Massima sottosequenza comune (ricerca branch and bound)
Matteo Capparrucci 280278

Si consideri un insieme finito S di stringhe ed un intero K.
Determinare, se esiste, la stringa x di lunghezza maggiore (di k) che sia sottosequenza di ogni stringa s∈S. Si risolva il problema utilizzando una ricerca branch and bound.
Come funzione di valutazione si usi la lunghezza della soluzione parziale.

Per prima cosa analizziamo le strutture dati utilizzate, le stringhe del problema sono gestite inizialmente utilizzando una semplice lista di stringhe, successivamente da ogni elemento della lista di parole viene generata una nuova lista contenente tutte le sotto-stringhe della parola originale. I dati saranno quindi mantenuti all'interno di una lista di liste.

Le nuove sotto-stringhe così generate rappresentano però un particolare tipo di grafo che può essere definito in maniera esplicita attraverso la costruzione di un vero e proprio albero binario completo (con dei “figli” ridondanti) o attraverso un albero binario asimmetrico nel quale i nodi “fratelli” hanno rispettivamente: un figlio in comune con il fratello destro (se esiste) e un figlio in comune con il fratello sinistro (se esiste).
Nella pratica tale albero viene implementato mediante la funzione makeTree e con il tipo di dato 'a → tree, costruendo un albero sbilanciato nel quale la maggior parte dei nodi ha solo un figlio sinistro, mentre il nodo più a destra è l'unico ad avere anche un figlio destro che garantisce il livello di ramificazione necessario a rappresentare in modo ordinato le sottostringhe. L'altezza complessiva dell'albero così realizzato sarà pari alla lunghezza della parola utilizzata per generarlo.

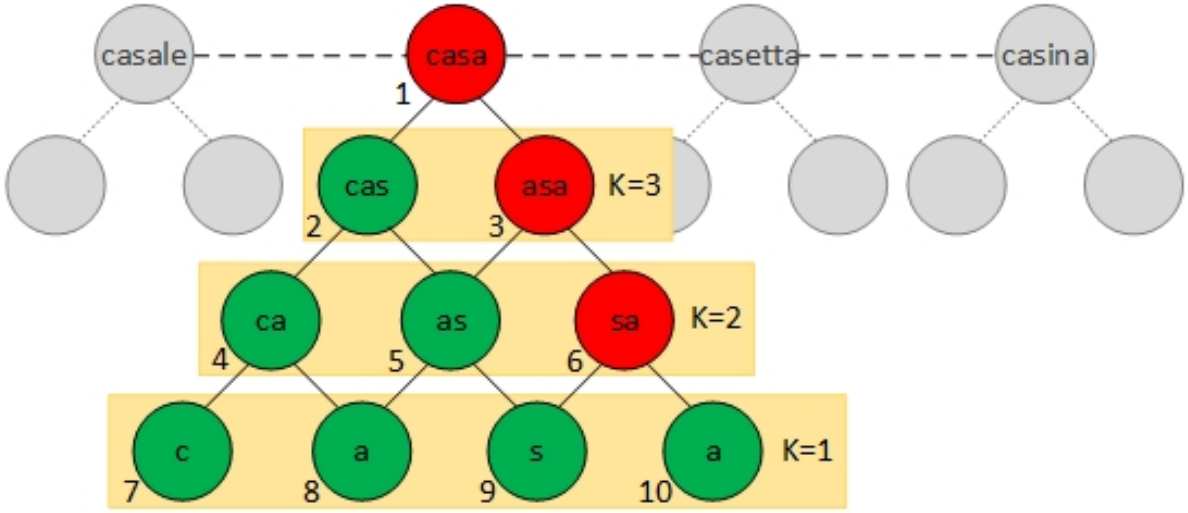
Per quanto riguarda l'algoritmo di ricerca, per prima cosa l'insieme di tutte le soluzioni possibili viene ridotto prendendo in esame come albero da visitare quello generato dalla stringa più corta, e quindi il più piccolo, basandoci sull'idea che non possa esistere una sotto-stringa comune di dimensione maggiore a quella rappresentata dalla parola di lunghezza minore. Dunque la ricerca viene eseguita ricorrendo su tutti i nodi dell'albero fino a raggiungere l'altezza k dalle foglie o le foglie stesse. Terminata la fase di ramificazione a partire dalle foglie ogni nodo testerà se stesso come soluzione parziale del problema (mediante la funzione confronta), in caso affermativo restituirà se stesso in caso contrario restituirà le soluzioni parziali generate concatenando le soluzioni parziali dei propri figli.

Dunque l'algoritmo cerca di implementare il metodo branch and bound utilizzando come versione “rilassata” del problema la ricerca nell'albero delle soluzioni più piccolo, ovvero quello costruito a partire dalla stringa di

dimensioni minori e come funzione di valutazione il controllo circa la lunghezza della soluzione corrente rispetto alla dimensione k.

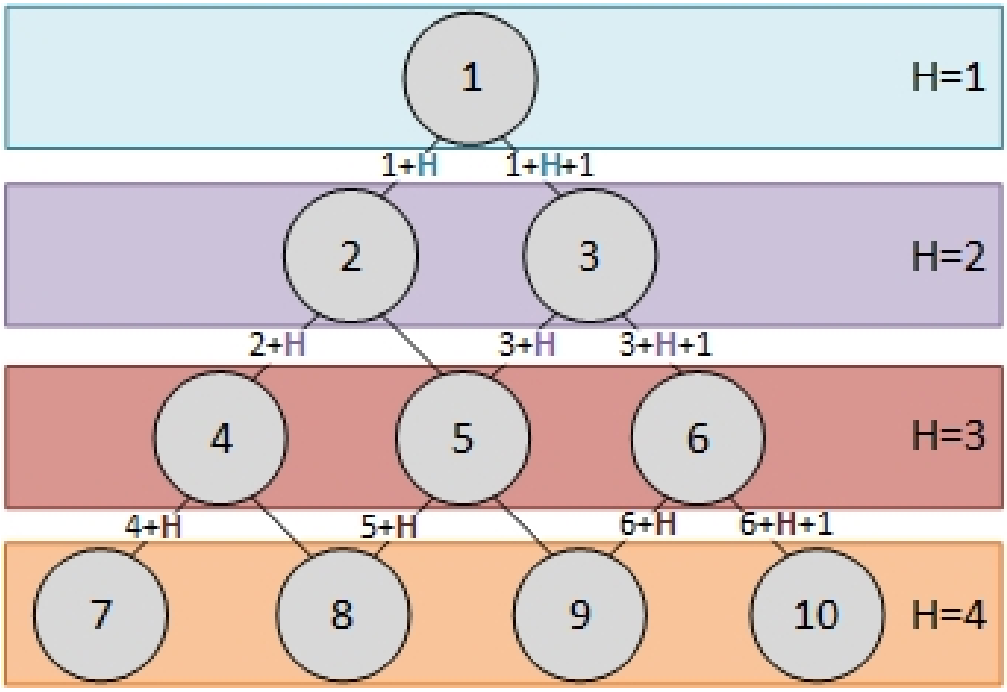
Esempio:
In questo esempio le stringhe in input sono rappresentate dalle parole: “casale”, “casa”,”casetta”,”casina”. L'albero logico da cui generare i confronti viene costruito a partire dalla stringa più breve, ovvero “casa”. A seconda del “k” scelto in fase di input l'algoritmo restituirà la stringa di dimensione maggiore confrontandola però con tutte le sotto-stringhe individuate come soluzione parziale fino a quel punto, ovvero:
\

- [] per k ≥ 4
- “cas” per k=3
- “cas” “ca” “as” per k=2
- “cas” “ca” “as” “c” “a” “s” “a” per k=1



Lista:	casa	cas	asa	ca	as	sa	c	a	s	a
Posizione:	1	2	3	4	5	6	7	8	9	10
Altezza:	1	2		3			4			

Nello specifico è possibile vedere come la funzione makeTree costruisca l'albero su cui compiere la ricerca, a partire dalla lista in input, utilizzando l'altezza corrente dell'albero per scegliere i nodi successori.
Nello specifico la funzione ad ogni chiamata ricorsiva la funzione ricorre sulla lista parziale ottenuta eliminando i primi h elementi (o h+1 nel caso del figlio destro) da quella corrente, assegna al nodo il valore della testa della sottolista corrente e incrementa il valore del parametro “h” di uno.


1
H=1
1+H
1+H+1
2
3
H=2
2+H
3+H
3+H+1
4
5
6
H=3
4+H
5+H
6+H
6+H+1
7
8
9
10
H=4