

NPTEL MOOC, JAN-FEB 2015
Week 4, Module 1

DESIGN AND ANALYSIS OF ALGORITHMS

Shortest paths in weighted graphs

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
<http://www.cmi.ac.in/~madhavan>

Recall that ...

- * BFS and DFS are two systematic ways to explore a graph
- * Both take time linear in the size of the graph with adjacency lists
- * Recover paths by keeping parent information
- * BFS can compute shortest paths, in terms of number of edges
- * DFS numbering can reveal many interesting features

Adding edge weights

- * Label each edge with a number—**cost**
- * Ticket price on a flight sector
- * Tolls on highway segment
- * Distance travelled between two stations
- * Typical time between two locations during peak hour traffic

Shortest paths

- * **Weighted graph**
 - * $G=(V,E)$ together with
 - * **Weight function**, $w : E \rightarrow \text{Reals}$
- * Let $e_1=(v_0,v_1)$, $e_2 = (v_1,v_2)$, ..., $e_n = (v_{n-1},v_n)$ be a path from v_0 to v_n
- * Cost of the path is $w(e_1) + w(e_2) + \dots + w(e_n)$
- * **Shortest path** from v_0 to v_n : minimum cost

Shortest paths ...

- * BFS finds path with fewest number of edges
- * In a weighted graph, need not be the shortest path



Shortest path problems

- * **Single source**

- * Find shortest paths from some fixed vertex, say 1, to every other vertex
- * Transport finished product from factory (single source) to all retail outlets
- * Courier company delivers items from distribution centre (single source) to addressees

Shortest path problems

- * **All pairs**

- * Find shortest paths between every pair of vertices i and j
- * Railway routes, shortest way to travel between any pair of cities

NPTTEL

This lecture...

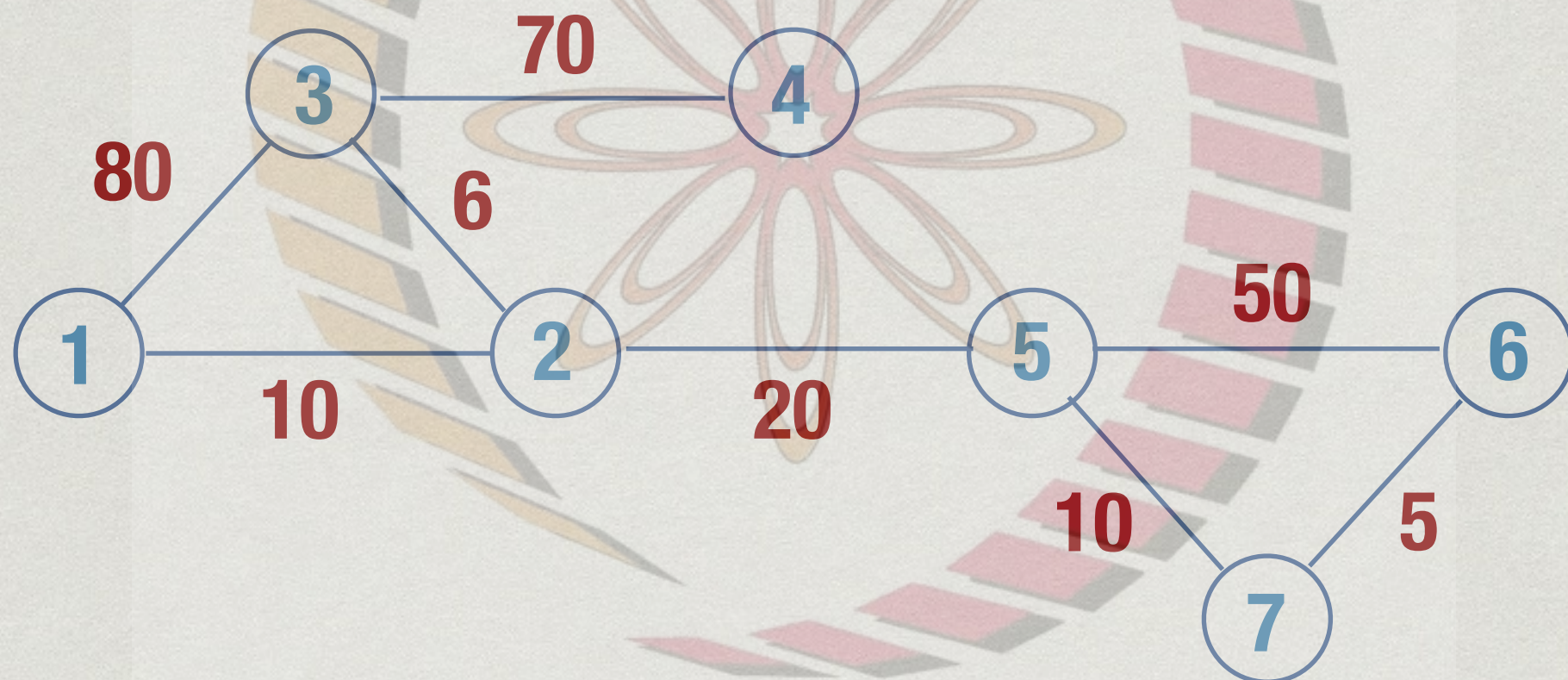
- * Single source shortest paths
- * For instance, shortest paths from 1 to 2,3,...,7



Single source shortest paths

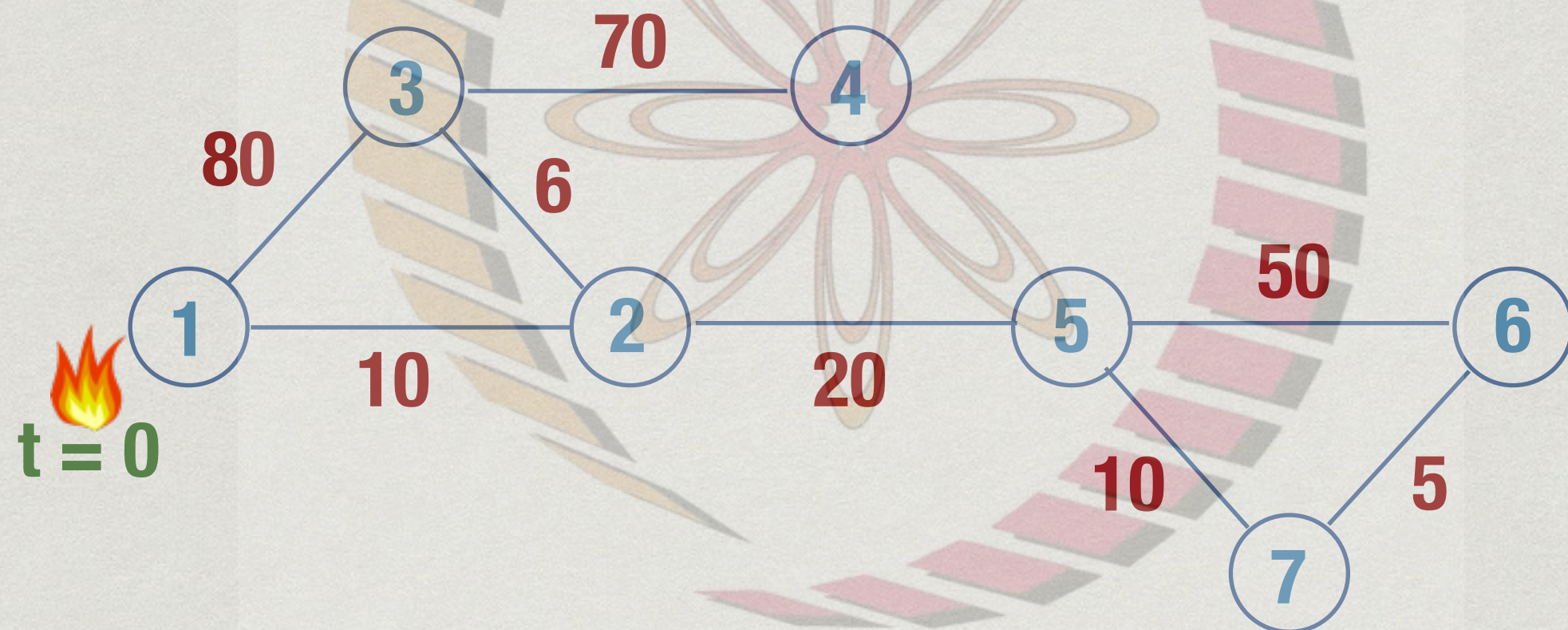
- * Imagine vertices are oil depots, edges are pipelines
- * Set fire to oil depot at vertex 1
 - * Fire travels at uniform speed along each pipeline
- * First oil depot to catch fire after 1 is nearest vertex
- * Next oil depot is second nearest vertex
- * ...

Single source shortest paths



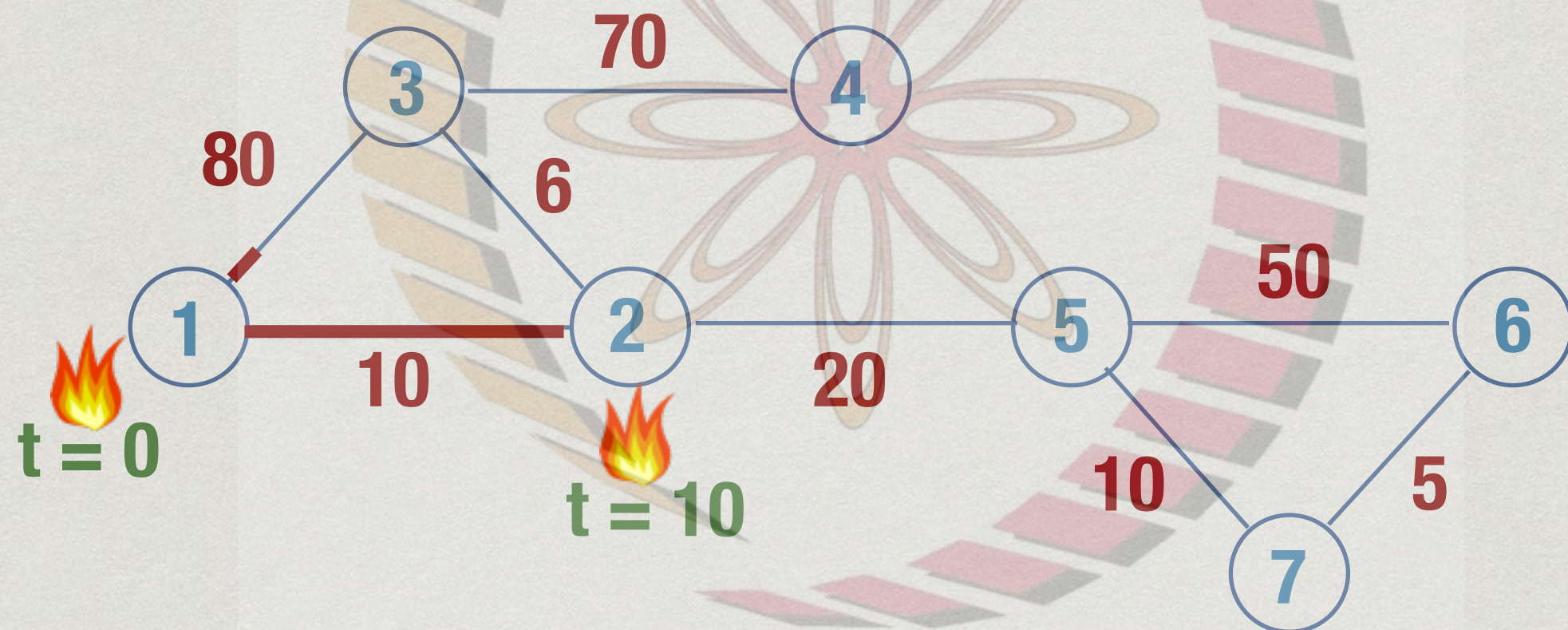
NPTTEL

Single source shortest paths

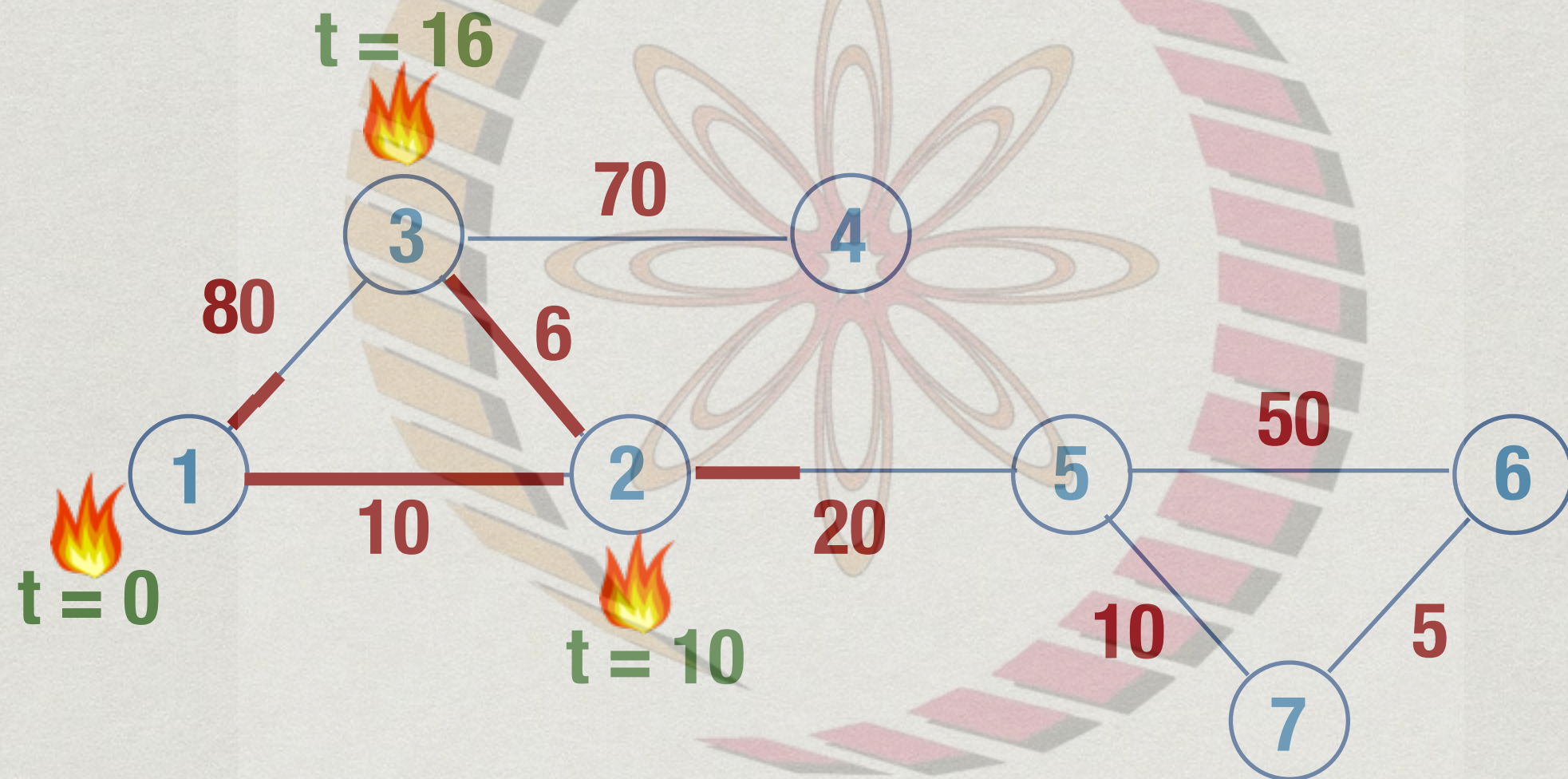


NPTTEL

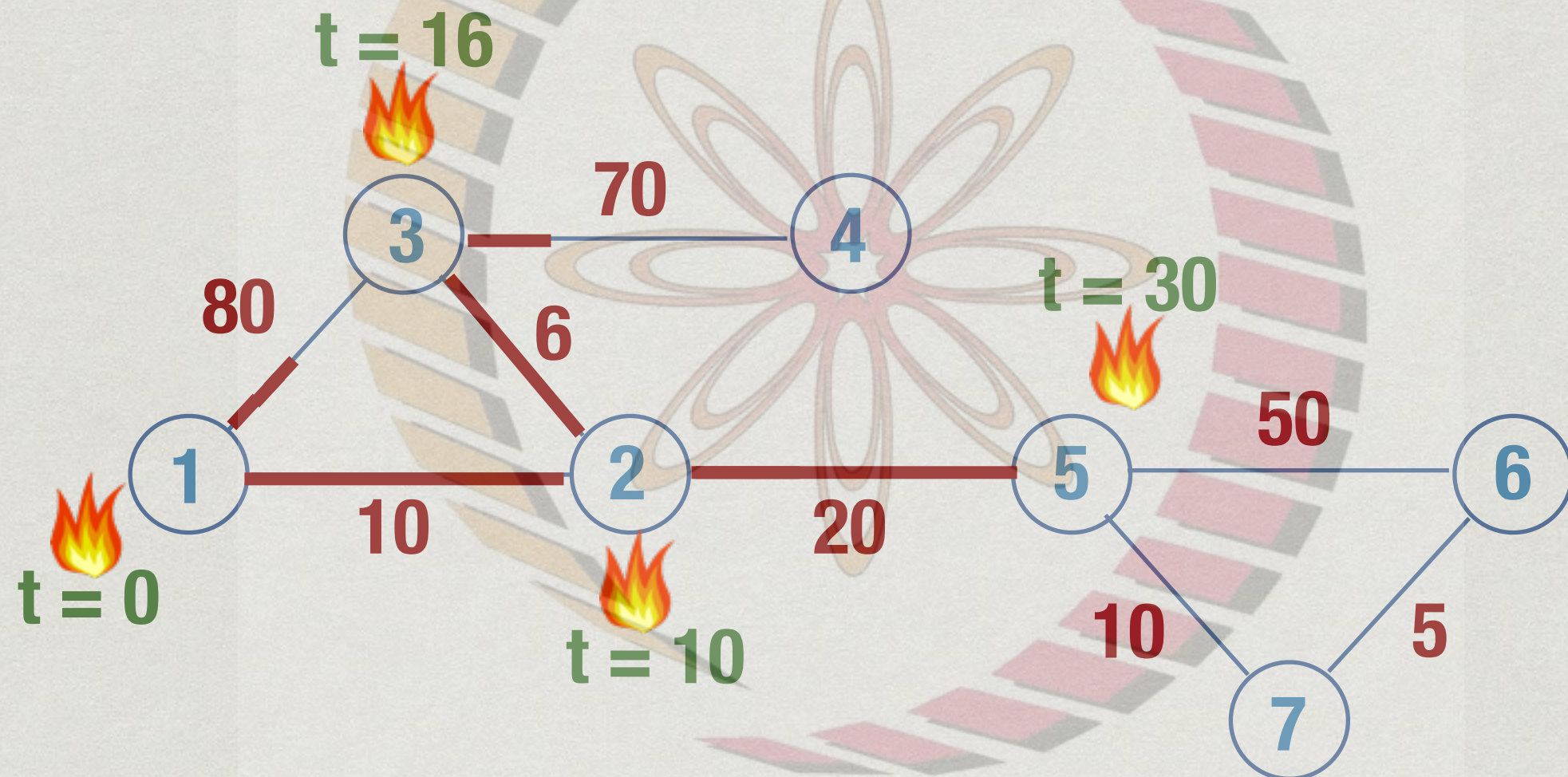
Single source shortest paths



Single source shortest paths

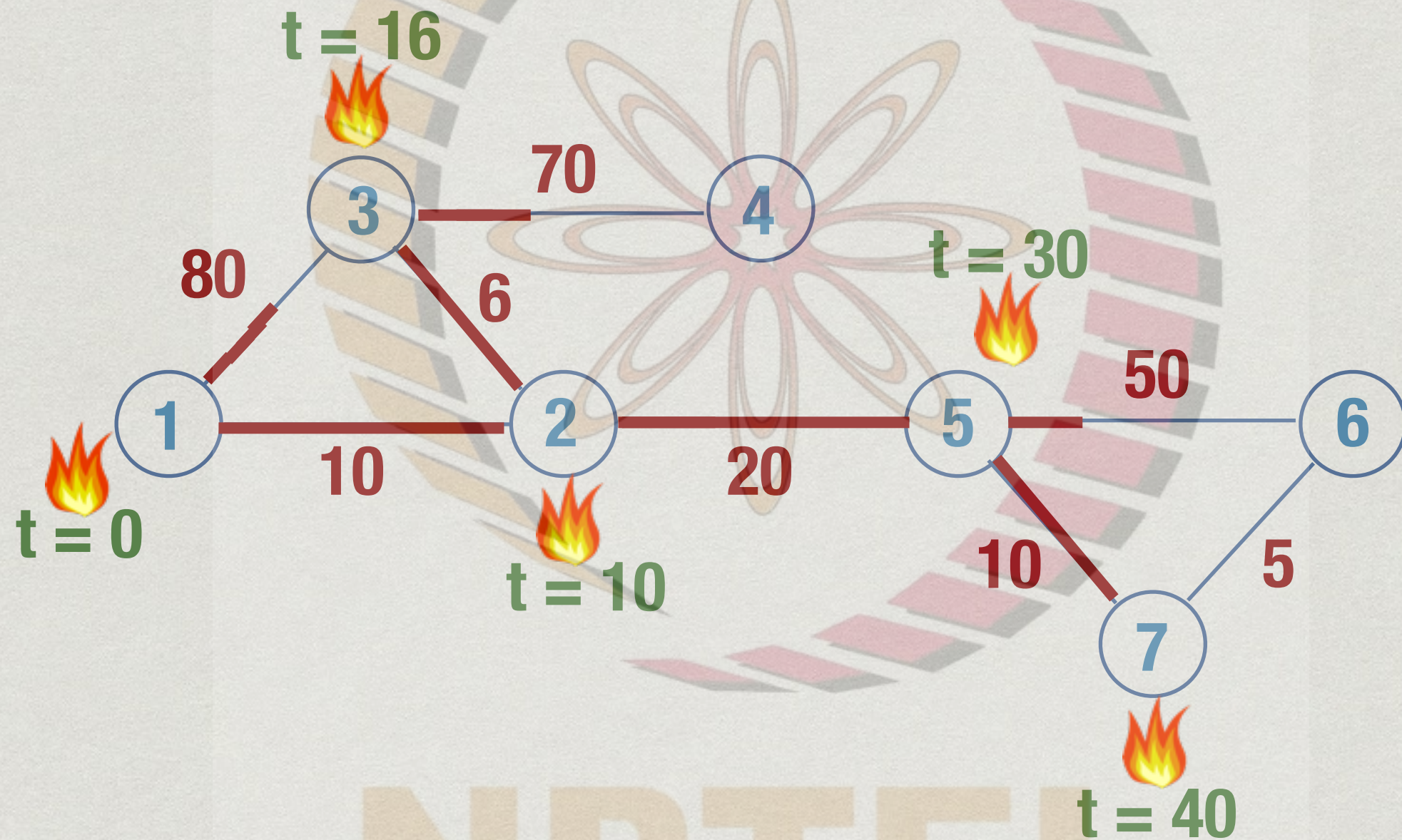


Single source shortest paths



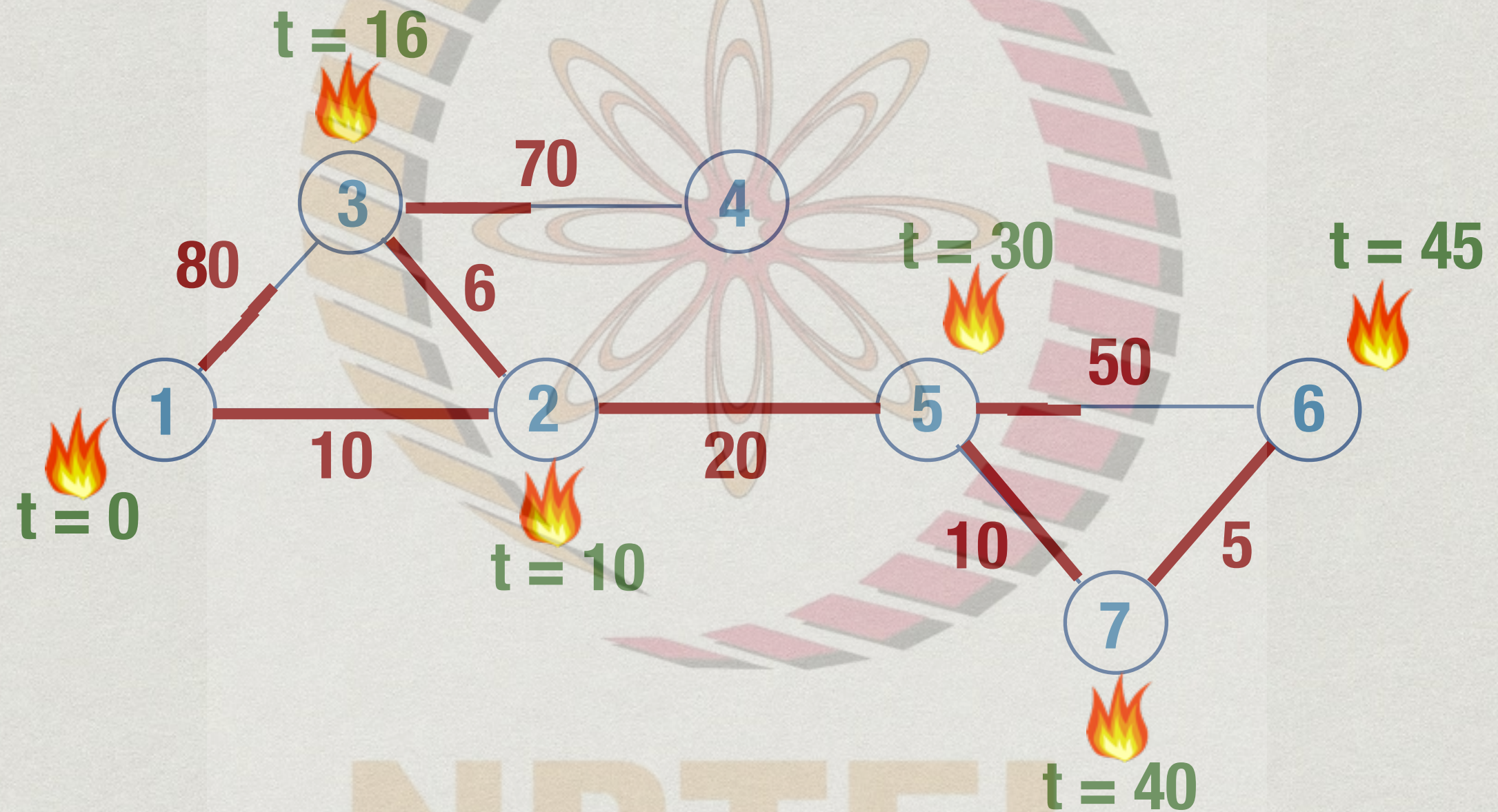
NPTTEL

Single source shortest paths

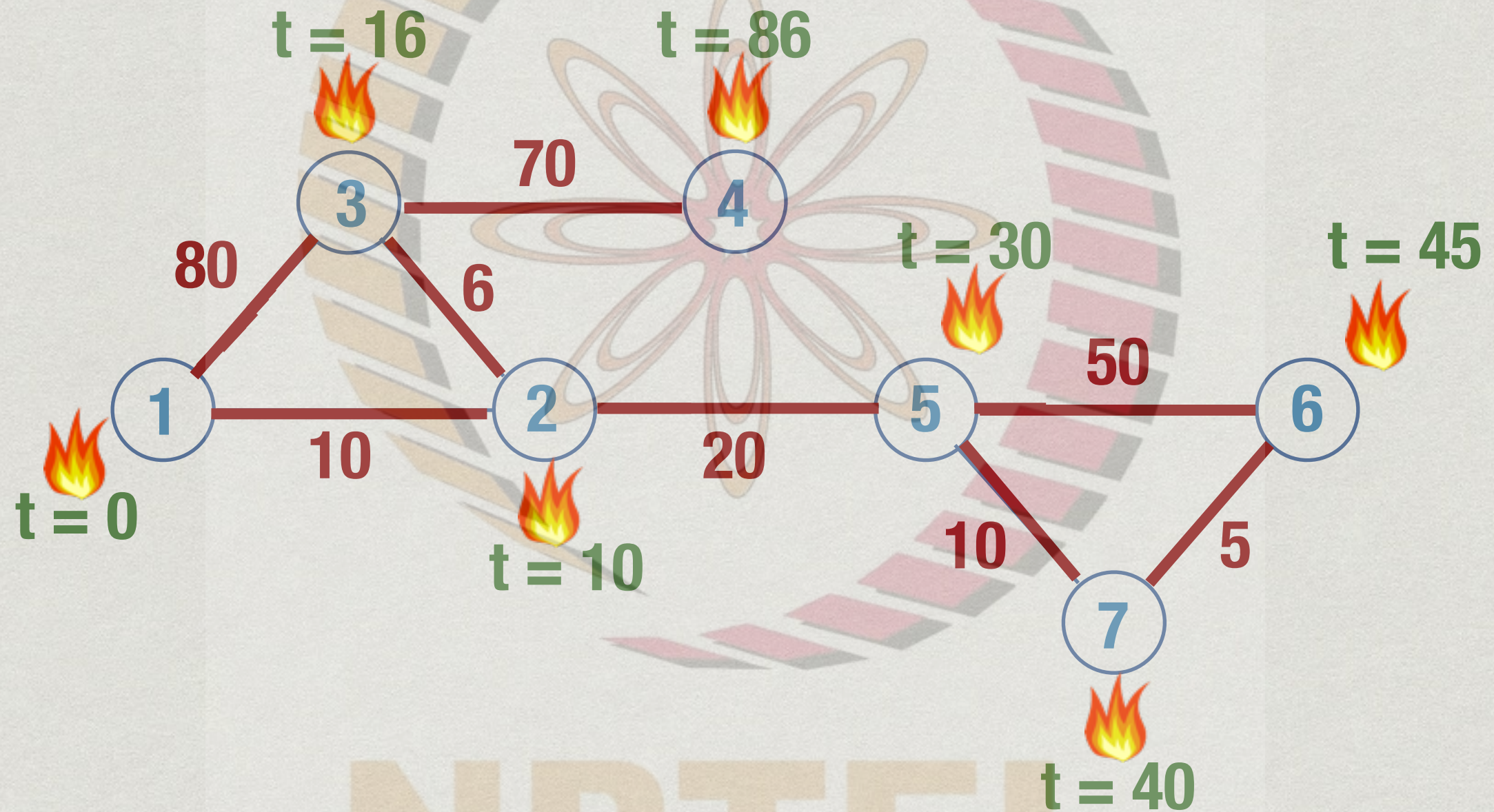


NPTTEL

Single source shortest paths



Single source shortest paths



NPTTEL

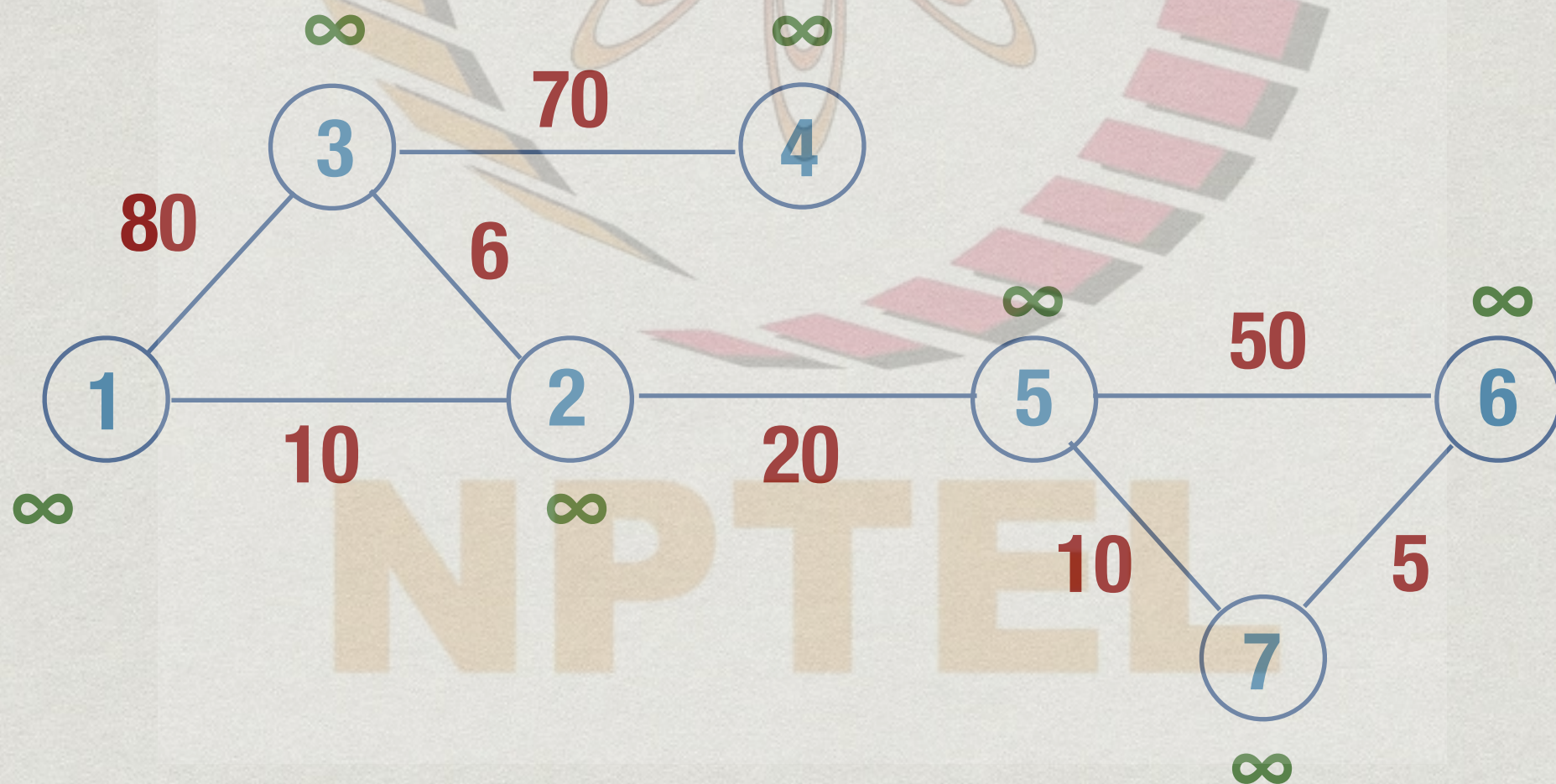
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



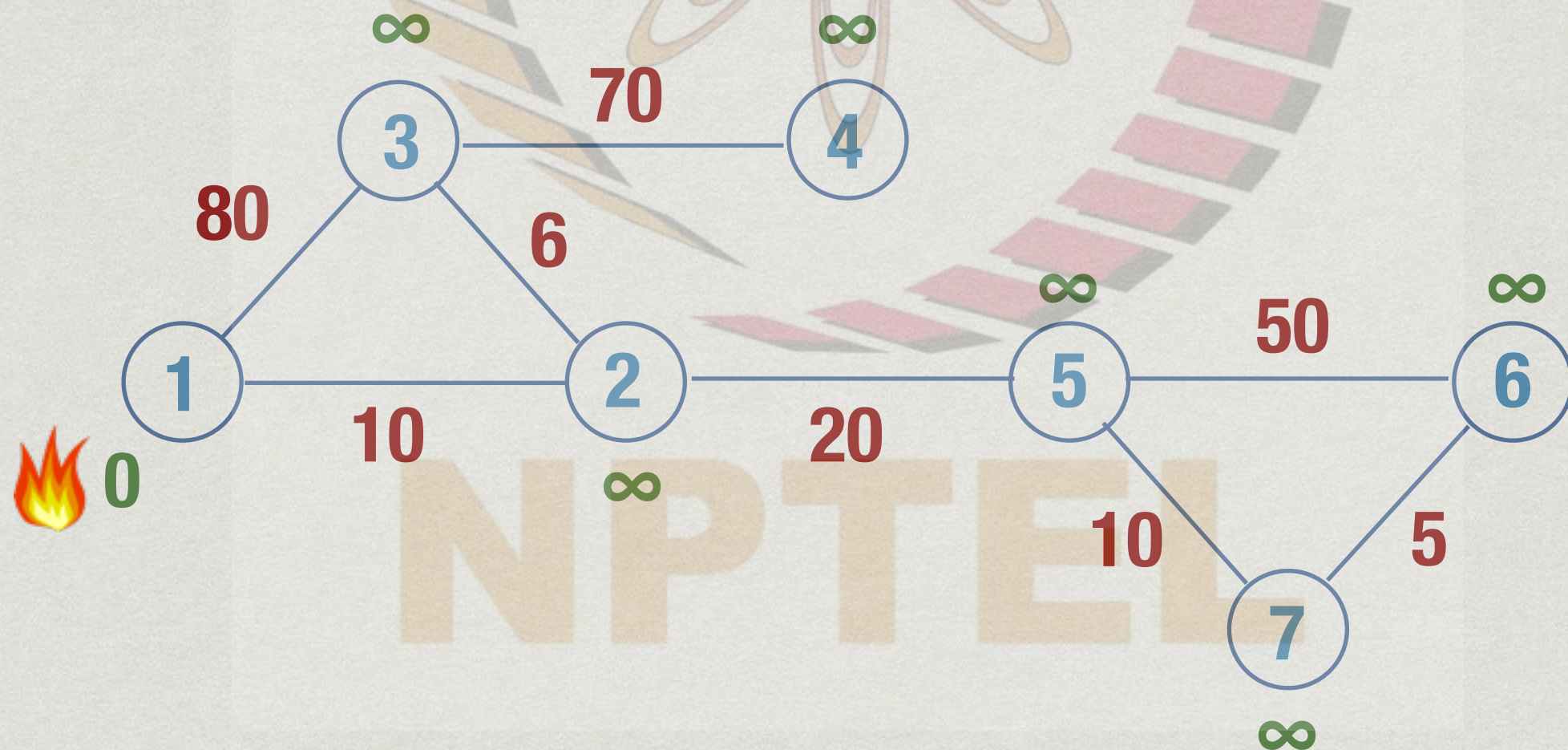
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



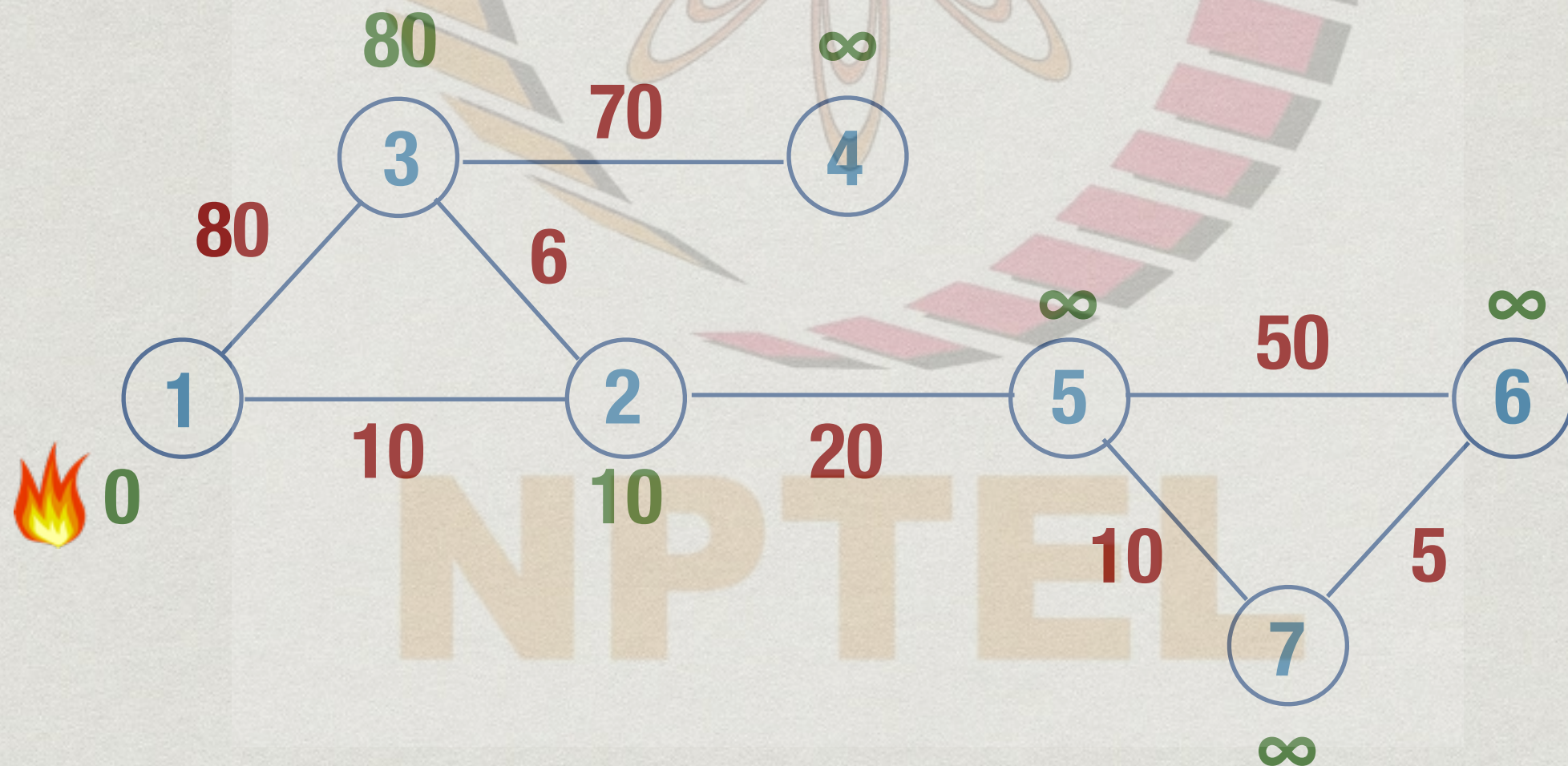
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



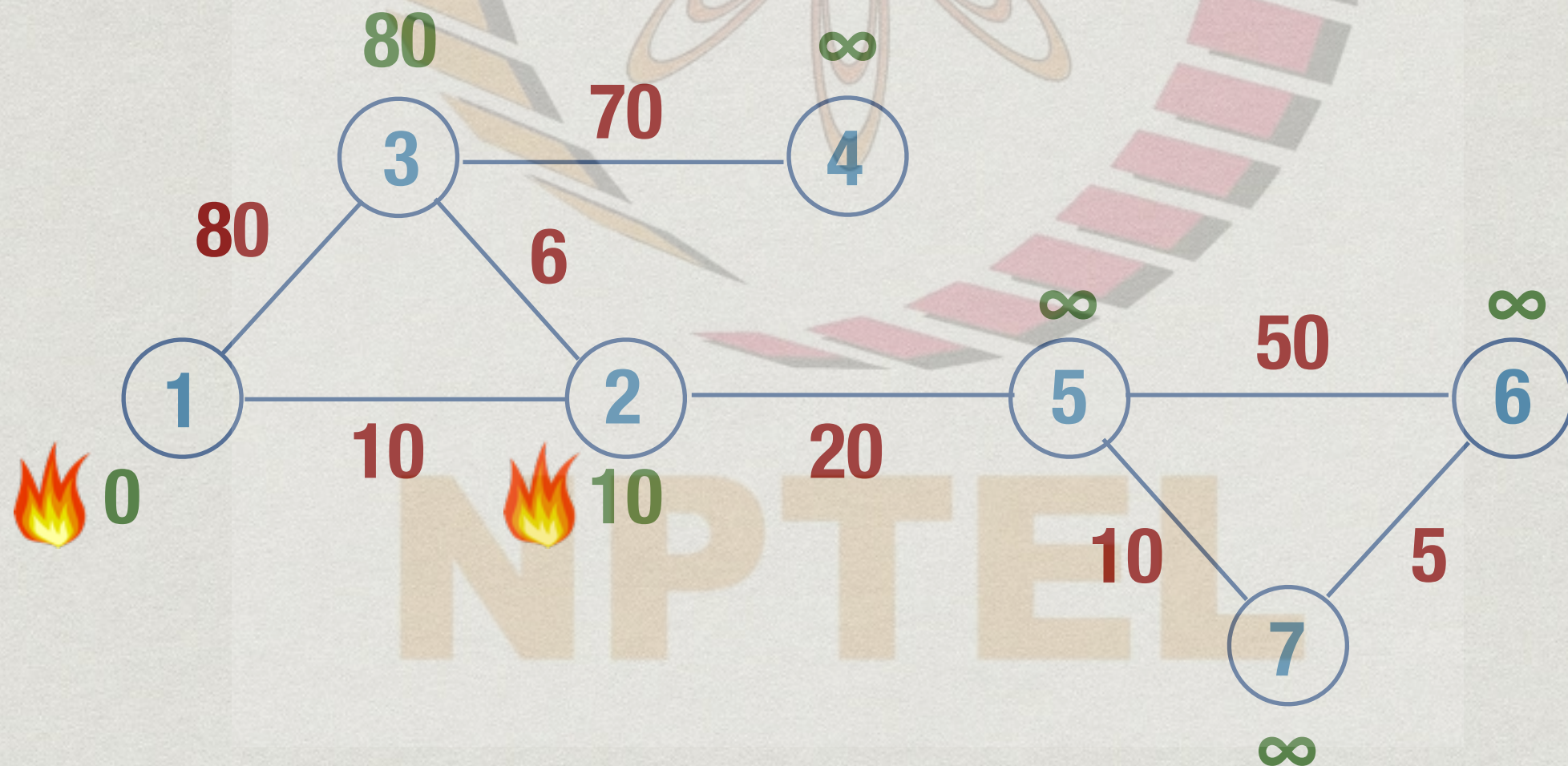
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



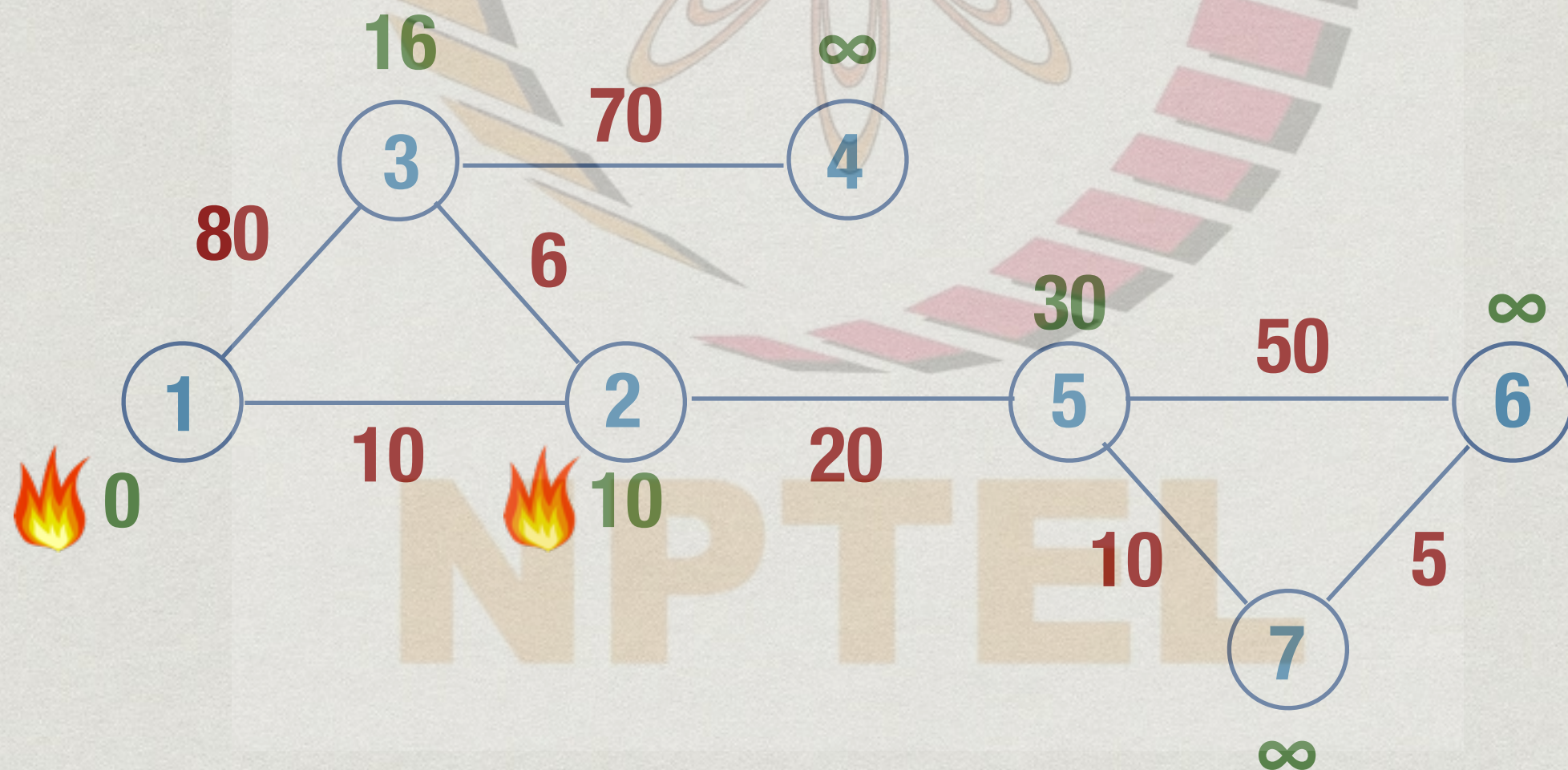
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



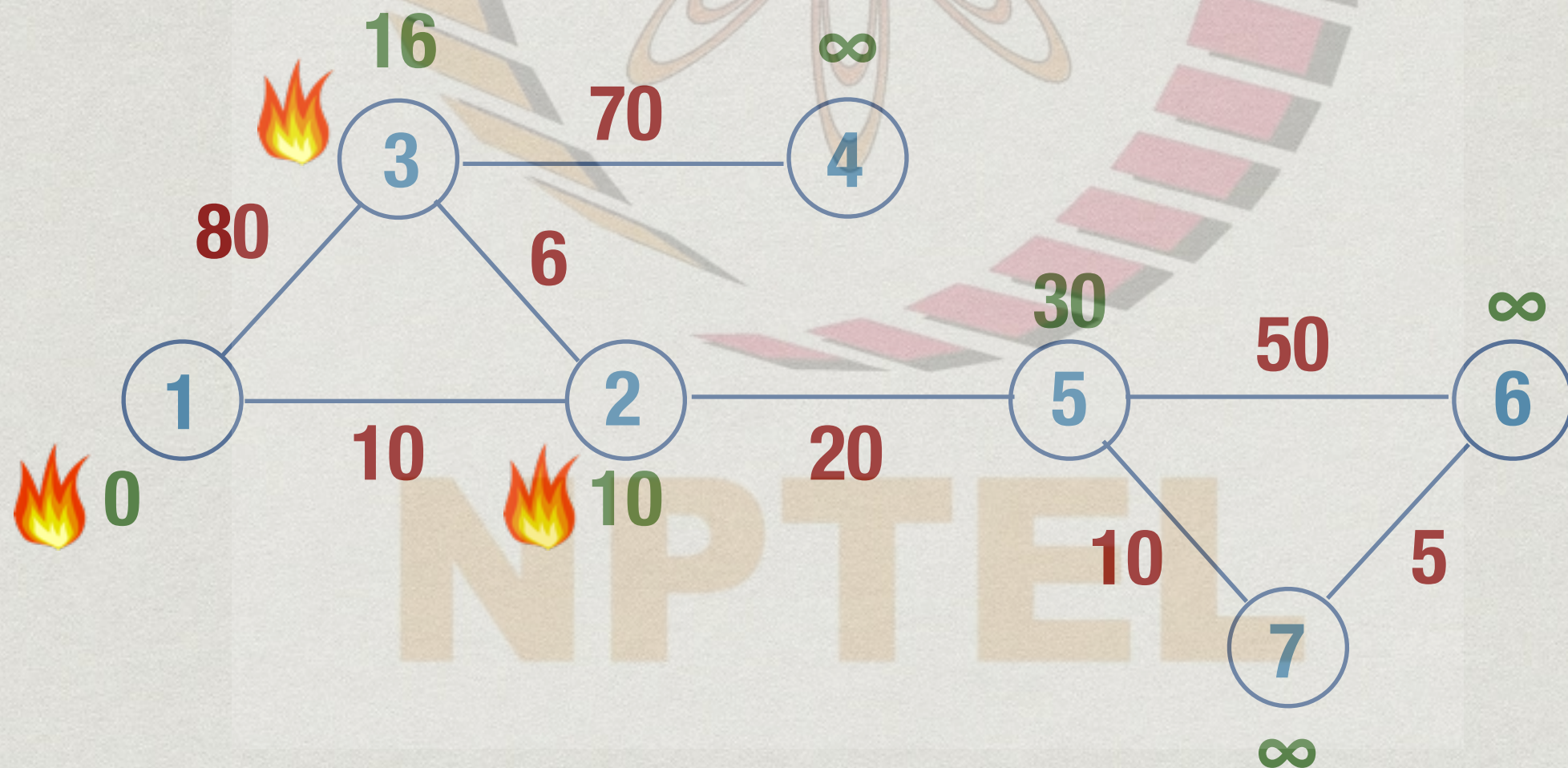
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



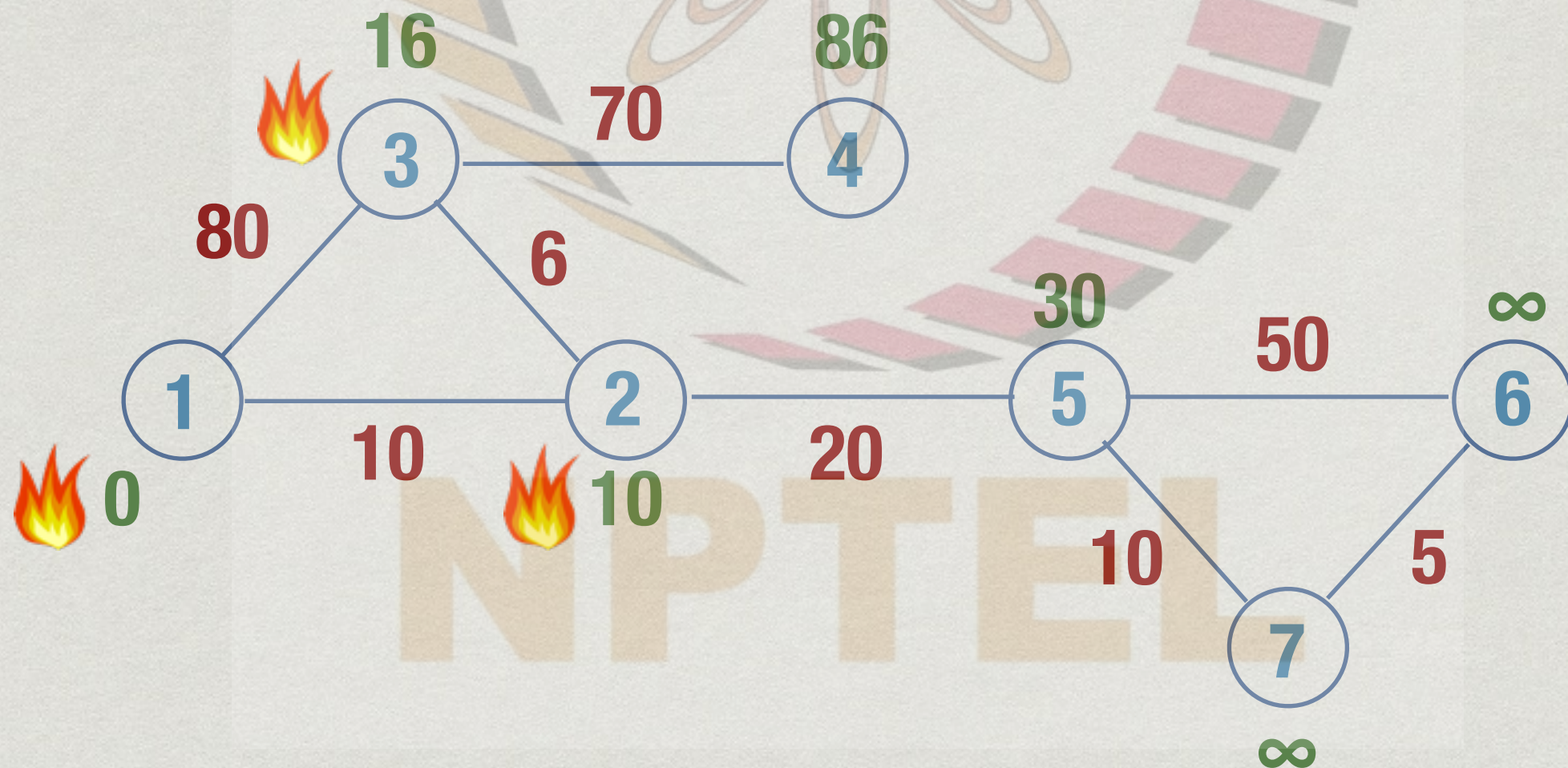
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



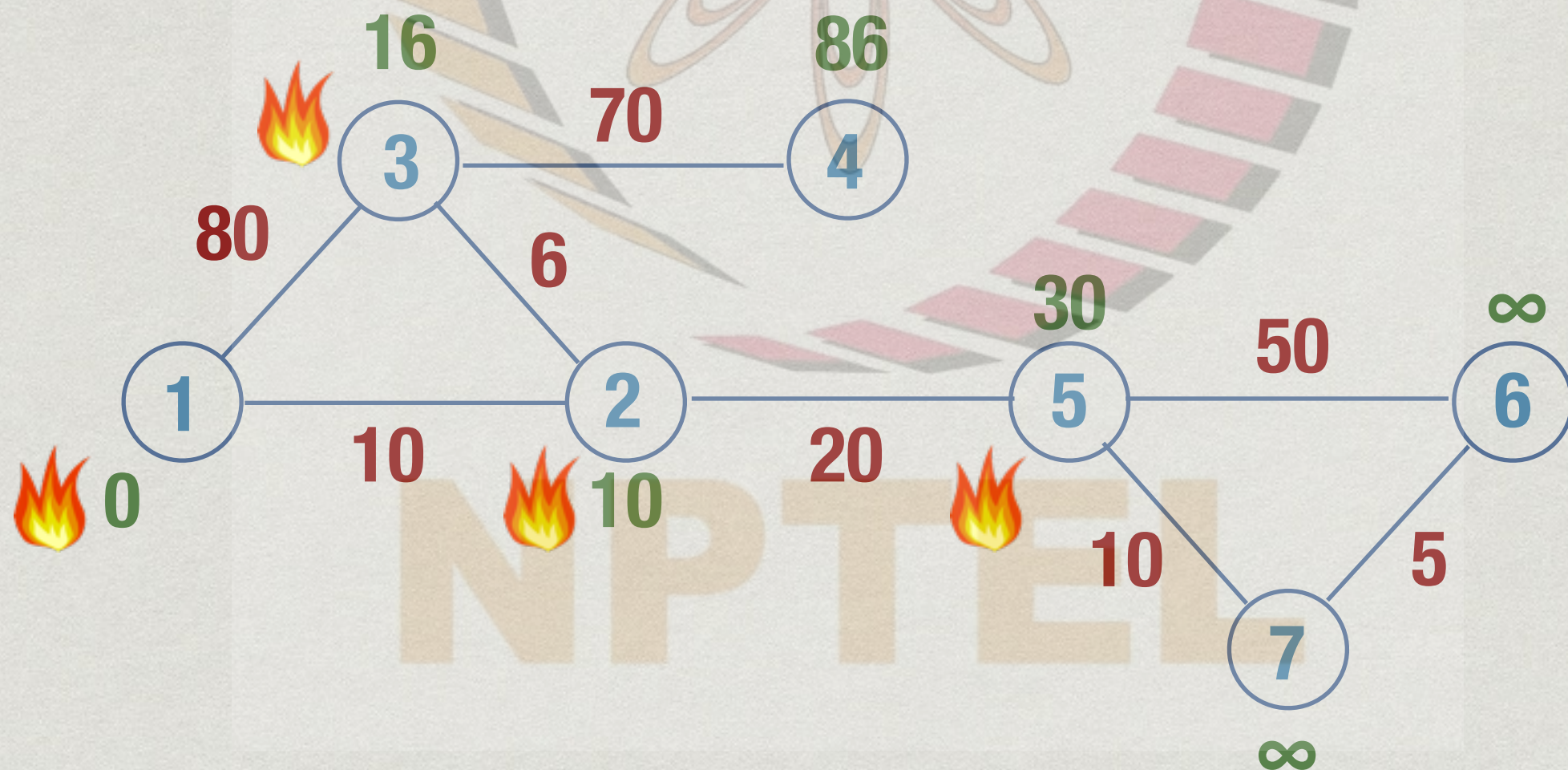
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



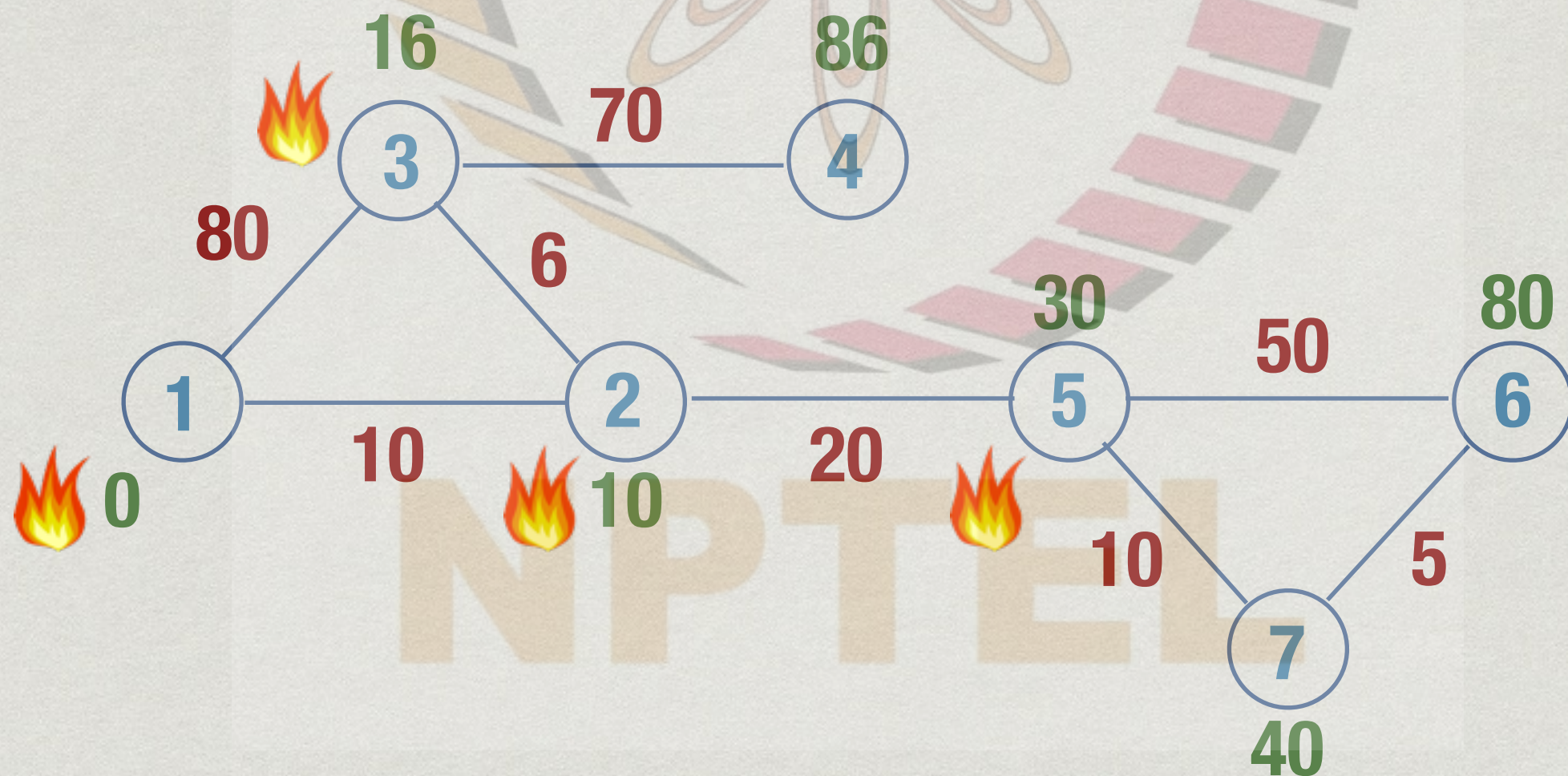
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



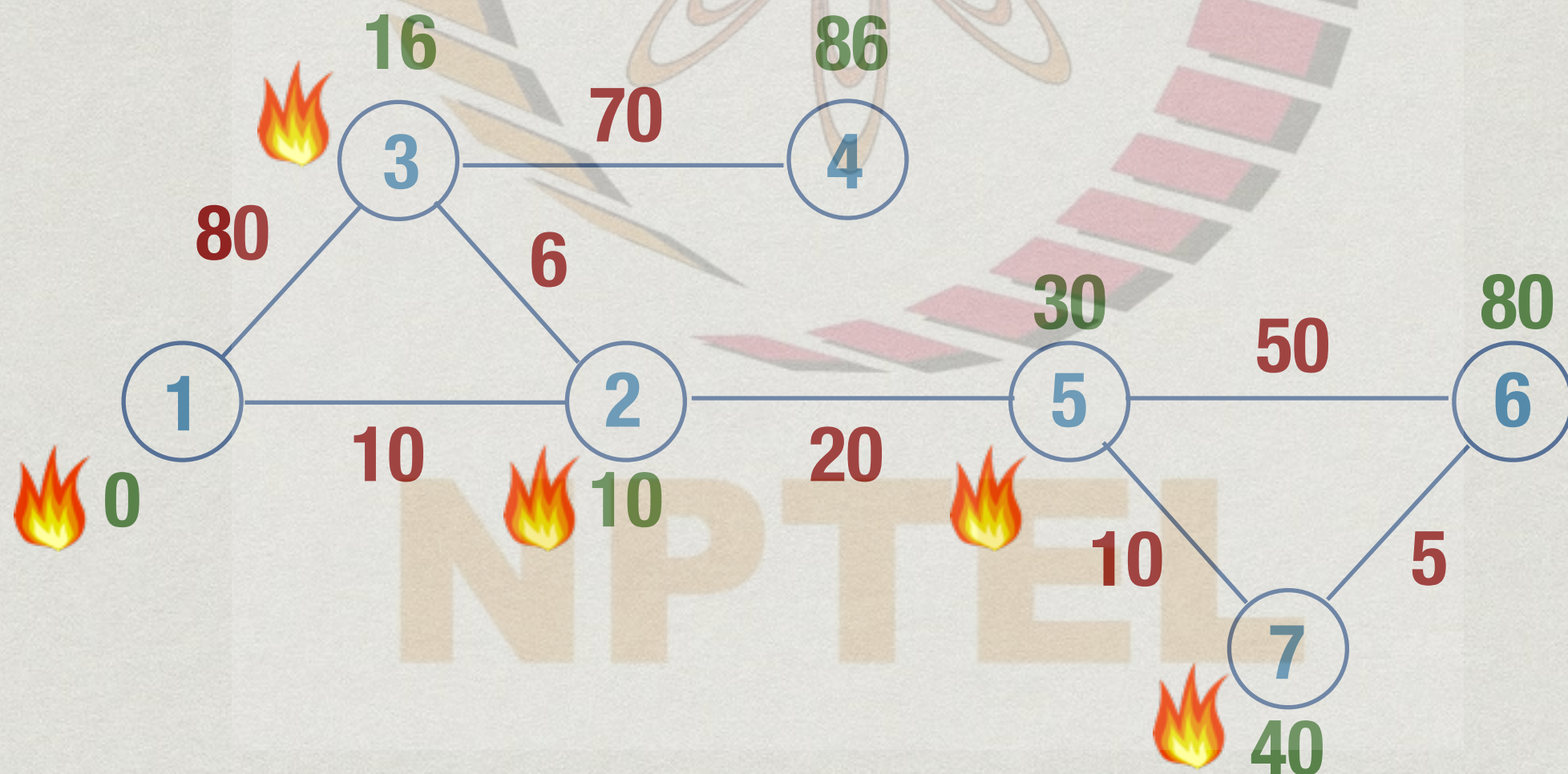
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



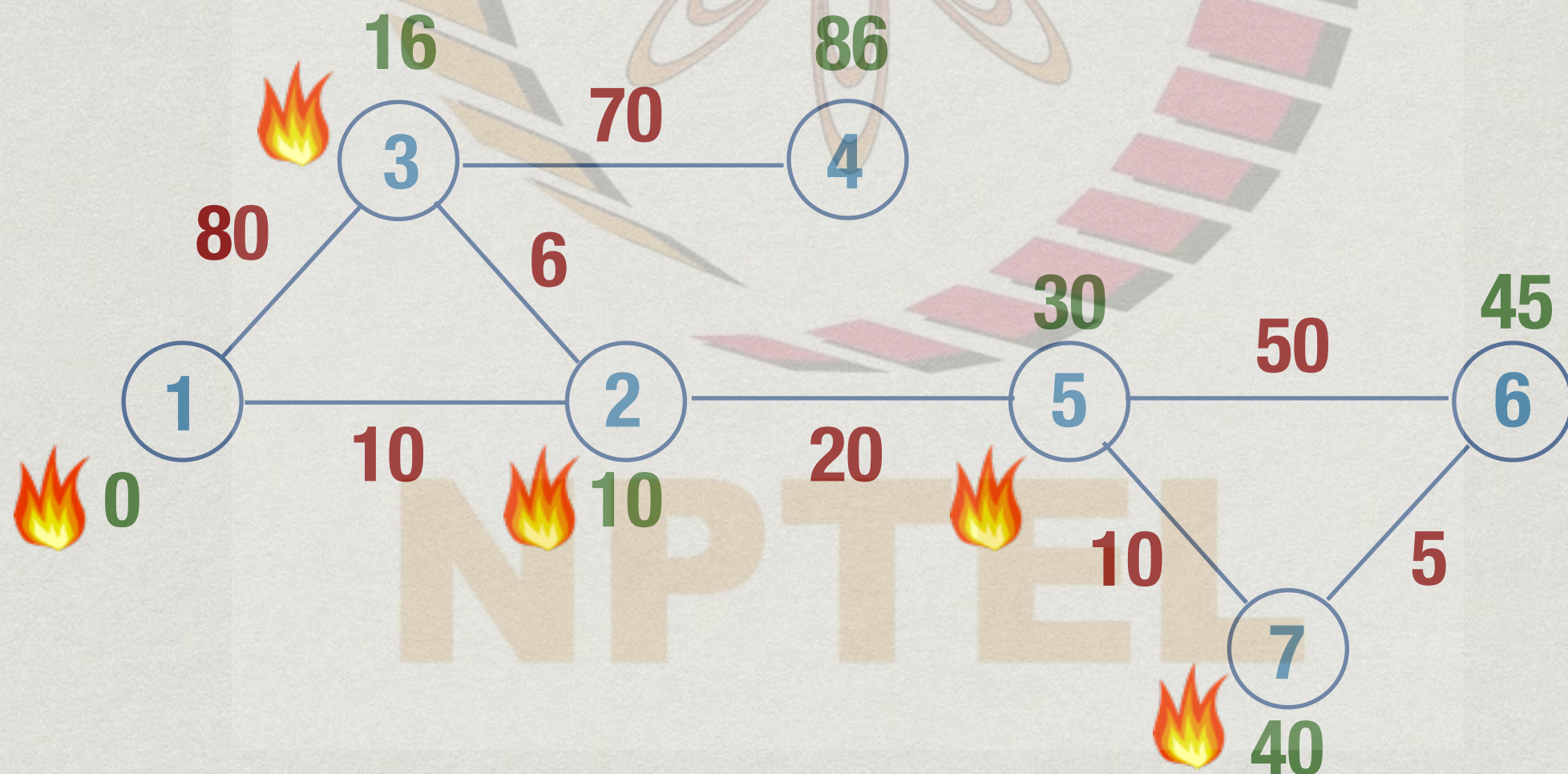
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



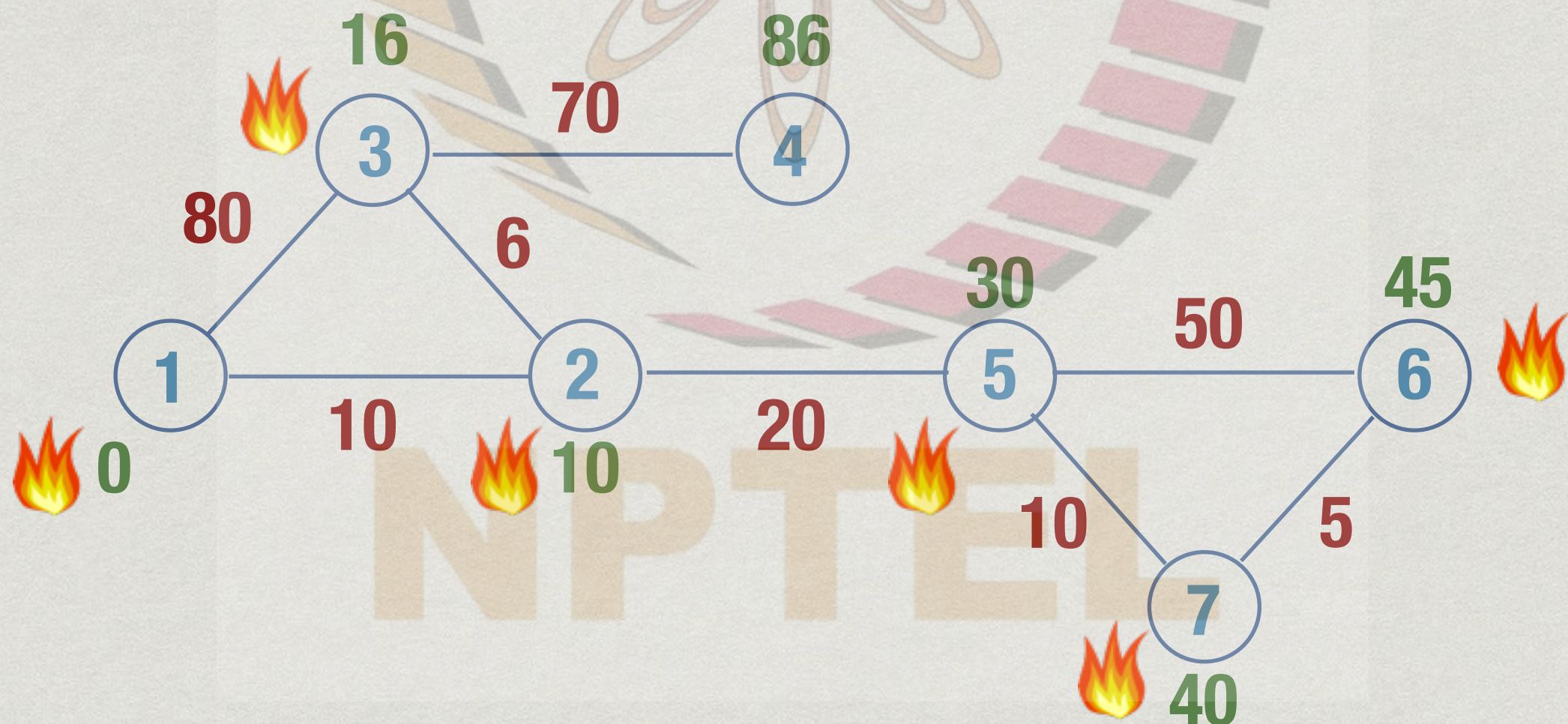
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



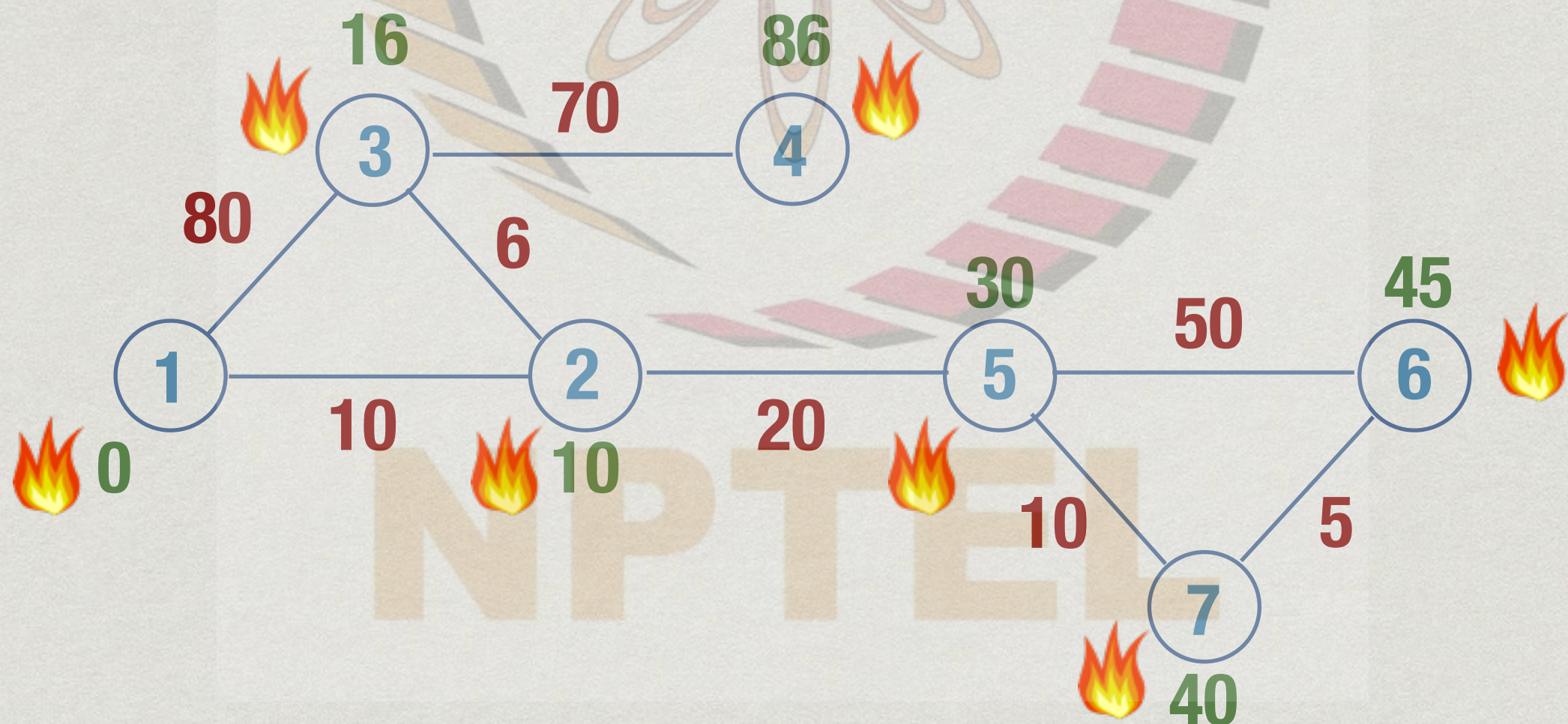
Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



Single source shortest paths

- * Compute expected time to burn of each vertex
- * Update this each time a new vertex burns



Algorithmically

- * Maintain two arrays
 - * `BurntVertices[]`, initially `False` for all i
 - * `ExpectedBurnTime[]`, initially ∞ for all i
 - * For ∞ , use sum of all edge weights + 1
- * Set `ExpectedBurnTime[1] = 0`
- * Repeat, until all vertices are burnt
 - * Find j with minimum `ExpectedBurnTime`
 - * Set `BurntVertices[j] = True`
 - * Recompute `ExpectedBurnTime[k]` for each neighbour k of j

Dijkstra's algorithm

```
function ShortestPaths(s){ // assume source is s
    for i = 1 to n
        BV[i] = False; EBT[i] = infinity

    EBT[s] = 0

    for i = 1 to n
        Choose u such that BV[u] == False
                        and EBT[u] is minimum

        BV[u] = True
        for each edge (u,v) with BV[v] == False
            if EBT[v] > EBT[u] + weight(u,v)
                EBT[v] = EBT[u] + weight(u,v)
```


Dijkstra's algorithm

```
function ShortestPaths(s){ // assume source is s
    for i = 1 to n
        Visited[i] = False; Distance[i] = infinity

    Distance[s] = 0

    for i = 1 to n
        Choose u such that Visited[u] == False
                           and Distance[u] is minimum
        Visited[u] = True
        for each edge (u,v) with Visited[v] == False
            if Distance[v] > Distance[u] + weight(u,v)
                Distance[v] = Distance[u] + weight(u,v)
```