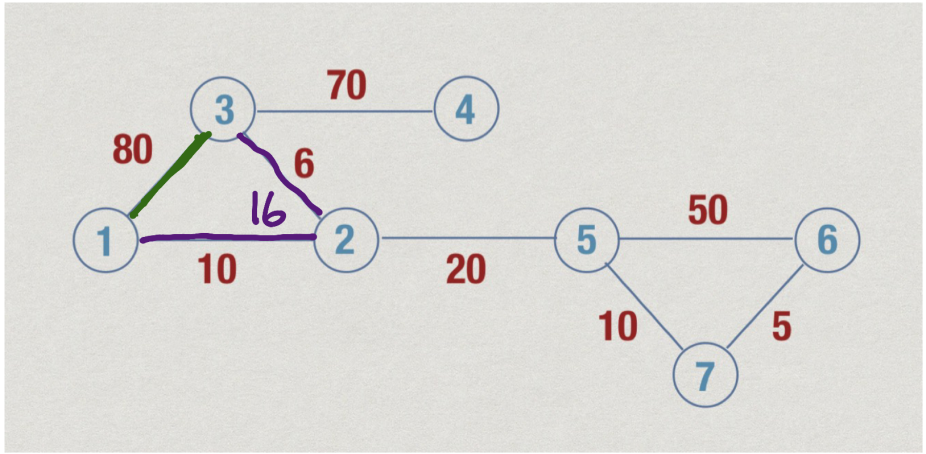
Shortest paths in weighted graphs



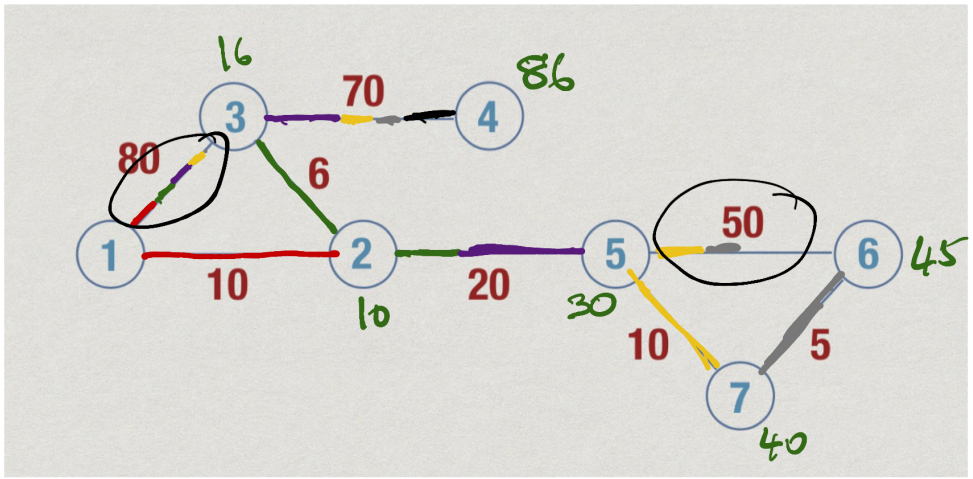
Dijkstra

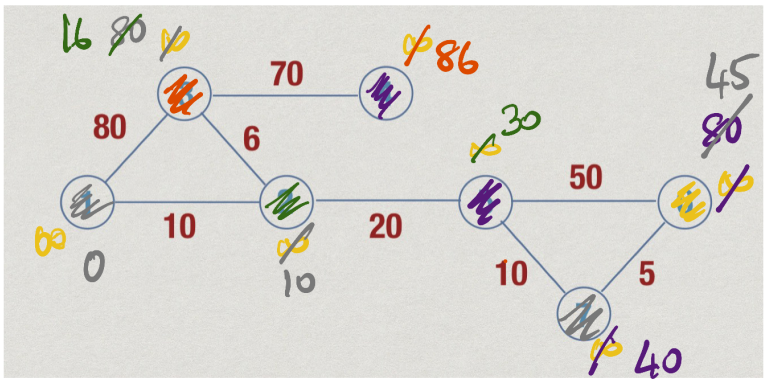
visited [i] – 0/1 Distance is final
- initially 0

distance [i] – best estimate of shortest distance
- initially ∞

if visited [i] = 1 and $(i,j) \in E$

distance [j] ≤ distance [i] + weight [(i,j)]





Dijkstra's solution

Shortest path from 1 to 7

Shortest path $v_1 \to v_2 \to v_3 \to v_4$

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 ?

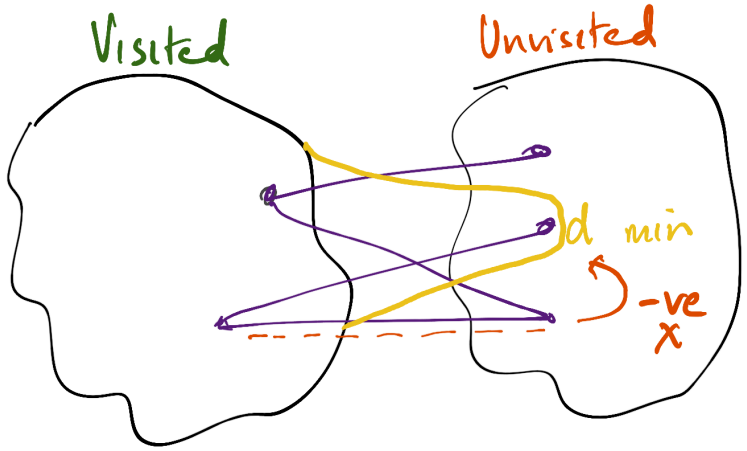
Dijkstra's algorithm

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

  Distance[s] = 0

  for i = 1 to n                                          O(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)
```

O(n) in naive case

O(degree(u)) with adj list

Correctness



Maintain distance[v] as a heap

Priority Queue

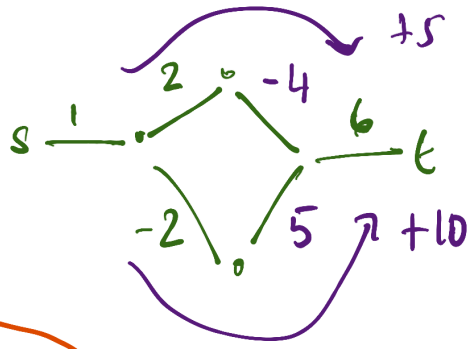
Insert / Update

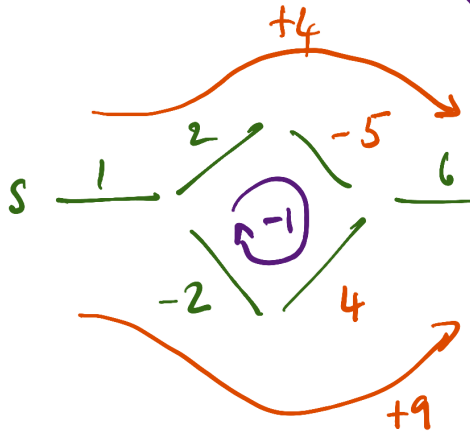
Extract_minimum

] log n

Negative edge weights ?

Negative cycle ?





Neg cycle ⇒ shortest path not well defined

Bellman-Ford algorithm

```
function BellmanFord(s)//source s, with -ve weights

for i = 1 to n
  Distance[i] = infinity

Distance[s] = 0

for i = 1 to n-1 //repeat n-1 times
  for each edge(j,k) in E
    Distance(k) = min(Distance(k),
                      Distance(j) + weight(j,k))
```

Neg edge weight, no neg cycle

Every prefix is shortest path

No loop

All Pairs Shortest Paths

n x n matrix with shortest path from every i to every j

$SP^k[i,j]$ = shortest distance with intermediate vertices in $1, 2, \ldots, k$

$SP^0[i,j] = ?$ ∞ if no edge (i,j); weight(i,j) otherwise

$SP^k[i,j]$ → don't need vertex k

$= SP^{k-1}[i,j]$

↓

need to use k

$SP^{k-1}[i,k]$

\+

$SP^{k-1}[k,j]$

min

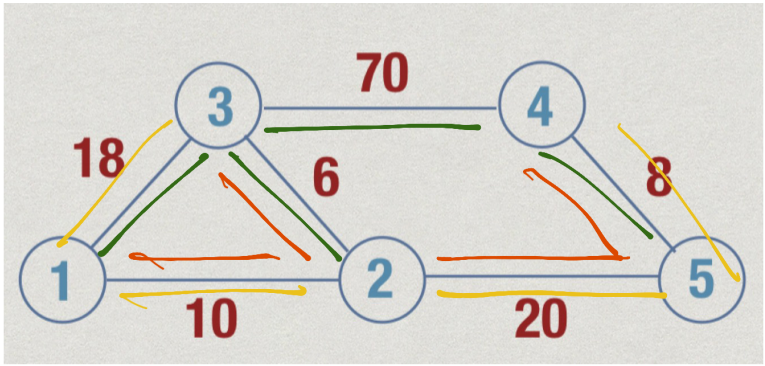
Floyd-Warshall algorithm

```
function FloydWarshall

for i = 1 to n
 for j = 1 to n
   W[i][j][0] = infinity

for each edge (i,j) in E
  W[i][j][0] = weight(i,j)

for k = 1 to n
  for i = 1 to n
    for j = 1 to n
      W[i][j][k] = min(W[i][j][k-1],
                       W[i][k][k-1] + W[k][j][k-1])
```

$W[i][j][k] \equiv SP^k[i,j]$

Spanning Trees

Minimum Cost



Assume Edge weights are distinct

1. Is lowest cost edge always in MCST

Tree + Edge

Creates Cycle

Remove any edge on cycle to restore tree

Start building an MCST with min cost edge

Prim's Algorithm

Grow the tree optimally

Kruskal's Algorithm

Add edges in increasing order