Esercitazione 4
Algoritmi greedy

Problema 9 (es.2 appello 18/02/2016 modulo 2)

Nel museo *Tor VerLouvre* c'è un lungo corridoio rettilineo in cui sono esposti n quadri nelle posizioni $0 \leq q_1 < q_2 < q_3 < \cdots < q_n$. Il direttore del museo deve decidere quanti custodi assumere per poter sorvegliare tutti i quadri, sapendo che ogni custode deve avere una postazione fissa e che da quella posizione riesce a sorvegliare tutti i quadri che si trovano a distanza al più $d > 0$.

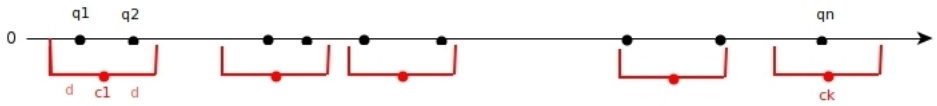
1. Descrivere un algoritmo che prende in input $q_1, \cdots, q_n$ e d, e in tempo $O(n)$ restituisce le posizioni dei custodi $c_1, \cdots, c_k$ in modo che tutti i quadri siano sorvegliati e il numero k di custodi sia il più piccolo possibile.

2. Il direttore del museo vorrebbe risparmiare qualche stipendio e assumere ancora meno custodi. Spiegategli in modo **rigoroso** perché non è possibile sorvegliare tutti i quadri con meno custodi di quanti ne usa il vostro algoritmo.

Soluzione

punto 1 Al fine di astrarre dal problema reale descritto nella traccia, possiamo modellare il corridoio con una semiretta ed immaginare ogni quadro come un punto su tale semiretta. Anche le posizioni dei custodi sono immaginabili come ulteriori punti sulla semiretta.

Una volta posizionato un generico custode in posizione c_j avremo che tutti i quadri posizionati nell' intervallo $[c_j - d, c_j + d]$ saranno coperti. Si vuole trovare l'insieme di punti $c_1, \cdots c_k$ che copra tutti i punti "quadro" $(q_1, \cdots, q_n)$ presenti sulla semiretta. Nella figura sottostante è riportata una rappresentazione del modello descritto.



Al fine di risolvere l'esercizio si vuole progettare un algoritmo greedy che trovi l'insieme di posizioni da assegnare ai custodi. Seguendo un approccio greedy vogliamo quindi trovare un criterio localmente ottimo con cui scegliere la posizione c_1 disinteressandoci delle successive posizioni da scegliere, poi determinare c_2 ancora ignorando le scelte future, e così via fino a quando tutti i quadri sono stati coperti. Ma quale criterio porta ad una soluzione ottima?

Proviamo ora a dare un primo criterio di scelta. Inizialmente nessun custode è stato posizionato e quindi banalmente nessun quadro è stato coperto, allora l'algoritmo si interesserà di coprire il primo quadro posizionando il primo custode davanti ad esso. Potenzialmente il primo custode potrebbe coprire più di un quadro, quindi l'algoritmo al fine di posizionare il secondo custode individuerà il primo quadro ancora scoperto e posizionerà il custode davanti ad esso. La regola appena descritta viene ripetuta fino a quando ci sono quadri ancora da coprire. Mostriamo ora come tale regola non porta ad una soluzione ottima, a tale scopo diamo un'istanza del problema sulla quale è facile verificare che la soluzione trovata non è ottima. Nell'istanza proposta ogni quadro (eccetto il primo) si trova a distanza $d + \epsilon$ dal precedente, come visibile in Figura 1 viene posizionato un custode per ogni quadro.

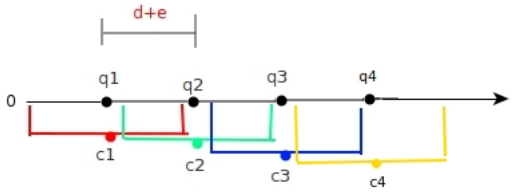


Figura 1:

Si può notare che semplicemente spostando il primo custode dalla posizione c_1 a $c_1 + \epsilon$, questo coprirebbe non più soltanto il primo quadro ma anche il secondo rendendo così superflo il custode posizionato in c_2. Cosa c'è che non va nel criterio proposto? Nella figura si può facilmente notare che il primo custode va a coprire inutilmente una porzione di semiretta a sinistra del punto q_1, mentre tra un quadro e l'altro c'è una porzione di semiretta coperta da 2 custodi, evidentemente risulta essere uno "spreco" far controllare la stessa porzione di corridoio a più di un custode. Queste semplici osservazioni ci suggeriscono il corretto modo in cui effettuare le scelte locali: in particolare si vorrebbe posizionare i custodi più a destra possibile non lasciando quadri scoperti alla loro sinistra. Qualitativamente, applicando tale criterio avremo che il campo visivo di un generico custode terminando più a destra possibile minimizzerà la porzione di corridoio ancora da controllare. Come posso ottenere tale proprietà? Mi basta posizionare il primo custode in modo che il suo campo visivo inizi esattamente dove è posizionato il primo quadro , quindi $c_1 = q_1 + d$ coprendo così tutti i quadri $q_i : 0 \leq q_i \leq q_1 + 2d$. Per posizionare il secondo custode si andrà a cercare il primo quadro ancora scoperto e sia q_j la sua posizione andremo a posizionare il custode in modo che il suo campo visivo inizi in q_j, quindi $c_2 = q_j + d$, saranno così coperti tutti i quadri $q_i : 0 \leq q_i \leq q_j + 2d$. Si procede seguendo questa regola fino a quando tutti i quadri sono stati coperti.

In Figura 2 è mostrato il risultato ottenuto applicando la regola descritta ad un'istanza del problema.

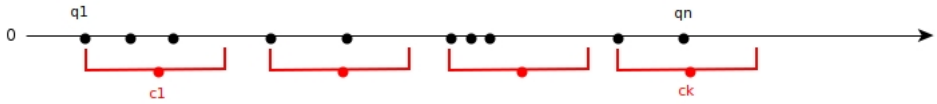


Figura 2:

Andiamo ora a definire un semplice pseudocodice che implementa il criterio di scelta locale appena proposto.

Algorithm 1: *posizionaCustodi*

Data: Q vettore ordinato contenente $q_1, \cdots, q_n$
Result: $C = \{c_1, \cdots, c_k\}$

```
1  begin
2      C = ∅
3      i = 1
4      last_position_covered = −1 // indica l'ultima posizione coperta dall'ultimo guardiano posizionato
5      for j = 1 to n do
6          if Q[j] > last_position_covered// il quadro non è stato ancora coperto then
7              c_i = Q[j] + d
8              last_position_covered = Q[j] + 2d
9              C = C ∪ {c_i}
10             i = i + 1
11     return C
```

L'algoritmo efffettua una semplice scansione del vettore dove per ogni elemento viene applicato un controllo effettuabile in tempo costante, da ciò deriva che la complessità dell'algoritmo è $O(n)$.

punto 2 Il secondo punto dell'esercizio ci sta chiedendo di dimostrare l'ottimalità dell'algoritmo trovato al punto precedente.

Sia $O = \{o_1, \cdots, o_m\}$ un insieme di posizioni calcolate da un qualche algoritmo ottimo e C l'insieme trovato dal nostro algoritmo, non possiamo mostrare che $C = O$ (in quanto potrebbero esserci più assegnazione di posizioni che coprono tutti i quadri usando lo stesso numero dei custodi), ma ci basta mostrare che $|C| = |O|$. Nella nostra dimostrazione useremo la tecnica nota come "greedy stays ahead".

L'idea è quella di comparare le soluzioni parziali dell'algoritmo greedy con le soluzioni parziali dell'algoritmo ottimo e mostrare che passo passo il greedy è non peggiore rispetto all'ottimo. Il metro su cui valutare il miglior algoritmo passo passo varia da caso a caso.

In questo esercizio per mostrare che le soluzioni parziali del nostro algoritmo sono non peggiori rispetto a quelle di un qualche algoritmo ottimo mostreremo che, data una generica scelta i-esima del greedy, questa ci fornisce una copertura che termina più a destra, o nella stessa posizione, rispetto alla copertura che ci fornisce l'i-esima scelta dell'algoritmo ottimo. Prima di formalizzare tale idea andiamo a definire la notazione che verrà utilizzata nella dimostrazione :

Siano $c_1, \cdots c_k$ le posizioni in C ordinate in modo crescente e siano $o_1 \cdots o_m$ le posizioni in O anch'esse ordinate in modo crescente , definiamo $f(c_i)$ come $c_i + d$ e ugualmente $f(o_i)$ come $o_i + d$, qualitativamente tale funzioni indicano il punto in cui termina il campo visivo di un custode posizionato in c_i o o_i.

Lemma 1. *$\forall r \leq k$ abbiamo $f(c_r) \geq f(o_r)$*

Dimostrazione. Per induzione su r:

- **caso base** $r = 1$

  L'algoritmo greedy per costruzione sceglie $c_1 = q_1 + d$ quindi $f(c_1) = q_1 + 2d$, qualunque algoritmo ottimo deve scegliere $o_1 \leq q_1 + d$ altrimenti il primo quadro rimarrebbe scoperto, allora abbiamo :

$$\begin{aligned} o_1 &\leq q_1 + d = c_1 \\ o_1 &\leq c_1 \\ o_1 + d &\leq c_1 + d \\ f(o_1) &\leq f(c_1) \end{aligned}$$

- **suppongo vero fino ad** $r-1$ **e dimostro per** r

  Per ipotesi induttiva abbiamo che $f(c_{r-1}) \geq f(o_{r-1})$.

  Distinguiamo due casi :

  1. esiste un quadro in posizione q_z tale che $f(o_{r-1}) < q_z \leq f(c_{r-1})$, ovvero un quadro che è stato già coperto dal greedy e non dall'ottimo. Allora $o_r \leq q_z + d$ mentre $c_r = q_{z+1} + d$ (il greedy va a coprire il prossimo quadro scoperto).
     Essendo $q_z < q_{z+1}$ otteniamo :

$$\begin{aligned} o_r \leq q_z + d < q_{z+1} + d &= c_r \\ o_r &< c_r \\ f(o_r) &< f(c_r) \end{aligned}$$

  2. se non esiste il quadro descritto al punto precedente, sia il greedy che l'ottimo avranno coperto tutti i quadri nell'intervallo $[0, f(c_{r-1})]$ e quindi entrambi gli algoritmi possono andare a coprire il prossimo quadro. In tale situazione è riutilizzabile l'argomentazione fatta per il caso base.

□

Dopo aver mostrato che passo passo l'algoritmo greedy è sempre non peggiore rispetto a un qualsiasi algoritmo ottimo, con una semplice dimostrazione per assurdo si riesce a mostrare l'ottimalità.

Teorema 1. *L'algoritmo greedy proposto è ottimo*

Dimostrazione. Supponiamo per assurdo che la soluzione C non sia di dimensione ottima, ovvero $m < k$ (Si ricordi $m = |O|, k = |C|$).

Applicando il lemma precedente con $r = m$ possiamo dire che $f(c_m) \geq f(o_m)$ Per costruzione se il nostro algoritmo ha incluso nella soluzione un elemento c_{m+1} doveva esistere un quadro in posizione $q_z : q_z > f(c_m)$ ovvero un quadro che ancora doveva coprire dopo aver posizionato l'$m - esimo$ custode (tale quadro si trova

necessariamente a destra di $f(c_m)$ in quanto è facile osservare che l'algoritmo prodotto per costruzione dopo aver posizionato un generico custode in posizione c_i non lascia quadri scoperti nell'intervallo $[0, f(c_i)]$), ma allora vale anche $q_z > f(o_m)$ e quindi l'algoritmo ottimo non avrebbe coperto tale quadro. Avendo ottenuto un assurdo deve necessariamente essere che $k = m$. □

Problema 10 (codice di Huffman)

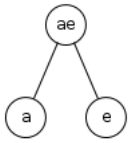
Sia dato un testo di N caratteri dell'alfabeto $\Sigma = a, b, c, d, e, f, g$. Le rispettive frequenze dei caratteri nel testo sono le seguenti : $F_a = 0.05; F_b = 0.20; F_c = 0.07; F_d = 0.25; F_e = 0.03; F_f = 0.06; F_g = 0.34$. Si costruisca il codice prefisso secondo l'algoritmo di Huffman. Quale è la lunghezza media del codice ottenuto?

Soluzione Prima di costruire il codice come richiesto, ricordiamo brevemente come l'algoritmo di Huffman lavora.

Dato un alfabeto e le relative frequenze, per costruire una codifica di Huffman si costruisce un albero binario, con i nodi foglia corrispondenti agli elementi dell'alfabeto. Gli archi di tali albero sono etichettati con 0 e 1 (0 per il figlio destro, 1 per il sinistro o viceversa) e la codifica di un generico elemento $x \in \Sigma$ è dato dalla sequenza di etichette lungo il cammino dalla radice alla foglia corrispondente ad x. Per costruire l'albero si selezionano i due simboli con frequenza minore e si costruisce un sottoalbero formato dai nodi corrispondenti a tali simboli con padre un meta-nodo. I due simboli sono ora rimossi dall'alfabeto, e viene aggiunto il meta-nodo, con frequenza pari alla somma delle frequenze dei nodi figli. La procedura si ripete fino a quando l'alfabeto sarà costituito da un solo simbolo, che sarà la radice dell'albero finale.

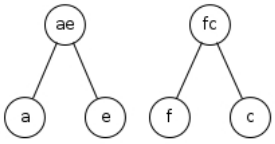
Vediamo ora passo passo come si comporta l'algoritmo sull'alfabeto attuale :

- **step 1** I simboli con frequenza minore sono: a, e; quindi vengono rimossi dall'alfabeto e rimpiazzati dal metanodo ae con $F_{ae} = 0.08$. Ricapitolando abbiamo $\Sigma = ae, b, c, d, f, g$ e $F_{ae} = 0.8, F_b = 0.20; F_c = 0.07; F_d = 0.25; F_f = 0.06; F_g = 0.34$ e avremo ottenuto il seguente albero :



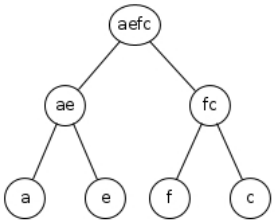
- **step 2**

  I caratteri con frequenza minore sono : f, c; una volta rimpiazzati con il corrispondente metanodo otteniamo $\Sigma = ae, b, fc, d, g$ con $F_{ae} = 0.08; F_b = 0.20; F_{fc} = 0.13; F_d = 0.25; F_g = 0.34$. I sottoalberi dopo il secondo step saranno :



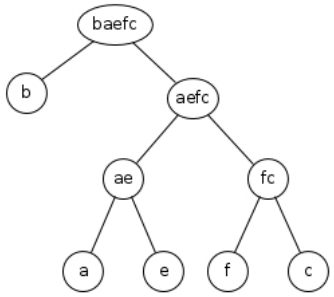
- **step 3**

  I caratteri con frequenza minore sono : ae, fc; una volta rimpiazzati con il corrispondente metanodo otteniamo $\Sigma = aefc, b, d, g$ con $F_{aefc} = 0.21; F_b = 0.20; F_d = 0.25; F_g = 0.34$. I sottoalberi dopo il terzo step saranno :



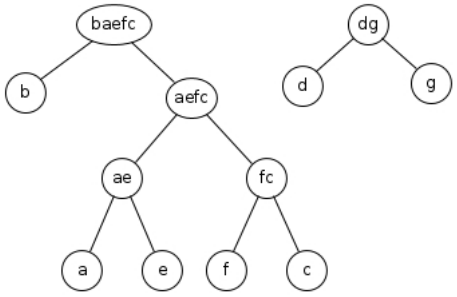
- **step 4**

  I caratteri con frequenza minore sono : $b, aefc$; una volta rimpiazzati con il corrispondente metanodo otteniamo $\Sigma = baefc, d, g$ con $F_{baefc} = 0.41; F_d = 0.25; F_g = 0.34$. I sottoalberi dopo il quarto step saranno :



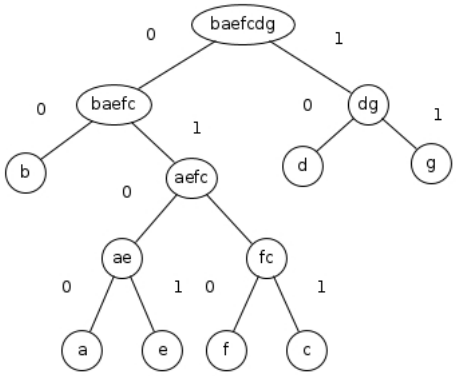
- **step 5**

  I caratteri con frequenza minore sono : d, g; una volta rimpiazzati con il corrispondente metanodo otteniamo $\Sigma = baefc, dg$ con $F_{baefc} = 0.41; F_{dg} = 0.59$. I sottoalberi dopo il quinto step saranno :



- **step 6**

  I caratteri con frequenza minore sono : $dg, baefc$; una volta rimpiazzati con il corrispondente metanodo otteniamo $\Sigma = baefcdg$ con $F_{baefcdg} = 1$. Avendo raggiunto $|\Sigma| = 1$ l'algoritmo termina e l'albero ottenuto con le relative etichette sugli archi è il seguente :



La codifica ottenuta sarà quindi la seguente : $c_a = 0100, c_b = 00, c_c = 0111, c_d = 10, c_e = 01010, c_f = 0110, c_g = 11$.

Andiamo ora a calcolare la lunghezza media del codice (ABL) : Ricordiamo che $ABL = \sum_x (F_x \cdot |c_x|)$ dove con $|c_x|$ si indica la lunghezza della codifica per il carattere x.

Nel nostro caso si ha : $ABL = 0.05 \cdot 4 + 0.20 \cdot 2 + 0.07 \cdot 4 + 0.25 \cdot 2 + 0.03 \cdot 4 + 0.06 \cdot 4 + 0.34 \cdot 2 = 2.42$.

Quindi usando il codice calcolato si ha in media un risparmio rispetto all' utilizzo di un codice a lunghezza fissa che, avrebbe richiesto $\lceil \log_2 |\Sigma| \rceil = \lceil \log_2 7 \rceil = 3$ bits per ogni carattere.