CH08-320201

Algorithms and Data Structures

ADS

Lecture 24

Dr. Kinga Lipskoch

Spring 2019

Complexity Analysis

$|V|$ times:

while $Q \neq \emptyset$
 do $u \leftarrow \textsc{Extract-Min}(Q)$
  $S \leftarrow S \cup \{u\}$

$degree(u)$ times:

  for each $v \in Adj[u]$
   do if $d[v] > d[u] + w(u, v)$
    then $d[v] \leftarrow d[u] + w(u, v)$

- Similar to Prim's minimum spanning tree algorithm, we get the computation time

$$\Theta(V \cdot T_{\textsc{Extract-Min}} + E \cdot T_{\textsc{Decrease-Key}})$$

- Hence, depending on what data structure we use, we get the same computation times as for Prim's algorithm.

Unweighted Graphs

- Suppose that we have an unweighted graph, i.e., the weights $w(u, v) = 1$ for all $(u, v) \in E$.
- Can we improve the performance of Dijkstra's algorithm?
- Observation: The vertices in our data structure Q are processed following the FIFO principle.
- Hence, we can replace the min-priority queue with a queue.
- This leads to a breadth-first search.

BFS Algorithm

```
d[s] := 0
for each v ϵ V\{s}
  d[v] := infinity
Enqueue (Q,s)
while Q != Ø
  u := Dequeue(Q)
  for each v ϵ Adj[u]
      if d[v] = infinity
      then d[v] := d[u] + 1
           pi[v] :=u
           Enqueue(Q,v)
```

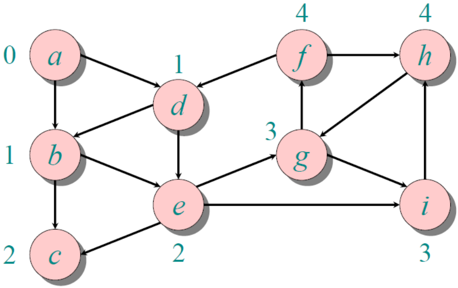
Analysis: BFS Algorithm

Correctness:

- The FIFO queue Q mimics the min-priority queue in Dijkstra's algorithm.
- Invariant:
  If v follows u in Q, then $d[v] = d[u]$ or $d[v] = d[u] + 1$.
- Hence, we always dequeue the vertex with smallest d.

Time complexity:
$O(|V|T_{Dequeue} + |E|T_{Enqueue}) = O(|V| + |E|)$

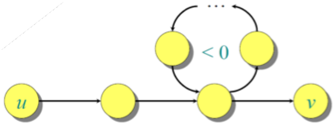
Example: BFS Algorithm



Q: a b d c e g i f h

Negative Weights

- We had postulated that all weights are nonnegative.
- How can we extend the algorithm to also handle negative entries?
- The problems are caused by negative weight cycles.



- Goal: Find shortest-path lengths from a source vertex $s \in V$ to all vertices $v \in V$ or determine the existence of a negative-weight cycle.

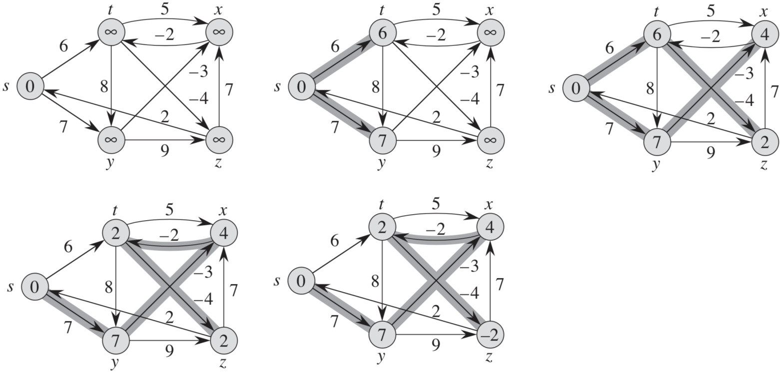
Bellmann-Ford Algorithm

```
d[s] := 0
for each v ∊ V\{s}
  d[v] := infinity
for i:=1 to |V|-1
    for each (u,v) ∊ E
        if d[v] > d[u] + w(u,v)
        then  d[v] := d[u] + w(u,v)
              pi[v] :=u

for each (u,v) ∊ E
  if d[v] > d[u] + w(u,v)
    report existence of negative-weight cycle
```

Time complexity: $O(|V| \cdot |E|)$

Example: Bellman-Ford Algorithm

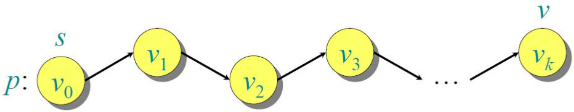



```
for i:=1 to |V|-1
    for each (u,v) ϵ E
        if d[v] > d[u] + w(u,v)
        then  d[v] := d[u] + w(u,v)
              pi[v] :=u
```

Bellmann-Ford Algorithm: Correctness (1)

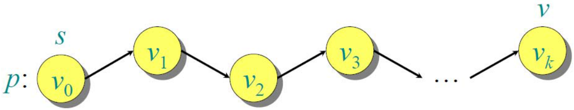
Theorem:
If $G = (V, E)$ contains no negative-weight cycles, then the Bellman-Ford algorithm terminates with $d[v] = \delta(s, v)$ for all $v \in V$.

Proof:
Let $v \in V$ be any vertex.
Consider a shortest path $p = (v_0, ..., v_k)$ from s to v.
Then, $\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i)$ for $i = 1, ..., k$.



Bellmann-Ford Algorithm: Correctness (2)

Initially, $d[v_0] = 0 = \delta(s, v_0)$.
According to our Lemma from Dijkstra's algorithm we have $d[v] \geq \delta(s, v)$, i.e., $d[v_0]$ is not changed.
After the 1st pass, we have $d[v_1] = \delta(s, v_1)$.
After the 2nd pass, we have $d[v_2] = \delta(s, v_2)$.
...
After the k^{th} pass, we have $d[v_k] = \delta(s, v_k)$.
Since G has no negative-weight cycles, p is a simple path, i.e., it has $\leq |V| - 1$ edges.

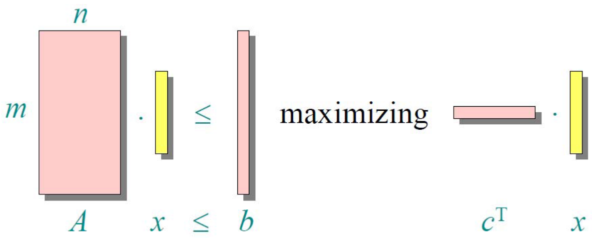


Detecting Negative-Weight Cycles

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

Excurse: Linear Programming

Linear programming problem:
Let A be matrix of size $m \times n$, b a vector of size m, and c a vector of size n.
Find a vector x of size n that maximizes $c^T x$ subject to $Ax \leq b$, or determine that no such solution exists.



Example: Difference Constraints

Linear programming example, where each row of A contains exactly one 1 and one -1, other entries are 0.

$$\left.\begin{aligned} x_1 - x_2 &\le 3 \\ x_2 - x_3 &\le -2 \\ x_1 - x_3 &\le 2 \end{aligned}\right\} \quad x_j - x_i \le w_{ij}$$

Goal: Find 3-vector x that satisfies these inequations.
Solution: $x_1 = 3$, $x_2 = 0$, $x_3 = 2$.
Build constraint graph (matrix A of size $|E| \times |V|$):

$$x_j - x_i \le w_{ij}$$

Case 1: Unsatisfiable Constraints

Theorem:
If the constraint graph contains a negative-weight cycle, then the constraints are unsatisfiable.
Proof:
Suppose we have a negative-weight cycle:

$$v_1 \to v_2 \to \cdots \to v_k \to v_1.$$

Then,

$$\begin{aligned} x_2 - x_1 &\le w_{12} \\ x_3 - x_2 &\le w_{23} \\ &\vdots \\ x_k - x_{k-1} &\le w_{k-1,k} \\ x_1 - x_k &\le w_{k1} \end{aligned}$$

Summing the inequations delivers: $LHS = 0$, $RHS < 0$.
Hence, no x exists that satisfies the inequations.

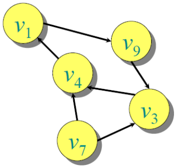
Case 2: Satisfiable Constraints (1)

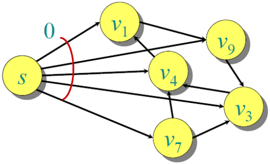
Theorem:

If no negative-weight cycle exists in the constraint graph, then the constraints are satisfiable.

Proof:

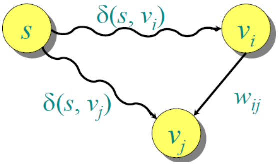
Add a vertex s with a 0-weight edge to all vertices. Note that this does not introduce a negative-weight cycle.





Case 2: Satisfiable Constraints (2)

Show that the assignments $x_i = \delta(s, v_i)$ for $i = 1, \ldots, n$ solve the constraints.
Consider any constraint $x_j - x_i \leq w_{ij}$.
Then, consider the shortest path from s to v_j and v_i.
The triangle inequality delivers $\delta(s, v_j) \leq \delta(s, v_i) + w_{ij}$.
Since $x_i = \delta(s, v_i)$ and $x_j = \delta(s, v_j)$, constraint $x_j - x_i \leq w_{ij}$ is satisfied.



Bellmann-Ford for Linear Programming

Corollary:
The Bellman-Ford algorithm can solve a system of m difference constraints on n variables in $O(m \cdot n)$ time.

Remark:
Single-source shortest paths is a simple linear programming problem.

All-Pairs Shortest Paths

Problem:

- So far, we considered the (single-source) shortest paths problem of finding the shortest paths from a source vertex $s \in V$.
- Now, we would like to extend this to finding all-pairs shortest paths.
- The input is, again, a directed graph $G = (V, E)$ with an edge-weight function $w : E \to \mathbb{R}$.
- Let $V = \{1, ..., n\}$.
- The output shall be an $n \times n$-matrix of shortest-path lengths $\delta(i, j)$ for all $i, j \in V$.

Use Single-Source Shortest Paths

- Idea:
  Run the single-source shortest paths algorithm for each vertex $s \in V$ being the source once.
- Dijkstra's algorithm (for non-negative weights):
  Computation time $= O(|V| \cdot (|E| + |V|) \cdot lg(|V|))$ [min-heap]
  Worst-case $= \Theta(|V|^3 \cdot lg(|V|))$
- Bellman-Ford algorithm (for general case):
  Computation time $= O(|V|^2 \cdot |E|)$
  Worst-case $= \Theta(|V|^4)$

Dynamic Programming for All-Pairs Shortest Paths (1)

Consider the substructure:
$d_{ij}^{(m)}$ = weight of a shortest path
from i to j that uses at most m edges.

Theorem:

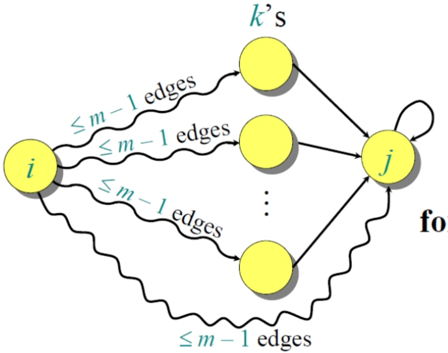
- Initially ($m = 0$), we have

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j; \end{cases}$$

- Then, for $m = 1, \ldots, n - 1$, we have
  $d_{ij}^{(m)} = \min_k \{d_{ik}^{(m-1)} + a_{kj}\}$
  where $A = (a_{ij})$ is the adjacency matrix

Dynamic Programming for All-Pairs Shortest Paths (2)

Proof:
$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$



for $k \leftarrow 1$ **to** n
do if $d_{ij} > d_{ik} + a_{kj}$
then $d_{ij} \leftarrow d_{ik} + a_{kj}$

Remark

- The dynamic programming strategy is to start with $m = 0$ and successively increase m until we reach $n - 1$.
- If we have no negative-weights cycles, we are done after $n - 1$ steps, i.e.,
  $\delta(i,j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots$

Implementation (1)

- The expression $d_{ij}^{(m)} = \min_k\{d_{ik}^{(m-1)} + a_{kj}\}$ updates all entries of the $n \times n$-matrix $D^{(m)} = (d_{ij}^{(m)})$ from the $n \times n$-matrices $D^{(m-1)}$ and A.
- We can use a matrix multiplication notation $D^{(m)} = D^{(m-1)} \cdot A$, where the typical operations "$+$" and "$\cdot$" are mapped to the operations "min" and "$+$".
- $D^{(0)}$ is the respective identity matrix

$$I = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^{(0)} = (d_{ij}^{(0)})$$

Implementation (2)

- The introduced matrix multiplication is associative and it can be shown that it forms a closed semi-ring (assuming real numbers).
- Hence, the dynamic programming algorithm executes the following computation steps:
  $D^{(1)} = D^{(0)} \cdot A = A^1$
  $D^{(2)} = D^{(1)} \cdot A = A^2$
  $\dots$
  $D^{(n-1)} = D^{(n-2)} \cdot A = A^{n-1}$
  where the result is stored in
  $D^{(n-1)} = (\delta(i,j))$

Analysis

- Since we are executing $n-1$ matrix multiplications for matrices of size $n \times n$, the computation time is $\Theta(n \cdot n^3) = \Theta(n^4)$.
- Since $n = |V|$, this is not better than running n times the Bellman-Ford algorithm.
- However, we can exploit the generalized power-of-a-number recursion, which reduces the time complexity to $\Theta(n^3 \cdot \lg n)$.
- Note that n does not need to be a power of 2, as $A^{n-1} = A^n = A^{n+1} = \dots$

Summary

- Directed and undirected graphs
- Adjacency matrix vs. adjacency lists
- Graph search: BFS or DFS in $\Theta(|V| + |E|)$
- MST: Prim in $O(|E| \lg(|V|))$ for min-heap
- Single-source Shortest Paths:
  - Dijkstra for non-negative weights in $O((|V| + |E|) \lg(|V|))$ for min-heap
  - BFS for non-weighted edges in $\Theta(|V| + |E|)$
  - Bellman-Ford for all cases in $\Theta(|V| \cdot |E|)$