Algoritmi e Strutture Dati – 29/06/2017

Esercizio 1

Andando per tentativi, proviamo con $\Theta(n^2)$. È facile vedere che la ricorrenza è $\Omega(n^2)$, per via della sua componente non ricorsiva. Proviamo quindi a dimostrare che $T(n) = O(n^2)$.

- Caso base: $T(n) = 1 \leq cn^2$, per tutti i valori di n compresi fra 1 e 8. Tutte queste disequazioni sono soddisfatte da $c \geq 1$.
- Ipotesi induttiva: $T(k) \leq ck^2$, per $k < n$
- Passo induttivo:

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + 4T(\lfloor n/4 \rfloor) + 15T(\lfloor n/8 \rfloor) + n^2 \\ &\leq 2c\lfloor n/2 \rfloor^2 + 4c\lfloor n/4 \rfloor^2 + 15\lfloor n/8 \rfloor^2 + n^2 \\ &\leq 2cn^2/4 + 4cn^2/16 + 15cn^2/64 + n^2 \\ &\leq 63/64cn^2 + n^2 \leq cn^2 \end{aligned}$$

  L'ultima disequazione è rispettata per $c \geq 64$.

Abbiamo quindi dimostrato che $T(n) = \Theta(n^2)$.

Esercizio 2

Il problema può essere risolto costruendo un grafo in cui i nodi sono le caselle e ogni casella è collegata alle sei caselle consecutive; ma se una delle caselle consecutive è una scala o un serpente, il collegamento è diretto con la sua casella di arrivo. Su questo grafo, è sufficiente fare una visita BFS partendo dalla casella 1 e misurando la distanza della casella n^2. La complessità risultante è $O(|V| + |E|) = O(n^2 + 6n^2) = O(n^2)$.

```
int snakesAndLadders(int[] V, int n)
    Graph G = Graph()
    for i = 1 to n^2 do
        G.insertNode(i)
    for i = 1 to n^2 do
        for k = 1 to 6 do
            if i + k ≤ n^2 then
                G.insertEdge(i, V[i + k])
    int distance = new int[1 ... n^2]
    distance(G, 1, distance)
    return distance[n^2];
```

Alcuni studenti hanno proposto di utilizzare Dijkstra, ma questo comporta una complessità più alta: $O(|V|^2) = O(n^4)$.
Si noti che soluzioni "greedy" che seguono le scale e evitano i serpenti non funzionano. Non dovete farvi fuorviare: in un gioco da tavolo ragionevole non dovrebbero esserci 6 caselle consecutive con scale o con serpenti, ma nella definzione del problema possono esserci. E magari l'unico modo per arrivare in fondo è seguire un serpente. Nemmeno preferire le scale ai serpenti è fattibile: e' possibile che esistano cicli scale-serpenti, una volta dentro non si riesce ad uscire più.

Esercizio 3

Il problema può essere risolto tramite un algoritmo greedy, in maniera simile al problema degli intervalli indipendenti (non pesati). Ogni irrigatore i definisce un intervallo $[D[i] - R[i], D[i] + R[i]]$; ordiniamo gli intervalli per estremo di inizio.
La variabile *last* contiene l'estremo superiore dell'innaffiatoio precedente, che dobbiamo innaffiare. All'inizio, questo valore è pari a $last = 0$, ad indicare che dobbiamo innaffiare a partire dall'inizio.
Fra tutti gli irrigatori il cui estremo inferiore è più piccolo di *last* (e quindi raggiungono *last*), si sceglie quello che ha come estremo superiore più alto. Si memorizza questo estremo superiore in *last*.
A questo punto, il problema si riduce al sottoproblema $[last, L]$, dove tutti gli intervalli che contenevano il valore *last* precedente non devono più essere considerati perché hanno estremo di fine inferiore al nuovo *last*.
Al termine, si verifica se il valore *last* ha superato L, che significa che siamo stati in grado di innaffiare tutta l'High Line e si restituisce *count*. Altrimenti, si restituisce -1.
Per dimostrarne la correttezza, si consideri un insieme ottimale di intervalli in cui non è stato scelto nessun intervallo che interseca quella precedente; ma questo ovviamente è impossibile, pena non aver innaffiato l'intera linea.
Supponiamo di aver scelto un intervallo che non ha il più alto estremo superiore. Sostituiamo questo intervallo con quello che ha il più alto estremo superiore. Si ottiene una nuova soluzione che ha la stessa dimensione della precedente e riesce a coprire l'High Line.
Il costo dell'algoritmo è lineare nel numero di intervalli, ma l'ordinamento richiede $O(n \log n)$.

```
int sprinkles(int[] D, int[] R, int n)
{ ordina D, R per D[i] − R[i] crescenti }
int last = 0                          % Ultimo estremo intervallo da intersecare
int count = 0                         % Numero innaffiatori da restituire
int i = 0
while i ≤ n and 0 ≤ last ≤ L do
    int max = −1
    while i ≤ n and D[i] − R[i] ≤ last do
        max = max(D[i] + R[i], max)
        i = i + 1
    last = max
    count = count + 1
return iif(last ≥ L, count, -1)
```

Il problema è risolvibile anche tramite programmazione dinamica.

Esercizio 4

Utilizziamo una rete di flusso, i cui nodi sono così definiti:

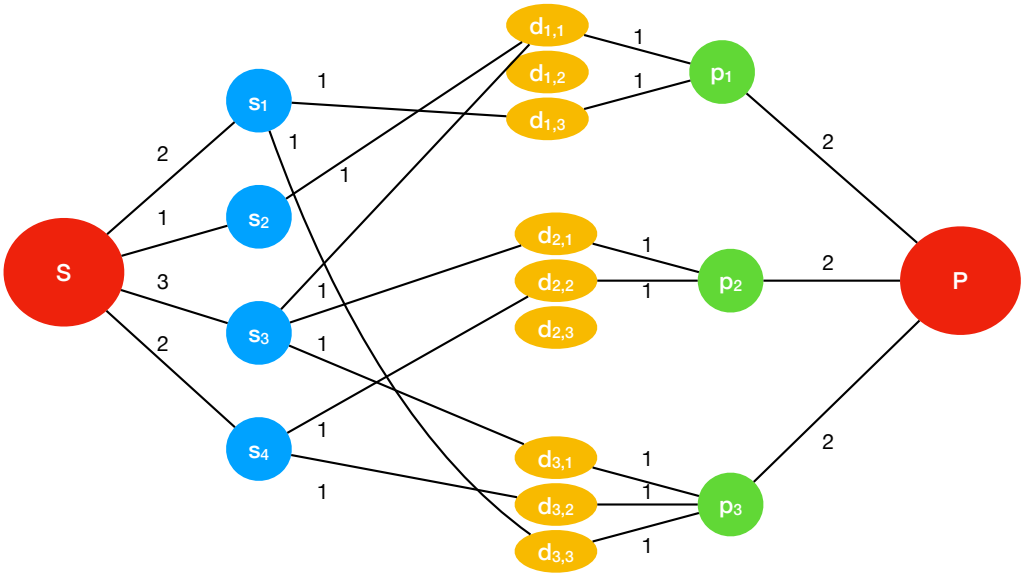
- Aggiungiamo una super-sorgente S e un superpozzo P.
- Per ogni strumento s_i e per ogni proprietà p_j, aggiungiamo un nodo
- Aggiungiamo t nodi (dove t è il numero di ditte) per ognuna delle proprietà; questi nodi sono gadget che rappresentano le ditte.

I nodi sono così collegati:

- La sorgente è collegata ad ognuno degli strumenti con una capacità pari al numero di proprietà che lo strumento è in grado di misurare;
- Il pozzo è collegato ad ognuno delle proprietà con una capacità pari al numero r di misurazioni che vogliamo effettuare (in figura, $r = 2$)

- Gli strumenti sono collegati ai gadget delle ditte, e di lì ai nodi proprietà, con archi con capacità 1. Lo strumento s_i è collegato al gadget $d_{j,k}$ e il gadget $d_{j,k}$ è collegato alla proprietà d_j se lo strumento s_i è prodotto dalla ditta d_k e se lo strumento s_i può misurare la proprietà p_j.

Lo scopo dei gadget è fare in modo che non sia possibile che strumenti della stessa ditta misurino la stessa proprietà; questo è garantito dalla capacità 1 fra gadget ditta e capacità.
Il massimo valore di flusso è pari a mr; se il flusso raggiunge esattamente questo valore, allora è possibile effettuare l'esperimento.



La complessità è data dalle seguenti misure:

$$|V| = n + m + t + 2$$
$$|E| = O(n + m + nm + tm)$$
$$|f^*| = mr$$

Utilizzando Ford-Fulkerson, il costo è limitato superiormente da $O(m^2(n + t)r)$.