Esercizio 1 Bisogna eseguire n lavori, si dispone di m operai, $m \geq n$, e di una matrice M di $n \times m$ interi dove $M[i,j]$ indica il tempo necessario ad eseguire l'i-esimo lavoro quando alla sua esecuzione sono assegnati j operai. Una distribuzione degli m operai agli n lavori è una sequenza $j_1, j_2, \ldots j_n$ di n interi positivi con $j_1 + j_2 + \ldots + j_n = m$ e determina un tempo di terminazione per l'esecuzione dei lavori pari a $\max\{M[1,j_1], M[2,j_2], \ldots, M[n,j_n]\}$. Il tempo di terminazione per l'esecuzione degli n lavori dipende ovviamente dalla distribuzione che si fa degli operai ai lavori. Si vuole determinare il minimo tempo di terminazione possibile per l'esecuzione degli n lavori.

Ad esempio: dato $n = 3$, $m = 6$ ed $M = \begin{pmatrix} 14 & 10 & 9 & 8 & 4 & 1 \\ 20 & 15 & 8 & 6 & 5 & 3 \\ 30 & 28 & 27 & 25 & 20 & 15 \end{pmatrix}$
la distribuzione $4, 1, 1$ determina un tempo di terminazione pari a 30 mentre la distribuzione $2, 2, 2$ determina un tempo di terminazione pari a 28. Il tempo di terminazione minimo è 25 ed è determinato dalla distribuzione $1, 1, 4$.

1. Descrivere in pesudo–codice un algoritmo che calcola il minimo tempo di terminazione con complessità $O(n \cdot m^2)$.
2. Modificare l'algoritmo proposto in modo che venga prodotta in output una distribuzione che determina il tempo minimo di terminazione.

Soluzione Esercizio 1

a) Usiamo il metodo della programmazione dinamica.

Per ogni $1 \leq i \leq n$ e $1 \leq j \leq m$ sia $T[i,j]$ pari al minimo tempo di esecuzione quando bisogna eseguire solo i primi i lavori e si dispone di j operai.

$T[i,j]$ è facile da definire in modo ricorsivo:

- se $j < i$ non si hanno sufficienti operai per portare a termine i lavori.
- Se $i = 1$ non si può far altro che assegnare tutti gli operai all'unico lavoro da eseguire.
- Negli altri casi all'i-esimo lavoro possono essere assegnati un numero t di operai che va da 1 a $j-1$ e per determinare la soluzione ottima basta trovare il valore t , $1 \leq t < j$ per cui risulta minima la quantità $\max\{M[i,t], T[i-1, j-t]\}$.

Riassumendo si ha:

Per ogni $1 \leq i \leq n$ e $1 \leq j \leq m$

$$T[i,j] = \begin{cases} +\infty & \text{se } j < i \\ M[1,j] & \text{se } i = 1 \\ \min_{1 \leq t < j}\{\max\{M[i,t], T[i-1, j-t]\}\} & \text{altrimenti} \end{cases}$$

Ovviamente la soluzione al nostro problema è data da $T[n,m]$. Quindi l'algoritmo può essere così descritto:

```
TEMP-ESEC-MIN:  INPUT i due interi n ed m e la matrice M
    FOR j ← 1 TO m DO T[1, j] ← M[1, j]
    FOR i ← 2 TO n DO
        FOR j ← 1 TO m DO
            T[i, j] ← +∞
            IF j ≥ i THEN
                FOR t ← 1 TO j − 1 DO
                    IF T[i, j] > max{M[i, t], T[i − 1, j − t]}THEN
                        T[i, j] ← max{M[i, t], T[i − 1, j − t]}
                ENDFOR
        ENDFOR
    ENDFOR
OUTPUT T[n, m].
```

Il calcolo del valore di una singola cella della tabella T richiede $O(m)$. Di conseguenza il costo per calcolare tutti i valori della tabella è $= O(n \cdot m^2)$.

b) Una volta calcolata la tabella T, ripercorrendo a ritroso le decisioni che hanno portato al calcolo di $T[n, m]$ si ricostruisce una distribuzione di operai con tempo di terminazione pari a $T[n, m]$.

L'algoritmo può essere così descritto:

```
i ← n
j ← m
WHILE i > 0 DO
    IF i = 1 THEN
        SOL[i] ← j
    ELSE
        t ← j − 1
        WHILE T[i, j] < max{M[i, t], T[i − 1, j − t]} DO t ← t − 1
        SOL[i] ← t
        j ← j − t
    ENDIF
i ← i − 1
ENDWHILE
OUTPUT SOL.
```

Osservando che il WHILE più interno esegue complessivamente al più $O(m)$ iterazioni, la complessità dell'algoritmo è $O(n + m)$.

Esercizio 2 Data una stringa binaria, una sottostringa di tutti simboli uguali che non si può estendere è detta *blocco*. Scrivere in pseudo–codice una procedura che preso in input un intero positivo n, stampi tutte le stringhe binarie lunghe n costituite da sequenze di blocchi di lunghezza strettamente crescente Ad esempio se $n = 6$ allora la procedura deve stampare (non necessariamente in quest'ordine): $000000, 001111, 011111, 011000, 110000, 100000, 100111, 111111$. La complessità della procedura **deve essere** O($nD(n)$), dove $D(n)$ è il numero di stringhe da stampare. **Motivare** la procedura proposta.

Soluzione Esercizio 2 Un algoritmo che risolve il problema e che si basa sulla tecnica del backtracking è il seguente. L'algoritmo CRESC, essendo ricorsivo, prende in input l'intero n, il vettore SOL in cui viene memorizzata la stringa da stampare, il numero i di simboli inseriti nel vettore SOL fino a questo momento, un intero *coda* che indica il numero di simboli uguali con cui termina la soluzione parziale e un intero *blocco* che indica la lunghezza dell'ultimo blocco terminato della soluzione parziale. Quindi la prima chiamata sarà CRESC(n, SOL, 0, 0, 0).

```
CRESC: INPUT l'intero n, il vettore SOL gli interi i, coda e blocco
    IF i = n THEN stampa la sequenza SOL[1], SOL[2], ..., SOL[n]
    ELSE
        IF i = 0 THEN
            SOL[1] ← 0
            CRESC(n, SOL, 1, 1, 0)
            SOL[1] ← 1
            CRESC(n, SOL, 1, 1, 0)
        ELSE
            SOL[i + 1] ← SOL[i]
            CRESC(n, SOL, i + 1, coda + 1, blocco)
            IF coda > blocco AND n − (i + 1) ≥ coda THEN
                SOL[i + 1] ← (SOL[i] + 1) mod 2
                CRESC(n, SOL, i + 1, 1, coda)
            ENDIF
        ENDIF
    ENDIF
```

La strategia utilizzata è quella di garantire che in ogni momento la soluzione parziale ha la proprietià che i blocchi completati sono di lunghezza crescente e ci sono simboli ancora da assegnare sufficienti a trasformare la coda in un blocco di lunghezza superiore all'ultimo blocco. Ad ogni passo posso decidere di allungare la coda con un ulteriore simbolo o trasformarla in blocco aggiungendo un simbolo diverso da quello che compone la coda. La scelta di allungare la coda non compromette la possibilitià di ottenere una soluzione mentre la creazione di un nuovo blocco può avvenire solo se la trasformazione della coda in blocco ha generato un blocco di lunghezza superiore all'ultimo blocco presente (vale a dire $coda > blocco$) e ci sono simboli ancora da assegnare a sufficienza per trasformare la nuova coda in un blocco di lunghezza perlomeno pari a quella del blocco appena creato (vale a dire $n-(i+1) \geq coda$) quindi la trasformazione di coda in blocco (vale a dire la scelta di un simbolo diverso dall'ultimo inserito nella soluzione parziale) avviene solo se risulta soddisfatto il test $coda > blocco$ AND $n - (i + 1) \geq coda$

Per quanto detto una chiamata ricorsiva sarà effettuata se e solo se la soluzione parziale fino a quel momento costruita può estendersi ad una soluzione da stampare. Il costo di un nodo dell'albero di decisione è $O(1)$ nel caso di nodo interno ed $O(n)$ nel caso di una foglia, l'altezza dell'albero è n. La complessità risultante sarà O($n \cdot D(n)$).