Complessità

Visite

Il costo di visita è $\mathcal{O}(n+adj)$: adj è il tempo impiegato a controllare se esiste un nodo v bianco adiacente ad u, e dipende dalla rappresentazione; n è il numero di vertici, che vengono inseriti e rimossi da D.
Il costo di adj è:

- con lista di archi: bisogna scandire l'intera lista ($\mathcal{O}(m)$) per n volte ($\mathcal{O}(n)$), quindi $\mathcal{O}(n)+\mathcal{O}(n*m) = \mathcal{O}(mn)$
- con matrice di adiacenza: bisogna scandire l'intera riga della matrice ($\mathcal{O}(n)$), quindi $\mathcal{O}(n) + \mathcal{O}(n*n) = \mathcal{O}(n^2)$
- con liste di adiacenza: si possono ottimizzare le prestazioni utilizzando dei puntatori che puntano all'inizio delle liste di adiacenza. Se l'elemento è grigio, il puntatore è spostato all'elemento successivo; quando il puntatore giunge alla fine della lista, il primo elemento è colorato di nero. Ogni lista è percorsa una volta sola, in tutte le iterazioni del ciclo. Complessità: $\mathcal{O}(n + m)$.

Dijkstra

Ad ogni ciclo della visita bisogna estrarre il minimo dalla coda; per ogni arco trovato potrebbe essere necessario decrementare la chiave di un vertice. La complessità è quindi: $\mathcal{O}(t_c + n * t_e + m * t_d)$.

La complessità totale dipende anche dalle complessità delle operazioni sulla coda:

- con coda di priorità realizzata come sequenza **non ordinata**: $\mathcal{O}(n^2 + m)$
- con coda di priorità realizzata come sequenza **ordinata**: $\mathcal{O}(n + n * m)$

La versione dell'algoritmo di Dijkstra con coda di priorità implementata come **heap** è chiamata algoritmo di Johnson. La complessità è $\mathcal{O}((m + n) \log n)$.

Kruskal e Prim

La complessità dell'algoritmo di Prim è la stessa di quella dell'algoritmo di Johnson, quindi $\mathcal{O}((m + n) \log n)$. Siccome il grafo è connesso, $m \geq n - 1$, quindi $\mathcal{O}(m \log n)$.

Per Kruskal, la complessità è pari a $\mathcal{O}(mlogn)$, come l'algoritmo di Prim.

LCS

La costruzione della matrice ha costo $\mathcal{O}(mn)$, così come il suo popolamento. La ricostruzione della soluzione ha costo $\mathcal{O}(m + n)$: in totale, $\mathcal{O}(mn)$.

Bellman-Ford e Floyd-Warshall

L'algoritmo di Bellman-Ford fa $n-1$ iterazioni del ciclo esterno. Per ogni iterazione, l'algoritmo considera tutti gli archi del grafo con una serie di operazioni di costo $\mathcal{O}(1)$. La complessità è quindi $\mathcal{O}(n * m)$

L'algoritmo di Bellman-Ford con ottimizzazione per DAG ha complessità $\mathcal{O}(m + n)$.

L'algoritmo di Floyd-Warshall senza ottimizzazioni utilizza spazio $\mathcal{O}(n^3)$ e tempo $\mathcal{O}(n^2)$, mentre quello ottimizzato spazio $\mathcal{O}(n^3)$.