3-8 single source shortes path

Jun Ma

majun@nju.edu.cn

November 12, 2020

TC 24.1-2

Corollary 24.3

Let $G = (V, E)$ be a weighted, directed graph with source vertex s and weight function $w : E \to \mathbb{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, for each vertex $v \in V$, there is a path from s to v if and only if BELLMAN-FORD terminates with $v.d < \infty$ when it is run on G.

Proof The proof is left as Exercise 24.1-2. ■

$\Rightarrow$

Proof.

By Lemma-24.2.

Lemma 24.2

Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $w : E \to \mathbb{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, after the $|V| - 1$ iterations of the **for** loop of lines 2–4 of BELLMAN-FORD, we have $v.d = \delta(s, v)$ for all vertices v that are reachable from s.

Proof.

Suppose $v.d < \infty$ when BELLMAN-FORD terminates.

- $v.d$ and $v.\pi$ are updated only if $u.d + w(u, v) < v.d$
- once $v.d$ is updated, there must be a path from s to v within in the predecessor subgraph.
  (Can be proved by introduction on the length of path from s to v)

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

TC 24.1-3

Given a weighted, directed graph $G = (V, E)$ with no negative-weight cycles, let m be the maximum over all vertices $v \in V$ of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in $m + 1$ passes, even if m is not known in advance.

BELLMAN-FORD(G, w, s)

```
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

RELAX(u, v, w)

```
1  if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

```
1: procedure BELLMAN-FORD(G, w, s)
2:     INITIALIZE-SINGLE-SOURCE(G, s)
3:     for i = 1 to |G.V| − 1 do
4:         Suc=FALSE
5:         for each edge (u, v) ∈ G.E do
6:             if RELAX(u, v, w) then
7:                 Suc=TRUE
8:         if Suc== FALSE then
9:             return
10: procedure RELAX(u, v, w)
11:     if v.d > u.d + w(u, v) then
12:         v.d = u.d + w(u, v)
13:         return TRUE
14:     return FALSE
```

TC 24.1-4

Modify the Bellman-Ford algorithm so that it sets $v.d$ to $-\infty$ for all vertices v for which there is a negative-weight cycle on some path from the source to v.

```
1: procedure BELLMAN-FORD(G, w, s)
2:     res = TRUE
3:     INITIALIZE-SINGLE-SOURCE(G, s)
4:     for i = 1 to |G.V| − 1 do
5:         for each edge (u, v) ∈ G.E do
6:             RELAX(u, v, w)
7:     for each edge (u, v) ∈ G.E do
8:         if v.d > u.d+w(u, v) then
9:             v.d = −∞
10:            res = FALSE
11:    for each vertex v ∈ G.V do
12:        UPDATE(v)
13:    return res
14: procedure RELAX(u, v, w)
15:    if v.d > u.d + w(u, v) then
16:        v.d = u.d + w(u, v)
```

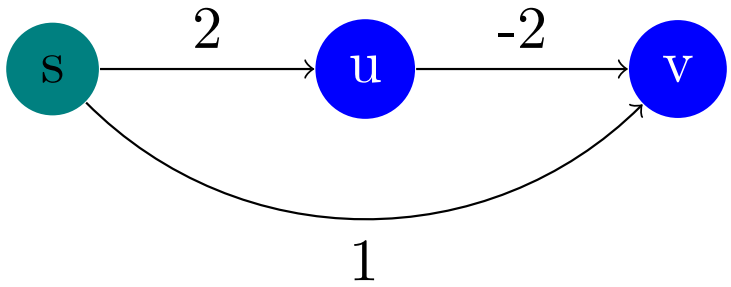
```
1: procedure BELLMAN-FORD(G, w, s)
2:     res = TRUE
3:     INITIALIZE-SINGLE-SOURCE(G, s)
4:     for i = 1 to |G.V| − 1 do
5:         for each edge (u, v) ∈ G.E do
6:             RELAX(u, v, w)
7:     for each edge (u, v) ∈ G.E do
8:         if v.d > u.d+w(u, v) then
9:             v.d = −∞
10:            res = FALSE
11:    for each vertex v ∈ G.V do
12:        UPDATE(v)
13:    return res
14: procedure RELAX(u, v, w)
15:    if v.d > u.d + w(u, v) then
16:        v.d = u.d + w(u, v)
17: procedure UPDATE(v)
18:    if v.d ≠ −∞ and v.π ≠ Nil then
19:        d = UPDATE(v.π)
20:        if d = −∞ then
21:            v.d = d
22:    return v.d
```

TC 24.3-2

Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Why doesn't the proof of Theorem 24.6 go through when negative-weight edges are allowed?

TC 24.3-2

Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Why doesn't the proof of Theorem 24.6 go through when negative-weight edges are allowed?



TC 24.3-4

Professor Gaedel has written a program that he claims implements Dijkstra's algorithm. The program produces $v.d$ and $v.\pi$ for each vertex $v \in V$. Give an $O(V+E)$-time algorithm to check the output of the professor's program. It should determine whether the d and π attributes match those of some shortest-paths tree. You may assume that all edge weights are nonnegative.

- Check $v.d = v.\pi.d + w(v.\pi, v)$ for each v with $v.\pi \neq Nil$

- Check $v.d = v.\pi.d + w(v.\pi, v)$ for each v with $v.\pi \neq Nil$
- Check $s.d = 0$ and $s.\pi = Nil$

- Check $v.d = v.\pi.d + w(v.\pi, v)$ for each v with $v.\pi \neq Nil$
- Check $s.d = 0$ and $s.\pi = Nil$
- Check $v.d = \infty$ for each $v \neq s$ with $v.\pi = Nil$

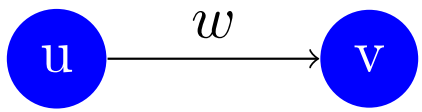
- Check $v.d = v.\pi.d + w(v.\pi, v)$ for each v with $v.\pi \neq Nil$
- Check $s.d = 0$ and $s.\pi = Nil$
- Check $v.d = \infty$ for each $v \neq s$ with $v.\pi = Nil$
- Run Bellman-Ford one round to see if any $v.d$ updated

TC 24.3-7

Let $G = (V, E)$ be a weighted, directed graph with positive weight function $w : E \to \{1, 2, \ldots, W\}$ for some positive integer W, and assume that no two vertices have the same shortest-path weights from source vertex s. Now suppose that we define an unweighted, directed graph $G' = (V \cup V', E')$ by replacing each edge $(u, v) \in E$ with $w(u, v)$ unit-weight edges in series. How many vertices does G' have? Now suppose that we run a breadth-first search on G'. Show that the order in which the breadth-first search of G' colors vertices in V black is the same as the order in which Dijkstra's algorithm extracts the vertices of V from the priority queue when it runs on G.

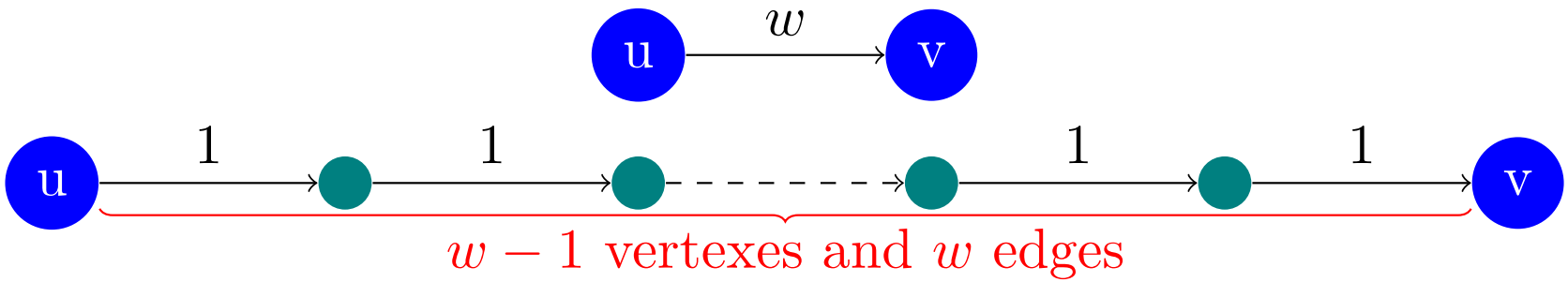
TC 24.3-7

Let $G = (V, E)$ be a weighted, directed graph with positive weight function $w : E \to \{1, 2, \ldots, W\}$ for some positive integer W, and assume that no two vertices have the same shortest-path weights from source vertex s. Now suppose that we define an unweighted, directed graph $G' = (V \cup V', E')$ by replacing each edge $(u, v) \in E$ with $w(u, v)$ unit-weight edges in series. How many vertices does G' have? Now suppose that we run a breadth-first search on G'. Show that the order in which the breadth-first search of G' colors vertices in V black is the same as the order in which Dijkstra's algorithm extracts the vertices of V from the priority queue when it runs on G.



TC 24.3-7

Let $G = (V, E)$ be a weighted, directed graph with positive weight function $w : E \to \{1, 2, \ldots, W\}$ for some positive integer W, and assume that no two vertices have the same shortest-path weights from source vertex s. Now suppose that we define an unweighted, directed graph $G' = (V \cup V', E')$ by replacing each edge $(u, v) \in E$ with $w(u, v)$ unit-weight edges in series. How many vertices does G' have? Now suppose that we run a breadth-first search on G'. Show that the order in which the breadth-first search of G' colors vertices in V black is the same as the order in which Dijkstra's algorithm extracts the vertices of V from the priority queue when it runs on G.

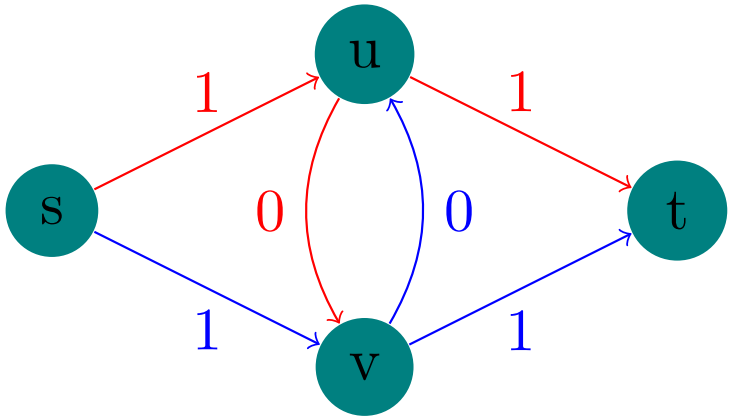


TC 24.5-2

Give an example of a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and source vertex s such that G satisfies the following property: For every edge $(u, v) \in E$, there is a shortest-paths tree rooted at s that contains (u, v) and another shortest-paths tree rooted at s that does not contain (u, v).

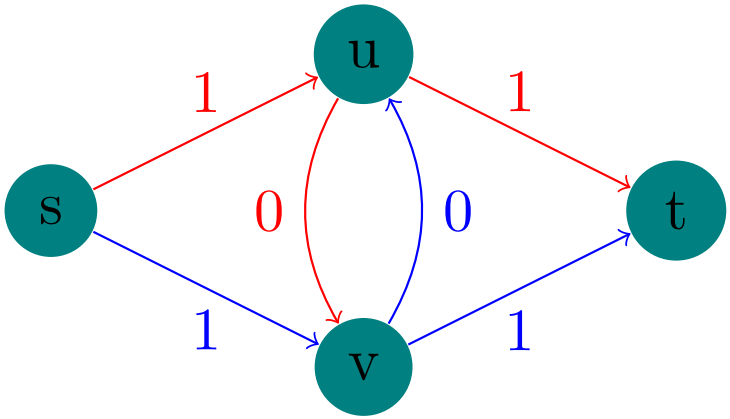
TC 24.5-2

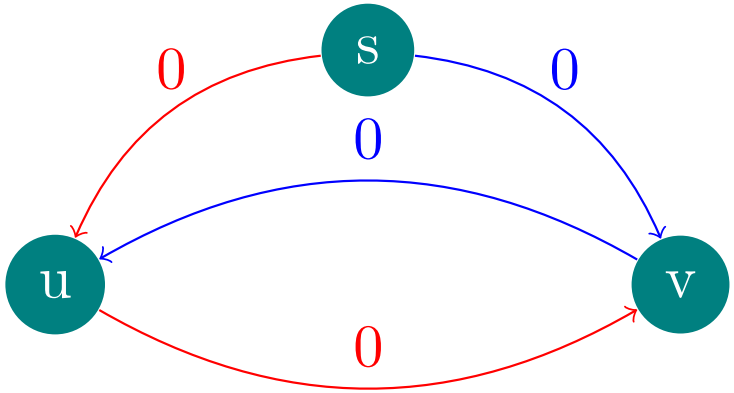
Give an example of a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and source vertex s such that G satisfies the following property: For every edge $(u, v) \in E$, there is a shortest-paths tree rooted at s that contains (u, v) and another shortest-paths tree rooted at s that does not contain (u, v).



TC 24.5-2

Give an example of a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and source vertex s such that G satisfies the following property: For every edge $(u, v) \in E$, there is a shortest-paths tree rooted at s that contains (u, v) and another shortest-paths tree rooted at s that does not contain (u, v).



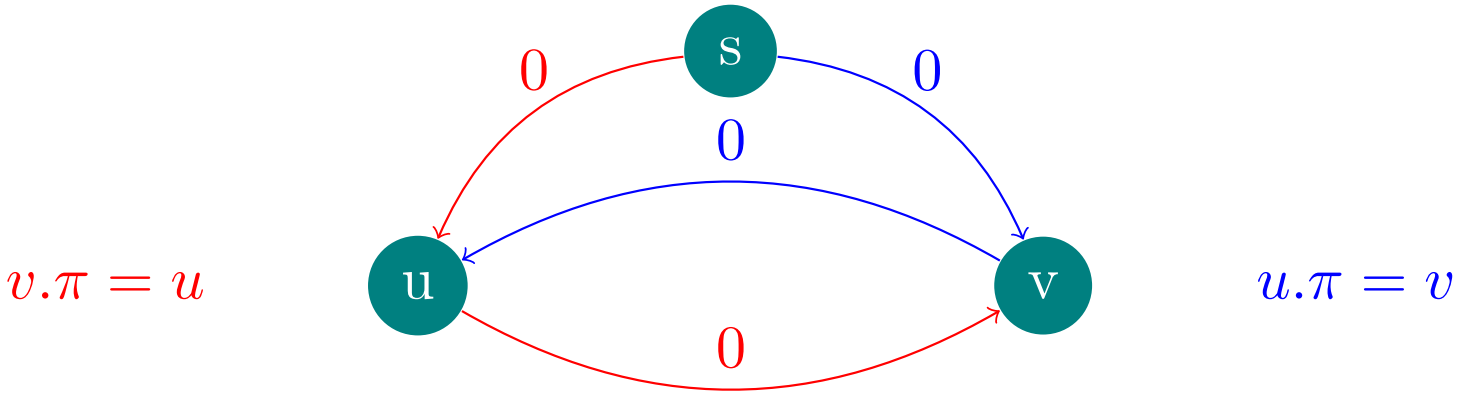


TC 24.5-5

Let $G = (V, E)$ be a weighted, directed graph with no negative-weight edges. Let $s \in V$ be the source vertex, and suppose that we allow $v.\pi$ to be the predecessor of v on *any* shortest path to v from source s if $v \in V - \{s\}$ is reachable from s, and NIL otherwise. Give an example of such a graph G and an assignment of π values that produces a cycle in G_π. (By Lemma 24.16, such an assignment cannot be produced by a sequence of relaxation steps.)

TC 24.5-5

Let $G = (V, E)$ be a weighted, directed graph with no negative-weight edges. Let $s \in V$ be the source vertex, and suppose that we allow $v.\pi$ to be the predecessor of v on *any* shortest path to v from source s if $v \in V - \{s\}$ is reachable from s, and NIL otherwise. Give an example of such a graph G and an assignment of π values that produces a cycle in G_π. (By Lemma 24.16, such an assignment cannot be produced by a sequence of relaxation steps.)



TC Problem 24-2 (Nesting Boxs)

A d-dimensional box with dimensions $(x_1, x_2, \ldots, x_d)$ ***nests*** within another box with dimensions $(y_1, y_2, \ldots, y_d)$ if there exists a permutation π on $\{1, 2, \ldots, d\}$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots, x_{\pi(d)} < y_d$.

a. Argue that the nesting relation is transitive.

b. Describe an efficient method to determine whether or not one d-dimensional box nests inside another.

c. Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

Describe an efficient method to determine whether or not one d-dimensional box nests inside another.

```
1: procedure isInside(B_a, B_b)
2:     d_a ← increasing sequence of dimensions of B_a
3:     d_b ← increasing sequence of dimensions of B_b
4:     for i = 1 to d do
5:         if d_a[i] ≥ d_b[i] then
6:             return FALSE
7:     return TRUE
```

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes.

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes.
- Construct a DAG G.

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes.
- Construct a DAG G.
  - each $v_i \in G.V$ represents a corresponding box B_i

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes.
- Construct a DAG G.
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes.

- Construct a DAG G.
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes.

- Construct a DAG G.
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.
  - Obtain topology order of G.

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes.

- Construct a DAG G.
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.
  - Obtain topology order of G.
  - With the topology order, find the longest path via DP.

$$dist[v] = \max_{(u,v) \in G.E} \{dist[u] + w(u, v)\}$$

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes. $O(nd \lg d)$
- Construct a DAG G.
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.
  - Obtain topology order of G.
  - With the topology order, find the longest path via DP.

$$dist[v] = \max_{(u,v) \in G.E} \{dist[u] + w(u, v)\}$$

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes. $O(nd \lg d)$
- Construct a DAG G. $O(n^2 d)$
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.
  - Obtain topology order of G.
  - With the topology order, find the longest path via DP.

$$dist[v] = \max_{(u,v) \in G.E} \{dist[u] + w(u, v)\}$$

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes. $O(nd \lg d)$
- Construct a DAG G. $O(n^2 d)$
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.
  - Obtain topology order of G. $O(n^2)$
  - With the topology order, find the longest path via DP.

$$dist[v] = \max_{(u,v) \in G.E} \{dist[u] + w(u, v)\}$$

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes. $O(nd \lg d)$
- Construct a DAG G. $O(n^2 d)$
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.
  - Obtain topology order of G. $O(n^2)$
  - With the topology order, find the longest path via DP. $O(n^2)$

$$dist[v] = \max_{(u,v) \in G.E} \{dist[u] + w(u, v)\}$$

Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots, B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots, B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

- Obtain the increasing sequence of dimensions for all boxes. $O(nd \lg d)$
- Construct a DAG G. $O(n^2 d)$
  - each $v_i \in G.V$ represents a corresponding box B_i
  - each edge $\langle v_i, v_j \rangle$ indicates isInside(B_i, B_j)=TRUE.
- Find a longest path in DAG G.$O(n^2)$
  - Obtain topology order of G. $O(n^2)$
  - With the topology order, find the longest path via DP. $O(n^2)$

$$dist[v] = \max_{(u,v) \in G.E} \{dist[u] + w(u, v)\}$$

TC Problem 24-3 (Arbitrage)

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 49 Indian rupees, 1 Indian rupee buys 2 Japanese yen, and 1 Japanese yen buys 0.0107 U.S. dollars. Then, by converting currencies, a trader can start with 1 U.S. dollar and buy $49 \times 2 \times 0.0107 = 1.0486$ U.S. dollars, thus turning a profit of 4.86 percent.

Suppose that we are given n currencies $c_1, c_2, \ldots, c_n$ and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i, j]$ units of currency c_j.

a. Give an efficient algorithm to determine whether or not there exists a sequence of currencies $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ such that

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1 \ .$$

Analyze the running time of your algorithm.

b. Give an efficient algorithm to print out such a sequence if one exists. Analyze the running time of your algorithm.

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

$$\frac{1}{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]} < 1$$

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

$$\frac{1}{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]} < 1$$

$$\lg \frac{1}{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]} < 0$$

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

$$\frac{1}{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]} < 1$$

$$\lg \frac{1}{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]} < 0$$

$$\lg \frac{1}{R[i_1, i_2]} + \lg \frac{1}{R[i_2, i_3]} + \cdots + \lg \frac{1}{R[i_{k-1}, i_k]} + \lg \frac{1}{R[i_k, i_1]} < 0$$

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

$$\frac{1}{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]} < 1$$

$$\lg \frac{1}{R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1]} < 0$$

$$\lg \frac{1}{R[i_1, i_2]} + \lg \frac{1}{R[i_2, i_3]} + \cdots + \lg \frac{1}{R[i_{k-1}, i_k]} + \lg \frac{1}{R[i_k, i_1]} < 0$$

- Applying Bellman-Ford to see if there is any negative-cycle

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

1: **procedure** BELLMAN-FORD(G, w, s)
2: INITIALIZE-SINGLE-SOURCE(G, s)
3: **for** $i = 1$ **to** $|G.V| - 1$ **do**
4: **for** each edge $(u, v) \in G.E$ **do**
5: RELAX(u, v, w)
6: $S \leftarrow \emptyset$
7: **for** each edge $(u, v) \in G.E$ **do**
8: **if** $v.d > u.d + w(u, v)$ **then**
9: $S \leftarrow S + \{v\}$
10: RELAX(u, v, w)
11: Let $G'.V = G.V$ and $G'.E = \{(v.\pi, v) | v.\pi \neq Nil\}$
12: pick any $v \in S$
13: Run DFS(G', v) to find a back edge point to x.
14: Print the cycle by following π edges start from x


Thank
You!