Advanced Core in Algorithm Design #7
算法設計要論 第7回

Yasushi Kawase
河瀬 康志

Nov. 15th, 2022

last update: 12:17pm, November 15, 2022

Schedule

Lec. #	Date	Topics
1	10/4	Introduction, Stable matching
2	10/11	Basics of Algorithm Analysis, Greedy Algorithms (1/2)
3	10/18	Greedy Algorithms (2/2)
4	10/25	Divide and Conquer (1/2)
5	11/1	Divide and Conquer (2/2)
6	11/8	Dynamic Programming (1/2)
7	11/15	Dynamic Programming (2/2)
—	11/22	Thursday Classes
8	11/29	Network Flow (1/2)
9	12/6	Network Flow (2/2)
10	12/13	NP and Computational Intractability
11	12/20	Approximation Algorithms (1/2)
12	12/27	Approximation Algorithms (2/2)
13	1/10	Randomized Algorithms

Outline

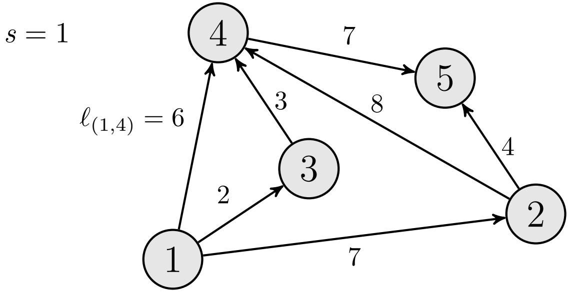
1. Shortest path problem (nonnegative lengths)
2. Shortest path problem with negative lengths
3. All-pairs shortest paths
4. Traveling Salesman Problem

Shortest path problem

Problem

- Input: Directed graph $G = (V, E)$, source $s \in V$, length $\ell_e \geq 0$ $(e \in E)$
- Goal: Compute shortest distance and path from s to each $t \in V \setminus \{s\}$

Example

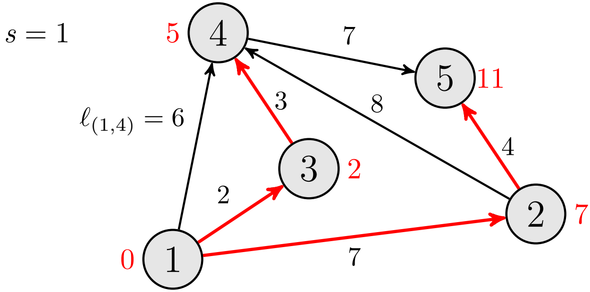


Shortest path problem

Problem

- Input: Directed graph $G = (V, E)$, source $s \in V$, length $\ell_e \geq 0$ $(e \in E)$
- Goal: Compute shortest distance and path from s to each $t \in V \setminus \{s\}$

Example

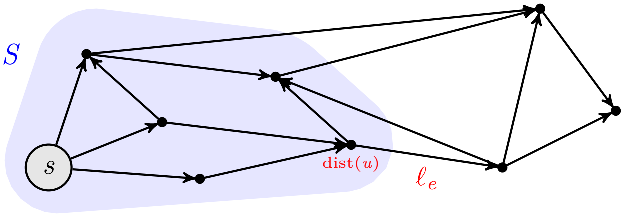


Approach

- Initialize $S \leftarrow \{s\}$, $\mathrm{dist}(s) \leftarrow 0$
- Repeatedly choose unexplored node $v \notin S$ which minimizes

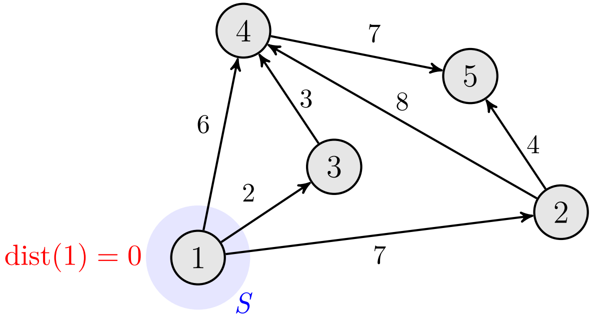
$$\min_{e=(u,v):\, u \in S} \mathrm{dist}(u) + \ell_e,$$

set $\mathrm{dist}(v)$ to be the above value, and $S \leftarrow S \cup \{v\}$



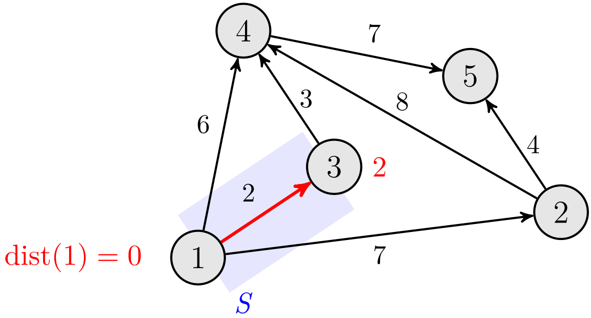
Behavior of Dijkstra algorithm

$s = 1$



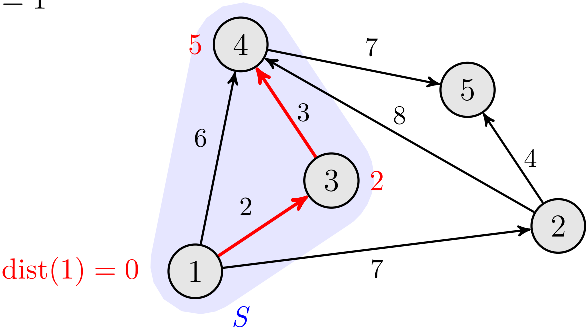
Behavior of Dijkstra algorithm

$s = 1$



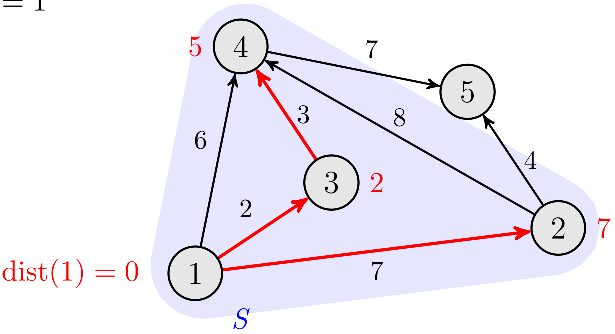
Behavior of Dijkstra algorithm

$s = 1$



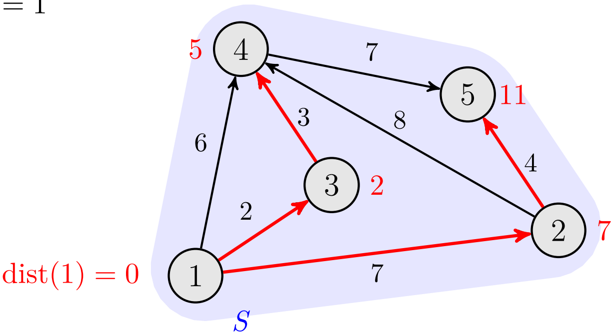
Behavior of Dijkstra algorithm

$s = 1$



Behavior of Dijkstra algorithm

$s = 1$



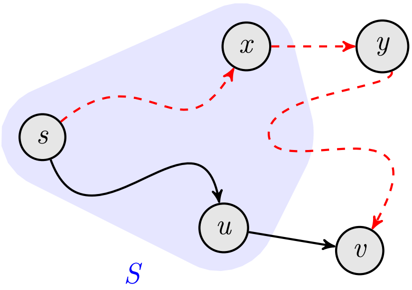
Correctness of Dijkstra algorithm

Induction

- Base case $S = \{s\}$ is clear $(\text{dist}(s) = 0)$
- Inductive step: $\ell(P) \geq \text{dist}(x) + \ell_{(x,\,y)} \geq \text{dist}(u) + \ell_{(u,v)}$

shortest s–v path

first edge in P that leaves S



Efficient implementation

Dijkstra algorithm

$\mathrm{dist}(s) \leftarrow 0$, $\mathrm{dist}(v) \leftarrow \infty$ $(\forall v \neq s)$;
Create an empty priority queue H;
foreach $v \in V$ **do** `insert`$(H, v, \mathrm{dist}(v))$;
while H *is not empty* **do**
    $u \leftarrow$ `deletemin`(H);
    foreach $e = (u, v) \in E$ **do**
        if $\mathrm{dist}(v) > \mathrm{dist}(u) + \ell_{(u,v)}$ **then**
            $\mathrm{dist}(v) \leftarrow \mathrm{dist}(u) + \ell_{(u,v)}$;
            `decreasekey`$(H, v, \mathrm{dist}(v))$;
Return dist;

- `insert`: n times
- `deletemin`: n times
- `decreasekey`: $\mathrm{O}(m)$ times

Running time

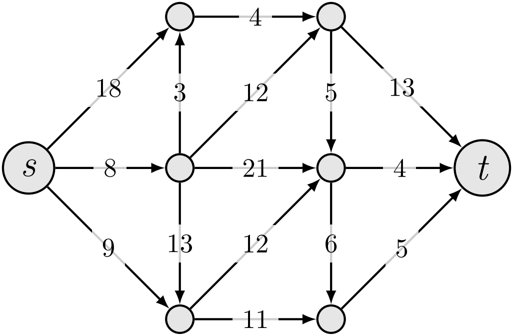
The running time depends on implementation of priority queue

Implementation	`insert`	`deletemin`	`decreasekey`	Total time
array	$\mathrm{O}(1)$	$\mathrm{O}(n)$	$\mathrm{O}(1)$	$\mathrm{O}(n^2)$
binary heap	$\mathrm{O}(\log n)$	$\mathrm{O}(\log n)$	$\mathrm{O}(\log n)$	$\mathrm{O}(m \log n)$
d-ary heap	$\mathrm{O}(d \log_d n)$	$\mathrm{O}(d \log_d n)$	$\mathrm{O}(\log_d n)$	$\mathrm{O}((nd + m) \log_d n)$
Fibonacci heap	$\mathrm{O}(1)$ (amortized)	$\mathrm{O}(\log n)$	$\mathrm{O}(1)$ (amortized)	$\mathrm{O}(m + n \log n)$
	n times	n times	$\mathrm{O}(m)$ times	

- dense graph $(m = \Omega(n^2))$ ⟶ array implementation is the faster
- sparse graph $(m = \mathrm{O}(n))$ ⟶ Fibonacci heap implementation is the faster

Quiz

Compute the shortest distance from s to t



Outline

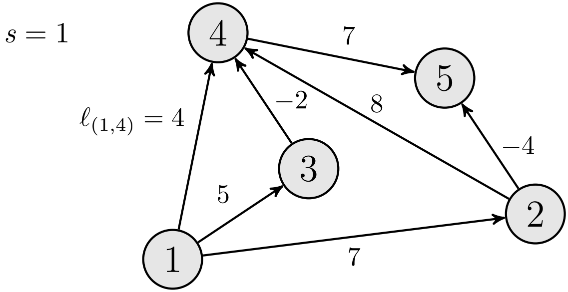
1. Shortest path problem (nonnegative lengths)
2. Shortest path problem with negative lengths
3. All-pairs shortest paths
4. Traveling Salesman Problem

Shortest path problem

Problem

- Input: Directed graph $G = (V, E)$, source $s \in V$, length ℓ_e $(e \in E)$
- Goal: Compute shortest distance and path from s to each $t \in V \setminus \{s\}$

Example

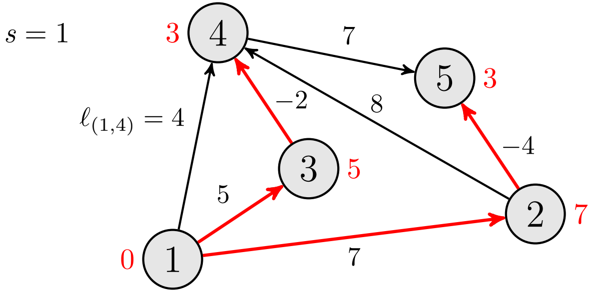


Shortest path problem

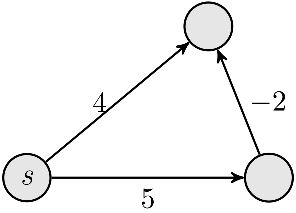
Problem

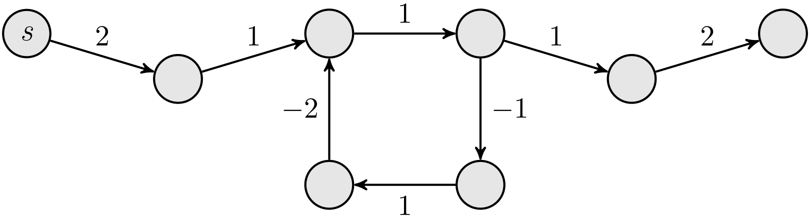
- Input: Directed graph $G = (V, E)$, source $s \in V$, length ℓ_e $(e \in E)$
- Goal: Compute shortest distance and path from s to each $t \in V \setminus \{s\}$

Example



Dijkstra algorithm may fail





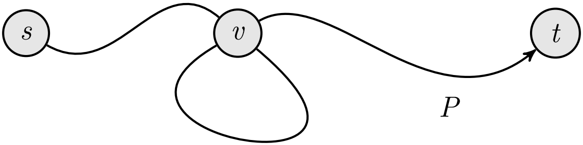
if there exists a negative cycle, then there may not exist a shortest path

No negative cycle case

Observation

If no negative cycles exists, there is a shortest s–t path that is simple

- Let P be the shortest s–t path that uses the fewest edges
- If P contains a directed cycle (whose length is nonnegative), it can be removed without increasing the length of P



→ there exists a shortest s–t path that has at most $n-1$ edges

Dynamic programming (Bellman–Ford algorithm)

$\mathrm{OPT}(i, v)$: length of shortest s–v path that uses at most i edges

Recursive formula

$$\mathrm{OPT}(i, v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = s \\ \infty & \text{if } i = 0 \text{ and } v \neq s \\ \min \left\{ \begin{array}{c} \mathrm{OPT}(i-1, v) \\ \min_{(u,v) \in E} \mathrm{OPT}(i-1, u) + \ell_{(u,v)} \end{array} \right\} & \text{if } i > 0 \end{cases}$$

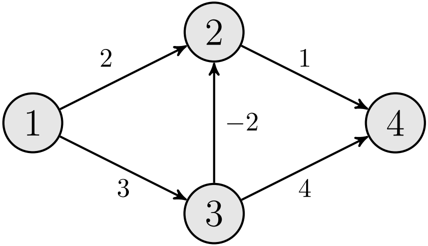
Bellman–Ford

```
dist(s) ← 0, dist(v) ← ∞ (∀v ≠ s);
for i ← 1, 2, ..., n − 1 do
    foreach e = (u, v) ∈ E do
        dist(v) ← min{dist(v), dist(u) + ℓ_(u,v)};
```

$\mathrm{O}(mn)$ subproblems, each one takse $\mathrm{O}(1)$ time ⟶ $\mathrm{O}(mn)$ time

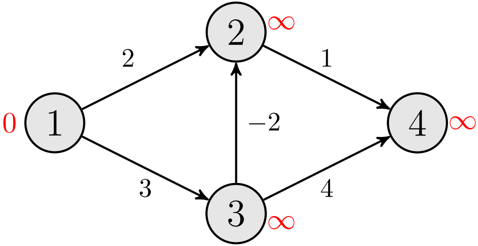
Behavior of Bellman–Ford

$s = 1$



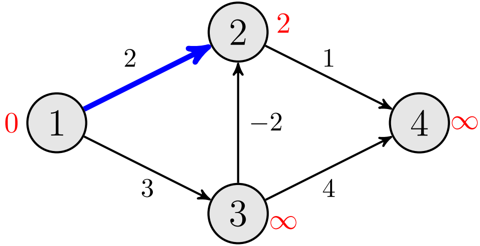
Behavior of Bellman–Ford

$s = 1$



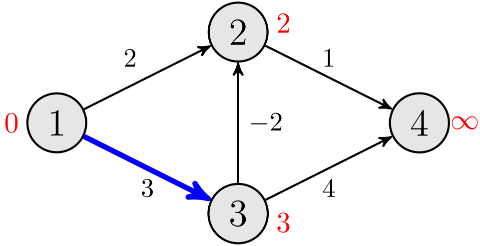
Behavior of Bellman–Ford

$s = 1$



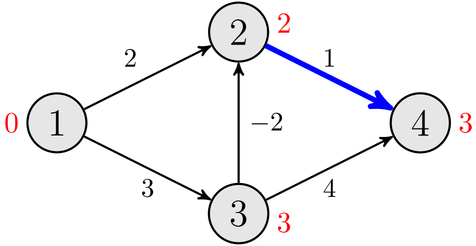
Behavior of Bellman–Ford

$s = 1$



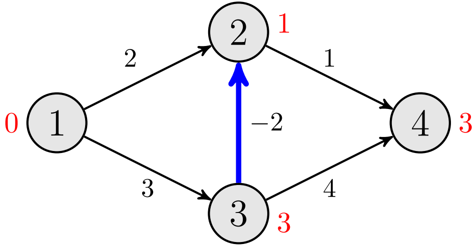
Behavior of Bellman–Ford

$s = 1$



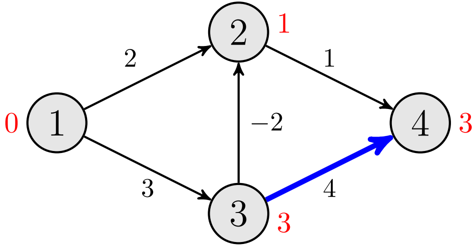
Behavior of Bellman–Ford

$s = 1$



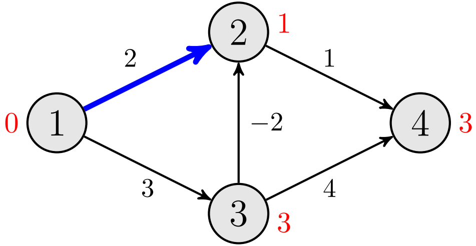
Behavior of Bellman–Ford

$s = 1$



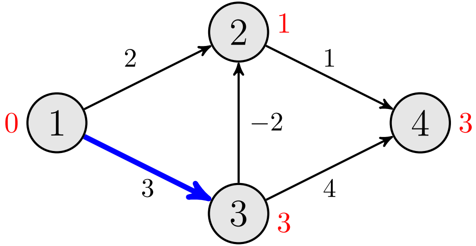
Behavior of Bellman–Ford

$s = 1$



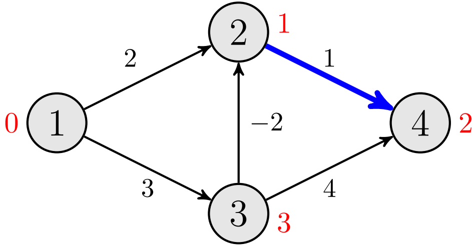
Behavior of Bellman–Ford

$s = 1$



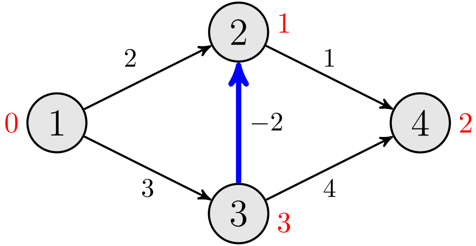
Behavior of Bellman–Ford

$s = 1$



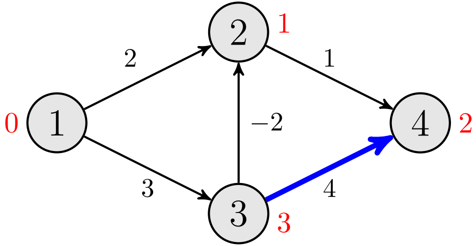
Behavior of Bellman–Ford

$s = 1$



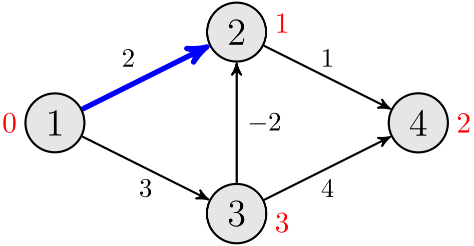
Behavior of Bellman–Ford

$s = 1$



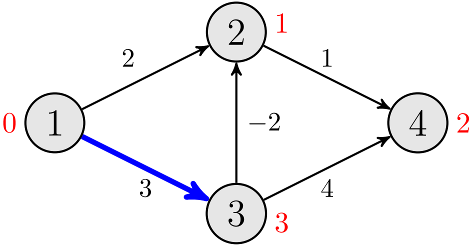
Behavior of Bellman–Ford

$s = 1$



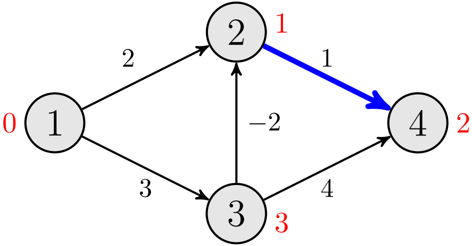
Behavior of Bellman–Ford

$s = 1$



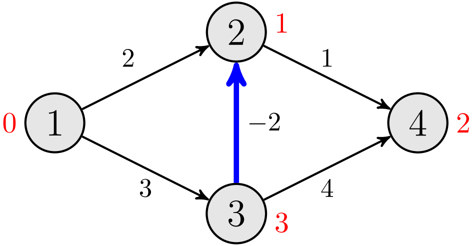
Behavior of Bellman–Ford

$s = 1$



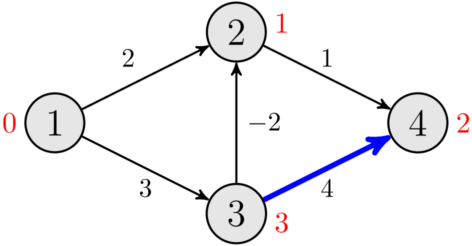
Behavior of Bellman–Ford

$s = 1$



Behavior of Bellman–Ford

$s = 1$



Detecting a negative cycle

Proposition

$\exists$ s–t path that contains a negative cycle $\Rightarrow \lim_{i\to\infty} \mathrm{OPT}(i,t) = -\infty$

Proposition

$\nexists$ s–t path that contains a negative cycle for any $t \in V$
$\iff \mathrm{OPT}(n,v) = \mathrm{OPT}(n-1,v)$ for any $v \in V$

- ($\Rightarrow$) because $\exists$ shortest s–v path that has at most $n-1$ edges
- ($\Leftarrow$) because $\mathrm{OPT}(n,v) = \mathrm{OPT}(n-1,v)$ $(\forall v \in V)$ implies $\lim_{i\to\infty} \mathrm{OPT}(i,v) = \mathrm{OPT}(n-1,v) > -\infty$

→ a negative cycle can be found in $\mathrm{O}(mn)$ time

Outline

1. Shortest path problem (nonnegative lengths)
2. Shortest path problem with negative lengths
3. All-pairs shortest paths
4. Traveling Salesman Problem

All-pairs shortest paths problem

Problem

- Input: Directed graph $G = (V, E)$ and length ℓ_e $(e \in E)$
- Goal: Compute shortest distance for every pair $(s, t) \in V^2$

applicable only when $\ell_e \geq 0$ $(\forall e \in E)$

- Running Dijkstra alg. for every $s \in V$ → $\mathrm{O}(mn + n^2 \log n)$ time
- Running Bellman–Ford alg. for every $s \in V$ → $\mathrm{O}(mn^2)$ time
- Better alternative: Floyed–Warshall alg. → $\mathrm{O}(n^3)$ time

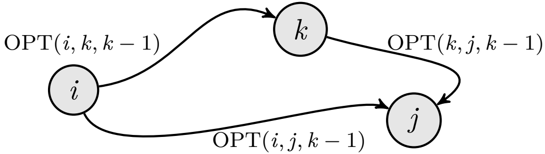
Dynamic programming (Floyd–Warshall algorithm)

- $V = \{1, 2, \ldots, n\}$
- $\mathrm{OPT}(i, j, k)$: shortest i–j dist. only using vertices in $\{1, 2, \ldots, k\}$

Recursive formula

$$\mathrm{OPT}(i,j,k) = \begin{cases} \min\left\{\begin{array}{c}\mathrm{OPT}(i,j,k-1) \\ \mathrm{OPT}(i,k,k-1) + \mathrm{OPT}(k,j,k-1)\end{array}\right\} & \text{if } k > 0 \\ 0 & \text{if } k = 0 \text{ and } i = j \\ \ell_{(i,j)} & \text{if } k = 0 \text{ and } (i,j) \in E \\ \infty & \text{otherwise} \end{cases}$$

$\mathrm{O}(n^3)$ subproblems, each one takes $\mathrm{O}(1)$ time → $\mathrm{O}(n^3)$ time



Implementation of Floyd–Warshall algorithm

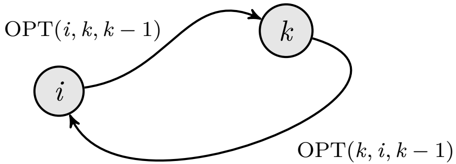
Floyd–Warshall algorithm

```
for i ← 1, 2, ..., n do
    for i ← 1, 2, ..., n do
        if i = j then dist(i, j) ← 0;
        else if (i, j) ∈ E then dist(i, j) ← ℓ_(i,j);
        else dist(i, j) ← ∞;
for k ← 1, 2, ..., n do
    for i ← 1, 2, ..., n do
        for j ← 1, 2, ..., n do
            dist(i, j) ← min{dist(i, j), dist(i, k) + dist(k, j)};
```

Detecting a negative cycle

Proposition

$\exists$ negative cycle including $i \in V \Rightarrow \mathrm{OPT}(i, i, n) < 0$



→ a negative cycle can be found in $\mathrm{O}(n^3)$ time

Another approach: reweighting edges

Dijkstra algorithm is applicable by reweighting edges

- reweight the edges by potential $\pi\colon V \to \mathbb{R}$

$$\ell'_{(u,v)} := \ell_{(u,v)} + \pi(u) - \pi(v) \ (\geq 0)$$

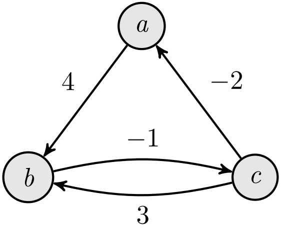
- compute reweighted shortest distances

$$\sum_{i=1}^{k} \ell'_{(v_i,v_{i+1})} = \sum_{i=1}^{k} \ell_{(v_i,v_{i+1})} + \underbrace{\pi(v_1) - \pi(v_{k+1})}_{\text{constant}}$$

Johnson's algorithm

1. add a new node s and add a new edge (s, v) with $\ell_{(s,v)} = 0$ $(\forall v \in V)$;
2. compute $\mathrm{dist}(s, v)$ for all $v \in V$ by Bellman–Ford alg.;
3. reweight the edges as $\ell'_{(u,v)} := \ell_{(u,v)} + \mathrm{dist}(s, u) - \mathrm{dist}(s, v)$;
4. compute reweighted shortest distances $\mathrm{dist}'(u, v)$ by Dijkstra alg.;
5. compute original shortest distances
   $\mathrm{dist}(u, v) = \mathrm{dist}'(u, v) - \mathrm{dist}(s, u) + \mathrm{dist}(s, v)$;

→ $\mathrm{O}(mn + n^2 \log n)$ time



Outline

1. Shortest path problem (nonnegative lengths)
2. Shortest path problem with negative lengths
3. All-pairs shortest paths
4. Traveling Salesman Problem

Traveling Salesman Problem

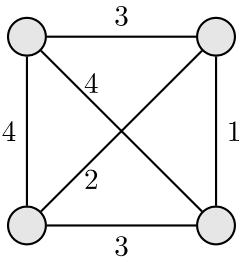
Problem

- Input: set of vertices V with distance $d\colon \binom{V}{2} \to \mathbb{R}_+$
- Goal: find a shortest cycle that visits all vertices exactly once

naive algorithm: $\Theta(n!)$ time ($(n-1)!$ possibilities)

→ improve it to $\mathrm{O}(n^2 2^n)$

Example



Traveling Salesman Problem

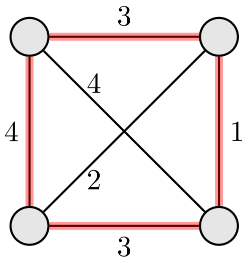
Problem

- Input: set of vertices V with distance $d\colon \binom{V}{2} \to \mathbb{R}_+$
- Goal: find a shortest cycle that visits all vertices exactly once

naive algorithm: $\Theta(n!)$ time ($(n-1)!$ possibilities)

→ improve it to $\mathrm{O}(n^2 2^n)$

Example



length $= 11$

Dynamic programming (Held–Karp algorithm)

- $V = \{1, 2, \ldots, n\}$
- $\mathrm{OPT}(S, j)$: length of the shortest 1–j path through every vertices in S

  $S \subseteq V \setminus \{1\}$, $j \in V \setminus S$. optimal value of the original problem is $\mathrm{OPT}(V \setminus \{1\}, 1)$

Recursive formula

$$\mathrm{OPT}(S, j) = \begin{cases} \min_{i \in S} \mathrm{OPT}(S \setminus \{i\}, i) + d(i, j) & \text{if } |S| > 0 \\ d(i, j) & \text{if } |S| = 0 \end{cases}$$

$\mathrm{O}(n2^n)$ subproblems, each one takes $\mathrm{O}(n)$ time → $\mathrm{O}(n^2 2^n)$ time