Esercizio 3 [13 punti]

In una strada del centro ci sono n stalli sui cui potete collocare delle attività commerciali. Potete scegliere attività di due tipi, di tipo A e di tipo B. Intuitivamente, l'attività di tipo B vi permette di guadagnare di più ma ha dei vincoli legislativi più stringenti. In particolare, se decidete di collocare un'attività di tipo B nello stallo i, dovete lasciare liberi gli stalli $i-1$ e $i+1$, su cui quindi non potete collocare nessuna attività, né di tipo A né di tipo B. L'attività di tipo A, invece, essenzialmente non ha vincoli (se non quello chiaramente di non poter essere adiacente ad un'attività di tipo B). Il vostro obiettivo è ovviamente quello di fare più soldi possibile. A tale fine avete fatto una ricerca di mercato e sapete che collocare un'attività di tipo A nello stallo i vi farebbe guadagnare a_i, mentre una di tipo B vi farebbe guadagnare b_i (con $b_i \geq a_i$). Progettate un algoritmo di programmazione dinamica che calcoli il guadagno massimo che potete ottenere dalla strada del centro.

ES, STALLI CON ATTIVITÀ TIPO A & B

1) SE PIAZZO ATTIVITÀ TIPO B, $i+1$ & $i-1$ VUOTI

2) " " " " A, POSSO METTERE ALTRE A VICINE O B LONTANE

- $b_i \geq a_i$

- SOTTO PROBLEMA

$OPT[i, x]$ = SCORE MAX OTTENUTO SULLE CELLE DA $i, i+1, \dots, m$ SAPENDO DI PIAZZARE UN'ATTIVITÀ DI TIPO x IN i

- GOAL

$MAX\{OPT[1, x], OPT[2, x]\}$

- CASI BASE

$OPT[i, x] = 0$ SE $i > m$

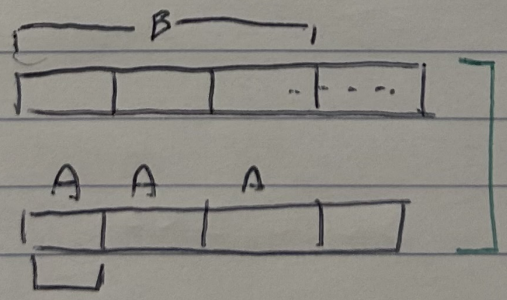
- EQ. BELLMAN

$$OPT[i, x] = \begin{cases} 0 & \text{SE } i > m \\ OPT[i+1, x] + a_i & \text{SE } x = A \\ MAX\{OPT[i+2, A] + b_i, \; OPT[i+2, B] + b_i\} & \text{SE } x = B \end{cases}$$

- COMPLESSITÀ

$O(2m) = O(m)$

- OSS = GOAL



→ 2 CASI GENERICI

Esercizio 3 [13 punti]

Si vogliono posizionare delle pedine su una scacchiera rettangolare $2 \times n$. Ad ogni cella (i, j) è assegnato un valore positivo $v_{i,j}$, $i \in \{1, 2\}, j \in \{1, \ldots, n\}$, che indica il guadagno che si ottiene se si posiziona una pedina sulla cella (i, j). Due pedine non devono mai essere adiacenti in senso verticale o orizzontale (ma possono esserlo diagonalmente). Si progetti un algoritmo di programmazione dinamica che calcoli il guadagno massimo ottenibile posizionando nel miglior modo possibile le pedine.

ES, MATRICE $2 \times m$ → ADIACENTI SOLO IN DIAGONALE

- (i, j) → PEDINA GUADAGNO V_{ij}
- PEDINE ADIACENTI SOLO IN ~~ORIZZONTALE~~ DIAGONALE

- SOTTOPROBLEMA

$OPT[i, j]$ = MAX SCORE OTTENUTO SULLE PRIME (i, j) CELLE

- GOAL

$MAX\{OPT[1, m], OPT[2, m]\}$

- CASI BASE

$OPT[i, j] = 0$ SE $j \leq 0$

- EQ. BELLMAN

$$OPT[i, j] = \begin{cases} MAX\{\overbrace{OPT[i, j-1], OPT[i+1, j-1]}^{\text{NON PIAZZO}}, \underbrace{OPT[i+1, j-1] + V_{ij}}_{\text{PIAZZO DIAGONALE}}\} & \text{SE } i = 1 \\ MAX\{OPT[i, j-1], OPT[i-1, j-1], OPT[i-1, j-1] + V_{ij}\} & \text{SE } i = 2 \end{cases}$$

- COMPLESSITÀ

$O(2m) = O(m)$

Esercizio 3 [13 punti]

Si vuole illuminare una pista ciclabile di n tratte. In ogni tratta è possibile, volendo, installare un lampione. Se installato sulla tratta i, un lampione è in grado di illuminare la tratta i e le due tratte immediatamente precedenti e immediatamente successive i (ovvero le tratte $i-2$ e $i-1$, se esistono, e le tratte $i+1$ e $i+2$, se esistono). Il costo di installazione, però, non è uniforme. Più precisamente, installare un lampione nella tratta i costa c_i. Progettare un algoritmo di programmazione dinamica che trova il costo minimo con cui è possibile illuminare l'intera pista ciclabile, ovvero il costo di una soluzione migliore possibile in cui ogni tratta è illuminata da almeno un lampione.

ES, PISTA CICLABILE e LAMPIONI

- LAMPIONE SU i, COSTA c_i e COPRE $[i-2, i+2]$
- SOTTOPROBLEMA

  $OPT[i]$ = COSTO MINIMO DI COPERTURA $(i, i+1, \dots, m)$ SE PIAZZO LAMPIONE SU i
- GOAL

  $MIN\{OPT[1], OPT[2], OPT[3]\}$
- CASI BASE

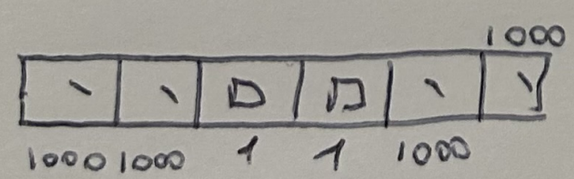
  $OPT[i] = 0$ SE $i > m$
- EQ. BELLMAN

$$OPT[i] = \begin{cases} 0 & \text{IF } i > m \\ MIN\{OPT[i+1]+c_i, OPT[i+2]+c_i, \\ \quad OPT[i+3]+c_i, OPT[i+4]+c_i, \\ \quad OPT[i+5]+c_i\} \end{cases}$$

- COMPLESSITÀ

  $O(m)$

- OSS, CASO PEGGIORE $(OPT[i+1], OPT[i+2])$



Esercizio 3 [13 punti]

Il prof. Gualà è preoccupato per la sua reputazione: sembra ormai ufficiale che la prof.ssa Di Ianni bocci agli esami più di quanto faccia lui. Ha quindi deciso, usando le sue competenze algoritmiche, di prendere dei provvedimenti per i prossimi n appelli che verranno. Sa che ad ogni appello può preparare compiti scritti di diversa difficoltà: facile, medio, difficile. Per ogni appello $i = 1, \ldots, n$, il prof. Gualà stima che il numero di bocciati saranno f_i, m_i o d_i se assegnerà uno scritto rispettivamente facile, medio, o difficile, con $f_i \leq m_i \leq d_i$. Deve però stare attento, perché se esagerasse i rappresentanti degli studenti lo dipingerebbero come un tiranno insensibile e ingiusto e quindi la sua reputazione crollerebbe. Deve quindi attenersi ai seguenti vincoli: se in un certo appello assegna un compito difficile, allora nei successivi tre (se ci sono)[1] lo scritto deve essere facile. Uno scritto medio deve sempre essere seguito (se c'è un appello successivo) da uno scritto facile.

Progettate un algoritmo di programmazione dinamica che calcoli la strategia migliore per il prof. Gualà, ovvero il modo migliore di assegnare gli scritti negli appelli al fine di massimizzare il numero di bocciati.

ESERCIZIO GIVALÀ e NUMERO BOCCIATI

- SOTTOPROBLEMA

OPT[i, x] = NUMERO MAX BOCCIATI SUI GIORNI DA i, i+1, ..., m, SAPENDO DI CONSIDERARE IL COMPITO DI DIFFICOLTÀ x.

- GOAL:

MAX{OPT[1, X]}

- CASI BASE:

OPT[i, x] = 0 SE i > m

- EQ. BELLMAN

1) SE x = F

OPT[i, x] = MAX{OPT[i+1, x]}

2) SE $x = M$

$$OPT[i,x] = MAX\{OPT[i+2, x]\} + m_i + f_{i+1}$$

3) SE $x = D$

$$OPT[i,x] = MAX\{OPT[i+4, x]\} + d_i + f_{i+1} + f_{i+2} + f_{i+3}$$

- COMPLESSITÀ =

$O(n) = O(3n)$

Esercizio 3 [13 punti]

Un drone equipaggiato con una batteria a energia solare si muove su un territorio descritto da una matrice M con n righe e m colonne. Deve andare dal punto in alto a sinistra, di cordinate $(1, 1)$, al punto in basso a destra, di cordinate (n, m). Soffia un forte vento che impedisce al drone di spostarsi in tutte le direzioni. In particolare, il drone può spostarsi ogni volta solo di una cella a destra o di una cella in basso rispetto alla posizione in cui si trova, operazione che gli fa consumare una unità di energia della batteria. La batteria ha un capacità massima di Δ unità, all'inizio completamente carica. Ovviamente il drone non può mouversi se il livello della sua batteria è a 0. Esso, però, è equipaggiato di pannelli ricettivi che gli permettono di ricaricare la batteria qualora passasse in punti in cui i raggi solari sono particolarmente forti. Più precisamente, per ogni posizione (i, j), è noto il numero $M[i, j] \geq 0$ di unità di energia che si ottengono quando si passa nella posizione (i, j). La batteria non può accumulare più di Δ unità di energia, ma non si danneggia se sottoposta a raggi solari quando è già al massimo della carica. Progettare un algoritmo di programmazione dinamica che calcoli quale è il massimo livello di batteria con cui il drone può arrivare a destinazione.

ES, DRONE, PANNELLI SOLARI, Δ.

- DA (1,1) A (n,m) FACENDOSI DI AVERE MAX BATTERIA.
- SOTTO PROBLEMA

  OPT[i,j] = MAX BATTERIA POSSIBILE & OTTENIBILE DALLA CELLA (1,1) A (i,j)
- GOAL

  OPT[n,m]
- CASI BASE

  OPT[i,j] = 0 SE i,j < 0

  OPT[1,1] = Δ
- EQUAZIONE BELLMAN
- OSS = DEVI VERIFICARE ~~SE~~ SE HA ALMENO 1 DI BATTERIA PER SPOSTARTI

$$
-\mathrm{OPT}[i,j] = \begin{cases} 0 & \text{SE } i \text{ VS } \emptyset \\ \mathrm{OPT}[i-1,j] + M[i,j] - 1 & \left[\begin{array}{l} \text{SE } \mathrm{OPT}[i-1,j] \geq 1 \text{ e} \\ \mathrm{OPT}[i,j-1] < 1 \end{array}\right] \\ \mathrm{OPT}[i,j-1] + M[i,j] - 1 & \left[\begin{array}{l} \text{SE } \mathrm{OPT}[i,j-1] \geq 1 \text{ e} \\ \mathrm{OPT}[i-1,j] < 1 \end{array}\right] \\ \mathrm{MAX}\{\mathrm{OPT}[i-1,j] + M[i,j] - 1, & [\text{ALTRIMENTI}] \\ \quad \mathrm{OPT}[i,j-1] + M[i,j] - 1\} & \end{cases}
$$

- <u>COMPLESSITÀ</u>

$O(mn)$

Esercizio 3 [13 punti]

Nella versione Beta di Pokemon Go la mappa è una sola strada di $n+1$ tratte. Tu sei nella posizione 0 e ad ogni istante di tempo sei costretto a spostarti a destra di una posizione, dalla posizione corrente i alla posizione $i+1$. In ogni posizione $i \geq 1$ ti si presenta davanti un pokemon. Tu puoi fare una sola delle seguenti due mosse:

- raccogliere le k_i pokemon ball che sono nella posizione i;
- lanciare al pokemon delle pokemon ball al fine di catturarlo. Il numero di ball che ti permettono di catturare il pokemon nella posizione i è p_i e tu, chiaramente, ne devi precedentemente aver raccolte a sufficienza.

All'inizio, quando sei nella posizione 0, non possiedi pokemon ball. Chiaramente $\Delta = \sum_{i=1}^{n} k_i$ è il numero massimo di pokemon ball che puoi raccogliere nell'intero percorso. Progettare un algoritmo di programmazione dinamica che calcoli il numero massimo di pokemon catturabili. L'algoritmo deve avere complessità polinomiale in n e Δ.

ES, POKEMON GO BETA

- SOTTOPROBLEMA

OPT[i, P] = OTTIMO GUADAGNATO SULLE PRIME i TRATTE AVENDO P POKEBALL A DISPOSIZIONE

- GOAL

OPT[m+1, Δ]

- CASI BASE

OPT[i, P] = 0 SE i ≤ 0

- EQ. BELLMAN

$$OPT[i, P] = \begin{cases} 0 & \text{SE } i \leq 0 \\ OPT[i-1, P] & \text{SE } p_i + k_i > P \\ MAX\{OPT[i-1, P], OPT[i-1, P-p_i-k_i]+1\} & \end{cases}$$

- COMPLESSITÀ =

$O(m\Delta)$, PSEUDOPOL.

Esercizio 3 [13 punti]

Nel corridoio del museo di Tor VerLouvre ci sono n vani, ognuno dei quali ospita un quadro. Si vogliono istallare delle telecamere per controllare gli n quadri del corridoio. In corrispondenza di ogni vano i, è possibile istallare una telecamera e regolarne il raggio di azione $r \in \{0, \ldots, n-1\}$. Una telecamera istallata nel vano i con raggio di azione r è in grado di controllare i quadri presenti dal vano i al vano $i+r$ e costa (in termini di consumo energetico) $c_{i,r}$. Si assuma che per ogni i, il costo $c_{i,r}$ è crescente al crescere del raggio r.

Progettare un algoritmo di programmazione dinamica che, presa in input la matrice dei costi $c_{i,r}$, calcola il costo minimo necessario per controllare gli n quadri del corridoio.

ES, TORVERGATA & TELECAMERE

- SOTTOPROBLEMA

  OPT[i] = COSTO MINIMO PER LA COPERTURA ~~DI~~ CON LE TELECAMERE DA $i, i+1, \dots, n$

- GOAL

  OPT[1]

- CASI BASE

  OPT[i] = 0 SE $i > n$

- EQ. BELLMAN

$$OPT[i] = \min_{R} \{ OPT[i+R+1] + c_{iR} \}$$

- COMPLESSITÀ

$$O(nR) = O(n \cdot n) = O(n^2)$$

Esercizio 3 [13 punti]

State progettando una traversata intergalattica con la vostra astronave e dovete fare n viaggi. La vostra astronave può volare in due diverse modalità (A e B). Effettuare il viaggio i-esimo nella modalità $x \in \{A, B\}$ ha un costo $c_{i,x}$. La vostra astronave si trova inizialmente in modalità A. Una volta scelta una modalità per un viaggio non è possibile cambiarla in volo. Però all'inizio di ogni viaggio potete decidere di riconfigurare l'astronave e cambiare modalità (anche all'inizio del primo viaggio). Il cambio di modalità non costa niente, ma avete pezzi di ricambio che vi consentono di cambiare modalità al più tre volte. Potete arrivare a destinazione in una qualsiasi modalità.

Progettare un algoritmo di programmazione dinamica che calcola il costo di una traversata di costo minimo.

ESAME 20/7/18, VIAGGI STELLARI, 2 MODALITÀ (A, B) MAX 3 CAMBI

=> PARTE MOD A e MAX 3 CAMBI

- SOTTOPROBLEMA:

$OPT[i, x, c]$ = COSTO MINIMO DELLA TRAVERSATA SULLE PRIME i TRATTE SAPENDO DI FARE L'i-ESIMA IN MODALITÀ x AVENDO A DISPOSIZIONE c CAMBI

- GOAL

$$\max_x \{OPT[n, x, C]\}$$

- CASI BASE

$OPT[i, x, c] = 0$ SE $i < 0$

$OPT[1, A, c] = C_{1A}$ $\forall c$

- EQ. BELLMAN

$i = 2$

$$OPT[i, x, c] = \begin{cases} \max\{OPT[i-1, A, v] + C_{iA}, OPT[i-1, B, v-2] + C_{iA}\} & \text{SE } x = A \\ \max\{OPT[i-1, B, v-1] + C_{iB}, OPT[i-1, A, v-1] + C_{iB}\} \end{cases}$$

SE $i > 2$

$$OPT[i, x, c] = \begin{cases} \max\{OPT[i-1, A, v] + C_{iA}, OPT[i-1, B, v-1] + C_{iA}\} & \text{SE } x = A \\ \max\{OPT[i-1, B, v] + C_{iB}, OPT[i-1, A, v-1] + C_{iB}\} & \text{SE } x = B \end{cases}$$

- COMPLESSITÀ

$O(n \cdot 2 \cdot 3) = O(n)$

Esercizio 3 [13 punti]

Su una scacchiera $2 \times n$ potete piazzare esattamente n pedine, una per ogni colonna. Per ogni pedina potete scegliere il colore, bianco o nero. Piazzare una pedina nella posizione (i, j) vi fa guadagnare $b_{i,j}$ se la pedina è bianca, e $n_{i,j}$ se è nera. Avete il seguente vincolo: per ogni $j > 1$, la pedina che piazzate sulla colonna j deve avere lo stesso colore oppure trovarsi nella stessa riga della pedina che avete piazzato nella colonna $j - 1$.

Progettare un algoritmo di programmazione dinamica che calcola il massimo punteggio ottenibile.

ES, $2 \times m$ SULLA STESSA RIGA O COLONE UGUALE

- m PEDINE, $\forall j \geq 1$:

1) PEDINA SU j CON COLONE $= j-1$
2) " " " SULLA STESSA RIGA $j-1$

- SOTTOPROBLEMA

$OPT[i, j, c]$ = MAX SCORE OTTENUTO FINO CELLA i, j PIAZZANDO UNA PEDINA DI COLONE c

- GOAL

$MAX_c \{OPT[2, m, c]\}$

- CASO BASE

$OPT[i, 5, c] = 0$ SE $i < 0$

- EQ. BELLMAN, $\forall j \geq 1$

- CASO RIGA 1, COL BIANCO

$$OPT[1, j, B] = MAX \begin{cases} OPT[1, j-1, B] + B_{1j}, \\ OPT[1, j-1, N] + B_{1j}, \\ OPT[2, j-1, B] + B_{1j} \end{cases}$$

STESSA RIGA, COLONE → NON RILEVANTE

OPPURE

→ STESSO COLORE

- CASO RIGA 1, COL NERO

$$OPT[1, j, N] = MAX \begin{cases} OPT[1, j-1, N] + N_{1j}, \\ OPT[1, j-1, B] + N_{1j}, \\ OPT[2, j-1, \cancel{N}] + N \end{cases}$$

- CASO RIGA 2, COL BIANCO

$$OPT[2, j, B] = MAX \begin{cases} OPT[2, j-1, B] + B_{2j}, \\ OPT[2, j-1, N] + B_{2j}, \\ OPT[1, j-1, B] + B_{2j} \end{cases}$$

- CASO RIGA 2, COL NERO

$$OPT[2, j, N] = MAX \begin{cases} OPT[2, j-1, N] + N_{2j}, \\ OPT[2, j-1, B] + N_{2j} \\ OPT[1, j-1, N] + N_{2j} \end{cases}$$

- COMPLESSITÀ

$O(2m2) = O(m)$

Esercizio 3 [13 punti]

Sia data una scacchiera rettangolare $1 \times n$ di n celle. Su ogni cella i è possibile posizionare una pedina nera ottenendo n_i punti, oppure una pedina bianca ottenendo b_i punti con $b_i \geq n_i$. Un piazzamento di pedine si dice *indipendente* se per ogni coppia di celle adiacenti almeno una delle due è vuota. Progettare un algoritmo di programmazione dinamica che trova un piazzamento indipendente di punteggio massimo, sapendo di avere a disposizione n pedine nere e solo una pedina bianca.

ES, 20/7/2018 N PALLINE NERE e 1 BIANCA, PDF ESAMI 1°.

=> ABBIAMO SCACCHIERA $1 \times m$, m NERE 1 BIANCA tc. $b_i \geq m_i$, VOGLIAMO GUADAGNARE MAX

- SOTTOPROBLEMA

$OPT[i, b]$ = GUADAGNO/SCONTO MAX SULLE PRIME i CELLE AVENDO A DISPOSIZIONE b PEDINE BIANCHE A DISP.

- GOAL

$OPT[m, B]$, CON $B = 1$

- CASI BASE

$OPT[i, b] = 0$ SE $i < 0$

- EQ. BELLMAN

$$OPT[i, b] = \begin{cases} 0 & \text{SE } i < 0 \\ \max\{OPT[i-2, b] + m_i, OPT[i-1, b]\} & \text{SE } b = 0 \\ \max\{OPT[i-2, b] + m_i, OPT[i-1, b], OPT[i-2, b-1] + b_i\} & \end{cases}$$

- COMPLESSITÀ

$O(mB) = (B = 1) = O(m)$

Esercizio 3 [11 punti] Un casinò di Las Vegas ha ideato il seguente gioco promozionale per attrarre nuovi giocatori. All'ingresso del casinò ci sono due grosse pile di fiches, etichettate A e B. Le pila A contiene n fiches ed il valore della generica i-esima fiche dal fondo di A è $a_i > 0$. La pila B contiene m fiches ed il valore della generica i-esima fiche dal fondo di B è $b_i > 0$. Il gioco funziona come segue: il giocatore effettua una sequenza di mosse, dove ogni mossa consiste nel selezionare una delle pile tra A e B ed in eseguire entrambe le seguenti azioni:

- rimuovere e tenere per sé la fiche in cima alla pila selezionata.
- rimuove e scartare due fiches dalla cima della pila non selezionata.

Il gioco continua fino al momento in cui in non esistono più mosse ammissibili (cioè fino a quando almeno una delle due pile diventa vuota, o entrambe le pile hanno esattamente una fiche). Progettate un algoritmo di programmazione dinamica che prende in input m, n, ed i valori di tutte le fiches nelle pile A e B e restituisce il massimo valore complessivo delle fiches che un giocatore riesce a tenere per sé. Si discuta la complessità temporale dell'algoritmo proposto.

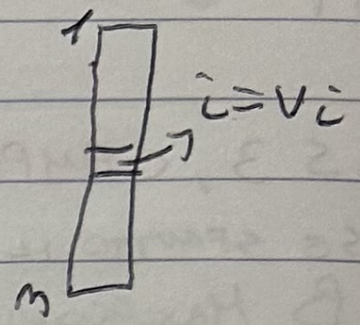
Esercizio 3 [11 punti] Avete davanti a voi una pila di n monete dove la moneta i-esima a partire dall'alto ha valore v_i. Giocate un gioco. Le mosse a disposizione sono di due tipi:

- `PrimaDiTre`: Vi mettete in tasca la moneta in cima alla pila ma dovete poi rimuovere e buttare le successive due monete (questa mossa può essere eseguita solo se la pila ha almeno 3 monete);
- `SecondaDiDue`: Rimuovete e buttate la prima moneta e vi mettete in tasca la successiva (questa mossa può essere eseguita solo se la pila ha almeno 2 monete).

Ovviamente siete interessati a trovare una sequenza di mosse che vi fa guadagnare il più possibile. Progettate un algoritmo di programmazione dinamica che calcola il massimo guadagno.

ES 3 MONETE IN PILA

1) 1° MONETA GUADAGNO e BUTTO 2 SUCC. } MOSSE
2) BUTTO 1° e GUAD. 2° } DISP.



- SOTTOPROBLEMA

OPT[i] = MAX GUADAGNO OTTENUTO SULLE MONETE DA i, i+1, ..., m

- GOAL

OPT[1]

- CASI BASE

OPT[i] = 0 SE i > m

- EQ BELLMAN

$$OPT[i] = \begin{cases} 0 & \text{SE } i > m \\ V_{i+1} + OPT[i+2] & \text{SE } i \leq m-1 \wedge i > m-2 \\ \max\{V_{i+1} + OPT[i+2],\ V_i + OPT[i+3]\} & \end{cases}$$

ho solo 2 Mon.

- COMPLESSITÀ = $O(m)$

Esercizio 3 [13 punti]

State progettando una vacanza intergalattica con la vostra astronave e dovete fare n viaggi. La vostra astronave può volare in quattro diverse modalità (A, B, C e D). Effettuare il viaggio i-esimo nella modalità $x \in \{A, B, C, D\}$ ha un costo $c_{i,x}$. Una volta scelta una modalità per un viaggio non è possibile cambiarla in volo. Però alla fine di ogni viaggio potete decidere di riconfigurare l'astronave e cambiare modalità. Il cambio di modalità ha un costo di riassetto che conoscete. In particolare, per ogni $x, y \in \{A, B, C, D\}$, sia $\delta_{x,y}$ il costo per passare dalla modalità x alla modalità y. Si assuma che $\delta_{x,x} = 0$ per ogni x. Per ragioni tecniche dovete partire per il primo viaggio con l'astronave in modalità A e arrivare a destinazione (dopo gli n viaggi) in modalità C o D.

Progettare un algoritmo di programmazione dinamica che calcola il costo di una vacanza di costo minimo.

ES, VIAGGIO INTERGALATTICO CON MODALITÀ (A, B, C, D)

- 1° VIAGGIO VA FATTO IN MODALITÀ A, SI FINISCE IN MODALITÀ C OD
- δ_{xy} COSTO CAMBIO MODALITÀ & C_{ix} COSTO VIAGGIO IN MOD. X.

- SOTTOPROBLEMA

OPT[i, M] = SCONE MINIMO (COSTO) SULLE PRIME i TRATTE FACENDO IL VIAGGIO i IN MODALITÀ M.

- GOAL

$\text{MIN}\{\text{OPT}[n, C], \text{OPT}[n, D]\}$

- CASI BASE

$\text{OPT}[1, M] = C_{1A}$ PER $i = 1 \wedge \forall M \in \{A, B, C, D\}$

$\text{OPT}[2, M] = \text{OPT}[1, A] + \delta_{AY} + C_{iY}$ $\forall Y \in \{A, B, C, D\} \wedge \forall M$

- EQ. BELLMAN, $\forall i > 2$

$$\text{OPT}[i, M] = \text{MAX}\{\text{OPT}[i-1, A] + \delta_{AM} + C_{iM},$$
$$\text{OPT}[i-1, B] + \delta_{BM} + C_{iM}$$
$$\text{OPT}[i-1, C] + \delta_{CM} + C_{iM}$$
$$\text{OPT}[i-1, D] + \delta_{DM} + C_{iM}\}$$

- OSS = $\delta_{AA} = \delta_{CC} = \dots = 0$
- COMPLESSITÀ

$O(4n) = O(n)$

Esercizio 3 [11 punti] A Tor Vergata sta per chiudere anche l'ultimo bar e gli studenti presto non sapranno più come procurarsi il pranzo. Così il prof. Gualà è stato incaricato dalla facoltà di assoldare una ditta che gestisce furgoni attrezzati alla vendita di panini.

I giorni in cui gli studenti rischiano di rimanere senza pranzo sono n, numerati da 1 ad n. La ditta ha a disposizione due furgoni, uno *rosso* e uno *giallo*. Il costo e le condizioni di noleggio dei due furgoni varia a seconda del furgone e del giorno del noleggio. In particolar modo, se si noleggia il furgone rosso nel giorno i, il furgone è a disposizione per tre giorni consecutivi, quindi i giorni i, $i+1$, e $i+2$ e il suo costo (complessivo per i tre giorni) è di r_i. Il furgone giallo invece viene noleggiato per due giorni alla volta e, se noleggiato il giorno i, costa complessivamente g_i e può essere usato i giorni i e $i+1$.

L'obiettivo del prof. Gualà è quello di sottoscrivere dei contratti di noleggio in modo tale da spendere il meno possibile garantendo che in ognuno degli n giorni ci sia almeno un furgone che sta vendendo panini agli studenti di Tor Vergata. Aiutate il prof. Gualà progettando un algoritmo di programmazione dinamica che calcoli la migliore soluzione possibile. Un esempio di istanza e soluzione è riportata in figura.

i	1	2	3	4	5	6	7
r_i	10	11	11	11	7	9	6
g_i	9	10	12	9	7	8	10

$$Opt = 10 + 9 + 7 = 26$$

Figura 1

ES 18/07/22, FURGONI ROSSO e GIALLO

i	1	2	3	4	5	6	7
r_i	10	11	11	11	7	9	6
g_i	9	10	12	9	7	8	10

OPT = 10 + 9 + 7 = 26

- SOTTOPROBLEMA

OPT[i, c] = COSTO MINIMO PER LA COPERTURA NEI GIORNI DA i, i+1, ..., m SAPENDO CHE NEL GIORNO i STO USANDO UN FURGONE DI COLORE c.

- GOAL =

$$\text{MIN}\{OPT[1,R], OPT[1,G]\}$$

- CASI BASE

$OPT[i,C] = 0$ SE $i > m$

- EQ. BELLMAN =

$$OPT(i,c) = \begin{cases} \text{MAX}\{OPT[i+1,R]+g_i, \\ \quad OPT[i+2,R]+g_i, \\ \quad OPT[i+2,G]+g_i\} & \text{SE } c=G \\ \\ \text{MAX}\{OPT[i+1,G]+r_i, \\ \quad OPT[i+2,G]+r_i, \\ \quad OPT[i+3,G]+r_i, \\ \quad OPT[i+3,R]+r_i\} & \text{SE } c=R \end{cases}$$

ALG (R, G, m)

$OPT[m+1,c] = OPT[m+2,c] = OPT[m+3,c] = 0 \quad \forall c \in \{G,R\}$

FOR $i = m$ TO 1 DO

FOR EACH $c \in \{G,R\}$

IF $c == G$ THEN

$OPT[i,c] = \text{MAX}\{OPT[i+1,R]+g_i, OPT[i+2,R]+g_i$
$OPT[i+2,G]+g_i\}$

ELSE

$OPT[i,c] = \text{MAX}\{OPT[i+1,G]+r_i,$
$OPT[i+2,G]+r_i,$
$OPT[i+3,G]+r_i,$
$OPT[i+3,R]+r_i,\}$

RETURN MIN$\{OPT[1,G],$
$OPT[1,R]\}$

- COMPLESSITÀ

$O(2m) = O(m)$ SPAZIO E TEMPO

Esercizio 3 [13 punti]

Vi chiedono di giocare al seguente gioco. Potete piazzare delle pedine su una scacchiera $1 \times n$. Avete a disposizione pedine dei seguenti tre colori, elencati dal più *chiaro* al più *scuro*: bianco, grigio e nero. Piazzare una pedina nella posizione i vi fa guadagnare b_i se la pedina è bianca, g_i se la pedina è grigia, n_i se è nera. Avete i seguenti vincoli:

- se piazzate due pedine adiacenti, diciamo in posizione i e $i+1$, allora la pedina in posizione i deve essere più chiara di quella in posizione $i+1$;
- se in una certa posizione i non avete piazzato pedine, allora nella posizione $i+1$ potete piazzare una pedina di un qualsiasi colore;
- la prima pedina da sinistra che piazzate deve essere bianca;

Progettare un algoritmo di programmazione dinamica che calcola il massimo punteggio ottenibile.

ESERCIZIO, SCACCHIERA 1Xn 3 COLORI

1) PRIMA PEDINA CHIARA

2) SE METTO 2 PEDINE DI FILA => i + CHIARA DI i+1

3) NON PIAZZO A i e i+1 QUALSIASI COSA

- SOTTOPROBLEMA

OPT[i, c] = MAX SCORE OTTENUTO SULLE CELLE DA i, i+1, ..., n, SAPENDO DI VOLER PIAZZARE UNA PEDINA DI COLORE c

- GOAL

OPT[1, c] CHIARO (BIANCO)

- CASI BASE

OPT[i, c] = 0 SE i > n

- EQ. BELLMAN

1) SE c = BIANCO

CASO 3, NON PIAZZO A i

$$OPT[i, B] = MAX\{MAX_c\{OPT[i+2, c] + c_{i+1}\}, OPT[i+1, G] + b_i, OPT[i+1, N] + b_i\}$$

CASO 2

2) SE c = GRIGIO

$\forall c \in \{B, G, N\}$

$$OPT[i, G] = MAX\{MAX_c\{OPT[i+2, c] + c_{i+1}\}, OPT[i+1, N] + g_i\}$$

3) SE c = NERO

$$OPT[i, N] = MAX_c\{OPT[i+2, c] + c_{i+1}\}$$

- COMPLESSITÀ

$O(cn) = O(3n) = O(n)$

Esercizio 3 [11 punti] I tuoi amici Alice e Bob ti hanno invitato a cena a casa loro e avete ordinato una teglia di pizza con gusti misti. La pizza arriva in un cartone rettangolare che contiene n pezzi di pizza già tagliati allineati in una lunga striscia. Non tutti i gusti ti piacciono allo stesso modo. Se mangi il pezzo i-esimo (immagina i pezzi numerati da sinistra a destra) godi g_i. Ora, quando prendi un pezzo di pizza, per questioni di praticità, puoi scegliere solo uno dei due pezzi esterni (quelli interni si porterebbero via troppa mozzarella dai pezzi adiacenti e i tuoi amici non sarebbero contenti). Tu sei ospite e sei invitato per primo a scegliere il primo pezzo da mangiare. Una volta scelto il pezzo, Alice e Bob si mangeranno i due più esterni fra quelli che restano, e poi toccherà ancora te scegliere. Se per esempio tu decidi di mangiare il pezzo numero 1 (quello più a sinistra), Alice e Bob mangeranno il numero 2 e il numero n. Mentre se scegli il pezzo più a destra, loro mangeranno il numero 1 e il numero $n-1$. E così via.

Progetta un algoritmo di programmazione dinamica che calcola il massimo godimento ottenibile, dove il godimento che ottieni da un sottoinsieme di pezzi $S \subseteq \{1, 2, \ldots, n\}$ è definito come

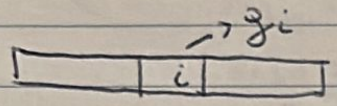
$$g(S) = \sum_{i \in S} g_i.$$

Si discuta la complessità temporale dell'algoritmo proposto.

ES 3 ESAME 13/06/24 PIZZA

- INPUT = PIZZA INTEGLIA con m PEZZI
- GOAL = MANGIARE PIZZA tc. $\forall i$ OGNI CELLA HA UN GODIMENTO g_i

$$G(S) = \sum_{i \in S} g_i$$

- 

- SOTTOPROBLEMA

OPT[i, j] = GODIMENTO MASSIMO OTTENIBILE SULLA PORZIONE DI PIZZA CHE VA DA $i, i+1, \dots, j$

- GOAL

OPT[1, m]

- CASO BASE

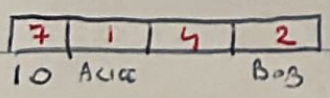
OPT[i, j] = 0 SE $i > j$

OPT[i, j] = g_i SE $i = j$

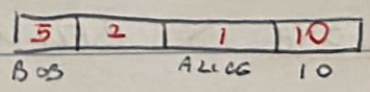
- EQ. BELLMAN

$$- \text{OPT}[i,j] = \begin{cases} 0 & \text{SE } i > j \\ g_i & \text{SE } i = j \\ \max\{g_i + \text{OPT}[i+2, j-1] \ (1), \\ \quad g_j + \text{OPT}[i+1, j-2] \ (2)\} & \text{ALTRIMENTI} \end{cases}$$

(1)



(2)



Esercizio 3 [13 punti] *Il signor Marche va a Venezia*

Il signor Marche si è recato in vacanza a Venezia e ha scoperto, con suo disappunto, quanto è cara la famosa città veneta. Ora si trova nella condizione di dover attraversare un canale in gondola e, guardando i prezzi sui volantini, gli è preso un colpo. Questo canale, che può essere percorso in un'unica direzione, passa per n diversi porticcioli, che per comodità numereremo da 1 a n. In ogni porticciolo è possibile imbarcarsi su una gondola e scendere lungo il canale verso porti successivi. Il signor Marche si trova nel porto 1 e deve raggiungere il porto n. Il costo di imbarco dipende dal porto di partenza e da quello di arrivo. Più precisamente, imbarcarsi nel porto i e scendere nel porto j costa $c(i, j)$ $(i > j)$. Progettate un algoritmo di programmazione dinamica che aiuti il signor Marche a raggiungere l'ultimo porto spendendo il meno possibile e appagando così, almeno in parte, il suo proverbiale attaccamento ai soldi. L'algoritmo, che deve calcolare la miglior soluzione prima del tramonto, deve avere complessità polinomiale nella dimensione dell'istanza.

ES, SIGNOR MARCHE IN GONDOLA

- M TRATTE, DA i A j COSTA c_{ij}

- SOTTOPROBLEMA

OPT[i] = COSTO MINIMO PER I VIAGGI DALLA TRATTA $i, i+1, \dots, M$

- GOAL

OPT[1]

- CASO BASE

OPT[i] = 0 SE $i > M$

- EQ. BELLMAN

$$OPT[i] = \min_j \{ c_{ij} + OPT[j+1] \} \quad \forall j = i+1, \dots, M$$

- COMPLESSITÀ

$O(mj) = O(m^2)$

Esercizio 3 [11 punti] Controlli uno scavatore robotico utilizzato per estrarre oro da una miniera. La miniera è divisa in n tratte disposte linearmente. Conosci la quantità di oro che è seppellita sotto ogni tratta. In particolare, sai che se scavi nella tratta i troverai v_i unità di oro. Lo scavatore ha una batteria di Δ unità, e inizialmente è posizionato sulla tratta 1 con la batteria completamente carica. I controlli dello scavatore ti permettono di eseguire le seguenti manovre:

- muovere lo scavatore in avanti di una o due posizioni: se lo scavatore si trova nella tratta i puoi spostarlo nella tratta $i+1$ o $i+2$. Questa manovra consuma s unità di batteria (indipendentemente se ti muovi di 1 o di 2);
- scavare: se lo scavatore è sulla tratta i questa manovra ti permette di estrarre le v_i unità di oro sotto la tratta. L'energia consumata dalla manovra è di t unità di batteria. Si assuma $t > s$.

Progettare un algoritmo di programmazione dinamica che calcoli il massimo numero di unità di oro estraibili dalla miniera. Si discuta la complessità temporale dell'algoritmo proposto.

ES 3 16/07/24 SCAVATRICI/ORO

- SOTTOPROBLEMA

OPT[i, b] = VALORE MAX DI ORO OTTENIBILE SULLE PRIME i TRATTE AVENDO A DISPOSIZIONE b QUANTITÀ DI BATTERIA RIMANENTE.

- GOAL

OPT[n, Δ]

- CASI BASE

OPT[i, b] = 0 SE i ≤ 0

- EQUAZIONE DI BELLMAN

$$
\text{-}OPT(i,b) = \begin{cases} 0 & \text{-SE } i \leq 0 \\ OPT[i-1, b] & \text{-SE } b < s \\ MAX\{OPT[i-1, b-s], OPT[i-2, b-s]\} & \text{-SE } b \geq s \wedge b < t \\ MAX\{v_i + OPT[i-1, b-s-t], v_i + OPT[i-2, b-s-t], \\ \quad OPT[i-1, b-s], OPT[i-2, b-s]\} & \end{cases}
$$

ALGORITMO (V, Δ, n)

```
FOR b = 0 TO Δ DO
    OPT[0, b] = 0
FOR i = 1 TO n DO
    FOR b = 0 TO Δ DO
        IF b < s THEN
            OPT[i, b] = OPT[i-1, b]
        ELSE IF ((b < t ∧ b ≥ s) ∨ b < t + s)
            OPT[i, b] = MAX{OPT[i-1, b-s], OPT[i-2, b-s]}
        ELSE
            OPT[i, b] = MAX{OPT[i-1, b-s], OPT[i-2, b-s],
                            OPT[i-1, b-s-t] + Vi,
                            OPT[i-2, b-s-t] + Vi}
RETURN OPT[n, Δ]
```

- COMPLESSITÀ =

$O(n\Delta)$ SPAZIO E TEMPO, $O(1)$ $\forall$ CELLA MATRICE.

$\Rightarrow$ COMPLESSITÀ PSEUDO POLINOMIALE

Esercizio 3 [13 punti]

Sonic si trova su una piattaforma composta da n caselle e deve raccogliere più anelli possibile. Le caselle, da sinistra a destra, sono numerate da 1 a n. Lui si trova nella casella 1 e il traguardo che deve superare è alla casella n. Ogni casella i ha un certo numero t_i di anelli a terra e un certo numero a_i di anelli in aria. Sonic ogni volta può fare due mosse: *spostarsi* a destra o *saltare* a destra. Se lui si trova nella casella i e si sposta a destra, allora finisce nella casella $i+1$ e raccoglie gli anelli a terra che si trovano in tale casella. Se invece decide si saltare a destra, allora si sposta nella casella $i+3$, raccoglie gli anelli che si trovano a terra nella casella $i+3$ e quelli che si trovano in aria nelle caselle $i+1$ e $i+2$. Progettare un algoritmo di programmazione dinamica che calcola in numero massimo di anelli che Sonic può raccogliere prima di tagliare il traguardo. Si assuma che non è necessario finire a terra nella casella n, ma è possibile tagliare il traguardo anche in volo.

ES, SONIC

- SOTTOPROBLEMA

OPT[i] = MAX SCORE OTTENUTO SULLE CELLE DA $i, i+1, \dots, n$

- GOAL

OPT[1]

- CASO BASE

OPT[i] = 0 SE $i \geq n$

- EQ. BELLMAN

$$OPT[i] = \begin{cases} 0 & \text{SE } i \geq n \\ \max\{OPT[i+1] + t_{i+1}, OPT[i+3] + t_{i+3} + a_{i+1} + a_{i+2}\} & \end{cases}$$

- COMPLESSITÀ

$O(n)$

Si vuole dotare una pista ciclabile di un buon servizio wifi. Per fare questo si devono installare dei ripetitori wireless. La pista ciclabile è lunga n tratte. Il costo di installazione di un ripetitore non è uniforme e dipende dalla tratta in cui si installa il ripetitore e dal tipo del ripetitore. In particolare, ci sono due tipi di ripetitori, uno di tipo *high* (H) e uno di tipo *low* (L).

Se nella tratta i si installa:
- un ripetitore di tipo L, il ripetitore è in grado fornire il servizio alle tratte i e $i+1$. Il costo di installazione è L_i
- un ripetitore di tipo H, il ripetitore è in grado fornire il servizio alle tratte i, $i+1$ e $i+2$. Il costo di installazione è $H_i \geq L_i$

Progettare un algoritmo di programmazione dinamica che calcoli una soluzione di costo totale minimo che fornisce il servizio a tutte le tratte.

ESERCIZIO WI-FI

2 RIPETITORI, HIGH e LOW

1) HIGH = $i, i+1, i+2$ COPERTE e COSTA $H_i \geq L_i$

2) LOW = $i, i+1$ COPRE e COSTA L_i

- ES

L_i =	2	1	3	4	4	7	1	→ COSTO 9
H_i =	5	8	3	5	4	8	5	

- SOTTOPROBLEMA

OPT[i] = COSTO MINIMO PER LA COPERTURA DA $i, i+1, \dots, m$

- GOAL

OPT[1]

- CASO BASE

OPT[i] = 0 SE $i > m$

- EQ. BELLMAN

$$-OPT[i] = \begin{cases} 0 \\ \min\{L_i + OPT[i+1],\ L_i + OPT[i+2],\ H_i + OPT[i+3]\} \end{cases}$$

- OSS = MANCANO CASI DI H_i, MA $L_i \leq H_i$ e QUINDI SONO COPERTI GIA TUTTI

- ALG (S)

```
OPT[m+1] = OPT[m+2] = 0
FOR i = m TO 1 DO
    OPT[i] = MIN{Li + OPT[i+1], Li + OPT[i+2], Hi + OPT[i+3]}
RETURN OPT[1]
```

COSTO $\Theta(m)$

Esercizio 3 [11 punti] State preparando la campagna elettorale in vista delle elezioni e volete pianificare dei comizi per aumentare la vostra chance di vincere. Davanti a voi avete n giorni ma gli esperti dicono che non tutti i giorni garantiscono la stessa visibilità. In particolare, sapete che fare un comizio il giorno i vi farà guadagnare v_i voti. Regole di fairness impongono che non potete fare comizi per due giorni consecutivi. Inoltre, i soldi a vostra disposizione per organizzare gli eventi sono limitati e il numero di comizi deve essere al più B.

Progettate un algoritmo di programmazione dinamica che calcoli il numero massimo di voti che potete guadagnare.

Una semplificazione: Un sottocaso più semplice del problema è lo scenario in cui siete schifosamente ricchi e quindi non avete vincoli sul numero di comizi che potete organizzare (ma il vincolo di fairness resta). Se non riuscite a progettare un algoritmo per il caso generale potete risolvere questo sottocaso (che però non dà diritto al punteggio pieno).

ES 3, CAMPAGNA ELETTORALE

- SE SFRUTTO IL GIORNO i GUADAGNO v_i E PER 2 GIORNI STO SENZA
- B MAX COMIZI

- SOTTOPROBLEMA:

$OPT[i,b]$ = NUMERO MAX VOTI OTTENIBILI SUI GIORNI DA $i, i+1, \dots, n$ CON DISPONIBILITÀ b TOT.

- GOAL

$OPT[1,B]$

- CASO BASE

$OPT[i,b] = 0$ SE $i \geq n$

- EQ. BELLMAN

$$OPT[i,b] = \begin{cases} 0 & \text{SE } i \geq n \\ OPT[i+1,b] & \text{SE } b \leq 0 \\ \max\{\underbrace{OPT[i+1,b]}_{\text{GIORNO } i \text{ NO COM.}}, \underbrace{OPT[i+3,b-1]}_{\text{GIORNO } i \text{ COMIZIO + 2 GIORNI NO}} + v_i\} & \end{cases}$$

- COMPLESSITÀ

$O(nB)$, PSEUDO POLINOMIALE

Esercizio 3 [13 punti]

Il signor Valter Bianchi, dopo aver fatto un bel gruzzoletto vendendo cristalli non proprio legali, ha deciso di diversificare la sua attività e si è messo a vendere della roba – anch'essa non proprio legale – che per comodità chiameremo stecca di cioccolata. La stecca di cioccolata può essere venduta tutta intera o può essere spezzata in segmenti più piccoli da vendere separatamente. La lunghezza della stecca di cioccolata è di L centimetri, con L intero. Si assuma che nello spezzare la stecca la lunghezza dei pezzi ottenuti (in centimetri) debba essere ancora un numero intero. Per esempio un pezzo lungo 3 centimetri può essere venduto così, spezzato in tre pezzi da 1 centimetro o in due pezzi: uno da 2 centimetri e l'altro da 1 centimetro (mentre non si possono fare due pezzi da mezzo centimetro e due centimetri e mezzo). Il guadagno che il signor Valter Bianchi riesce a fare se vende un pezzo lungo t centimetri è $G(t)$, $t = 1, 2, \ldots, L$. Progettare un algoritmo di programmazione dinamica che aiuti il signor Valter Bianchi a guadagnare il più possibile. La complessità temporale dell'algoritmo deve essere polinomiale in L.

ES CIOCCOLATA

- $G(t) = 1, \dots, m$ COSTO DEI PEZZI DA 1 A .. m

- SOTTOPROBLEMA:

  $OPT[i]$ = GUADAGNO MAX SULLA STECCA DA 1 A i

- GOAL

  $OPT[m]$

- CASO BASE

  $OPT[i] = 0$ SE $i \leq 0$

- EQ. BELLMAN

$$OPT[i] = \begin{cases} 0 & \text{SE } i \leq 0 \\ \max\limits_{1 \leq j \leq i} \{ G(i-j) + OPT[j] \} & \text{ALTRIMENTI} \end{cases}$$

- COSTO

  $O(mI) = O(m^2)$

ESERCIZIO SIGNOR MARCHE LAVORO

-> n GIORNI, 2 POSS.

1) LAVORO FORTE e X 2 GIORNI NON PUÒ LAV.

2) " PIANO e X 1 GIORNO " " " $(f_i \geq p_i)$

- SOTTOPROBLEMA

OPT[i] = MAX GUADAGNO SU I GIORNI DA $i, i+1, \ldots, n$

- GOAL

OPT[1]

- CASI BASE

OPT[i] = 0 SE $i > n$

- EQ. BELLMAN

$$OPT[i] = \begin{cases} 0 & \text{SE } i > n \\ \max\{OPT[i+2] + p_i, OPT[i+3] + f_i \end{cases}$$

- COMPLESSITÀ

$O(n)$

Esercizio 3 [13 punti]

Il prof. Gualà è in ufficio e deve necessariamente rimanere sveglio per le prossime n ore (indicizzate da 1 ad n). Per non addormentarsi, all'inizio di ogni ora, può scegliere di prendere un caffè. Esistono due tipi di caffè: (i) il caffè *ristretto* costa r euro e tiene sveglio per 2 ore (quella corrente e quella successiva), (ii) il caffè *lungo* costa ℓ euro, con $\ell > r$ e tiene svegli per 4 ore consecutive (quella corrente e le successive tre). Inoltre, in alcune ore, il prof. Gualà è impegnato in riunioni importanti (e noiose) e non può procurarsi il caffè. In particolare, poniamo $b_i = 1$ se è prevista una riunione nell'i-esima ora, e $b_i = 0$ altrimenti. Progettare un algoritmo di programmazione dinamica che, presi in input il numero di ore n, i costi dei caffè r ed ℓ ed i valori di b_i, calcoli quando e quali caffè deve comprare il prof. Gualà in modo da spendere il meno possibile senza addormentarsi.

ESERCIZIO, GUALÀ e CAFFÈ

- SOTTOPROBLEMA

OPT[i] = COSTO MINIMO DEI CAFFÈ SUI GIORNI DA $i, i+1, \dots n$

- GOAL

OPT[1]

- CASO BASE

OPT[i] = 0 SE $i > n$

OPT[i] = +∞ SE $b_i = 1$ (COSÌ SICURO NON VA IN OTTIMO)

- EQ. BELLMAN

$$OPT[i] = \begin{cases} 0 & \text{SE } i > n \\ \infty & \text{SE } b_i = 1 \\ \min\{OPT[i+4] + \ell_i, OPT[i+2] + p_i\} & \text{ALTRIMENTI.} \end{cases}$$

- COMPLESSITÀ

- n SOTTOPROBLEMI RISOLTI IN $O(1) = O(n)$

Esercizio 3 [13 punti] Il signor Valter Bianchi, fuorviato da una recente serie TV americana, ha deciso di fare soldi vendendo cristalli non proprio legali. Ha a disposizione una quantità iniziale Δ di cristalli, e conosce n potenziali compratori. Il compratore i vuole esattamente x_i cristalli ed è disposto a pagare p_i. Progettare un algoritmo di programmazione dinamica che calcoli il massimo guadagno ottenibile dal signore Valter Bianchi. L'algoritmo deve avere complessità temporale $O(n\Delta)$.

ES, VALTER BIANCHI & Δ CRISTALLI

- SOTTOPROBLEMA:

$OPT[i, c]$ = MAX SCONG SULLE VENDITE DEI PRIMI i CRISTALLI AVENDO A DISPOSIZIONE c CRISTALLI

- GOAL

$OPT[m, \Delta]$

- CASI BASE

$OPT[i, c] = 0$ SE $i \leq 0$

- EQ. BELLMAN:

$$OPT[i, c] = \begin{cases} 0 & \text{SE } i = 0 \\ OPT[i-1, c] & \text{SE } w < x_i \\ MAX\{OPT[i-1, c], OPT[i-1, c - x_i] + p_i\} & \end{cases}$$

- COMPLESSITÀ

$O(m\Delta)$