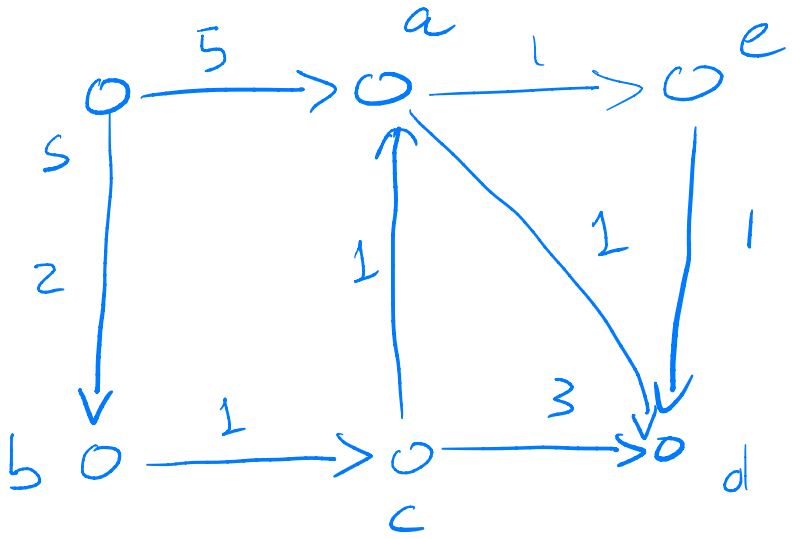
CS II Lecture 6

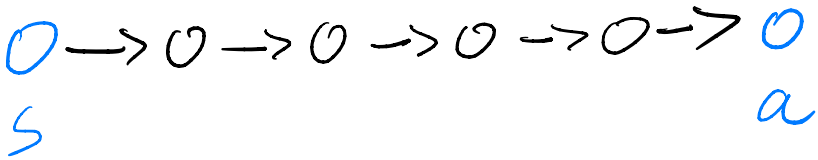
Dijkstra's Algorithm

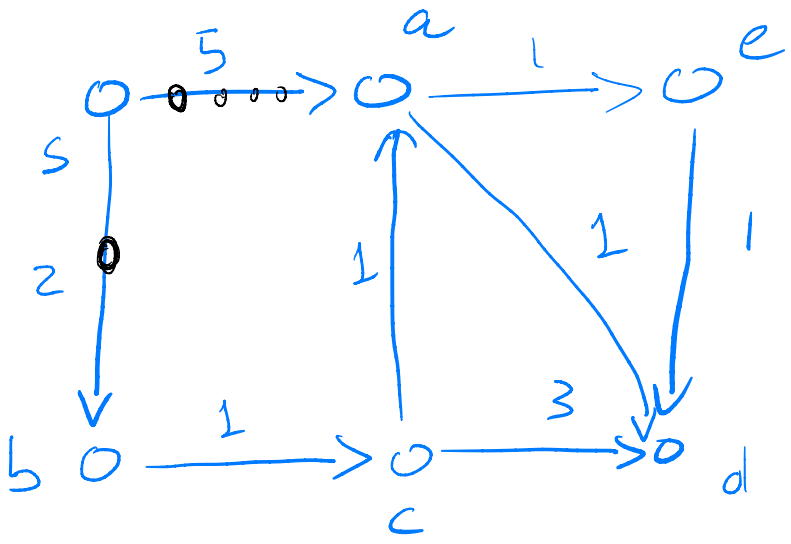
Ideas of Dijkstra



Simulate BFS in a graph where

is replaced by





- Q = s, 0

process s

- Q = a, 5 b, 2

process b

- Q = a, 5 c, 3

process c

- Q = d, 6 a, 4

process a

- Q = d, 5 e, 5

Data structure

Priority queue

Data structure in which elements have a number associated with them

We want to

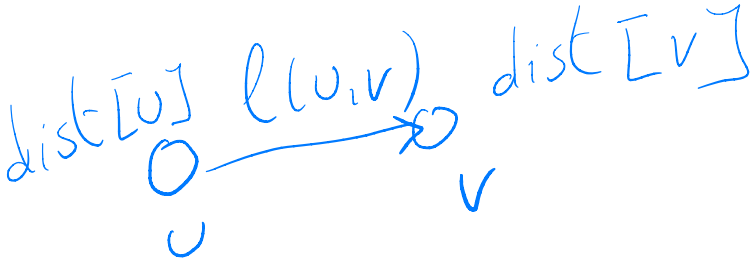
- create empty p.q.
- insert element with priority
- Find and remove with smallest priority number
- modify (decrease) priority number of a given element

Dijkstra

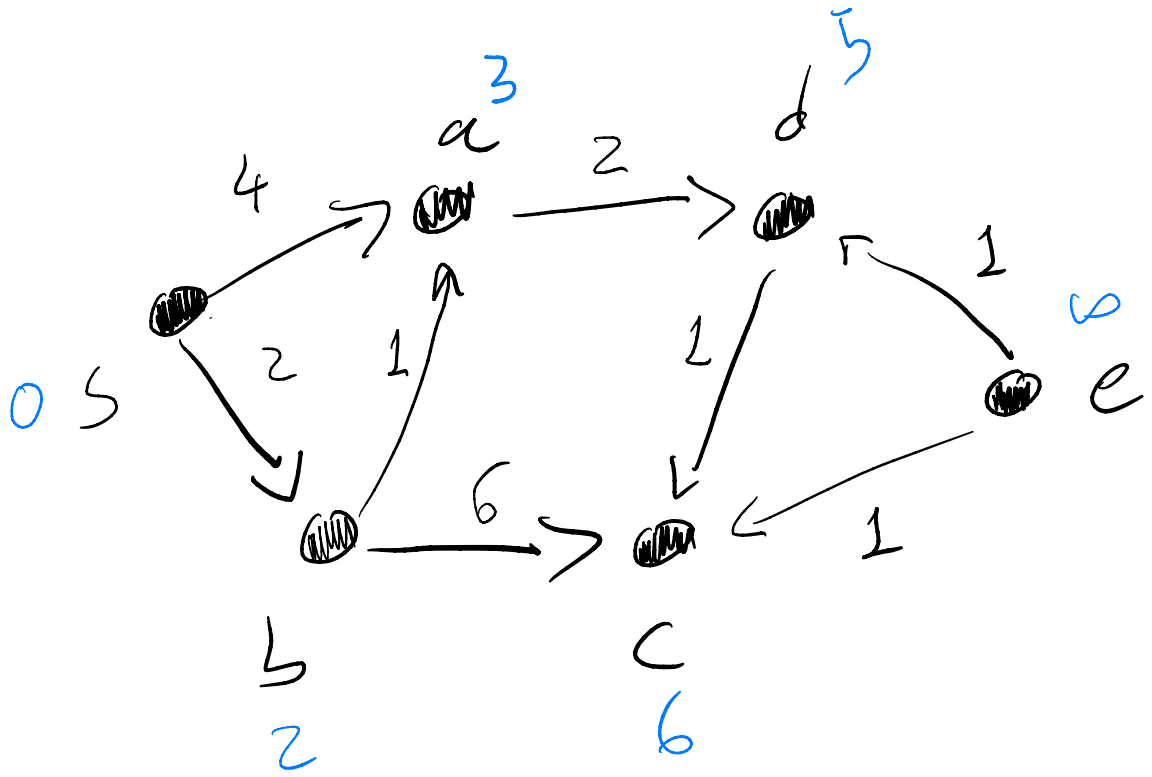
```
def Dijkstra(G, l, s)
    dist[] = array indexed by vertices
             initialized to ∞

    dist[s] = 0
    Q = priority queue with vertices,
        using dist[] as key

    while not Q.empty():
        u = Q.removemin()
        for each neighbor v of u
            if dist[v] > dist[u] + l(u,v):
                dist[v] = dist[u] + l(u,v)
                Q.decreasekey(v, dist[v])
```



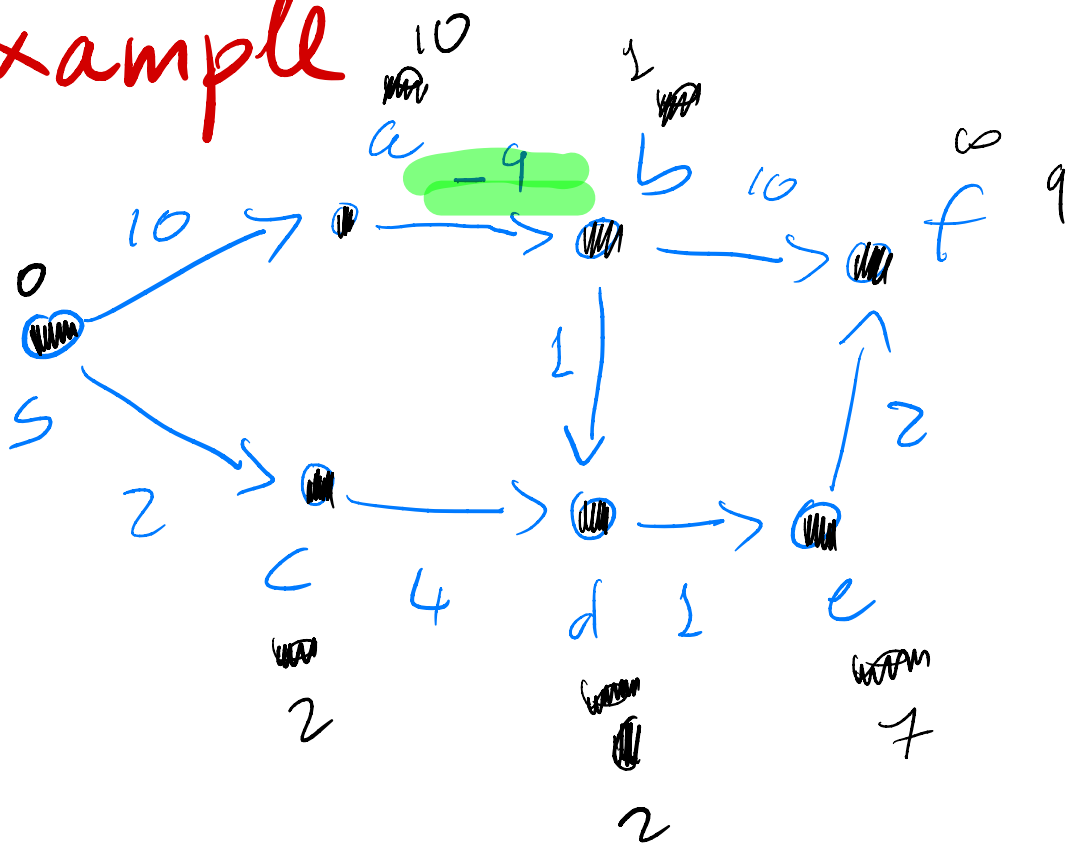
Example



Order in which nodes come out of Q

s, b, a, d, c, e

Example



Analysis

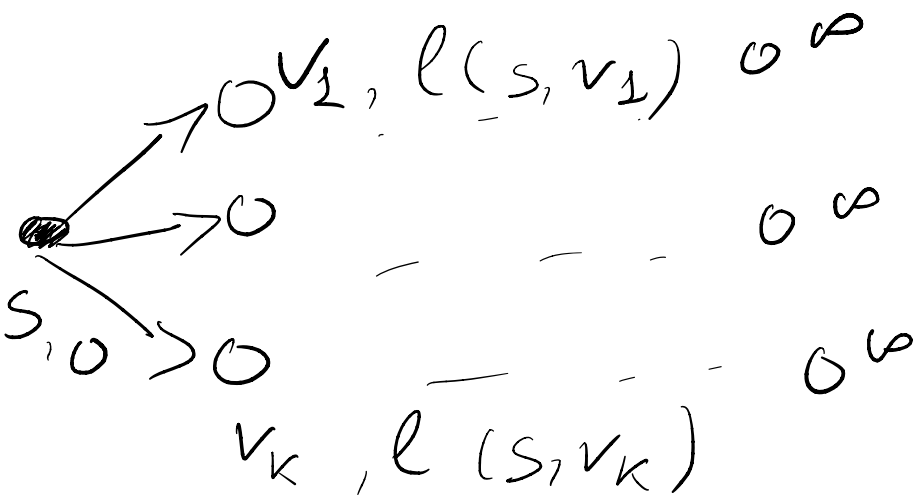
At each step, call

- black: nodes removed from Q
- white: nodes still in Q

Lemma: Assume edge weights ≥ 0.
After each iteration of "while"

- For every black v, dist[v] is correct distance from s to v
- for every node v, dist[v] is length of shortest path from s to v among paths that use only black vertices

After first iteration



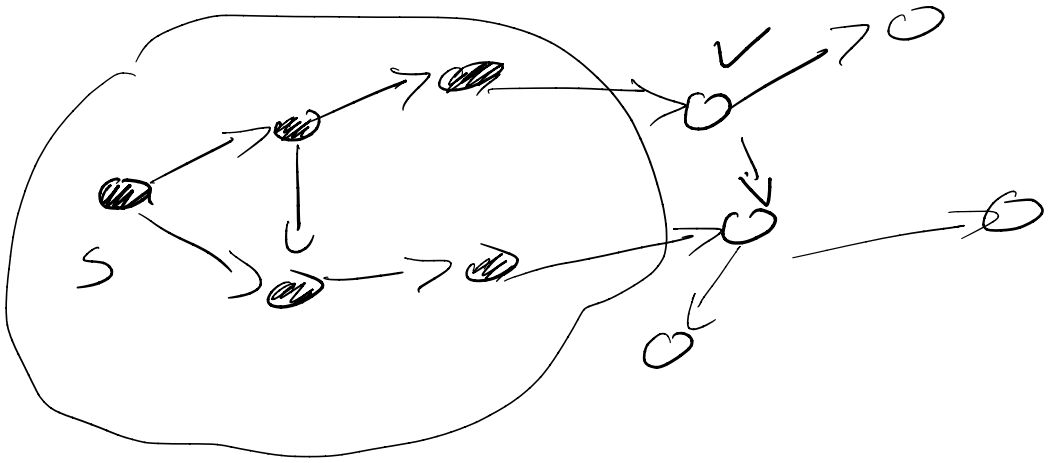
Black: only s

$dist[s] = 0$

white: all others

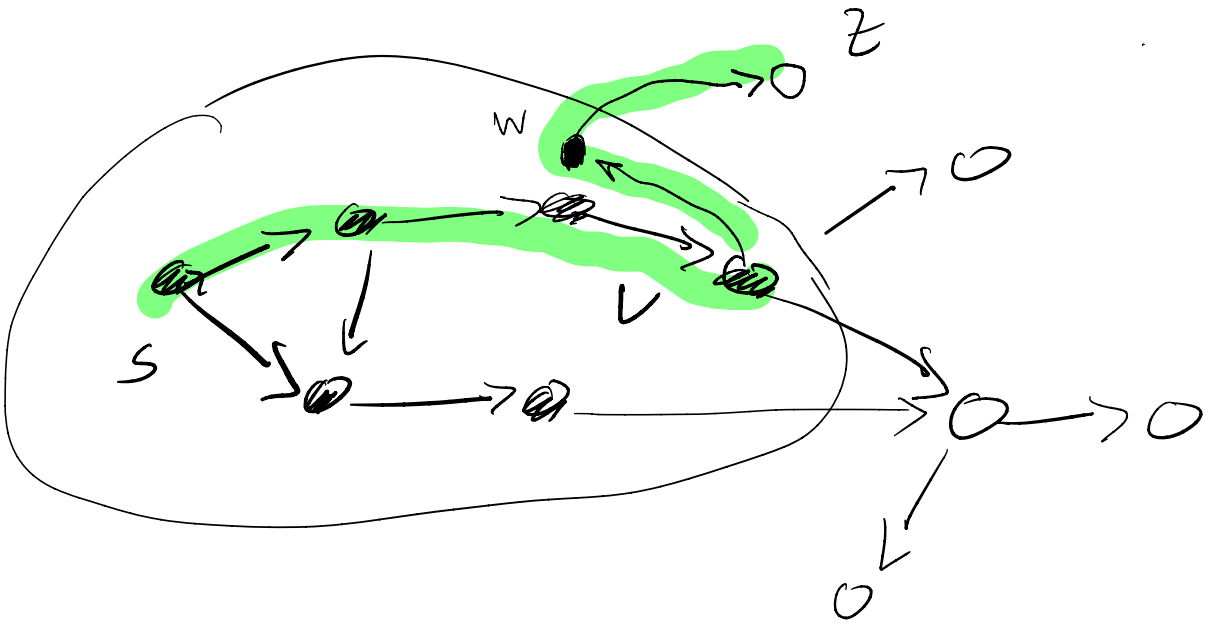
$$dist[v] = \begin{cases} \ell(s,v) & \text{if } (s,v) \text{ is edge} \\ \infty & \text{otherwise} \end{cases}$$

After t iterations



Let v be node of smallest dist [ ]

After $t+1$ iterations



Running time

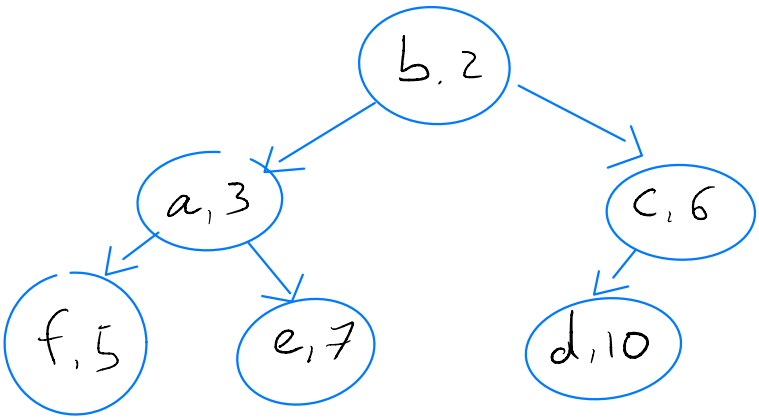
In graph with n vertices m edges

- create queue with n elements
- n removemin() operations
- $\leq m$ decreasekey(·,·) operations
- $O(n+m)$ other operations

$$O(n + n^2 + m) = O(n^2 + m)$$

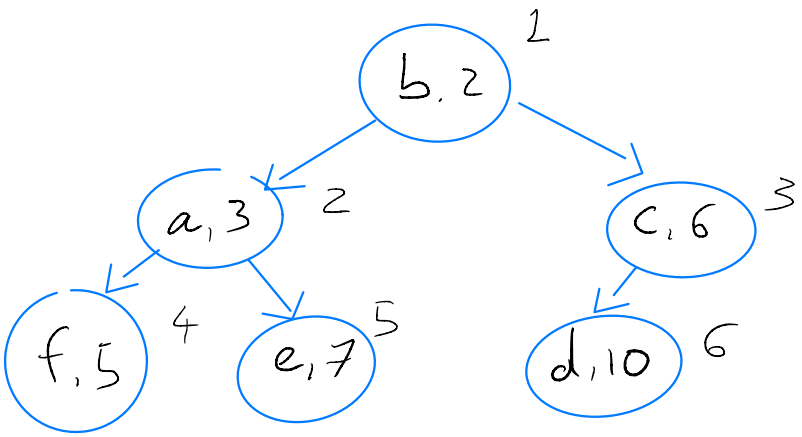
Implementing a Priority Queue as Binary Heap

a, 3 b, 2, c, 6 , d, 10, e, 7, f, 5



- Data items in a binary "full" tree
- Children's priority ≥ than parent
- Every operation takes time O(depth)
  ≤ O(log n)

Store Tree as Array



priority	2	3	6	5	7	10	∅
node	b	a	c	f	e	d	∅
	1	2	3	4	5	6	

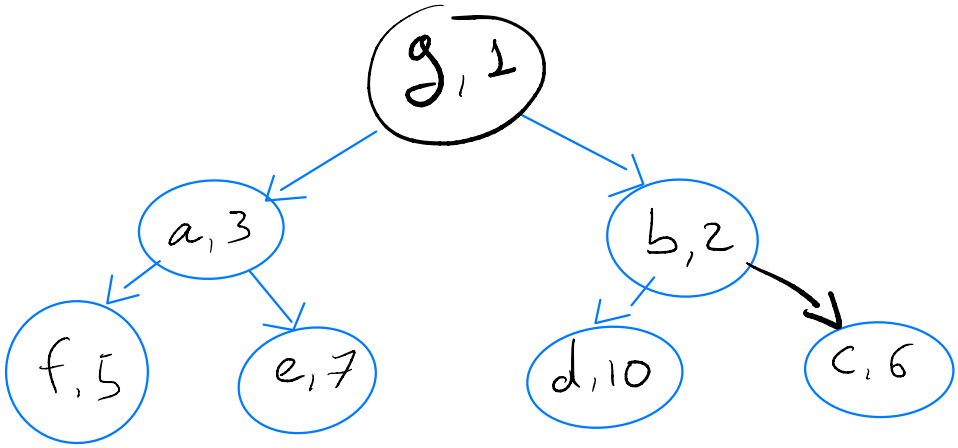
position

position	2	1	3	6	5	4	∅
	a	b	c	d	e	f	

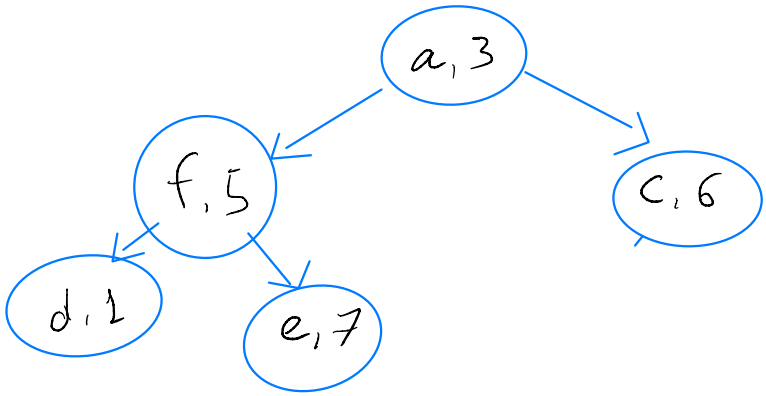
node

If a node is in position i:

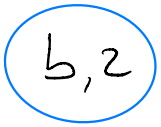
- Parent is in position $\lfloor i/2 \rfloor$
- Children are in position $2i$, $2i+1$



insert g, 1



removemin



decreasekey(d, 1)