# Δυναμικός Προγραμματισμός (ΔΠ)

# Περίληψη

- Δυναμικός Προγραμματισμός
- Αρχή του Βέλτιστου
- Παραδείγματα

# Δυναμικός Προγραμματισμός ΔΠ (Dynamic Programming DP)

- Μέθοδος σχεδιασμού αλγορίθμων
  - Είναι μια γενική μεθοδολογία και δεν υπάρχει ένα πρότυπο διατύπωσης /επίλυσης προβλημάτων
- Αρχικά ξεκίνησε σαν μαθηματική μέθοδος για τη λήψη σειράς αλληλοσυνδεόμενων αποφάσεων (sequence of interrelated decisions)
- Εξελίχτηκε σε μέθοδο επίλυσης προβλημάτων με «επικαλυπτόμενα» (overlapping) υποπροβλήματα.
  - Το πρόβλημα υποδιαιρείται σε ένα αριθμό σταδίων (stages).
- Το αποτέλεσμα του ΔΠ είναι μια «πολιτική» (policy) η οποία προσδιορίζει τι πρέπει να γίνει σε κάθε στάδιο.
- Η μέθοδος μπορεί να χρησιμοποιηθεί για γραμμικά καθώς και μη-γραμμικά προβλήματα. Επίσης για προβλήματα των οποίων οι παράμετροι είναι γνωστοί με βεβαιότητα καθώς και για στοχαστικά προβλήματα.

# Δυναμικός Προγραμματισμός ΔΠ

- Πολλές φορές η επίλυση προβλημάτων χρησιμοποιώντας αναδρομικές σχέσεις δεν είναι αποδοτική.
  - Επιλύει το ίδιο πρόβλημα πολλές φορές
  - **Παράδειγμα**: Fibonacci Numbers

```
Fib(n)
   if n=0 return 0;
   if n=1 return 1;
   return Fib(n-1)+Fib(n-2);
```

- Υπενθύμιση: ο αλγόριθμος αυτός καλεί τον εαυτό του

# Δυναμικός Προγραμματισμός ΔΠ

- Βασική ιδέα του ΔΠ είναι η χρήση της μνήμης για να αποθηκεύονται οι λύσεις των υπο-προβλημάτων έτσι ώστε να μην είναι αναγκαία η επίλυση τους πολλαπλές φορές
- Θυμηθείτε την λύση του προβλήματος χρησιμοποιώντας πίνακα η οποία είναι $O(n)$!

```
Fib(n)
  F[0]=0;  F[1]=1;
  for i=2 to n
     F[i]=F[i-1]+F[i-2];
  return F[n];
```

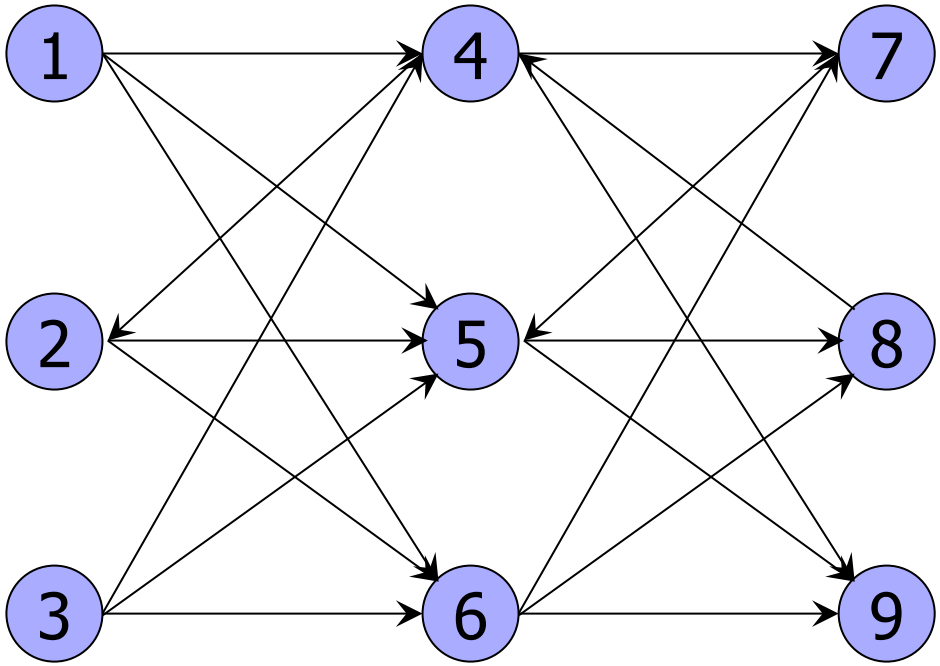
# Transitive Closure

- Το Transitive Closure ενός κατευθυνόμενου γράφου είναι ένας δυαδικός $n$x$n$ πίνακας του οποίου τα στοιχεία

- **Πρόβλημα:** Σχεδιάστε ένα αλγόριθμο που να υπολογίζει το Transitive Closure $T$.

# Transitive Closure

- **Πρώτη Λύση:** Ξεκινώντας από κάθε κόμβο, τρέχουμε ένα αλγόριθμο «εξερεύνησης» γράφου (BFS: Breadth First Search, DFS: Depth-First Search) για να συμπληρώσουμε την κάθε σειρά του πίνακα



# Transitive Closure

- **Λύση ΔΠ:** Ορίζουμε τον πίνακα $T^{(k)}$ ο οποίος μας δίνει τη λύση στο πρόβλημα εάν επιτρέψουμε **μόνο** $k$ βήματα (στάδια). Δηλαδή τα στοιχεία του πίνακα θα δίνονται από:

$$\left[T_{ij}^{(k)}\right] = \begin{cases} 1 & \text{Εάν υπάρχει μονοπάτι από τον } i \text{ στον } j \text{ μέσω των κόμβων μέχρι } k \\ 0 & \text{Εάν δεν υπάρχει μονοπάτι} \end{cases}$$

- Σπάζουμε έτσι το πρόβλημα σε υποπροβλήματα τα οποία θα πρέπει να λυθούν σταδιακά.

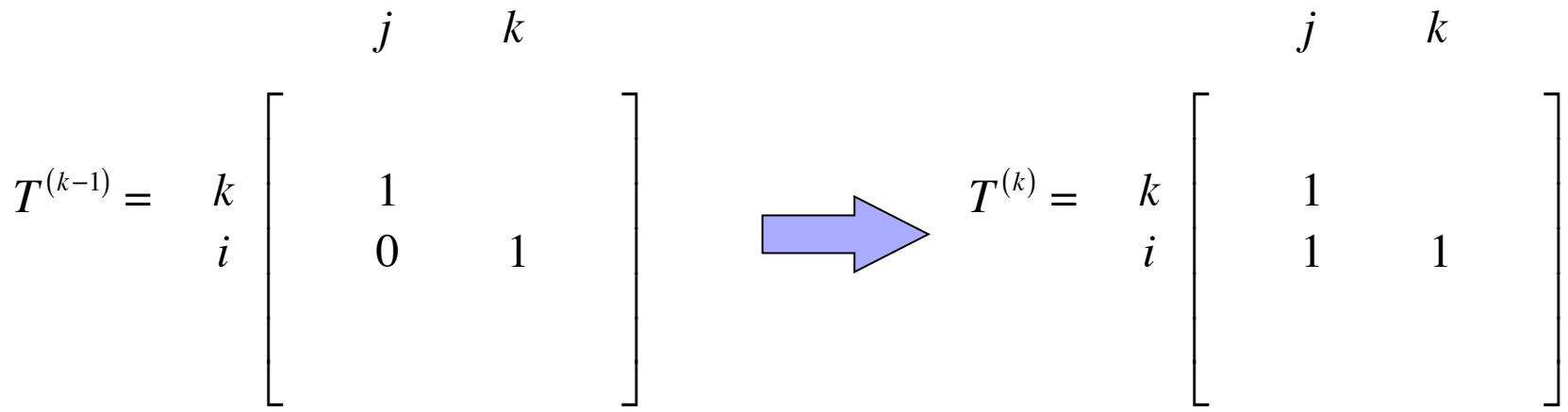
$$T^{(0)}, T^{(1)}, T^{(2)}, \cdots, T^{(k-1)}, T^{(k)}, \cdots, T^{(n)}.$$

- Πως βρίσκουμε τις λύσεις στα πιο πάνω υποπροβλήματα;

# Transitive Closure

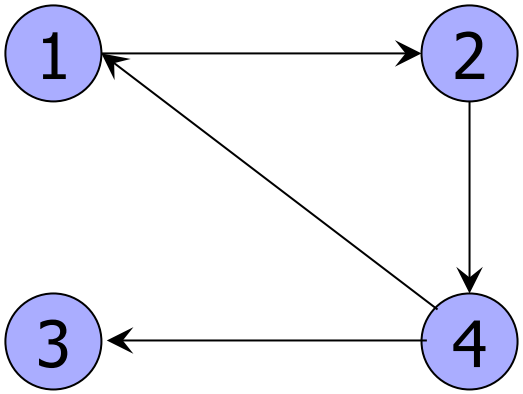
- Το αρχικό στάδιο $T^{(0)}$ δίνεται από τον πίνακα γειτνίασης (Adjacency Matrix), $A$. Γιατί;
- Εάν στο στάδιο $n$-1 έχουμε βρει μονοπάτι από τον κόμβο $v_i$ στον κόμβο $v_j$, τότε το μονοπάτι θα υπάρχει και στο στάδιο $n$.
- Εάν στο στάδιο $n$-1 έχουμε βρει μονοπάτι από τον κόμβο $v_i$ στον κόμβο $v_k$ **και** από τον κόμβο $v_k$ στον κόμβο $v_j$. Τότε συνεπάγετε ότι υπάρχει και μονοπάτι από τον κόμβο $v_i$ στον κόμβο $v_j$ το οποίο εισάγετε στον πίνακα του σταδίου $n$.
- Οι πιο πάνω δύο κανόνες, περιγράφονται με τη σχέση

# Transitive Closure:Αλγόριθμος Warshall




```
Warshall(A)
  T(0)= A;
  for k=1 to n
    for i=1 to n
      for j= 1 to n
        T(k)[i,j]=T(k-1)[i,j]OR(T(k-1)[i,k]AND T(k-1)[k,j])
```

# Transitive Closure: Παράδειγμα




$$T^{(0)} = A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \quad T^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & & 1 & 0 \end{bmatrix}$$

$$T^{(2)} = \begin{bmatrix} 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & \end{bmatrix} \quad T^{(3)} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad T^{(4)} = \begin{bmatrix} & 1 & & 1 \\ & & & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

# Ελάχιστα Μονοπάτια

- **Πρόβλημα:** Υπολογίστε το ελάχιστο μονοπάτι από κάθε σε κόμβο σε όλους του υπόλοιπους κόμβους.
- **Πιθανή Λύση:**
- **Πρόβλημα Απόδοσης**: Ορισμένα μονοπάτια θα υπολογίζονται πολλαπλές φορές.

# Ελάχιστα Μονοπάτια

- **Λύση ΔΠ:** Ορίζουμε τον πίνακα $D(k)$ ο οποίος μας δίνει τη λύση στο πρόβλημα εάν επιτρέψουμε **μόνο** $k$ βήματα (στάδια). Δηλαδή τα στοιχεία του πίνακα δίνουν το ελάχιστο μονοπάτι από τον κόμβο $i$ στον $j$, εάν επιτρέψουμε μονοπάτια μέχρι $k$ κόμβους.

- Σπάζουμε έτσι το πρόβλημα σε υποπροβλήματα τα οποία θα πρέπει να λυθούν σταδιακά.

$$D^{(0)}, D^{(1)}, D^{(2)}, \cdots, D^{(k-1)}, D^{(k)}, \cdots, D^{(n)}.$$

- Πως βρίσκουμε τις λύσεις στα πιο πάνω υποπροβλήματα;
  - Η βασική ιδέα του αλγορίθμου είναι η ίδια με αυτή του Transitive Closure.

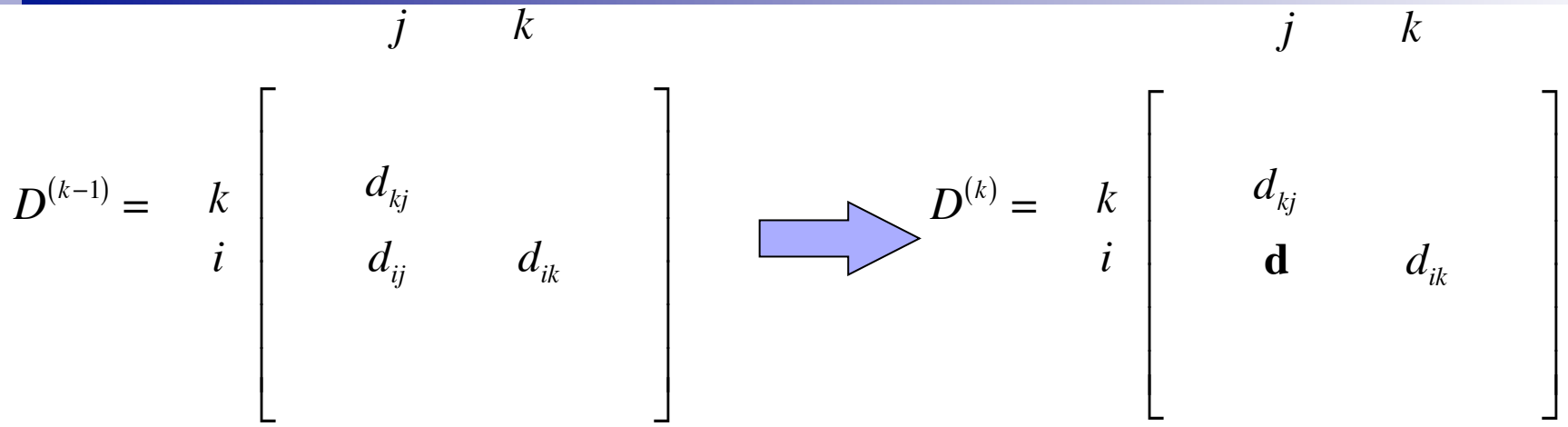
# Ελάχιστα Μονοπάτια

- Το πρώτο στάδιο $D^{(0)}$ δίνεται από τον πίνακα κόστους.

$$\left[D_{ij}^{(0)}\right] = \begin{cases} w_{ij} & \text{Εάν υπάρχει ακμή από τον } i \text{ στον } j \\ \infty & \text{Εάν δεν υπάρχει ακμή} \end{cases}$$

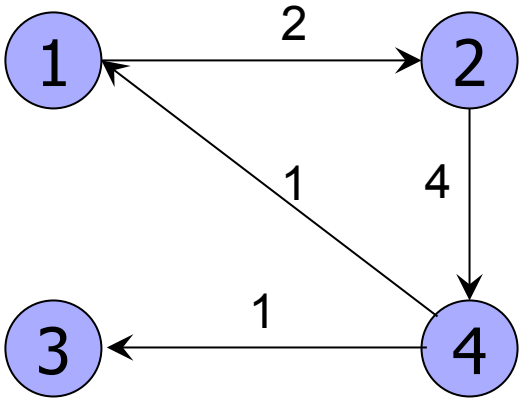
- Υποθέστε πως στο στάδιο $k$-1 έχουμε βρει μονοπάτι από τον κόμβο $v_i$ στον κόμβο $v_j$ με κόστος $c_{ij}$.
- Υποθέστε πως στο στάδιο $k$-1 έχουμε βρει μονοπάτι από τον κόμβο $v_i$ στον κόμβο $v_k$ με κόστος $c_{ik}$ **και** από τον κόμβο $v_k$ στον κόμβο $v_j$ με κόστος $c_{kj}$.
- Τότε, απλά συγκρίνουμε τα $c_{ij}$ και $c_{ik}$+ $c_{kj}$.
- Δηλαδή έχουμε τη σχέση

# Ελάχιστα Μονοπάτια: Αλγόριθμος Floyd




```
Floyd(W)
  D(0)= W;
  for k=1 to n
    for i=1 to n
      for j= 1 to n
        D(k)[i,j]=min{D(k-1)[i,j],D(k-1)[i,k]+D(k-1)[k,j])}
```

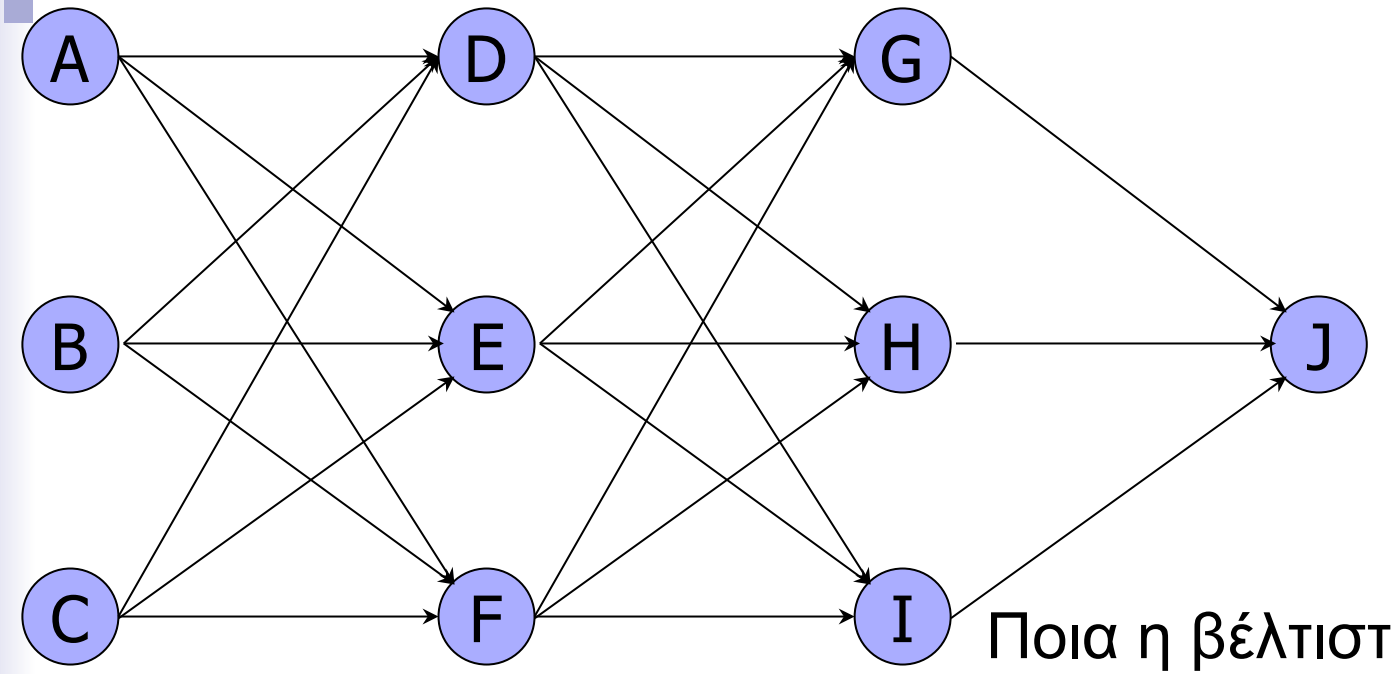
# Ελάχιστα Μονοπάτια: Παράδειγμα




$$D^{(0)} = A = \begin{bmatrix} 0 & 2 & \infty & \infty \\ \infty & 0 & \infty & 4 \\ \infty & \infty & 0 & \infty \\ 1 & \infty & 1 & 0 \end{bmatrix} \quad D^{(1)} = \begin{bmatrix} 0 & 2 & \infty & \infty \\ \infty & 0 & \infty & 4 \\ \infty & \infty & 0 & \infty \\ 1 & & 1 & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & 2 & \infty & \\ \infty & 0 & \infty & 4 \\ \infty & \infty & 0 & \infty \\ 1 & 3 & 1 & 0 \end{bmatrix} \quad D^{(3)} = \begin{bmatrix} 0 & 2 & \infty & 6 \\ \infty & 0 & \infty & 4 \\ \infty & \infty & 0 & \infty \\ 1 & 3 & 1 & 0 \end{bmatrix} \quad D^{(3)} = \begin{bmatrix} 0 & 2 & & 6 \\ & 0 & & 4 \\ \infty & \infty & 0 & \infty \\ 1 & 3 & 1 & 0 \end{bmatrix}$$

# Αρχή του Βέλτιστου (Optimality Principle)



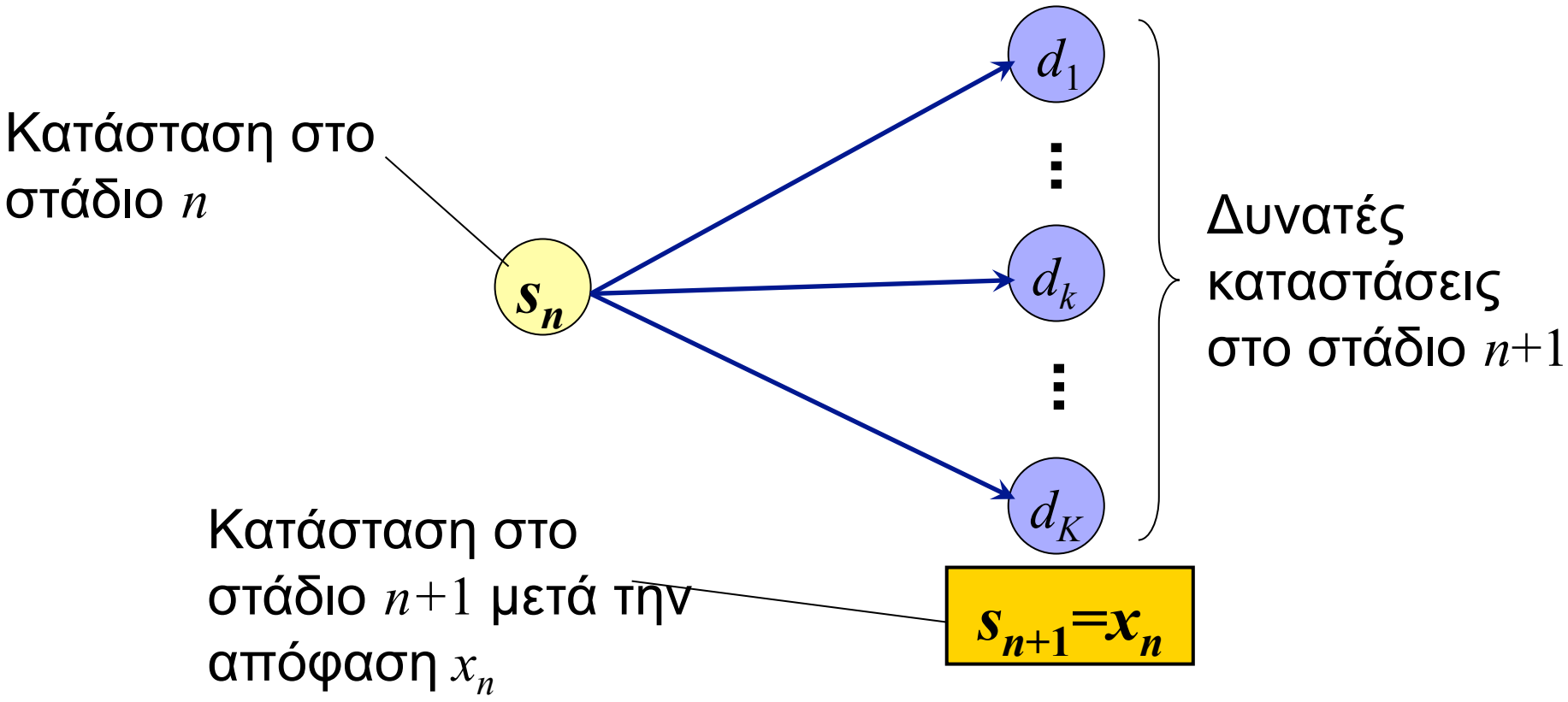

Ποια η βέλτιστη διαδρομή E→J?

**Αρχή του Βέλτιστου (Optimality Principle)**: Η βέλτιστη διαδρομή από μια κατάσταση (state) σε όλα τα υπόλοιπα στάδια είναι ανεξάρτητη από όλες τις αποφάσεις που λήφθηκαν στο παρελθόν.

# Διατύπωση

Στάδιο 1 ··· Στάδιο $n$ Στάδιο $n$+1 ··· Στάδιο $N$

Αποφάσεις: $x_1$ $x_n$ $x_{n+1}$ $x_N$

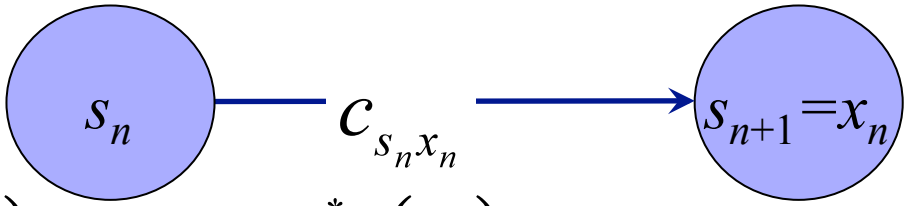


# Διατύπωση

$f_n(s_n, x_n)$: Κόστος από $s_n$ μέχρι τον προορισμό εάν στο σκέλος $n$ πάρουμε την απόφαση $x_n$ και βέλτιστες αποφάσεις σε όλα τα επακόλουθα σκέλη

$f_n^*(s_n)$: Βέλτιστο κόστος από $s_n$ έως τον προορισμό

$x_n^*$: Βέλτιστη απόφαση στην κατάσταση $n$




$$f_n(s_n, x_n) = c_{s_n x_n} + f_{n+1}^*(x_n)$$

$$f_{n+1}^*(s_{n+1}) = f_{n+1}(s_{n+1}, x_{n+1}^*)$$

Βέλτιστο κόστος από την κατάσταση $s_{n+1}$ μέχρι τον τελικό προορισμό

Σε *κάθε* κατάσταση θα λύνουμε ένα πρόβλημα βελτιστοποίησης!

# Δυναμικός Προγραμματισμός (Βελτιστοποίηση)

- Ο αλγόριθμος εύρεσης της βέλτιστης στρατηγικής/πολιτικής ξεκινά από το τελευταίο στάδιο $N$ (last stage).
- Η λύση είναι ένας επαναληπτικός αλγόριθμος και βασίζεται στην ακόλουθη επαναληπτική σχέση: Η βέλτιστη πολιτική στο σκέλος $n$ δίνεται από τη βέλτιστη πολιτική στο σκέλος $n+1$ και τη λύση ενός επιμέρους προβλήματος βελτιστοποίησης

$$f_n^*(s_n) = \min_{x_n} f_n(s_n, x_n) = f_n(s_n, x_n^*)$$

$$= \min_{x_n} \{c_{s_n x_n} + f_{n+1}^*(x_n)\}$$

# Πρόβλημα υπολογισμού νομισμάτων

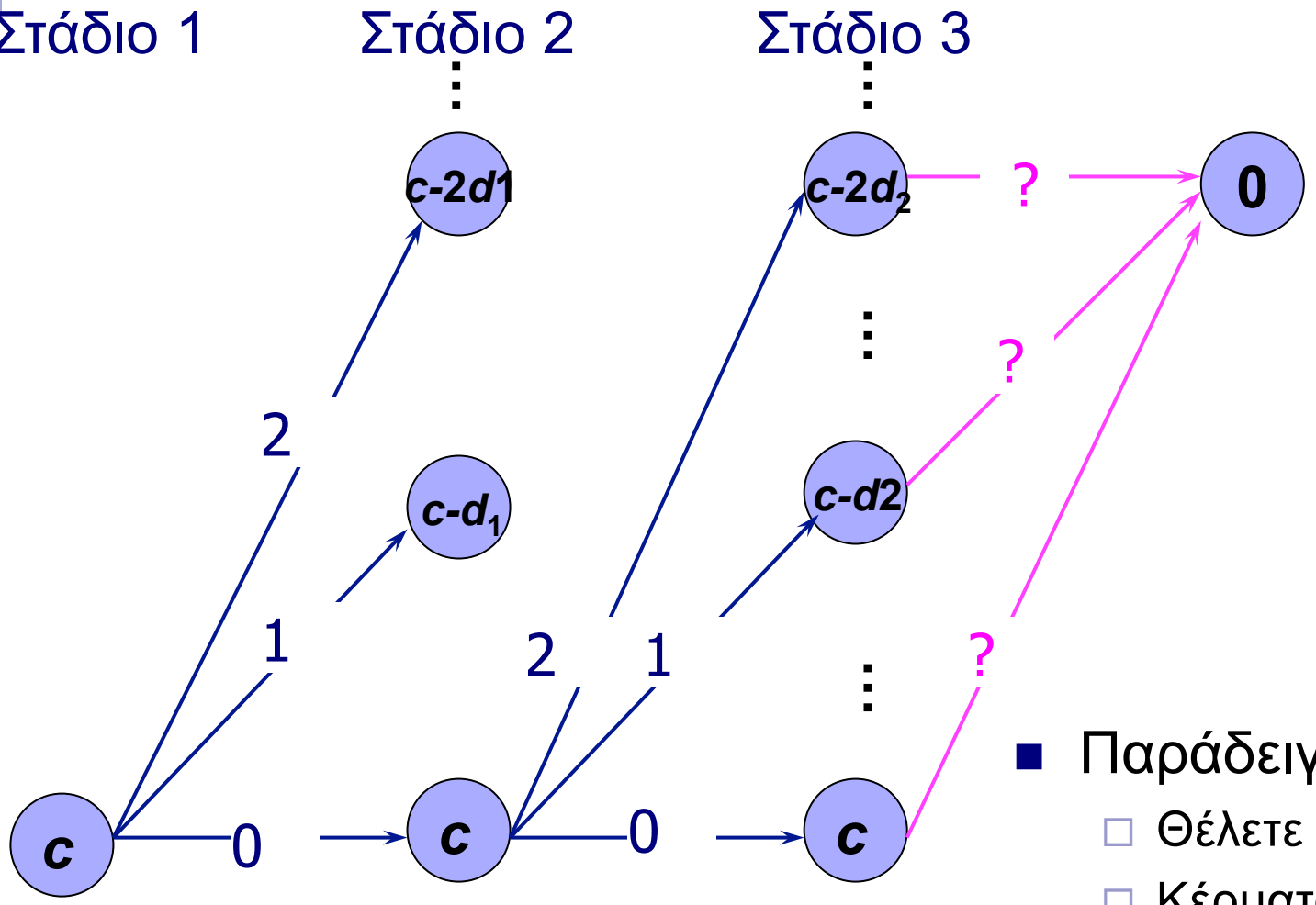
- Υποθέστε πως υπάρχουν κέρματα 1c, 5c, 10c.
- Περιγράψτε ένα αλγόριθμο για τον υπολογισμό του συνδυασμού των νομισμάτων που πρέπει κάποιος να επιστρέψει για ρέστα έτσι που να ελαχιστοποιείται ο αριθμός των κερμάτων.
- Παράδειγμα:
  - Θέλετε να επιστρέψετε 18c. Πως θα το καταφέρεται;
  - 18c= 1x10c + 1x5c + 3x1c
  - Σύνολο 5 κέρματα.
  - Υπάρχει καλύτερη λύση;
  - Ο αλγόριθμος δουλεύει για οποιοδήποτε συνδυασμό κερμάτων;
    - Τι θα συμβεί εάν έχουμε κέρματα 1c, 6c και 10c;

# Πρόβλημα υπολογισμού νομισμάτων

## Διατύπωση (μεταβλητές/αποφάσεις):

- $c$ το ποσό το οποίο θα πρέπει να συμπληρώσουμε
- $d_i$ αξία του κέρματος $i$,
  - π.χ. $d_1$=10c, $d_2$=6c, $d_3$=1c,
- $s_i$ κατάσταση: υπολειπόμενο ποσό μέχρι το $c$
- $x_i$ αριθμός κερμάτων αξίας $d_i$ (απόφαση που πρέπει να ληφθεί)

# Πρόβλημα υπολογισμού νομισμάτων



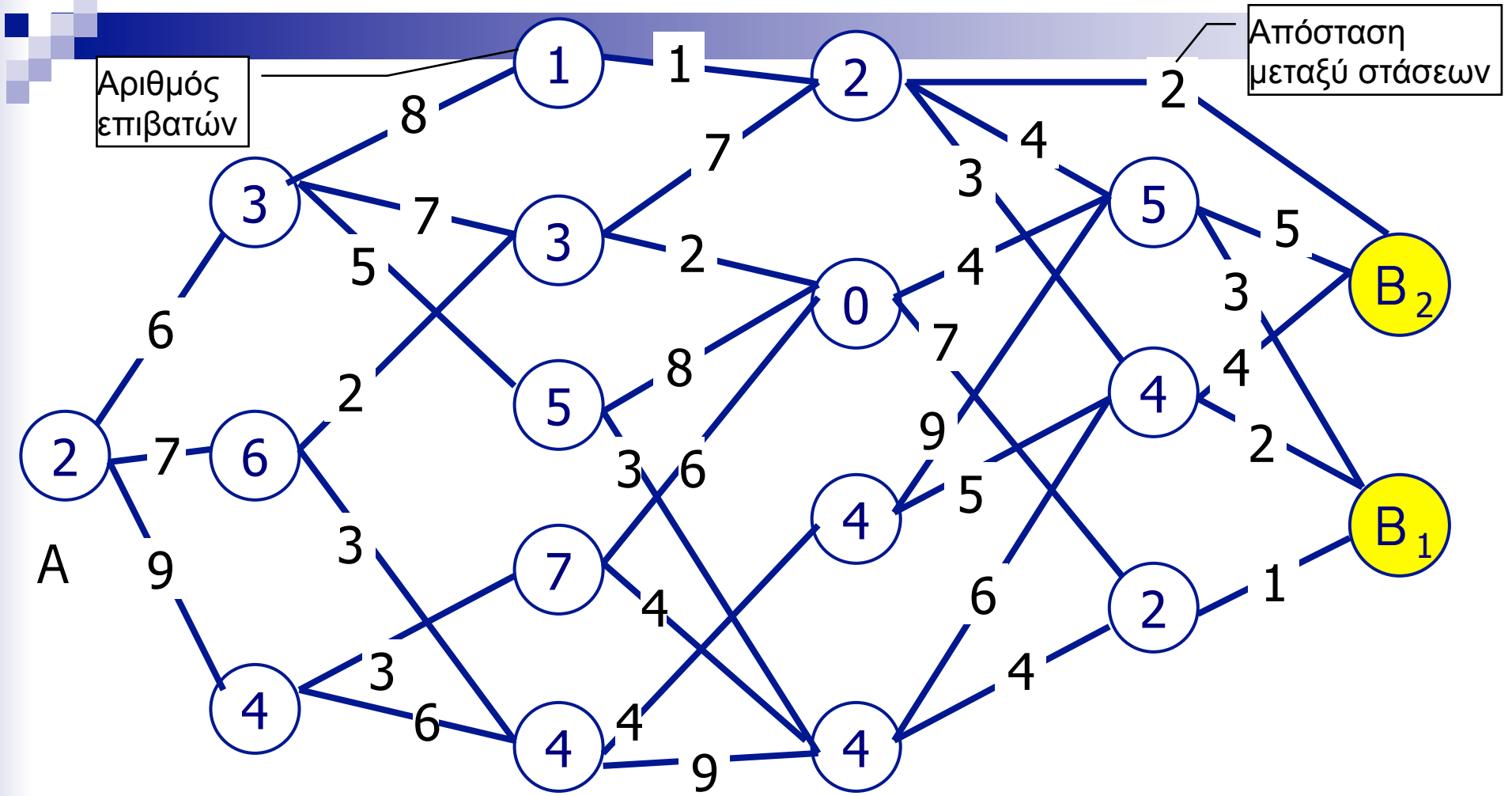

- Παράδειγμα:
  - Θέλετε 18c.
  - Κέρματα 1c, 6c και 10c;

# Πρόβλημα υπολογισμού νομισμάτων

Στάδιο 1 Στάδιο 2 Στάδιο 3

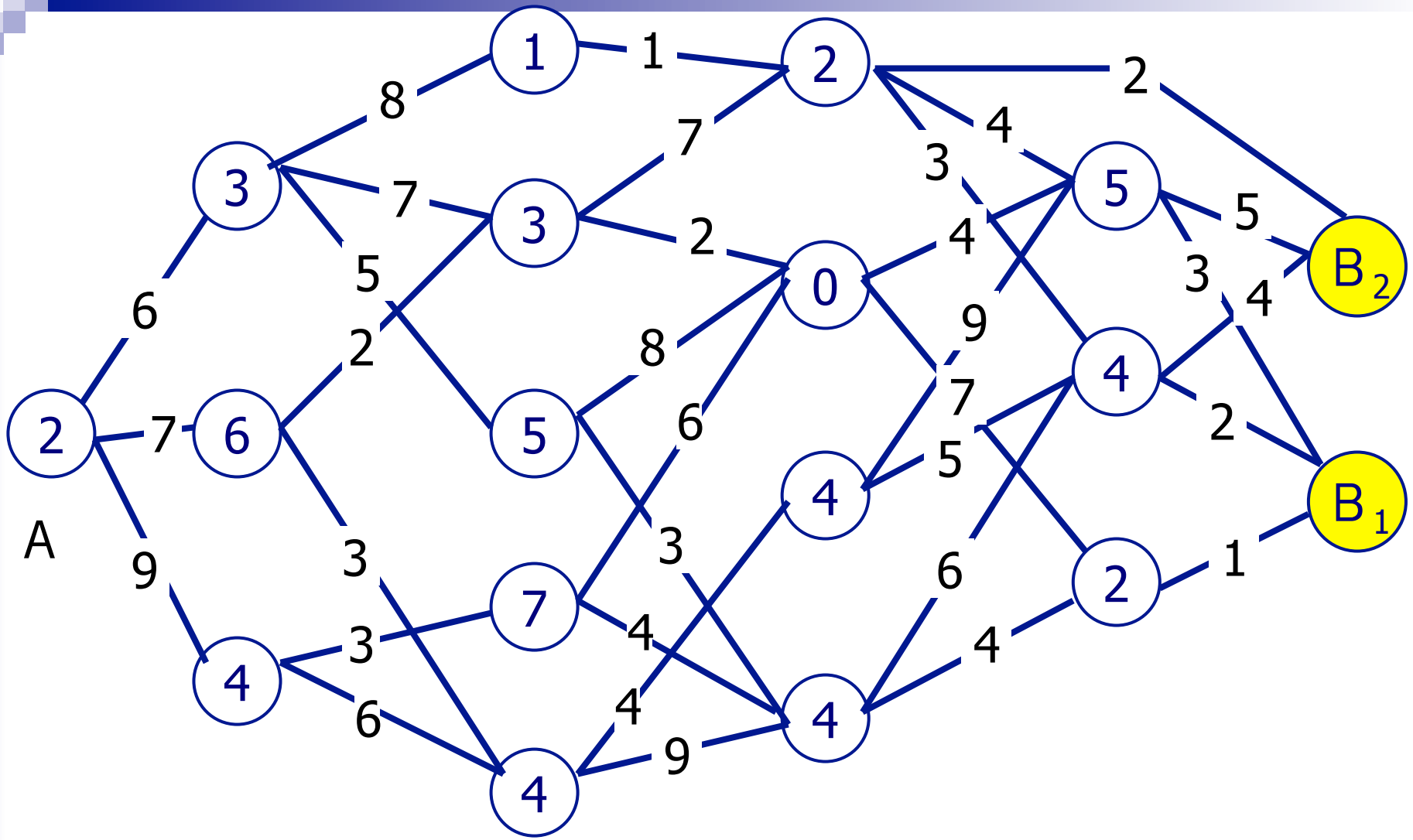
- Παράδειγμα:
  - Θέλετε 18c.
  - Κέρματα 1c, 6c και 10c;

# Λεωφορεία Λευκωσίας

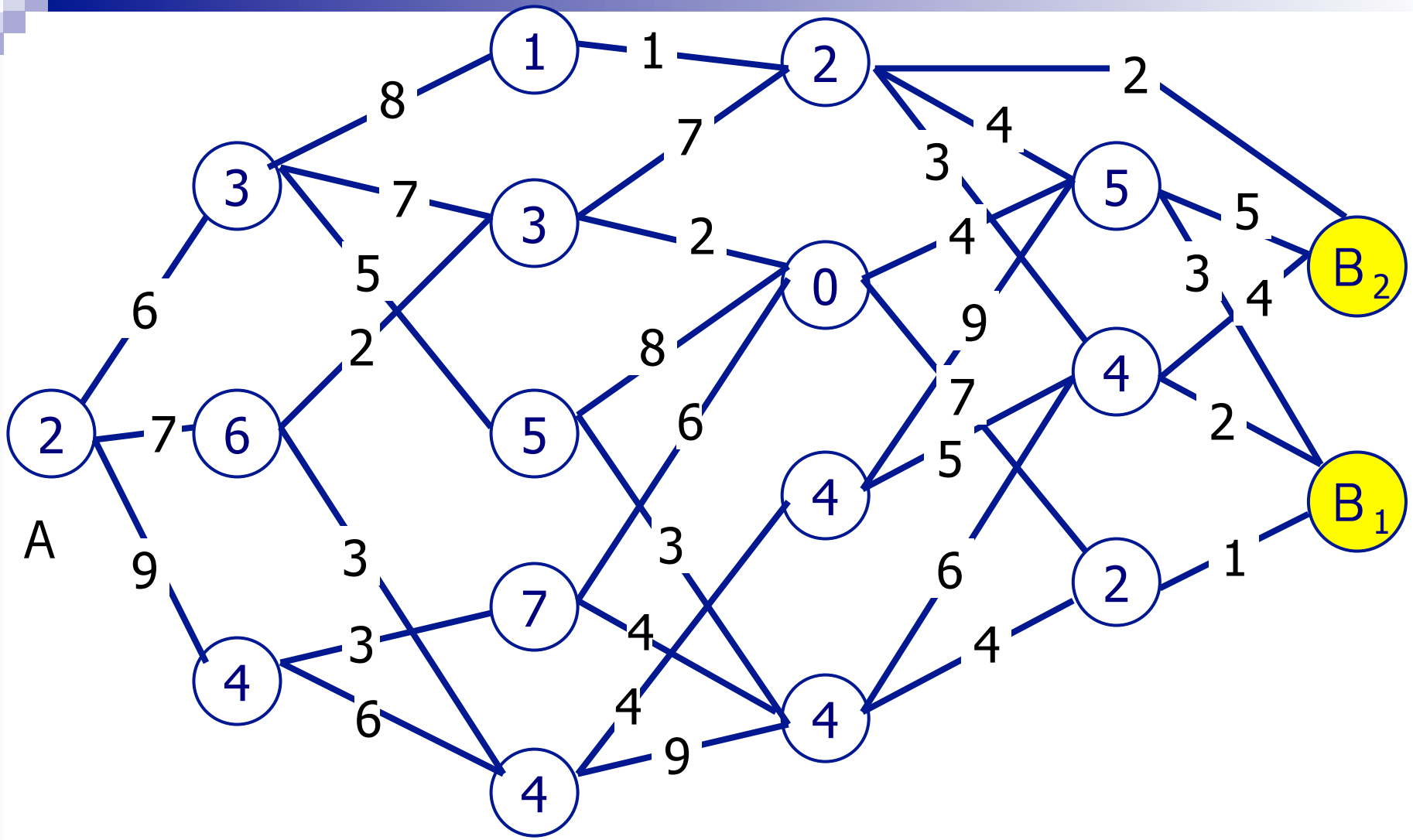



1. Ποια η πιο σύντομη διαδρομή από το A στο $B_2$;
2. Ποια διαδρομή εξυπηρετεί περισσότερους επιβάτες για $B_1$;

# Λεωφορεία Λευκωσίας – Συντομότερη Διαδρομή

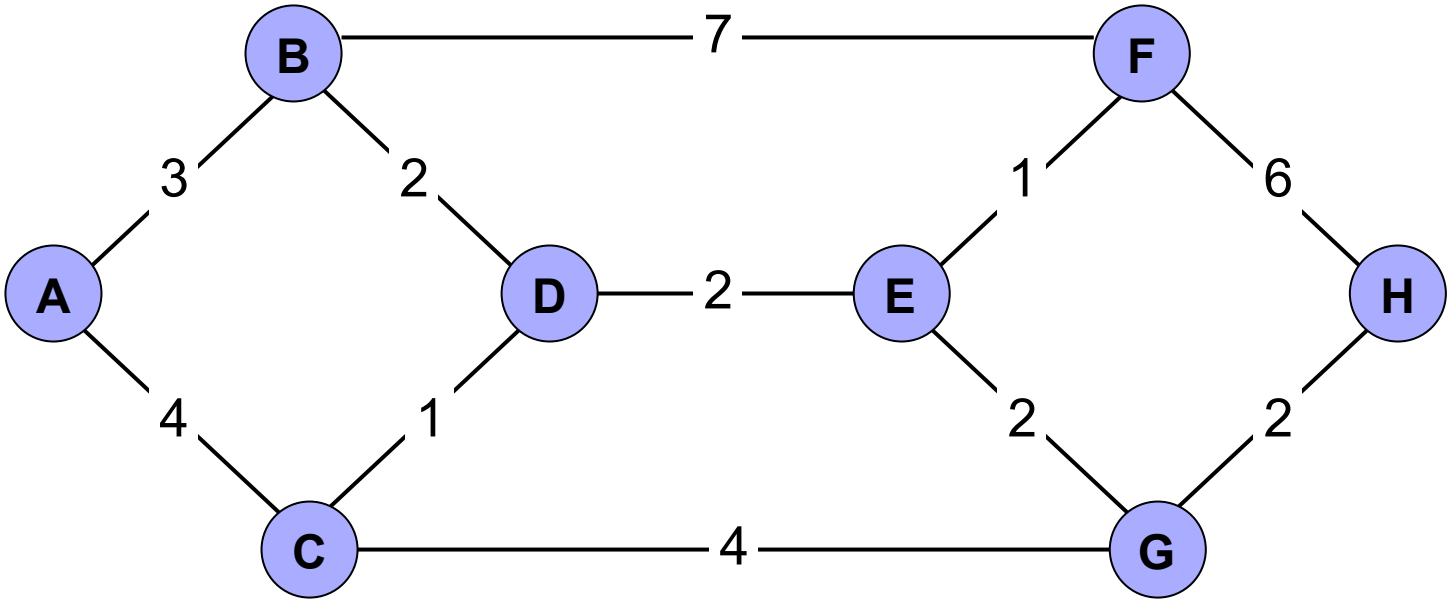


# Λεωφορεία Λευκωσίας – Μέγιστος Αριθμός Επιβατών



# Πρόβλημα Ελάχιστης Απόστασης

Βρέστε το μονοπάτι με το μικρότερο κόστος από Α στο Η




Τι πήγε λάθος;