Computer Systems Tutorial

December 8^{th} 2023

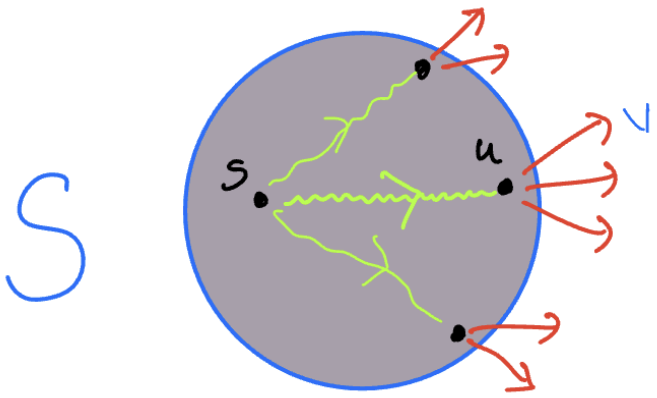
1 Preliminaries

Graph A graph is a pair of two sets: a set of *edges* and a set of *verticies* or nodes. We will denote it like (V, E) where elements of E are also pairs like (u, v) that stands for an edge from vertex u (the source) to vertex v (the target). We will then consider weighted graphs such that edges are of the form (u, w, v) where w denotes the weight of an edge, e.g. how 'costly' it is to follow that particular edge. A *path* from vertex u to vertex v in a graph is a sequence of nodes $(v_1, \ldots, v_n)$ such that the first vertex in the sequence $v_1 = u$ and the last vertex is $v_n = v$ and there is an edge between v_i and v_{i+1} for all $i < n$.

Dijkstra Algorithm This algorithm finds the distances of shortest paths from one source vertex v to all other vertices in a weighted graph such that all weights are positive. The pseudo-code is given below.

```
# s is the source vertex
    fun dijkstra(s):
    # D stores distances from s to all other vertices
        D = {}
        S = {s}

        for v in V:
            if (s,v) in E:
                D[v] = c(s,v) # c is the cost of the edge
            else:
                D[v] = float('inf')

        while S != V:
            v = min(D) and v not in S
            S.add(v)

            for (v,u) in E:
                D[u] = min(D[v] + c(v,u), D[u])

```

S is the set of currently visited vertices v' and it is guaranteed that the distance from s to $v' \in S$ is the shortest. Then the algorithm picks a vertex v not from S such that the distance to it from the source node is minimal across all candidates and updates the distances to all v's neighbours.



Note that if one also would like to retrieve the shortest paths themselves, it is needed to maintain the predecessor of each vertex, e.g. in an array *pred*, and then, to get the path from s to some given vertex v we need to trace $pred[v]$ until we reach s and reverse the sequence. For example, if $pred = [0, 0, 1, 2]$ (the value in the array for index i gives the predecessor of vertex i) and $v = 3$ and $s = 0$ it would mean that the predecessor of s is s, the predecessor of vertex 1 is s, the predecessor of vertex 2 is vertex 1 and the predecessor of vertex 3 is vertex 2. So the path from s to v is $(s, 1, 2, v)$. The changes in the code would be the following.

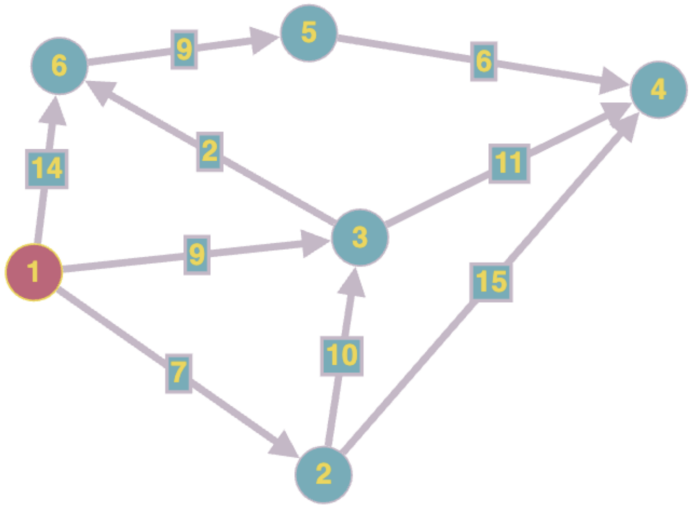
```
for v in V:
    if (s,v) in E:
        D[v] = c(s,v)
        pred[v] = s
```

and

```
for (v,u) in E:
    dist = D[v] + c(v,u)
    if dist < D[u]:
        D[u] = dist
        pred[u] = v
```

2 Exercises

Exercise 1 Find the distances of all shortest paths from source 1 in the graph below by using Dijkstra algorithm.



What is the set S at each step? What are the distances at each step?

Solution We will maintain the sets S and D across the iterations. First, according to lines 3-12 of the pseudo-code, we need to initialise D and S and hence $S = \{s\}$ and we will represent D as a table:

Node	Distance
1	0
2	7
3	9
4	∞
5	∞
6	14

Where Distance corresponds to the distance from the source node (1) to the Node in Node column. Then we go to lines 14-15 and pick the node with the minimal distance to it from s such that the node is not in S. This is node 2. So now $S = \{1, 2\}$ and we need to update the distances from node 2 to all its adjacent nodes, that is, to nodes 3 and 4:

Node	Distance
1	0
2	7
3	9
4	22
5	∞
6	14

The previous distance to node 3 was 9 which is shorter than the distance to 2 plus 10 (the cost of the edge between 2 and 3) and hence we keep the old distance for 3. In contrast, the distance to 4 was ∞ and we replace it with the distance to 2 plus the cost of the edge from 2 to 4 (15). We again need to pick the node with the shortest distance from s and this time it is node 9. Hence, $S = \{1, 2, 3\}$ and we update distances to $4, 6$.

The next node we pick is 6 and $S = \{1, 2, 3, 6\}$ and we update the distance to 5.

Node	Distance
1	0
2	7
3	9
4	20
5	∞
6	11

Node	Distance
1	0
2	7
3	9
4	20
5	20
6	11

Now we can pick either 4 or 5 as our next node. Let's pick 4. It has no adjacent nodes, so we proceed with $S = \{1, 2, 3, 6, 4\}$ and pick node 5. The previous distance to 4 is shorter than the distance to 5 plus the cost of the edge from 5 to 4 (6) and hence we keep the previous distance. Since $S = V$ we exit the `while` loop. The distances are the following.

Node	Distance
1	0
2	7
3	9
4	20
5	20
6	11

Exercise 2 What is the complexity of the above algorithm in terms of V and E?

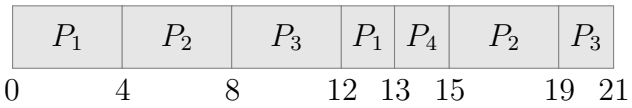
Solution The lines 4-12 take $\mathcal{O}(V)$ steps. The `while` loop performs at most V iterations. To find a minimum in an array D we need V steps. And the loop at line 17 takes no more than E steps. Hence, in total, we have $\mathcal{O}(V + V \times (V + E)) = \mathcal{O}(V^2 + VE)$. If we assume that there must be at most 1 edge between any two nodes, the number of steps in the loop at line 17 will be bounded by $V - 1$ (as any given node can be connected at most to every other node except itself) and the complexity will be $\mathcal{O}(V^2 + V^2) = \mathcal{O}(V^2)$. With a more sophisticated data structure for retrieving the minimum value, the complexity can be improved.

Exercise 3 Consider the following set of processes:

What is the average response time assuming Round Robin CPU scheduling policy with a Time Quantum (q = 4 ms). Also, note that newly arriving processes are added to the tail of the ready queue. The response time is the time it takes for the process to be first allocated a CPU after it arrives.

Process	Arrival Time	Burst time
P_1	0 ms	5 ms
P_2	2 ms	8 ms
P_3	3 ms	6 ms
P_4	5 ms	2 ms

Solution We will show the Gantt diagram and the ready queue at each timestamp.



Time	Queue (head to the right)
0	P_1
2	P_2
3	$P_3; P_2$
4	$P_1; P_3; P_2$
5	$P_4; P_1; P_3$
8	$P_2; P_4; P_1; P_3$
12	$P_3; P_2; P_4; P_1$
13	$P_3; P_2; P_4$
15	$P_3; P_2;$
19	P_3
23	

The response time for each process is given below (it is the time the process is first given the CPU minus its arrival time)

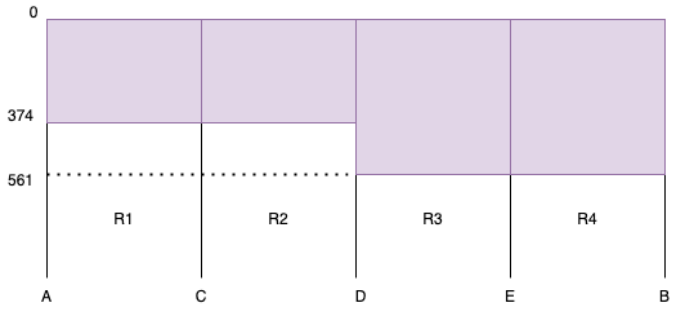
$$\begin{array}{cc} P_1 & 0-0=0 \\ P_2 & 4-2=2 \\ P_3 & 8-3=5 \\ P_4 & 13-5=8 \end{array}$$

And the average is $(0+2+5+8)/4 = \frac{15}{4} = 3.75$.

Exercise 4 Suppose Host A wants to send a large file to Host B. The path from Host A to Host B has four links, of rates R1 = 12 Mbps, R2 = 15 Mbps, R3 = 8 Mbps and R4 = 25 Mbps. What will be the minimum end-to-end delay of transferring a video clip of 535 MegaBytes? (We assume that there is no store-and-forward delay on the routers connecting the links)

Solution First thing we need to do is convert data size into bits because the bandwidth is measured in megabits. 535 MegaBytes is $535 \cdot 1024 \cdot 1024 \cdot 8 = 4\,487\,905\,280$ bits. We will also assume that once a bit of data has been sent it is immediately received by the receiver. Pictorially we can represent this scenario as follows.

As there is no propagation delay the first bit immediately arrives at B at time 0. Then, it takes 535 MegaBytes / R1's bandwidth seconds to send the data from A. In other words, the last bit sent from A



arrives at C at time 374 seconds. Since R2's bandwidth is higher than the one of R1 there is no delay between C receiving the data and forwarding it and R3 receives the last bit also at time 374 seconds. Then, R3's bandwidth is lower than R2's and thus D can not send the data over R3 at the same rate it receives it. It takes 535 MegaBytes / R3's bandwidth seconds to put all the data on channel R3, or 561 seconds (560.98816, to be precise). Because R4's bandwidth is higher than R3's there is no delay between E receiving and sending. Therefore, the transmission is bounded by the bandwidth of the slowest channel.

Exercise 5 In the **SYNACK** message of the TCP connection establishing handshake, the server sends a 1-byte TCP segment to the client. That segment has Sequence number = 900, SYN/ACKbits = 1. Which of the following TCP headers are correct for the corresponding **ACK** message from the client to the server?

a) Sequence number = 901, Acknowledgement number = 901, SYNbit = 0, ACKbit = 1

b) Sequence number = 500, Acknowledgement number = 901, SYNbit = 0, ACKbit = 1

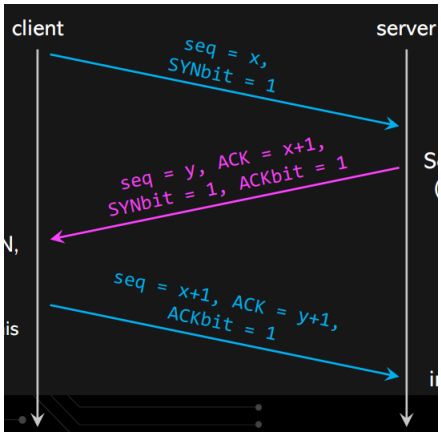
c) Sequence number = 901, Acknowledgement number = 483, SYNbit = 0, ACKbit = 1

d) Sequence number = 483, Acknowledgement number = 901, SYNbit = 1, ACKbit = 1

e) Sequence number = 901, Acknowledgement number = 901, SYNbit = 1, ACKbit = 1

f) Sequence number = 901, Acknowledgement number = 483, SYNbit = 1, ACKbit = 1

Solution Recall TCP handshake diagram.



TCP segment header

Offsets	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0 0				CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size															
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.)																															
⋮	⋮																																
56	448																																

and the TCP header structure.

We are interested in the lowest arrow which goes from the client to the server. According to the protocol, ACK bit should be 1, SYNbit should be 0 and the ACK number should be sequence number + 1 where the sequence number is from the server's response (900). That is, ACKbit = 1, SYNbit = 0, ACK = 901. Only the first two options fit these constraints. I encourage you to check out how actual TCP packets look like in this video: `https://youtu.be/3Zb_EebU22o?si=F9MnfE1mGEJv53nz`