Announcements

- Midterm 1 officially postponed to Oct 15'th, 9 am to 11 pm
  - Should take you about 2.5 hours to complete
  - No class neither office hours on that day
  - Will cover all materials until today's: D-and-C, DP, Greedy, MST and shortest paths

- HW 2 due next Monday, 6 pm

- Additional OH tomorrow at 3 pm by Fan; Also Jibang changed his Monday OH to Sunday night

- HW3 will be out this weekend or early next week

CS6161: Design and Analysis of Algorithms (Fall 2020)

Shortest Paths with Negative Weights

Instructor: Haifeng Xu

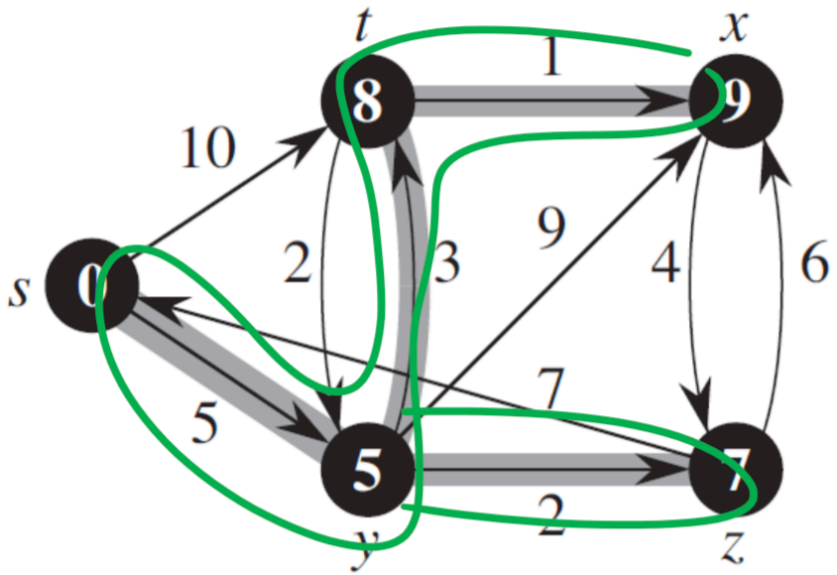
Outline

Shortest Paths with Negative Weights

- Bellman-Ford for Single-Source Shortest Path
- Floyd-Warshall for All-Pair Shortest Path

Recall Shortest Paths Problems

- Last lecture: Dijkstra's Algorithm for single-source shortest paths
  - Again, we will focus on weighted directed graph
- Dijkstra's algorithm fails when there are negative weights
  - Example? HW exercise



Recall Shortest Paths Problems

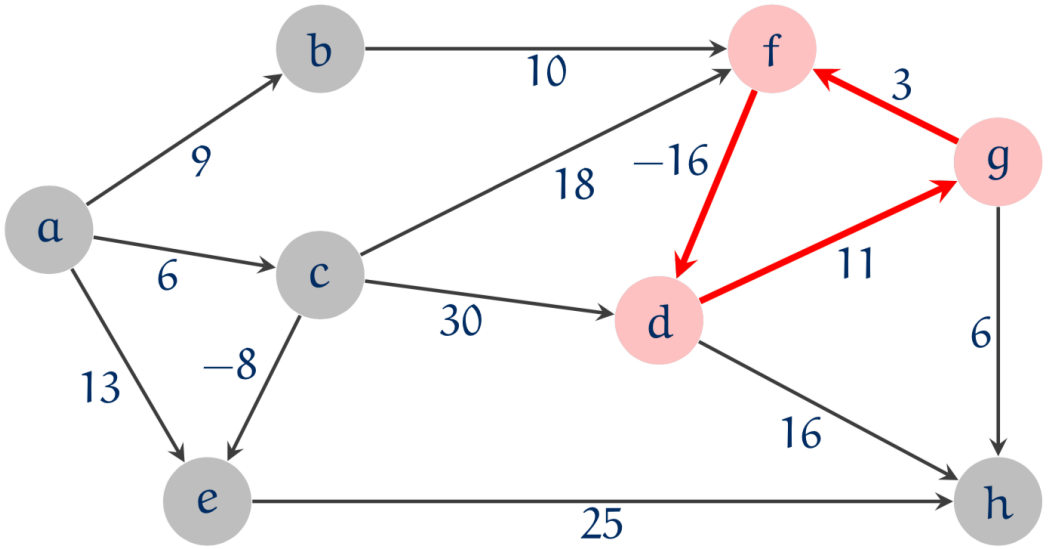
- Last lecture: Dijkstra's Algorithm for single-source shortest paths
  - Again, we will focus on weighted directed graph
- Dijkstra's algorithm fails when there are negative weights
  - Example? HW exercise

Why does Dijkstra's Algorithm Fail?

- It relies on property that if $(v_0, v_1, \cdots, v_k)$ is a shortest path from v_0 to v_k, then $dist(v_0, v_i) \leq dist(v_0, v_k)$ for any $0 < i < k$
- Holds true only for non-negative edge weights

Negative Cycles

- A negative cycle is a directed cycle such that the sum of its edge weights is negative



Shortest Paths and Negative Cycles

Lemma 1: If some path from s to t contains a negative cycle, then there does not exist a shortest path from s to t.

Proof: If there is such a cycle, can build a $s \rightsquigarrow t$ path of arbitrarily negative weight by detouring around the cycle as many times as desired.

Lemma 2: If G has no negative cycles, then there exists a shortest path from s to t that is simple.

Recall: a path is simple if it does not have any cycle

Shortest Paths and Negative Cycles

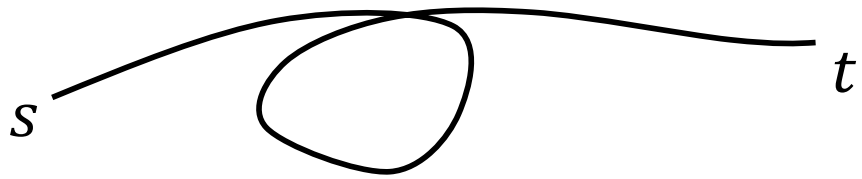
Lemma 1: If some path from s to t contains a negative cycle, then there does not exist a shortest path from s to t.

Proof: If there is such a cycle, can build a $s \rightsquigarrow t$ path of arbitrarily negative weight by detouring around the cycle as many times as desired.

Lemma 2: If G has no negative cycles, then there exists a shortest path from s to t that is simple.

Proof:

- Consider a shortest $s \rightsquigarrow t$ path P that contains a cycle C
- Can remove portion of P corresponding to C without increasing the total cost.



Shortest Path and Negative Cycle Problems

Negative Cycle Problem

Given directed weighted graph $G = (V, E)$ with arbitrary edge weights, find a negative cycle or asserts G does not have one.

Shortest Path Problem

Given directed weighted graph $G = (V, E)$ with arbitrary edge weights and no negative cycles, find a shortest path from s to t

How to Deal with Negative Weights?

Dynamic Programming!

- Sub-problem idea: paths of fewer edges.
- $d(v,k)$: shortest path from s to v using at most k edges
  - $dist(s,v) = d(v, n-1)$ (simple paths has length at most $n-1$)

Recursion for $d(v,k)$

1) If shortest path P uses $\leq k-1$ edges: $d(v,k) = d(v,k-1)$.
2) If shortest path P uses exactly k edes: let (u,v) be the last edge, then P selects best $s \rightsquigarrow u$ path using $\leq k-1$ edges.

$$d(v,k) = \min \begin{cases} d(v,k-1) \\ \min\limits_{u \in Adj(v)} (d(u,k-1) + c(u,v)) \end{cases}$$

Base case: $d(s,0) = 0$ and $d(v,0) = \infty$ for all $v \neq s$.

Bellman-Ford Algorithm

Algorithm: Bellman-Ford(G, s):

foreach $u \in V$ **do**

$d(u, 0) = \infty$;

$d(s, 0) = 0$;

Bellman-Ford Algorithm

Algorithm: Bellman-Ford(G, s):

foreach $u \in V$ **do**

$d(u, 0) = \infty$;

$d(s, 0) = 0$;

for $k = 1$ *to* $n - 1$ **do**

Bellman-Ford Algorithm

Algorithm: Bellman-Ford(G, s):

foreach $u \in V$ **do**
$d(u, 0) = \infty$;
$d(s, 0) = 0$;
for $k = 1$ *to* $n - 1$ **do**
foreach $v \in V$ **do**
$d(v, k) = d(v, k - 1)$;

Bellman-Ford Algorithm

Algorithm: Bellman-Ford(G, s):

foreach $u \in V$ **do**
$d(u, 0) = \infty$;
$d(s, 0) = 0$;
for $k = 1$ *to* $n - 1$ **do**
foreach $v \in V$ **do**
$d(v, k) = d(v, k-1)$;
foreach *edge* $(u, v) \in E$ **do**
$d(v, k) = \min\{d(v, k), d(u, k-1) + c(u, v)\}$;

Bellman-Ford Algorithm

Algorithm: Bellman-Ford(G, s):

foreach $u \in V$ **do**
  $d(u, 0) = \infty$;
$d(s, 0) = 0$;
for $k = 1$ *to* $n - 1$ **do**
  foreach $v \in V$ **do**
    $d(v, k) = d(v, k-1)$;
    foreach *edge* $(u, v) \in E$ **do**
      $d(v, k) = \min\{d(v, k), d(u, k-1) + c(u, v)\}$;
foreach $v \in V$ **do**
  $dist(s, v) = d(v, n-1)$;

Bellman-Ford Algorithm: Time Analysis

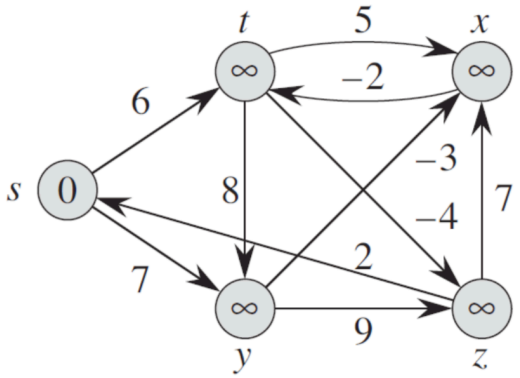
Algorithm: Bellman-Ford(G, s):

foreach $u \in V$ **do**
  $d(u, 0) = \infty$;
$d(s, 0) = 0$;
for $k = 1$ *to* $n - 1$ **do**
  foreach $v \in V$ **do**
    $d(v, k) = d(v, k-1)$;
    foreach *edge* $(u, v) \in E$ **do**
      $d(v, k) = \min\{d(v, k), d(u, k-1) + c(u, v)\}$;
foreach $v \in V$ **do**
  $dist(s, v) = d(v, n-1)$;

Each edge visited once
$O(m)$ time in total

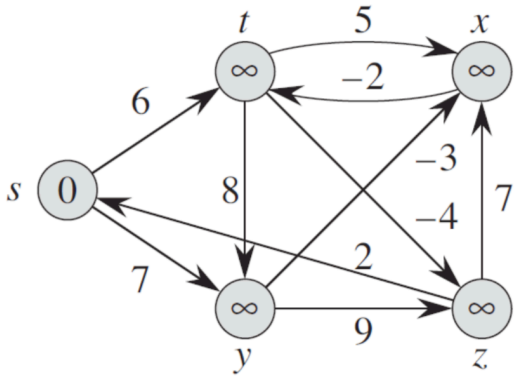
Total time: $O(mn)$

Example Executions

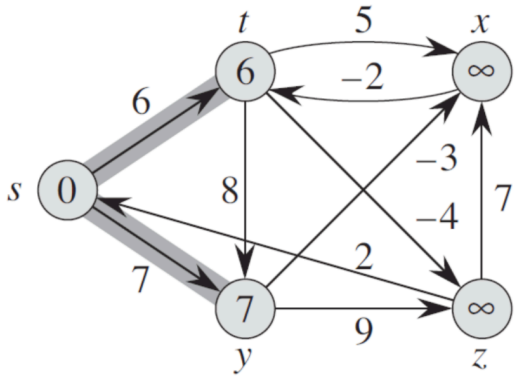


(a)

Example Executions

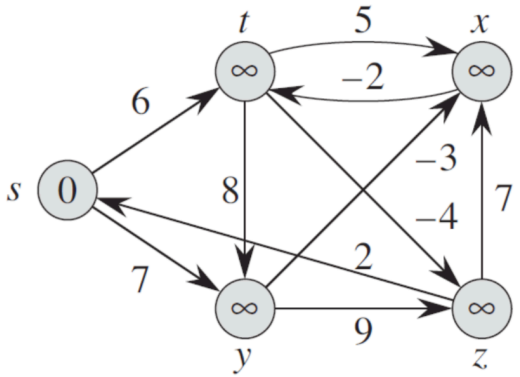


(a)

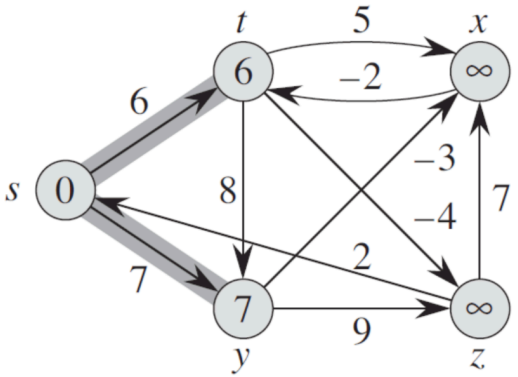


(b)

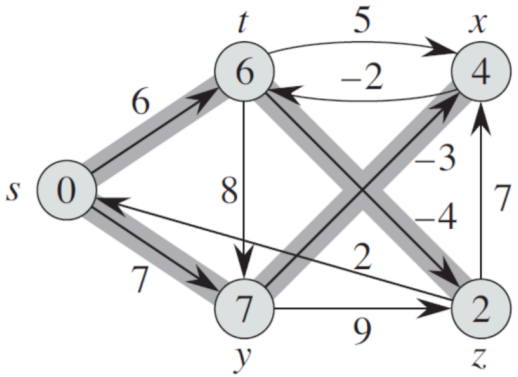
Example Executions



(a)

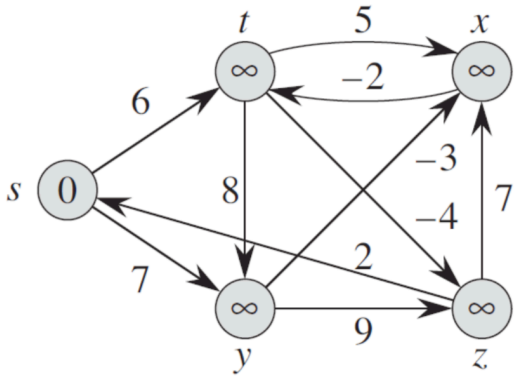


(b)

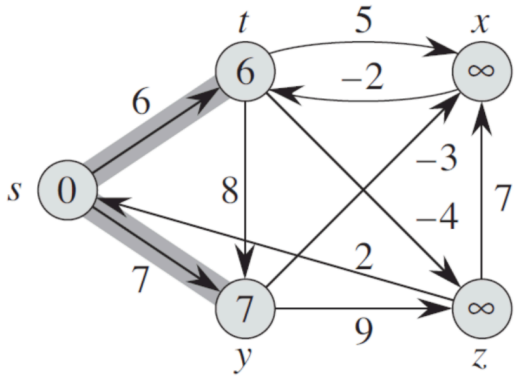


(c)

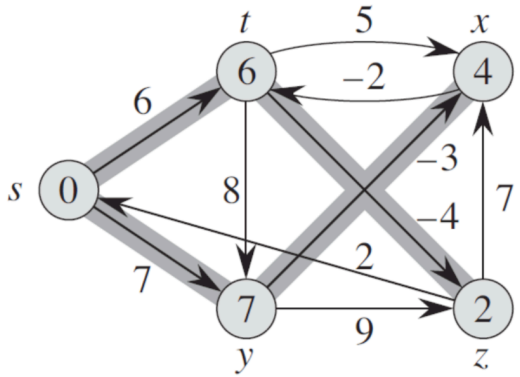
Example Executions



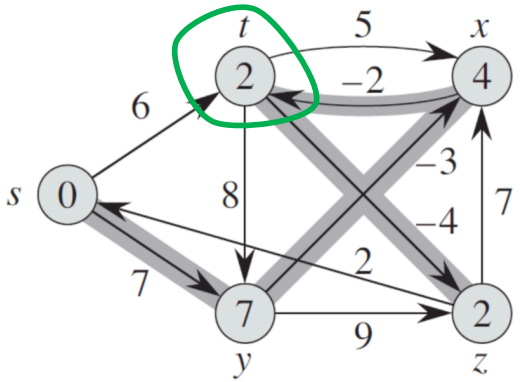
(a)



(b)

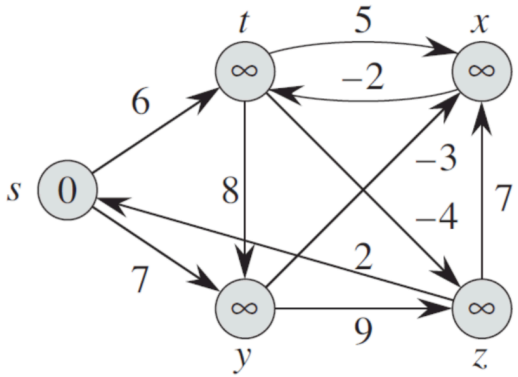


(c)

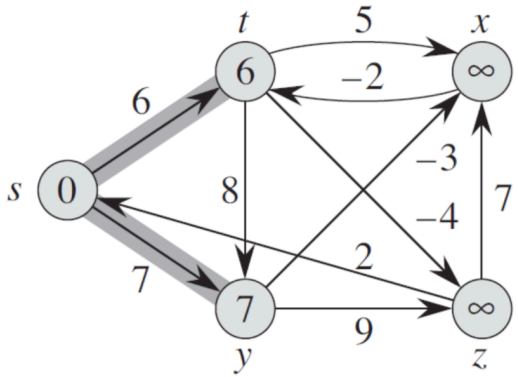


(d)

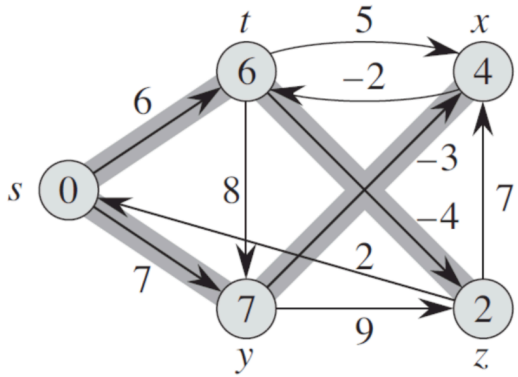
Example Executions



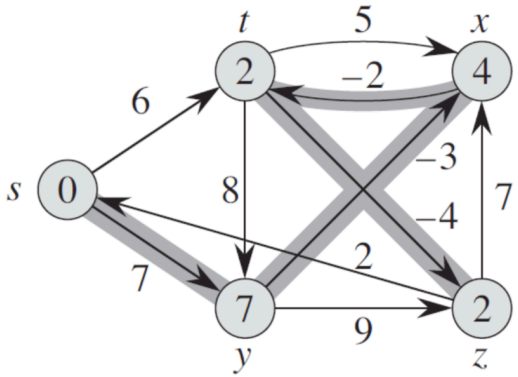
(a)



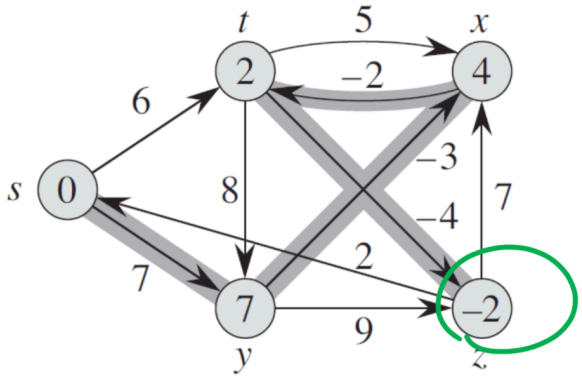
(b)



(c)



(d)



(e)

Bellman-Ford Algorithm: Correctness

- Correct, if graph has no negative cycles
  - By definition of the DP

What if the graph has negative cycles?

Algorithm: Bellman-Ford(G, s):

foreach $u \in V$ **do**
  $d(u, 0) = \infty$;
$d(s, 0) = 0$;
for $k = 1$ *to* $n - 1$ **do**
  foreach $v \in V$ **do**
    $d(v, k) = d(v, k-1)$;
    foreach *edge* $(u, v) \in E$ **do**
      $d(v, k) = \min\{d(v, k), d(u, k-1) + c(u, v)\}$;
foreach $v \in V$ **do**
  $dist(s, v) = d(v, n-1)$;

Bellman-Ford Detects Negative Cycles!

Lemma: G has a negative cycle reachable from s if and only if there exists some vertex v such that $d(v, n) < d(v, n-1)$.

Bellman-Ford Detects Negative Cycles!

Lemma: G has a negative cycle reachable from s if and only if there exists some vertex v such that $d(v,n) < d(v,n-1)$.

Proof: if direction

- Prove the contrapositive: no negative cycle ⇒ $d(v,n) = d(v,n-1)$ for all vertex v
- This is true simply there is a shortest path that is simple with at most $n-1$ edges, so $d(v,n-1)$ already is the shortest length and cannot decrease further

Bellman-Ford Detects Negative Cycles!

Lemma: G has a negative cycle reachable from s if and only if there exists some vertex v such that $d(v,n) < d(v,n-1)$.

Proof: only if direction

- Suppose not. Let $C = v_1 \to v_2 \to \cdots \to v_h \to v_1$ be a negative cycle reachable from s.
- By assumption, there is no change in nth iteration, which means: $d(v_i,n) \le d(v_{i-1},n-1) + c(v_{i-1},v_i)$ for $2 \le i \le h$ and $d(v_1,n) \le d(v_h,n-1) + c(v_h,v_1)$
- Adding up all these h inequalities implies $0 \le c(C)$, contradiction.

$$d(v_1,n) + \sum_{i=2}^{h} d(v_i,n) \le d(v_h,n-1) + c(v_h,v_1) + \sum_{i=2}^{h}[d(v_{i-1},n-1) + c(v_{i-1},v_i)]$$

$$\Rightarrow \quad \sum_{i=1}^{h} d(v_i,n) \le \sum_{i=1}^{h} d(v_i,n-1) + c(C)$$

$$\Rightarrow \quad 0 \le c(C)$$

Bellman-Ford with Negative Cycle Detection

Algorithm: Bellman-Ford(G, s):

```
foreach u ∈ V do
    d(u) = ∞;
d(s) = 0;
for k = 1 to n − 1 do
    foreach v ∈ V do
        foreach edge (u, v) ∈ E do
            d(v) = min{d(v), d(u) + c(u, v)};
/* One more iteration to check if distances change   */
foreach v ∈ V do
    foreach edge (u, v) ∈ E do
        if d(v) > d(u) + c(u, v) then
            return "Negative Cycle"
```

One additional iteration to detect negative cycles

Single-Source Shortest Path Summary

- Dijkstra's algorithm for non-negative edge weights
  - Running time $O(m \log n)$ with binary heaps
  - Best running time $O(m + n \log n)$ with advanced priority queues

- Bellman-Ford algorithm for arbitrary edge weights
  - Running time $O(mn)$

Outline

Shortest Paths with Negative Weights

- Bellman-Ford for Single-Source Shortest Path
- Floyd-Warshall for All-Pair Shortest Path

All-Pair Shortest Path

All-Pair Shortest Path Problem

What if we want to find shortest paths for all pairs of nodes?

Apply single-source algorithms for each node

- **Non-negative weights**: $O(nm + n^2 \log n)$ using advanced priority queues
- **Arbitrary edge weights**: $O(mn^2)$
  - $O(n^4)$ if $m = \Omega(n^2)$

Can we do better?

Yes! Dynamic Programming Again

- $d(i, j, k)$: weights of shortest path from v_i to v_j that uses only vertices $v_1, v_2, \cdots, v_k$ as *intermediate* vertices

Note:

- Different from Bellman-Ford, length of the path does not matter here
- The algorithm will work for arbitrary weights

Yes! Dynamic Programming Again

- $d(i,j,k)$: weights of shortest path from v_i to v_j that uses only vertices $v_1, v_2, \cdots, v_k$ as *intermediate* vertices

$$d(i,j,k) = \min \begin{cases} d(i,j,k-1) \\ d(i,k,k-1) + d(k,j,k-1) \end{cases}$$

Base case: $dist(i,j,0) = c(v_i, v_j)$ (∞ if no edge)

- Correctness: If a shortest path from v_i to v_j goes through v_k, then v_k occurs only once on the path – otherwise there is a negative cycle
- Detect negative cycle: if $dist(k,k,n) < 0$, then there is a negative cycle containing k

Floyd-Warshall Algorithm

Algorithm: Floyd-Warshall(G):

Floyd-Warshall Algorithm

Algorithm: Floyd-Warshall(G):

```
for i = 1 to n do
    for j = 1 to n do
        dist(i, j, 0) = c(v_i, v_j);
```

Floyd-Warshall Algorithm

Algorithm: Floyd-Warshall(G):

```
for i = 1 to n do
    for j = 1 to n do
        dist(i, j, 0) = c(v_i, v_j);
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
```

Floyd-Warshall Algorithm

Algorithm: Floyd-Warshall(G):

for $i = 1$ *to* n **do**
  for $j = 1$ *to* n **do**
    $\text{dist}(i, j, 0) = c(v_i, v_j)$;
for $k = 1$ *to* n **do**
  for $i = 1$ *to* n **do**
    for $j = 1$ *to* n **do**

$$\text{dist}(i, j, k) = \min \begin{cases} \text{dist}(i, j, k-1) \\ \text{dist}(i, k, k-1) + \text{dist}(k, j, k-1) \end{cases}$$

Floyd-Warshall Algorithm

Algorithm: Floyd-Warshall(G):

for $i = 1$ *to* n **do**
  for $j = 1$ *to* n **do**
    $\text{dist}(i, j, 0) = c(v_i, v_j)$;
for $k = 1$ *to* n **do**
  for $i = 1$ *to* n **do**
    for $j = 1$ *to* n **do**

$$\text{dist}(i, j, k) = \min \begin{cases} \text{dist}(i, j, k-1) \\ \text{dist}(i, k, k-1) + \text{dist}(k, j, k-1) \end{cases}$$

for $i = 1$ *to* n **do**
  if $\text{dist}(i, i, n) < 0$ **then**
    return there is a negative cycle in G.

Running time: $O(n^3)$

Floyd-Warshall Algorithm

- Correctness follows from induction
- Simple algorithm without fancy data structures
  - Constant facto is small
- Finding the paths, instead of only path weights?
  - Exercise

Shortest Path Problems Summary

- Single-source shortest paths

Non-negative edge weight	Dijkstra	$O(n \log n + m)$
Arbitrary edge weights	Bellman-Ford	$O(mn)$

- All-Pair shortest paths

Non-negative edge weight	$n\times$Dijkstra	$O(n^2 \log n + mn)$
Arbitrary edge weights	$n\times$Bellman-Ford	$O(mn^2)$
	Floyd-Warshall	$O(n^3)$

End of Midterm 1 Materials

Thank You

Haifeng Xu

University of Virginia

hx4ad@virginia.edu