Appunti di
Elementi di Bioinformatica

A cura di:
Francesco Refolli
Matricola 865955

Anno Accademico 2022-2023

Chapter 1

Note sul Corso

todo: segnare delle note

Chapter 2

Pattern Matching

2.1 Introduzione

Pattern Matching Per **Pattern Matching** si intende trovare tutte le occorrenze di un pattern P di lunghezza m all'interno del testo T di lunghezza n.

Notazione Data una stringa o sequenza S, si identifica con $S[i : j]$ la sottostringa che contiene gli elementi da i a j (compreso).

2.2 Algoritmo Banale

Ragionamento L'algoritmo piu' semplice a cui si puo' pensare consiste nello scorrere linearmente il pattern P e provare per ogni posizione $i \in [1, n]$ la corrispondenza con una porzione di testo di uguale lunghezza.

Algorithm Confronta Stringhe

```
procedure ConfrontaStringhe(X, Y)
    for i ← 1 to n do
        if X[i] ≠ Y[i] then
            return false
        end if
    end for
    return true
end procedure
```

Algorithm Pattern Matching banale

```
procedure PMB(T, P)
    for i ← 1 to n do
        if ConfrontaStringhe(T[i : i + m − 1], P) then
            print(i)
        end if
    end for
end procedure
```

Procedura

Complessita' Come si puo' notare, sia PMB che $ConfrontaStringhe$ sono procedure la cui complessita' e' legata principalmente al singolo ciclo che contengono.
PMB contiene un ciclo di n iterazioni fisse, quindi il suo tempo nel caso medio sara' $T_{PMB} = \Theta(n)$.
ConfrontaStringhe al contrario contiene si un ciclo di m iterazioni, ma il numero di volte in cui saranno ripetuto il confronto dipendera' dalla similitudine delle due stringhe.
Nel caso medio sara' circa meta' dei caratteri, quindi $T_{ConfrontaStringhe} = \Theta(m/2) = \Theta(m)$.
Visto che PMB incorpora una chiamata a ConfrontaStringhe, la complessita' totale sara': $T(n, m) = \Theta(n * m)$.
Per quanto riguarda lo spazio, e' facilmente intuibile che sia $S(n) = \Theta(n + m)$.

2.3 Baeza-Yates-Gonnet

Approccio Bit-Parallel Quando si devono effettuare piu' operazioni dello stesso tipo poco costose e ripetitive e' possibile ridurre il problema ad azioni elementari che la CPU puo' processare in parallelo per ridurre il numero di passi da fare per completare un algoritmo. Per esempio se si devono sommare vettori di bit e' possibile assemblare una word che li contenga in modo da poi effettuare operazioni bit-wise (ovvero bit-a-bit, ogni bit non interferisce con quello successivo) per ottimizzare il calcolo. Si puo' applicare in certi casi anche agli algoritmi di pattern matching.

Ragionamento Si supponga di disporre di una matrice M di dimensione $m \cdot n$. Ogni cella e' riempita come segue:
Per ogni $i > 0$:

$$M[i][j] = \begin{cases} 1 & \text{sse P[:i] = T[j - i + 1 : j]} \\ 0 & \text{altrimenti} \end{cases}$$

Algorithm

```
procedure
end procedure
```

Procedura