The shortest path problem

Raimondas Čiegis

Matematinio modeliavimo katedra, email: rc@vgtu.lt

April 25 d., 2025

In this lecture, we will consider in detail the problem of finding the shortest path between the vertices of a graph. This is a very frequently solved logistics problem, and it is important to examine the main methods for solving it

First, we will present the most important definitions, many of which you have already studied in discrete mathematics lectures.

Graphs

Let us have a set of vertices $V = \{v_1, v_2, \ldots, v_N\}$ and a set of edges $E = \{e_1, e_2, \ldots, e_K\}$.

An edge is a pair of vertices $e_j = (v_{1j}, v_{2j})$.

Graph is denoted by $G = (V, E)$.

Graphs

Let us have a set of vertices $V = \left\{v_1, v_2, \ldots, v_N\right\}$ and a set of edges $E = \left\{e_1, e_2, \ldots, e_K\right\}$.

An edge is a pair of vertices $e_j = (v_{1j}, v_{2j})$.

Graph is denoted by $G = (V, E)$.

The simplest example of a graph is a map of a country's roads: cities and towns form a set of vertices, and roads form a set of edges.

If the edges $e_j = (v_{1j}, v_{2j})$ and $e_k = (v_{2j}, v_{1j})$ are different (the direction of the connection is also important), then they are called *directed*, and the graph consisting of such edges is called *directed* graph.

If the edges $e_j = (v_{1j}, v_{2j})$ and $e_k = (v_{2j}, v_{1j})$ are different (the direction of the connection is also important), then they are called *directed*, and the graph consisting of such edges is called *directed* graph.

On city roads, we also encounter a similar situation where only one-way traffic is allowed on the street.

Real numbers can be assigned to the edges of a graph that evaluate distance, time, weight and similar attributes.

Such a graph is called *weighted*.

We will denote the weight of the edge $e_j \in E$ by $w(e_j)$.

Real numbers can be assigned to the edges of a graph that evaluate distance, time, weight and similar attributes.

Such a graph is called *weighted*.

We will denote the weight of the edge $e_j \in E$ by $w(e_j)$.

The set of neighbors of vertex v is defined by

$$N(v) = \{\, u : \ \ u \in V, \ (u, v) \in E \text{ or } (v, u) \in E \,\}$$

and it is called the *neighborhood* of vertex v.

The degree of a vertex v is denoted by $\deg(v)$ and it is equal to the number of its neighbors.

Examples of some important cases of graphs are presented in this figure .

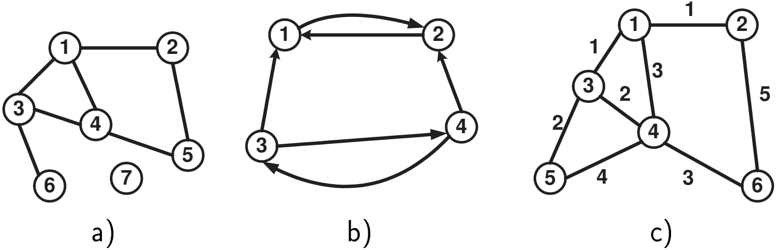

FIGURE: Examples of graphs: a) undirected graph, $|V| = 7$, $|E| = 7$, the degree of vertices v_1, v_3, v_4 is equal to 3, the degree of vertices v_2, v_5 is equal to 2, v_6 and end vertex, v_7 is an isolated vertex, b) directed graph, $|V| = 4$, $|E| = 6$, c) weighted graph, $|V| = 6$, $|E| = 8$.

A set of vertices

$$p = \left\{ v_{i_0},\ v_{i_1},\ \ldots,\ v_{i_k} \right\}$$

is called *a path* if all adjacent vertices are connected by edges, i.e.

$$(v_{i_j}, v_{i_{j+1}}) \in E, \ \ j = 0, 1, \ldots, k-1\,.$$

A set of vertices

$$p = \left\{ v_{i_0},\ v_{i_1},\ \ldots,\ v_{i_k} \right\}$$

is called *a path* if all adjacent vertices are connected by edges, i.e.

$$(v_{i_j}, v_{i_{j+1}}) \in E, \quad j = 0, 1, \ldots, k-1\,.$$

The graph is called *connected* if for each pair of vertices x and y of the graph, there is a path joining x and y.

A set of vertices

$$p = \left\{ v_{i_0},\ v_{i_1},\ \ldots,\ v_{i_k} \right\}$$

is called *a path* if all adjacent vertices are connected by edges, i.e.

$$(v_{i_j}, v_{i_{j+1}}) \in E, \ \ j = 0, 1, \ldots, k-1 \,.$$

The graph is called *connected* if for each pair of vertices x and y of the graph, there is a path joining x and y.

For example, if we have a road map and the graph is connected, then it is possible to drive from any city or settlement to another one.

During spring floods, some settlements become inaccessible.

Let's consider a weighted graph. The length of a path p is defined as

$$W(p) = \sum_{j=0}^{k-1} w\left(v_{i_j}, v_{i_{j+1}}\right).$$

In the case where the weights of the edges of a graph are not specified, the path length is the number of edges on the path.

Let's consider a weighted graph. The length of a path p is defined as

$$W(p) = \sum_{j=0}^{k-1} w\left(v_{i_j}, v_{i_{j+1}}\right).$$

In the case where the weights of the edges of a graph are not specified, the path length is the number of edges on the path.

The shortest path connecting two vertices $a, b \in V$ of the graph G is the path

$$p = \{a,\ v_{i_1},\ \dots,\ v_{i_k}, b\},$$

satisfying the condition

$$W(p) \leqslant W(p^{'}),$$

where $p^{'}$ is any other path connecting a and b.

Let's consider a weighted graph. The length of a path p is defined as

$$W(p) = \sum_{j=0}^{k-1} w\left(v_{i_j}, v_{i_{j+1}}\right).$$

In the case where the weights of the edges of a graph are not specified, the path length is the number of edges on the path.

The shortest path connecting two vertices $a, b \in V$ of the graph G is the path

$$p = \left\{a,\ v_{i_1},\ \ldots,\ v_{i_k}, b\right\},$$

satisfying the condition

$$W(p) \leqslant W(p^{'}),$$

where $p^{'}$ is any other path connecting a and b.
If weights of all edges are positive numbers, then the shortest path always exist.

We will solve the problem of finding the shortest paths between one vertex s of a graph G and all other vertices $v \in V$ of this graph.

We will solve the problem of finding the shortest paths between one vertex s of a graph G and all other vertices $v \in V$ of this graph.

It is clear that the shortest path exists only to vertices that are reachable from s.

We will solve the problem of finding the shortest paths between one vertex s of a graph G and all other vertices $v \in V$ of this graph.

It is clear that the shortest path exists only to vertices that are reachable from s.

For graphs with different structure, special efficient shortest path finding algorithms are developed that best utilize information about the graph structure.

Now let's introduce some more notations.

Now let's introduce some more notations.

Let $d[v]$ denote the length of the best path already found from vertex s to vertex v (this onformation is dynamicaly updated during search algorithm).

Now let's introduce some more notations.

Let $d[v]$ denote the length of the best path already found from vertex s to vertex v (this onformation is dynamicaly updated during search algorithm).

Let $\pi[v]$ denote the vertex from which we reach the vertex v in the already known shortest path.

Now let's introduce some more notations.

Let $d[v]$ denote the length of the best path already found from vertex s to vertex v (this onformation is dynamicaly updated during search algorithm).

Let $\pi[v]$ denote the vertex from which we reach the vertex v in the already known shortest path.

Let $\delta(s, v)$ denote the length of the shortest path from vertex s to vertex v.

In graphs (as in real road maps), the triangle inequality is not necessary valid. It states that the length of the edge connecting two vertices a and b is not greater than the length of the path passing through one (or several) intermediate vertices.

In graphs (as in real road maps), the triangle inequality is not necessary valid. It states that the length of the edge connecting two vertices a and b is not greater than the length of the path passing through one (or several) intermediate vertices.

However, it is easy to prove two important properties of shortest paths, they make a basis for algorithms that are used to construct the shortest paths.

1. Consider the shortest path connecting vertices s and v. Let it consist of several intermediate paths, e.g. connecting s with a, then a with b, and finally b with v. Those intermediate paths can connect a few inner vertices.

1. Consider the shortest path connecting vertices s and v. Let it consist of several intermediate paths, e.g. connecting s with a, then a with b, and finally b with v. Those intermediate paths can connect a few inner vertices.

Then all intermediate paths are also shortest paths connecting the corresponding vertices:

$$\delta(s, v) = \delta(s, a) + \delta(a, b) + \delta(b, v).$$

1. Consider the shortest path connecting vertices s and v. Let it consist of several intermediate paths, e.g. connecting s with a, then a with b, and finally b with v. Those intermediate paths can connect a few inner vertices.

Then all intermediate paths are also shortest paths connecting the corresponding vertices:

$$\delta(s, v) = \delta(s, a) + \delta(a, b) + \delta(b, v).$$

The proof is simple. If any intermediate path is not the shortest, then we can replace it with the shortest one and then the path $\delta(s, v)$ will be shortened. But this cannot be the case, because this path is the shortest.

2. Let us take any three vertices of the graph a, b and c. Then the triangle inequality is true

$$\delta(a, b) \leq \delta(a, c) + \delta(c, b).$$

In all the algorithms presented in this lecture we will use the following basic operation, which is designed to refine the shortest path approximation:

$$
\begin{aligned}
&\text{Relax } (u, v, w)\ : \\
&\quad \textbf{if } d[v] > d[u] + w(u, v) : \\
&\qquad d[v] = d[u] + w(u, v) \\
&\qquad \pi[v] = u
\end{aligned}
$$

In all the algorithms presented in this lecture we will use the following basic operation, which is designed to refine the shortest path approximation:

$$\text{Relax } (u, v, w) \ :$$
$$\textbf{if } d[v] > d[u] + w(u, v) :$$
$$d[v] = d[u] + w(u, v)$$
$$\pi[v] = u$$

It is easy to verify that such an operation is safe:

if $\quad d[u] \geq \delta(s, u), \quad$ then $\quad d[v] \geq \delta(s, v)$.

Now we will present one quite general structure of the shortest path finding algorithms.

It is always useful to construct a new algorithm by using a clear fixed scheme.

Now we will present one quite general structure of the shortest path finding algorithms.

It is always useful to construct a new algorithm by using a clear fixed scheme.

$$
\begin{aligned}
&\text{Initialize} \quad : \\
&\quad d[u] = \infty, \ \pi[u] = s, \ \forall u \in V \\
&\quad d[s] = 0 \\
&\text{Repeat} \quad : \\
&\quad \text{Select an edge } (u, v) \\
&\quad \text{Relax}(u, v, w) \\
&\text{Until all edges have } d[v] \leq d[u] + w(u, v)
\end{aligned}
$$

It is clear that the main goal of every algorithm is to create an efficient edge selection order.

It is clear that the main goal of every algorithm is to create an efficient edge selection order.

In the worst case, when we check all edges of the graph many times, the complexity of the algorithm is exponential (such algorithms are inefficient).

Let us consider a directed, weighted graph $G = (V, E)$ with no cycles (DAG).

Let us consider a directed, weighted graph $G = (V, E)$ with no cycles (DAG).

We will find the shortest paths from s to all other vertices of the graph, which are reachable from s.

Let us consider a directed, weighted graph $G = (V, E)$ with no cycles (DAG).

We will find the shortest paths from s to all other vertices of the graph, which are reachable from s.

1. First, we topologically sort the vertices of the graph G. The complexity of this step is

$$\Theta(|V| + |E|).$$

Let us consider a directed, weighted graph $G = (V, E)$ with no cycles (DAG).

We will find the shortest paths from s to all other vertices of the graph, which are reachable from s.

1. First, we topologically sort the vertices of the graph G. The complexity of this step is

$$\Theta(|V| + |E|).$$

2. Second, in the general scheme for construction shortest paths, we visit vertices v in sorted order and modify the values of each neighbor of v. The complexity of this stage is also equal to

$$\Theta(|V| + |E|).$$

Let us consider a directed, weighted graph $G = (V, E)$ with no cycles (DAG).

We will find the shortest paths from s to all other vertices of the graph, which are reachable from s.

1. First, we topologically sort the vertices of the graph G. The complexity of this step is

$$\Theta(|V| + |E|).$$

2. Second, in the general scheme for construction shortest paths, we visit vertices v in sorted order and modify the values of each neighbor of v. The complexity of this stage is also equal to

$$\Theta(|V| + |E|).$$

We have developed an efficient algorithm for solving the shortest path problem when the graph has a special DAG structure.

Dijkstra algorithm

Now we will consider a more general case of graphs where we only know that the weights of all edges of the graph are positive numbers.

Dijkstra algorithm

Now we will consider a more general case of graphs where we only know that the weights of all edges of the graph are positive numbers.

Let S be the set of vertices to which we have already found the shortest path. Initially, this set includes only the initial vertex s.

When executing the algorithm, at each step we add one new vertex to the set S.

Dijkstra algorithm

Now we will consider a more general case of graphs where we only know that the weights of all edges of the graph are positive numbers.

Let S be the set of vertices to which we have already found the shortest path. Initially, this set includes only the initial vertex s.

When executing the algorithm, at each step we add one new vertex to the set S.

In the set Q we store the vertices to which the shortest path is not yet known.

Dijkstra (G, w, s) :

Initialize :

$S = \{s\},\ Q = V \setminus S$

$d[u] = \infty,\ \pi[u] = s,\ \forall u \in Q$

$d[s] = 0,\ d[u] = w(s, u), \forall u \in Q:\ (s, u) \in E$

while $Q \neq \emptyset$:

$u = \text{Extract\_Min } (Q)$

$S = S \cup \{u\}$

for each $v \in \text{Adj}[u] \subset Q$:

Relax(u, v, w)

1. We store the elements of the set Q using a binary min heap structure. The root of the heap stores the vertex with the smallest value $d[v]$

1. We store the elements of the set Q using a binary min heap structure. The root of the heap stores the vertex with the smallest value $d[v]$

2. The algorithm is based on a greedy strategy. Theoretical analysis confirms that such an algorithm calculates the shortest paths from a given vertex to all remaining vertices of the graph G.

Complexity of the Dijkstra algorithm

1. Initial construction of the binary heap requires $\Theta(|V|)$ operations.

2. Remove from the heap Q the vertex to which the shortest path is known, the total costs of this part of the algorithm are given by $\Theta(|V| \log |V|)$.

3. The costs of Relax operation are equal to $\Theta(|E|)$.

Example: find the shortest paths in the directed graph

We have the wighted directed graph:

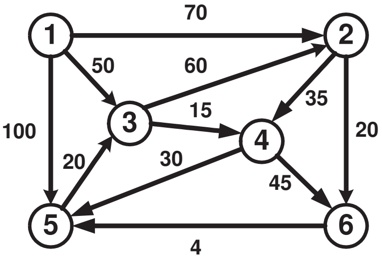

FIGURE: The weighted directed graph.

The details of search procedure are defined as:

$$\begin{aligned}
&\text{i} = 1: \quad S = \{v_1,\ v_3\}, \\
&d = (0,\ 70,\ 50,\ 65,\ 100,\ \infty),\quad \pi = (1,\ 1,\ 1,\ 3,\ 1,\ 1), \\
&\text{i} = 2: \quad S = \{v_1,\ v_3,\ v_4\}, \\
&d = (0,\ 70,\ 50,\ 65,\ 95,\ 110),\quad \pi = (1,\ 1,\ 1,\ 3,\ 4,\ 4), \\
&\text{i} = 3: \quad S = \{v_1,\ v_3,\ v_4,\ v_2\}, \\
&d = (0,\ 70,\ 50,\ 65,\ 95,\ 90),\quad \pi = (1,\ 1,\ 1,\ 3,\ 4,\ 2), \\
&\text{i} = 4: \quad S = \{v_1,\ v_3,\ v_4,\ v_2,\ v_6\}, \\
&d = (0,\ 70,\ 50,\ 65,\ 94,\ 90),\quad \pi = (1,\ 1,\ 1,\ 3,\ 6,\ 2), \\
&\text{i} = 5: \quad S = \{v_1,\ v_3,\ v_4,\ v_2,\ v_6,\ v_5\}, \\
&d = (0,\ 70,\ 50,\ 65,\ 94,\ 90),\quad \pi = (1,\ 1,\ 1,\ 3,\ 6,\ 2).
\end{aligned}$$