Corso di laurea in Informatica
Progettazione d'algoritmi
Didattica blended

Esercizi di programmazione dinamica

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

ESERCIZIO 1

Data una matrice binaria di dimensioni $n \times n$ vogliamo verificare se nella matrice è possibile raggiungere la cella in basso a destra partendo da quella in alto a sinistra senza mai toccare celle che contengono il numero 1.

Si tenga conto che dalla generica cella (i, j) ci si può spostare solo nella cella in basso (vale a dire la cella $(i+i, j)$) o nella cella a destra (vale a dire la cella $(i, j+1)$).

Progettare un algoritmo che risolve il problema in tempo $O(n^2)$

Ad esempio: per la matrice A la risposta deve essere SI mentre per la matrice B la risposta deve essere NO.

$A =$

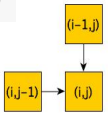
0	0	0	0	0	1
0	1	0	1	1	1
0	0	0	1	0	1
0	1	0	0	0	0
0	0	0	0	1	0
1	1	0	1	0	0

$B =$

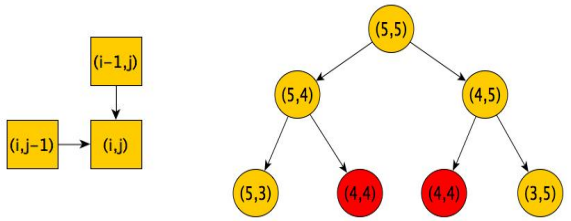
0	0	0	0	0	0
0	1	1	1	0	0
0	1	0	0	0	0
0	1	0	1	1	1
0	1	0	0	0	0
0	0	0	0	1	0

- A noi interessa sapere se la cella $(n-1, n-1)$ è raggiungibile a partire dalla cella $(0,0)$ nella matrice M.
- Progettiamo una funzione $es(i, j, M)$, $0 \leq i, j < n$, che restituisce 1 se la cella (i, j) di M è raggiungibile a partire dalla cella $(0,0)$, 0 altrimenti.
- La soluzione al nostro problema si avrà invocando $es1(n-1, n-1, M)$

```
def es1(i,j,M):
    if M[i][j]==1: return 0
    if i==0 and j==0: return 1
    if i==0: return es1(i,j-1,M)
    if j==0: return es1(i-1,j,M)
    return es1(i,j-1,M) or es1(i-1,j,M)
```



è presente "overlapping" di sottoproblemi come mostra ad esempio la porzione dell'albero di ricorsione in figura che viene fuori quando la matrice ha $n = 6$



MEMOIZZAZIONE

```
def es1M(M):
    n=len(M)
    T=[[-1 for _ in range(n)] for _ in range(n)]
    return es1Mb(n-1,n-1,M,T)
```

```
def es1Mb(i,j,M,T):
    if T[i][j]!=-1:
        return T[i][j]
    if M[i][j]==1:
        T[i][j]=0
    elif i==0 and j==0:
        T[i][j]= 1
    elif i==0:
        T[i][j]= es1Mb(i,j-1,M,T)
    elif j==0:
        T[i][j]= es1Mb(i-1,j,M,T)
    else:
        T[i][j]= es1Mb(i,j-1,M,T) or es1Mb(i-1,j,M,T)
    return T[i][j]
```

- L'inizializzazione della tabella T richiede $\Theta\left(n^2\right)$
- L'esecuzione di $es1(n-1,n-1,M,T)$ richiede $\Theta\left(n^2\right)$ (perché per ogni coppia(i,j), $0 \leq i,j < n$, il valore $es1Mb(i,j,M,T)$ verrà calcolato esattamente una volta)
- **La complessità dell'algoritmo è $\Theta\left(n^2\right)$**

Dall'approccio top-down all'approccio bottom-up

Utilizziamo una tabella T di dimensione $n \times n$

$T[i,j] = 1$ se dalla cella $(0,0)$ si può raggiungere la cella (i,j), 0 altrimenti

la soluzione la troveremo in $T[n-1,n-1]$

Possiamo calcolare i valori delle celle della tabella per righe e per colonne crescenti in base alla seguente regola:

$$T[i,j] = \begin{cases} 0 & \text{se } M[i,j] = 1 \\ 1 & \text{se } i = j = 0 \\ T[0,j-1] & \text{se } i = 0 \\ T[i-1,0] & \text{se } j = 0 \\ T[i-1,j] \ OR \ T[i,j-1] & \text{altrimenti} \end{cases}$$

Dall'approccio top-down all'approccio bottom-up

```
def es1(M):
    n=len(M)
    T=[[-1 for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            if M[i][j]==1: T[i][j]=0
            elif i==j==0: T[i][j]=1
            elif i==0: T[0][j]=T[0][j-1]
            elif j==0: T[i][0]=T[i-1][0]
            else: T[i][j]=T[i-1][j] or T[i][j-1]
    return T[n-1][n-1]
```

Nota che: i valori della matrice T vengono calcolati per righe e poi per colonne crescenti quindi al momento del calcolo della cella (i,j) i valori delle celle $(i-1,j)$ e $(i,j-1)$ sono disponibili.

La complessità dell'algoritmo è $\Theta\left(n^2\right)$

ESERCIZIO 2

Data una matrice di interi positivi e dimensione $n \times n$ vogliamo contare il numero di cammini di costo k che partono dalla cella in alto a sinistra e raggiungono la cella in basso a destra.

Il costo di un cammino è dato dalla somma dei valori delle celle toccate dal cammino inoltre, nel corso del cammino, dalla generica cella (i,j) ci si può spostare nella cella in basso (vale a dire la cella $(i+i,j)$) o nella cella a destra (vale a dire la cella $(i,j+1)$).

Progettare un algoritmo che risolve il problema in tempo $O(n^2k)$

Ad esempio: per la matrice

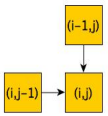
1	2	3
4	6	5
3	2	1

e $k = 12$ la risposta è 2. I cammini sono:

$1 \to 2 \to 6 \to 2 \to 1$

$1 \to 2 \to 3 \to 5 \to 1$

- A noi interessa sapere il numero di cammini della matrice M che partono dalla cella $(0,0)$, raggiungono la cella $(n-1, n-1)$ ed hanno costo k.
- Progettiamo una funzione $es2(i,j,t,M)$, $0 \le i,j < n$, $0 \le t \le k$ che restituisce il numero di cammini che partono dalla cella $(0,0)$, raggiungono la cella (i,j) ed hanno costo t.
- la soluzione al nostro problema si avrà invocando $es3(n-1, n-1, k, M)$
- In generale il numero di cammini che partono dalla cella $(0,0)$, raggiungono la cella (i,j) ed hanno costo t è dato dalla somma di queste due quantità:
  - il numero di cammini che partono dalla cella $(0,0)$, raggiungono la cella $(i, j-1)$ ed hanno costo $t - M[i,j]$
  - il numero di cammini che partono dalla cella $(0,0)$, raggiungono la cella $(i-1, j)$ ed hanno costo $t - M[i,j]$

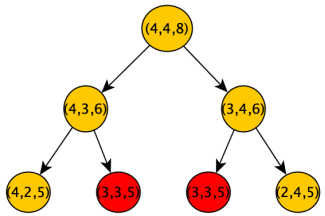


- la soluzione al nostro problema si avrà invocando $es3(n-1, n-1, k, M)$
- In generale il numero di cammini che partono dalla cella $(0,0)$, raggiungono la cella (i,j) ed hanno costo t è dato dalla somma di queste due quantità:
  - il numero di cammini che partono dalla cella $(0,0)$, raggiungono la cella $(i, j-1)$ ed hanno costo $t - M[i,j]$
  - il numero di cammini che partono dalla cella $(0,0)$, raggiungono la cella $(i-1, j)$ ed hanno costo $t - M[i,j]$

```
def es2(i,j,t):
    x=M[i][j]
    if t < x: return 0
    if i==j==0 and M[0][0]==t: return 1
    if i==j==0: return 0
    if i==0: return es2(0, j-1, t-x)
    if j==0: return es2(i-1, 0, t-x)
    return  es2(i, j-1, t-x) + es2(i-1, j, t-x)
```


è presente "overlapping" di sottoproblemi come mostra ad esempio la porzione dell'albero di ricorsione in figura che viene fuori per $k = 8$ sulla matrice A (che ha $n = 5$)

$$A = \begin{array}{|c|c|c|c|c|}\hline 3 & 6 & 2 & 5 & 6 \\ \hline 7 & 1 & 9 & 2 & 7 \\ \hline 4 & 8 & 5 & 2 & 2 \\ \hline 7 & 4 & 9 & 9 & 1 \\ \hline 1 & 8 & 3 & 1 & 2 \\ \hline \end{array}$$



MEMOIZZAZIONE

Utilizzo una tabella di dimensione $(k+1) \times n \times n$ dove nella cella $T[i,j,t]$ ho il valore di $es2Mb(i,j,t)$ se già calcolato, -1 altrimenti.

```
def es2M(M,k):
    n=len(M)
    T=[[[-1 for _ in range(k+1)] for _ in range(n)]for _ in range(n)]
    return es2Mb(n-1,n-1,k,M,T)
```

```
def es2Mb(i,j,t,M,T):
    if T[i][j][t]!=-1:
        return T[i][j][t]
    x=M[i][j]
    if t < x: T[i][j][t]= 0
    elif i==j==0 and M[0][0]==t: T[i][j][t]= 1
    elif i==j==0: T[i][j][t]= 0
    elif i==0: T[i][j][t]= es2Mb(0, j-1, t-x,M,T)
    elif j==0: T[i][j][t]= es2Mb(i-1, 0, t-x,M,T)
    else:  T[i][j][t]=es2Mb(i, j-1, t-x,M,T) + es2Mb(i-1, j, t-x,M,T)
    return T[i][j][t]
```

- L'inizializzazione della tabella T richiede $\Theta\left(n^2 \cdot k\right)$
- L'esecuzione di $es2(n-1, n-1, M, T)$ richiede $\Theta\left(n^2 \cdot k\right)$ (perché per ogni tripla (i,j,t), $0 \le i,j < n$, $0 \le t \le k$, il valore $es2Mb(i,j,t,M,T)$ verrà calcolato esattamente una volta)
- **La complessità dell'algoritmo è $\Theta\left(n^2 \cdot k\right)$**

Dall'approccio top-down all'approccio bottom-up

Utilizziamo una tabella T di dimensione $k \times n \times n$

$T[i,j,t] =$ il numero di cammini di costo t che dalla cella $(0,0)$ portano allla cella (i,j).

la soluzione la troveremo in $T[n-1, n-1, k]$

Possiamo calcolare i valori delle celle della tabella per righe e per colonne e per costo crescenti in base alla seguente regola dove con x indichiamo il valore $t - M[i,j]$:

$$T[i,j,t] = \begin{cases} 0 & \text{se } x < 0 \\ 1 & \text{se } i = j = 0 \text{ e } x = 0 \\ 0 & \text{se } i = j = 0 \\ T[0, j-1, x] & \text{se } i = 0 \\ T[i-1, 0, x] & \text{se } j = 0 \\ T[i-1, j, x] \ + \ T[i, j-1, x] & \text{altrimenti} \end{cases}$$

Dall'approccio top-down all'approccio bottom-up

```
def es2(M,k):
    n=len(M)
    T=[[[-1 for _ in range(k+1)] for _ in range(n)]for _ in range(n)]
    for t in range(k+1):
        for i in range(n):
            for j in range(n):
                x=t-M[i][j]
                if x<0: T[i][j][t]=0
                elif i==j==0 and x!=0: T[i][j][t]=0
                elif i==j==0: T[i][j][t]=1
                elif i==0: T[i][j][t]=T[i][j-1][x]
                elif j==0: T[i][j][t]=T[i-1][j][x]
                else: T[i][j][t]=T[i][j-1][x]+T[i-1][j][x]
    return T[n-1][n-1][k]
```

Nota che: i valori della matrice T vengono calcolati per i, j e t crescenti quindi al momento del calcolo della cella $T[i,j,t]$ i valori delle celle che servono al suo calcolo sono disponibili.

La complessità dell'algoritmo è $\Theta\left(n^2 \cdot k\right)$

ESERCIZIO 3

Data una matrice di dimensione $n \times n$ le cui celle sono numerate con numeri distinti che vanno da 1 a n^2, vogliamo trovare la massima lunghezza possibile per cammini che toccano celle con numerazione crescente e incremento di 1.

I cammini possono partire da una qualunque cella e, nel corso del cammino, dalla generica cella (i,j) ci si può spostare in una qualunque cella adiacente in orizzontale o verticale (vale a dire in una delle celle $(i,j+1), (i+1,j), (i,j-1), (i-1,j)$). La lunghezza di un cammino è data dal numero di nodi toccati dal cammino.

Progettare un algoritmo che risolve il problema in tempo $O(n^2)$

Ad esempio: per la matrice A la risposta è 1 mentre per la matrice B, grazie al cammino $2 \to 3 \to 4 \to 5 \to 6 \to 7$, la risposta è 6

$$A = \begin{array}{|c|c|c|}\hline 3 & 6 & 2 \\ \hline 7 & 1 & 9 \\ \hline 4 & 8 & 5 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|c|}\hline 9 & 7 & 6 \\ \hline 8 & 2 & 5 \\ \hline 1 & 3 & 4 \\ \hline \end{array}$$

- A noi interessa sapere la lunghezza massima per cammini che toccano celle con numerazione crescente e incremento unitario nella matrice M.
- Progettiamo una funzione $es3(i,j,M)$, $0 \le i,j < n$, che restituisce la lunghezza massima per il cammino che tocca celle con numerazione crescente e incremento unitario che parte dalla cella (i,j)
- la soluzione al nostro problema si avrà invocando $es4(i,j,M)$ sulle n^2 celle di M e tenendo conto del massimo.
- In generale il valore di $es4(i,j,M)$ è:
  - 1 se nessuna delle 4 celle adiacenti alla cella (i,j) ha il valore $M[i,j]+1$
  - $1 + es4(i',j')$ se c'è una cella (i',j') adiacente alla cella (i,j) con valore $M[i,j]+1$

- la soluzione al nostro problema si avrà invocando $es4(i,j,M)$ sulle n^2 celle di M e tenendo conto del massimo.
- In generale il valore di $es4(i,j,M)$ è:
  - 1 se nessuna delle 4 celle adiacenti alla cella (i,j) ha il valore $M[i,j]+1$
  - $1+es4(i',j')$ se c'è una cella (i',j') adiacente alla cella (i,j) con valore $M[i,j]+1$

```
def es3(M):
    n=len(M)
    m=0
    for i in range(n):
        for j in range(n):
            m=max(m, es3b(i,j,M))
    return m
```

```
def es3b(i,j,M):
    x =  M[i][j]+1
    if i!=0 and x == M[i-1][j]: return 1 + es3b(i-1,j,M)
    elif j!=len(M)-1 and x == M[i][j+1]: return 1 + es3b(i,j+1,M)
    elif i!=len(M)-1 and x == M[i+1][j]: return 1 + es3b(i+1,j,M)
    elif j!=0 and x == M[i][j-1]: return 1 + es3b(i,j-1,M)
    else: return 1
```


è presente "overlapping" di sottoproblemi come mostra ad esempio l'esecuzione dell'algoritmo sulla matrice A

$$A = \begin{array}{|c|c|c|c|} \hline 13 & 14 & 15 & 16 \\ \hline 12 & 11 & 10 & 9 \\ \hline 5 & 6 & 7 & 8 \\ \hline 4 & 3 & 2 & 1 \\ \hline \end{array}$$

MEMOIZZAZIONE

Utilizzo una tabella di dimensione $n \times n$ dove nella cella $T[i,j]$ ho il valore di $es3bM(i,j)$ se già calcolato, -1 altrimenti.

```
def es3M(M):
    n=len(M)
    T=[[-1 for _ in range(n)] for _ in range(n)]
    m=0
    for i in range(n):
        for j in range(n):
            m=max(m, es3bM(i,j,M,T))
    return m
```

```
def es3bM(i,j,M,T):
    if T[i][j]!=-1:
        return T[i][j]
    x =  M[i][j]+1
    if i!=0 and x == M[i-1][j]: T[i][j]= 1 + es3bM(i-1,j,M,T)
    elif j!=len(M)-1 and x == M[i][j+1]: T[i][j]= 1 + es3bM(i,j+1,M,T)
    elif i!=len(M)-1 and x == M[i+1][j]: T[i][j]= 1 + es3bM(i+1,j,M,T)
    elif j!=0 and x == M[i][j-1]: T[i][j]= 1 + es3bM(i,j-1,M,T)
    else: T[i][j]= 1
    return T[i][j]
```