ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 4: Algorithms on Graphs
Lecture 10: All Pairs Shortest Paths

Maxim Buzdalov
Saint Petersburg 2016

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers
- Let's use the adjacency matrix representation

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers
- Let's use the adjacency matrix representation
- If edge lengths are non-negative, run the Dijkstra algorithm from each vertex
  - $|V|$ times $O(|V|^2)$, so $O(|V|^3)$ in total

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers
- Let's use the adjacency matrix representation
- If edge lengths are non-negative, run the Dijkstra algorithm from each vertex
  - $|V|$ times $O(|V|^2)$, so $O(|V|^3)$ in total
- But there exists the Floyd-Warshall algorithm, which is:

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers
- Let's use the adjacency matrix representation
- If edge lengths are non-negative, run the Dijkstra algorithm from each vertex
  - $|V|$ times $O(|V|^2)$, so $O(|V|^3)$ in total
- But there exists the Floyd-Warshall algorithm, which is:
  - also $O(|V|^3)$

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers
- Let's use the adjacency matrix representation
- If edge lengths are non-negative, run the Dijkstra algorithm from each vertex
  - $|V|$ times $O(|V|^2)$, so $O(|V|^3)$ in total
- But there exists the Floyd-Warshall algorithm, which is:
  - also $O(|V|^3)$
  - but faster

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers
- Let's use the adjacency matrix representation
- If edge lengths are non-negative, run the Dijkstra algorithm from each vertex
  - $|V|$ times $O(|V|^2)$, so $O(|V|^3)$ in total
- But there exists the Floyd-Warshall algorithm, which is:
  - also $O(|V|^3)$
  - but faster
  - and simpler

Problem: given a graph $G = \langle V, E \rangle$, find shortest paths between all pairs of vertices $v_1, v_2 \in V$

- Even if considering only path lengths, this is $\Theta(|V|^2)$ numbers
- Let's use the adjacency matrix representation
- If edge lengths are non-negative, run the Dijkstra algorithm from each vertex
  - $|V|$ times $O(|V|^2)$, so $O(|V|^3)$ in total
- But there exists the Floyd-Warshall algorithm, which is:
  - also $O(|V|^3)$
  - but faster
  - and simpler
  - and does not require non-negative edge lengths!

Algorithm idea:

Algorithm idea:

- $D_k[i][j]$: the length of the shortest path from i to j using only vertices from $[1; k]$ in between

Algorithm idea:

- $D_k[i][j]$: the length of the shortest path from i to j using only vertices from $[1; k]$ in between
- $D_0[i][j]$: just the contents of the adjacency matrix

Algorithm idea:

- $D_k[i][j]$: the length of the shortest path from i to j using only vertices from $[1; k]$ in between
- $D_0[i][j]$: just the contents of the adjacency matrix
- Assume we computed all $D_{k-1}[i][j]$. Then:

Algorithm idea:

- $D_k[i][j]$: the length of the shortest path from i to j using only vertices from $[1; k]$ in between
- $D_0[i][j]$: just the contents of the adjacency matrix
- Assume we computed all $D_{k-1}[i][j]$. Then:
  - By definition, $D_k[i][j] \leftarrow \min(D_{k-1}[i][j], D_{k-1}[i][k] + D_k[k][k] + D_{k-1}[k][j])$

Algorithm idea:

- $D_k[i][j]$: the length of the shortest path from i to j using only vertices from $[1; k]$ in between
- $D_0[i][j]$: just the contents of the adjacency matrix
- Assume we computed all $D_{k-1}[i][j]$. Then:
  - By definition, $D_k[i][j] \leftarrow \min(D_{k-1}[i][j], D_{k-1}[i][k] + D_k[k][k] + D_{k-1}[k][j])$
  - If $D_{k-1}[k][k] < 0$, then $D_k[k][k] = -\infty$
    - so is $D_k[i][j]$ unless $D_{k-1}[i][k] = \infty$ or $D_{k-1}[k][j] = \infty$

Algorithm idea:

- $D_k[i][j]$: the length of the shortest path from i to j using only vertices from $[1; k]$ in between
- $D_0[i][j]$: just the contents of the adjacency matrix
- Assume we computed all $D_{k-1}[i][j]$. Then:
  - By definition, $D_k[i][j] \leftarrow \min(D_{k-1}[i][j], D_{k-1}[i][k] + D_k[k][k] + D_{k-1}[k][j])$
  - If $D_{k-1}[k][k] < 0$, then $D_k[k][k] = -\infty$
    - so is $D_k[i][j]$ unless $D_{k-1}[i][k] = \infty$ or $D_{k-1}[k][j] = \infty$
  - If $D_{k-1}[k][k] = 0$, then $D_k[i][j] \leftarrow D_{k-1}[i][k] + D_{k-1}[k][j]$
    - enables dynamic programming

Algorithm idea:

- $D_k[i][j]$: the length of the shortest path from i to j using only vertices from $[1; k]$ in between
- $D_0[i][j]$: just the contents of the adjacency matrix
- Assume we computed all $D_{k-1}[i][j]$. Then:
  - By definition, $D_k[i][j] \leftarrow \min(D_{k-1}[i][j], D_{k-1}[i][k] + D_k[k][k] + D_{k-1}[k][j])$
  - If $D_{k-1}[k][k] < 0$, then $D_k[k][k] = -\infty$
    - so is $D_k[i][j]$ unless $D_{k-1}[i][k] = \infty$ or $D_{k-1}[k][j] = \infty$
  - If $D_{k-1}[k][k] = 0$, then $D_k[i][j] \leftarrow D_{k-1}[i][k] + D_{k-1}[k][j]$
    - enables dynamic programming
- It never hurts if we use $D_k[i][k]$ instead of $D_{k-1}[i][k]$, same for $[k][j]$
  - So we can happily use a single $D[][]$ array for the entire computation!

Version for non-negative cycles

```
procedure FloydWarshall(V, E)
    N ← |V|, A ← adjacency matrix of ⟨V, E⟩
    for k from 1 to N do
        for i from 1 to N do
            for j from 1 to N do
                A[i][j] ← min(A[i][j], A[i][k] + A[k][j])
            end for
        end for
    end for
end procedure
```

Version supporting negative cycles

procedure $\textsc{FloydWarshall}(V, E)$
  $N \leftarrow |V|$, $A \leftarrow$ adjacency matrix of $\langle V, E\rangle$
  for k **from** 1 **to** N **do**
    for i **from** 1 **to** N **do**
      if $A[i][i] < 0$ **then** $A[i][i] \leftarrow -\infty$ **end if**
      for j **from** 1 **to** N **do**
        if $i \neq j$ **and** $A[i][k] + A[k][j] < \infty$ **then**
          $A[i][j] \leftarrow \min(A[i][j], A[i][k] + A[k][j])$
        end if
      end for
    end for
  end for
end procedure

How to restore the actual paths with Floyd-Warshall?

- $B_k[i][j]$: what vertex to go in the shortest path from i to j using vertices in $[1; k]$
- Use a single $B[i][j]$ for the entire run, similar to $A[i][j]$
- On update of $A[i][j]$ by $A[i][k] + A[k][j]$, also set $B[i][j]$ to $B[i][k]$

How to restore the actual paths with Floyd-Warshall?

- $B_k[i][j]$: what vertex to go in the shortest path from i to j using vertices in $[1; k]$
- Use a single $B[i][j]$ for the entire run, similar to $A[i][j]$
- On update of $A[i][j]$ by $A[i][k] + A[k][j]$, also set $B[i][j]$ to $B[i][k]$

How to count the number of shortest paths?

- $C_k[i][j]$: number of shortest paths from i to j using vertices in $[1; k]$
- Use a single $C[i][j]$ for the entire run, similar to $A[i][j]$
- When $A[i][j]$ is reset to a new value, set $C[i][j] \leftarrow 0$
- If $A[i][j] = A[i][k] + A[k][j]$, set $C[i][j] \leftarrow C[i][j] + C[i][k] \cdot C[k][j]$

Thank you for listening!

Thank you for listening!

- This was the last lecture of the course

Thank you for listening!

- This was the last lecture of the course
- Good luck!

Thank you for listening!

- This was the last lecture of the course
- Good luck!
  - On the course exam

Thank you for listening!

- This was the last lecture of the course
- Good luck!
  - On the course exam
  - In programming competitions

Thank you for listening!

- This was the last lecture of the course
- Good luck!
  - On the course exam
  - In programming competitions
  - . . . and wishing you **Correct**, **Efficient** and **Happy** Programming!

- This was the last lecture of the course
- Good luck!
  - On the course exam
  - In programming competitions
  - ...and wishing you **Correct**, **Efficient** and **Happy** Programming!

The course team