Άσκηση: Βέλτιστο Δυαδικό Δέντρο Αναζήτησης

Έστω ότι κατασκευάζουμε ένα πρόγραμμα αυτόματης μετάφρασης κειμένου από τα Ελληνικά στα Γαλλικά. Αποθηκεύουμε σε ένα δυαδικό δέντρο αναζήτησης (ΔΔΑ) όλες τις ελληνικές λέξεις μαζί με την αντίστοιχη γαλλική μετάφραση. Λέξεις των οποίων η μετάφρασή τους δεν είναι γνωστή, δεν είναι αποθηκευμένες στο δυαδικό δέντρο αναζήτησης.

Έστω ότι είναι γνωστή εκ των προτέρων η πιθανότητα αναζήτησης ενός κλειδιού-λέξης. Πως θα κατασκευάζατε ένα βέλτιστο δυαδικό δέντρο αναζήτησης, έτσι ώστε να ελαχιστοποιείται το συνολικό κόστος αναζήτησης όλων των κλειδιών;

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης

- Για κάθε λέξη-κλειδί k_i υπάρχει και αντίστοιχη πιθανότητα αναζήτησης p_i:
  - οι λέξεις-κλειδιά με μεγάλη πιθανότητα αναζήτησης πρέπει να βρίσκονται κοντά στη ρίζα του δυαδικού δέντρου αναζήτησης.
  - Οι λέξεις-κλειδιά με μικρή πιθανότητα αναζήτησης, πρέπει να βρίσκονται μακριά από τη ρίζα του δυαδικού δέντρου αναζήτησης.
- Μια αντίθετη οργάνωση του δέντρου θα αύξανε το συνολικό κόστος αναζήτησης:
  - για την αναζήτηση μίας λέξης-κλειδί που βρίσκεται στο m-οστό επίπεδο του δέντρου, χρειάζεται να επισκεφτούμε $m + 1$ κόμβους του δέντρου.

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης

Το ζητούμενο είναι η εύρεση του λεγόμενου **Βέλτιστου Δυαδικού Δέντρου Αναζήτησης (optimal binary search tree)**

Έστω:

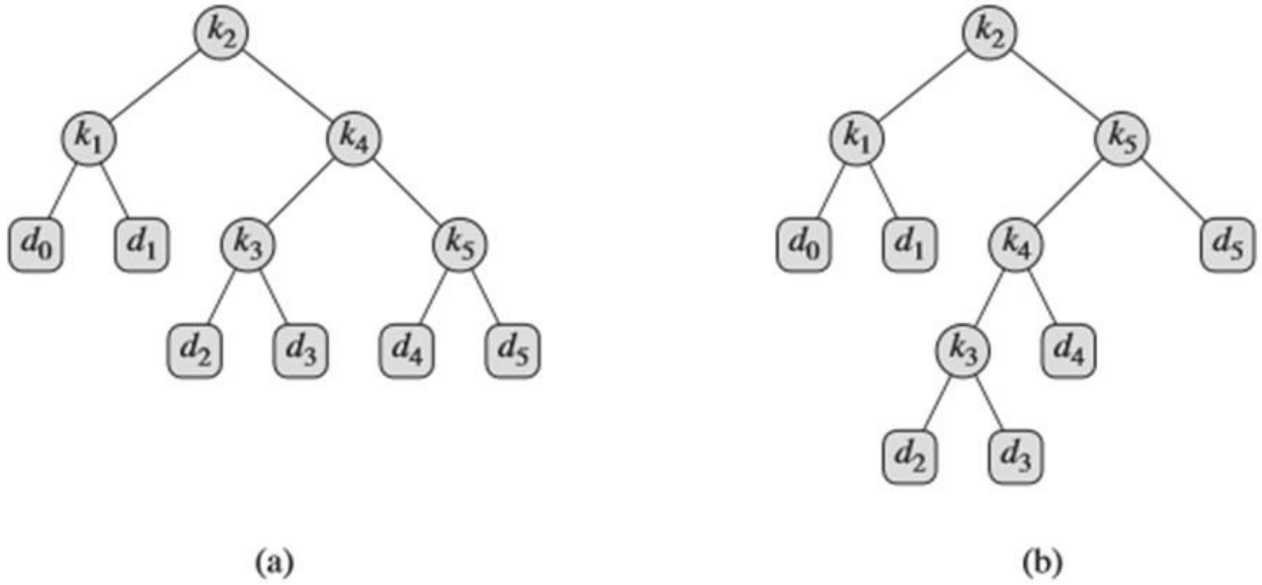
- Μία ταξινομημένη ακολουθία $\boldsymbol{K = (k_1, k_2, \cdots, k_n)}$, n διακριτών λέξεων κλειδιών (με $k_1 < k_2 < \cdots < k_n$)
- Για κάθε λέξη-κλειδί $\boldsymbol{k_i}$ υπάρχει αντίστοιχη πιθανότητα αναζήτησης της λέξης $\boldsymbol{p_i}$.
- Υπάρχει επίσης η πιθανότητα να αναζητηθούν λέξεις οι οποίες δεν υπάρχουν στο λεξικό K, για το λόγο αυτό δημιουργούμε $n + 1$ "πλασματικές" λέξεις-κλειδιά $\boldsymbol{d_0, d_1, d_2, \cdots, d_n}$, που αντιπροσωπεύουν τις λέξεις που δεν υπάρχουν στο K.

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης

Πιο συγκεκριμένα:

- Το κλειδί $\boldsymbol{d_0}$ αντιπροσωπεύει όλες τις τιμές που είναι μικρότερες του $\boldsymbol{k_1}$.
- Το κλειδί $\boldsymbol{d_n}$ αντιπροσωπεύει όλες τις τιμές που είναι μεγαλύτερες του $\boldsymbol{k_n}$.
- Το κλειδί $\boldsymbol{d_i}$ αντιπροσωπεύει όλες τις τιμές ανάμεσα στα $\boldsymbol{k_i}$ και $\boldsymbol{k_{i+1}}$, για κάθε $i = 1,2, \dots, n-1$.
- Για κάθε κλειδί $\boldsymbol{d_i}$ υπάρχει αντίστοιχη **πιθανότητα** αναζήτησης της λέξης $\boldsymbol{q_i}$.

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης

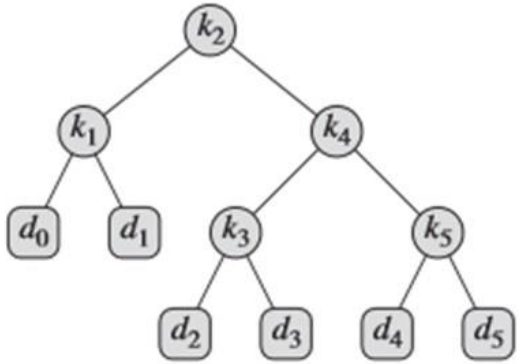


Δύο ΔΔΑ για ένα σύνολο 5 κλειδιών με τις ακόλουθες πιθανότητες

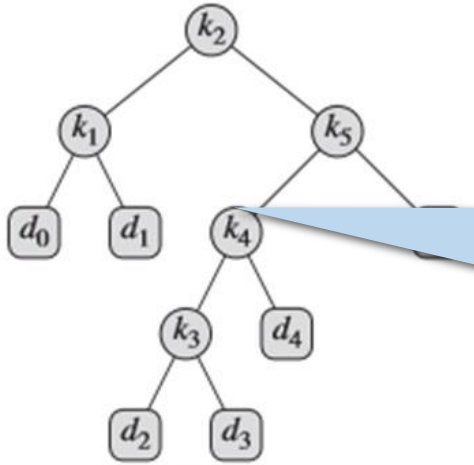
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

a) Ένα ΔΔΑ με εκτιμώμενο κόστος αναζήτησης 2.80, b) ένα ΔΔΑ με εκτιμώμενο κόστος 2.75 (βέλτιστο)

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης



(a)



(b)

Κάθε αναζήτηση είναι είτε επιτυχής και καταλήγει σε κάποιο κλειδί $\boldsymbol{k_i}$, είτε ανεπιτυχής και καταλήγει σε κάποιο πλασματικό κλειδί $\boldsymbol{d_i}$.

Κάθε κλειδί $\boldsymbol{k_i}$ είναι εσωτερικός κόμβος του δέντρου. Ενώ κάθε πλασματικό κλειδί $\boldsymbol{d_i}$ είναι φύλλο του δέντρου.

Δύο ΔΔΑ για ένα σύνολο 5 κλειδιών με τις ακόλουθες πιθανότητες

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

a) Ένα ΔΔΑ με εκτιμώμενο κόστος αναζήτησης 2.80, b) ένα ΔΔΑ με εκτιμώμενο κόστος 2.75 (βέλτιστο)

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης

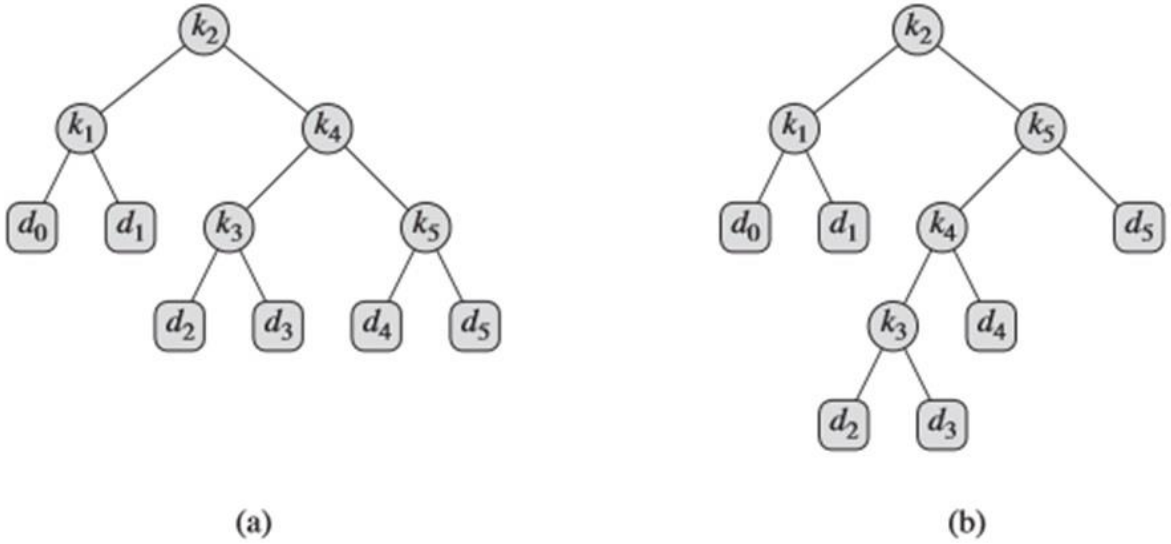
$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 \qquad \text{(Εξ.1)}$$

Το αναμενόμενο κόστος αναζήτησης ενός δυαδικού δέντρου αναζήτησης Τ συνολικά είναι:

$$E[search\ cost\ in\ T] =$$
$$= \sum_{i=1}^{n} (depth_T(k_i) + 1) \cdot\ p_i + \sum_{i=0}^{n} (depth_T(d_i) + 1) \cdot q_i$$
$$= 1 + \sum_{i=1}^{n} depth_T(k_i) \cdot\ p_i + \sum_{i=0}^{n} \big(depth_T(d_i)\big) \cdot q_i \qquad \text{(Εξ.2)}$$

όπου $depth_T$ το βάθος ενός κόμβου στο δέντρο T

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης



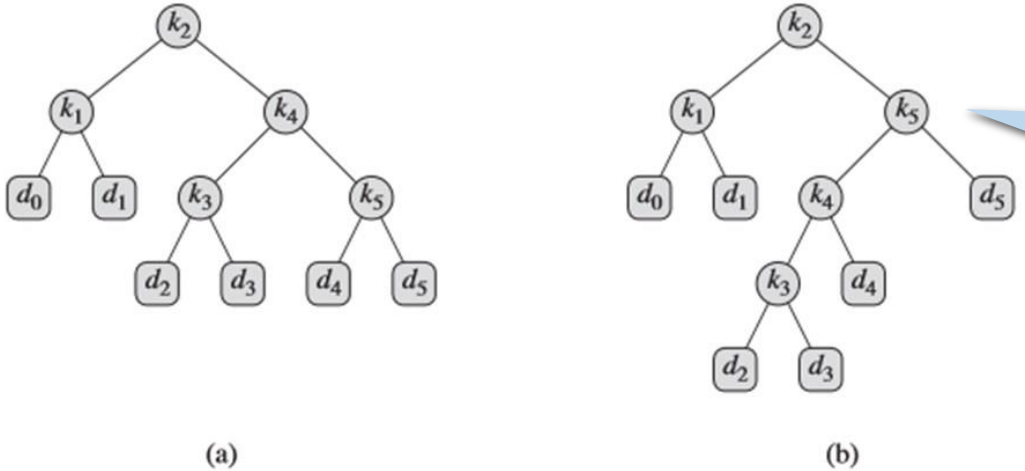
Δύο ΔΔΑ για ένα σύνολο 5 κλειδιών με τις ακόλουθες πιθανότητες

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Κόμβος	Βάθος	Πιθανότητα	Συμβολή
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.10	0.15
d_1	2	0.05	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Σύνολο			2.80

a) Ένα ΔΔΑ με εκτιμώμενο κόστος αναζήτησης 2.80, b) ένα ΔΔΑ με εκτιμώμενο κόστος 2.75 (βέλτιστο)

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης



Η εικόνα δείχνει ένα βέλτιστο ΔΔΑ για το δεδομένο πρόβλημα (αναμενόμενο κόστος αναζήτησης 2.75). Αυτό καταδεικνύει ότι το βέλτιστο ΔΔΑ δεν είναι απαραίτητα και αυτό με το μικρότερο ύψος.

Δύο ΔΔΑ για ένα σύνολο 5 κλειδιών με τις ακόλουθες πιθανότητες

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Επίσης, δεν είναι απαραίτητο το κλειδί με τη μεγαλύτερη πιθανότητα αναζήτησης να είναι στη ρίζα του βέλτιστου ΔΔΑ.
Το μικρότερο δυνατό κόστος αναζήτησης ενός δέντρου με ρίζα το k_5 είναι 2.85.

a) Ένα ΔΔΑ με εκτιμώμενο κόστος αναζήτησης 2.80, b) ένα ΔΔΑ με εκτιμώμενο κόστος 2.75 (βέλτιστο)

Βέλτιστο Δυαδικό Δέντρο Αναζήτησης

- Μία προφανής λύση στο πρόβλημα θα ήταν να εξετάσουμε όλα τα πιθανά ΔΔΑ.
- Αποδεικνύεται ότι ο αριθμός όλων των διαφορετικών ΔΔΑ με n κόμβους είναι:

$$\Omega\left(\frac{4^n}{n^{3/2}}\right)$$

- Συνεπώς, θα έπρεπε να εξετάσουμε **εκθετικό** αριθμό ΔΔΑ, με έναν εξαντλητικό αλγόριθμο.

- Καλύτερη μέθοδος → Δυναμικός Προγραμματισμός

Κατασκευή του Βέλτιστου Δυαδικού Δέντρου Αναζήτησης

Για να προσδιορίσουμε ακριβώς το Βέλτιστο ΔΔΑ, θα ξεκινήσουμε με μία παρατήρηση που αφορά στα υποδέντρα του.

- Θεωρείστε οποιοδήποτε υποδέντρο ενός ΔΔΑ. Αυτό θα πρέπει να περιέχει τα κλειδιά στο συνεχές διάστημα $k_i, \dots, k_j$ για $1 \leq i \leq j \leq n$.
- Επιπρόσθετα το υποδέντρο που περιέχει τα κλειδιά $k_i, \dots, k_j$ θα έχει ως φύλλα τα πλασματικά κλειδιά $d_{i-1}, \dots, d_j$

Παρατήρηση

- Αν ένα Βέλτιστο ΔΔΑ T έχει ένα υποδέντρο T' το οποίο περιέχει τα κλειδιά $k_i, \dots, k_j$, τότε αυτό το **υποδέντρο πρέπει να είναι Βέλτιστο ΔΔΑ για το υποπρόβλημα** με κλειδιά $k_i, \dots, k_j$ και πλασματικά κλειδιά $d_{i-1}, \dots, d_j$

Κατασκευή του Βέλτιστου Δυαδικού Δέντρου Αναζήτησης

- Αν ένα Βέλτιστο ΔΔΑ Τ έχει ένα υποδέντρο Τ′ το οποίο περιέχει τα κλειδιά $k_i, \dots, k_j$, τότε αυτό το **υποδέντρο πρέπει να είναι Βέλτιστο ΔΔΑ για το υποπρόβλημα** με κλειδιά $k_i, \dots, k_j$ και πλασματικά κλειδιά $d_{i-1}, \dots, d_j$

Απόδειξη:

- Με το τέχνασμα της *αποκοπής και επικόλησης*.
- Έστω ότι υπήρχε υποδέντρο Τ′′ του οποίου το αναμενόμενο κόστος ήταν μικρότερο από αυτό του Τ′.
- Τότε θα μπορούσαμε να αποκόψουμε το Τ′ από το Τ και στη θέση του να επικολλήσουμε το Τ′′.
- Με αυτό τον τρόπο θα κατασκευάζαμε ένα νέο ΔΔΑ με μικρότερο κόστος από το Τ, γεγονός το οποίο δεν είναι εφικτό δεδομένου ότι το Τ είναι βέλτιστο.

Κατασκευή του Βέλτιστου Δυαδικού Δέντρου Αναζήτησης

- Επομένως θα χρησιμοποιήσουμε την παρατήρηση για το βέλτιστο υποδέντρο για να δείξουμε πως θα κατασκευάσουμε τη βέλτιστη λύση για το πρόβλημα, από τις βέλτιστες λύσεις για τα υποπροβλήματα.
- Έστω τα κλειδιά $k_i, \dots, k_j$, και ένα εξ αυτών το $\boldsymbol{k_r}$ (με $i \leq r \leq j$), η **ρίζα του βέλτιστου υποδέντρου**.
- Το αριστερό υποδέντρο του k_r περιέχει τα κλειδιά $k_i, \dots, k_{r-1}$ (και τα πλασματικά κλειδιά $d_{i-1}, \dots, d_{r-1}$) και το δεξί υποδέντρο του k_r περιέχει τα κλειδιά $k_{r+1}, \dots, k_j$ (και τα πλασματικά κλειδιά $d_r, \dots, d_j$).
- Εφόσον εξετάσουμε όλες τις πιθανές ρίζες k_r (με $i \leq r \leq j$), και εντοπίσουμε όλα τα βέλτιστα υποδέντρα που περιέχουν τα κλειδιά $k_i, \dots, k_{r-1}$ και $k_{r+1}, \dots, k_j$, τότε με αυτό τον τρόπο σίγουρα θα βρούμε το Βέλτιστο ΔΔΑ.

Κατασκευή του Βέλτιστου Δυαδικού Δέντρου Αναζήτησης

Σημαντική Λεπτομέρεια: **Τα κενά υποδέντρα**

- Υποθέτουμε ότι σε ένα υποδέντρο με κλειδιά τα $k_i, \dots, k_j$ επιλέγουμε το κλειδί k_i ως ρίζα.
- Τότε το αριστερό υποδέντρο θα περιέχει τα κλειδιά $k_i, \dots, k_{i-1}$. Το οποίο συνεπάγεται ότι δεν θα περιέχει κανένα κλειδί.
- Μην ξεχνάτε όμως ότι τα υποδέντρα περιέχουν και πλασματικά κλειδιά .
- Συνεπώς το υποδέντρο των κλειδιών $k_i, \dots, k_{i-1}$ δεν θα περιέχει κανένα κλειδί παρά μόνο το πλασματικό κλειδί d_{i-1}.
- Αντίστοιχα, αν επιλέξουμε το k_j ως ρίζα, το δεξί του υποδέντρο των κλειδιών $k_{j+1}, \dots, k_j$ δεν θα περιέχει κανένα κλειδί παρά μόνο το πλασματικό κλειδί d_j.

Αναδρομική Λύση

- Έστω το υποπρόβλημα της εύρεσης του βέλτιστου ΔΔΑ που περιέχει τα κλειδιά $k_i, \dots, k_j$ με $i \geq 1, j \leq n$ $\kappa\alpha\iota$ $j \geq i-1$.
  - (όταν $j = i-1$ τότε δεν υπάρχουν κλειδιά παρά μόνο το πλασματικό κλειδί d_{i-1})

- Έστω $\boldsymbol{e[i,j]}$ το αναμενόμενο κόστος αναζήτησης σε ένα βέλτιστο ΔΔΑ που περιέχει τα κλειδιά $k_i, \dots, k_j$.

- Επομένως το ζητούμενο είναι ο υπολογισμός του $e[1,n]$.

- Η εύκολη περίπτωση ισχύει όταν $j = i-1$. Στην περίπτωση αυτή έχουμε μόνο το πλασματικό κλειδί d_{i-1}. Επομένως το αναμενόμενο κόστος αναζήτησης είναι $\boldsymbol{e[i,j] = q_{i-1}}$.

Αναδρομική Λύση

- Όταν $και\ j \geq i$, πρέπει να επιλέξουμε ένα κλειδί k_r ως ρίζα και να κατασκευάσουμε ένα βέλτιστο ΔΔΑ με τα κλειδιά $k_i, \ldots, k_{r-1}$, ως το αριστερό υποδέντρο του και ένα βέλτιστο ΔΔΑ με τα κλειδιά $k_{r+1}, \ldots, k_j$, ως το δεξί υποδέντρο του.

Τι συμβαίνει στο αναμενόμενο κόστος αναζήτησης ενός υποδέντρου όταν γίνεται το υποδέντρο ενός κόμβου;

Το βάθος κάθε κόμβου στο υποδέντρο αυτό αυξάνεται κάτα 1. Συνεπώς, σύμφωνα με την (Εξ.2), το αναμενόμενο κόστος αναζήτησης στο υποδέντρο αυξάνεται όσο και το άθροισμα όλων των πιθανοτήτων των κόμβων του υποδέντρου.

$$= 1 + \sum_{i=1}^{n} (depth_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (depth_T(d_i) + 1) \cdot q_i =$$

$$1 + \sum_{i=1}^{n} depth_T(k_i) \cdot p_i + \sum_{i=0}^{n} depth_T(d_i) \cdot q_i + \sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i$$

Αναδρομική Λύση

Για ένα υποδέντρο με κλειδιά τα $k_i, \dots, k_j$:

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_i \qquad \text{(Εξ.3)}$$

Εφόσον k_r είναι η ρίζα ενός βέλτιστου ΔΔΑ που περιέχει τα κλειδιά $k_i, \dots, k_j$:

$$e[i,j] = p_r + (e[i,r-1] + w[i,r-1]) + (e[r+1,j] + w[r+1,j])$$

Σημειώνεται ότι:

$$\mathrm{w}[i,j] = w[i,r-1] + p_r + w[r+1,j]$$

Επομένως:

$$e[i,j] = e[i,r-1] + e[r+1,j] + w[i,j] \qquad \text{(Εξ.4)}$$

Αναδρομική Λύση

- Η εξίσωση (Εξ.4) προϋποθέτει ότι γνωρίζουμε ποιον κόμβο k_r θα επιλέξουμε για ρίζα.
- Επιλέγουμε για ρίζα τον κόμβο που μας δίνει το χαμηλότερο αναμενόμενο κόστος αναζήτησης, έτσι προκύπτει ο ακόλουθος **αναδρομικός τύπος:**

$$e[i,j] = \begin{cases} q_{i-1} & if\ j = i-1 \\ \min\limits_{i \le r \le j}\{e[i,r-1] + e[r+1,j] + w(i,j)\} & if\ i \le j \end{cases} \qquad \text{(Εξ.5)}$$

- Για να ανακτήσουμε εύκολα τη δομή του βέλτιστου ΔΔΑ, ορίζουμε $\boldsymbol{root[i,j]}$ για $1 \le i \le j \le n$, το δείκτη r για τον οποίο το k_r είναι η ρίζα ενός βέλτιστου ΔΔΑ που περιέχει τα κλειδιά $k_i, \dots, k_j$.

Υπολογισμός Κόστους Κατασκευής Βέλτιστιου ΔΔΑ

- Δεν είναι αποτελεσματικό να υπολογίσουμε τη λύση χρησιμοποιώντας απευθείας την αναδρομική εξίσωση (Εξ.5).
- Χρησιμοποιούμε ένα δυσδιάστατο πίνακα $\boldsymbol{e[1 \ldots n+1, 0 \ldots n]}$ για να υπολογίσουμε όλα τα $e[i, j]$.
- Ο πρώτος δείκτης φτάνει μέχρι $n+1$ και όχι n διότι χρειάζεται να υπολογίσω και το υποδέντρο που περιέχει μόνο το πλασματικό κλειδί d_n στην θέση του πίνακα $e[n+1, n]$.
- Ο δεύτερος δείκτης ξεκινά από το 0 διότι χρειάζεται να υπολογίσω και το υποδέντρο που περιέχει μόνο το πλασματικό κλειδί d_0 στην θέση του πίνακα $e[1,0]$.
- Χρησιμοποιούμε μόνο τις θέσεις για τις οποίες ισχύει ότι $j \geq i-1$.

Αλγόριθμος Εύρεσης Βέλτιστου ΔΔΑ

Optimal-BST(p, q, n)

1. Έστω $e[1..n+1, 0..n],\ w[1..n+1, 0..n], \text{root}[1..n, 1..n]$
2. **for** $i = 1$ **to** $n+1$
3. $\quad e[i, i-1] = q_{i-1}$
4. $\quad \text{w}[i, i-1] = q_{i-1}$
5. **for** $l = 1$ **to** n
6. $\quad$ **for** $i = 1$ to $n - l + 1$
7. $\quad\quad j = i + l - 1$
8. $\quad\quad e[i, j] = \infty$
9. $\quad\quad w[i, j] = w[i, j-1] + p_j + q_j$
10. $\quad\quad$ **for** $r = i$ **to** j
11. $\quad\quad\quad t = e[i, r-1] + e[r+1, j] + w[i, j]$
12. $\quad\quad\quad$ **if** $t < e[i, j]$
13. $\quad\quad\quad\quad e[i, j] = t$
14. $\quad\quad\quad\quad root[i, j] = r$
15. **return** $e, root$

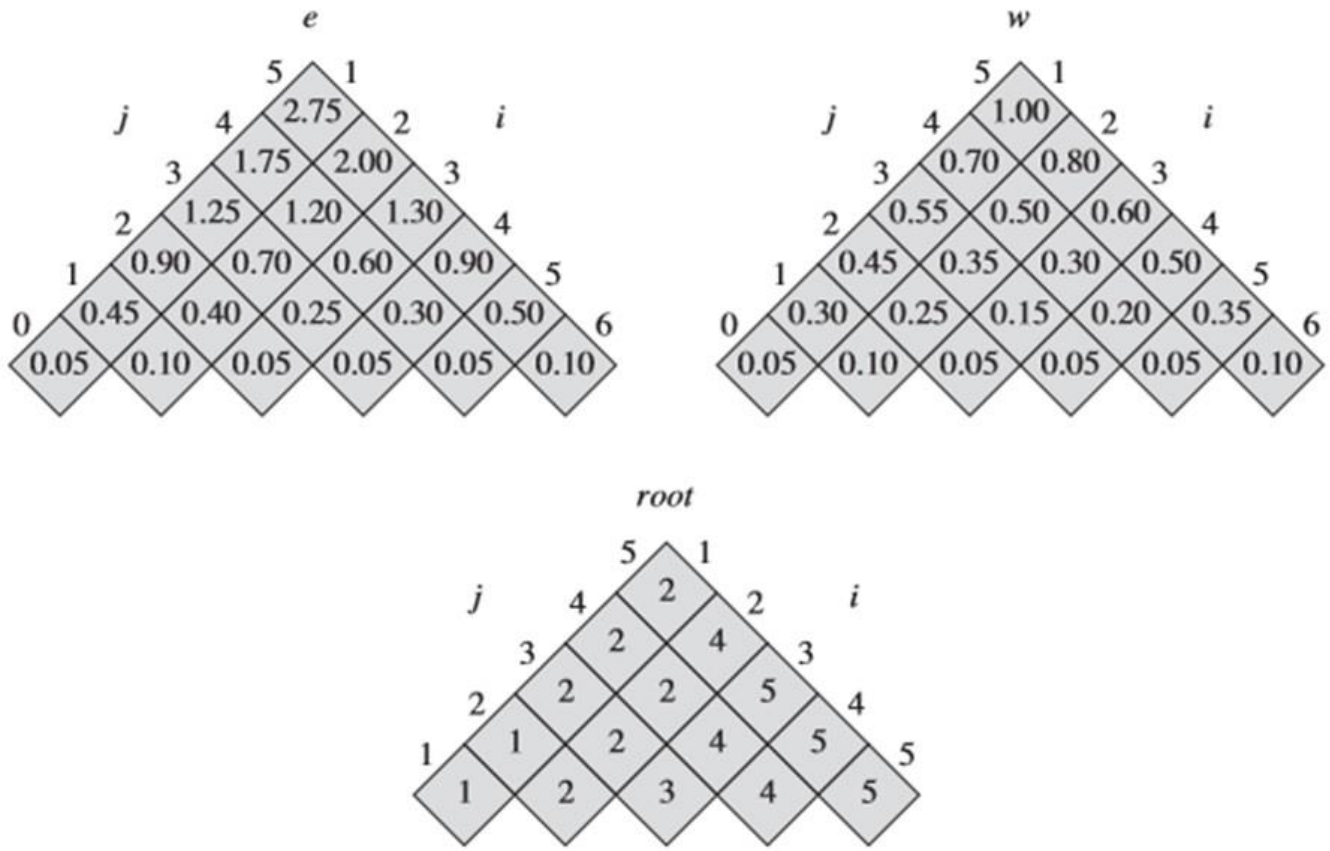
Το for loop στις γραμμές 2-4 αρχικοποιεί τα $e[i, i-1]$ και $w[i, i-1]$

Το for loop στις γραμμές 5-14 χρησιμοποιεί τις αναδρομικές εξισώσεις για να υπολογίσει τα $e[i, j], w[i, j]\ \forall\ 1 \le i \le j \le n$

Το εσωτερικό for loop στις γραμμές 10-14 δοκιμάζει κάθε υποψήφιο δείκτη r για να αποφασίσει ποια θα είναι η ρίζα του δέντρου k_r το οποίο θα περιέχει τα κλειδιά $k_i, \dots, k_j$

Στην 1η επανάληψη του for loop στις γραμμές 5-14, για $l = 1$ υπολογίζονται τα $e[i, i], w[i, i]\ \forall\ 1 \le i \le n$.
Στη 2η επανάληψη, για $l = 2$ υπολογίζονται τα $e[i, i+1], w[i, i+1]\ \forall\ 1 \le i \le n-1$

Βέλτιστο ΔΔΑ – Αναπαράσταση Λύσης Optimal-BST



Πίνακες $e[i,j], w[i,j], root[i,j]$ του αλγορίθμου Optimal-BST

Υπολογισμός Κόστους Κατασκευής ΒΔΔΑ

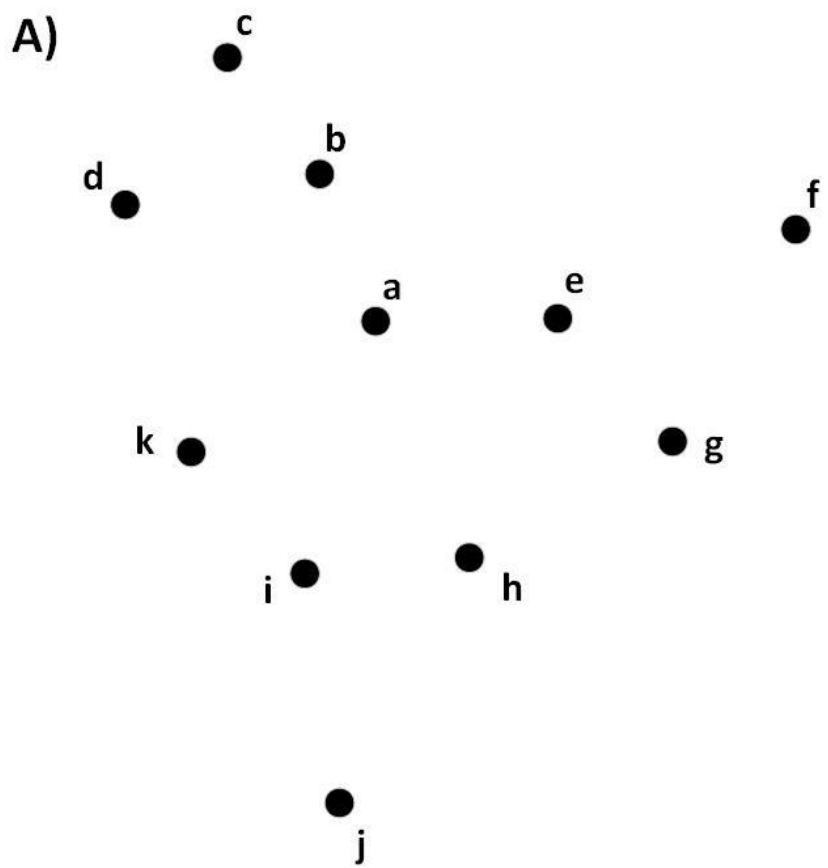
- Πολυπλοκότητα κατασκευής Βέλτιστου Δυαδικού Δέντρου Αναζήτησης:
$$O(n^3)$$
- Γιατί;

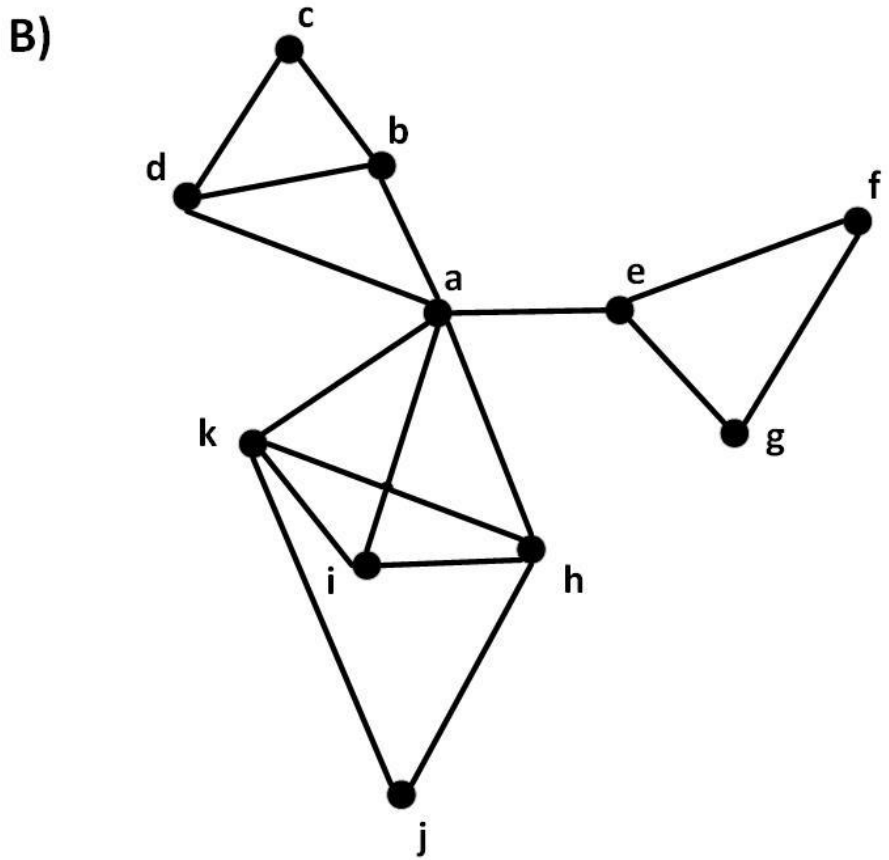
  Σκεφτείτε τα for loops του αλγορίθμου

Άσκηση: Τοποθέτηση Ελάχιστου Αριθμού Σχολείων

- Σε μια αναπτυσσόμενη περιοχή θέλουμε να αποφασίσουμε που θα τοποθετήσουμε τα δημόσια σχολεία, έτσι ώστε να εξυπηρετούνται όλες οι πόλεις της περιοχής.
- Θα πρέπει τα σχολεία να τοποθετηθούν σε πόλεις, με τέτοιο τρόπο ώστε κάθε μαθητής να μη διανύει πάνω από 30 χλμ. ώστε να προσεγγίσει το κοντινότερο σχολείο.
- Σας δίνεται το σύνολο των πόλεων B και για κάθε πόλη i δίνεται το υποσύνολο S_i με τις πόλεις που απέχουν από την πόλη i λιγότερο από 30 χλμ.
- Ποιος είναι ο ελάχιστος αριθμός σχολείων που πρέπει να ιδρυθούν και σε ποιες πόλεις πρέπει να τοποθετηθούν έτσι ώστε να ικανοποιούνται οι περιορισμοί του προβλήματος;

Παράδειγμα Πόλεων

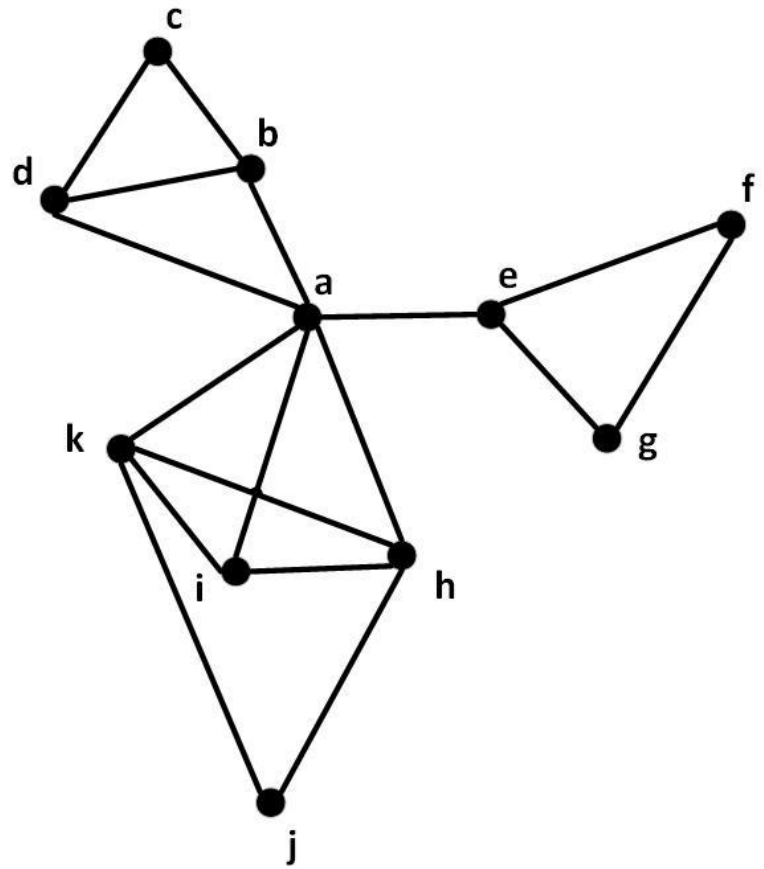




Ιδέα Απληστίας

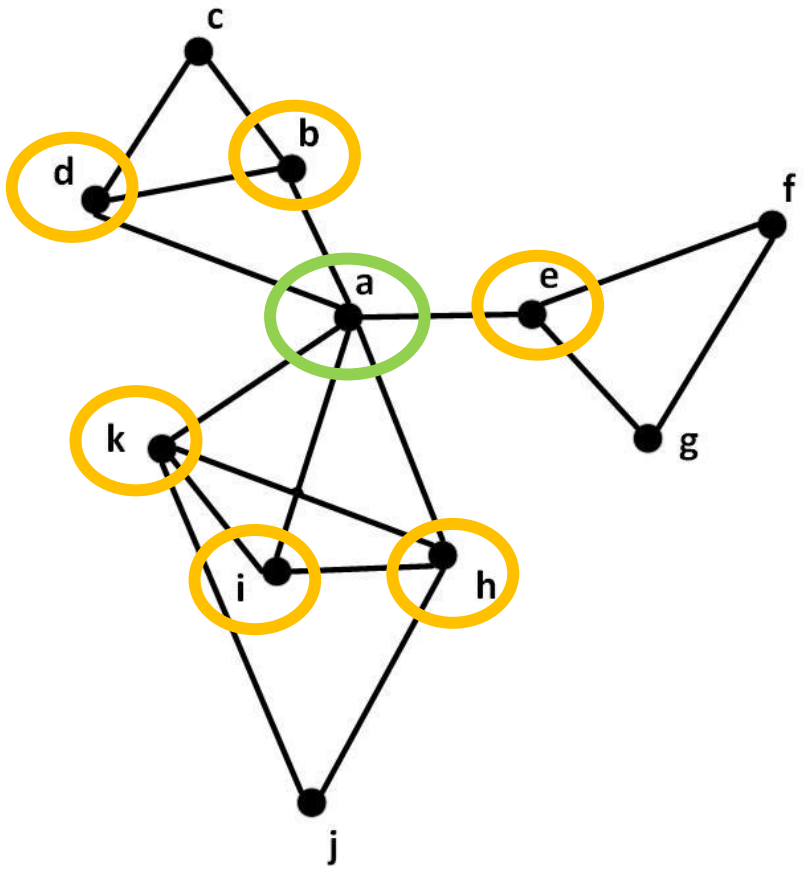
- Επιλέγω διαισθητικά την πόλη που «καλύπτει» το μεγαλύτερο πλήθος πόλεων (που δεν έχουν ήδη εξυπηρετηθεί).
- Επαναλαμβάνω το ίδιο μέχρι να εξυπηρετούνται όλες οι πόλεις.

Άπληστη Λύση



- Άπληστη λύση:

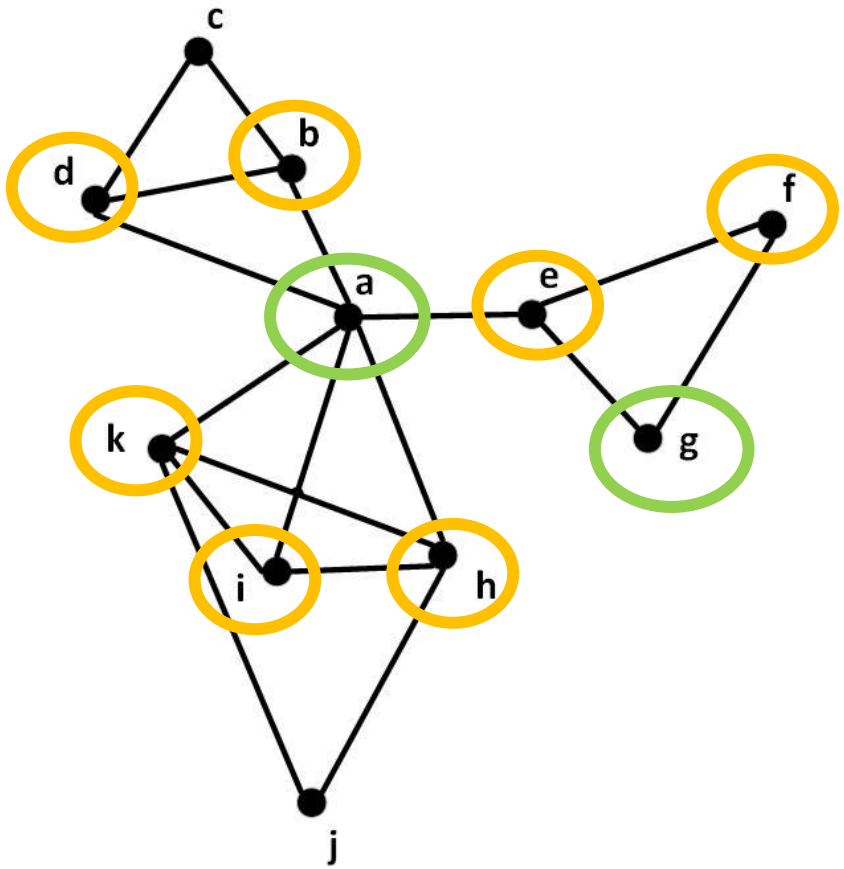
Άπληστη Λύση



- Άπληστη λύση:

Πόλεις: a

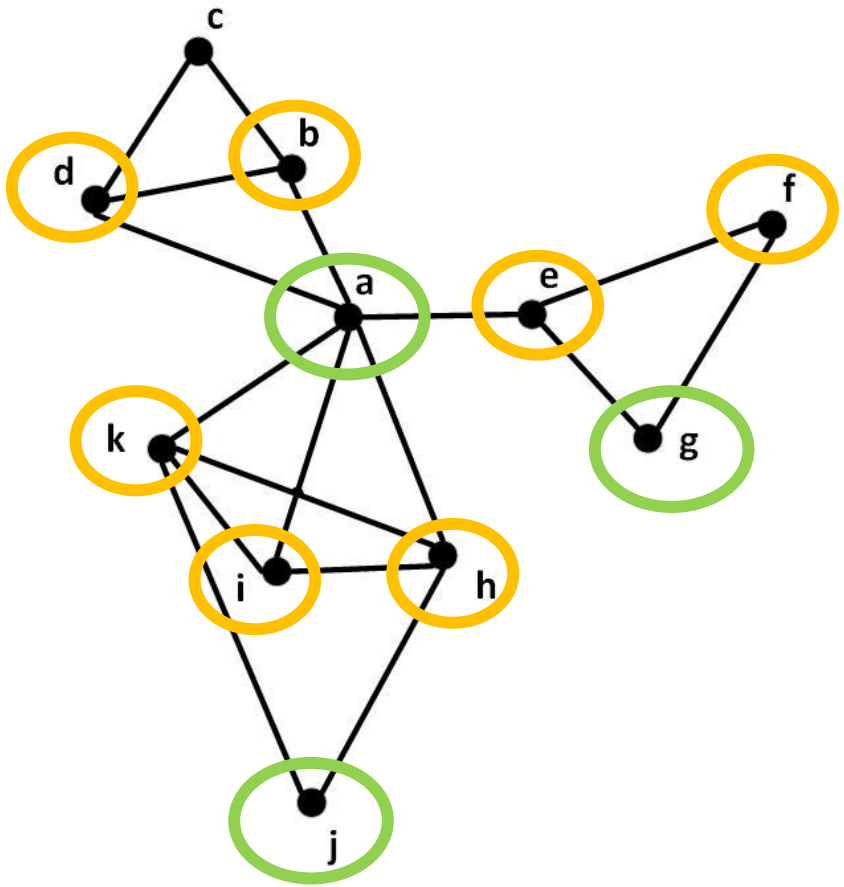
Άπληστη Λύση



- Άπληστη λύση:

Πόλεις: a, g (ή f), j

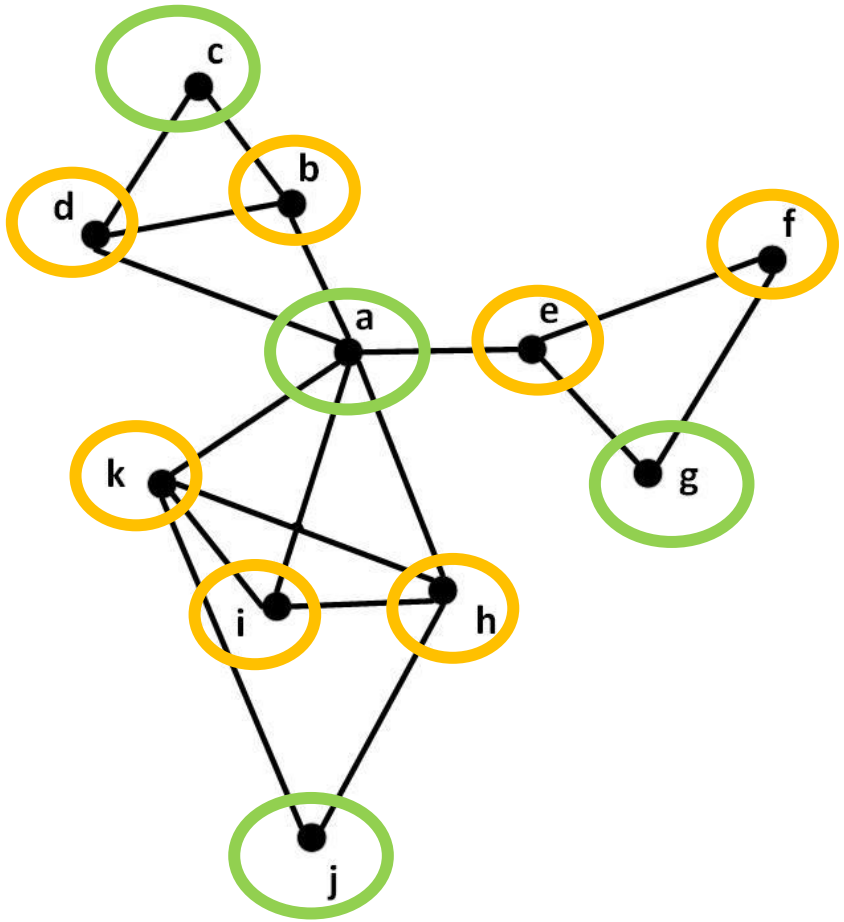
Άπληστη Λύση



- Άπληστη λύση:

Πόλεις: a, g (ή f), j

Άπληστη Λύση



- Άπληστη λύση:

Πόλεις: a, g (ή f), j, c

❖Είναι βέλτιστη η λύση αυτή;

ΟΧΙ

Αντιπαράδειγμα:

Λύση: b, e, h

Άπληστη Λύση – Λόγος Προσέγγισης

Ισχυρισμός: Έστω ότι το σύνολο των πόλεων B περιέχει n στοιχεία και ότι η βέλτιστη κάλυψη περιέχει k σχολεία. Τότε ο άπληστος αλγόριθμος θα χρησιμοποιήσει το πολύ $k \ln n$ σχολεία.

Απόδειξη:

- Έστω n_t το πλήθος των πόλεων που δεν έχουν καλυφθεί έπειτα από t επαναλήψεις του άπληστου αλγορίθμου ($\Rightarrow n_0 = n$).
- Δεδομένου ότι αυτές οι εναπομείνασες πόλεις καλύπτονται από τα k σχολεία της βέλτιστης λύσης ⇒ υπάρχει σχολείο που καλύπτει τουλάχιστον $\frac{n_t}{k}$ από τις πόλεις αυτές.
- Επομένως ο άπληστος αλγόριθμος εξασφαλίζει ότι $n_{t+1} \leq n_t - \frac{n_t}{k} = n_t \left(1 - \frac{1}{k}\right)$
- Μετά από πλήθος εφαρμογών του αλγορίθμου: $n_t \leq n_0 \left(1 - \frac{1}{k}\right)^t$

Άπληστη Λύση – Λόγος Προσέγγισης

Ισχυρισμός: Έστω ότι το σύνολο των πόλεων B περιέχει n στοιχεία και ότι η βέλτιστη κάλυψη περιέχει k σχολεία. Τότε ο άπληστος αλγόριθμος θα χρησιμοποιήσει το πολύ $k \ln n$ σχολεία.

Απόδειξη (συν.):

- Εύρεση πιο σφιχτού ορίου
- Ισχύει $1 - x \leq e^{-x}$ $\forall$ x, όπου η ισότητα ισχύει για $x = 0$
- Βασιζόμενοι στην ανισότητα: $n_t \leq n_0 \left(1 - \frac{1}{k}\right)^t < n_0 \left(e^{-\frac{1}{k}}\right)^t = ne^{-\frac{t}{k}}$
- Μετά την $t = k \ln n$ επανάληψη ισχύει $n_t < ne^{-\ln n} = 1 \Rightarrow$ όλες οι πόλεις έχουν καλυφθεί

Άπληστη Λύση – Λόγος Προσέγγισης

Ισχυρισμός: Έστω ότι το σύνολο των πόλεων B περιέχει n στοιχεία και ότι η βέλτιστη κάλυψη περιέχει k σχολεία. Τότε ο άπληστος αλγόριθμος θα χρησιμοποιήσει το πολύ $k \ln n$ σχολεία.

Το όριο που βρήκαμε ονομάζεται λόγος προσέγγισης (approximation factor) του άπληστου αλγορίθμου

Άσκηση 6

Διάφορα νομίσματα είναι τοποθετημένα τυχαία στα κελιά ενός $m \times n$ πλέγματος (ένα νόμισμα σε κάθε κελί). Ένα ρομπότ, με αρχική θέση την πάνω αριστερή γωνία του πλέγματος, πρέπει να συλλέξει όσο το δυνατόν περισσότερα νομίσματα γίνεται και να τα φέρει στην κάτω δεξιά γωνία του πλέγματος. Σε κάθε βήμα, το ρομπότ μπορεί να μετακινηθεί είτε ένα κελί κάτω είτε ένα κελί δεξιά από την τρέχουσα θέση του. Όταν το ρομπότ επισκέπτεται κελί με νόμισμα, το συλλέγει. Σχεδιάστε έναν αλγόριθμο που βρίσκει το μέγιστο αριθμό νομισμάτων που μπορεί ένα ρομπότ να συλλέξει και το μονοπάτι που πρέπει να ακολουθήσει για να το επιτύχει.

Άσκηση 6

- Έστω $F(i,j)$ το μέγιστο πλήθος νομισμάτων που μπορεί ένα ρομπότ να συλλέξει και να τα φέρει στο κελί (i,j), της i-οστής γραμμής και j-ιοστής στήλης του πλέγματος.
- Μπορεί να προσεγγίσει το κελί (i,j) είτε από το πάνω κελί $(i-1,j)$ είτε από το αριστερό του κελί $(i,j-1)$.
- Ο μέγιστος αριθμός νομισμάτων που μπορεί να φέρει σε αυτά τα γειτονικά κελιά είναι $F(i-1,j)$ και $F(i,j-1)$ αντίστοιχα.
- Προφανώς για την πρώτη γραμμή ισχύει ότι δεν έχουν επάνω γειτονικά κελιά και για την πρώτη στήλη ότι δεν έχουν αριστερά γειτονικά κελιά. Επομένως για τα κελιά αυτά υποθέτουμε ότι ισχύει $F(i-1,j)=0$ και $F(i,j-1)=0$ για τους μη υπάρχοντες γείτονες.

Άσκηση 6

- Επομένως ο μέγιστος αριθμός νομισμάτων που μπορεί να φέρει το ρομπότ στο κελί (i, j) είναι ο μέγιστος αριθμός εκ των $F(i-1, j)$ και $F(i, j-1)$ συν ένα πιθανό νόμισμα που μπορεί να υπάρχει στο κελί (i, j).

$$F(i,j) = \max\{F(i-1,j), F(i,j-1)\} + c_{ij} \quad \gamma\iota\alpha \quad 1 \le i \le n, 1 \le j \le m$$
$$F(0,j) = 0 \quad \gamma\iota\alpha \; 1 \le j \le m$$
$$F(i,0) = 0 \quad \gamma\iota\alpha \; 1 \le i \le n$$

Όπου $c_{ij} = 1$ αν υπάρχει νόμισμα στο κελί (i, j) και $c_{ij} = 0$ σε αντίθετη περίπτωση.

- Χρησιμοποιώντας τους παραπάνω τύπους μπορούμε να γεμίσουμε ένα $m \times n$ πίνακα με τις τιμές $F(i, j)$ είτε γραμμή-γραμμή, είτε στήλη-στήλη.

Άσκηση 6

ALGORITHM *RobotCoinCollection*$(C[1..n, 1..m])$

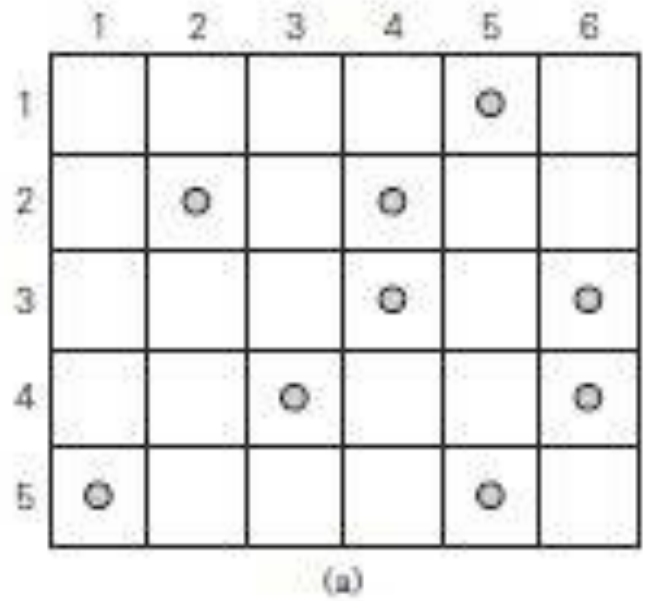
//Applies dynamic programming to compute the largest number of
//coins a robot can collect on an $n \times m$ board by starting at $(1, 1)$
//and moving right and down from upper left to down right corner
//Input: Matrix $C[1..n, 1..m]$ whose elements are equal to 1 and 0
//for cells with and without a coin, respectively
//Output: Largest number of coins the robot can bring to cell (n, m)

$F[1, 1] \leftarrow C[1, 1]$; **for** $j \leftarrow 2$ **to** m **do** $F[1, j] \leftarrow F[1, j-1] + C[1, j]$
for $i \leftarrow 2$ **to** n **do**
    $F[i, 1] \leftarrow F[i-1, 1] + C[i, 1]$
    for $j \leftarrow 2$ **to** m **do**
        $F[i, j] \leftarrow \max(F[i-1, j], F[i, j-1]) + C[i, j]$
return $F[n, m]$

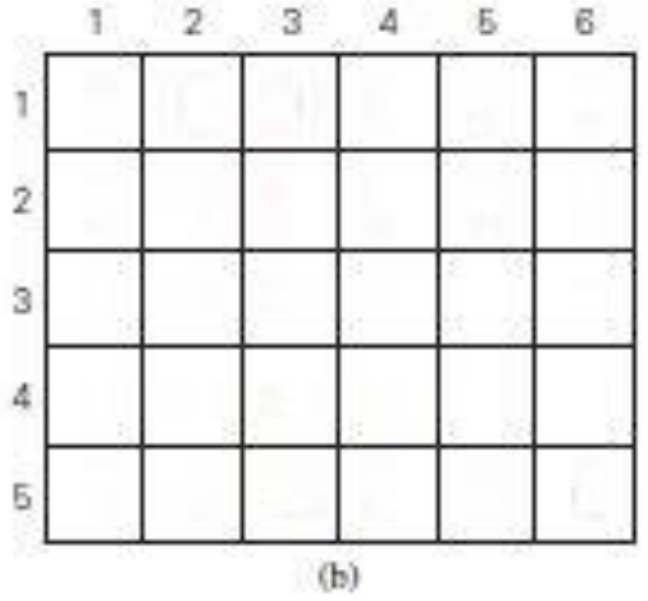
Πολυπλοκότητα Αλγορίθμου:
$\mathbf{O}(mn)$

Πως γίνεται όμως η ανάκτηση του βέλτιστου μονοπατιού;
Ξεκινάμε από την κάτω δεξιά γωνία του πλέγματος και αποφασίζω από ποιο από τα 2 γειτονικά κελιά μετέβει το ρομπότ συγκρίνοντας τις τιμές των $F(i-1,j)$ και $F(i,j-1)$. Προφανώς η μετάβαση έγινε από το κελί με τη μεγαλύτερη τιμή.
Πολυπλοκότητα: $\mathbf{O}(m+n)$

Άσκηση 6

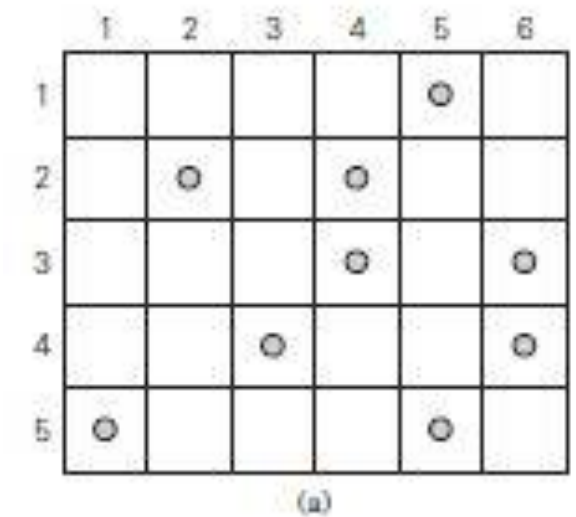


(a)



(b)

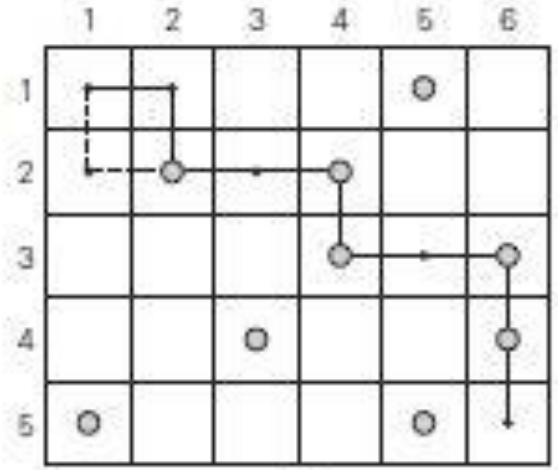
Άσκηση 6



(a)

	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	**5**

(b)



(c)

Δρομολόγηση με στόχο την ελαχιστοποίηση της αργοπορίας (lateness)

Ένας πόρος επεξεργάζεται ένα έργο τη φορά.

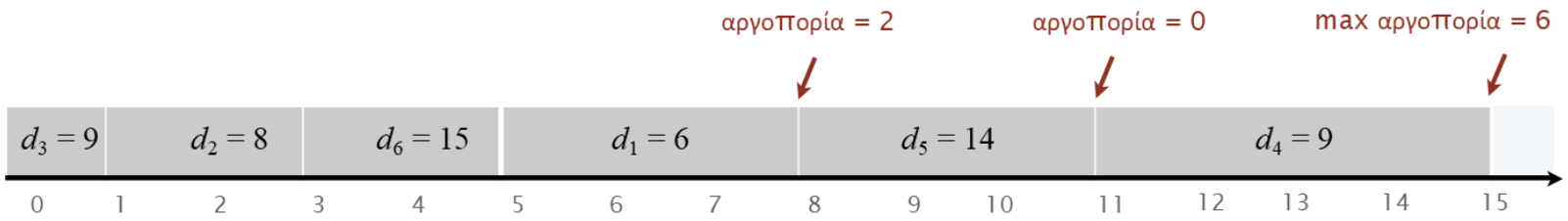
Το έργο j απαιτεί t_j χρονικές μονάδες επεξεργασίας και λήγει τη χρονική στιγμή d_j.

Αν το έργο j αρχίζει τη χρονική στιγμή s_j, ολοκληρώνεται τη χρονική στιγμή $f_j = s_j + t_j$.

Αργοπορία : $\ell_j = \max\{0, f_j - d_j\}$.

Στόχος: δρομολόγησε όλα τα έργα για την ελαχιστοποίηση της μέγιστής αργοπορίας $L = \max_j \ell_j$.

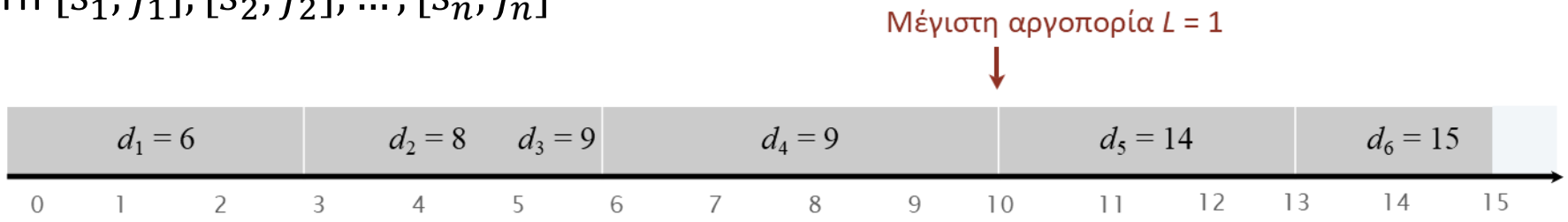
	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Ελαχιστοποίηση αργοπορίας: Αλγόριθμος earliest-deadline-first

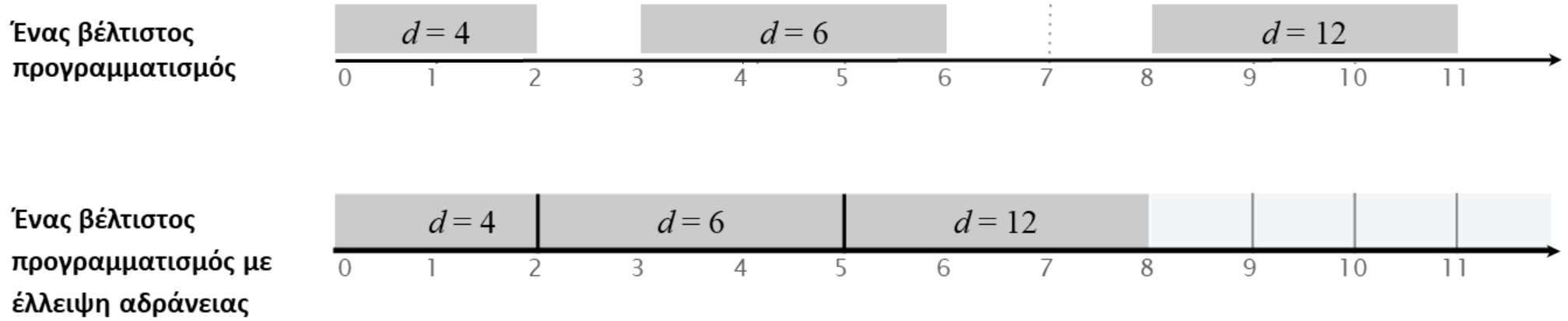
Earliest-Deadline-First$(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

1. Ταξινόμησε τα έργα με βάση το χρόνο που πρέπει να έχουν ολοκληρωθεί, αρίθμησε και πάλι τα έργα έτσι ώστε $d_1 \leq d_2 \leq \cdots \leq d_n$
2. $t \leftarrow 0$
3. for $j = 1$ to n
4. Προγραμμάτισε το έργο j στο χρονικό διάστημα $[t, t + t_j]$
5. $s_j \leftarrow t; f_j \leftarrow t + t_j; t \leftarrow t + t_j$
6. return $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$



Ελαχιστοποίηση αργοπορίας: έλλειψη αδρανούς διαστήματος

Παρατήρηση 1. Υπάρχει ένας βέλτιστος προγραμματισμός με μη αδρανή διαστήματα.



Παρατήρηση 2. Ο προγραμματισμός earliest-deadline-first δεν έχει αδρανή διαστήματα.

Ελαχιστοποίηση αργοπορίας: αναστροφές

Ορισμός: Δοθέντος ενός προγράμματος S, μία αναστροφή είναι ένα ζεύγος έργων i και j τέτοιο ώστε: $i < j$ αλλά το έργο j δρομολογείται πριν το i.

Ένα πρόγραμμα με αναστροφή

Σημείωση: υποθέτουμε ότι τα έργα αριθμούνται έτσι ώστε $d_1 \leq d_2 \leq \cdots \leq d_n$

Παρατήρηση 3: Η δρομολόγηση earliest-deadline-first είναι η μοναδική χρονο-δρομολόγηση χωρίς αδρανή διαστήματα και αναστροφές.

1	2	3	4	5	6	…	n

Ελαχιστοποίηση αργοπορίας: αναστροφές

Παρατήρηση 4: Αν ένα πρόγραμμα χωρίς αδρανή διαστήματα έχει μία αναστροφή, τότε έχει μία γειτονική αναστροφή.

Απόδειξη:

- Έστω $i-j$ η πλησιέστερη αναστροφή.
- Έστω k το έργο αμέσως δεξιά του j.
- Περίπτωση 1. $[j > k]$ Τότε $j-k$ είναι μία γειτονική αναστροφή.
- Περίπτωση 2. $[j < k]$ Τότε $i-k$ είναι πλησιέστερη αναστροφή αφού $i < j < k$

Ελαχιστοποίηση αργοπορίας: αναστροφές

Βασικός ισχυρισμός: Ανταλλάσσοντας δύο γειτονικά, αναστραμμένα έργα i και j μειώνει το πλήθος των αναστροφών κατά 1 και δεν προκαλεί αύξηση στη μέγιστη αργοπορία.

Έστω ℓ η αργοπορία πριν την ανταλλαγή, και ℓ' η αργοπορία μετά την ενέργεια αυτή.

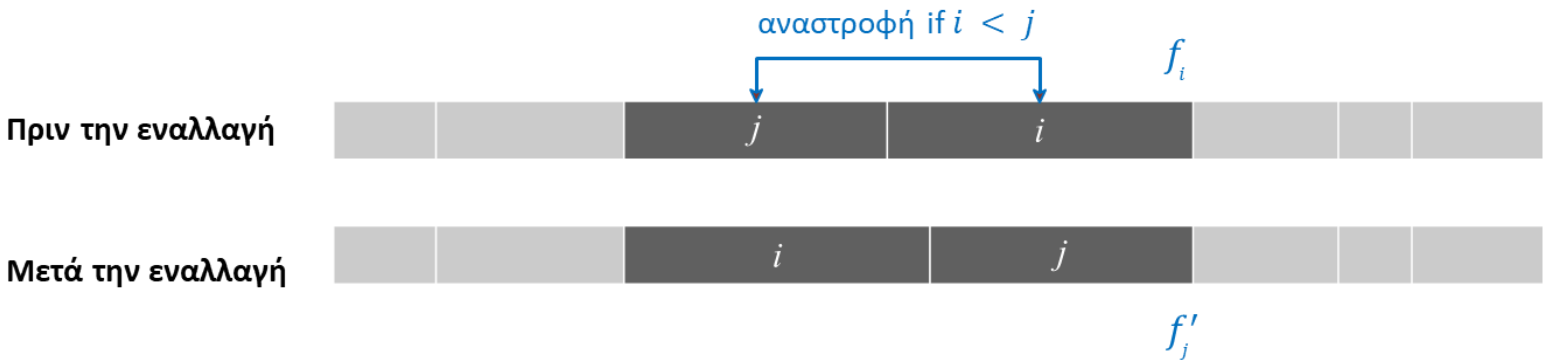
$\ell'_k = \ell_k \; \forall \; k \neq i,j \; , \ell'_i \leq \ell_i$

Αν το έργο j είναι αργοπορημένο, $\ell'_j = f'_j - d_j$ ← Ορισμός

$= f_i - d_j$ ← j τώρα τελειώνει τη χρονική στιγμή f_i

$\leq f_i - d_i$ ← $i < j \Rightarrow d_i \leq d_j$

$\leq l_i$ ← Ορισμός



Ελαχιστοποίηση αργοπορίας: ανάλυση αλγορίθμου earliest-deadline-first

Θεώρημα: Η χρόνο-δρομολόγηση earliest-deadline-first S είναι βέλτιστη.

Απόδειξη (με εις Άτοπο Απαγωγή):

Έστω S^* μία βέλτιστη χρόνο-δρομολόγηση με το λιγότερο πλήθος αναστροφών.

- S^* δεν έχει διαστήματα αδράνειας ← Παρατήρηση 1
- Περίπτωση 1. [S^* δεν έχει αναστροφές] Τότε $S = S^*$. ← Παρατήρηση 3
- Περίπτωση 2. [S^* έχει αναστροφή]
  - Έστω $i-j$ μία γειτονική αναστροφή ← Παρατήρηση 4
  - Η ανταλλαγή των έργων i και j μειώνει το πλήθος των αναστροφών κατά 1 χωρίς αύξηση της μέγιστης αργοπορίας ← Βασικός ισχυρισμός
  - Άτοπο γιατί η S^* έχει το ελάχιστο πλήθος αναστροφών

Πρόβλημα 16.1 (Βιβλίο “Introduction to Algorithms”, Cormen)

Εξετάστε το πρόβλημα της επιστροφής χρηματικού υπολοίπου (ρέστα), χρησιμοποιώντας τον μικρότερο αριθμό κερμάτων. Ας υποθέσουμε ότι η αξία κάθε νομίσματος είναι ακέραιος.

1. Περιγράψτε έναν άπληστο αλγόριθμο για να επιστρέψετε ρέστα που αποτελούνται από κέρματα των 1 cent, 5 cents, 10 cents και 25 cents. Δείξτε ότι ο αλγόριθμός σας αποδίδει μια βέλτιστη λύση.
2. Υποθέστε ότι τα διαθέσιμα νομίσματα έχουν ονομαστικές αξίες οι οποίες είναι δυνάμεις μιας σταθεράς c. Για παράδειγμα οι ονομαστικές αξίες $c^0, c^1, \cdots, c^k$ για $c > 1$ και $k \geq 1$. Δείξτε ότι ο άπληστος αλγόριθμος δίνει πάντα μια βέλτιστη λύση.
3. Δώστε ένα σετ νομισμάτων για τα οποία ο άπληστος αλγόριθμος δεν αποδίδει τη βέλτιστη λύση. Το σετ σας πρέπει να περιλαμβάνει νόμισμα του 1 cent έτσι ώστε να υπάρχει μια λύση για κάθε τιμή του n.

Πρόβλημα 16.1: Coin changing Ερώτημα 1

- Πάντα δώστε το υψηλότερο νόμισμα που μπορείτε χωρίς να ξεπεράσετε το χρηματικό ποσό που πρέπει να επιστρέψετε.
- Στη συνέχεια, επαναλάβετε αυτήν τη διαδικασία έως ότου το ποσό της υπόλοιπης αλλαγής μειωθεί στο 0.

Πρόβλημα 16.1: Coin changing Ερώτημα 2

- Έστω μία βέλτιστη λύση $(x_0, x_1, \cdots, x_k)$ όπου το x_i αντιστοιχεί στον αριθμό των νομισμάτων με ονομαστική αξία c^i.

- Θα δείξουμε πρώτα ότι πρέπει να ισχύει $\boldsymbol{x_i < c}$ **για κάθε** $\boldsymbol{i < k}$.

Έστω ότι είχαμε κάποιο $x_i \geq c$. Τί θα μπορούσε να συμβεί σε αυτή την περίπτωση;

Τότε θα μπορούσαμε να μειώσουμε το x_i κατά c και να αυξήσουμε το x_{i+1} κατά 1.

Αυτό το σύνολο νομισμάτων έχει την ίδια αξία και έχει $c - 1$ λιγότερα νομίσματα, επομένως η αρχική λύση δεν μπορεί να είναι η βέλτιστη. **ΑΤΟΠΟ**

- Αυτή η διαμόρφωση των νομισμάτων είναι ακριβώς η ίδια με εκείνη που θα δημιουργούσατε εάν επιλέγατε με απληστία το μεγαλύτερο δυνατό κέρμα.

Γιατί;;;

- Έστω c=2 και k=3
- Νομίσματα: 1,2,4,8
- Έστω βέλτιστη λύση (1,3,1,1)
- Αντί να δώσω 3 νομίσματα αξίας 2 (συνολική αξία 6) με συμφέρει να δώσω 1 νόμισμα αξίας 2 και 1 νόμισμα αξίας 4
- Επομένως η βέλτιστη λύση γίνεται (1,1,2,1)
- Αντί να δώσω 2 νομίσματα αξίας 4 (συνολική αξία 8) με συμφέρει να δώσω 1 νόμισμα αξίας 8
- Επομένως η βέλτιστη λύση (1,1,0,2)

Πρόβλημα 16.1: Coin changing Ερώτημα 2

- Με τον άπληστο αλγόριθμο, για να επιστρέψουμε ένα χρηματικό ποσό αξίας V, θα επιλέγαμε:

$$x_k = \lfloor V/c^k \rfloor = \lfloor Vc^{-k} \rfloor$$

Και για $i < k$ ισχύει $x_i = \lfloor (V mod c^{i+1}) c^{-i} \rfloor$

Αυτή είναι η μόνη λύση που ικανοποιεί τον περιορισμό ότι δεν μπορούν να υπάρχουν στη βέλτιστη λύση παραπάνω από c νομίσματα, εκτός από το νόμισμα με τη μεγαλύτερη ονομαστική αξία. Και αυτό διότι ο τύπος που δίνει τον αριθμό των νομισμάτων είναι μία δύναμη του c, με συντελεστή $V mod c^k$

$$x_k = \lfloor V/c^k \rfloor = \lfloor V c^{-k} \rfloor$$

x_k το πλήθος των νομισμάτων με τη μεγαλύτερη αξία

Και για $i < k$ ισχύει $x_i = \lfloor (V mod c^{i+1}) c^{-i} \rfloor$

Έστω ότι c=4 και k=2.

Νόμισμα 1ο : 1

Νόμισμα 2ο: 4 $x_1 = \lfloor (V \ mod \ 16)/4 \rfloor = \lfloor (V \ mod \ 4^2)/4^1 \rfloor$

Νόμισμα 3ο: 16 $x_2 = \lfloor V/16 \rfloor = \lfloor V/4^2 \rfloor$

Πρόβλημα 16.1: Coin changing Ερώτημα 3

- Έστω ότι οι ονομαστικές αξίες των νομισμάτων είναι {1, 3, 4} και το χρηματικό ποσό που πρέπει να επιστρέψετε είναι 6 ευρώ.

- Η άπληστη λύση θα είχε ως αποτέλεσμα το σύνολο κερμάτων {1, 1, 4} αλλά η βέλτιστη λύση θα ήταν {3, 3}.

- Ποσο 9
- Νόμισμα 1 : 1 2
- Νόμισμα 2 : 3 1
- Νόμισμα 3 : 6 1
- Νόμισμα 4 : 7 1