Algoritmi e Strutture Dati – 11/07/2023

Esercizio A1 – Complessità – Punti ≥ 8

È facile vedere che $T(n)$ è $\Omega(n^3)$, per via della parte non ricorsiva. Verifichiamo se $T(n)$ è anche $O(n^3)$.

- **Caso base**: per $n = 1, \ldots, 5$, si deve dimostrare che $T(n) = 1 \leq cn^3$; questa disequazione è vera per qualunque $c \geq 1$, $c \geq 1/8$, $c \geq 1/27$, $c \geq 1/64$, $c \geq 1/125$ rispettivamente; è quindi vera per ogni $c \geq 1$.
- **Ipotesi induttiva**: ipotizziamo che per ogni valore k, $1 \leq k < n$: $T(k) \leq ck^3$;
- **Passo induttivo**: consideriamo i casi con $n \geq 6$, per i quali $\lfloor n/3 \rfloor$ e $\lfloor n/6 \rfloor$ sono maggiori o uguali a 1 (il primo caso base); applichiamo la sostituzione e otteniamo:

$$
\begin{aligned}
T(n) &= 9T(\lfloor n/3 \rfloor) + 36T(\lfloor n/6 \rfloor) + n^3 & \\
&\leq 9c\lfloor n/3 \rfloor^3 + 36c\lfloor n/6 \rfloor^3 + n^3 & \text{Sostituzione} \\
&\leq 9cn^3/27 + 36cn^3/216 + n^3 & \text{Eliminazione } \lfloor\ \rfloor \\
&\leq cn^3/3 + cn^3/6 + n^3 & \text{Passaggio algebrico} \\
&= \frac{1}{2}cn^3 + n^3 \overset{?}{\leq} cn^3 & \text{Passaggio algebrico}
\end{aligned}
$$

L'ultima disequazione è vera per $c \geq 2$.

Abbiamo quindi dimostrato che $T(n) = O(n^3)$, per $m = 1$ e $c = 2$. Poiché $T(n)$ è anche $\Omega(n^3)$, abbiamo che $T(n) = \Theta(n^3)$.

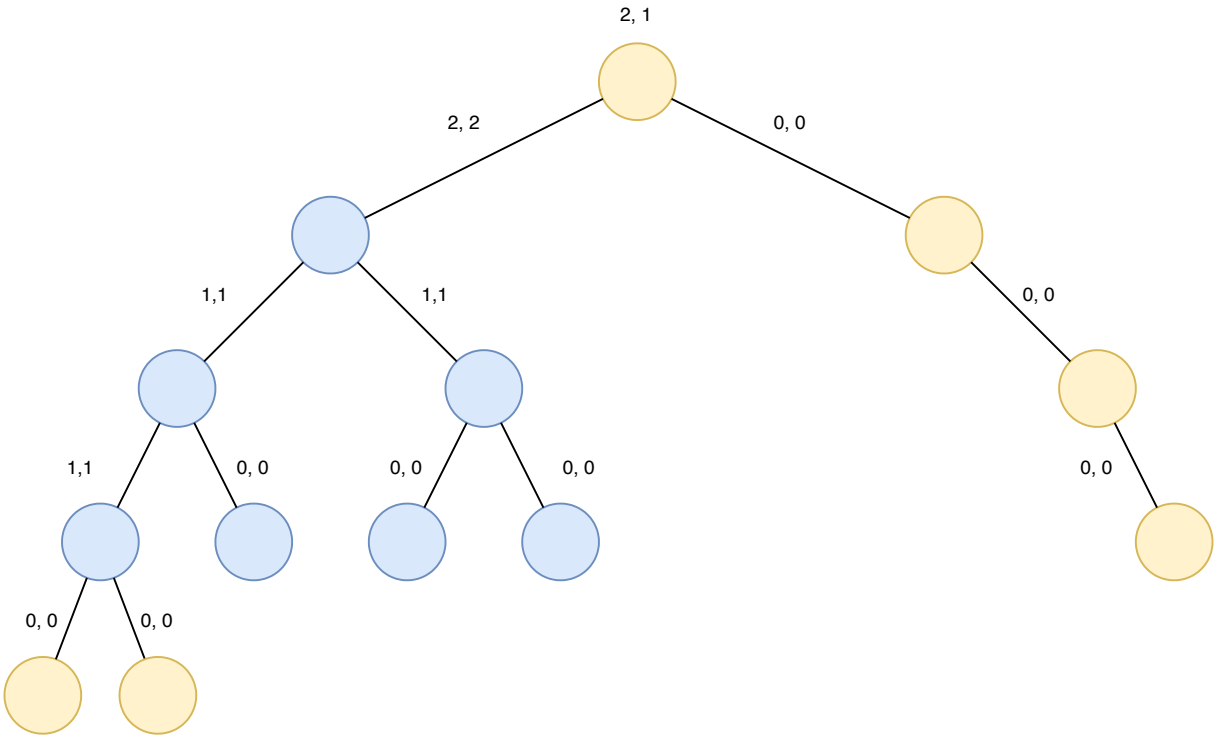
Esercizio A2 – Perfect inside – Punti ≥ 10

La funzione principale richiama una funzione ricorsiva che prende in input un nodo t e restituisce due valori: l'altezza del più grande albero binario perfetto che ha radice nel nodo t e l'altezza del più grande albero binario perfetto contenuto nel sottoalbero radicato in t.
Per calcolare l'altezza del più grande albero perfetto radicato in t, prende il minimo fra le altezze dei più grandi alberi perfetti radicati nei suoi figli sinistro e destro, e somma 1. Nel caso un nodo sia **nil**, restituisce -1 come altezza.
Vediamo alcuni casi particolari: se un nodo è una foglia, le altezze che ottiene dall'esecuzione ricorsiva sui due figli **nil** sono entrambe -1; quindi il minimo di tali valori + 1 è pari a zero, che è l'altezza di un nodo singolo. Se un nodo ha un solo figlio, l'altro è **nil**, quindi l'altezza è ancora 0.
Per calcolare l'altezza del più grandi albero perfetto contenuto in t, possiamo prendere semplicemente il massimo fra l'altezza dei più grandi alberi perfetti contenuti nei figli sinistro e destro e l'altezza del sottoalbero più grande radicato in t.
La complessità computazionale è ovviamente $O(n)$, in quando si tratta di una post-visita di un albero.

```
int largestPerfect(TREE T)
    int maxSoFar, current = lpRec(T)
    return maxSoFar
```

```
int, int lpRec(TREE t)
    if t == nil then
        return 0, −1
    else
        int, int maxSoFar_L, current_L = lpRec(t.left)
        int, int maxSoFar_R, current_R = lpRec(t.right)
        int current = min(current_L, current_R) + 1
        return max(maxSoFar_L, maxSoFar_R, current), current
```

L'esecuzione dell'algoritmo sull'albero di esempio è mostrata nella figura sottostante, dove il valore di ritorno di un nodo è mostrato nell'arco che conduce al genitore (o sopra di esso nel caso della radice).



Esercizio A3 – Island Hopping – Punti ≥ 12

Il problema può essere risolto in modo semplice con quattro cicli annidati, che confrontano la distanza fra ogni cella con valore pari a 1 da ogni cella con valore pari a 2.

```
int distance(int[][] M, int n)
    int maxSoFar
    for x1 = 1 to n do
        for y1 = 1 to n do
            if M[x1][y1] == 1 then
                for x2 = 1 to n do
                    for y2 = 1 to n do
                        if M[x2][y2] == 2 then
                            maxSoFar = max(maxSoFar, |x1 − x2| + |y1 − y2|)
    return maxSoFar
```

Per trovare una soluzione più efficiente, effettuiamo una visita in ampiezza a partire da tutte le celle con valore 1 (a cui assegniamo una distanza 0 e inseriamo tutte insieme nella coda). Identificare tutte le celle con valore 1 costa $O(n^2)$, effettuare la visita costa $O(n^2)$, quindi l'algoritmo ha costo totale $O(n^2)$.

```
int distance(int[][] M, int n)
    QUEUE Q = Queue()
    int[][] dist = new int[1...n][1...n] = {−1}
    for x_1 = 1 to n do
        for y_1 = 1 to n do
            if M[x_1][y_1] == 1 then
                Q.enqueue(⟨x_1, y_1⟩)
                dist[x_1][y_1] = 0
    int[] dx = [−1; 0; +1; 0]
    int[] dy = [0; +1; 0; −1]
    while not Q.isEmpty() do
        int, int x, y = Q.dequeue()
        for i = 1 to 4 do
            int nx = x + dx[i]
            int ny = y + dy[i]
            if 1 ≤ nx ≤ n and 1 ≤ ny ≤ n and dist[nx][ny] < 0 then
                dist[nx][ny] = dist[x][y] + 1
                if M[nx][ny] == 2 then
                    return dist[nx][ny]
                Q.enqueue(⟨nx, ny⟩)
    return +∞
```

Esercizio B1 – SWERC – Punti ≥ 8

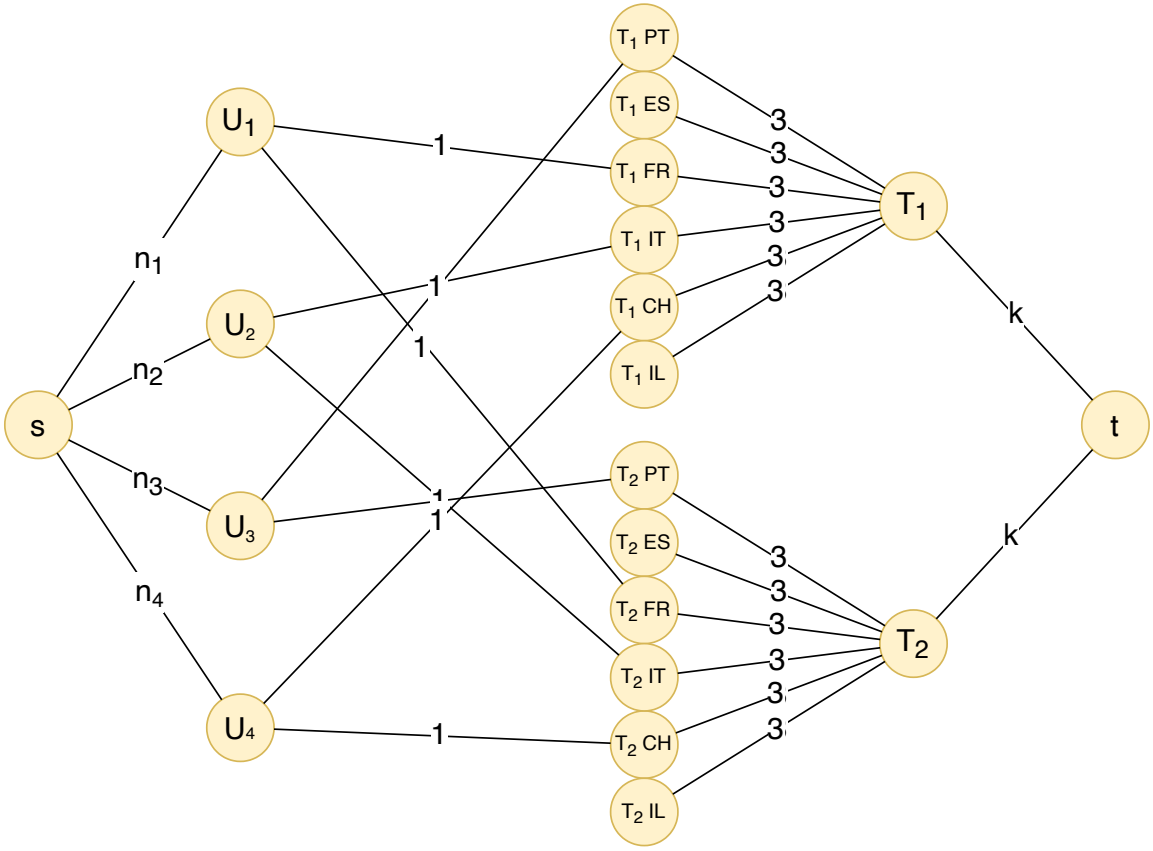
Risolviamo il problema tramite una rete di flusso.
Aggiungiamo i seguenti nodi

- Una sorgente e un pozzo (2 nodi)
- Un nodo per ogni università (n_u nodi)
- Un nodo per ogni coppia (tavolo, paese) ($6n_t$ nodi)
- Un nodo per ogni tavolo (n_t nodi)

Aggiungiamo i seguenti archi:

- Un arco dalla sorgente s all'università U_i, con capacità n_i, per limitare il numero di studenti partecipanti dell'università da assegnare ai tavoli.
- Un arco dall'Università U_i ad ogni nodo $(T_j, P(U_i))$, con capacità 1. In pratica, al massimo uno studente può essere assegnato a un qualunque tavolo, selezionando però la coppia che associa il tavolo al paese dell'università.
- Un arco da ogni nodo (T_j, p) ad ogni tavolo T_j, con capacità 3, ad indicare che ogni paese p può contribuire con al massimo 3 studenti ad ogni tavolo T_j.
- Un arco da ogni nodo T_j al pozzo t, con capacità k, per limitare il numero di studenti per tavolo.

Nella figura seguente, trovate una rete con 4 università e 2 tavoli.



Per il calcolo della complessità, utilizziamo Ford-Fulkerson.

$$V = 2 + n_u + 7n_t E = n_u + n_t \cdot n_u + 7n_t|f| = N$$

La complessità è quindi $O(Nn_t n_u)$.
Il ruolo della coppia tavolo-paese è quello di ricevere singoli studenti dalle università (limitando così a 1 il numero di studenti per tavolo) e di limitare a tre il numero di partecipanti da ogni nodo per ogni paese. Notate che inserendo solo i paesi in mezzo fra università e tavoli, viene perso il vincolo che ogni università può mandare al massimo uno studente al tavolo.

Esercizio B2 – 10.000 passi – Punti $\geq$ 10

Utilizziamo un algoritmo di backtracking che non utilizza un vettore *visited*, ma semplicemente evita di tornare indietro dal nodo da cui si arriva.
Per terminare l'algoritmo, si utilizza il parametro *len* della funzione ricorsiva che misura la distanza percorsa finora. Poiché i pesi sono positivi, quando questo valore supera 5000, è inutile proseguire. Quindi l'algoritmo proverà tutti i percorsi possibili che non tornino indietro sull'arco appena trascorso e la cui lunghezza sia inferiore o uguale a 5000 passi.
In questo particolare caso, la complessità è.... polinomiale. Nel caso pessimo, ad ogni passo dopo il primo posso fare al più $n-2$ scelte possibili (escludendo il nodo stesso e il nodo di provenienza); il numero di passi possibili è limitato da 5000, in quanto i pesi sono maggiori o uguali a 1. Per questo motivo, la complessità è $O(n^{5000})$....

```
searchPath(GRAPH G, NODE src, NODE dest)
    STACK pila = Stack()
    pila.push(src)
    spRec(G, src, nil, dest, pila, 5000)
```

```
boolean spRec(Graph G, Node curr, Node prev, Node dest, Stack pila, int len)
    if curr = dest and len == 0 then
        print pila
        return true
    else if len > 0 then
        foreach next ∈ curr.adj() − {prev} do
            pila.push(next)
            if spRec(G, next, curr, dest, pila, len − w(curr, next)) then
                return true
            pila.pop()
        return false
```

Esercizio B3 – Made-up – Punti ≥ 12

Per risolvere questo problema, proviamo prima un soluzione basata su programmazione dinamica $DP[n][\ell]$ che rappresenta il numero di sottoalberi ma-come-se-li-inventa contenenti n nodi e in cui il livello della radice del sottoalbero è ℓ.
La formula per $DP[n][\ell]$ è la seguente:

$$DP[n][\ell] = \begin{cases} 1 & n = 1 \wedge \ell \text{è pari} \\ 0 & n = 1 \wedge \ell \text{è dispari} \\ \sum_{i=1}^{n-2} DP[i][\ell+1] \cdot DP[(n-1)-i][\ell+1] & n > 1 \wedge \ell \text{è pari} \\ 2DP[n-1][\ell+1] & n > 1 \wedge \ell \text{è dispari} \end{cases}$$

Spieghiamo i quattro casi:

- Il numero di sottoalberi la cui radice si trova a livello pari e contengono un nodo è pari a 1 – il nodo stesso, che può avere 0 figli perché il livello è pari.
- Il numero di sottoalberi la cui radice si trova a livello dispari e contengono un nodo è pari a 0 - perchè questo nodo dovrebbe avere un figlio, ma esiste un nodo solo.
- Se $n > 1$ e il livello è pari, un nodo viene "consumato" come radice e i restanti $(n-1)$ nodi devono essere divisi fra il figlio sinistro e destro; sommiamo quindi tutti i casi possibili con $i > 1$ nodi a sinistra e $(n-1)-i$ nodi a destra. Questi casi si moltiplicano fra loro.
- Se $n > 1$ e il livello è dispari, un nodo viene "consumato" come radice e i restanti $(n-1)$ nodi possono trovrasi nel sottoalbero sinistro o nel sottoalbero destro; quindi abbiamo 2 volte il valore di $DP[n-1, \ell+1]$.

Volendo calcolare questa relazione abbiamo una matrice $n \times L$, dove L è il livello massimo possibile - da calcolare esso stesso. Ma notate che del valore del livello, noi utilizziamo solo l'informazione se è pari o dispari. Possiamo quindi semplificare la relazione ricorsiva così:

$$DP[n][p] = \begin{cases} 1 & n = 1 \wedge p = 0 \\ 0 & n = 1 \wedge p = 1 \\ \sum_{i=1}^{n-2} DP[i][1] \cdot DP[(n-1)-i][1] & n > 1 \wedge p = 0 \\ 2DP[n-1][0] & n > 1 \wedge \ell p = 1 \end{cases}$$

Come vedete, utilizziamo ora una matrice $n \times 2$. Un'altra possibilità è utilizzare due vettori, che rappresentano le due righe della matrice. Personalmente lo trovo più chiaro e scriverò il codice di questa versione.
$DP_p[n]$ restituisce il numero di sottoalberi contenenti n nodi che rispettano le regole, quando il livello della radice del sottoalbero è pari; $DP_d[n]$ restituisce il numero di sottoalberi contenenti n nodi che rispettano le regole, quando il livello della radice del sottoalbero è dispari.
La formula per $DP_p[n]$ è la seguente:

$$DP_p[n] = \begin{cases} 1 & n = 1 \\ \sum_{i=1}^{n-2} DP_d[i] \cdot DP_d[(n-1)-i] & n > 1 \end{cases}$$

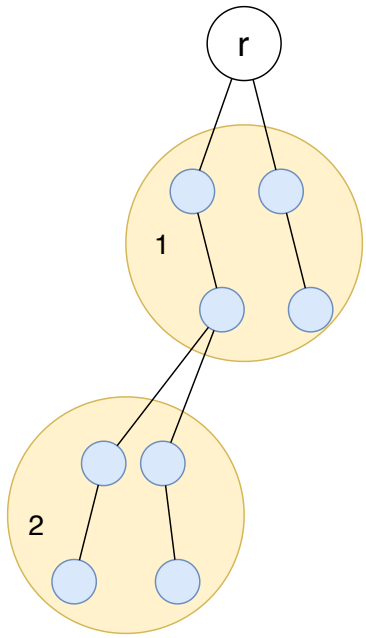
Se $n = 1$, poichè il livello della radice del sottoalbero è pari, è lecito avere 0 figli, quindi esiste un sottoalbero che rispetta le regole. Se $n > 1$, la radice del sottoalbero deve avere 2 figli, quindi si dividono i restanti $n-1$ fra sottoalbero sinistro e sottoalbero destro, mettendo almeno un nodo per sottoalbero. Le radici di questi sottoalberi sono di livello dispari.
La formula per $DP_d[n]$ è la seguente:

$$DP_d[n] = \begin{cases} 0 & n = 1 \\ 2DP_p[n-1] & n > 1 \end{cases}$$

Se $n = 1$, il numero di alberi possibili è zero, perché tale nodo dovrebbe avere un figlio, essendo a livello dispari, ma essendo da solo non può averne. Se $n > 1$, ci sono due possibilità: tutti i restanti $n - 1$ nodi sono nel sottoalbero di sinistra o nel sottoalbero di destra, che hanno la radice a livello pari.
Possiamo tradurre queste due formule tramite programmazione dinamica. Il risultato finale si troverà in $DP_p[n]$. La complessità è $O(n^2)$, per via dei due cicli annidati.

```
int countMakeup(int n)
    int[] DP_p = new int[1...n]
    int[] DP_d = new int[1...n]
    DP_p[1] = 1
    DP_d[1] = 0
    for i = 2 to n do
        DP_d[i] = 2 · DP_p[i − 1]
        DP_p[i] = 0
        for j = 1 to i − 2 do
            DP_p[i] = DP_p[i] + DP_d[j] · DP_d[(i − 1) − j]
    return DP_p[n]
```

Addendum Emanuele Zini propone una soluzione alternativa, qui riassunta. Innanzitutto, si noti che l'unico modo per costruire un albero *ma come se li inventa* è partire da una radice e aggiungere quattro nodi, disposti in una delle quattro disposizioni illustrate nel testo del compito. Non è possibile aggiungere 1,2,3 nodi, perché le proprietà non verrebbero rispettate. Questo spiega fra l'altro il fatto che n debba essere uguale a $4k + 1$, con k non negativo.
Due di questi quattro nodi si attaccano alla radice, due sotto. A questo punto, supponiamo di voler aggiungere altri quattro nodi: nella figura qui sotto, vengono agganciati al nodo sinistro del gruppo 1. Se si volessero aggiungere altri quattro nodi, potrebbero essere aggiunti attancandoli al nodo destro del gruppo 1, oppure ai nodi sinistro o destro del gruppo 2.



Ora, consideriamo i gruppi di quattro nodi come se fossero dei supernodi; nella figura, sono rappresentati dai cerchi color ocra. I supernodi formano un albero binario composto da k nodi, dove $k = (n - 1)/4$. In uno degli esercizi presenti nel sito, abbiamo già calcolato il numero di alberi binari strutturalmente diversi contenenti k nodi (Esercizio 1.8 di questo file http://disi.unitn.it/~montreso/asd/appunti/esercizi/05-alberi.pdf). Il numero di alberi binari strutturalmente è pari al k-esimo numero di Catalan (https://it.wikipedia.org/wiki/Numero_di_Catalan), che può essere calcolato, oltre che con la programmazione dinamica, tramite questa formula basata su fattoriali:

$$tree(k) = \frac{2k!}{(k+1)!k!}$$

Attenzione però: ognuno dei supernodi in realtà può contenere internamente una delle quattro disposizioni presentate nel testo del problema; quindi, dato ognuno degli alberi strutturalmente diversi contati da $tree(k)$, i k supernodi possono assumere 4^k disposizioni diverse.
Il numero totale di alberi *ma come se li inventa* è quindi pari a

$$madeup(k) = 4^k \cdot tree(k) = 4^k \cdot \frac{2k!}{(k+1)!k!}$$

oppure, espresso come valore di n:

$$madeup(n) = 4^{(n-1)/4} \cdot \frac{[(n-1)/2]!}{[(n-1)/4+1]![(n-1)/4]!}$$

Calcolare il fattoriale richiede $n-1$ moltiplicazioni, ma i numeri coinvolti crescono rapidamente. Un algoritmo efficiente per calcolare il fattoriale di n ha costo $O(n \log^2 n)$, come descritto qui: https://en.wikipedia.org/wiki/Factorial#Computation.
La formula in k genera i valori 1, 4, 32, 320, 3584, 43008, 540672, 7028736, 93716480, 1274544128, 17611882496, ..., che corrisponde a questa sequenza: https://oeis.org/A052707, ma mi fermo qui: non saprei dare un'intuizione del significato matematico di un tipo di alberi che mi sono... inventato.