Lecture 8.
Shortest paths in graphs.

Algorithms and Data Structures
Ivan Solomatin
Harbour.Space@UTCC Bangkok

Outline

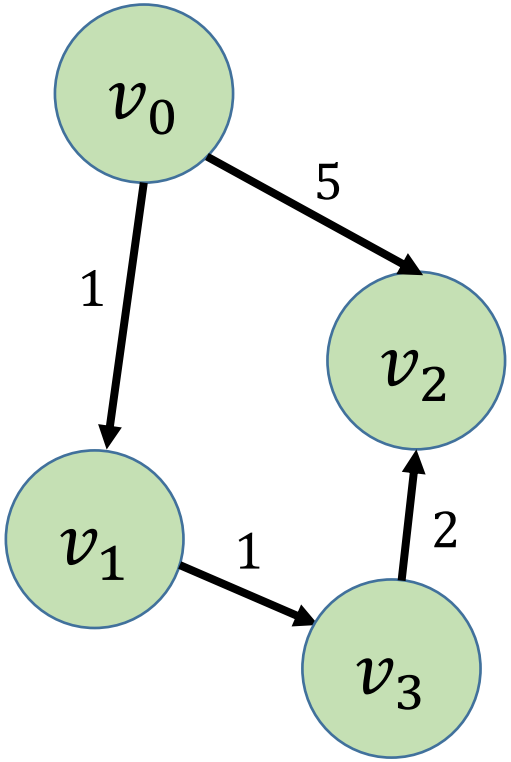
- Shortest paths in weighted graph:
  - Problem statement
  - Dijkstra's algorithm
  - Bellman-Ford algorithm
  - Floyd-Warshall algorithm

Shortest paths in weighted graph

- **Problem statement**
- Dijkstra's algorithm
  - Idea
  - Example
  - Implementation
- Bellman-Ford algorithm
  - Idea
  - Negative cycles
- Floyd-Warshall algorithm
  - Idea
  - Implementation

Problem statement

Definitions



$w(0,1) = 1$ $\quad$ $\rho(0,1) = 1$
$w(0,2) = 5$ $\quad$ $\rho(0,3) = 2$
$w(1,3) = 1$ $\quad$ $\rho(0,2) = 4$
$w(3,2) = 2$ $\quad$ $\rho(1,2) = 3$
$w(1,2) = \infty$ $\quad$ $\rho(1,0) = \infty$

1. **Weighted graph:** $G = (V, E, W)$

$$V = [0, N) \cap \mathbb{Z}$$
$$E \subseteq \{\{v,u\} : v,u \in V\} \text{ (undirected)}$$
$$E \subseteq V \times V \text{ (directed)}$$

$W : E \to \mathbb{R}$ – weight of edge

$$w(v,u) = \begin{cases} W((v,u)), & if\ (v,u) \in E, \\ 0, & if\ v = u, \\ \infty, & otherwise \end{cases}$$

2. **Weight (length, cost) of path:** $p = (v_0, v_1, \dots, v_{N-1})$

$$(v_i, v_{i+1}) \in E\ \forall i \in [0, n-1)$$
$$w(p) = \sum_{i=0}^{n-2} w(v_i, v_{i+1}),$$

3. **Distance between vertices** – minimum weight of path:

$$\rho(v,u) = \min_{p=(v,\dots,u)} w(p)$$

There are two types of problems (find d such that):

1. **Distance from one vertex** (s) **to all others:**

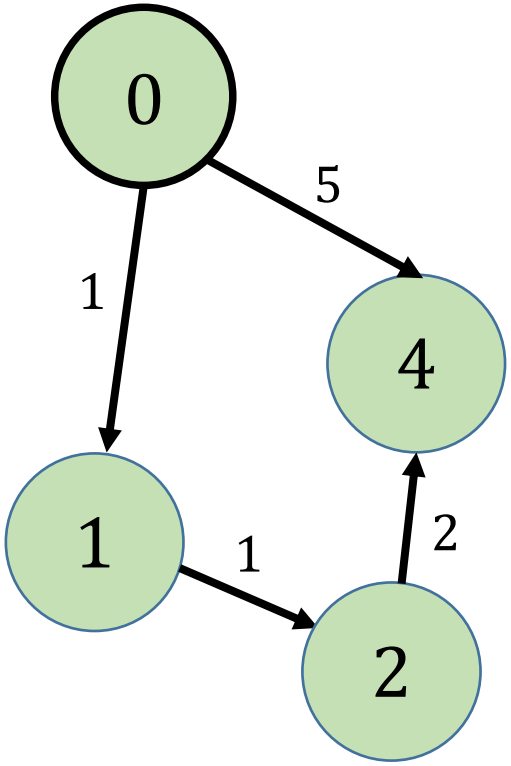
$$\forall v \in V:\ d[v] = \rho(s,v)$$

2. **Distance between each pair of vertices:**

$$\forall v,u \in V:\ d[v,u] = \rho(v,u)$$

Problem statement

Definitions



Theorem: Each subpath of optimal path is optimal.

Proof:

Let's suppose that $p = (\overbrace{v, \dots, t}^{q}, \dots u)$ is optimal:

$$w(p) = \rho(v, u)$$

But, subpath $q = (v, \dots, t)$ is not: $\exists q' = (v, \dots, t)$:

$$w(q') < w(q).$$

But this means that exists p':

$$p' = (\underbrace{v, \dots, t}_{q'}, \dots, u): \quad w(p') < w(p) = \rho(v, u)$$

Contradiction $\Rightarrow$ each subpath of optimal path is optimal.

Relaxation procedure.

We will calculate $d[i]$. Values of d will be showed as numbers in nodes.

Common procedure for shortest path algorithms: relaxation. relax(v, u) – "relax" an edge means, to check if this edge provides a better path for u than obtained before:

$$d[u] = \min\big(d[u], d[v] + w(v, u)\big)$$

We also can store array of previous states (like we did it in DP) $p[v]$ – a previous vertex in optimal path: $(s, \dots, p[v], v)$. If relax(v, u) is successful, $p[u] = v$.

Problem statement

Ways to store weighted graphs

Weighted graphs, can be stored in adjacency matrix:

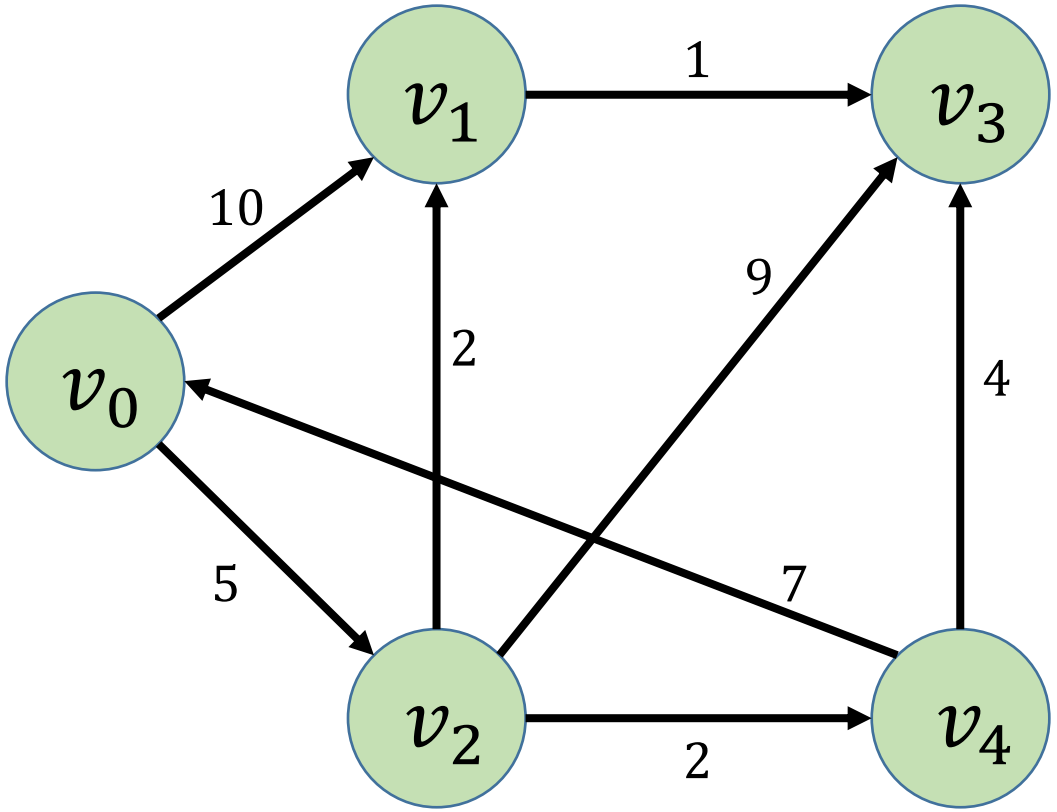
$$A_{v,u} = w(v,u) = \begin{cases} W\big((v,u)\big), & if\ (v,u) \in E, \\ 0, & if\ v = u, \\ \infty, & otherwise \end{cases}$$

	v_0	v_1	v_2	v_3	v_4
v_0	0	10	2	1	∞
v_1	5	0	5	∞	∞
v_2	∞	∞	0	∞	∞
v_3	∞	∞	∞	0	+
v_4	1	2	∞	∞	0

Problem statement

Ways to store weighted graphs

Weighted graphs, can be stored as edges list:



$$E = \begin{matrix} v & u & w \end{matrix}$$

```
[(0, 1, 10),
 (0, 2, 5),
 (1, 3, 1),
 (2, 1, 2),
 (2, 3, 9),
 (2, 4, 2),
 (4, 0, 7),
 (4, 3, 4)]
```

Problem statement

Ways to store weighted graphs

Also, we will use adjacency dicts:

Initialization:

```
G = {v: {} for v in V}
```

Adding edge $v \to u$ with weight $w(v, u)$:

```
G[v][u] = w(v,u)
```

List of edges adjacent with v:

```
G[v].keys()
```

Iteration over vertices adjacent with v:

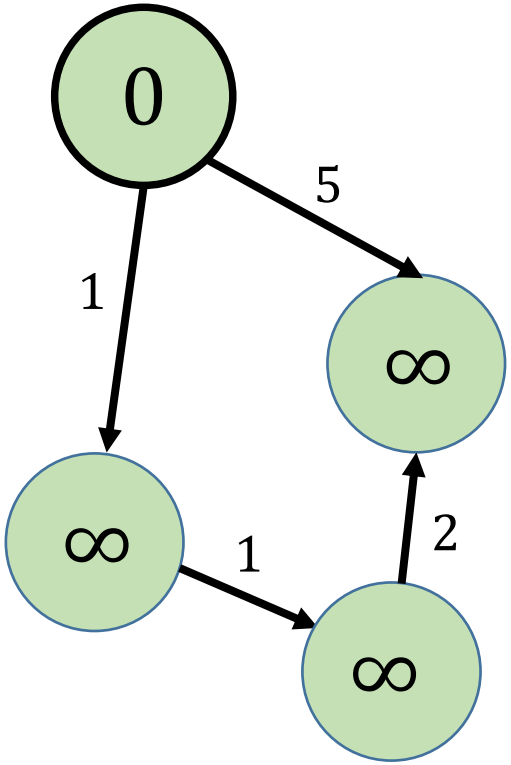
```
for u in G[v]:
    …
```

Shortest paths in weighted graph

- Problem statement
- **Dijkstra's algorithm**
  - **Idea**
  - Example
  - Implementation
- Bellman-Ford algorithm
  - Idea
  - Negative cycles
- Floyd-Warshall algorithm
  - Idea
  - Implementation

Dijkstra's algorithm

Idea



Dijkstra algorithm allows to find distances from one vertex to all others for graphs with **non-negative weights**. Given G, s, find d:

$$\forall v \in V\colon\ d[v] = \rho(s, v)$$

We will define set of vertices S and add vertices one by one.
Also, let's define $Q = V \setminus S$.
Invariant:

$$\forall v \in S\colon d[v] = \rho(s, v).$$
$$\forall v \in Q\colon d[v] = \min_{u \in S}\big(\rho(s, u) + w(u, v)\big).$$

d[s] supposed to be 0

Initially:

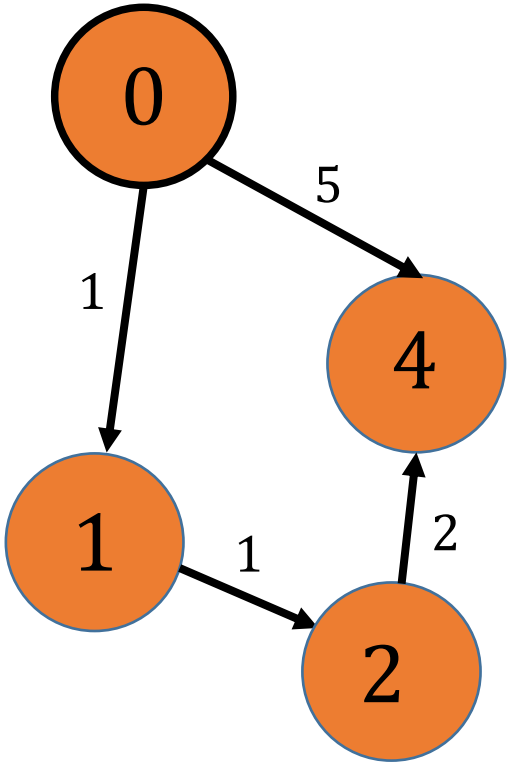
$$\begin{cases} S = \emptyset \\ d[v] = \infty \Rightarrow \text{Invariant is true.} \\ d[s] = 0 \end{cases}$$

Then, we are adding vertices one-by one and recalculating d and keep invariant correct.
Finally:

$$\begin{cases} S = V \\ d[v] = \rho(s, v) \end{cases} \Rightarrow \text{Problem solved.}$$

Dijkstra's algorithm

Idea



Dijkstra algorithm allows to find distances from one vertex to all others for graphs with **non-negative weights**. Given G, s, find d:

$$\forall v \in V\colon\ d[v] = \rho(s, v)$$

We will define set of vertices S and add vertices one by one.
Also, let's define $Q = V \setminus S$.
Invariant:

$$\forall v \in S\colon d[v] = \rho(s, v).$$
$$\forall v \in Q\colon d[v] = \min_{u \in S}\big(\rho(s, u) + w(u, v)\big).$$

d[s] supposed to be 0

Initially:

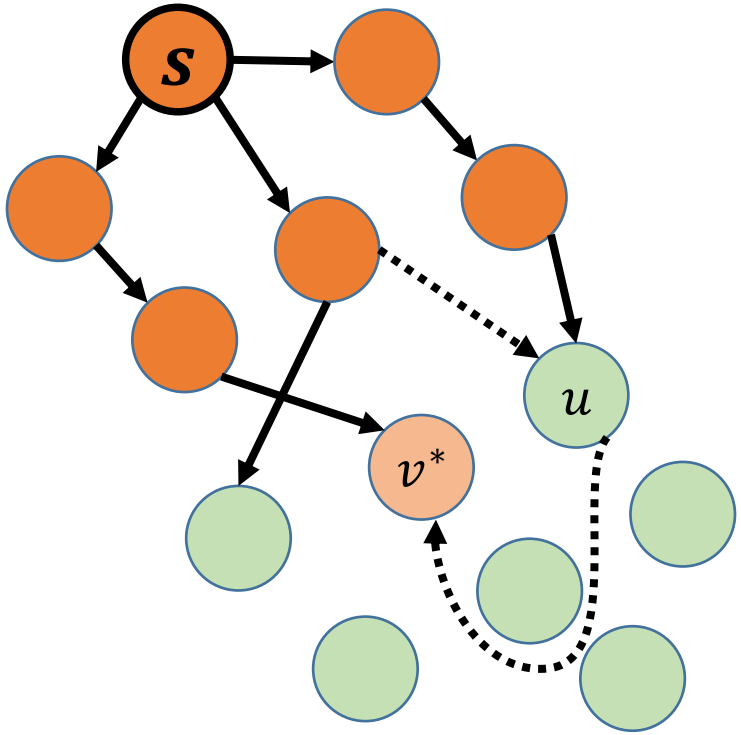
$$\begin{cases} S = \emptyset \\ d[v] = \infty \Rightarrow \text{Invariant is true.} \\ d[s] = 0 \end{cases}$$

Then, we are adding vertices one-by one and recalculating d and keep invariant correct.
Finally:

$$\begin{cases} S = V \\ d[v] = \rho(s, v) \end{cases} \Rightarrow \text{Problem solved.}$$

Dijkstra's algorithm

Idea



Invariant:

$$\forall v \in S: d[v] = \rho(s, v).$$
$$\forall v \in Q: d[v] = \min_{u \in S}\big(\rho(s, u) + w(u, v)\big).$$

Theorem:

For $v^* = \underset{v \in Q}{\operatorname{argmin}}\, d[v]$: $d[v^*] = \rho(s, v^*)$

Proof:

Let's suppose, there's path more optimal than $d[v^*]$:

$$p = (s, \dots, v^*): \; w(p) < d[v^*].$$

If it consists only of vertices from S, it should already have been counted in $d[v^*]$, so it can't be more optimal. So, it contains at least one more vertex from Q (except v^* itself). Let's denote first vertex of this path which is in Q as u. Weight of this path is:

$$w(p) = w(s, \dots, u) + w(u, \dots, v^*).$$

$(s, \dots, u)$ is a path with one edge from S, so: $w(p) = d[u] + w(u, \dots, v^*)$.

Second term is non-negative

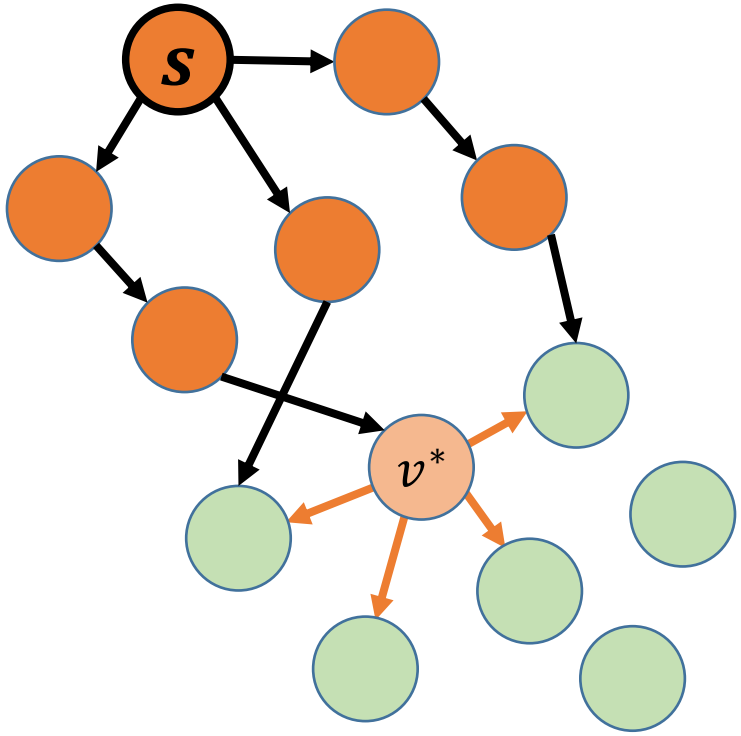
(because we don't have negative edges), so: $w(p) \geq d[u]$.

From the other side, $d[v^*] \leq d[u]$, so: $w(p) \geq d[u] \geq d[v^*]$.

But we supposed that p is more optimal. Contradiction. This means that $d[v^*] = \rho(s, v^*)$.

Dijkstra's algorithm

Idea



Invariant:

$$\forall v \in S: d[v] = \rho(s, v).$$
$$\forall v \in Q: d[v] = \min_{u \in S}\big(\rho(s, u) + w(u, v)\big).$$

So, on each step, we know which vertex should be added to S: v^* – one with minimum d. That will automatically satisfy first condition of invariant. To satisfy second one we need to relax all edges incident with v^*.

This leads us to algorithm (pseudocode):

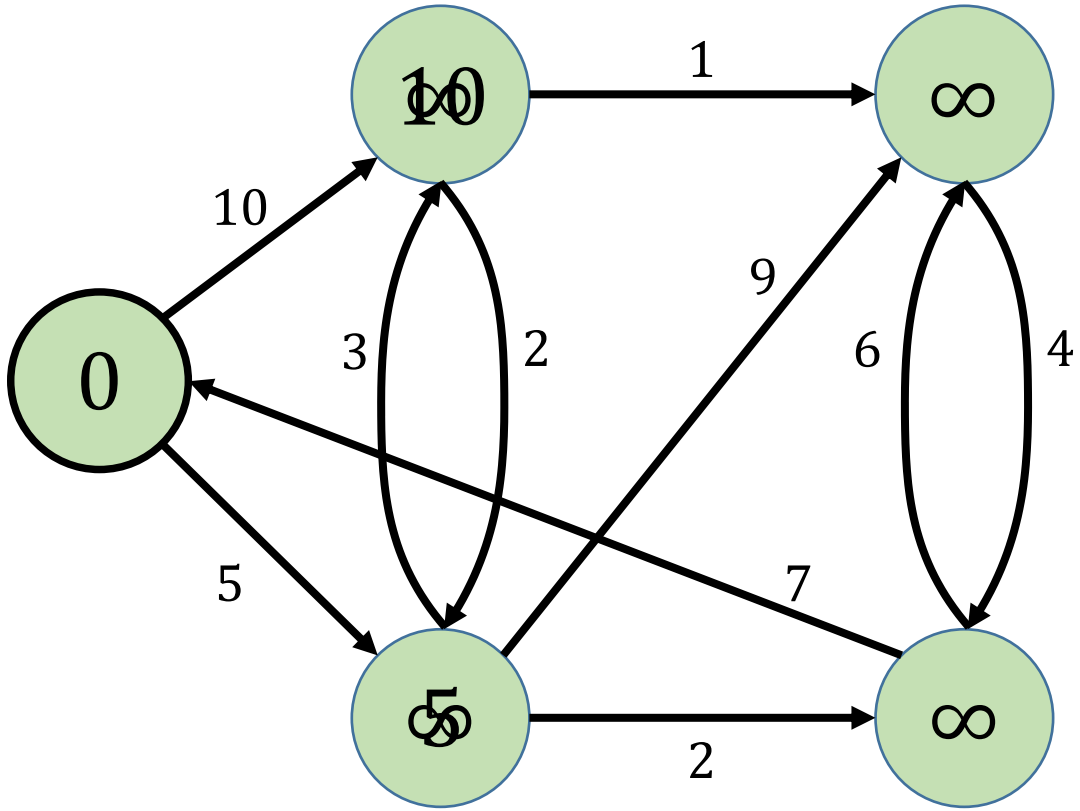
```
S = set()
Q = set(V)
d = [∞] * len(Q)
d[s] = 0
while Q:
    v = pop_vertex_with_min_d(Q)
    S.add(v)
    for u in G[v]:
        d[v] = min(d[u], d[v] + w[v, u])
```

Shortest paths in weighted graph

- Problem statement
- **Dijkstra's algorithm**
  - Idea
  - **Example**
  - Implementation
- Bellman-Ford algorithm
  - Idea
  - Negative cycles
- Floyd-Warshall algorithm
  - Idea
  - Implementation

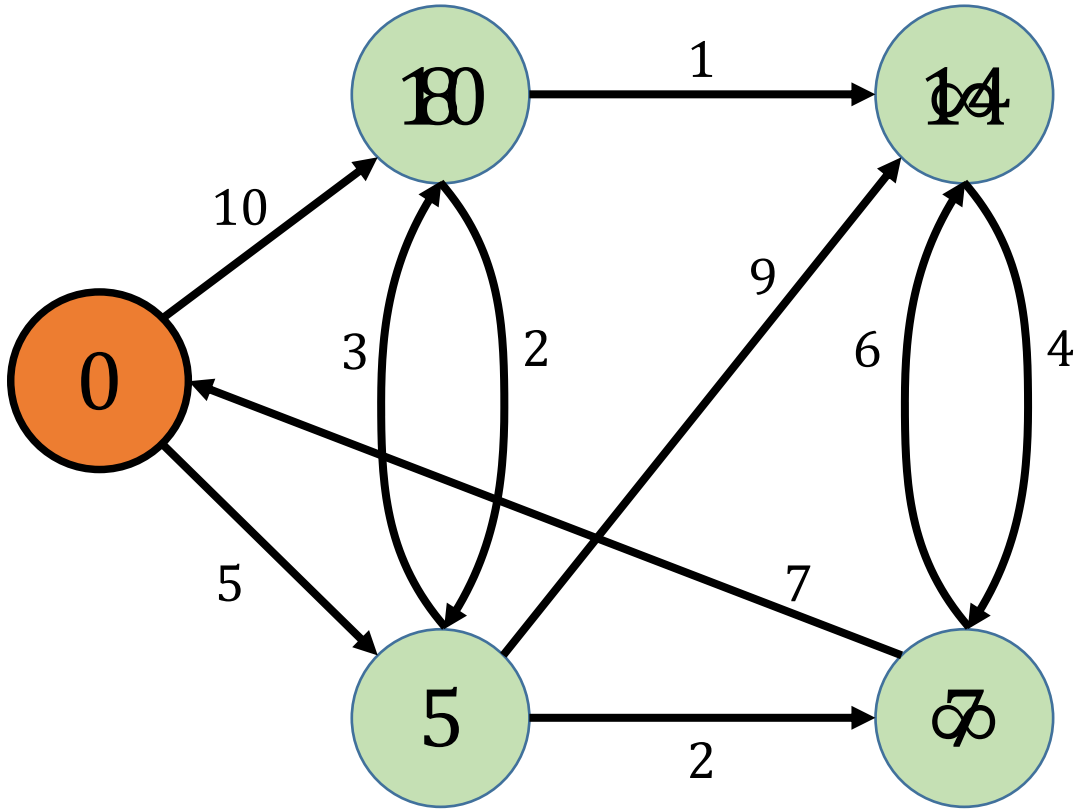
Dijkstra's algorithm

Example



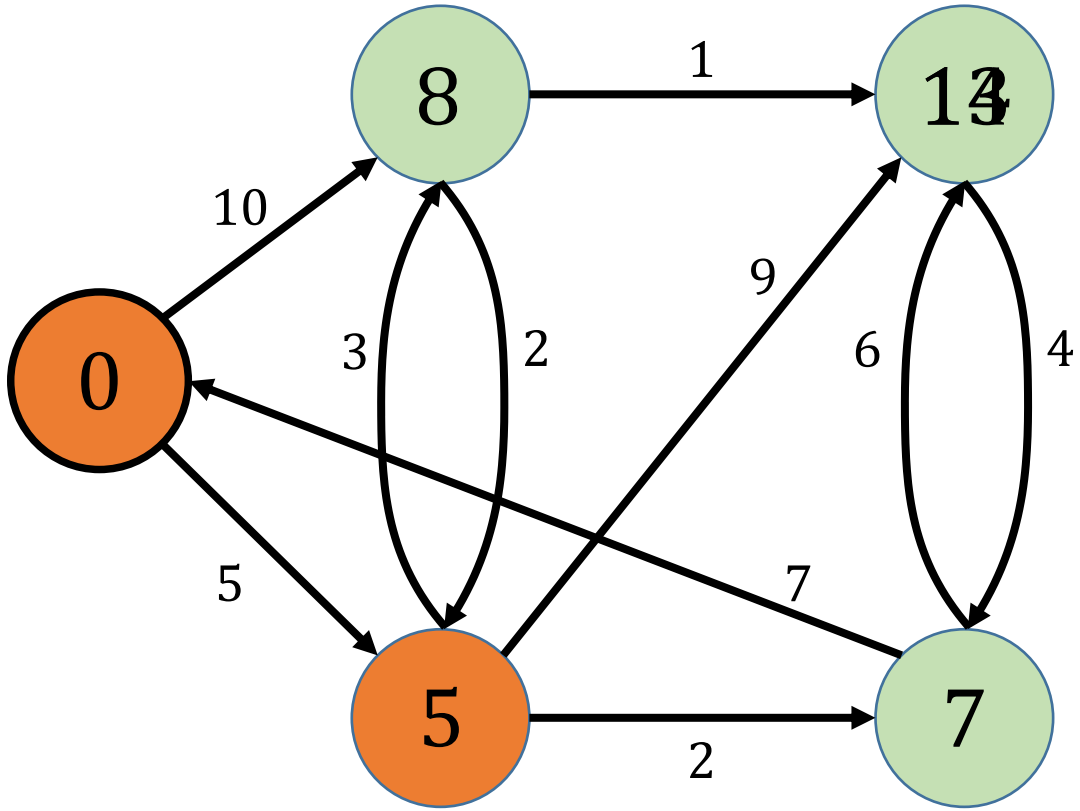
Dijkstra's algorithm

Example



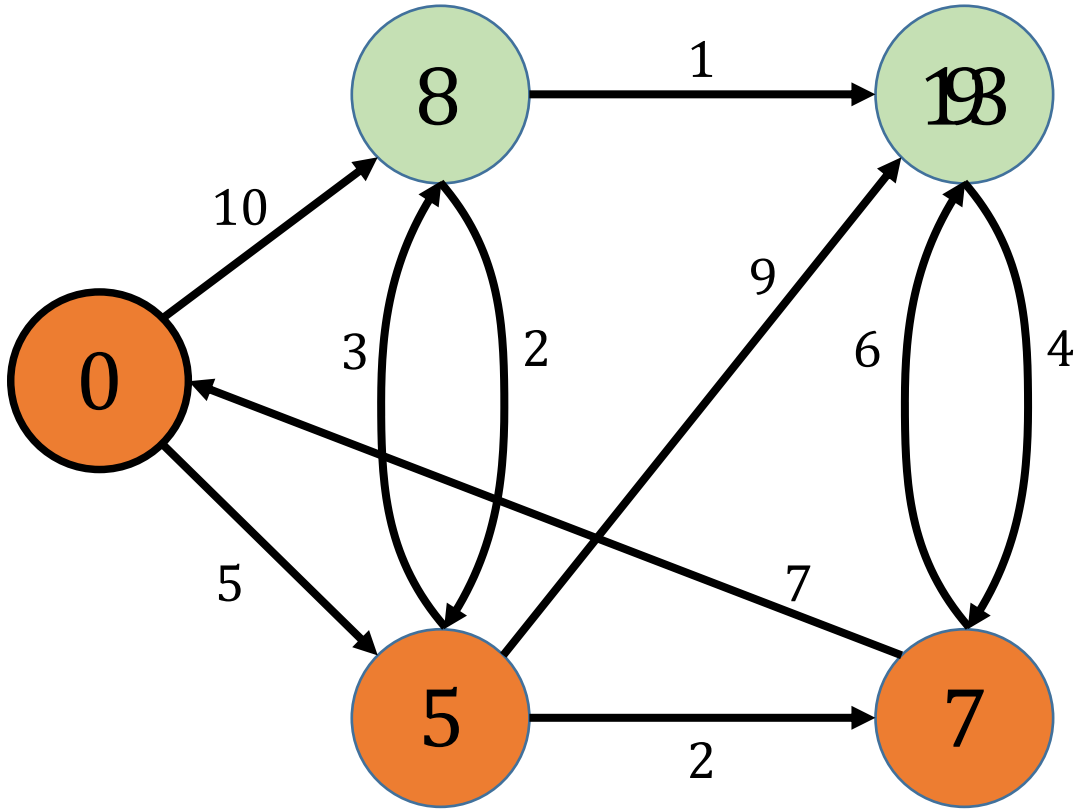
Dijkstra's algorithm

Example



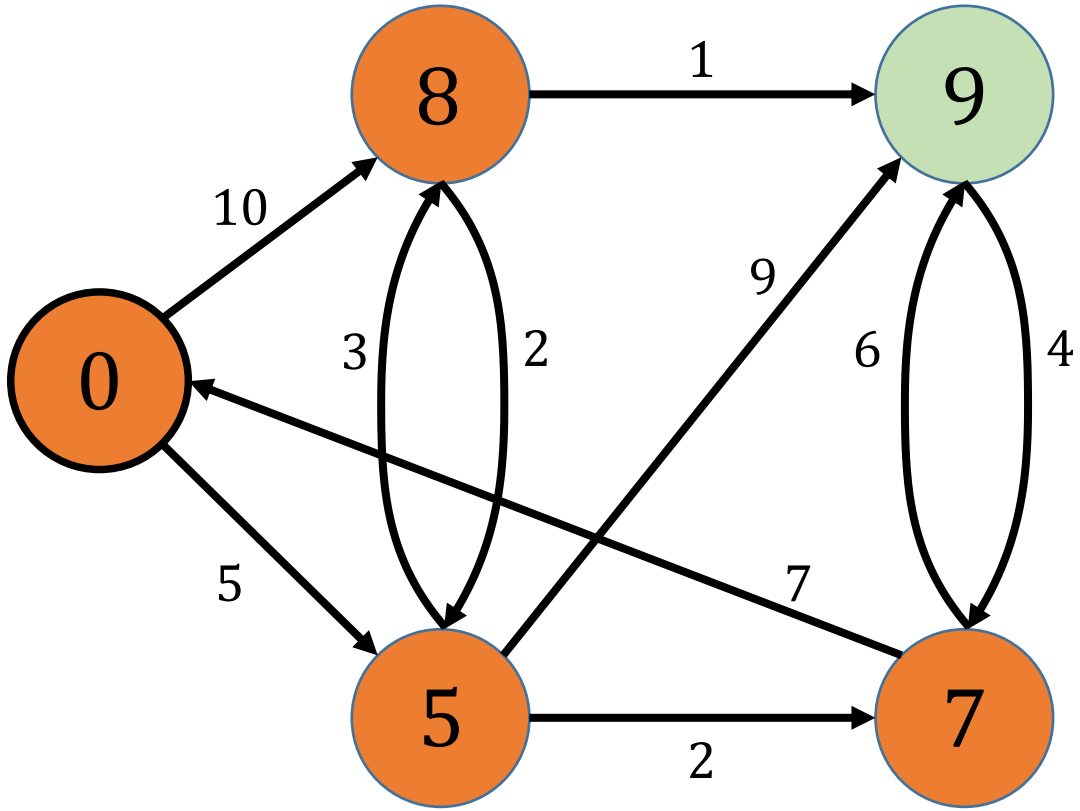
Dijkstra's algorithm

Example



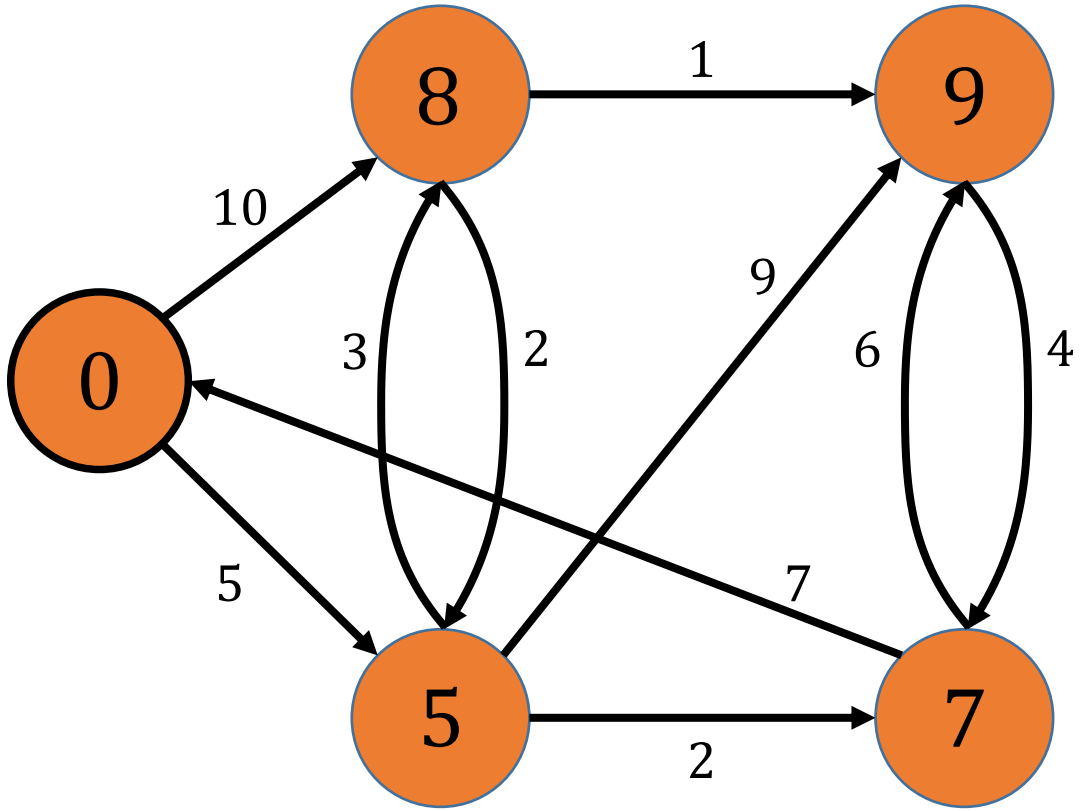
Dijkstra's algorithm

Example



Dijkstra's algorithm

Example



Shortest paths in weighted graph

- Problem statement
- **Dijkstra's algorithm**
  - Idea
  - Example
  - **Implementation**
- Bellman-Ford algorithm
  - Idea
  - Negative cycles
- Floyd-Warshall algorithm
  - Idea
  - Implementation

Dijkstra's algorithm

Implementation

```
S = set()
Q = set(V)
d = [∞] * len(Q)
d[s] = 0
while Q:
    v = pop_vertex_with_min_d(Q)
    S.add(v)
    for u in G[v]:
        d[v] = min(d[u], d[v] + w[v, u])
```

`pop_vertex_with_min_d(Q)` can be implemented on heap, but a problem is that key value d may be changed. If elements in heap are just vertices, each relaxation may require searching for a vertex in heap and updating the value. Update is $O(\log N)$, but search of particular vertex in heap is $O(N)$, because we need to iterate over whole heap. It's possible to store for each vertex link to it's node in heap, but it's rather complicated.

Another solution is to store all relaxation events (d, v) in heap instead. Notice that one vertex may have several relaxation events. On each step heap will store all **actual** relaxation events for current values of d for vertices from Q: $\{(d[v], v) : v \in Q\}$. And also, it may store several outdated relaxation events, there are two types of outdated events:

1. $(d, v): v \in Q,\ d > d[v]$ – all previous relaxations of v. They have bigger d, because each relaxation decreases d. So, they will be returned after **actual** event for this v: $(d[v], v)$. So, we can guarantee that v will be added to S before this outdated event will be returned from heap.
2. $(d, v): v \in S$ – previously events of type 1. Events for which vertex was added to S. We can just ignore these events because $d[v]$ is already guaranteed to be optimal.

So, outdated events of type 1 cannot be returned from heap, and events of type 2 can easily be detected and ignored.

Dijkstra's algorithm

Implementation

```
h = []
d = [inf] * N
v = -1
# added fictitious vertex to simplify
# while with heappop:
S = {-1}
d[s] = 0
heappush(h, (d[s], s))
while h:
    while v in S and h: # all outdated events are skipped
        v = heappop(h)[1]
    if v not in S:
        S.add(v)
        for u in G[v]:
            new_d = d[v] + G[v][u]
            if new_d < d[u]:
                heappush(h, (new_d, u))
                d[u] = new_d
```

Complexity:

$$O(|E|\log|V|)$$

Shortest paths in weighted graph

- Problem statement
- Dijkstra's algorithm
  - Idea
  - Example
  - Implementation
- **Bellman-Ford algorithm**
  - **Idea**
  - Negative cycles
- Floyd-Warshall algorithm
  - Idea
  - Implementation

Bellman-Ford algorithm

Idea

Bellman-Ford algorithm allows to find distances from one vertex to all others. Given G, s, find d:

$$\forall v \in V\colon\ d[v] = \rho(s, v) = \min_{p=(s,\dots,v)} w(p)$$

Let's define $|p|$ – number of edges in the path.

The idea of BF algorithm is based on dynamic programming:

1. Subproblem: $\hat{d}[l][v] = \min\limits_{\substack{p=(s,\dots,v)\\ |p|\le l}} w(p)$

2. Basis: $\hat{d}[0][s] = 0,\ \hat{d}[0][v] = \infty\ \forall v \neq s$.

3. Inductive step: $\hat{d}[l][u]$ Let's look at optimal (for length $\le l$) path $p = (s, \dots, v, u)$: $|p| \le l$.

It's subpath $p' = (s, \dots, v)$: $|p'| \le l - 1$ is optimal (for length $\le l - 1$), and it's weight is $\hat{d}[l-1][v]$.

$$\hat{d}[l][u] = \min_{(v,u)\in E} \left(\hat{d}[l-1][v] + w(v,u)\right)$$

4. If finite shortest path exists, exists shortest path with $l < |V|$ **(it will be discussed in the next part)**

So, answer for initial problem is $d = \hat{d}[|V|-1][:]$.

Bellman-Ford algorithm

Idea

Let's notice that $\hat{d}[l+1][v] \le \hat{d}[l][v]$.

And let's notice that we don't need optimal paths with exact length limitation.

It's enough to guarantee that: $d[l][v] = w(s, \dots, v) \le \min\limits_{\substack{p=(s,\dots,v) \\ |p| \le l}} w(p) = \hat{d}[l][v]$.

In this case for inductive step we can use any $l' \ge l-1$, because:

$$d[l][u] = \min_{(v,u)\in E}\left(d[l'][v] + w(v,u)\right) \le \min_{(v,u)\in E}\left(\hat{d}[l-1][v] + w(v,u)\right) = \hat{d}[l][v].$$

So, we don't need to store values for exact l. We can perform relaxation in 1D array and guarantee that on l-th step: $d[v] \le \hat{d}[l][v]$.

That leads us to idea of the algorithm: let's just relax all edges $|V|-1$ times.

```
d = [inf] * N
d[s] = 0
for i in range(N - 1):
    for (v, u) ∈ E:
        d[u] = min(d[u], d[v] + w(v, u))
```

On step $(|V|-1)$ **(because exist shortest paths with *l*<|*V*|):**

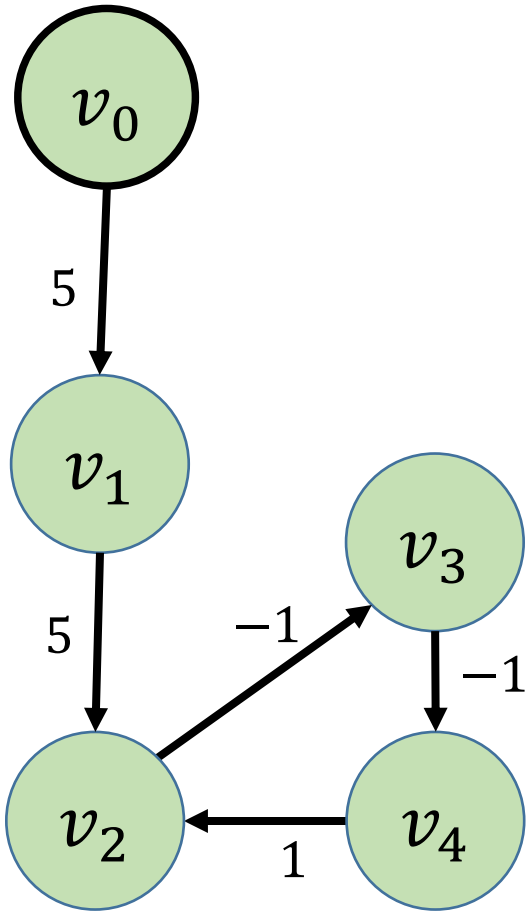
$$d[v] = w(s, \dots, v) \le \min_{p=(s,\dots,v)} w(p) = \rho(s,v) \quad \Rightarrow \quad d[v] = \rho(s,v).$$

Shortest paths in weighted graph

- Problem statement
- Dijkstra's algorithm
  - Idea
  - Example
  - Implementation
- **Bellman-Ford algorithm**
  - Idea
  - **Negative cycles**
- Floyd-Warshall algorithm
  - Idea
  - Implementation

Bellman-Ford algorithm

Negative cycles



$$\rho(0,1) = 5$$
$$\rho(0,2) = 10\ ?$$

$$p_0 = (0,1,2):\ w(p_0) = 10$$
$$p_1 = (0,1,2,3,4,2):\ w(p_0) = 9$$
$$p_2 = (0,1,2,3,4,2,3,4,2):\ w(p_0) = 8$$
$$p_i = (0,1,2,[3,4,2] \times i):\ w(p_i) = 10 - i$$
$$\forall n\ \exists i: w(p_i) < n$$

$$\rho(0,2) = \min_{p=(v,\dots,u)} w(p) = -\infty$$

Negative cycle: cycle $c = (v_0, \dots, v_0): w(c) < 0$

If graph contains a negative cycle, finite shortest path may not exist. In our course we will not work with such graphs. But BF algorithm allows to check if there is a negative cycle in the graph.

Bellman-Ford algorithm

Negative cycles

Theorem BF1: If graph has no negative cycles, $\forall v, u \in V$ exists an optimal path $p = (v, \dots, u)$: $|\mathrm{p}| < |V|$.

Proof:

Let's look at an optimal path $p = (s, \dots, v)$. Let's suppose that $|p| \geq |V|$ and there's no optimal paths $s \to v$ shorter. By Dirichlet's box principle (pigeonhole principle), there is at least one vertex that occurs twice in p. Path between these two occurrences is a cycle. This cycle has non-negative weight, so, it can be removed from path which will give us new path:

$$p' = (s, \dots, v): \; w(p') \leq w(p), \qquad |p'| < |p|.$$

Bellman-Ford algorithm

Implementation

The idea of checking for negative cycles is to try to relax edges one more time. If at least one relaxation succeeds, this means that:

$$\exists p = (v, \dots, u): |p| = |V|: \quad \forall p' = (v, \dots, u): |p'| < |V| \quad \hookrightarrow \quad w(p') > w(p)$$

That contradicts with conclusion of the theorem BF1. Applying negation of the theorem BF1 we can conclude that there's a negative cycle in our graph.

```
def bellman_ford(G, s):
    d = [inf] * len(G)
    d[s] = 0
    for i in range(N - 1):
        for (v, u) in E:
            d[u] = min(d[u], d[v] + w(v, u))
    for (v, u) in E:
        if d[u] > d[v] + w(v, u):
            return None # negative cycle exists
    return d
```

Complexity:

$$O(|V||E|)$$

Shortest paths in weighted graph

- Problem statement
- Dijkstra's algorithm
  - Idea
  - Example
  - Implementation
- Bellman-Ford algorithm
  - Idea
  - Negative cycles
- **Floyd-Warshall algorithm**
  - **Idea**
  - Implementation

Floyd-Warshall algorithm
Idea

Floyd-Warshall algorithm allows to find distances between each pair of vertices. Given G, find d:

$$\forall v, u \in V\colon\ d[v][u] = \rho(v, u)$$

Floyd-Warshall also uses DP approach. Let's do like we did several times before: let's define DP solution for several parameters, and then get rid of one.

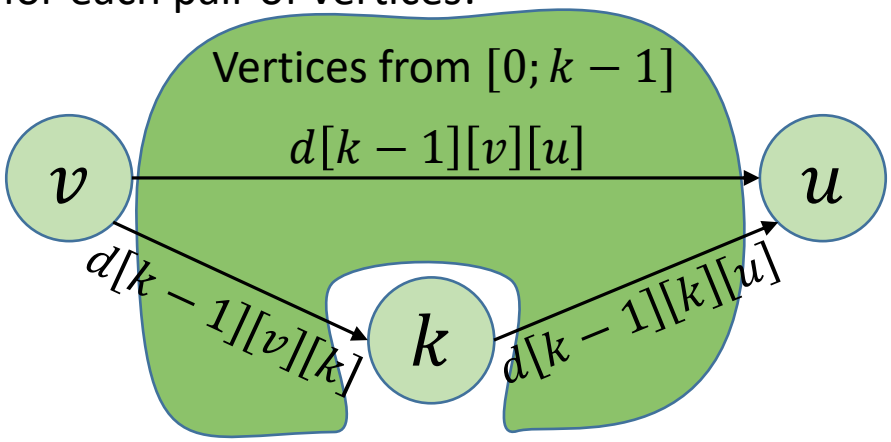
1. Subproblems:
$d[k][v][u]$ – weight of optimal path $(v, \underbrace{\ldots}, u)$ such that all internal vertices are from $[0; k]$.
2. Basis. $d[-1][v][u]$ – no internal vertices allowed. So, only paths (v, u) are possible (ones which consists of one edge only), or empty ones.

$$d[-1][v][u] = \begin{cases} w(v,u), & if\ (v,u) \in E; \\ 0, & if\ v = u; \\ \infty, & otherwise \end{cases}$$

Actually, $d[-1][v][u]$ is adjacency matrix of the graph

Floyd-Warshall algorithm
Idea

3. Inductive step:
When we allow k-th vertex for internal usage, we should try to create paths though this vertex for each pair of vertices:



$$d[k][v][u] = \min(d[k-1][v][u], d[k-1][v][k] + d[k-1][k][u])$$

4. $d[|V|-1][v][u]$ – desired matrix of distances

Shortest paths in weighted graph

- Problem statement
- Dijkstra's algorithm
  - Idea
  - Example
  - Implementation
- Bellman-Ford algorithm
  - Idea
  - Negative cycles
- **Floyd-Warshall algorithm**
  - Idea
  - **Implementation**

Floyd-Warshall algorithm

Implementation

Inductive step:

$$d[k][v][u] = \min(d[k-1][v][u], d[k-1][v][k] + d[k-1][k][u])$$

Let's try to get rid of k parameter.

Let's look at $d[k][v][k]$:

$$d[k][v][k] = \min(d[k-1][v][k], d[k-1][v][k] + \cancelto{0}{d[k-1][k][k]})$$

Otherwise, negative cycle exists

$$d[k][v][k] = d[k-1][v][k]$$

Similarly: $$d[k][k][u] = d[k-1][k][u]$$

So, values which are used for recalculating dp values on k-th step are not changing during this step. So, we don't need to store d for previous k, and can overwrite values for each step inplace in 2D array. This leads us to the final implementation:

```
d = a
# a – adjacency matrix with zeroes on main diagonal
for k in range(|V|):
    for v in range(|V|):
        for u in range(|V|):
            d[v][u] = min(d[v][u], d[v][k] + d[k][u])
```

Complexity: $O(|V|^3)$

Floyd-Warshall algorithm

One more usage scenario

Floyd-Warshall algorithm may be used to calculate transitive closure of the graph:

$$G^* = (V, E^*): \quad E^* = \{(v, u): u \text{ reachable from } v\}$$

Or, in other words, E^* defines reachability relation on V explicitly.

Similarly to previous slide:

```
d = a
# a – Boolean adjacency matrix with True on main diagonal
for k in range(|V|):
    for v in range(|V|):
        for u in range(|V|):
            d[v][u] = d[v][u] or (d[v][k] and d[k][u])
```

$d[v][u]$ – is u reachable from v

Complexity: $O(|V|^3)$

Conclusion

Thank you for watching!