Routing and Packet Forwarding

Tina Schmidt

September 2008

Contents

1 Introduction **1**
1.1 The Different Layers of the Internet 2

2 IPv4 **3**

3 Routing and Packet Forwarding **5**
3.1 The Shortest Path Problem 6
3.2 Dijkstras Algorithm 7
3.3 Practical Realizations of Routing Algorithms 8

4 Autonomous Systems **10**
4.1 Intra-AS-Routing 10
4.2 Inter-AS-Routing 11

5 Other Services of IP **11**

A Dijkstra's Algorithm **13**

1 Introduction

The history of the internet and Peer-to-Peer networks starts in the $60s$ with the ARPANET (Advanced Research Project Agency Network). Back then, when computers were expensive and linked to several terminals, the aim of ARPANET was to establish a continuous network inbetween mainframe computers in the U.S. Starting with only three computers in 1969, mainly universities were involved in establishing the ARPANET. The ARPANET became a network inbetween networks of mainframe computes and therefore was called inter-net. Today it became the internet which links millions of computers. For a long time nobody knew what use such a Wide Area Netwok (WAN) could have for the population. The internet was used for e-mails, newsgroups, exchange of scientific data and some special software, all in all services that were barely known in public. The spread of

Application Layer	Peer-to-Peer-networks, e.g. Telnet = Telecommunication Network FTP = File Transfer Protocol HTTP = Hypertext Transfer Protocol SMTP = Simple Mail Transfer Protocol
Transport Layer	TCP = Transmission Control Protocol UDP = User Datagram Protocol
Internet Layer	IP = Internet Protocol ICMP = Internet Control Message Protocol IGMP = Internet Group Management Protocol
Host-to-Network Layer or Link Layer	device drivers (e.g. Ethernet or Token Ring drivers)

Table 1: Layers of the Internet [MS07]

PCs and the network access for individuals caused the internet to grow fast and to be the preferred way of communication. The protocols of the internet enable computers produced by different companies and with different system softwares to communicate with each other [MS07, Dav89].

1.1 The Different Layers of the Internet

The Internet is not an independent network, it is more a network inbetween lokal networks. As shown in table 1, the internet consists of four layers. The lowest layer of the internet is the link layer, which is the closest layer to the physical connection of computers. There can be arbitrary networks, such as Ethernet, Token Ring or Wireless Lokal Area Networks (WLAN) at the bottom of the internet layers.
The most important layers are the internet and the tranport layer. This composition will focus on the the internet layer and its protocols, whereas the next composition will focus on the transport layer and its protocols TCP and UDP. The internet layer concerns the transport of datagrams, also called packets, from the originating host across the network to the destination host. Its mainprotocol, the Internet Protocol (IP) has two subprotocols: the Internet Control Message Protocol (ICMP) and the Internet Group Message Protocol (IGMP).

In each layer, the communicating computers exchange control information. This information can be placed as a header in front of the packet or at the end of the packet as a trailer. Figure 1 shows how headers and trailers are added to a messages as it passes through the different layers. An application adds its header to the user data. Then the TCP protocol adds its header to this application data. Afterwards the IP protocol adds its header. All in all the IP header, TCP header and the application data are called an IP datagram. The length of an IP datagram is 46 to 1500bytes. At last, the Ethernet driver will add its header and trailer. The TCP header is only read at the final destination, whereas the IP header is changed as the packet passes through a router [MS07].

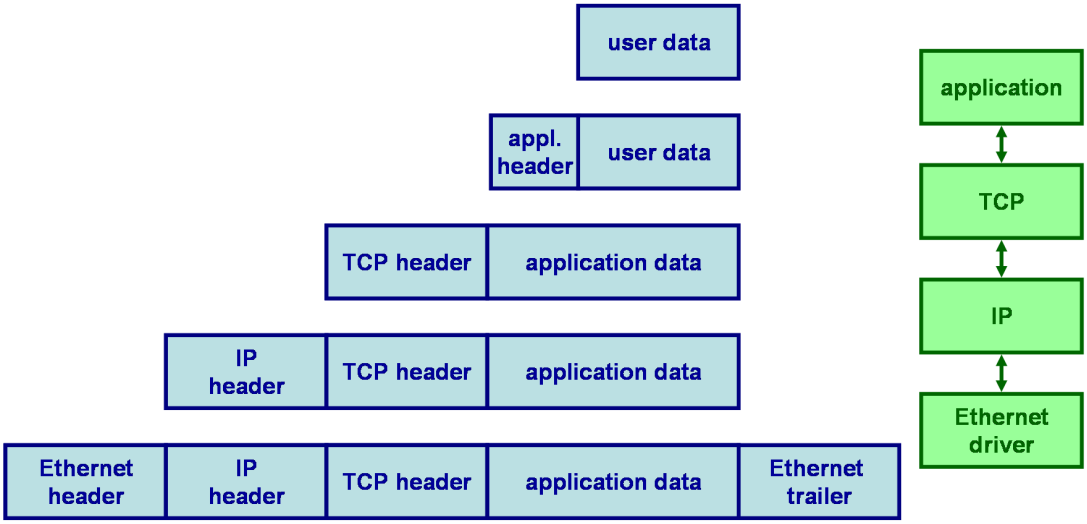

Figure 1: onion-like data encapsulation

2 IPv4

The Internet Protocol (IP) is the main protocol of the Internet Layer. IP and its two subprotocols ICMP and IGMP are used for the connection of lokal networks, e.g. to deliver datagrams. A datagram contains of a datapackage, the addresser's and the receiver's addresses. The task of IP is to transmit datagrams without any changes from addresser to receiver. If a direct connection inbetween receiver and addresser is impossible then intermediate computers, called routers, are used. Troubleshooting of IP is held very simple. If there is a problem with delivering a datagram, then the datagram is deleted and an ICMP-message is sent to its addresser. If also a problem with this ICMP-message appears, it is deleted as well but no other ICMP-message is sent. No copies of the datagram are created; consequently if a router is not working, all its datagrams are lost. Therefore IP is called an unreliable service. Delivery of data works as a Hop-to-Hop protocol from one router to the next one, where each datagram is treated independently. Therefore the delivery of the datagrams to the destination computer can be out of order.

Today, two versions of the Internet Protocol are in use: version 4 and version 6. Version 5 was skipped since it was developed but never introduced to public. Rightnow there is a change from IPv4 to IPv6 to solve some historical restrictions.

Figure 2 shows the header of an IP datagram when version 4 of the Internet Protocol is used. All in all, this header is at least 20 Bytes long.

Version The version field contains the versionnumber of the used Internet Protocol.

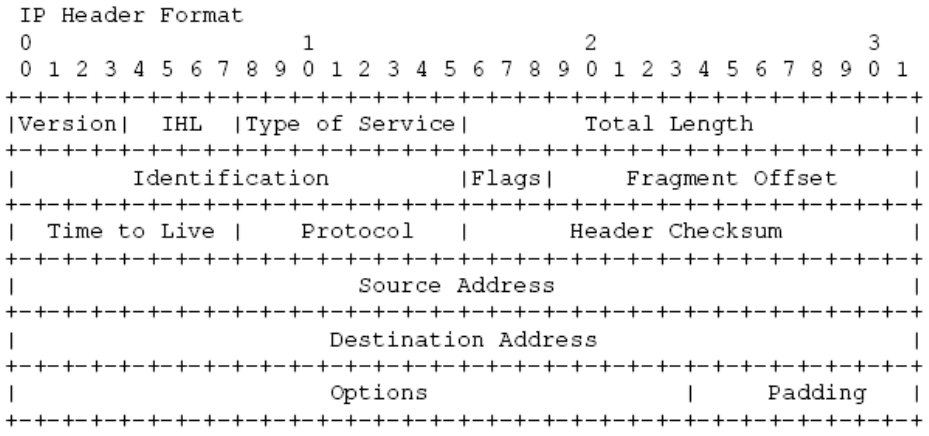

Figure 2: IPv4 header

IHL The IP Header Length (IHL) contains the length of the IP-header as a multiple of 32-Bit. The minimum value is 5. The Length of the IP header is not always the same, since it can contain a variable number of options.

Type of Service (TOS) Four bits of this fiels are used to specify weather delay, throughput, reliability or monetary cost should be optimized. But in fact, nearly all routers take no notice of this entry.

Total Length This entry contains the total length of the IP-packet. This entry has a maximum value of 64 kBytes and therefore IPv4 limits the packet size to 64 kBytes. Usually, this entry only contains a value of some kBytes. The minimum value that this field can contain is 20bytes, which is the length of a header and zero bytes data.

Identification This entry identifies a packet uniquely. Usually this value is increased by one with every sent packet.

Time to Live (TTL) This entry contains the maximum number of routers which the packet may pass. Each router decreases this entry by one. If this entry is zero and the packet's destination is not yet reached, an ICMP-message is sent and the IP-packet (the datagram) is discarded. If the packet had already been an ICMP-message, no further ICMP-message is created.

Protocol When a packet has reached its destination, it needs to be passed on to the next layer. This field states which protocol, such as TCP or UDP is used in the transport layer.

Checksum This checksum only refers to the IP Header and is used to find transmission errors. It is compared at each hop and useless datagrams or datagrams with a

transmission errors in the header are eliminated. This checksum is not used to check whether there is a error within the transmitted data.

Source Address This field contains the 32-Bit-IP-Address of the source vertex of the packet.

Destination Address This field contains the 32-Bit-IP-Address of the destination vertex of the packet.

Options Using the options field, additional options, such as safety- and administration restrictions for military reasons, recorded route, loose or strict source routing, can be specified.

Padding Padding is needed to ensure that the IP header ends at a 32-Bit boundary [Dav89].

An IP address identifies the interface of a computer, which can be reached through the internet. It contains two parts: the Net-ID, used to specify the subnet in which the computer is and the Host-ID, used to specify the computer within the subnet. Since not all subnets contain the same amount of computers, there is a variable boundary inbetween these two numbers. To specify this boundary, a subnetmask is used. An IP address as well as a subnetmask is 32bit long. A subnetmask starts with a sequence of ones, followed by a sequence of zeros. The number of ones denotes how many binary digets of the IP address refer the Net-ID. So if we used the following subnetmask for the following IP

IP-Address: `10000100.11100110.10010110.11110011`

Subnetmask: `11111111.11111111.11111111.00000000`

`10000100.11100110.10010110` would be the binary version of the Net-ID and `11110011` would be the Host-ID. Note that the dots are only used for a better legibility.

For a user it would be rather inconvenient to enter such an IP address. Therefore, the Domain Name System (DNS) was established, making it possible to enter names such as `cone.informatik.uni-freiburg.de` rather than number. These names are linked to the right IP addresses through a database provided by DNS [WR].

3 Routing and Packet Forwarding

When transmitting data packets, IP distinguished between routing, which refers to choosing the path for a packet, and packet forwarding, which is the actual delivery of the packet. Routing can either happen automatically using an algorithm or manually by the administrator. Its result is saved in Routing-Tables, where the information is saved to which vertex a packet needs to be delivered next in order to go to its destination. If the destination address of an incoming IP packet is the interface's IP, then it is delivered to the transport layer. Otherewise it is forwarded using the information of the routing table. If the packet's delivery address is not found in the routing table, it is delivered to a default gateway. The TTL-mechanism prevents an infinite packet forwarding. Infinite packet forwarding could

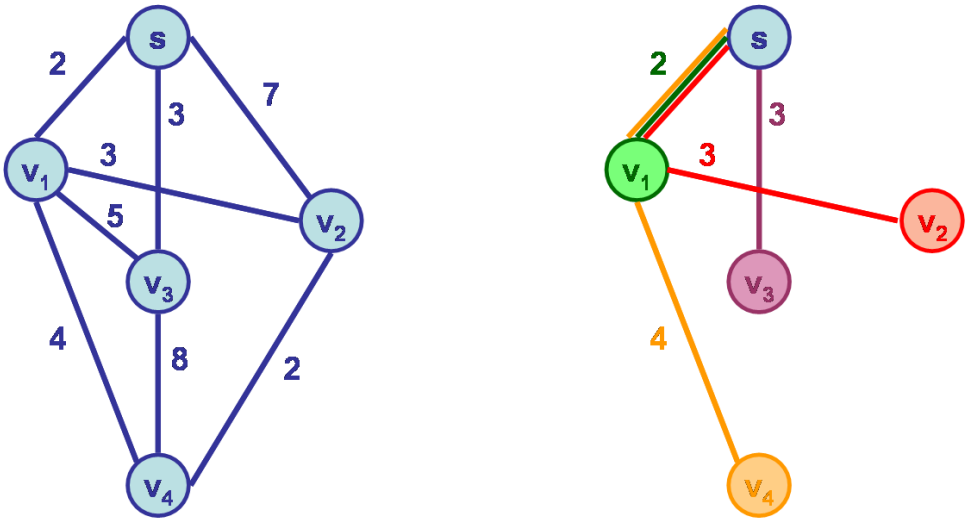

Figure 3: On the left a weighted graph G is shown. On the right its shortest path tree is shwon, which can be created by using Dijkstra's Algorithm.

happen if there is an error in the routing tables and the routing tables would imply a circle. Then the packet is send around in this circle until its TTL-entry is zero and an ICMP-message is created. This ICMP-message has, as every IP-packet, also a TTL field and consequently it can only cover a certain distance before being discarded [Mal].

3.1 The Shortest Path Problem

When a routing table is created, the shortest path problem is solved. The network is modelled by using a weighted graph. Vertices denote routers. If there is an edge incident with two vertices, it means that the two routers denoted by the vertices are connected directly. The weights on the edges can stand for the time or cost needed to forward a packet inbetween those two routers. In this section the term weight is used synonymously with distance.

Shortest Path Problem

Given is a weighted graph $G = (V, E)$, a starting vertex $s \in V$ and weights $w_e \in \mathbb{R}_0^+$ for all edges $e \in E$. Searched are all paths from the starting vertex to all other vertices, which have the least sum of weights.

The Shortest Path Tree is part of the union of the shortest paths from the starting vertex to all other vertices. If there is more than one possibility for a shortest path from the starting vertex to another vertex, only one is presented in the shortest path tree. Note

Vertex	Predecessor
v_1	s
v_2	v_1
v_3	s
v_4	v_1

Table 2: Table of Predecessors for the graph G of Figure 3

that choosing a different starting vertex results in a different tree.

3.2 Dijkstras Algorithm

Dijkstra's Algorithm is used to calculate the shortest path tree of a weighted graph. The algorithm yields a table of predecessors, which describes the edges $(\text{Predecessor}(u), u)$ of the shortest path tree. Figure 3 shows an example of a shortest path tree of the graph G and table 2 shows the table of predecessors for G. Dijkstra's Algorithm builds the Shortest Path Tree gradually beginning with the starting vertex. It is a greedy algorithm so it chooses gradually the next best element.

Dijkstra's Algorithm

Input: $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_0^+$ **for all** $v \in V$ **do**
    Distance$[v] \leftarrow \infty$
    Predecessor$[v] \leftarrow v$
Distance$[s] \leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Element from Q with minimal value Distance$[u]$
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for all $v \in V : (u, v) \in E$ **do**
        if Distance$[u] + w_{(u,v)} <$ Distance$[v]$ **then**
            Distance$[v] \leftarrow$ Distance$[u] + w_{(u,v)}$
            Predecessor$[v] \leftarrow u$
return $\{(\text{Predecessor}(u), u) | u \in V \text{ and } u \neq \text{Predecessor}(u)\}$

In each run of the while loop, the vertex which has the shortest distance and is not yet part of the tree, is added to the tree. For a more detailed explanation see appendix A. This construction of the while loop is also used to proof that the algorithm is correct. Assume that the shortest path to the newly chosen vertex v would have to go through

Distance Table for A

from A	over B	over E	Routing Table Entry
to B	2	11	B
C	3	19	B
D	7	10	B
E	13	9	E

Distance Table for C

from C	over B	over D	over E	Routing Table Entry
to A	3	14	18	B
B	1	9	0	B
D	6	4	11	D
E	7	5	10	D

Table 3: Distance Vector Tables for the graph in figure 4
Each row is used to determine the shortest path to one other vertex. This path could start with either one of the starting vertices neighbors. In a column the lenghts of different paths to all other vertices starting with one particular neighbor of the starting vertex are displayed. The last column displays the routing table entry, here the name of the neighbor of the starting point with the shortest path appears.

another vertex w which is not yet part of the tree, then the distance of w is shorter than the distance of v and so v cannot be the vertex with the minimum distance [Cho, Wil96].

3.3 Practical Realizations of Routing Algorithms

One practical realization of Dijkstra's Algorithm is Link-State-Routing. Here the routing tables are determined with Dijkstra's Algorithm. Therefore every vertex needs information about the entire network. Information about the connections of the network are spread by broadcast which means that one router sends information about its connections to all its neighbors. At first every router or computer spreads the information about its current connections, later, in the running state, only information about changes in the network is exchanged. So the costs for communication in a running state are quite low.

The distance vector routing algorithm uses distance-vector-tabels to save distances of all vertices. An example is given in table 3 which refers to figure 4. In the ideal case the distances in the table of a vertex v are the shortest path using a certain first edge to one of v's neighbors. To create such a table, Dijkstra's algorithm does not need to be used: If one vertex sends all information about its neighbors to all its neighbors, they can update their table and pass this information on. This distributed updating process converges to the correct entries.

Using distance vector tables, the count-to-infinity problem can appear. An example is given in figure 4 with the routers A, B and C. At first, there is a connection inbetween B and A and a connection inbetween B and C. Then C is disconnected, so B cannot reach C anymore directly. Therefore the column with the entries about paths over C are deleted. Consequently router B thinks, that C can be reached through router A. B spreads this change in its routing table. Consequently A also increases its values for going to C and spreads this information through the network. When this update reaches router B, it increases its values for going to C again. Consequently bad news, as the disconnection of a router, take a long time to spread through the network, whereas good news spread quickly

Distance Table for A

from A	over B	Routing Table Entry
to B	2	B
C	3	B

Distance Table for B

from B	over A	over C	Routing Table Entry
to A	2	4	A
C	5	1	C

Distance Table for A

from A	over B	Routing Table Entry
to B	2	B
C	3	B

Distance Table for B

from B	over A	over C	Routing Table Entry
to A	2	-	A
C	5	-	A

from A	over B	Routing Table Entry
to B	2	B
C	7	B

from B	over A	over C	Routing Table Entry
to A	2	-	A
C	5	-	A

from A	over B	Routing Table Entry
to B	2	B
C	7	B

from B	over A	over C	Routing Table Entry
to A	2	-	A
C	9	-	A

Table 4: An example for the count-to-infinity problem: Router B looses the connection to router C, but thinks that C can still be reached using router A, who uses router B to reach C.

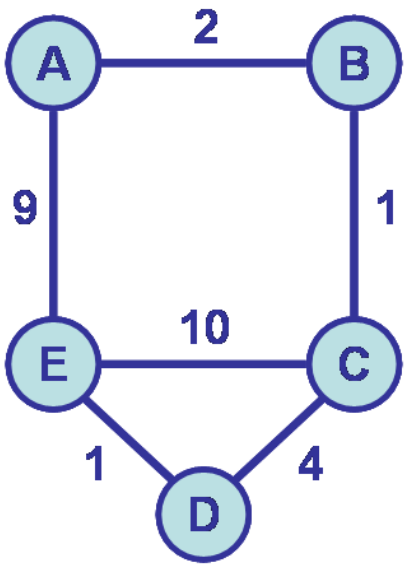

Figure 4: Graph for table 3

[MS07].

4 Autonomous Systems

Link-State-Routing and Distance-Vector-Routing need an entry for each computer of the network, Link-State-Routing also requires information about all connections of the network. Since the internet is a network of millions of computers, it is not possible to generate routing-tables for the entire network. Therefore the internet is hierarchically partitioned into autonomous systems. An autonomous system is a subnet of the internet with its own administration. Ususally an autonomous system belongs to an internet service provider, e.g. a university or a company. Each autonomous system has at least one boundary router, which connects it to the other boundary systems. Now the size of routing tables only depends on the size of the own autonomous system. Packet Forwarding now includes two parts: Routing within the own autonomous system, which is called Intra-AS-Routing, and Inter-AS-Routing, which means routing inbetween autonomous systems.

4.1 Intra-AS-Routing

Here, three protocol for Intra-AS-Routing are presented:

Routing Information Protocol (RIP) RIP uses distance-vector-tables. When a computer is newly connected to the network, it only has information about its neighbors and builds its routing-table gradually stating with this information. To spread information about new computers or disconnected computers, routing tables are ex-

changed frequently, if there have been changes or not. The offer of new routers is called advertisement. If after a certain time no advertisement has happened, and therefore nothing changed to the entry of a routing-table, the entry is erased. Therefore the route over the neighbor is declared impossible and a fast reaction to topological changes is possible. To check if a connection is still possible, advertisements are sent to this neighbor. If any changes appeared in the neighbor's table, it will also send its information back and it is recorded again in the table. Using RIP, routing-information does not spread fast through the network and the count-to-infty problem appears [WR, Mal].

Open Shortest Path First (OSPF) The issues of RIP are solved, when using OSPF. OSPF uses Dijkstra's algorithm to calculate the shortest paths inbetween routers. The information about the network connections is saved as a list of connected computers, called Link-State-Database. Changes of routing information are spread by advertisements. Lager networks are devided into two layers: the local area and the backbone. Local area border routers summarize all distances in their local area and offer them to other border routers as advertisements. The backbone router is used to administrate the entire network, using Dijkstra's algorithm to calculate the shortest path inbetween border routers. OSPF is used today as a routing protocol for IPv4 and IPv6 [WR, Mal].

Interior-Gateway-Routing-Protocol (IGRP) Another issue of RIP is, that when a router is more than 15 vertices away, this router will be called out of reach. The intension when developing IGRP in 1980 was to overcome this maximum distance of 15 vertices. IGRP can also create metrics of costs using information about available bandwidth, the resulting delay, the capacity reliability and the efficency of this capacity. Usually the bandwidth and its delay are used to calculate the costs of a path [WR, Mal].

4.2 Inter-AS-Routing

Earlier the Exterior-Gateway-Protocol (EGP) was used for routing inbetween autonomous systems. Today EGP is replaced by the Border Gateway Protocol (BGP). The basis of BGP is the Path-Vector-Protocol which is similar to the earlier presented Distance-Vector-Protocol. The difference is that using the Path-Vector-Protocol, all information about a path is saved. So the count-to-infinity problem is solved since paths with multiple use of one vertex can be discarded [MS07, Mal].

5 Other Services of IP

IP is not only used for routing and packet forwarding, but also includes other services.

ICMP = Internet Control Message Protocol is used to send control messages, e.g. when the TTL-counter is expired before a packet has reached its destination.

Ping was developed to test the connection inbetween two computers. An ICMP-Echo-Request-packet is sent to a receptor who replies with an ICMP-Echo-Reply-packet.

Traceroute is an algorithm which can return information about the route connecting two computers. Traceroute sends ICMP packets to a destination, where the TTL-values are set 1, 2, 3, ... and waits for ICMP error messages. These error messages are evaluated to give a list of all routers that are used on the path to the destination [MS07].

References

[Cho] D. Chowdhury. *Unified IP Internetworking.*

[Dav89] J. Davidson. *An Introduction to TCP/IP.* Springer-Verlag, New York, 2 edition, 1989.

[Mal] R. Malhotra. *IP Routing.*

[MS07] P. Mahlmann and C. Schindelhauer. *Peer-to-Peer-Netzwerke.* Springer, Heidelberg, 2007.

[Wil96] R. Wilson. *Introduction to Graph Theory.* Prentice Hall, Essex, 4 edition, 1996.

[WR] J.D. Wegner and R. Rockell. *IP Addressing & Subnetting.*

A Dijkstra's Algorithm

The following picture series explains how Dijkstra's Algorithm works step for step.

Input: $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_0^+$
for all $v \in V$ **do**
    Distance[v] $\leftarrow \infty$
    Predecessor[v] $\leftarrow v$
Distance[s] $\leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Element from Q with minimal value Distance[u]
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for all $v \in V : (u, v) \in E$ **do**
        if Distance[u] + $w_{(u,v)}$ < Distance[v] **then**
            Distance[v] $\leftarrow$ Distance[u] + $w_{(u,v)}$
            Predecessor[v] $\leftarrow u$
return $\{(\text{Predecessor}(u), u) | u \in V \text{ and } u \neq \text{Predecessor}(u)\}$

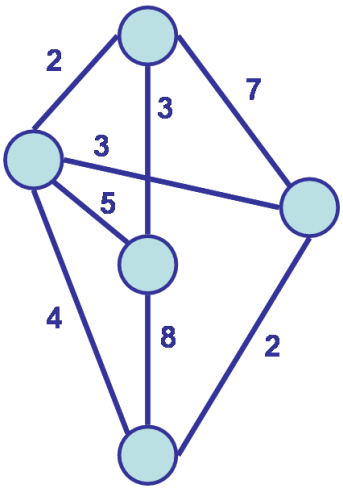

Figure 5: The Input for Dijkstra's Algorithm is a weighted graph G.

Input: $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_0^+$
for all $v \in V$ **do**
    Distance[v] $\leftarrow \infty$
    Predecessor[v] $\leftarrow v$
Distance[s] $\leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Element from Q with minimal value Distance[u]
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for all $v \in V : (u, v) \in E$ **do**
        if Distance[u] + $w_{(u,v)}$ < Distance[v] **then**
            Distance[v] $\leftarrow$ Distance[u] + $w_{(u,v)}$
            Predecessor[v] $\leftarrow u$
return $\{(\text{Predecessor}(u), u) | u \in V \text{ and } u \neq \text{Predecessor}(u)\}$

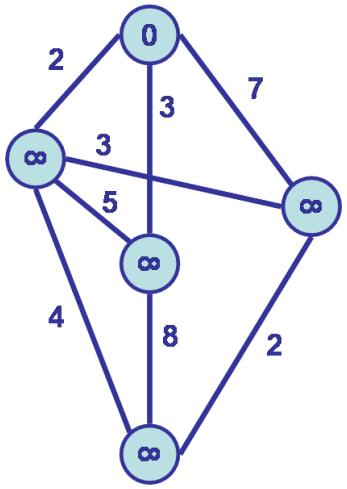

Figure 6: At first the distance of all vertices is set to infinity, and the predecessor of a vertex is set to be itself. In the example graph the distance of a vertex is the number written within that vertex. The distance of the starting vertex is set zero.

Input: $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_0^+$
for all $v \in V$ **do**
    Distance[v] $\leftarrow \infty$
    Predecessor[v] $\leftarrow v$
Distance[s] $\leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Element from Q with minimal value Distance[u]
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for all $v \in V : (u, v) \in E$ **do**
        if Distance[u] + $w_{(u,v)}$ < Distance[v] **then**
            Distance[v] $\leftarrow$ Distance[u] + $w_{(u,v)}$
            Predecessor[v] $\leftarrow u$
return $\{(\text{Predecessor}(u), u) | u \in V \text{ and } u \neq \text{Predecessor}(u)\}$

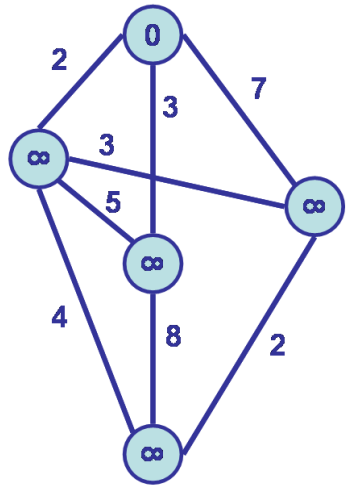

Figure 7: There are two sets defined. S is at first the empty set. When a vertex belongs to S, it will be colored green. The other set is V, which contains at first all vertices of G. A vertex of the example graph belongs to V, when it is not colored green (here always blue, but later on it might be purple or orange as well).

Input: $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_0^+$
for all $v \in V$ **do**
    Distance[v] $\leftarrow \infty$
    Predecessor[v] $\leftarrow v$
Distance[s] $\leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Element from Q with minimal value Distance[u]
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for all $v \in V : (u, v) \in E$ **do**
        if Distance[u] + $w_{(u,v)}$ < Distance[v] **then**
            Distance[v] $\leftarrow$ Distance[u] + $w_{(u,v)}$
            Predecessor[v] $\leftarrow u$
return {(Predecessor(u), u)|$u \in V$ and $u \neq$ Predecessor(u)}

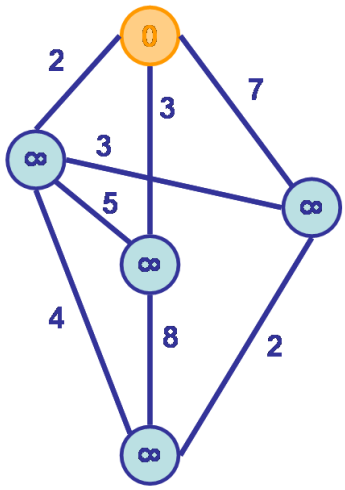

Figure 8: The element with the minimal distance is the starting vertex. In the example this element is colored orange.

Input: $G = (V, E)$, $s \in V$, $w : E \to \mathbb{R}_0^+$
for all $v \in V$ **do**
    Distance[v] $\leftarrow \infty$
    Predecessor[v] $\leftarrow v$
Distance[s] $\leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$ **do**
    $u \leftarrow$ Element from Q with minimal value Distance[u]
    $S \leftarrow S \cup \{u\}$
    $Q \leftarrow Q \setminus \{u\}$
    for all $v \in V : (u, v) \in E$ **do**
        if Distance[u] + $w_{(u,v)}$ < Distance[v] **then**
            Distance[v] $\leftarrow$ Distance[u] + $w_{(u,v)}$
            Predecessor[v] $\leftarrow u$
return {(Predecessor(u), u)|$u \in V$ and $u \neq$ Predecessor(u)}

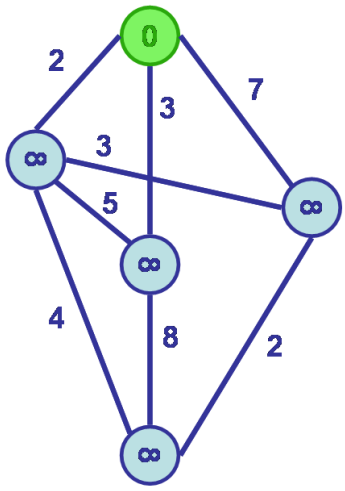

Figure 9: The vertex u is removed from the set Q and added to the set S, therefore the starting vertex is now colored green in the example graph.

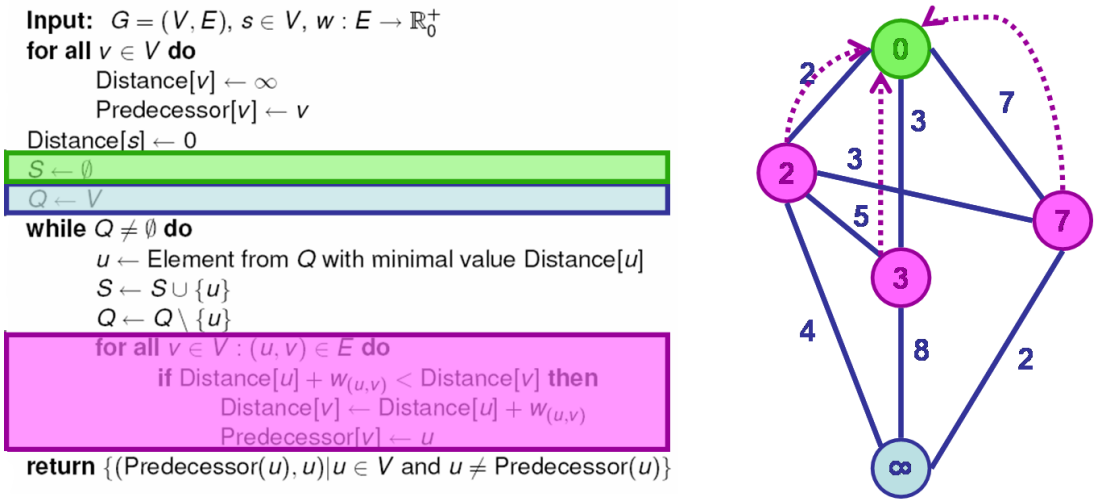

Figure 10: In the next step we look at all vertices, which are adjacent to u. In the example graph they are colored purple. Then it is checked whether their distance is greater than the distance that can be calculated by using vertex u. If the distance using vertex u is less than the current distance of that vertex, its predecessor is set to u. This is indicated by the dotted arrows in the example graph.

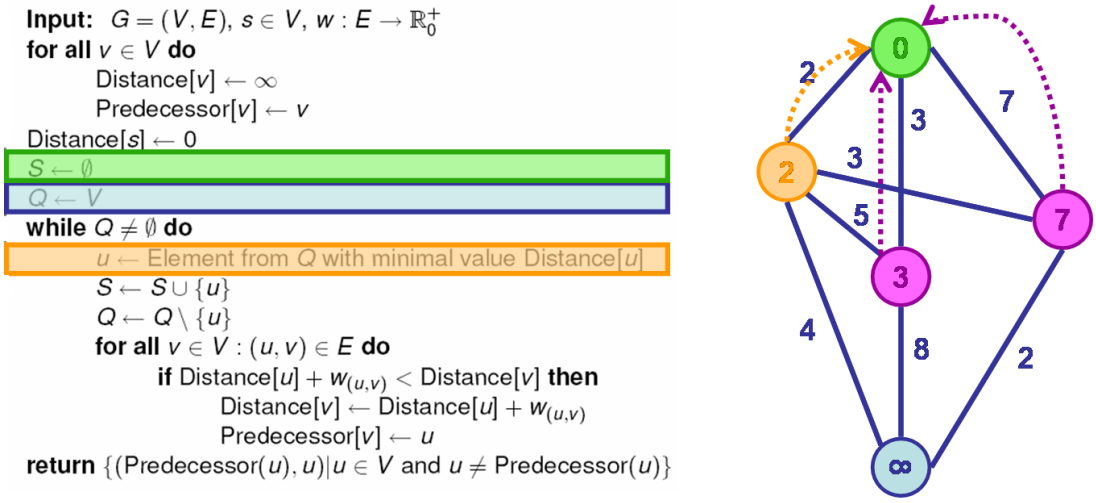

Figure 11: Now the next run of the while loop starts. Once again a vertex u from the set Q is chosen, which is colored orange in the example graph.

```
Input: G = (V, E), s ∈ V, w : E → ℝ₀⁺
for all v ∈ V do
        Distance[v] ← ∞
        Predecessor[v] ← v
Distance[s] ← 0
S ← ∅
Q ← V
while Q ≠ ∅ do
        u ← Element from Q with minimal value Distance[u]
        S ← S ∪ {u}
        Q ← Q \ {u}
        for all v ∈ V : (u, v) ∈ E do
                if Distance[u] + w(u,v) < Distance[v] then
                        Distance[v] ← Distance[u] + w(u,v)
                        Predecessor[v] ← u
return {(Predecessor(u), u)|u ∈ V and u ≠ Predecessor(u)}
```

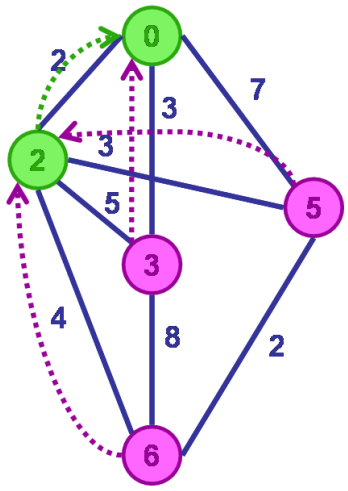

Figure 12: Note that the predecessor and the distance of the vertex on the right have changed. Its new distance is 5, since it can be reached from the starting vertex over the green coloured vertex on the right.

```
Input: G = (V, E), s ∈ V, w : E → ℝ₀⁺
for all v ∈ V do
        Distance[v] ← ∞
        Predecessor[v] ← v
Distance[s] ← 0
S ← ∅
Q ← V
while Q ≠ ∅ do
        u ← Element from Q with minimal value Distance[u]
        S ← S ∪ {u}
        Q ← Q \ {u}
        for all v ∈ V : (u, v) ∈ E do
                if Distance[u] + w(u,v) < Distance[v] then
                        Distance[v] ← Distance[u] + w(u,v)
                        Predecessor[v] ← u
return {(Predecessor(u), u)|u ∈ V and u ≠ Predecessor(u)}
```

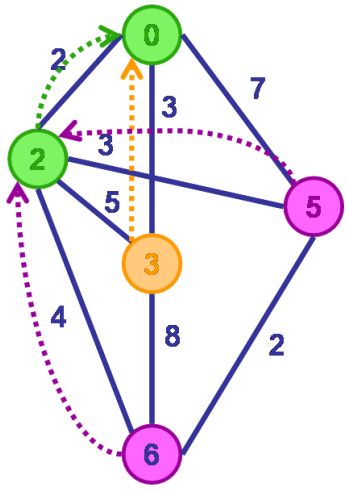

Figure 13: The next run of the while loop begins. Note that all vertices that are not colored green belong to the set Q.

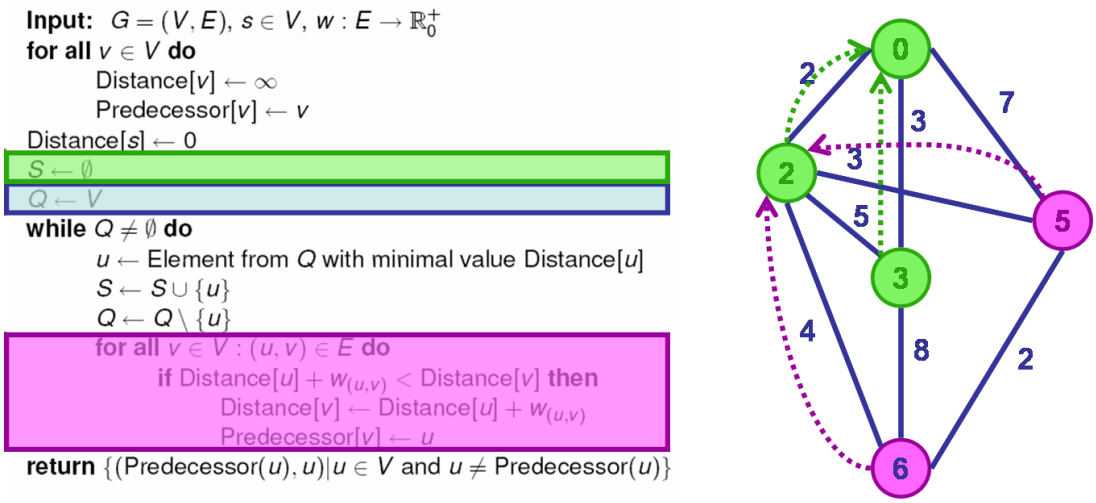

Figure 14: Move vertex u from set Q to set S and check the distance of its neighbors.

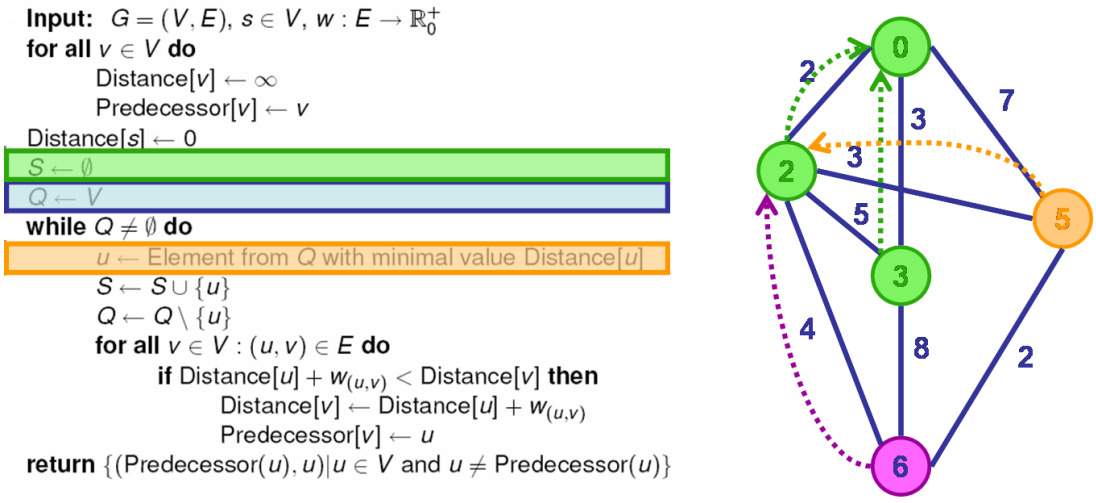

Figure 15: Next run of the while loop.

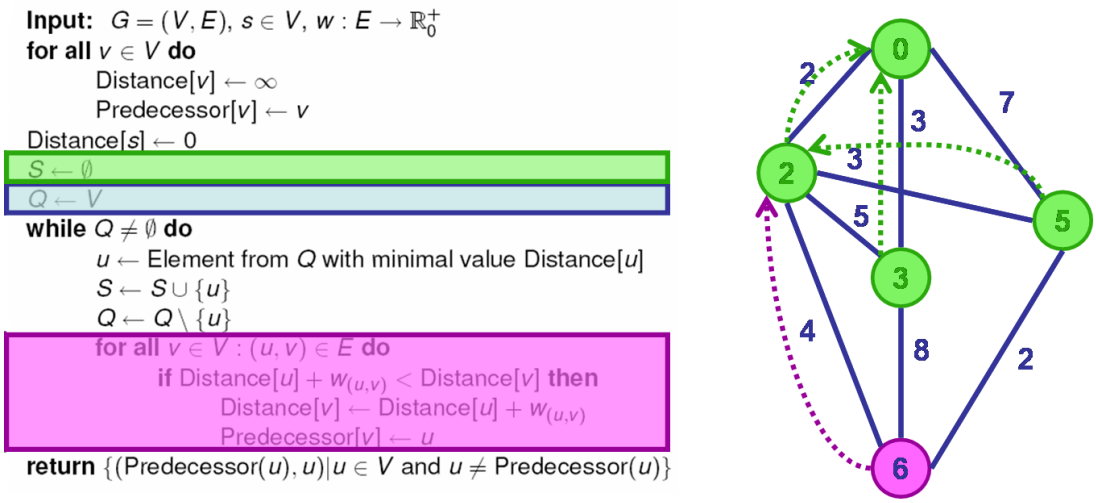

Figure 16: As in 14: Moving vertex u and Checking its neighbors.

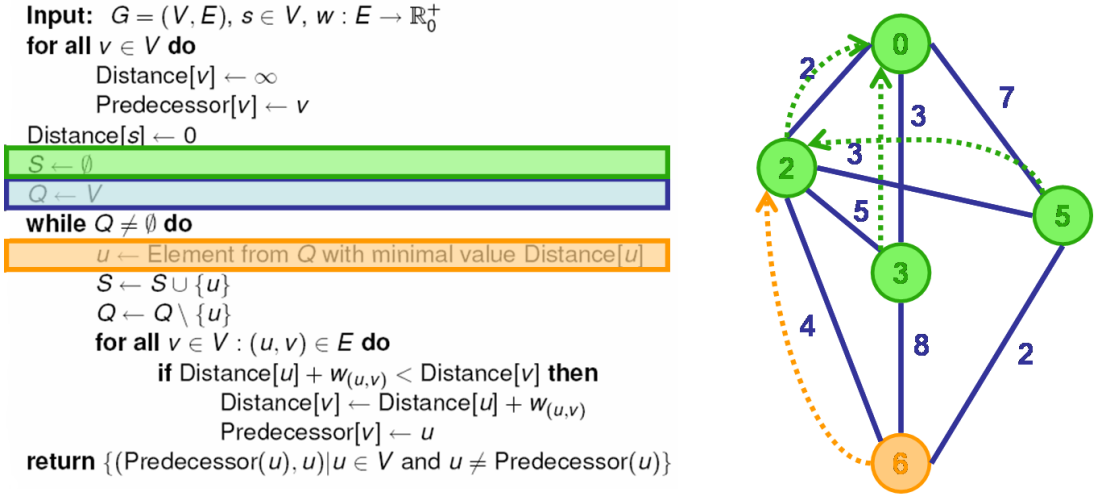

Figure 17: The last run of the while loop begins.

```
Input: G = (V, E), s ∈ V, w : E → ℝ₀⁺
for all v ∈ V do
        Distance[v] ← ∞
        Predecessor[v] ← v
Distance[s] ← 0
S ← ∅
Q ← V
while Q ≠ ∅ do
        u ← Element from Q with minimal value Distance[u]
        S ← S ∪ {u}
        Q ← Q \ {u}
        for all v ∈ V : (u, v) ∈ E do
                if Distance[u] + w(u,v) < Distance[v] then
                        Distance[v] ← Distance[u] + w(u,v)
                        Predecessor[v] ← u
return {(Predecessor(u), u)|u ∈ V and u ≠ Predecessor(u)}
```

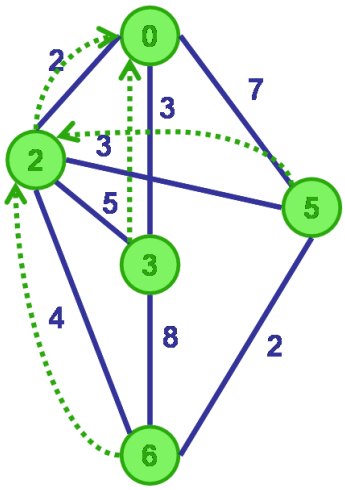

Figure 18: Now the last vertex from set Q is moved to set S. S contains all vertices of G; and Q has become the empty set, therefore the while loop is finished.

```
Input: G = (V, E), s ∈ V, w : E → ℝ₀⁺
for all v ∈ V do
        Distance[v] ← ∞
        Predecessor[v] ← v
Distance[s] ← 0
S ← ∅
Q ← V
while Q ≠ ∅ do
        u ← Element from Q with minimal value Distance[u]
        S ← S ∪ {u}
        Q ← Q \ {u}
        for all v ∈ V : (u, v) ∈ E do
                if Distance[u] + w(u,v) < Distance[v] then
                        Distance[v] ← Distance[u] + w(u,v)
                        Predecessor[v] ← u
return {(Predecessor(u), u)|u ∈ V and u ≠ Predecessor(u)}
```

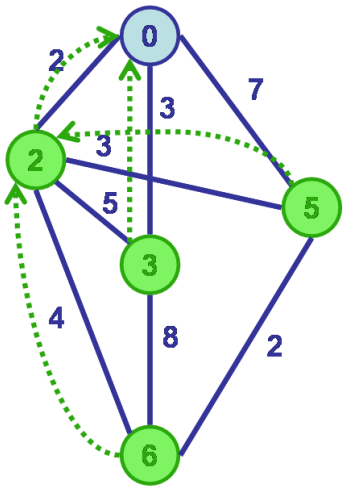

Figure 19: As a result the algorithm returns a table of predecessors for each vertex of G, except the starting vertex s. The predecessor of a vertex is indicated by the dotted arrow.

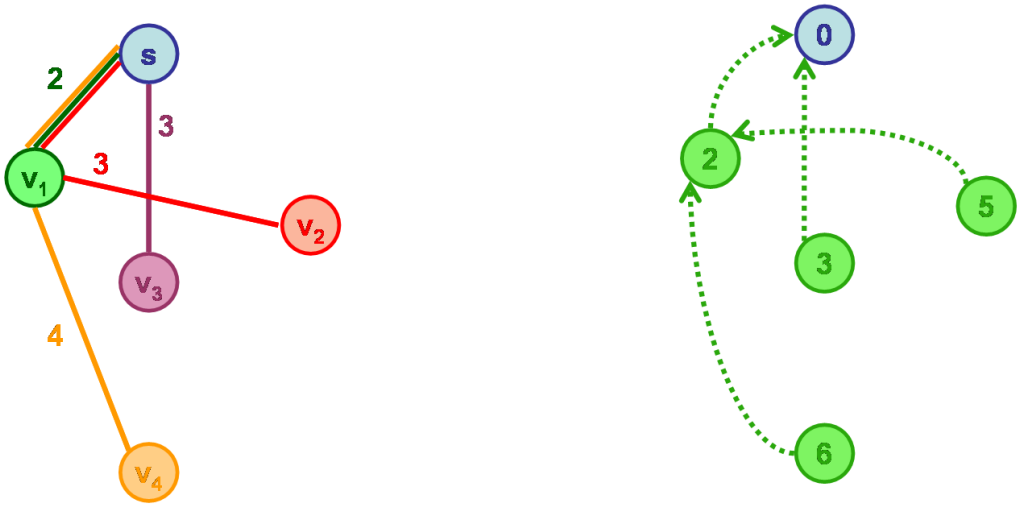

Figure 20: Note that if G is a graph on n vertices, this table of predecessors contains the $n-1$ edges of a spanning tree of G. This spanning tree is the shortest path tree of G as can be seen by comparison with figure 3. To find the shortest path from an arbitrary vertex v to the the starting vertex, one has to use the path containing v, its predecessor v', the predecessor of v' and so on until s is the predecessor.