NPTEL MOOC,JAN-FEB 2015
Week 4, Module 2

DESIGN AND ANALYSIS OF ALGORITHMS

Dijkstra’s algorithm: analysis

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

Dijkstra's algorithm

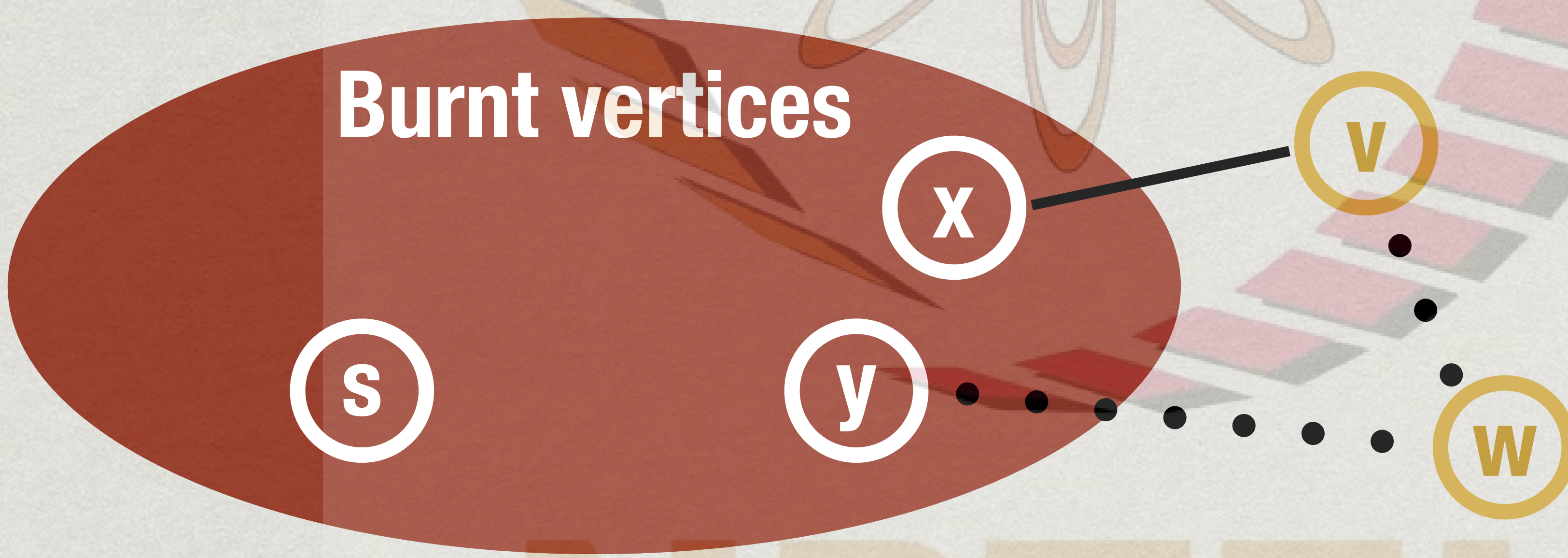
- Maintain two arrays
  - `Visited[ ]`, initially `False` for all i
  - `Distance[ ]`, initially ∞ for all i
    - For ∞, use sum of all edge weights + 1
- Set `Distance[1] = 0`
- Repeat, until all vertices are burnt
  - Find `j` with minimum `Distance`
  - Set `Visited[j] = True`
  - Recompute `Distance[k]`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

Greedy algorithms

- Algorithm makes a sequence of choices
- Next choice is based on “current best value”
  - Never go back and change a choice
- Dijkstra’s algorithm is greedy
  - Select vertex with minimum expected burn time
- Need to prove that greedy strategy is optimal
- Most times, greedy approach fails
  - Current best choice may not be globally optimal

Correctness

- Each new shortest path we discover extends an earlier one
- By induction, assume we have identified shortest paths to all vertices already burnt



- Next vertex to burn is v, via x
- Cannot later find a shorter path from y to w to 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)
```

Complexity

- Outer loop runs n times
  - In each iteration, we burn one vertex
  - O(n) scan to find minimum burn time vertex
- Each time we burn a vertex v, we have to scan all its neighbours to update burn times
  - O(n) scan of adjacency matrix to find all neighbours
- Overall $O(n^2)$

Complexity

- Does adjacency list help?
  - Scan neighbours to update burn times
  - O(m) across all iterations
- However, identifying minimum burn time vertex still takes O(n) in each iteration
- Still O(n^2)

Complexity

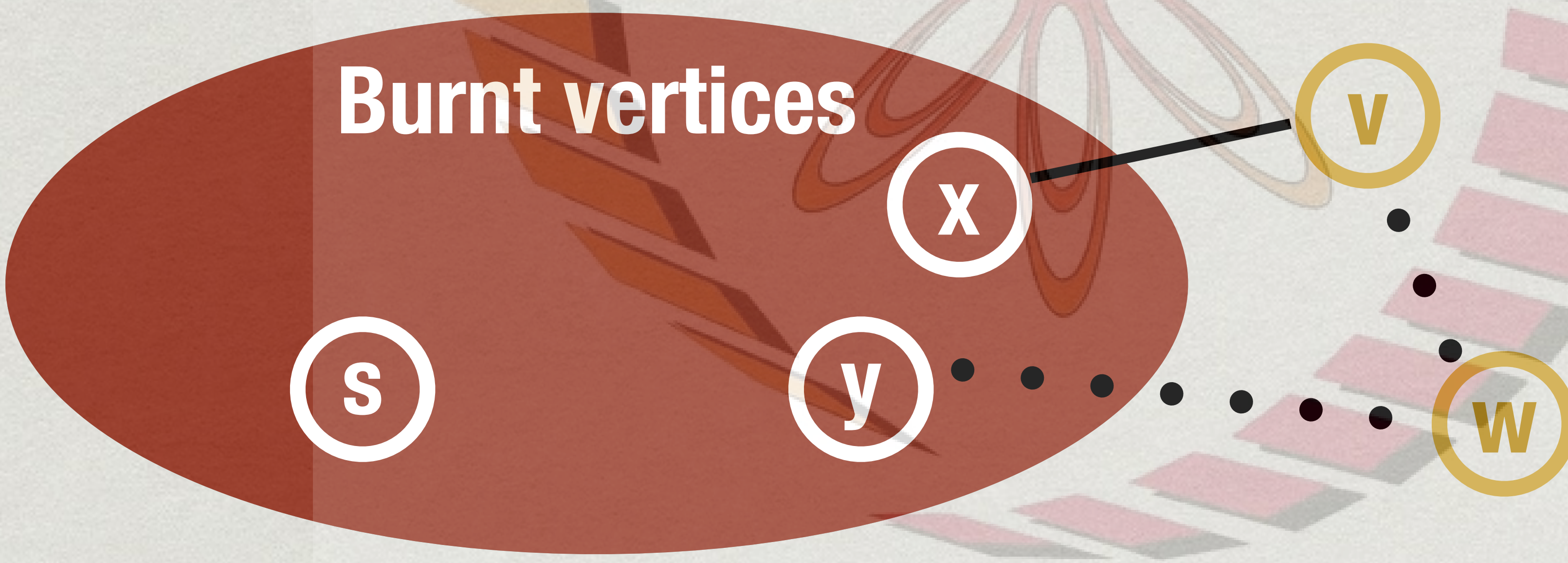
- Can maintain `ExpectedBurnTime` in a more sophisticated data structure
  - Different types of trees (heaps, red-black trees) allow both of the following in O(log n) time
    - find and delete minimum
    - insert or update a value

Complexity

- With such a tree
  - Finding minimum burn time vertex takes O(log n)
  - With adjacency list, updating burn times take O(log n) each, total O(m) edges
- Overall O(n log n + m log n) = O((n+m) log n)

Limitations

- What if edge weights can be negative?
- Our correctness argument is no longer valid



- Next vertex to burn is v, via x
- Might find a shorter path later with negative weights from y to w to v

Why negative weights?

- Weights represent money
  - Taxi driver earns money from airport to city, travels empty to next pick-up point
  - Some segments earn money, some lose money
- Chemistry
  - Nodes are compounds, edges are reactions
  - Weights are energy absorbed/released by reaction

Handling negative edges

- **Negative cycle**: loop with a negative total weight
  - Problem is not well defined with negative cycles
  - Repeatedly traversing cycle pushes down cost without a bound
- With negative edges, but no negative cycles, other algorithms exist (will see later)
  - Bellman-Ford
  - Floyd-Warshall all pairs shortest path