Algoritmi e Strutture Dati - 29/06/2017

Esercizio 1

Andando per tentativi, proviamo con $\Theta(n^2)$. E' 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 $O(|V| + |E|) = O(n^2 + 6n^2) = O(n^2)$.
Alcuni studenti hanno proposto di utilizzare Dijkstra, ma questo comporta una complessità più alta: $O|V|^2 = O(n^4)$.
Più semplicemente, è possibile costruire la seguente relazione di ricorrenza, dove $DP[i]$ rappresenta il minimo numero di tiri del dado necessari per raggiungere la casella i-esima.

$$DP[i] = \begin{cases} 0 & i \geq n^2 \\ DP[V[i]] & i < n^2 \wedge V[i] \neq i \\ \min_{1 \leq k \leq 6}\{DP[i+k]\} + 1 & i < n^2 \wedge V[i] = i \end{cases}$$

In altre parole, nel caso si sia raggiunto o superato n^2, il numero di mosse è pari a 0 (il caso in cui si supera, in realtà, non è molto rilevante: se riesco a superare la casella finale, riesco sicuramente a raggiungerla). Se ci troviamo in una scala o serpente, il numero di mosse è pari a quello della casella di arrivo, perchè ci si sposta automaticamente; altrimenti, ci si sposta in una delle sei caselle consecutive, sommando 1 per via del tiro del dado.
Si noti che questo problema non è risolvibile con programmazione dinamica, perchè le scale e i serpenti portano avanti e indietro nel tabellone; usiamo quindi memoization.

```
snakesAndLadders(int[] V, int n, int[] DP, int i)
if i ≥ n^2 then
    return 0
if DP[i] = ⊥ then
    if V[i] ≠ i then
        DP[i] ← snakesAndLadders(V, n, DP, V[i])
    else
        DP[i] ← +∞
        for k ← 1 to 6 do
            DP[i] ← min(DP[i], snakesAndLadders(V, n, DP, i + k))
return DP[i]
```

Il costo della procedura è ovviamente $O(n^2)$, ovvero lineare nel numero di caselle (quindi ottimale). La chiamata iniziale è snakesAndLadders$(V, n,$
Un'ultima nota: 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.
Partendo dal primo intervallo, si considerino tutti gli intervalli che intersecano l'intervallo precedente (all'inizio rappresentato dal metro 0 della linea. Si sceglie fra essi quello che ha l'intervallo di fine più alto. Se nessun intervallo interseca quello precedente, si termina perchè è impossibile innaffiare l'intera linea. Si aggiorna quindi l'intervallo scelto e si ricomincia. Al termine, si verifica se l'ultimo intervallo è superiore ad L.
Per dimostrarne la correttezza, si consideri un insieme di intervalli corretti 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 destro. Sostituiamo questo intervallo con quello che ha il più alto estremo destro. Si ottiene una nuova soluzione che ha la stessa dimensione della precedente ma contiene l'intervallo che ha il più alto estremo destro.
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
    int i ← 0
    int max ← −∞
    int count ← 0
    while i ≤ n do
        if D[i] − R[i] ≤ last then
            max ← max(D[i] + R[i], max)
            i ← i + 1
        else
            last ← max
            max ← −∞
            count ← count + 1
    return iif(last ≥ L, count, −1)
```

Il problema è risolvibile anche tramite programmazione dinamica.

Esercizio 4

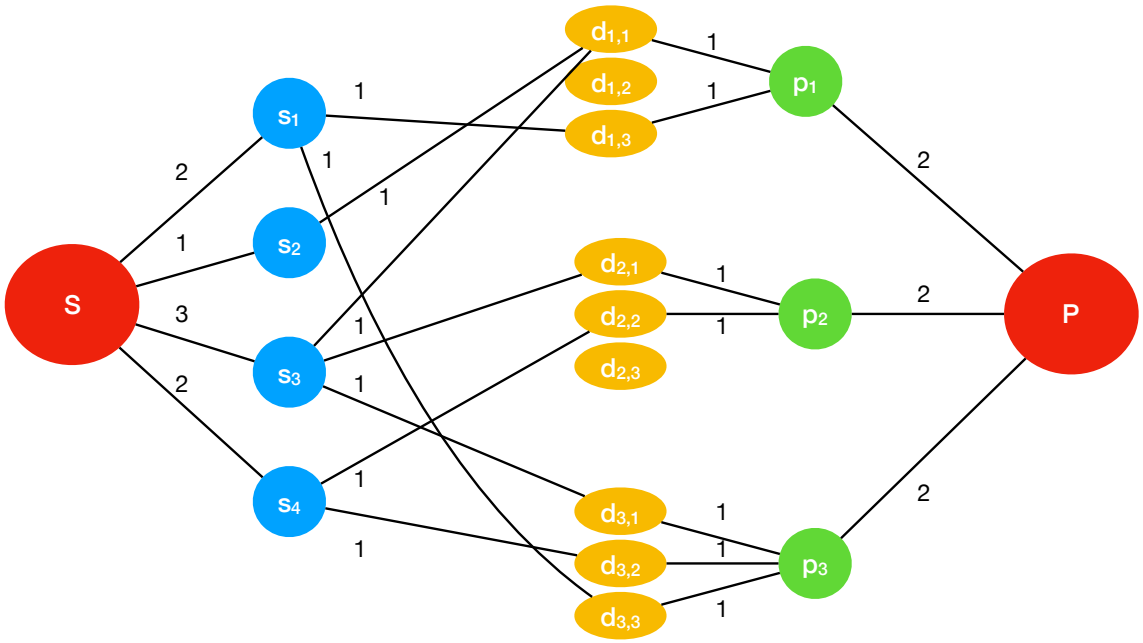
Utilizziamo una rete di flusso, i cui nodi sono così definiti:

- 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 $O(m^2(n + t)r)$.