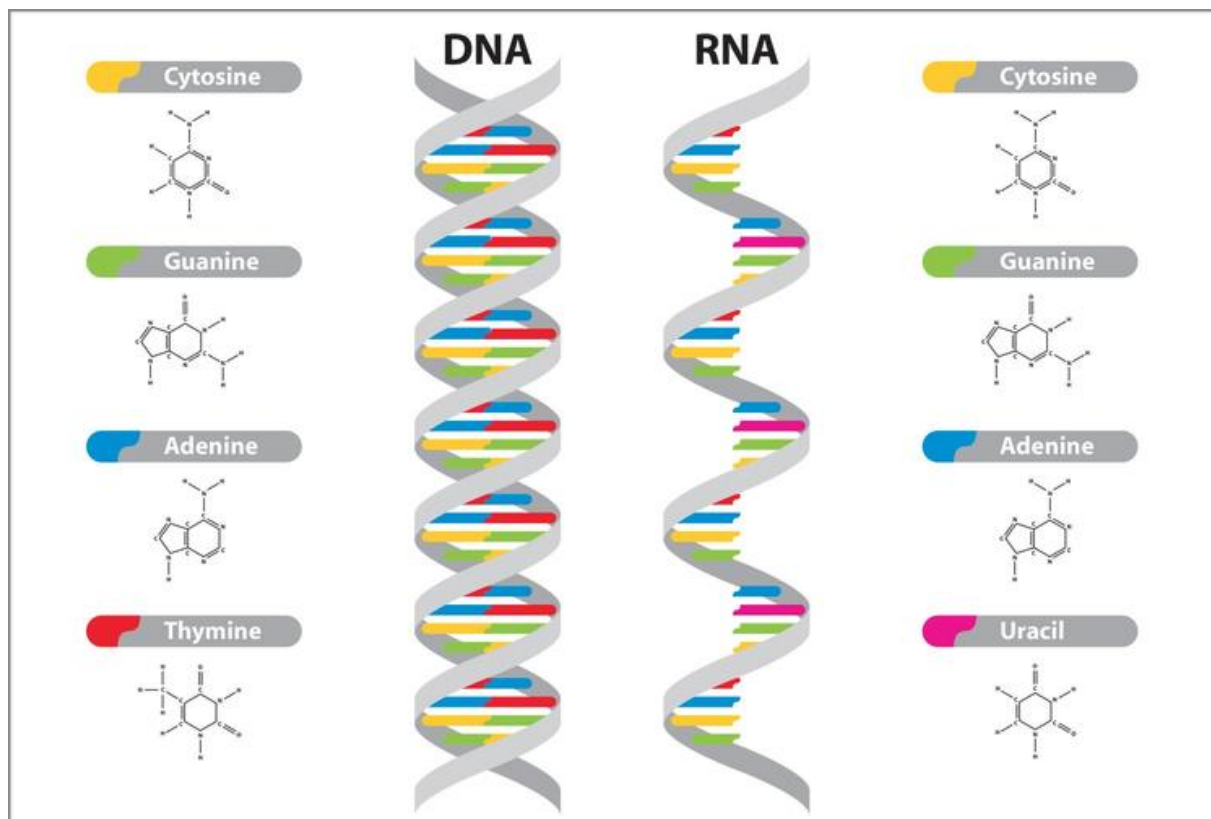


Relazione Laboratorio di Algoritmi e Strutture Dati

Interactive Nussinov Folder



Edoardo Papa
Anno Accademico 2018-2019

*Corso di Laurea in Informatica
Università di Camerino
Prof. Luca Tesei*

Relazione Laboratorio di Algoritmi e Strutture Dati

Algoritmo di Nussinov

L'algoritmo di Nussinov è un algoritmo che sfrutta l'approccio alla programmazione dinamica, il quale cerca di predire la struttura secondaria dell'RNA, senza pseudonodi, tramite la seguente ricorrenza:

$$N_{ij} = \max \begin{cases} N_{ij-1} \\ \max_{\substack{i \leq k < j \\ S_k, S_j \text{ complementary}}} N_{ik-1} + N_{k+1j-1} + 1 \end{cases}$$

A seguire, andremo ad analizzare com'è stato implementato l'algoritmo tramite l'utilizzo di questa tecnica.

```
@Override
public void fold() {

    // NON ESISTE NESSUNA SEQUENZA
    if(this.primarySequence == null)
    {
        throw new NullPointerException("ERRORE : non si puo eseguire il fold senza alcun dato inserito.");
    }

    //CASO BASE
    for(int i = 1; i <= len; i++)
    {
        this.matrice[i][i] = 0;
        this.matrice[i][i-1] = 0;
    }

    //CASO RICORSIVO
    // CICLO PRINCIPALE CHE CONTROLLA DA H = 1 ----> LEN
    for (int h = 1; h < len; h++)
    {
        // RIEMPIAMO LA DIAGONALE [i][j]=i+h
        for (int i = 1; i <= len - h; i++)
        {
            // J = COLONNA
            // I = RIGHA
            // H = DIAGONALE
            int j = i + h;
            for (int k = i; k < j; k++)
            {
                int v=-1;
                // VERIFICO CHE I NUCLEOTIDI IN POSIZIONE (K - 1) E (J - 1) POSSANO FORMARE UNA COPPIA AMMISSIBILE
                if(isPaired(this.primarySequence.charAt(k - 1) , this.primarySequence.charAt(j - 1)) == 1)
                {
                    // CALCOLO IL MASSIMO
                    v = this.matrice[i][k - 1] + this.matrice[k + 1][j - 1] + 1;
                    // VERIFICO SE IL MASSIMO TROVATO E' MAGGIORE DEL MASSIMO CHE SI TROVA NELLA CASELLA [I][J - 1]
                    if (v > this.matrice[i][j - 1])
                    {
                        // INSERISCO IL MASSIMO NELLA POSIZIONE [I][J]
                        this.matrice[i][j] = v;
                    }
                }
                else
                {
                    // I NUCLEOTIDI IN POSIZIONE (K - 1) E (J - 1) NON FORMANO UNA COPPIA AMMISSIBILE, QUINDI
                    // VERIFICO SE L'ELEMENTO IN POSIZIONE [I][J] E' MINORE DELL'ELEMENTO IN [I][J - 1]
                    if(this.matrice[i][j] < this.matrice[i][j-1])
                    {
                        // INSERISCO IL MASSIMO CHE SI TROVA NELLA CASELLA [I][J - 1] DELLA CASELLA ADIACENTE IN POSIZIONE [I][J]
                        this.matrice[i][j] = this.matrice[i][j-1];
                    }
                }
            }
        }
    }

    this.folded = true;
}
```

La programmazione dinamica si usa nei casi in cui esiste una definizione ricorsiva del problema, ma la trasformazione diretta di tale algoritmo genera un programma di complessità esponenziale a causa del calcolo ripetuto sugli stessi sottoinsiemi di dati, da parte delle diverse chiamate ricorsive.

La prima cosa che andremo a fare per la realizzazione del nostro algoritmo è il passo base della nostra ricorrenza, ovvero l'inizializzazione della matrice di Nussinov (per l'inizializzazione della matrice si intende porre a zero le due diagonal, la prima diagonale è la diagonale che parte da [1][0] e la seconda da [1][1]). Successivamente andremo a svolgere il cuore dell'algoritmo, ovvero il caso ricorsivo, con il quale potremo vedere come è stata implementata la programmazione dinamica.

Come possiamo vedere abbiamo tre cicli "for" annidati. Il primo for con l'indice H rappresenta la diagonale della matrice. Ad ogni incremento di H ci sposteremo a riempire la diagonale successiva. Con il secondo for otteniamo l'indice I che rappresenta le righe della matrice e l'indice J rappresenta le colonne. Attraverso l'ultimo for, con indice K che deve essere compreso fra I e J, andremo a verificare le condizioni principali:

- La prima condizione che dovremo andare a vedere riguarda la posizione dei nucleotidi, cioè verificare se quelli presenti nella posizione (k - 1) e (j - 1) formano una coppia ammissibile. ATTENZIONE: andiamo a verificare i nucleotidi che si trovano in posizione k - 1 e j - 1 e non quelli in k e j poiché nella sequenza primaria il primo nucleotide si trova in posizione 0 e non in posizione 1.

Se formano una coppia ammissibile, ci ricaviamo il massimo sfruttando la ricorrenza mostrata precedentemente, in particolare :

$$\max_{\substack{i \leq k < j \\ S_k, S_j \text{ complementary}}} N_{ik-1} + N_{k+1j-1} + 1$$

Una volta trovato il massimo, lo si confronta con quello che si trova in [I][J-1]; se quest'ultimo è minore, allora il massimo verrà aggiunto in posizione [I][J].

- Se la prima condizione non dovesse essere verificata, andremo a vedere se l'elemento che si trova in [I][J] è minore dell'elemento che si trova in [I][J - 1]. Se così fosse, il massimo sarà l'elemento in posizione [I][J - 1] che


dovremo inserire nella posizione $[i][j]$. In questa maniera avremo sfruttato la ricorrenza mostrata precedentemente, in particolare : N_{ij-1}

A seguire, verrà rappresentata la matrice di Nussinov durante la fase precedentemente descritta, quindi quella del caso ricorsivo.

La sequenza primaria presa in ingresso è : "AUAU"



Inizializzazione della matrice :

0	1	2	3	4		
	A	U	A	U		
0	0				A	1
	0	0			U	2
		0	0		A	3
			0	0	U	4





0	1	2	3	4		
	A	U	A	U		
0	0	1			A	1
	0	0			U	2
		0	0		A	3
			0	0	U	4



$$N_{ij} = \max \begin{cases} N_{ij-1} \leftarrow \\ \max_{\substack{i \leq k < j \\ S_k, S_j \text{ complementary}}} N_{ik-1} + N_{k+1j-1} + 1 \leftarrow \end{cases}$$

						
0	1	2	3	4		
	A	U	A	U		
0	0	1			A	1
	0	0	1		U	2
		0	0		A	3
			0	0	U	4

Note: In the original image, a red arrow points from the value 1 at (row 4, col 3) to the value 0 at (row 4, col 2). A green arrow points from the value 0 at (row 4, col 2) to the value 0 at (row 5, col 2). Another green arrow points from the value 0 at (row 5, col 2) to the value 0 at (row 6, col 3).




						
0	1	2	3	4		
	A	U	A	U		
0	0	1			A	1
	0	0	1		U	2
		0	0	1	A	3
			0	0	U	4





Note: In the original image, a red arrow points from the value 1 at (row 5, col 4) to the value 0 at (row 5, col 3). A green arrow points from the value 0 at (row 5, col 3) to the value 0 at (row 6, col 3). Another green arrow points from the value 0 at (row 6, col 3) to the value 0 at (row 7, col 4).

						
0	1	2	3	4		
	A	U	A	U		
0	0	1	1		A	1
	0	0	1		U	2
		0	0	1	A	3
			0	0	U	4

Note: In the original image, a red arrow points from the value 1 at (row 4, col 3) to the value 1 at (row 4, col 2).

$$N_{ij} = \max \begin{cases} N_{ij-1} \quad \leftarrow \\ \max_{\substack{i \leq k < j \\ S_k, S_j \text{ complementary}}} N_{ik-1} + N_{k+1j-1} + 1 \quad \leftarrow \end{cases}$$

						
0	1	2	3	4		
	A	U	A	U		
0	0	1	1		A	1
	0	0	1	 1	U	2
		0	0	1	A	3
			0	0	U	4

						
0	1	2	3	4		
	A	U	A	U		
0	 0	1	1	 2	A	1
	0	0	1	1	U	2
		0	0	1	A	3
			0	0	U	4

L'algoritmo ora è terminato e nella casella [1][4] (in termini generali sarebbe [1][len]) troviamo il numero massimo di legami.

len: n. di nucleotidi presenti nella struttura primaria

$$N_{ij} = \max \left\{ \begin{array}{l} N_{ij-1} \quad \text{red arrow} \\ \max_{\substack{i \leq k < j \\ S_k, S_j \text{ complementary}}} N_{ik-1} + N_{k+1j-1} + 1 \quad \text{green arrow} \end{array} \right.$$

Se prendiamo ancora una volta come sequenza d'ingresso "AUAU", possiamo mostrare come lavora l'algoritmo di Nussinov, tenendo in considerazione gli indici dei tre for annidati.





Relazione Laboratorio di Algoritmi e Strutture Dati

Algoritmo TraceBack

L'algoritmo di traceback è un'algoritmo ricorsivo che, dopo aver eseguito l'algoritmo di Nussinov, utilizza la matrice generata e prende in ingresso gli indici I e J (inizialmente $I = 1$ e $J = n$, dove n è la lunghezza della sequenza primaria) per trovare i vari legami deboli.

```
private void traceBack( int i, int j){
    // VERIFICO CHE L'INDICE I NON SIA MAGGIORE O UGUALE ALL'INDICE J
    if(i>=j)
        return ;
    // CONTROLLO CHE LA RIGA (i+1)(j) SIA UGUALE A ALLA RIGA SOPRA
    if((this.matrice[i + 1][j] == this.matrice[i][j]))
    {
        // SE E' UGUALE, RIESEGUO IL TRACEBACK SULLA RIGA (i+1)
        this.traceBack(i + 1,j);
        return;
    }
    // VERIFICO CHE LA COLONNA PRECEDENTE SIA UGUALE ALLA COLONNA ATTUALE
    if((this.matrice[i][j - 1] == this.matrice[i][j]))
    {
        // SE E' UGUALE, RIESEGUO IL TRACEBACK SULLA COLONNA PRECEDENTE
        this.traceBack(i,j - 1);
        return;
    }
    // CONTROLLO SE IL VALORE ALL'INTERNO DELLA DIAGONALE SX DELLA POSIZIONE CORRENTE SOMMATO CON 1(SOLAMENTE SE LE COPPIE SONO AMMISSIBILI) E'
    // UGUALE AL VALORE NELLA POSIZIONE CORRENTE
    if(((this.matrice[i + 1][j - 1]) + this.isPaired(this.primarySequence.charAt(i - 1), this.primarySequence.charAt(j - 1))) == this.matrice[i][j])
    {
        // SE E' UGUALE, MI SALVO IL NODO E RIESEGUO IL TRACEBACK SULLA DIAGONALE SX DELLA POSIZIONE CORRENTE
        this.bondArray.add(new WeakBond(i, j));
        this.traceBack(i + 1,j - 1);
        return;
    }
}

for(int k = i + 1; k < j; k++)
{
    if((this.matrice[i][k] + this.matrice[k + 1][j]) == this.matrice[i][j])
    {
        this.traceBack(i, k);
        this.traceBack(k + 1,j);
        return;
    }
}

return;
```

L'algoritmo è diviso in 4 parti:

- La prima parte verifica se la riga $[I+1][J]$ della matrice è uguale alla riga precedente, quindi $[I][J]$. Se così fosse, eseguo nuovamente l'algoritmo di traceback($I + 1, J$).

- La seconda parte verifica se la colonna $[I][J+1]$ della matrice è uguale alla colonna precedente, quindi $[I][J]$. Se così fosse, eseguo l'algoritmo di $\text{traceback}(I, J+1)$.
- Nella terza parte sommo 1 al valore all'interno della diagonale sinistra $[I+1][J-1]$ (rispetto alla posizioni in cui ci troviamo), solo se $I-1$ e $J-1$ sono coppie ammissibili; altrimenti sommo 0.

Il risultato che ottengo lo devo confrontare con il valore presente all'interno della posizione corrente ($[I][J]$). Nel caso in cui fosse uguale, mi salvo il legame debole ed eseguo il traceback sulla diagonale sinistra rispetto alla posizione attuale ($[I+1][J-1]$).

- Nella quarta parte vado ad eseguire un ciclo for da $K=I \rightarrow J$ (J non compreso) e vado a vedere se la somma dei valori contenuti della posizione $[I][K]$ e $[K+1][J]$, è uguale al valore contenuto in $[I][J]$. Se così fosse, eseguo due traceback :

1) $\text{traceback}[I][K]$

2) $\text{traceback}[K+1][J]$.

Relazione Laboratorio di Algoritmi e Strutture Dati

isPseudoknotted

Questo metodo determina se la struttura secondaria contiene pseudonodi e verifica che non ci siano legami con estremi uguali tra loro. Nel caso in cui la struttura secondaria contenga pseudonodi o ci siano legami con estremi uguali, il valore di ritorno sarà TRUE, altrimenti FALSE.

```
public boolean isPseudoknotted() {
    if(this.bonds.isEmpty())
        return true;

    for (WeakBond dato : this.bonds)
    {
        int iPrimo = dato.getI();
        int jPrimo = dato.getJ();
        for (WeakBond temp : this.bonds)
        {
            int i = temp.getI();
            int j = temp.getJ();
            // SE MI TROVO IN PRESENZA DEL NODO CORRENTE( ovvero quello del ciclo for precedente) NON LO CONSIDERO.
            if(dato.equals(temp))
            {
                // NON FACCIO NULLA
            }
            // VEDO SE C'E' ALMENO UN LEGAME DEBOLE CHE HA UN ESTREMO UGUALE AD UN ALTRO LEGAME DEBOLE
            else if((iPrimo == i)||jPrimo == j)||iPrimo == j)||jPrimo == i))
                return true;
            //VERIFICO SE CI SONO PSEUDONODI
            else if(((iPrimo < i)&&(i < j)&&(j < jPrimo))||(((i < iPrimo)&&(iPrimo < jPrimo)&&(jPrimo < j)))||((jPrimo < i))||((j < iPrimo)))
            {
                // NON FACCIO NULLA , NON CI SONO SPEUDONODI.
            }
            else
                return true;    // HO TROVATO UN PSEUDONODO.
        }
    }
    return false;
}
```

Per l'implementazione di questo algoritmo ho creato due cicli for annidati, i quali mi permettono di scorrere e confrontare tutti i legami deboli. Come prima cosa entro dentro il primo for e mi salvo la posizione del primo elemento che utilizzerò per il confronto. Successivamente entro dentro il secondo for e scorro tutti i legami deboli, vedendo se ho almeno un legame debole che ha un estremo uguale a quello salvato nel primo for oppure se ho almeno un legame debole che si incrocia. Successivamente al termine del secondo for, verrà aggiornato l'elemento salvato per il confronto con il legame debole successivo.

Questo meccanismo si ripeterà fino al termine del primo for. Se durante questi cicli abbiamo almeno un legame debole con un estremo uguale ad un altro

oppure un legame debole che si incrocia, questo metodo ritornerà TRUE, altrimenti FALSE.

Relazione Laboratorio di Algoritmi e Strutture Dati

Grafici

L'algoritmo di Nussinov, con il calcolo della matrice in maniera bottom-up, permette di risolvere il problema in un tempo $O(n^3)$.

A seguire, si potranno visualizzare differenti tipi di grafici per la valutazione delle prestazioni dell'algoritmo, in particolare il confronto tra il grafico di N^2 e N^3 .

