node t

t=EXTRACT-MIN(Q)

S = \{s, y, z, t\}

Q = \{x\}
Test whether we can improve the shortest path to $y$ found so far by going through $t$. 

$$RELAX(t, y, w)$$

$S = \{s, y, z, t\}$

$Q = \{x\}$
Test whether we can improve the shortest path to x found so far by going through t
Update $x.d = 9$ and $x.\pi = t$

\begin{align*}
\text{RELAX}(t, x, w) \\
S &= \{s, y, z, t\} \\
Q &= \{x\}
\end{align*}
All edges leaving t have been tested

\[ S = \{s, y, z, t\} \]
\[ Q = \{x\} \]
We are at node \( x \)

\[ x = \text{EXTRACT-MIN}(Q) \]
\[ S = G.V \]
\[ Q = \emptyset \]
Test whether we can improve the shortest path to \( z \) found so far by going through \( x \)
All edges leaving x have been tested.
Every vertex’s shortest path from s has been determined. We are done.
Dijkstra's Algorithm

- correctness proof in the book
  - idea: proof that for each SP, there is a relaxation sequence of its edges in path-order

- Running Time depends on implementation of queue operations
  - $|V|$ * extract-min
  - $|E|$ * decrease key (at relaxation)

- Total
  - $O(V^*T_{\text{extract-min}} + E^*T_{\text{decrease-key}})$
  - with Fibonacci heaps: extract-min is $O(\log V)$ and decrease-key is $O(1)$; total $O(E + V \log V)$

**Dijkstra**($G, w, s$)

1. **Initialize-Single-Source**($G, s$)
2. $S = \emptyset$
3. $Q = G.V$
4. while $Q \neq \emptyset$
5.  
6.  
7.  
8.  

all edges from u

Dijkstra algorithm implementation:

1. Initialize $S$ to the empty set.
2. Set $Q$ to the set of all vertices.
3. While $Q$ is not empty:
   - Extract the minimum vertex $u$ from $Q$.
   - Add $u$ to $S$.
   - For each neighbor $v$ of $u$ in $G$:
     - Relax the edge from $u$ to $v$. 

**Dijkstra**($G, w, s$)

1. **Initialize-Single-Source**($G, s$)
2. $S = \emptyset$
3. $Q = G.V$
4. while $Q \neq \emptyset$
5.  
6.  
7.  
8.  

all edges from u

Dijkstra algorithm implementation:

1. Initialize $S$ to the empty set.
2. Set $Q$ to the set of all vertices.
3. While $Q$ is not empty:
   - Extract the minimum vertex $u$ from $Q$.
   - Add $u$ to $S$.
   - For each neighbor $v$ of $u$ in $G$:
     - Relax the edge from $u$ to $v$. 

**Dijkstra**($G, w, s$)

1. **Initialize-Single-Source**($G, s$)
2. $S = \emptyset$
3. $Q = G.V$
4. while $Q \neq \emptyset$
5.  
6.  
7.  
8.  

all edges from u

Dijkstra algorithm implementation:

1. Initialize $S$ to the empty set.
2. Set $Q$ to the set of all vertices.
3. While $Q$ is not empty:
   - Extract the minimum vertex $u$ from $Q$.
   - Add $u$ to $S$.
   - For each neighbor $v$ of $u$ in $G$:
     - Relax the edge from $u$ to $v$. 

Graphs II - Shortest paths

Lesson 2: All Sources Shortest Paths
Task: find all shortest paths, between any two vertices (no fixed source)

Slow: run Bellman Ford separately from each vertex as source.

- running time $|V| \times \text{BF-time} = V \times O(VE) = O(V^2E)$
- that is $O(V^4)$ if graph dense $E \approx V^2$
Instead, we will use dynamic programming.

\( C_{ij} = \min \text{ SP weight (objective) between vertices } i,j \)

**optimal solution structure:**

- If path \( P(i\rightarrow j) \) from \( i \) to \( j \) in optimal and passes vertex \( k \), then the subpaths \( P(i\rightarrow k) \) and \( P(k\rightarrow j) \) must be also optimal.

- Optimal = shortest.
ASSP dynamic programming

- two options for dynamic programming
  - A. go by the number of edges used in a path
    - $C_{ij}^{(m)}$: minimum path weight between $i$ and $j$ using at most $m$ edges
    - $C_{ij}^{(1)}$: weight of edge $i \rightarrow j$, if exists (one edge)
    - $C_{ij}^{(2)}$: min weight of any path $i \rightarrow k \rightarrow j$ (max 2 edges)
    - $C_{ij}^{(0)}$: we 0 if $i \neq j$, $\infty$ otherwise (no edge)
  - B. by the intermediary nodes in a certain fixed order
    - fix order of all vertices 1,2,3,...,$|V|$ 
    - $C_{ij}^{(m)}$: minimum path weight between $i$ and $j$ using only intermediary vertices $\{1,2,...,m\}$
    - similar to discrete knapsack idea, see module 6
ASSP dynamic programming by edges

\[ C_{ij}^{(m)} = \min_k \{ C_{ij}^{(m-1)}, C_{ik}^{(m-1)} + w_{kj} \} \]  //bottom up computation

- the \( C_{ij} \) using \( m \) edges is either
  - the same as \( C_{ij} \) using \( m-1 \) edges, OR
  - \( C_{ik} \) using \( m-1 \) edges to intermediary \( k \), plus an edge from \( k \) to \( j \) \( w_{kj} \)
  - all nodes \( k \) are eligible as possible “last” intermediary
ASSP dynamic programming by edges

- Compute the $C^{(m)}$ matrix from $C^{(m-1)}$ matrix using edges matrix $W$

- **Extend-SP** $(C^{(m-1)}, W)$
  ```
  for i=1:n
    for j=1:n
      a=\infty;
      for k=1:n
        a=min\{a, C^{ik^{(m-1)}} + w_{kj}\};
      C^{ij^{(m)}}=a
  ```

- **ASSP-slow(W)**
  ```
  C^{(1)} = W
  for m=2:n-1
    C^{(m)}=Extend-SP(C^{(m-1)}, W)
  return C^{(n-1)}
  ```
ASSP dynamic programming by edges

• Extend-SP looks like matrix multiplication!
  - Extend-SP running time $O(n^3)$

• ASSP-slow is $n \times O(n^3) = O(n^4)$, same as running Bellman Ford separately from each vertex

\begin{align*}
  \text{Extend-SP} \ (C^{(m-1)}, W) & \quad \text{D=multiply}(C,W) \\
  \text{for } i=1:n & \quad \text{for } i=1:n \\
  \quad \text{for } j=1:n & \quad \text{for } j=1:n \\
  \quad \quad \ a=\infty; & \quad \ a=0; \\
  \quad \quad \text{for } k=1:n & \quad \quad \text{for } k=1:n \\
  \quad \quad \quad \ a=\min\{a, C_{ik}^{(m-1)} + w_{kj}\}; & \quad \quad \quad \ a=a+ C_{ik} \times w_{kj}; \\
  \quad \quad \quad \ C_{ij}^{(m)}=a & \quad \quad \quad \ D_{ij}=a
\end{align*}
ASSP dynamic programming by edges

- Think of Extending-SP as of matrix multiplication
  - $C^{(1)} = C^{(0)} \times W = W$; the "\times" means "$a = \min\{a, C_{ik}^{(m-1)} + w_{kj}\}$" inner operation
  - $C^{(2)} = C^{(1)} \times W = W_2$
  - $C^{(3)} = C^{(2)} \times W = W_3$
  - .......

- Only need $C^{(n-1)}$, not the intermediary ones
  - $C^{(1)} = W$
  - $C^{(2)} = W^2 = (W^1)^2$
  - $C^{(4)} = W^4 = (W^2)^2$
  - $C^{(8)} = W^8 = (W^4)^2$, etc
ASSP dynamic programming by edges

- ASSP-fast(W)
  - $C^{(1)} = W$
  - while $m < n-1$
    - $C^{(m)} = \text{Extend-SP}(C^{(m-1)}, C^{(m-1)}, W)$
    - $m = 2 \times m$
  - return $C^{(m)}$

- After $\lceil \log(n) \rceil$ iterations we have computed $C^{(m)}$ with $m \geq n-1$. It's ok to "overshoot" as $C$ doesn't change after finding the SP.

- Running time $\Theta(V^3 \log V)$
ASSP dynamic programming by vertices

- "Floyd-Warshall" algorithm
- Fix a vertex order: 1, 2, 3, ..., n
  - \( S_k \) = set first k of vertices = \( \{v_1, v_2, ..., v_k\} \)
- \( C_{ij}^{(m)} \) = the weight of SP(i,j) going only through intermediary vertices in set \( S_k \)
  
  \( m=0 \): no intermediary allowed; \( C_{ij}^{(0)} = w_{ij} \)
  
  \( m=1 \): only \( k = v_1 \) intermediary allowed
  - \( C_{ij}^{(1)} = \min \{ w_{ij}, w_{ik} + w_{kj} \} \)
ASSP dynamic programming by vertices

- **dynamic recursion**

- \( C_{ij}^{(m)} = \min \{ C_{ij}^{(m-1)}, C_{im}^{(m-1)} + C_{mj}^{(m-1)} \} \)

  \(- C_{ij}^{(m)} = \text{minimum between } C_{ij}^{(m-1)} \text{ and the SP including vertex } v_m \text{ and only other intermediaries } <m. \)
ASSP dynamic programming by vertices

- **bottom up computation**
  - **Floyd-Warshall-ASSP(W)**
    ```
    for m=1:n
      for i=1:n
        for j=1:n
          C_{ij}^{(m)} = min\{ c_{ij}^{(m-1)}, c_{im}^{(m-1)} + c_{mj}^{(m-1)} \}
    
    return C^{(n)}
    ```

- **Running time** $\Theta(V^3)$
  - for dense graphs $E \approx V^2$, Floyd-Warshall-ASSP same cost as Bellman-Ford-SSSP