Experiment-5
From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm

Greedy Technique

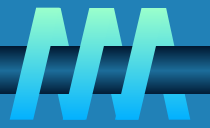
Constructs a solution to an *optimization problem* piece by piece through a sequence of choices that are:

- ***feasible, i.e. satisfying the constraints***
- ***locally optimal (with respect to some neighborhood definition)***
- ***greedy (in terms of some measure), and irrevocable***

Defined by an objective function and a set of constraints

For some problems, it yields a globally optimal solution for every instance. For most, does not but can be useful for fast approximations. We are mostly interested in the former case in this class.

Applications of the Greedy Strategy



- **Optimal solutions:**
  - **change making for "normal" coin denominations**
  - **minimum spanning tree (MST)**
  - **single-source shortest paths**
  - **simple scheduling problems**
  - **Huffman codes**

- **Approximations/heuristics:**
  - **traveling salesman problem (TSP)**
  - **knapsack problem**
  - **other combinatorial optimization problems**

Shortest paths – Dijkstra's algorithm

Single Source Shortest Paths Problem: Given a weighted connected(directed) graph G, find shortest paths from source vertex *s* to each of the other vertices

Dijkstra's algorithm: With a different way of computing numerical labels: Among vertices not already in the tree, it finds vertex *u* with the smallest sum

$$d_v + w(v,u)$$

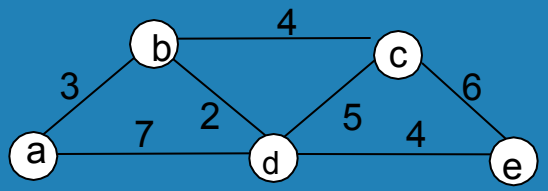
where

v is a vertex for which shortest path has been already found on preceding iterations (such vertices form a tree rooted at *s*)

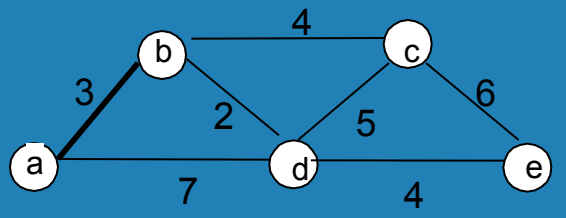
d_v is the length of the shortest path from source *s* to *v*

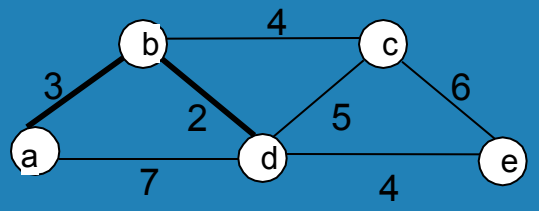
$w(v,u)$ is the length (weight) of edge from *v* to *u*

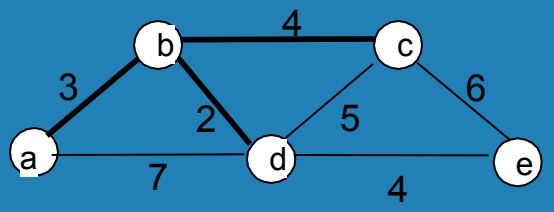
Example

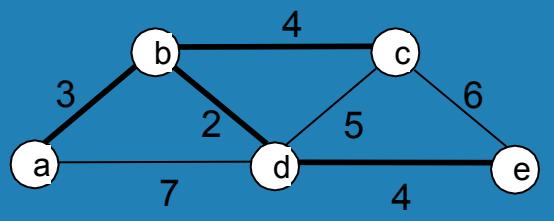


Tree vertices	Remaining vertices	
a(-,0)	b(a,3)	c(-,∞) d(a,7) e(-,∞)
b(a,3)		c(b,3+4) d(b,3+2) e(-,∞)
d(b,5)		c(b,7) e(d,5+4)
c(b,7)		e(d,9)
e(d,9)		









Dijkstra's algorithm - Pseudocode

```
Dijkstra(s)
{
    dist[s] ←0                    (distance to source vertex is zero)
    for  all v ∈ V–{s}
         do  dist[v] ←∞           (set all other distances to infinity)
    S←∅                           (S, the set of visited vertices is initially empty)
    Q←V                           (Q, the queue initially contains all vertices)
    while Q ≠∅   do               (while the queue is not empty)
        u ← mindistance(Q,dist)(select the element of Q with the min. distance)
     S←S ∪ {u}                    (add u to list of visited vertices)
    for all v ∈ neighbors[u]  do
                  if   dist[v] > dist[u] + w(u, v) (if new shortest path found)
                  then      d[v] ←d[u] + w(u, v)(set new value of shortest path)

    return dist
}
```