Breadth-first search

Dato un grafo $G = (V, E)$ e un nodo sorgente s, la visita in ampiezza calcola la distanza tra s e ogni nodo raggiungibile a partire da s. Produce inoltre un breadth-first tree con radice s, in cui ogni percorso tra la radice e un altro nodo coincide con un cammino minimo tra s e quel nodo in G.

L'algoritmo visita i nodi in ordine di distanza da s (da qui il nome BFS), utilizzando una coda come struttura d'appoggio e colorando i nodi durante l'esecuzione: un nodo bianco non è ancora stato scoperto, uno grigio è stato scoperto ma non visitato (è nella coda), uno nero è stato visitato (è stato estratto dalla coda). La distinzione tra grigio e nero non è strettamente necessaria, ma semplifica le dimostrazioni. Diciamo che il nodo u è il predecessore di v se questo viene scoperto scansionando i vicini di u.

Con liste di adiacenza:

```
BFS(G, s)
 1  // inizializzazione
 2  for u ∈ G.V
 3       u.color = WHITE
 4       u.dist = ∞
 5       u.pred = NIL
 6  s.color = GRAY
 7  s.dist = 0
 8  Q = ∅
 9  ENQUEUE(Q, s)
10
11  // visita
12  while Q ≠ ∅
13       u = DEQUEUE(Q)
14       for v ∈ G.adj[u]
15            if v.color == WHITE
16                 v.color = GRAY
17                 v.dist = u.dist + 1
18                 v.pred = u
19                 ENQUEUE(Q, v)
20       u.color = BLACK
```

Complessità

I nodi non vengono mai colorati di bianco dopo l'inizializzazione e vengono colorati prima di essere aggiunti a Q, quindi ogni nodo non viene inserito e estratto dalla coda più di una volta ($O(V)$). La scansione delle liste di adiacenza avviene soltanto al momento dell'estrazione, quindi una volta per nodo ($O(E)$, lunghezza cumulativa delle liste). Il resto è $O(1)$, per un totale di $O(V + E)$, ovvero $O(n)$ se n è la dimensione della rappresentazione del grafo tramite liste di adiacenza.