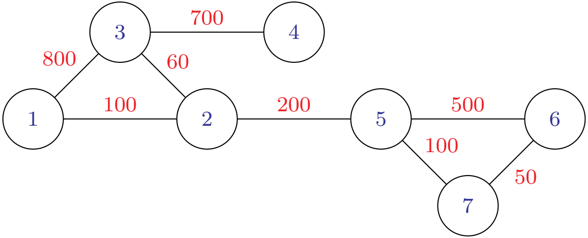
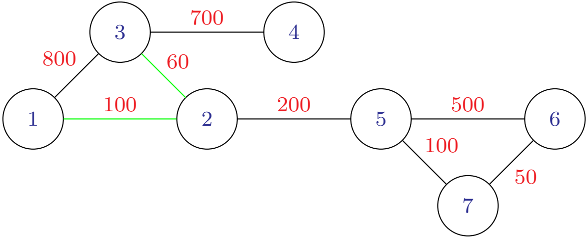
The Travel desk at CmI would like to reduce the money spent on airtickets.

- The rules permit CmI members to fly only by the state airline which uses a rather obscure pricing policy.
- It is not always the case that the cost of a ticket is proportional to the distance.
- The travel desk has decided that they would like to find the cheapest way to reach each destination from CmI.



The Travel desk at CmI would like to reduce the money spent on airtickets.

- The rules permit CmI members to fly only by the state airline which uses a rather obscure pricing policy.
- It is not always the case that the cost of a ticket is proportional to the distance.
- The travel desk has decided that they would like to find the cheapest way to reach each destination from CmI.



How to find the length of the shortest paths from vertex **1** to all the other vertices?

Solution: An arsonist's delight.

How to find the length of the shortest paths from vertex **1** to all the other vertices?

Solution: An arsonist's delight.

- Imagine that each vertex is a tank of fuel.

How to find the length of the shortest paths from vertex **1** to all the other vertices?

Solution: An arsonist's delight.

- Imagine that each vertex is a tank of fuel.
- Let each edge be a fuel pipeline connecting two tanks. The length of the pipeline is the weight associated.

How to find the length of the shortest paths from vertex **1** to all the other vertices?

Solution: An arsonist's delight.

- Imagine that each vertex is a tank of fuel.
- Let each edge be a fuel pipeline connecting two tanks. The length of the pipeline is the weight associated.
- Set fire to the tank at the starting vertex.

How to find the length of the shortest paths from vertex **1** to all the other vertices?

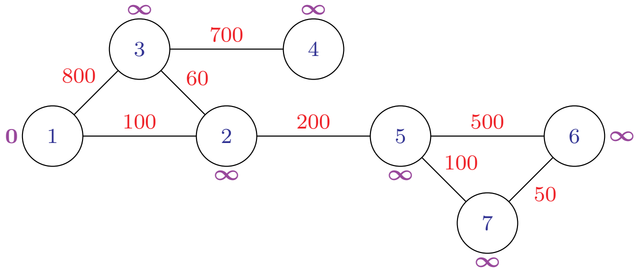
Solution: An arsonist's delight.

- Imagine that each vertex is a tank of fuel.
- Let each edge be a fuel pipeline connecting two tanks. The length of the pipeline is the weight associated.
- Set fire to the tank at the starting vertex.
- Assume that fire travels along the pipe line at the rate of one unit per second.

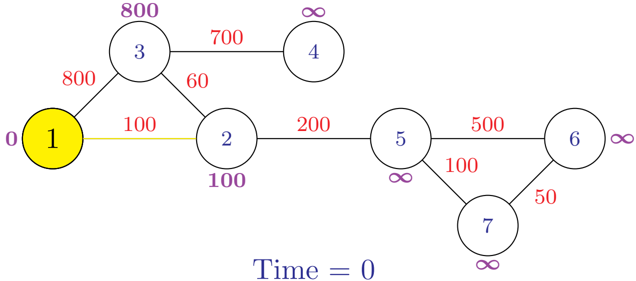
How to find the length of the shortest paths from vertex $\mathbf{1}$ to all the other vertices?

Solution: An arsonist's delight.

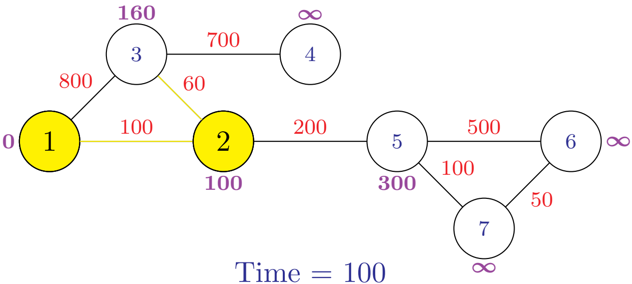
- Imagine that each vertex is a tank of fuel.
- Let each edge be a fuel pipeline connecting two tanks. The length of the pipeline is the weight associated.
- Set fire to the tank at the starting vertex.
- Assume that fire travels along the pipe line at the rate of one unit per second.
- The length of the shortest path to a vertex $\boldsymbol{v}$ is the time at which $\boldsymbol{v}$ starts burning.



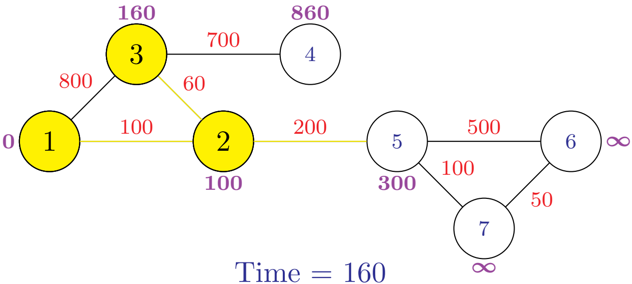
1
2
3
4
5
6
7
0
∞
∞
∞
∞
∞
∞
800
60
700
100
200
500
100
50



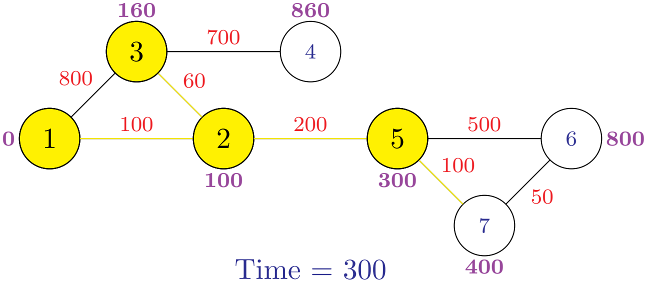
Time = 0



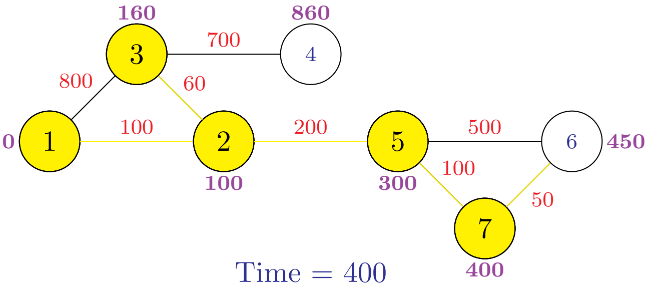
Time = 100



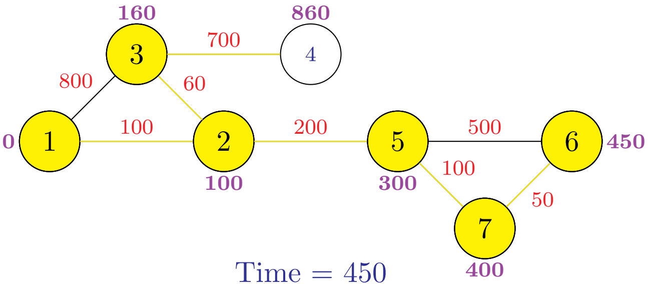
Time = 160



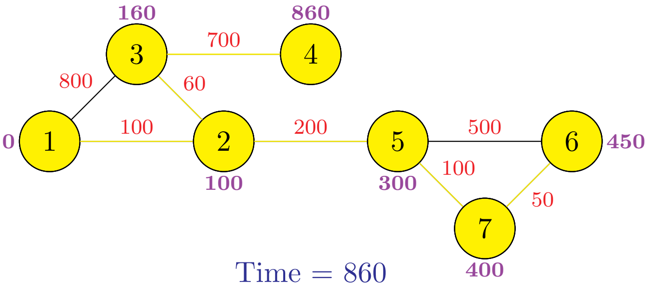
Time = 300



Time = 400



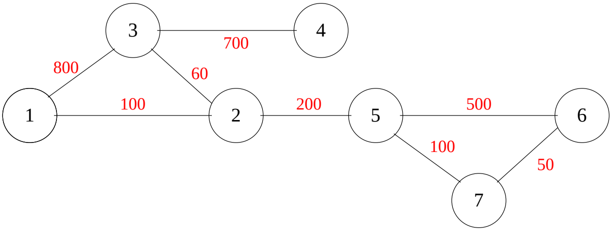
Time = 450



Time = 860

How do we translate this into an algorithm?

- For each vertex maintain information on when it is expected to burn.
- This value depends on when its neighbours start burning and the length of the edges connecting it to its neighbours.
- Initially, vertex 1 is the only one expected to burn.



The Details (Dijkstra's Algorithm)

Use two arrays $\boldsymbol{Burnt}[]$ and $\boldsymbol{ExpBurnTime}[]$.

Use two arrays $\boldsymbol{Burnt}[]$ and $\boldsymbol{ExpBurnTime}[]$.

We set $\boldsymbol{Burnt}[\boldsymbol{i}]$ to $\mathbf{1}$ when the vertex $\boldsymbol{i}$ is burnt.

The Details (Dijkstra's Algorithm)

Use two arrays $\boldsymbol{Burnt}[]$ and $\boldsymbol{ExpBurnTime}[]$.

We set $\boldsymbol{Burnt}[\boldsymbol{i}]$ to $\mathbf{1}$ when the vertex $\boldsymbol{i}$ is burnt.

We use $\boldsymbol{ExpBurnTime}[\boldsymbol{i}]$ to keep track of when we expect vertex $\boldsymbol{i}$ to burn.

Use two arrays $\boldsymbol{Burnt}[]$ and $\boldsymbol{ExpBurnTime}[]$.

We set $\boldsymbol{Burnt}[\boldsymbol{i}]$ to $\mathbf{1}$ when the vertex $\boldsymbol{i}$ is burnt.

We use $\boldsymbol{ExpBurnTime}[\boldsymbol{i}]$ to keep track of when we expect vertex $\boldsymbol{i}$ to burn.

Repeat the following steps till all the vertices are burnt:

The Details (Dijkstra's Algorithm)

Use two arrays $Burnt[]$ and $ExpBurnTime[]$.

We set $Burnt[i]$ to $\mathbf{1}$ when the vertex i is burnt.

We use $ExpBurnTime[i]$ to keep track of when we expect vertex i to burn.

Repeat the following steps till all the vertices are burnt:

1. The unburnt vertex with minimum expected burning time will burn next.

The Details (Dijkstra's Algorithm)

Use two arrays $\boldsymbol{Burnt}[]$ and $\boldsymbol{ExpBurnTime}[]$.

We set $\boldsymbol{Burnt}[\boldsymbol{i}]$ to $\mathbf{1}$ when the vertex $\boldsymbol{i}$ is burnt.

We use $\boldsymbol{ExpBurnTime}[\boldsymbol{i}]$ to keep track of when we expect vertex $\boldsymbol{i}$ to burn.

Repeat the following steps till all the vertices are burnt:

1. The unburnt vertex with minimum expected burning time will burn next.
2. The burning time of this vertex is used to update the expected burning time of its neighbours.

How long does it take?

- In each round we examine all the vertices twice.

How long does it take?

- In each round we examine all the vertices twice.
- In each round one vertex gets burnt.

How long does it take?

- In each round we examine all the vertices twice.
- In each round one vertex gets burnt.

About $\boldsymbol{N^2}$ Steps

Will the Adjacency List representation help?

- In each round we examine all the vertices twice.
- In each round one vertex gets burnt.

About $\boldsymbol{N^2}$ Steps

Will the Adjacency List representation help? NO.

You have to at least look at the entire graph once and there could up $\boldsymbol{N^2}$ edges.

In real life many graphs have very few edges. Often the number of edges is not more than the number of vertices.

In real life many graphs have very few edges. Often the number of edges is not more than the number of vertices.

Can we improve the algorithm for sparse graphs?

In real life many graphs have very few edges. Often the number of edges is not more than the number of vertices.

Can we improve the algorithm for sparse graphs?

Yes, we can use Heaps to implement Dijkstra's algorithm in O(nloge).

Keep the vertices and their expected burn time in a Heap.

In each step

Dijkstra's Algorithm with Heaps

Keep the vertices and their expected burn time in a Heap.

In each step

- Pull out the vertex with minimum expected burn time. Mark it as burnt.
- For each neighbour of this vertex, if the burn time changes then modify the corresponding entry in the heap appropriately.

Dijkstra's Algorithm with Heaps

Keep the vertices and their expected burn time in a Heap.

In each step

- Pull out the vertex with minimum expected burn time. Mark it as burnt.
- For each neighbour of this vertex, if the burn time changes then modify the corresponding entry in the heap appropriately.

Repeat till the heap is empty.

Keep the vertices and their expected burn time in a Heap.

In each step

- Pull out the vertex with minimum expected burn time. Mark it as burnt.
- For each neighbour of this vertex, if the burn time changes then modify the corresponding entry in the heap appropriately.

Repeat till the heap is empty.

Note, each edge may modify the heap only once and hence the complexity.