INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

DP 3: All-pairs Shortest Paths

Johannes Kapfhammer

11 November 2019

All-pairs shortest path

Weighted graph V, E with $|V| = N$
Weights correspond to lengths ≥ 0.
What is the shortest path from u to v? (for all $u, v \in V$)

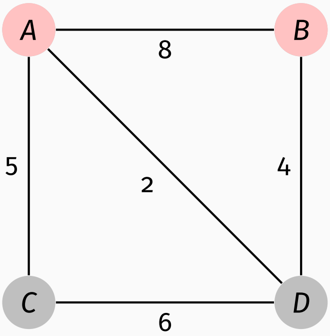
All-pairs shortest path

Adjacency Matrix
$G[i][j]$: length of edge from i to j.
$G[i][j] = \infty$: no edge from i to j.
$G[i][i] = 0$

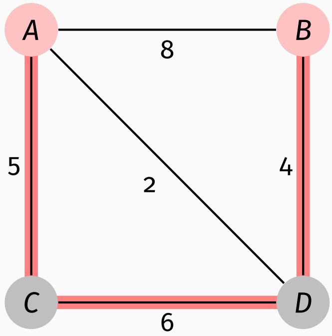
The edge weights should be non-negative, since cycles of negative length are problematic.

All-pairs shortest path – example

INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

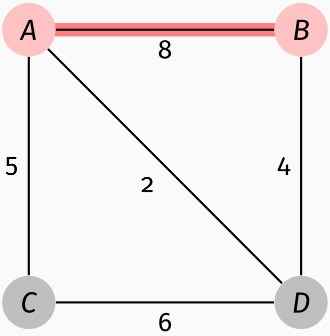


All-pairs shortest path – example

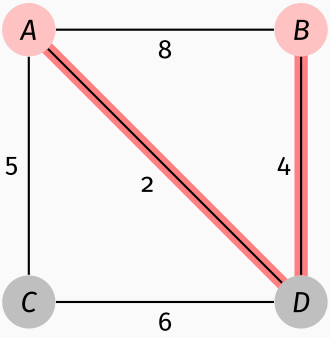


All-pairs shortest path – example

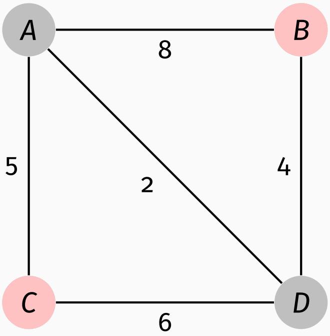
INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA



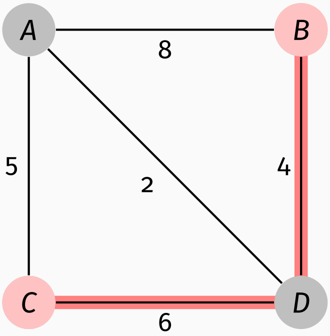
All-pairs shortest path – example



All-pairs shortest path – example



All-pairs shortest path – example



Important properties of shortest paths

A shortest path doesn’t contain cycles.

Important properties of shortest paths

INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

A shortest path doesn't contain cycles.

Every subpath of a shortest path is also a shortest path.

First DP approach

$\mathrm{DP}[i][j]$: length of a shortest path from *i* to *j*.

Optimal substructure
If *k* is located on the shortest path:

$$\mathrm{DP}[i][j] = \mathrm{DP}[i][k] + \mathrm{DP}[k][i]$$

First DP approach

$\mathrm{DP}[i][j]$: length of a shortest path from i to j.

Optimal substructure
If k is located on the shortest path:

$$\mathrm{DP}[i][j] = \mathrm{DP}[i][k] + \mathrm{DP}[k][i]$$

Computation?

$$\mathrm{DP}[i][j] = \min\left(G[i][j], \min_{k \in V} (\mathrm{DP}[i][k] + \mathrm{DP}[k][j])\right)$$

First DP approach – difficulties

Computation?

$$\mathrm{DP}[i][j] = \min\left(G[i][j], \min_{k \in V}\left(\mathrm{DP}[i][k] + \mathrm{DP}[k][j]\right)\right)$$

First DP approach – difficulties

Computation?

$$\mathrm{DP}[i][j] = \min\left(G[i][j], \min_{k\in V}\left(\mathrm{DP}[i][k] + \mathrm{DP}[k][j]\right)\right)$$

Cyclic dependencies

$$\mathrm{DP}[1][2] \Leftarrow \mathrm{DP}[1][3] \Leftarrow \mathrm{DP}[1][2] \Leftarrow \dots$$

DP state is too small!

Repaired DP approach

$\mathrm{DP}[l][i][j]$: length of a shortest path from i to j using at most l edges.

Repaired DP approach

$\mathrm{DP}[l][i][j]$: length of a shortest path from i to j using at most l edges.

Optimal substructure
The first $l-1$ edges of a shortest path are also a shortest path.

$$\mathrm{DP}[l][i][j] = \min_{k \in V} \left(\mathrm{DP}[l-1][i][k] + G[k][j]\right)$$
$$\mathrm{DP}[1][i][j] = G[i][j]$$

$\Rightarrow$ no more cyclic dependencies.

Repaired DP approach – analysis

$\mathrm{DP}[l][i][j]$: length of a shortest path from i to j using at most l edges.

$$\mathrm{DP}[l][i][j] = \min_{k \in V} \left(\mathrm{DP}[l-1][i][k] + G[k][j] \right)$$

Repaired DP approach – analysis

$\mathrm{DP}[l][i][j]$: length of a shortest path from i to j using at most l edges.

$$\mathrm{DP}[l][i][j] = \min_{k \in V} \left(\mathrm{DP}[l-1][i][k] + G[k][j] \right)$$

$\Theta(N^3)$ states, each requiring $\Theta(N)$ time.
$\Theta(N^4)$ running time and $\Theta(N^3)$ memory.

Repaired DP approach – top-down


```cpp
vector<vector<vector<int>>> cache(n,
 ↪  vector<vector<int>>(n, vector<int>(n, -1)));
const int INF = 1e9;
int dp(int l, int i, int j) {
    if (l == 0) return G[i][j];
    int &c = cache[l][i][j];
    if (c != -1) return c;
    c = INF;
    for (int k = 0; k < n; ++k) {
        c = min(c, dp(l-1, i, k) + G[k][j]);
    }
    return c;
}
```

Improved DP approach

$\mathrm{DP}[k][i][j]$: length of a shortest path from i to j through vertices with index $< k$.

Improved DP approach

$\mathrm{DP}[k][i][j]$: length of a shortest path from i to j through vertices with index $< k$.

Optimal substructure
The shortest path either goes through vertex k, or it avoids k.

$$\mathrm{DP}[k+1][i][j] = \min\left(\mathrm{DP}[k][i][k] + \mathrm{DP}[k][k][j], \mathrm{DP}[k][i][j]\right)$$
$$\mathrm{DP}[0][i][j] = G[i][j]$$

Improved DP approach – top-down

```cpp
vector<vector<vector<int>>> cache(n,
    vector<vector<int>>(n, vector<int>(n, -1)));
int dp(int k, int i, int j) {
    if (k == 0) return G[k][i][j];
    if (cache[k][i][j] != -1) return cache[k][i][j];
    cache[k][i][j] = min(
        dp(k-1, i, k-1) + dp(k-1, k-1, j),
        dp(k-1, i, j)
    );
    return cache[k][i][j];
}
```

Improved DP approach – top-down

INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

```
vector<vector<vector<int>>> cache(n, vector<vector<int>>(n, vector<int>(n, -1)));
int dp(int k, int i, int j) {
    if (k == 0) return G[k][i][j];
    if (cache[k][i][j] != -1) return cache[k][i][j];
    cache[k][i][j] = min(
        dp(k-1, i, k-1) + dp(k-1, k-1, j),
        dp(k-1, i, j)
    );
    return cache[k][i][j];
}
```

$\Theta(N^3)$ states, each requiring $\Theta(1)$ time.

$\Theta(N^3)$ running time and memory.

Improved DP approach – bottom up

```
vector<vector<vector<int>>> DP(n, vector<vector<int>>(n, vector<int>(n, INF)));
for (int i=0;i<n;++i)
  for (int j=0;j<n;++j)
    DP[0][i][j] = G[i][j];
for (int k = 0; k < n; ++k)
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      DP[k+1][i][j] = min(DP[k][i][k] + DP[k][k][j],
                          DP[k][i][j]);
```

$\Theta(N^3)$ states, each requiring $\Theta(1)$ time.

$\Theta(N^3)$ running time and memory.

Improved DP approach – less memory

INFORMATICS.
OLYMPIAD.CH
INFORMATIK-OLYMPIADE
OLYMPIADES D'INFORMATIQUE
OLIMPIADI DELL'INFORMATICA

```
vector<vector<int>> DP = G;
for (int k = 0; k < n; ++k) // k is outermost
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      DP[i][j] = min(DP[i][k] + DP[k][j], DP[i][j]);
```

$\Theta(N^3)$ running time, $\Theta(N^2)$ memory.

Applications

Everything that needs shortest paths

- Diameter of a graph.
- Possibly some task of the first round

Applications

Everything that needs shortest paths

- Diameter of a graph.
- Possibly some task of the first round

The algorithm also works on weighted graphs with negative weights.