Esercizio 1 L'offerta didattica del vostro corso di laurea per questo semestre è di n corsi. Il numero di crediti che si consegue per ciascun esame è funzione del numero di ore effettivamente dedicate a quel corso. Potete dedicare solo m ore del vostro tempo all'università e disponendo di una matrice M di $n \times m$ interi dove $M[i, j]$ indica il numero di crediti che si ottengono seguendo j ore dell' i-esimo corso volete sapere qual'è il numero massimo di crediti che potete conseguire ripartendo in modo opportuno il vostro tempo tra i vari corsi (non necessariamente tutti i corsi vanno seguiti).

Ad esempio: dato $n = 3$, $m = 6$ ed $M = \begin{pmatrix} 2 & 3 & 4 & 5 & 7 & 8 \\ 1 & 3 & 4 & 5 & 9 & 10 \\ 3 & 4 & 5 & 6 & 8 & 10 \end{pmatrix}$
la distribuzione $4, 1, 1$ del tempo, vale a dire dedicare 4 ore al primo corso e un'ora a ciascuno dei rimanenti, permette di cumulare un numero di crediti pari a 9 mentre la distribuzione $2, 2, 2$ dove si decide di ripartire in modo equo il proprio tempo sui tre corsi frutta 10 crediti. Il numero massimo di crediti ottenibili è 12 ed è determinato dalla distribuzione $0, 5, 1$ dove si sceglie di trascurare del tutto il primo corso, dedicare 5 ore di studio al secondo e un'ora al terzo.

1. Descrivere in pesudo–codice un algoritmo che calcola il numero massimo di crediti conseguibili con complessità $O(n \cdot m^2)$. (*Suggerimento:* programmazione dinamica)
2. Modificare l'algoritmo proposto in modo che venga prodotta in output una distribuzione del tempo che massimizza il numero di crediti conseguibili.

Soluzione Esercizio 1

a) Usiamo il metodo della programmazione dinamica.

Per ogni $1 \leq i \leq n$ e $0 \leq j \leq m$ sia $T[i, j]$ pari al numero massimo di crediti conseguibili quando sono attivati solo i primi i corsi e si dispone di j ore.

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

- se $j = 0$ non si dedica tempo allo studio e quindi non si ottengono crediti.
- Se $i = 1$ non si può far altro che dedicare tutto il proprio tempo all'unico esame.
- Negli altri casi all'i-esimo corso possono essere dedicate un numero di ore che va da 0 a j. Dedicare al corso i zero ore permetterà di ottenere $T[i-1, j]$ crediti mentre dedicare al corso i t ore, con $t > 0$ permetterà di ottenere $M[i, t] + T[i-1, j-t]$ crediti. Il valore di $T[i, j]$ è quindi dato dal massimo tra $T[i-1, j]$ e $\max_{1 \leq t \leq j}\{M[i, t] + T[i-1, j-t]\}$.

Riassumendo si ha:

Per ogni $1 \leq i \leq n$ e $0 \leq j \leq m$

$$T[i, j] = \begin{cases} 0 & \text{se } j = 0 \\ M[1, j] & \text{se } i = 1 \text{ e } j > 0 \\ \max\{T[i-1, j], \max_{1 \leq t \leq j}\{M[i, t] + T[i-1, j-t]\}\} & \text{altrimenti} \end{cases}$$

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

```
MAX-CREDITI:  INPUT i due interi n ed m e la matrice M
    FOR i ← 1 TO n DO T[i, 0] ← 0
    FOR j ← 1 TO n DO T[1, j] ← M[1, j]
    FOR i ← 2 TO n DO
        FOR j ← 1 TO m DO
            T[i, j] ← T[i − 1, j]
            FOR t ← 1 TO j DO
                IF T[i, j] > M[i, t] + T[i − 1, j − t] THEN
                    T[i, j] ← 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 la distribuzione delle ore che produce un numero di crediti pari a $T[n, m]$ si può costruire ripercorrendo a ritroso le decisioni che hanno portato al calcolo di $T[n, m]$.

L'algoritmo può essere così descritto:

```
i ← n
j ← m
WHILE i > 0 DO
    IF i = 1 THEN
        SOL[i] ← j
    ELSE IF T[i, j] = T[i − 1, j] THEN
        SOL[i] ← 0
    ELSE
        t ← j
        WHILE T[i, j] > 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 che non contengono blocchi di lunghezza pari. Ad esempio se $n = 5$ allora la procedura deve stampare (non necessariamente in quest'ordine): $00000, 00010, 01000, 01010, 10001, 01110, 11101, 10111, 11111$. 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 CONS, 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 ed una variabile booleana *pari* che indica se la soluzione parziale termina con un numero pari di simboli uguali. Quindi la prima chiamata sarà CONS(n, SOL, 0, $falso$).

```
CONS:  INPUT l'intero n, il vettore SOL l'intero i e la variabile booleana pari
    IF i = n THEN stampa la sequenza SOL[1], SOL[2], ..., SOL[n]
    ELSE
        IF pari = vero THEN
            SOL[i+1] ← SOL[i]
            CONS(n, SOL, i+1, falso)
        ELSE
            IF i = n − 1 THEN
                SOL[i+1] ← (SOL[i] + 1) mod 2
                CONS(n, SOL, i+1, falso)
            ELSE
                SOL[i+1] ← SOL[i]
                CONS(n, SOL, i+1, vero)
                SOL[i+1] ← (SOL[i] + 1) mod 2
                CONS(n, SOL, i+1, falso)
            ENDIF
        ENDIF
    ENDIF
```

Per decidere se inserire o meno un elemento nella soluzione parziale senza compromettere la possibilità di ottenere una stringa da stampare si segue la seguente strategia:

Se $pari = vero$ (vale a dire la soluzione parziale termina con un numero pari di simboli uguali) allora per evitare di generare una sottostringa con un numero pari di simboli uguali bisogna necessariamente aggiungere un ulteriore simbolo uguale all'ultimo ottenuto.

Se $pari = falso$ (vale a dire la soluzione parziale non termina con un numero pari di simboli uguali) allora:
la scelta può rendere impossibile il completamento della soluzione parziale in una stringa da stampare solo se si tratta di inserire l'ultimo simbolo (vale a dire $i = n - 1$) e si decide di allungare la soluzione con un simboli uguale all'ultimo inserito (producendo quindi una sottostringa di lunghezza pari che non potrà essere ulteriormente allungata). Per evitare ciò si effettua il test $i = n - 1$ e se questo ha esito positivo la sottosequenza di lunghezza dispari con cui termina la soluzione parziale viene interrotta inserendo il simbolo diverso (completando la soluzione e generando una nuova sottostringa di lunghezza dispari pari ad uno). Se al contrario $i \neq n - 1$ allora si può aggiungere alla soluzione parziale sia il simbolo 0 che il simbolo 1 (con entrambe le scelte sarà sempre possibile completare la soluzione parziale in una stringa da stampare) e bisogna solo fare attenzione ad aggiornare in modo opportuno la variabile booleana *pari*.

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 $\mathrm{O}(n \cdot D(n))$.