CSE 202
Dynamic Programming IV

An induced subgraph of the collaboration graph (with Erdos number at most 2).
Made by Fan Chung Graham and Lincoln Lu in 2002.
Chapter 6
Dynamic Programming
6. Dynamic Programming II

- sequence alignment
- Hirschberg's algorithm
- Bellman-Ford algorithm
- distance vector protocols
- negative cycles in a digraph
String similarity

Q. How similar are two strings?

Ex. occurrence and occurrence.

- 6 mismatches, 1 gap
- 1 mismatch, 1 gap
- 0 mismatches, 3 gaps
Edit distance

**Edit distance.** [Levenshtein 1966, Needleman-Wunsch 1970]

- Gap penalty $\delta$; mismatch penalty $\alpha_{pq}$.
- Cost = sum of gap and mismatch penalties.

$\begin{align*}
\text{cost} &= \delta + \alpha_{CG} + \alpha_{TA} \\
\end{align*}$

**Applications.** Unix diff, speech recognition, computational biology, ...
Sequence alignment

Goal. Given two strings $x_1 x_2 \ldots x_m$ and $y_1 y_2 \ldots y_n$ find min cost alignment.

Def. An alignment $M$ is a set of ordered pairs $x_i - y_j$ such that each item occurs in at most one pair and no crossings.

Def. The cost of an alignment $M$ is:

$$
cost(M) = \sum_{(x_i, y_j) \in M} \alpha_{x_i y_j} + \sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta
$$

An alignment of $CTACCG$ and $TACATG$:

$M = \{ x_2 - y_1, x_3 - y_2, x_4 - y_3, x_5 - y_4, x_6 - y_6 \}$
Sequence alignment: problem structure

Def. $OPT(i, j) = \min$ cost of aligning prefix strings $x_1 \ldots x_i$ and $y_1 \ldots y_j$.

Case 1. $OPT$ matches $x_i - y_j$.
Pay mismatch for $x_i - y_j + \min$ cost of aligning $x_1 \ldots x_{i-1}$ and $y_1 \ldots y_{j-1}$.

Case 2a. $OPT$ leaves $x_i$ unmatched.
Pay gap for $x_i + \min$ cost of aligning $x_1 \ldots x_{i-1}$ and $y_1 \ldots y_j$.

Case 2b. $OPT$ leaves $y_j$ unmatched.
Pay gap for $y_j + \min$ cost of aligning $x_1 \ldots x_i$ and $y_1 \ldots y_{j-1}$.

$$OPT(i, j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \\ i\delta \end{cases} & \text{otherwise} \\ \end{cases}$$
Sequence alignment: algorithm

**SEQUENCE-ALIGNMENT** \((m, n, x_1, \ldots, x_m, y_1, \ldots, y_n, \delta, \alpha)\)

**FOR** \(i = 0 \text{ TO } m\)

\[ M[i, 0] \leftarrow i \delta. \]

**FOR** \(j = 0 \text{ TO } n\)

\[ M[0, j] \leftarrow j \delta. \]

**FOR** \(i = 1 \text{ TO } m\)

**FOR** \(j = 1 \text{ TO } n\)

\[ M[i, j] \leftarrow \min \{ \alpha[x_i, y_j] + M[i-1, j-1], \]
\[ \delta + M[i-1, j], \]
\[ \delta + M[i, j-1] \}. \]

**RETURN** \(M[m, n].\)
**Sequence alignment: analysis**

**Theorem.** The dynamic programming algorithm computes the edit distance (and optimal alignment) of two strings of length $m$ and $n$ in $\Theta(mn)$ time and $\Theta(mn)$ space.

**Pf.**
- Algorithm computes edit distance.
- Can trace back to extract optimal alignment itself. ■

**Q.** Can we avoid using quadratic space?

**A.** Easy to compute optimal value in $O(mn)$ time and $O(m + n)$ space.
- Compute $\text{OPT}(i, \bullet)$ from $\text{OPT}(i – 1, \bullet)$.
- **But,** no longer easy to recover optimal alignment itself.
6. Dynamic Programming II

- sequence alignment
- Hirschberg's algorithm
- Bellman-Ford algorithm
- distance vector protocols
- negative cycles in a digraph

Section 6.7
Sequence alignment in linear space

**Theorem.** There exist an algorithm to find an optimal alignment in $O(mn)$ time and $O(m + n)$ space.

- Clever combination of divide-and-conquer and dynamic programming.
- Inspired by idea of Savitch from complexity theory.

---

**A Linear Space Algorithm for Computing Maximal Common Subsequences**

D.S. Hirschberg
Princeton University

The problem of finding a longest common subsequence of two strings has been solved in quadratic time and space. An algorithm is presented which will solve this problem in quadratic time and in linear space.

Key Words and Phrases: subsequence, longest common subsequence, string correction, editing

CR Categories: 3.63, 3.73, 3.79, 4.22, 5.25
Hirschberg's algorithm

Edit distance graph.

- Let $f(i, j)$ be shortest path from $(0,0)$ to $(i,j)$.
- Lemma: $f(i, j) = OPT(i, j)$ for all $i$ and $j$. 

![Diagram of edit distance graph with nodes and edges labeled with symbols such as $\epsilon$, $\alpha_{x,y}$, and $\delta$.]
Hirschberg's algorithm

Edit distance graph.
- Let \( f(i, j) \) be shortest path from \((0,0)\) to \((i, j)\).
- Lemma: \( f(i, j) = OPT(i,j) \) for all \( i \) and \( j \).

Pf of Lemma. [by strong induction on \( i + j \)]
- Base case: \( f(0, 0) = OPT(0, 0) = 0 \).
- Inductive hypothesis: assume true for all \((i', j')\) with \( i' + j' < i + j \).
- Last edge on shortest path to \((i, j)\) is from \((i - 1, j - 1), (i - 1, j), \) or \((i, j - 1)\).
- Thus,

\[
\begin{align*}
f(i, j) &= \min\{\alpha_{x_i y_j} + f(i - 1, j - 1), \delta + f(i - 1, j), \delta + f(i, j - 1)\} \\
&= \min\{\alpha_{x_i y_j} + OPT(i - 1, j - 1), \delta + OPT(i - 1, j), \delta + OPT(i, j - 1)\} \\
&= OPT(i, j)
\end{align*}
\]
Hirschberg's algorithm

Saving space:
Method #1. To fill a row, we only need the values in previous row. Only need $m \times 2$ array B to hold the values.
Hirschberg's algorithm

Saving space:
Method #1. To fill a row, we only need the values in previous row. Only need $m \times 2$ array $B$ to hold the values.
Method #2. Dynamic programming + Divide and Conquer
Hirschberg’s algorithm

Edit distance graph.
- Let $f(i, j)$ be shortest path from $(0,0)$ to $(i,j)$.
- Lemma: $f(i, j) = OPT(i, j)$ for all $i$ and $j$.
- Can compute $f(\cdot, j)$ for any $j$ in $O(mn)$ time and $O(m + n)$ space.
Hirschberg's algorithm

Observation 1. The cost of the shortest path that uses \((i, j)\) is \(f(i, j) + g(i, j)\).
**Observation 2.** let $q$ be an index that minimizes $f(q, n/2) + g(q, n/2)$. Then, there exists a shortest path from $(0, 0)$ to $(m, n)$ uses $(q, n/2)$. 

![Diagram](image-url)
Hirschberg's algorithm

**Divide.** Find index $q$ that minimizes $f(q, n/2) + g(q, n/2)$; align $x_q$ and $y_{n/2}$.

**Conquer.** Recursively compute optimal alignment in each piece.
Hirschberg's algorithm: running time analysis

**Theorem.** Let $T(m, n) = \max$ running time of Hirschberg's algorithm on strings of length at most $m$ and $n$. Then, $T(m, n) = O(mn)$.

**Pf.** [by induction on $n$]
- $O(mn)$ time to compute $f(\cdot, n/2)$ and $g(\cdot, n/2)$ and find index $q$.
- $T(q, n/2) + T(m – q, n/2)$ time for two recursive calls.
- Choose constant $c$ so that:
  - $T(m, 2) \leq cm$
  - $T(2, n) \leq cn$
  - $T(m, n) \leq cmn + T(q, n/2) + T(m – q, n/2)$
- **Claim.** $T(m, n) \leq 2cmn$.
- Base cases: $m = 2$ or $n = 2$.
- **Inductive hypothesis:** $T(m, n) \leq 2cmn$ for all $(m', n')$ with $m' + n' < m + n$.

\[
T(m, n) \leq T(q, n/2) + T(m – q, n/2) + cmn
\leq 2cq n/2 + 2c(m – q) n/2 + cmn
= cqn + cmn – cqn + cmn
= 2cmn \blacksquare
\]
6. Dynamic Programming II

- sequence alignment
- Hirschberg's algorithm
- Bellman-Ford
- distance vector protocols
- negative cycles in a digraph
Shortest paths

Shortest path problem. Given a digraph $G = (V, E)$, with arbitrary edge weights or costs $c_{vw}$, find cheapest path from node $s$ to node $t$.

![Graph](image)

cost of path = $9 - 3 + 1 + 11 = 18$
**Shortest paths: failed attempts**

**Dijkstra.** Can fail if negative edge weights.

![Graph with labels and weights]

**Reweighting.** Adding a constant to every edge weight can fail.
Shortest paths: failed attempts

**Dijkstra.** Can fail if negative edge weights.

![Graph showing Dijkstra example]

**Reweighting.** Adding a constant to every edge weight can fail.

![Graph showing reweighting example]
**Negative cycles**

Def. A **negative cycle** is a directed cycle such that the sum of its edge weights is negative.

\[
\text{a negative cycle } W : \quad c(W) = \sum_{e \in W} c_e < 0
\]
Shortest paths and negative cycles

**Lemma 1.** If some path from \( v \) to \( t \) contains a negative cycle, then there does not exist a cheapest path from \( v \) to \( t \).

**Pf.** If there exists such a cycle \( W \), then can build a \( v \rightarrow t \) path of arbitrarily negative weight by detouring around cycle as many times as desired. □
Shortest paths and negative cycles

Lemma 2. If $G$ has no negative cycles, then there exists a cheapest path from $v$ to $t$ that is simple (and has $\leq n - 1$ edges).

**Pf.**

- Consider a cheapest $v \rightarrow t$ path $P$ that uses the fewest number of edges.
- If $P$ contains a cycle $W$, can remove portion of $P$ corresponding to $W$ without increasing the cost. □

\[ c(W) \geq 0 \]
**Shortest path and negative cycle problems**

**Shortest path problem.** Given a digraph $G = (V, E)$ with edge weights $c_{vw}$ and no negative cycles, find cheapest $v \rightarrow t$ path for each node $v$.

**Negative cycle problem.** Given a digraph $G = (V, E)$ with edge weights $c_{vw}$, find a negative cycle (if one exists).

![shortest-paths tree](image1.png)  ![negative cycle](image2.png)
Shortest paths: dynamic programming

**Def.** $OPT(i, v) =$ cost of shortest $v \rightarrow t$ path that uses $\leq i$ edges.

- **Case 1:** Cheapest $v \rightarrow t$ path uses $\leq i-1$ edges.
  - $OPT(i, v) = OPT(i-1, v)$

- **Case 2:** Cheapest $v \rightarrow t$ path uses exactly $i$ edges.
  - if $(v, w)$ is first edge, then $OPT$ uses $(v, w)$, and then selects best $w \rightarrow t$ path using $\leq i-1$ edges

\[
OPT(i, v) = \begin{cases} 
\infty & \text{if } i = 0 \\
\min\left\{ OPT(i-1, v), \min_{(v, w) \in E} \left\{ OPT(i-1, w) + c_{vw} \right\} \right\} & \text{otherwise}
\end{cases}
\]

**Observation.** If no negative cycles, $OPT(n-1, v) =$ cost of cheapest $v \rightarrow t$ path.

**Pf.** By Lemma 2, cheapest $v \rightarrow t$ path is simple. □
Shortest paths: implementation

**Shortest-Paths** \((V, E, c, t)\)

**Foreach** node \(v \in V\)

\[M[0, v] \leftarrow \infty.\]

\[M[0, t] \leftarrow 0.\]

**For** \(i = 1\) **to** \(n - 1\)

**Foreach** node \(v \in V\)

\[M[i, v] \leftarrow M[i - 1, v].\]

**Foreach** edge \((v, w) \in E\)

\[M[i, v] \leftarrow \min \{ M[i, v], M[i - 1, w] + c_{vw} \} .\]
Shortest paths: implementation

**Theorem 1.** Given a digraph $G = (V, E)$ with no negative cycles, the dynamic programming algorithm computes the cost of the cheapest $v \to t$ path for each node $v$ in $\Theta(mn)$ time and $\Theta(n^2)$ space.

**Pf.**
- Table requires $\Theta(n^2)$ space.
- Each iteration $i$ takes $\Theta(m)$ time since we examine each edge once. •

**Finding the shortest paths.**
- Approach 1: Maintain a $successor(i, v)$ that points to next node on cheapest $v \to t$ path using at most $i$ edges.
- Approach 2: Compute optimal costs $M[i, v]$ and consider only edges with $M[i, v] = M[i - 1, w] + c_{vw}$. 
Shortest paths: practical improvements

Space optimization. Maintain two 1d arrays (instead of 2d array).

- \( d(v) \) = cost of cheapest \( v \rightarrow t \) path that we have found so far.
- \( \text{successor}(v) \) = next node on a \( v \rightarrow t \) path.

Performance optimization. If \( d(w) \) was not updated in iteration \( i - 1 \), then no reason to consider edges entering \( w \) in iteration \( i \).
**Bellman-Ford: efficient implementation**

**Bellman-Ford** \( (V, E, c, t) \)

**Foreach** node \( v \in V \)

\[ d(v) \leftarrow \infty. \]

\[ \text{successor}(v) \leftarrow \text{null}. \]

\[ d(t) \leftarrow 0. \]

**For** \( i = 1 \text{ to } n - 1 \)

**Foreach** node \( w \in V \)

**If** \( d(w) \) was updated in previous iteration

**Foreach** edge \( (v, w) \in E \)

**If** \( d(v) > d(w) + c_{vw} \)

\[ d(v) \leftarrow d(w) + c_{vw}. \]

\[ \text{successor}(v) \leftarrow w. \]

**If** no \( d(w) \) value changed in iteration \( i \), **Stop**.
Bellman-Ford: analysis

**Claim.** After the $i^{th}$ pass of Bellman Ford, $d(v)$ equals the cost of the cheapest $v \rightarrow t$ path using at most $i$ edges.

**Counterexample.** Claim is false!

If nodes $w$ considered before node $v$, then $d(v) = 3$ after 1 pass.
Bellman-Ford: analysis

**Lemma 3.** Throughout Bellman-Ford algorithm, $d(v)$ is the cost of some $v \rightarrow t$ path; after the $i^{th}$ pass, $d(v)$ is no larger than the cost of the cheapest $v \rightarrow t$ path using $\leq i$ edges.

**Pf.** [by induction on $i$]

- Assume true after $i^{th}$ pass.
- Let $P$ be any $v \rightarrow t$ path with $i + 1$ edges.
- Let $(v, w)$ be first edge on path and let $P'$ be subpath from $w$ to $t$.
- By inductive hypothesis, $d(w) \leq c(P')$ since $P'$ is a $w \rightarrow t$ path with $i$ edges.
- After considering $v$ in pass $i+1$: $d(v) \leq c_{vw} + d(w) \leq c_{vw} + c(P') = c(P)$ •

**Theorem 2.** Given a digraph with no negative cycles, Bellman-Ford computes the costs of the cheapest $v \rightarrow t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

**Pf.** Lemmas 2 + 3. •

can be substantially faster in practice
Bellman-Ford: analysis

Claim. Throughout the Bellman-Ford algorithm, following successor($v$) pointers gives a directed path from $v$ to $t$ of cost $d(v)$.

Counterexample. Claim is false!
- Cost of successor $v \rightarrow t$ path may have strictly lower cost than $d(v)$.

Consider nodes in order: $t$, 1, 2, 3

<table>
<thead>
<tr>
<th>Node</th>
<th>$s$ Value</th>
<th>$d$ Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>$t$</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>$t$</td>
<td>1</td>
</tr>
<tr>
<td>t</td>
<td>$t$</td>
<td>0</td>
</tr>
</tbody>
</table>

Diagram:
- Node 2 to Node 1: cost 10
- Node 1 to Node 3: cost 10
- Node 3 to Node $t$: cost 1

Path from 2 to $t$: $2 \rightarrow 1 \rightarrow 3 \rightarrow t$ with cost 22
Bellman-Ford: analysis

**Claim.** Throughout the Bellman-Ford algorithm, following \( \text{successor}(v) \) pointers gives a directed path from \( v \) to \( t \) of cost \( d(v) \).

**Counterexample.** Claim is false!
- Cost of successor \( v \rightarrow t \) path may have strictly lower cost than \( d(v) \).

Consider nodes in order: \( t, 1, 2, 3 \)
Bellman-Ford: analysis

Claim. Throughout the Bellman Ford algorithm, following $\text{successor}(v)$ pointers gives a directed path from $v$ to $t$ of cost $d(v)$.

Counterexample. Claim is false!
- Cost of successor $v\rightarrow t$ path may have strictly lower cost than $d(v)$.
- Successor graph may have cycles.

consider nodes in order: $t, 1, 2, 3, 4$
Bellman-Ford: analysis

Claim. Throughout the Bellman-Ford algorithm, following $\text{successor}(v)$ pointers gives a directed path from $v$ to $t$ of cost $d(v)$.

Counterexample. Claim is false!
- Cost of successor $v \rightarrow t$ path may have strictly lower cost than $d(v)$.
- Successor graph may have cycles.

Consider nodes in order: $t, 1, 2, 3, 4$
Bellman-Ford: finding the shortest path

Lemma 4. If the successor graph contains a directed cycle $W$, then $W$ is a negative cycle.

Pf.

- If $\text{successor}(v) = w$, we must have $d(v) \geq d(w) + c_{vw}$.
  (LHS and RHS are equal when $\text{successor}(v)$ is set; $d(w)$ can only decrease; $d(v)$ decreases only when $\text{successor}(v)$ is reset)
- Let $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ be the nodes along the cycle $W$.
- Assume that $(v_k, v_1)$ is the last edge added to the successor graph.
- Just prior to that:  
  
  \[
  \begin{align*}
  d(v_1) &\geq d(v_2) + c(v_1, v_2) \\
  d(v_2) &\geq d(v_3) + c(v_2, v_3) \\
  \vdots &\vdots \\
  d(v_{k-1}) &\geq d(v_k) + c(v_{k-1}, v_k) \\
  d(v_k) &> d(v_1) + c(v_k, v_1)
  \end{align*}
  \]

  holds with strict inequality since we are updating $d(v_k)$

- Adding inequalities yields $c(v_1, v_2) + c(v_2, v_3) + \ldots + c(v_{k-1}, v_k) + c(v_k, v_1) < 0$.  

$W$ is a negative cycle
Bellman-Ford: finding the shortest path

Theorem 3. Given a digraph with no negative cycles, Bellman-Ford finds the cheapest $s \to t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

Pf.

• The successor graph cannot have a negative cycle. [Lemma 4]
• Thus, following the successor pointers from $s$ yields a directed path to $t$.
• Let $s = v_1 \to v_2 \to ... \to v_k = t$ be the nodes along this path $P$.
• Upon termination, if $\text{successor}(v) = w$, we must have $d(v) = d(w) + c_{vw}$.

(LHS and RHS are equal when $\text{successor}(v)$ is set; $d(\cdot)$ did not change)

• Thus,
  
  $$
  \begin{align*}
  d(v_1) &= d(v_2) + c(v_1, v_2) \\
  d(v_2) &= d(v_3) + c(v_2, v_3) \\
  &\vdots \\
  d(v_{k-1}) &= d(v_k) + c(v_{k-1}, v_k)
  \end{align*}
  $$

Adding equations yields $d(s) = d(t) + c(v_1, v_2) + c(v_2, v_3) + \ldots + c(v_{k-1}, v_k)$. $\blacksquare$
Section 6.9

6. Dynamic Programming II

- sequence alignment
- Hirschberg's algorithm
- Bellman-Ford
- distance vector protocols
- negative cycles in a digraph
Distance vector protocols

Communication network.
- Node ≈ router.
- Edge ≈ direct communication link.
- Cost of edge ≈ delay on link.

Dijkstra's algorithm. Requires global information of network.

Bellman-Ford. Uses only local knowledge of neighboring nodes.

Synchronization. We don't expect routers to run in lockstep. The order in which each foreach loop executes is not important. Moreover, algorithm still converges even if updates are asynchronous.
Distance vector protocols

Distance vector protocols. ["routing by rumor"]

- Each router maintains a vector of shortest path lengths to every other node (distances) and the first hop on each path (directions).
- Algorithm: each router performs \( n \) separate computations, one for each potential destination node.

**Ex.** RIP, Xerox XNS RIP, Novell's IPX RIP, Cisco's IGRP, DEC's DNA Phase IV, AppleTalk's RTMP.

**Caveat.** Edge costs may change during algorithm (or fail completely).

\[ d(s) = 2 \quad \text{d(v) = 1} \quad \text{d(t) = 0} \]

"counting to infinity"
Path vector protocols

**Link state routing.**

- Each router also stores the entire path.
- Based on Dijkstra's algorithm.
- Avoids "counting-to-infinity" problem and related difficulties.
- Requires significantly more storage.

**Ex.** Border Gateway Protocol (BGP), Open Shortest Path First (OSPF).
6. Dynamic Programming II

- sequence alignment
- Hirschberg's algorithm
- Bellman-Ford
- distance vector protocol
- negative cycles in a digraph

Section 6.10
Detecting negative cycles

**Negative cycle detection problem.** Given a digraph $G = (V, E)$, with edge weights $c_{vw}$, find a negative cycle (if one exists).
Detecting negative cycles: application

Currency conversion. Given $n$ currencies and exchange rates between pairs of currencies, is there an arbitrage opportunity?

Remark. Fastest algorithm very valuable!

$$0.741 \times 1.366 \times .995 = 1.00714497$$
Detecting negative cycles

Lemma 5. If $OPT(n, v) = OPT(n - 1, v)$ for all $v$, then no negative cycle can reach $t$.

**Pf.** Bellman-Ford algorithm. ■

Lemma 6. If $OPT(n, v) < OPT(n - 1, v)$ for some node $v$, then (any) cheapest path from $v$ to $t$ contains a cycle $W$. Moreover $W$ is a negative cycle.

**Pf.** [by contradiction]

- Since $OPT(n, v) < OPT(n - 1, v)$, we know that shortest $v \to t$ path $P$ has exactly $n$ edges.
- By pigeonhole principle, $P$ must contain a directed cycle $W$.
- Deleting $W$ yields a $v \to t$ path with $< n$ edges $\Rightarrow W$ has negative cost. ■

![Diagram of cycle](image-url)
Detecting negative cycles

**Theorem 4.** Can find a negative cycle in $\Theta(mn)$ time and $\Theta(n^2)$ space.

**Pf.**
- Add new node $t$ and connect all nodes to $t$ with 0-cost edge.
- $G$ has a negative cycle iff $G'$ has a negative cycle than can reach $t$.
- If $OPT(n, v) = OPT(n - 1, v)$ for all nodes $v$, then no negative cycles.
- If not, then extract directed cycle from path from $v$ to $t$.
(cycle cannot contain $t$ since no edges leave $t$) □
Detecting negative cycles

**Theorem 5.** Can find a negative cycle in $O(mn)$ time and $O(n)$ extra space.

**Pf.**
- Run Bellman-Ford for $n$ passes (instead of $n - 1$) on modified digraph.
- If no $d(v)$ values updated in pass $n$, then no negative cycles.
- Otherwise, suppose $d(s)$ updated in pass $n$.
- Define $\text{pass}(v) =$ last pass in which $d(v)$ was updated.
- Observe $\text{pass}(s) = n$ and $\text{pass}(\text{successor}(v)) \geq \text{pass}(v) - 1$ for each $v$.
- Following successor pointers, we must eventually repeat a node.
- Lemma 4 $\Rightarrow$ this cycle is a negative cycle. □

**Remark.** See p. 304 for improved version and early termination rule.
(Tarjan's subtree disassembly trick)