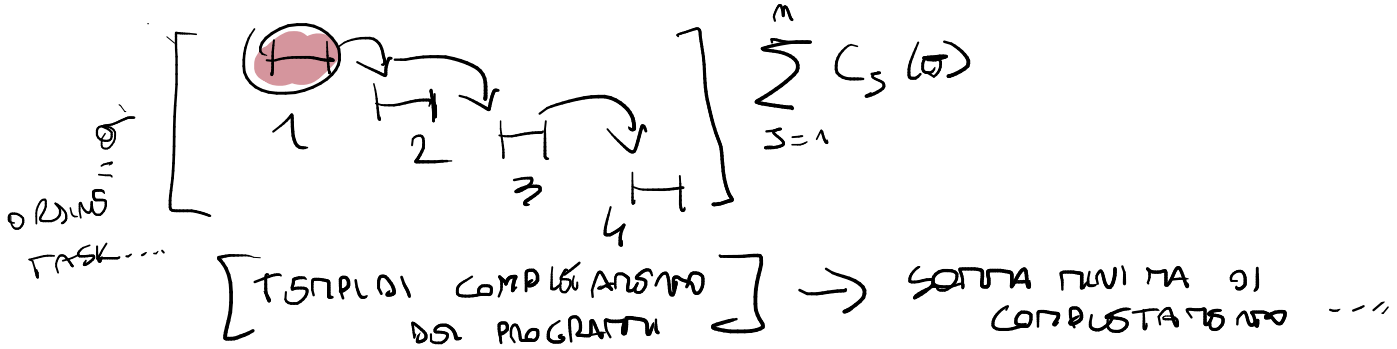
Esercizio 2 (10 punti) Abbiamo n programmi da eseguire sul nostro computer. Ogni programma j, dove $j \in \{1, 2, \ldots, n\}$, ha lunghezza ℓ_j, che rappresenta la quantità di tempo richiesta per la sua esecuzione. Dato un ordine di esecuzione $\sigma = j_1, j_2, \ldots, j_n$ dei programmi (cioè, una permutazione di $\{1, 2, \ldots, n\}$), il tempo di completamento $C_{j_i}(\sigma)$ del j_i-esimo programma è dato quindi dalle somma delle lunghezze dei programmi $j_1, j_2, \ldots, j_i$. L'obiettivo è trovare un ordine di esecuzione σ che minimizza la somma dei tempi di completamento di tutti i programmi, cioè $\sum_{j=1}^{n} C_j(\sigma)$.

(a) Dare un semplice algoritmo greedy per questo problema, e valutarne la complessità.

(b) Dimostrare la proprietà di scelta greedy dell'algoritmo del punto (a), cioè che esiste un ordine di esecuzione ottimo $\sigma^\star$ che contiene la scelta greedy.

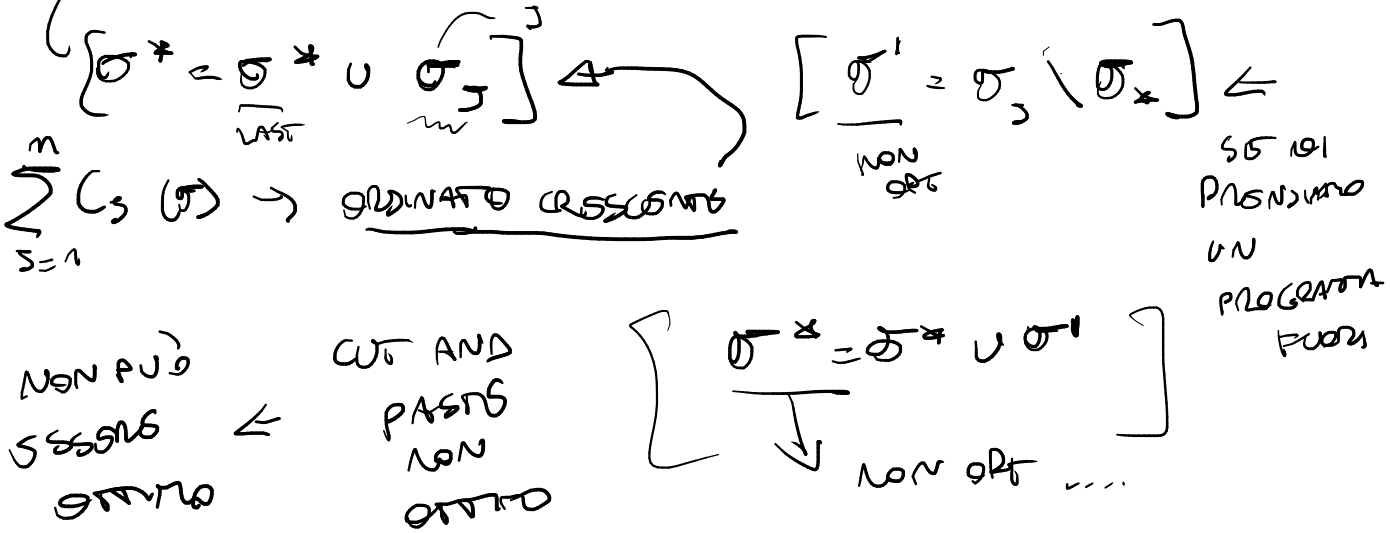



```
GREEDY(C, MAX) → MAX = Somma prevista di completamento

n = size(C)
last = 1 // indice ultima posizione scelta
SORT(C) // ordino crescente
OPT = {C_1} // primo = minore → OPT = greedy

while C_i + C_last ≤ MAX
        OPT = OPT U C_i;

return OPT
```

Esercizio 2 (10 punti) Abbiamo n programmi da eseguire sul nostro computer. Ogni programma j, dove $j \in \{1, 2, \ldots, n\}$, ha lunghezza ℓ_j, che rappresenta la quantità di tempo richiesta per la sua esecuzione. Dato un ordine di esecuzione $\sigma = j_1, j_2, \ldots, j_n$ dei programmi (cioè, una permutazione di $\{1, 2, \ldots, n\}$), il tempo di completamento $C_{j_i}(\sigma)$ del j_i-esimo programma è dato quindi dalle somma delle lunghezze dei programmi $j_1, j_2, \ldots, j_i$. L'obiettivo è trovare un ordine di esecuzione σ che minimizza la somma dei tempi di completamento di tutti i programmi, cioè $\sum_{j=1}^{n} C_j(\sigma)$.

~~(a)~~ Dare un semplice algoritmo greedy per questo problema, e valutarne la complessità.

(b) Dimostrare la proprietà di scelta greedy dell'algoritmo del punto (a), cioè che esiste un ordine di esecuzione ottimo $\sigma^\star$ che contiene la scelta greedy.



FORMALISMO → (SCAUSTA A)

1. Prendi una soluzione ottima (resta ottima perché l'insieme dei valori è ordinato)
2. Attacco la soluzione ottima che resta ottima sempre perché se ne attacco una fuori dall'ordine crolla tutto
3. That's it.

Esercizio 2 (9 punti) Lungo una strada ci sono, in vari punti, n parcheggi liberi e n auto. Un posteggiatore ha il compito di parcheggiare tutte le auto, e lo vuole fare minimizzando lo spostamento totale da fare. Formalmente, dati n valori reali $p_1, p_2, \ldots, p_n$ e altri n valori reali $a_1, a_2, \ldots, a_n$, che rappresentano le posizioni lungo la strada rispettivamente di parcheggi e auto, si richiede di assegnare ad ogni auto a_i un parcheggio $p_{h(i)}$ minimizzando la quantità

$$\sum_{i=1}^{n} |a_i - p_{h(i)}|.$$

L'idea di questo algoritmo, se fosse in codice, sarebbe simile a Metric Matching, quindi cerco di minimizzare la differenza partendo dalla fine (quindi, vedendo quante auto e quanti parcheggi ci sono) e verificando quale, a coppie, ha la minima differenza.

1. Si consideri il seguente algoritmo greedy. Si individui la coppia (auto, parcheggio) con la minima differenza. Si assegni quell'auto a quel parcheggio. Si ripeta con le auto e i parcheggi restanti fino a quando tutte le auto sono parcheggiate. Dimostrare che questo algoritmo non è corretto, esibendo un controesempio.

2. Si consideri il seguente algoritmo greedy. Si assuma che i valori $p_1, p_2, \ldots, p_n$ e $a_1, a_2, \ldots, a_n$ siano ordinati in modo non decrescente. Si produca l'assegnazione $(a_1, p_1), (a_2, p_2), \ldots, (a_n, p_n)$. Dimostrare la correttezza di questo algoritmo per il caso $n = 2$.

Questo significa che l'assegnazione parte dagli ultimi parcheggi e dalle ultime auto (quindi, massima somma al contrario significa minima differenza). Essendo elementi a coppia, si possono "mischiare" le assegnazioni come si vede sotto.

Soluzione:

1. Si consideri il seguente input:

$$p_1 = 5, p_2 = 10 \quad \text{e} \quad a_1 = 9, a_2 = 14.$$

L'algoritmo produce l'assegnazione $(a_1, p_2), (a_2, p_1)$, che ha costo $1 + 9 = 10$, mentre l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $4 + 4 = 8$.

2. Ci sono vari casi possibili:

Dal ragionamento detto, matematicamente, si vede che basta prendere un qualsiasi ordinamento tra le due auto e i due parcheggi di due generiche differenze e si esprime la somma in termini matematici (l'idea concreta è quella spiegata da me).

   (a) Caso $a_1 \leq p_1 \leq p_2 \leq a_2$
   - l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $p_1 - a_1 + a_2 - p_2 = (a_2 - a_1) - (p_2 - p_1)$
   - l'assegnazione $(a_1, p_2), (a_2, p_1)$ ha costo $p_2 - a_1 + a_2 - p_1 = (a_2 - a_1) + (p_2 - p_1)$; siccome $p_2 - p_1 \geq 0$, questa assegnazione ha costo non inferiore rispetto alla precedente

   (b) Caso $a_1 \leq p_1 \leq a_2 \leq p_2$
   - l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $p_1 - a_1 + p_2 - a_2 = (p_2 - a_1) - (a_2 - p_1)$
   - l'assegnazione $(a_1, p_2), (a_2, p_1)$ ha costo $p_2 - a_1 + a_2 - p_1 = (p_2 - a_1) + (a_2 - p_1)$; siccome $a_2 - p_1 \geq 0$, questa assegnazione ha costo non inferiore rispetto alla precedente

   (c) Caso $a_1 \leq a_2 \leq p_1 \leq p_2$
   - l'assegnazione $(a_1, p_1), (a_2, p_2)$ ha costo $p_1 - a_1 + p_2 - a_2 = (p_2 - a_1) + (p_1 - a_2)$
   - l'assegnazione $(a_1, p_2), (a_2, p_1)$ ha costo $p_2 - a_1 + p_1 - a_2 = (p_2 - a_1) + (p_1 - a_2)$, uguale a quello precedente

Tutti gli altri casi sono simmetrici e si dimostrano nella stessa maniera.

Esercizio 2 (9 punti) Data una stringa $X = \langle x_1, x_2, \ldots, x_n \rangle$, si consideri la seguente quantità $\ell(i,j)$, definita per $1 \le i \le j \le n$:

$$\ell(i,j) = \begin{cases} 1 & \text{se } i = j \\ 2 & \text{se } i = j-1 \\ 2 + \ell(i+1, j-1) & \text{se } (i < j-1) \text{ e } (x_i = x_j) \\ \sum_{k=i}^{j-1} (\ell(i,k) + \ell(k+1,j)) & \text{se } (i < j-1) \text{ e } (x_i \neq x_j). \end{cases}$$

(a) Scrivere una coppia di algoritmi INIT_L(X) e REC_L(X, i, j) per il calcolo memoizzato di $\ell(1,n)$.

(b) Determinarne la complessità *al caso migliore* $T_{\text{best}}(n)$, supponendo che le uniche operazioni di costo unitario e non nullo siano i confronti tra caratteri.

```
INIT_L(X)
if (n = 2) return 1;
if (n = 3) return 2;

for i = 1 to n
        l(i, i) = 1
for j = 1 to n
        l(j - 1, j) = 2

for i = 2 to n - 1
        for j = i + 1 to n - 1
                l(i, j) = 0

return REC_L(X, 0, n)

REC_L(X, i, j)
        if( l[i, j] = 0 )
                if (X[i] == X[j])
                        l[i, j] = 2 + l(i + 1, j - 1)
                else
                        for k = i to j - 1
                                l[i, j] = l(i, k) + l(k + 1, j)
        return X[n, n]
```

$$\ell(i,j) = \begin{cases} 2 + \ell(i+1, j-1) & \text{se } (i < j-1) \text{ e } (x_i = x_j) \\ \sum_{k=i}^{j-1} (\ell(i,k) + \ell(k+1,j)) & \text{se } (i < j-1) \text{ e } (x_i \neq x_j). \end{cases}$$

(a) Scrivere una coppia di algoritmi INIT_L(X) e REC_L(X, i, j) per il calcolo memoizzato di $\ell(1,n)$.

(b) Determinarne la complessità *al caso migliore* $T_{\text{best}}(n)$, supponendo che le uniche operazioni di costo unitario e non nullo siano i confronti tra caratteri.

(b)

$$\sum_{i=2}^{n-1} \sum_{j=i+1}^{n-1} 2 = \sum_{i=2}^{n-1} n-i-2 = \sum_{k=2}^{n} k = \frac{(n-2)(n-1)}{2}$$

$$\sum_{k=1}^{n} k \rightarrow \frac{n(n-1)}{2} \rightarrow \text{GAUSS}$$