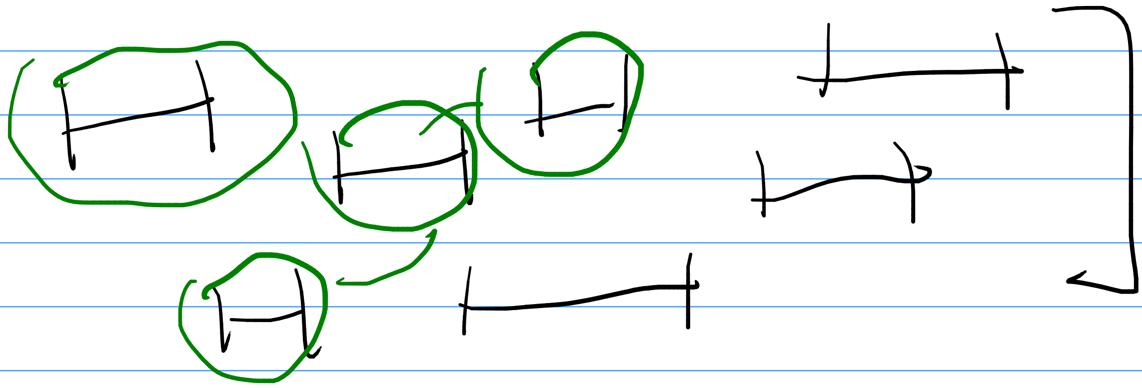
13/11

[GREEDY] → SELEZIONE ATTIVITÀ

PROBLEMA CLASSICO

$a_1 = \dots a_n$] attività



ATTIVITÀ

PROPR. DI SOTTOSTRUTTURA

OTTIMA → SCELTA CHE CONTINUA AD ESSERE BUONA

ACTIVITY SELECTION

GREEDY-SEL(S, f)

$n = S.length$
$A = \{a_1\}$
$last = 1$ // indice dell'ultima attività selezionata
for $m = 2$ **to** n
    if $s_m \geq f_{last}$
        $A = A \cup \{a_m\}$
        $last = m$
return A

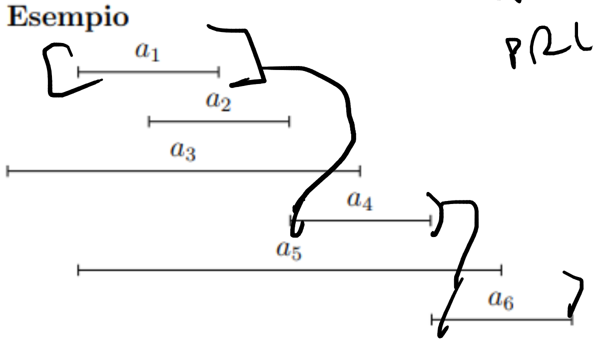
// SALVI SCELTA BUONA

± UNIONE BUONA

INIZIA PER PRIMA DOPO LA FINE

Esempio

PIVOT



$A = a_1$, $last = 1$ | $A = a_1, a_4$, $last = 4$ | $A = a_1, a_4, a_6$, $last = 6$

SOTTOSTR. OTTIMA] FORMALE

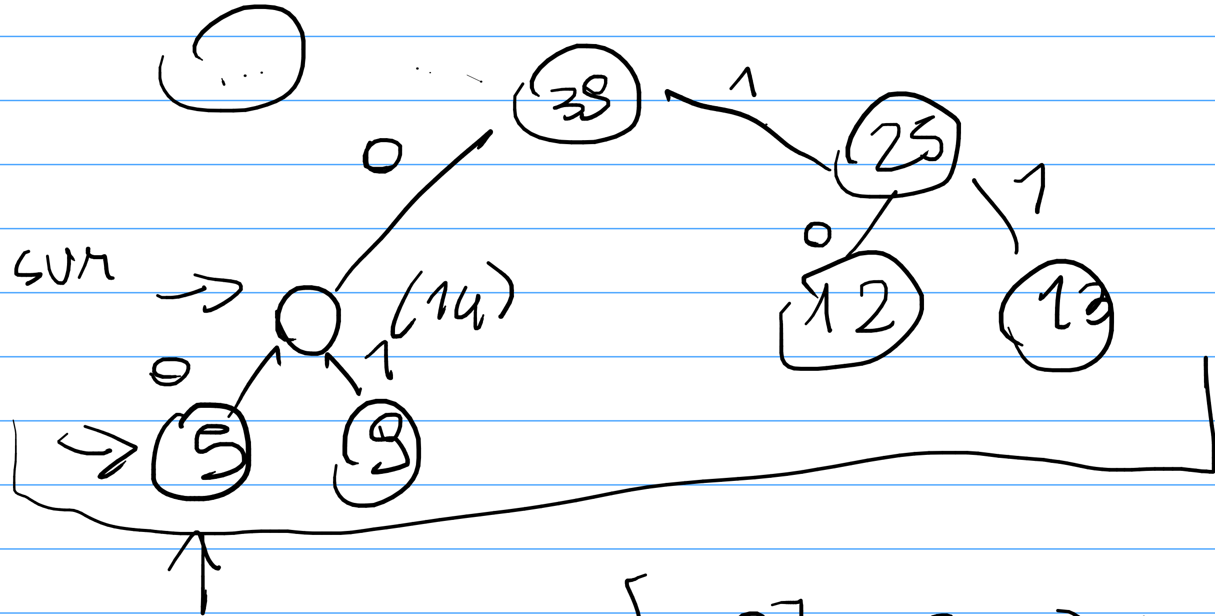
HUFFMAN → COMPRESSIONE DATI

ALBERO COME STRUTTURA DATI

FREQUENZA CON CUI CI SONO I CARATTERI

ORDINA PER FREQUENZA E
COSTRUISCI PARTENDO DALLA MINORE

	a	b	c	d	e	f
frequenza (%)	45	13	12	16	9	5



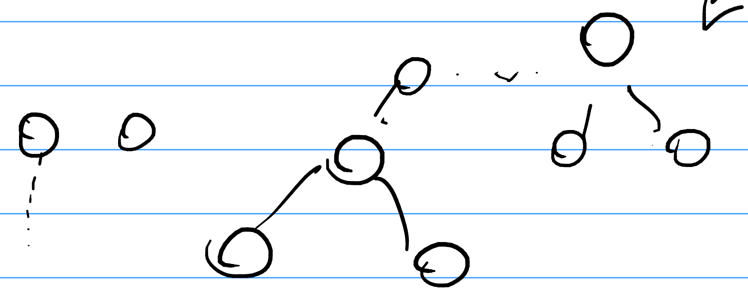
5 → F → [00] 12 → C → [0]

Domanda 44 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

a	b	c	d	e	f	g
3	8	7	12	6	23	21

Spiegare il processo di costruzione del codice.

OTTIMO



L'ALBERO NON È ALBERO DA PREFISSI

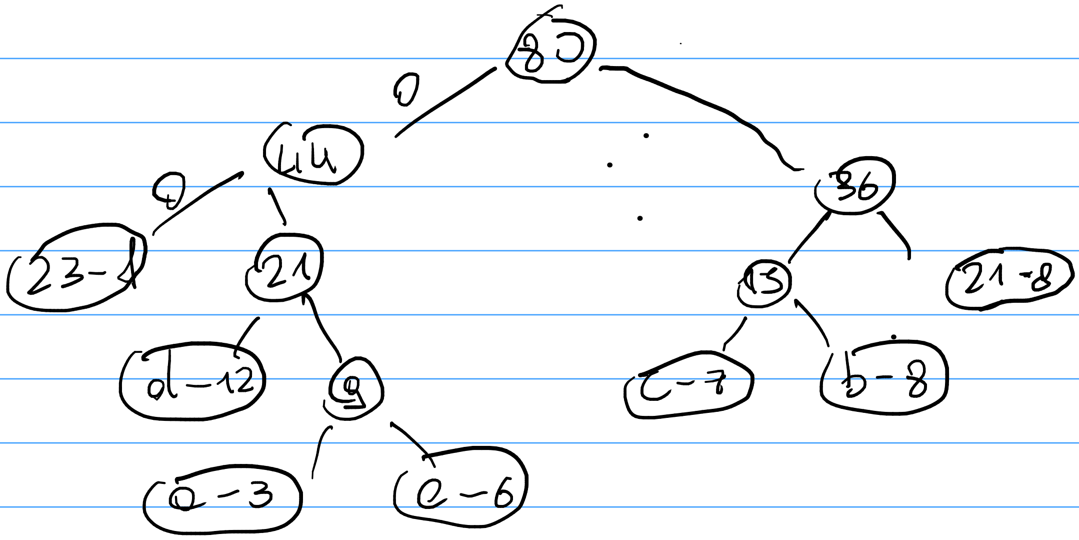
UN SOLO NODO SOMMA

Domanda 44 Indicare il codice prefisso ottenuto utilizzando l'algoritmo di Huffmann per l'alfabeto $\{a, b, c, d, e, f, g\}$, supponendo che ogni simbolo appaia con le seguenti frequenze.

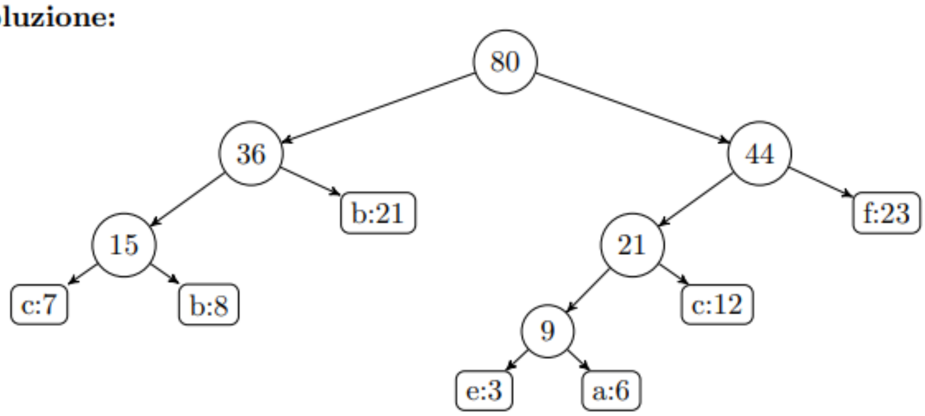
a	b	c	d	e	f	g
3	8	7	12	6	23	21

→ PRENDERE I <

Spiegare il processo di costruzione del codice.



Soluzione:



Domanda B (6 punti) Si consideri un insieme di 7 attività $a_i, 1 \leq i \leq 7$, caratterizzate dai seguenti vettori **s** e **f** di tempi di inizio e fine:

$$\mathbf{s} = (1, 4, 2, 3, 7, 8, 11) \qquad \mathbf{f} = (3, 6, 9, 10, 11, 12, 13).$$

Determinare l'insieme di massima cardinalità di attività mutuamente compatibili selezionato dall'algoritmo greedy GREEDY_SEL visto in classe. Motivare il risultato ottenuto descrivendo brevemente l'algoritmo.

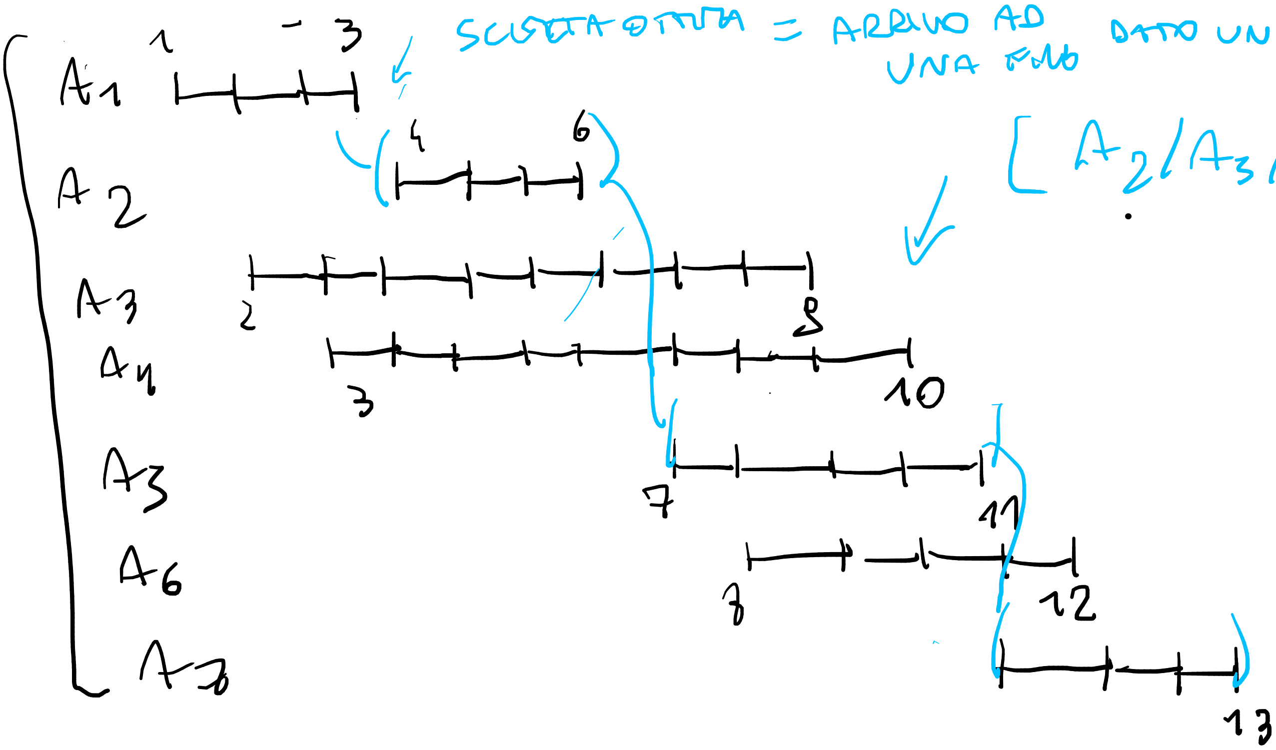
Domanda B (6 punti) Si consideri un insieme di 7 attività $a_i, 1 \leq i \leq 7$, caratterizzate dai seguenti vettori **s** e **f** di tempi di inizio e fine:

$$\mathbf{s} = (1, 4, 2, 3, 7, 8, 11) \qquad \mathbf{f} = (3, 6, 9, 10, 11, 12, 13).$$

Determinare l'insieme di massima cardinalità di attività mutuamente compatibili selezionato dall'algoritmo greedy GREEDY_SEL visto in classe. Motivare il risultato ottenuto descrivendo brevemente l'algoritmo.

SCELTA OTTIMA = ARRIVO AD UNA FINE DATO UN INIZIO

[A2/A3/A7]

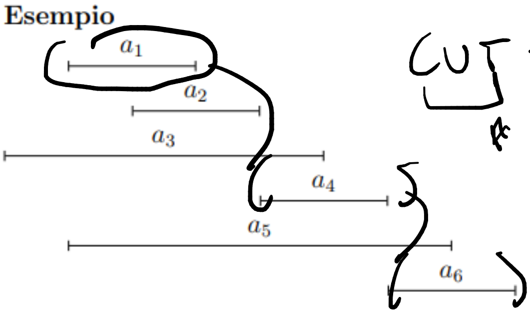


GREEDY-SEL(S, f)

$n = S.length$
$A = \{a_1\}$
$last = 1$ // indice dell'ultima attività selezionata
for $m = 2$ **to** n
  if $s_m \geq f_{last}$
    $A = A \cup \{a_m\}$
    $last = m$
return A

→ SCEGLI LA PRIMA E ATTACCHI TUTTO LO SCELTO BUONO

Esempio



$A = a_1$	$A = a_1, a_4$	$A = a_1, a_4, a_6$
$last = 1$	$last = 4$	$last = 6$

CUT-AND-PASTE

* METTI LA SEL. MIGLIORE

→ (a_1, a_4, a_6)

BIN-PACKING

Esercizio 2 (10 punti) Dato un insieme di n numeri reali positivi e distinti $S = \{a_i, a_2, \ldots, a_n\}$, con $0 < a_i < a_j < 1$ per $1 \le i < j \le n$, un *(2,1)-boxing* di S è una partizione $P = \{S_1, S_2, \ldots, S_k\}$ di S in k

ORDINATO

sottoinsiemi (cioè, $\bigcup_{j=1}^{k} S_j = S$ e $S_r \cap S_t = \emptyset, 1 \le r \ne t \le k$) che soddisfa inoltre i seguenti vincoli:

$$\rightarrow \quad |S_j| \le 2 \quad \text{e} \quad \sum_{a \in S_j} a \le 1, \quad 1 \le j \le k.$$

A = [0.1/0.2/0.3...]

In altre parole, ogni sottoinsieme contiene al più due valori la cui somma è al più uno. Dato S, si vuole determinare un (2,1)-boxing che minimizza il numero di sottoinsiemi della partizione.

$\underset{1}{\sqcup}\ \underset{2}{\sqcup} \ge \Sigma\, 1$

1. Scrivere il codice di un algoritmo greedy che restituisce un (2,1)-boxing ottimo in tempo lineare. (Suggerimento: si creino i sottoinsiemi in modo opportuno basandosi sulla sequenza ordinata.)
2. Si enunci la proprietà di scelta greedy per l'algoritmo sviluppato al punto precedente e la si dimostri, cioè si dimostri che esiste sempre una soluzione ottima che contiene la scelta greedy.

[0.1/0.9] → 1 → [2 − 1] BOXING

2 ELEMENTI CON $\Sigma = 1$

→ [$\underbrace{0.2/0.2/0.3/0.1/}_{5}$0.1] → 1

↓ 1

[0.1/0.3/0.4/0.2/0.8] ↓ 2−1 BOXING

↓ ≤ 1

DUE ELEMENTI LA CUI SOMMA È 1

2−1 BOXING (A)

1. L'idea è provare ad accoppiare il numero più piccolo (a_1) con quello più grande (a_n). Se la loro somma è al massimo 1, allora $S_1 = \{a_1, a_n\}$, altrimenti $S_1 = \{a_n\}$. Poi si procede analogamente sul sottoproblema $S \setminus S_1$.

N. EL. INSIEME →
SOL. →
FISSO L'INDICE →

```
(2,1)-BOXING(S)
n <- |S|
P <- empty_set
first <- 1
last <- n
while (first <= last)
   if (first < last) and a_first + a_last <= 1 then
      P <- P U {{a_first, a_last}}
      first <- first + 1
   else
      P <- P U {{a_last}}
   last <- last - 1
return P
```

Esempio:
1 2 3 4 5 6 7
→ P = (7, 6, 5, 4, 3, 2, 1)

Inizializziamo l'insieme, la partizione e gli estremi.

Si considera un ciclo per cui "first" è <= "last" (perché scansioniamo gli estremi, come detto)

Se l'estremo inf. è < dell'estremo sup.
non abbiamo ancora salvato nulla in P (non abbiamo estr. inf)
allora salvo in P sia l'estremo inf che l'estremo sup
e incremento first. Salvandolo una volta sola, so che la somma è sempr

Altrimenti,
salvo solo l'estremo superiore migliore, la cui somma
è sempre >= 1

0.1/0.8

0.2/0.8 → 1

0.2/0.9 → 0.2/0.8

MAKE SPAN

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.

$\sigma = J_1 J_2 \ldots J_n$ → [MAKESPAN-PROGRAMS (C)]

N = LENGTH (C)

[0.4 0.5 0.8 0.3] → [(0.3)(0.4) 0.5 0.8]

PROGRAMMI CON T. DI COMPLETAMENTO

SORT (C)

GREEDY = {C_1}

MERGESORT $O(n \log n)$

[σ = 0.9]

WHILE $J \leq n$

T. COMPL. MAX

[0.3] [0.4] → 0.8 ≤ ....

IF GREEDY + $C_J \leq \sigma$

GREEDY = GREEDY ∪ {C_J}

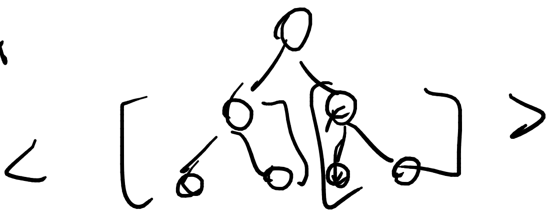
J++;

ORDINA PER LUNGH. CRESCENTE

RETURN C_n

$O(n \underbrace{\log(n)}_{\text{SCELTA}})$ → PERMUTAZIONE [. . . .]

$O(n)$



LCS

Esercizio 2 (11 punti) Una *longest common substring* di due stringhe X e Y è una sottostringa di X e di Y di lunghezza massima. Si vuole progettare un algoritmo efficiente per calcolare la lunghezza di una longest common substring. Per semplicità si assuma che entrambe le stringhe di input abbiano stessa lunghezza n.

(a) Qual è la complessità dell'algoritmo esaustivo (BRUTEFORCE) che analizza tutte le possibili sottostringhe comuni? → $O(n^2)$

(b) Assumendo di conoscere un algoritmo che determina se una stringa di m caratteri è sottostringa di un'altra stringa di n caratteri in tempo $O(m+n)$, come si può modificare l'algoritmo del punto precedente per renderlo più efficiente? [MAX = MAX + L Y]

(c) Progettare un algoritmo di programmazione dinamica più efficiente di quello del punto precedente. Sono richiesti relazione di ricorrenza sulle lunghezze (senza dimostrazione) e algoritmo bottom-up. (Suggerimento: considerare la lunghezza della longest common substring dei prefissi $X_i = \langle x_1, \ldots, x_i \rangle$ e $Y_j = \langle y_1, \ldots, y_j \rangle$ *che termina con* x_i *e* y_j, rispettivamente.)

[ABCDE X(n)
BCD Y(n)]

LCS = 3

$Y \leq X$

Y SUBSTRING
X SUPER STRING

(1) X → n, Y → n → $O(n)$ → STESSA LUNGHEZZA

PROG. DINAMICA

FOR I TO N
  FOR J TO N
    LCS [I, J] = 0

LCS = MAX (LCS) + SUBSTRING

. . . . .