Input: G(v, e), $s \in V$ — Nodo da cui partiamo a fare la visita

Output: dist[i] = numero di archi sul cammino minimo da s ad i
prev[i] = chi è il padre del nodo del nodo i nell'albero nato dalla BFS

```
BFS(G, s)
    for all u ∈ V
        dist[u] := +∞
        prev[u] := NIL
    dist[s] := 0
```
O(n)

```
    Q = new_queue()  //fifo
    enqueue(Q, s)
```
O(1)

```
    while not is_empty_queue(Q)
        u := dequeue(Q)
```
O(n)

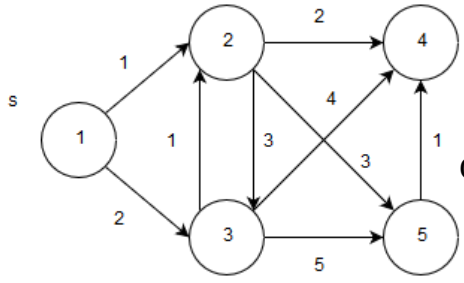
```
    for all (u, v) ∈ E
        if dist[v] = +∞
            then
                enqueue(Q, v)
                prev[v] := u
                dist[v] := dist[u] + 1
    return prev, dist
```
O(m) dipende linearmente dal numero degli archi

Costo totale BFS = O(n + m)

Cammini minimi su grafo pesato

Input: G(v, e), $s \in V$
$l: E \to R^+$

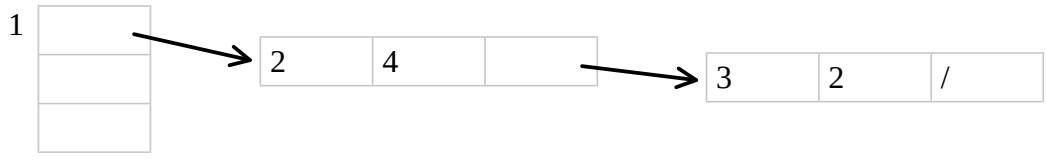
Output: albero dei cammini minimi radicato in s



G[1, 2] = 4

Il cammino minimo è quello in cui la somma dei tempi di percorrenza degli alberi è minore.

G

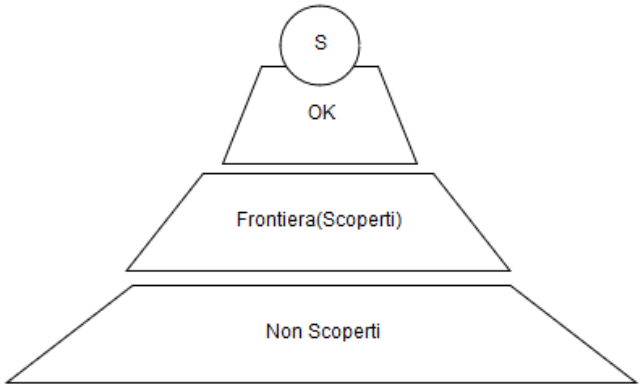


Fino a quando non ho trovato la distanza vera dalla sorgente a quel nodo non posso ritenermi soddisfatto.

La BFS si basa sul numero di archi quindi non funziona con archi pesati.

Dijkstra(shortest paths single source)

Data una sorgente voglio trovare i cammini minimi per raggiungere tutti gli altri nodi



Ok - finito di analizzarli, non dovrò più cambiare la distanza

Frontiera - nodi scoperti ma non so se il cammino che ho trovato è quello più piccolo

Dovremo utilizzare coda con priorità.
Lavoreremo in modo simile alla BFS per scoprire cammini più corti.

Coda con Priorità

(e, p) e: elemento che identifica i nodi
p: priorità

Primitive

MAKE-QUEUE(k) →Q
$k \subseteq E \; x \; P$

INSERT(Q, e, p) modifica Q con aggiunta (e, p)

MINIMUM(Q): restituisce elemento con massima priorità

DELETE_MIN(Q): modifica Q eliminando (e, p) con massima priorità e restituisce l'elemento eliminato

DECREASE_KEY(Q, e, p): modifica la priorità di e e la mette a p

```
DIJKSTRA(G, s, l)
    for all u ∈ V
        dist[u] := ∞
        prev[u] := NIL
    dist[s] := 0
    Q := make_queue(k)
    while not is_empty_queue(Q)
        u := delete_min(Q)
        for all (u, v) ∈ E
            if dist[v] > dist[u] + l(u,v)
                then
                    dist[v] := dist[u] + l(u, v)
                    prev[v] := u
                    decrease_key(Q, v, dist[v])
    return dist, prev
```

k = {(i, d) | i ∈ V, d = dist[i]}

nodi con la loro distanza

il primo estratto sarà la sorgente

RILASSAMENTO DELL'ARCO (u, v)

Esempio