Grafi – Distanza fra partizioni

- Dato un grafo G e due sottoinsiemi V_1 e V_2 dei suoi vertici, si definisce distanza tra V_1 e V_2 la distanza minima per andare da un nodo in V_1 ad un nodo in V_2, misurata in numero di archi.
- Nel caso V_1 e V_2 non siano disgiunti, allora la distanza è 0.
- Scrivere un algoritmo mindist(Graph G, Set V_1, Set V_2) che restituisce la distanza minima fra V_1 e V_2.
- Discutere complessità e correttezza, assumendo che l'implementazione degli insiemi sia tale che il costo di verificare l'appartenenza di un elemento all'insieme abbia costo $O(1)$.
- Nota: è facile scrivere un algoritmo $O(nm)$; esistono tuttavia algoritmi di complessità $O(n^2)$ (con matrice di adiacenza) e $O(m+n)$.

Griglia quadrata

Si consideri un griglia quadrata $n \times n$ celle.

- Ogni cella è colorata con un colore in $\{1, 2, 3\}$
- Per semplicità, supponete che nella griglia sia presente almeno una cella di colore 1 e almeno una cella di colore 3.
- Supponete di partire da una cella di colore 1
- Ad ogni passo potete muovervi di una cella in alto, in basso, a destra o a sinistra
- L'obiettivo è raggiungere una cella con colore 3

Scrivere un algoritmo che prende in input una griglia rappresentata da una matrice di interi e restituisca il numero minimo di passi *necessari* per raggiungere una qualunque cella di colore 3 a partire da una qualunque cella di colore 1.
Discutere correttezza e complessità dell'algoritmo proposto.

Griglia quadrata

Ad esempio, si consideri la matrice seguente:

1	2	2	3
2	1	2	3
2	2	2	**3**
3	2	**1**	**2**

La risposta da dare è 2, perchè non esistono celle 1 e 3 adiacenti ma esistono percorsi formati da due passi (come quello evidenziato in grassetto, che però non è l'unico).

Grafi – Connetti il grafo

- Progettare un algoritmo efficiente che dato un grafo non orientato, restituisca il numero minimo di archi da aggiungere per renderlo connesso.
- Progettare un algoritmo efficiente che dato un grafo non orientato, aggiunga il numero minimo di archi necessari a renderlo connesso.

Maggioranza

Scrivere una funzione

boolean hasMajority(**int**[ ] A, **int** n)

che prenda in input un vettore ordinato A contenente n interi e restituisca **true** se A contiene un valore di maggioranza, ovvero un valore che compare più di $n/2$ volte; restituisca **false** altrimenti. Soluzioni di costo computazionale lineare non verranno prese in considerazione.

Discutere correttezza e complessità dell'algoritmo proposto.