Αλγόριθμοι και Πολυπλοκότητα

Αρχοντία Γιαννοπούλου
Όλγα Φουρτουνέλλη

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Δυναμικός Προγραμματισμός

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

Αλγοριθμικά Μοντέλα

Απληστία. Χτίσε μια λύση σταδιακά, βελτιστοποιώντας μυωπικά κάποιο τοπικό κριτήριο.

Διαίρει και κυρίευε. Διάσπασε ένα πρόβλημα σε δύο υποπροβλήματα, λύσε κάθε υποπρόβλημα ανεξάρτητα, και συνδύασε τις λύσεις των υποπροβλημάτων για να δημιουργήσεις την λύση του αρχικού προβλήματος.

Αλγοριθμικά Μοντέλα

Απληστία. Χτίσε μια λύση σταδιακά, βελτιστοποιώντας μυωπικά κάποιο τοπικό κριτήριο.

Διαίρει και κυρίευε. Διάσπασε ένα πρόβλημα σε δύο υποπροβλήματα, λύσε κάθε υποπρόβλημα ανεξάρτητα, και συνδύασε τις λύσεις των υποπροβλημάτων για να δημιουργήσεις την λύση του αρχικού προβλήματος.

Δυναμικός προγραμματισμός.

Αλγοριθμικά Μοντέλα

Απληστία. Χτίσε μια λύση σταδιακά, βελτιστοποιώντας μυωπικά κάποιο τοπικό κριτήριο.

Διαίρει και κυρίευε. Διάσπασε ένα πρόβλημα σε δύο υποπροβλήματα, λύσε κάθε υποπρόβλημα ανεξάρτητα, και συνδύασε τις λύσεις των υποπροβλημάτων για να δημιουργήσεις την λύση του αρχικού προβλήματος.

Δυναμικός προγραμματισμός. Διάσπασε ένα πρόβλημα σε μια σειρά από επικαλυπτόμενα υποπροβλήματα, και δόμησε σωστές λύσεις για όλο και μεγαλύτερα υποπροβλήματα.

Ιστορία Δυναμικού Προγραμματισμού

Bellman. Ήταν πρωτοπόρος της συστηματικής μελέτης του δυναμικού προγραμματισμού την δεκαετία του 1950.

Ιστορία Δυναμικού Προγραμματισμού

Bellman. Ήταν πρωτοπόρος της συστηματικής μελέτης του δυναμικού προγραμματισμού την δεκαετία του 1950.

Ετυμολογία. (Dynamic Programming)

Ιστορία Δυναμικού Προγραμματισμού

Bellman. Ήταν πρωτοπόρος της συστηματικής μελέτης του δυναμικού προγραμματισμού την δεκαετία του 1950.

Ετυμολογία. (Dynamic Programming)
Δυναμικός Προγραμματισμός = σχεδιασμός με την πάροδο του χρόνου.

Ιστορία Δυναμικού Προγραμματισμού

Bellman. Ήταν πρωτοπόρος της συστηματικής μελέτης του δυναμικού προγραμματισμού την δεκαετία του 1950.

Ετυμολογία. (Dynamic Programming)
Δυναμικός Προγραμματισμός = σχεδιασμός με την πάροδο του χρόνου.

- Η τότε κυβέρνηση των ΗΠΑ ήταν αντίθετη με την έρευνα στο πεδίο των μαθηματικών.
- Ο Bellman αναζήτησε ένα εντυπωσιακό όνομα για να αποφύγει την αντιπαράθεση.
  - "it's impossible to use dynamic in a pejorative sense" (είναι αδύνατον να χρησιμοποιηθεί η λέξη *δυναμικός* με υποτιμητική έννοια)
  - "something not even a Congressman could object to" (κάτι στο οποίο δεν θα μπορούσε να διαφωνήσει ούτε ένα μέλος του κογκρέσου)

Αναφορά: Bellman, R. E. Eye of the Hurricane, An Autobiography.

Εφαρμογές Δυναμικού Προγραμματισμού

Περιοχές.

- Βιοπληροφορική.
- Θεωρία του ελέγχου.
- Θεωρία της πληροφορίας.
- Ερευνητικές δραστηριότητες.
- Επιστήμη υπολογιστών: θεωρία, γραφικά, τεχνητή νοημοσύνη, συστήματα.

Μερικοί **διάσημοι αλγόριθμοι** δυναμικού προγραμματισμού.

- Viterbi για κρυμμένα Μαρκοβιανά μοντέλα.
- Unix diff για σύγκριση δύο αρχείων.
- Smith-Waterman για ευθυγράμμιση ακολουθιών.
- Bellman-Ford για δρομολόγηση συντομότερης διαδρομής σε δίκτυα.
- Cocke-Kasami-Younger για ανάλυση γραμματικών χωρίς συμφραζόμενα.

Η Ακολουθία Fibonacci

- Η ακολουθία Fibonacci ορίζεται μαθηματικά με τον ακόλουθο αναδρομικό τύπο:

$$F(n) = F(n-1) + F(n-2).$$

- Με τις βασικές αρχικές συνθήκες: $F(0) = 0$ και $F(1) = 1$.

- Η μορφή αυτού του τύπου καθιστά την ακολουθία το κλασικότερο ίσως παράδειγμα για την εισαγωγή στην **αναδρομή** στην Πληροφορική.

Η Αφελής Αναδρομική Προσέγγιση

- Η απευθείας μεταφορά του μαθηματικού τύπου σε ψευδογλώσσα φαίνεται κάπως έτσι:

Διαδικασία fib(n)
Εάν $n \leq 1$
    Επίστρεψε n
Επίστρεψε fib($n-1$) + fib($n-2$)

- **Το Πρόβλημα:** Αν και ο κώδικας είναι κομψός, έχει εκθετική χρονική πολυπλοκότητα $\approx O(2^n)$. Αυτό τον καθιστά απαγορευτικό για μεγάλες τιμές του n.

Το Πρόβλημα: Επικάλυψη Υποπροβλημάτων

- Γιατί καθυστερεί τόσο;

Το Πρόβλημα: Επικάλυψη Υποπροβλημάτων

- Γιατί καθυστερεί τόσο; Επειδή υπολογίζει τα ίδια πράγματα ξανά και ξανά!
- Ας δούμε τι συμβαίνει κατά τον υπολογισμό του $F(5)$:

Το Πρόβλημα: Επικάλυψη Υποπροβλημάτων

- Γιατί καθυστερεί τόσο; Επειδή υπολογίζει τα ίδια πράγματα ξανά και ξανά!
- Ας δούμε τι συμβαίνει κατά τον υπολογισμό του $F(5)$:

$$F(5) = F(4) + F(3)$$
$$F(4) = F(3) + F(2)$$
$$F(3) = F(2) + F(1)$$

- Το $F(3)$ υπολογίζεται **2** φορές.
- Το $F(2)$ υπολογίζεται **3** φορές.
- Καθώς το n μεγαλώνει, το δέντρο αναδρομής εκρήγνυται. (Έχουμε πολλά επικαλυπτόμενα προβλήματα).

Η Λύση: Δυναμικός Προγραμματισμός & Μεμοιζατιον

- Η ιδέα του **Δυναμικού Προγραμματισμού** (Dynamic Programming) έρχεται να λύσει ακριβώς αυτό το πρόβλημα.

- **Απομνημόνευση (Memoization):** Κάθε φορά που υπολογίζουμε έναν όρο $F(k)$, δεν τον ξεχνάμε. Τον αποθηκεύουμε σε μια δομή δεδομένων (π.χ. έναν πίνακα.

- Όταν η αναδρομή ζητήσει ξανά το $F(k)$, αντί να ξεκινήσουμε το δέντρο υπολογισμών από την αρχή, **επιστρέφουμε απλώς την αποθηκευμένη τιμή**.

Υλοποίηση με Απομνημόνευση

- Ο κώδικάς μας μεταμορφώνεται προσθέτοντας απλώς έναν πίνακα:

Υλοποίηση με Απομνημόνευση

- Ο κώδικάς μας μεταμορφώνεται προσθέτοντας απλώς έναν πίνακα:

Διαδικασία fib2(n)
Εάν $n \leq 1$
    Επίστρεψε n
Αρχικοποίησε πίνακα $A[0, \ldots, n]$
$A[0] \leftarrow 0$
$A[1] \leftarrow 1$
Για κάθε $2 \leq i \leq n$
    $A[i] \leftarrow A[i-1] + A[i-2]$
Επίστρεψε $A[n]$

Υλοποίηση με Απομνημόνευση

- Ο κώδικάς μας μεταμορφώνεται προσθέτοντας απλώς έναν πίνακα:

Διαδικασία fib2(n)
Εάν $n \leq 1$
    Επίστρεψε n
Αρχικοποίησε πίνακα $A[0, \ldots, n]$
$A[0] \leftarrow 0$
$A[1] \leftarrow 1$
Για κάθε $2 \leq i \leq n$
    $A[i] \leftarrow A[i-1] + A[i-2]$
Επίστρεψε $A[n]$

- **Αποτέλεσμα:** Η χρονική πολυπλοκότητα πέφτει κατακόρυφα από εκθετική $O(2^n)$ σε γραμμική $O(n)$. Το πρόβλημα πλέον λύνεται γρήγορα!

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Σύνολο αιτημάτων ή εργασιών $\{1, \ldots, n\}$.

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Σύνολο αιτημάτων ή εργασιών $\{1, \ldots, n\}$.
- Το αίτημα j ξεκινά την στιγμή s_j, τελειώνει την στιγμή f_j, και έχει βαρύτητα v_j.

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

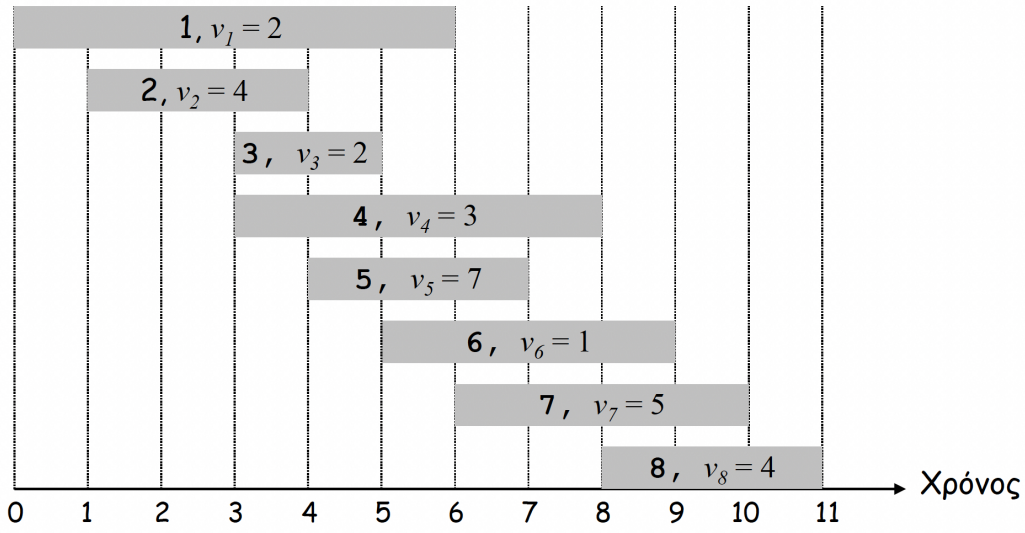
- Σύνολο αιτημάτων ή εργασιών $\{1, \ldots, n\}$.
- Το αίτημα j ξεκινά την στιγμή s_j, τελειώνει την στιγμή f_j, και έχει βαρύτητα v_j.
- Δύο αιτήματα είναι συμβατά αν δεν επικαλύπτονται.

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Σύνολο αιτημάτων ή εργασιών $\{1, \ldots, n\}$.
- Το αίτημα j ξεκινά την στιγμή s_j, τελειώνει την στιγμή f_j, και έχει βαρύτητα v_j.
- Δύο αιτήματα είναι συμβατά αν δεν επικαλύπτονται.
- Στόχος: εύρεση υποσυνόλου $S \subseteq \{1, \ldots, n\}$ συμβατών αιτημάτων μέγιστης βαρύτητας $\sum_{i \in S} v_i$.

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Σύνολο αιτημάτων ή εργασιών $\{1, \ldots, n\}$.
- Το αίτημα j ξεκινά την στιγμή s_j, τελειώνει την στιγμή f_j, και έχει βαρύτητα v_j.
- Δύο αιτήματα είναι συμβατά αν δεν επικαλύπτονται.
- Στόχος: εύρεση υποσυνόλου $S \subseteq \{1, \ldots, n\}$ συμβατών αιτημάτων μέγιστης βαρύτητας $\sum_{i \in S} v_i$.



Υπενθύμιση

Αν όλες οι βαρύτητες είναι 1 τότε το πρόβλημα λύνεται με άπληστο αλγόριθμο.

Υπενθύμιση

Αν όλες οι βαρύτητες είναι 1 τότε το πρόβλημα λύνεται με άπληστο αλγόριθμο.

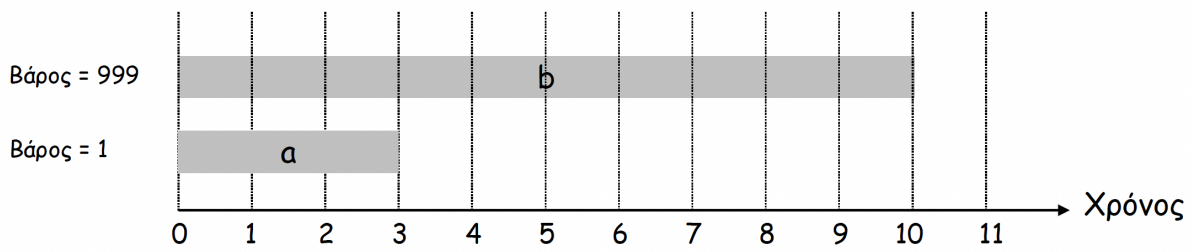
- Θεωρούμε τα αιτήματα σε αύξουσα σειρά ως προς το χρόνο λήξης.
- Προσθέτουμε το αίτημα στο υποσύνολο αν είναι συμβατό με τα αιτήματα που έχουν ήδη επιλεχθεί.

Υπενθύμιση

Αν όλες οι βαρύτητες είναι 1 τότε το πρόβλημα λύνεται με άπληστο αλγόριθμο.

- Θεωρούμε τα αιτήματα σε αύξουσα σειρά ως προς το χρόνο λήξης.
- Προσθέτουμε το αίτημα στο υποσύνολο αν είναι συμβατό με τα αιτήματα που έχουν ήδη επιλεχθεί.

Παρατήρηση. Ο άπληστος αλγόριθμος μπορεί να αποτύχει θεαματικά αν επιτραπούν αυθαίρετες βαρύτητες.



Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Διατάσσουμε τα αιτήματα σύμφωνα με το χρόνο λήξης τους:

$$f_1 \leq f_2 \leq \cdots \leq f_n$$

- $p(j) =$ μεγαλύτερος δείκτης $i < j$ τέτοιος ώστε το αίτημα i να είναι συμβατό με το j.

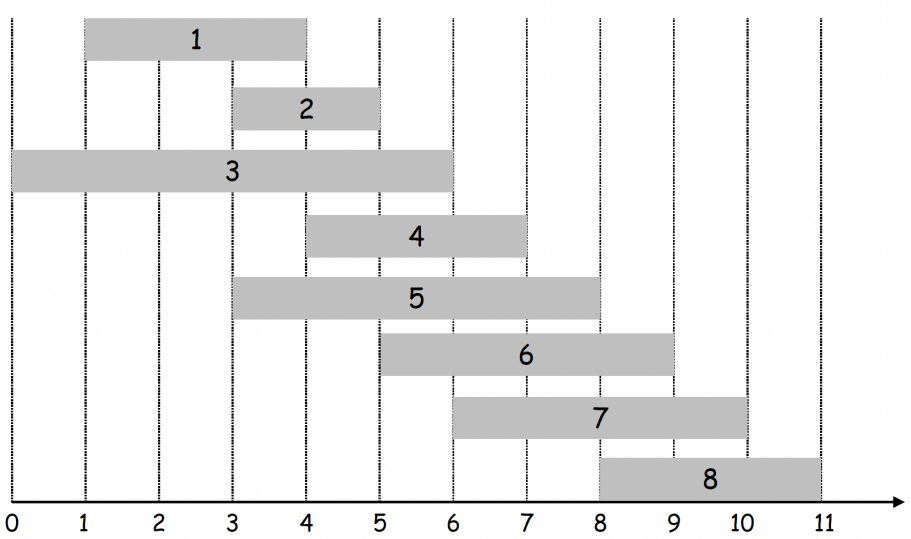
Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Διατάσσουμε τα αιτήματα σύμφωνα με το χρόνο λήξης τους:

$$f_1 \leq f_2 \leq \cdots \leq f_n$$

- $p(j) =$ μεγαλύτερος δείκτης $i < j$ τέτοιος ώστε το αίτημα i να είναι συμβατό με το j.

Παράδειγμα: $p(8) = 5$, $p(7) = 3$, $p(2) = 0$.



Δυναμικός Προγραμματισμός: Δυαδική επιλογή

$\mathrm{OPT}(j) =$ τιμή της βέλτιστης λύσης προβλήματος αποτελούμενο από αιτήματα $1, 2, \ldots, j$

Δυναμικός Προγραμματισμός: Δυαδική επιλογή

$\mathrm{OPT}(j) =$ τιμή της βέλτιστης λύσης προβλήματος αποτελούμενο από αιτήματα $1, 2, \ldots, j$

- Το αίτημα j με βαρύτητα v_j είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$

- Το αίτημα j με βαρύτητα v_j δεν είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$

Δυναμικός Προγραμματισμός: Δυαδική επιλογή

$\mathrm{OPT}(j) =$ τιμή της βέλτιστης λύσης προβλήματος αποτελούμενο από αιτήματα $1, 2, \ldots, j$

- Το αίτημα j με βαρύτητα v_j είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$
  - Δεν μπορούν να χρησιμοποιηθούν τα ασύμβατα αιτήματα $\{p(j)+1, p(j)+2, \ldots, j-1\}$

- Το αίτημα j με βαρύτητα v_j δεν είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$

Δυναμικός Προγραμματισμός: Δυαδική επιλογή

$\mathrm{OPT}(j) =$ τιμή της βέλτιστης λύσης προβλήματος αποτελούμενο από αιτήματα $1, 2, \ldots, j$

- Το αίτημα j με βαρύτητα v_j είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$
  - Δεν μπορούν να χρησιμοποιηθούν τα ασύμβατα αιτήματα $\{p(j)+1, p(j)+2, \ldots, j-1\}$
  - Πρέπει να περιέχει την βέλτιστη λύση $\mathrm{OPT}(p(j))$ στο πρόβλημα που αποτελείται από τα συμβατά αιτήματα $\{1, 2, \ldots, p(j)\}$ που έχουν απομείνει
- Το αίτημα j με βαρύτητα v_j δεν είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$

Δυναμικός Προγραμματισμός: Δυαδική επιλογή

$\mathrm{OPT}(j) =$ τιμή της βέλτιστης λύσης προβλήματος αποτελούμενο από αιτήματα $1, 2, \ldots, j$

- Το αίτημα j με βαρύτητα v_j είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$
  - Δεν μπορούν να χρησιμοποιηθούν τα ασύμβατα αιτήματα $\{p(j)+1, p(j)+2, \ldots, j-1\}$
  - Πρέπει να περιέχει την βέλτιστη λύση $\mathrm{OPT}(p(j))$ στο πρόβλημα που αποτελείται από τα συμβατά αιτήματα $\{1, 2, \ldots, p(j)\}$ που έχουν απομείνει

- Το αίτημα j με βαρύτητα v_j δεν είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$
  - Πρέπει να περιέχει την βέλτιστη λύση $\mathrm{OPT}(j-1)$ στο πρόβλημα που αποτελείται από τα συμβατά αιτήματα $\{1, 2, ..., j-1\}$ που έχουν απομείνει

Δυναμικός Προγραμματισμός: Δυαδική επιλογή

$\mathrm{OPT}(j) =$ τιμή της βέλτιστης λύσης προβλήματος αποτελούμενο από αιτήματα $1, 2, \ldots, j$

- Το αίτημα j με βαρύτητα v_j είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$
  - Δεν μπορούν να χρησιμοποιηθούν τα ασύμβατα αιτήματα $\{p(j)+1, p(j)+2, \ldots, j-1\}$
  - Πρέπει να περιέχει την βέλτιστη λύση $\mathrm{OPT}(p(j))$ στο πρόβλημα που αποτελείται από τα συμβατά αιτήματα $\{1, 2, \ldots, p(j)\}$ που έχουν απομείνει

- Το αίτημα j με βαρύτητα v_j δεν είναι μέρος της βέλτιστης λύσης $\mathrm{OPT}(j)$
  - Πρέπει να περιέχει την βέλτιστη λύση $\mathrm{OPT}(j-1)$ στο πρόβλημα που αποτελείται από τα συμβατά αιτήματα $\{1, 2, ..., j-1\}$ που έχουν απομείνει

$$\mathrm{OPT}(j) = \begin{cases} 0 & j = 0 \\ \max\{v_j + \mathrm{OPT}(p(j)), \mathrm{OPT}(j-1)\} & j \geq 1 \end{cases}$$

Ωμή βία

Είσοδος: n, s_1, s_2, $\ldots$, s_n, f_1, f_2, $\ldots$, f_n, v_1, v_2, $\ldots$, v_n

Ταξινόμησε τα αιτήματα κατά το χρόνο λήξης $f_1 \leq f_2 \leq \cdots \leq f_n$

Υπολόγισε τα $p(1)$, $p(2)$, $\ldots$, $p(n)$
Υπολόγισε τα ComputeOPT(0), ComputeOPT(1), $\ldots$, ComputeOPT(n)

Ωμή βία

Είσοδος: n, s_1, s_2, $\ldots$, s_n, f_1, f_2, $\ldots$, f_n, v_1, v_2, $\ldots$, v_n

Ταξινόμησε τα αιτήματα κατά το χρόνο λήξης $f_1 \leq f_2 \leq \cdots \leq f_n$

Υπολόγισε τα $p(1)$, $p(2)$, $\ldots$, $p(n)$
Υπολόγισε τα ComputeOPT(0), ComputeOPT(1), $\ldots$, ComputeOPT(n)

ComputeOPT(j)
εάν $j = 0$
    επίστρεψε 0
αλλιώς
    επίστρεψε $\max\{v_j + \text{ComputeOPT}(p(j)), \text{ComputeOPT}(j-1)\}$

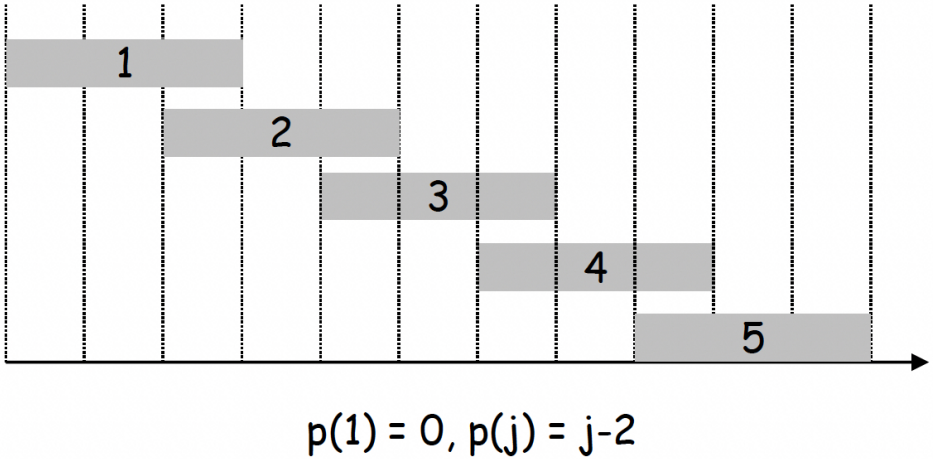
Ωμή βία

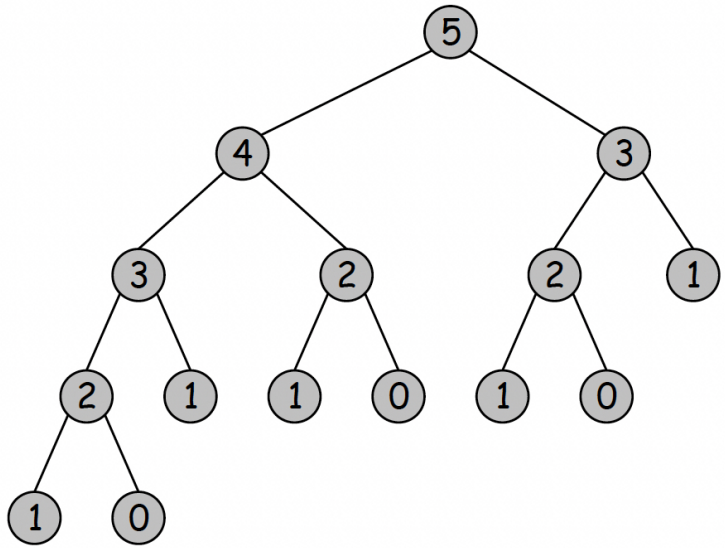
Παρατήρηση. Ο αναδρομικός αλγόριθμος αποτυγχάνει θεαματικά λόγω επανϋπολογισμού πλεοναζόντων υποπροβλημάτων (εκθετικός αλγόριθμος).

Ωμή βία

Παρατήρηση. Ο αναδρομικός αλγόριθμος αποτυγχάνει θεαματικά λόγω επανϋπολογισμού πλεοναζόντων υποπροβλημάτων (εκθετικός αλγόριθμος).

Παράδειγμα. Ο αριθμός των αναδρομικών κλήσεων στην παρακάτω περίπτωση μεγαλώνει σαν μια ακολουθία Fibonacci $T(n) = T(n-1) + T(n-2)$ (δηλαδή εκθετική).





Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- **Παρατήρηση 1.** Ο ComputeOPT επιλύει $n + 1$ διαφορετικά υποπροβλήματα ComputeOPT(0), ComputeOPT(1), ..., ComputeOPT(n).

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Παρατήρηση 1. Ο ComputeOPT επιλύει $n + 1$ διαφορετικά υποπροβλήματα ComputeOPT(0), ComputeOPT(1), ..., ComputeOPT(n).

- Παρατήρηση 2. Η εκθετική πολυπλοκότητα του ComputeOPT οφείλεται στον εντυπωσιακό πλεονασμό κλήσεων κάθε ενός από τα ComputeOPT(j) (τα επανυπολογίζουμε ξανά και ξανά κάθε φορά).

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Παρατήρηση 1. Ο ComputeOPT επιλύει $n + 1$ διαφορετικά υποπροβλήματα ComputeOPT(0), ComputeOPT(1), ..., ComputeOPT(n).

- Παρατήρηση 2. Η εκθετική πολυπλοκότητα του ComputeOPT οφείλεται στον εντυπωσιακό πλεονασμό κλήσεων κάθε ενός από τα ComputeOPT(j) (τα επανυπολογίζουμε ξανά και ξανά κάθε φορά).

- Ερώτημα. Πως απαλείφουμε αυτόν τον πλεονασμό;

Σταθμισμένος Χρονοπρογραμματισμός Διαστημάτων

- Παρατήρηση 1. Ο ComputeOPT επιλύει $n + 1$ διαφορετικά υποπροβλήματα ComputeOPT(0), ComputeOPT(1), ..., ComputeOPT(n).

- Παρατήρηση 2. Η εκθετική πολυπλοκότητα του ComputeOPT οφείλεται στον εντυπωσιακό πλεονασμό κλήσεων κάθε ενός από τα ComputeOPT(j) (τα επανυπολογίζουμε ξανά και ξανά κάθε φορά).

- Ερώτημα. Πως απαλείφουμε αυτόν τον πλεονασμό;

Απομνημονεύουμε!

Απομνημόνευση

- Αποθήκευσε τα αποτελέσματα κάθε υποπροβλήματος σε μια καθολικά προσπελάσιμη θέση μνήμης.
- Αναζήτηση όποτε χρειαστεί.

Απομνημόνευση

- Αποθήκευσε τα αποτελέσματα κάθε υποπροβλήματος σε μια καθολικά προσπελάσιμη θέση μνήμης.
- Αναζήτηση όποτε χρειαστεί.

Είσοδος: n, s_1, s_2, $\ldots$, s_n, f_1, f_2, $\ldots$, f_n, v_1, v_2, $\ldots$, v_n

Ταξινόμησε τα αιτήματα κατά το χρόνο λήξης $f_1 \leq f_2 \leq \cdots \leq f_n$.

Υπολόγισε τα $p(1)$, $p(2)$, $\ldots$, $p(n)$.

Για κάθε $j = 1$ έως n
    $M[j] =$ κενό
$M[0] = 0$
MComputeOPT(n)

MComputeOPT(j)
εάν $M[j] =$ κενό
    $M[j] = \max\{v_j + M[p(j)], M[j-1]\}$
επίστρεψε $M[j]$

Χρόνος εκτέλεσης

Ισχυρισμός. Η εκδοχή με απομνημόνευση του αλγόριθμου παίρνει χρόνο $\mathcal{O}(n \log n)$.

- Ταξινόμηση ως προς χρόνο λήξης: $\mathcal{O}(n \log n)$.

Χρόνος εκτέλεσης

Ισχυρισμός. Η εκδοχή με απομνημόνευση του αλγόριθμου παίρνει χρόνο $\mathcal{O}(n \log n)$.

- Ταξινόμηση ως προς χρόνο λήξης: $\mathcal{O}(n \log n)$.
- Υπολογισμός $p(j)$: $\mathcal{O}(n)$ μετά από ταξινόμηση ως προς τον χρόνο έναρξης.

Χρόνος εκτέλεσης

Ισχυρισμός. Η εκδοχή με απομνημόνευση του αλγόριθμου παίρνει χρόνο $\mathcal{O}(n \log n)$.

- Ταξινόμηση ως προς χρόνο λήξης: $\mathcal{O}(n \log n)$.
- Υπολογισμός $p(j)$: $\mathcal{O}(n)$ μετά από ταξινόμηση ως προς τον χρόνο έναρξης.
- MComputeOPT(j): Κάθε κλήση απαιτεί χρόνο $\mathcal{O}(1)$ και είτε
  - (i) επιστρέφει μια υπάρχουσα τιμή
  - (ii) είτε συμπληρώνει μία νέα εγγραφή $M[j]$ και εκτελεί δύο αναδρομικές κλήσεις.

Χρόνος εκτέλεσης

Ισχυρισμός. Η εκδοχή με απομνημόνευση του αλγόριθμου παίρνει χρόνο $\mathcal{O}(n \log n)$.

- Ταξινόμηση ως προς χρόνο λήξης: $\mathcal{O}(n \log n)$.
- Υπολογισμός $p(j)$: $\mathcal{O}(n)$ μετά από ταξινόμηση ως προς τον χρόνο έναρξης.
- MComputeOPT(j): Κάθε κλήση απαιτεί χρόνο $\mathcal{O}(1)$ και είτε
  - (i) επιστρέφει μια υπάρχουσα τιμή
  - (ii) είτε συμπληρώνει μία νέα εγγραφή $M[j]$ και εκτελεί δύο αναδρομικές κλήσεις.
- Μέτρο προόδου Φ: πλήθος μη-κενών εγγραφών του $M[]$.
  - Αρχικά $\Phi = 0$ και κατά τη διάρκεια της εκτέλεσης $\Phi \leq n$
  - Το (ii) αυξάνει το Φ κατά 1 κάνοντας το πολύ 2 αναδρομικές κλήσεις

Χρόνος εκτέλεσης

Ισχυρισμός. Η εκδοχή με απομνημόνευση του αλγόριθμου παίρνει χρόνο $\mathcal{O}(n \log n)$.

- Ταξινόμηση ως προς χρόνο λήξης: $\mathcal{O}(n \log n)$.
- Υπολογισμός $p(j)$: $\mathcal{O}(n)$ μετά από ταξινόμηση ως προς τον χρόνο έναρξης.
- $\mathrm{MComputeOPT}(j)$: Κάθε κλήση απαιτεί χρόνο $\mathcal{O}(1)$ και είτε
  - (i) επιστρέφει μια υπάρχουσα τιμή
  - (ii) είτε συμπληρώνει μία νέα εγγραφή $M[j]$ και εκτελεί δύο αναδρομικές κλήσεις.
- Μέτρο προόδου Φ: πλήθος μη-κενών εγγραφών του $M[]$.
  - Αρχικά $\Phi = 0$ και κατά τη διάρκεια της εκτέλεσης $\Phi \leq n$
  - Το (ii) αυξάνει το Φ κατά 1 κάνοντας το πολύ 2 αναδρομικές κλήσεις
- Ο συνολικός χρόνος εκτέλεσης του $\mathrm{MComputeOPT}(n)$ είναι $\mathcal{O}(n)$.

Χρόνος εκτέλεσης

Ισχυρισμός. Η εκδοχή με απομνημόνευση του αλγόριθμου παίρνει χρόνο $\mathcal{O}(n \log n)$.

- Ταξινόμηση ως προς χρόνο λήξης: $\mathcal{O}(n \log n)$.
- Υπολογισμός $p(j)$: $\mathcal{O}(n)$ μετά από ταξινόμηση ως προς τον χρόνο έναρξης.
- $\mathrm{MComputeOPT}(j)$: Κάθε κλήση απαιτεί χρόνο $\mathcal{O}(1)$ και είτε
  - (i) επιστρέφει μια υπάρχουσα τιμή
  - (ii) είτε συμπληρώνει μία νέα εγγραφή $M[j]$ και εκτελεί δύο αναδρομικές κλήσεις.
- Μέτρο προόδου Φ: πλήθος μη-κενών εγγραφών του $M[]$.
  - Αρχικά $\Phi = 0$ και κατά τη διάρκεια της εκτέλεσης $\Phi \leq n$
  - Το (ii) αυξάνει το Φ κατά 1 κάνοντας το πολύ 2 αναδρομικές κλήσεις
- Ο συνολικός χρόνος εκτέλεσης του $\mathrm{MComputeOPT}(n)$ είναι $\mathcal{O}(n)$.

Αν μας δώσουν τα αιτήματα ταξινομημένα κατά σειρά έναρξης και σειρά λήξης ποιά είναι η πολυπλοκότητα του αλγόριθμου;

Εύρεση Λύσης

Ο αλγόριθμος υπολογίζει τη μέγιστη τιμή. Πώς βρίσκουμε τη λύση που αντιστοιχεί σε αυτή την τιμή;

Εύρεση Λύσης

Ο αλγόριθμος υπολογίζει τη μέγιστη τιμή. Πώς βρίσκουμε τη λύση που αντιστοιχεί σε αυτή την τιμή;

Εκτελούμε ένα επιπλέον βήμα μετεπεξεργασίας του M, «ινχηλατώντας» τη λύση.

Εύρεση Λύσης

Ο αλγόριθμος υπολογίζει τη μέγιστη τιμή. Πώς βρίσκουμε τη λύση που αντιστοιχεί σε αυτή την τιμή;

Εκτελούμε ένα επιπλέον βήμα μετεπεξεργασίας του M, «ινχηλατώντας» τη λύση.

Βρες-Λύση(j)
εάν $j = 0$
    μην εκτυπώσεις τίποτα
αλλιώς
    εάν $(v_j + M[p(j)]) > M[j-1]$
        εκτύπωσε το j
        Βρες-Λύση$(p(j))$
    αλλιώς
        Βρες-Λύση$(j-1)$

Εύρεση Λύσης

Ο αλγόριθμος υπολογίζει τη μέγιστη τιμή. Πώς βρίσκουμε τη λύση που αντιστοιχεί σε αυτή την τιμή;

Εκτελούμε ένα επιπλέον βήμα μετεπεξεργασίας του M, «ινχηλατώντας» τη λύση.

Βρες-Λύση(j)
εάν $j = 0$
    μην εκτυπώσεις τίποτα
αλλιώς
    εάν $(v_j + M[p(j)]) > M[j-1]$
        εκτύπωσε το j
        Βρες-Λύση$(p(j))$
    αλλιώς
        Βρες-Λύση$(j-1)$

Πλήθος επαναληπτικών κλήσεων $\leq n$ και άρα παίρνει $\mathcal{O}(n)$ χρόνο.

Διαφορετικά

Είσοδος: n, s_1, s_2, $\ldots$, s_n, f_1, f_2, $\ldots$, f_n, v_1, v_2, $\ldots$, v_n

Ταξινόμησε τα αιτήματα κατά το χρόνο λήξης $f_1 \leq f_2 \leq \cdots \leq f_n$

Υπολόγισε τα $p(1)$, $p(2)$, $\ldots$, $p(n)$

$M[0] = 0$
Για κάθε $j = 1$ έως n
$\quad M[j] = \max\{v_j + M[p(j)], M[j-1]\}$

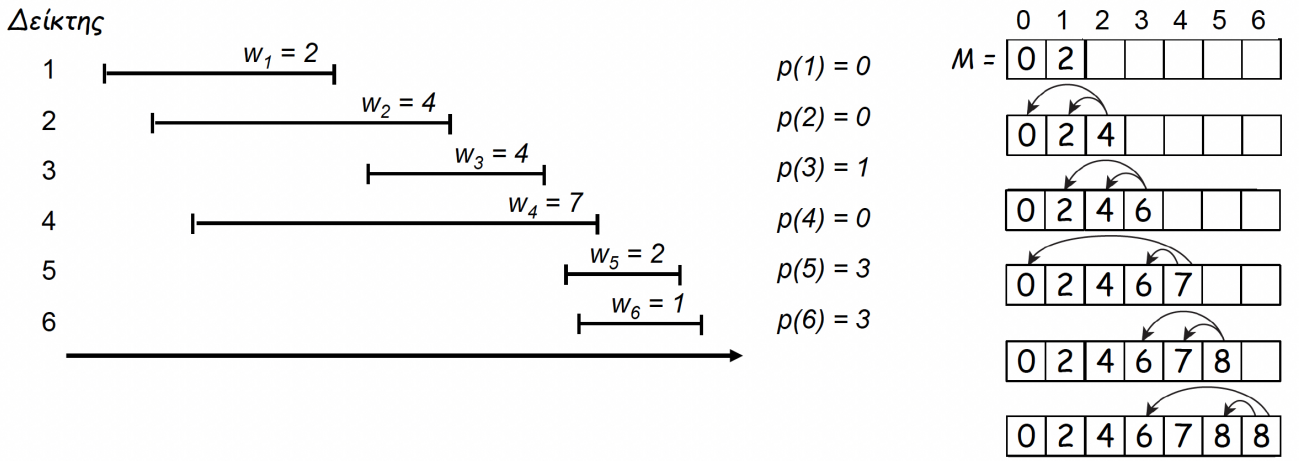
Διαφορετικά

Είσοδος: $n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n, v_1, v_2, \ldots, v_n$

Ταξινόμησε τα αιτήματα κατά το χρόνο λήξης $f_1 \leq f_2 \leq \cdots \leq f_n$

Υπολόγισε τα $p(1), p(2), \ldots, p(n)$

$M[0] = 0$
Για κάθε $j = 1$ έως n
$\quad M[j] = \max\{v_j + M[p(j)], M[j-1]\}$



Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$

Έστω $s_1, s_2, ..., s_n$ αύξουσα αρίθμηση των αιτημάτων με βάση το χρόνο έναρξης και $f_1, f_2, ..., f_n$ αύξουσα αρίθμηση των αιτημάτων με βάση το χρόνο λήξης.

Ας θεωρήσουμε και μία εργασία 0 με χρόνο έναρξης και λήξης το 0.

$i \leftarrow 1, j \leftarrow 1$
Όσω $i \leq n$ και $j \leq n$.
    εάν λήξη$(f_i) \leq$ έναρξη(s_j)
        $i \leftarrow i + 1$
    αλλιώς εάν λήξη$(f_i) >$ έναρξη(s_j)
        $p(s_j) \leftarrow f_{i-1}$
        $j \leftarrow j + 1$

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$

Έστω $s_1, s_2, ..., s_n$ αύξουσα αρίθμηση των αιτημάτων με βάση το χρόνο έναρξης και $f_1, f_2, ..., f_n$ αύξουσα αρίθμηση των αιτημάτων με βάση το χρόνο λήξης.

Ας θεωρήσουμε και μία εργασία 0 με χρόνο έναρξης και λήξης το 0.

$i \leftarrow 1, j \leftarrow 1$
Όσω $i \leq n$ και $j \leq n$.
    εάν λήξη$(f_i) \leq$ έναρξη(s_j)
        $i \leftarrow i + 1$
    αλλιώς εάν λήξη$(f_i) >$ έναρξη(s_j)
        $p(s_j) \leftarrow f_{i-1}$
        $j \leftarrow j + 1$

Εάν $i - 1 = 0$ τότε ορίσαμε $s_0 = f_0 = 0$.

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$

Έστω $s_1, s_2, ..., s_n$ αύξουσα αρίθμηση των αιτημάτων με βάση το χρόνο έναρξης και $f_1, f_2, ..., f_n$ αύξουσα αρίθμηση των αιτημάτων με βάση το χρόνο λήξης.

Ας θεωρήσουμε και μία εργασία 0 με χρόνο έναρξης και λήξης το 0.

$i \leftarrow 1, j \leftarrow 1$
Όσω $i \leq n$ και $j \leq n$.
    εάν λήξη$(f_i) \leq$ έναρξη(s_j)
        $i \leftarrow i + 1$
    αλλιώς εάν λήξη$(f_i) >$ έναρξη(s_j)
        $p(s_j) \leftarrow f_{i-1}$
        $j \leftarrow j + 1$

Εάν $i - 1 = 0$ τότε ορίσαμε $s_0 = f_0 = 0$.

Η παραπάνω διαδικασία εκτελείται σε χρόνο $\mathcal{O}(n)$.

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$ (ορθότητα)

Η παραπάνω διαδικασία επιστρέφει σωστά την τιμή $p(k)$ για κάθε $k \in \{1, 2, \ldots, n\}$.

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$ (ορθότητα)

Η παραπάνω διαδικασία επιστρέφει σωστά την τιμή $p(k)$ για κάθε $k \in \{1, 2, \ldots, n\}$.

Έστω ότι $k = 1$ και έστω ότι βρίσκεται στη j'-οστή θέση με βάση το χρόνο έναρξής, δηλαδή $s_{j'} = k$.

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$ (ορθότητα)

Η παραπάνω διαδικασία επιστρέφει σωστά την τιμή $p(k)$ για κάθε $k \in \{1, 2, \ldots, n\}$.

Έστω ότι $k = 1$ και έστω ότι βρίσκεται στη j'-οστή θέση με βάση το χρόνο έναρξής, δηλαδή $s_{j'} = k$.

Αν $p(s_{j'}) = 0 = f_{i-1}$ τότε αυτό μπορεί να έχει προέλθει μόνο από το $i = 1$ άρα το i δεν έχει αυξηθεί καθόλου. Συνεπώς, λήξη$(f_1) >$ έναρξη$(s_{j'})$ που σημαίνει ότι το $s_{j'}$ δεν είναι συμβατό με κανένα αίτημα.

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$ (ορθότητα)

Η παραπάνω διαδικασία επιστρέφει σωστά την τιμή $p(k)$ για κάθε $k \in \{1, 2, \ldots, n\}$.

Έστω ότι $k = 1$ και έστω ότι βρίσκεται στη j'-οστή θέση με βάση το χρόνο έναρξής, δηλαδή $s_{j'} = k$.

Αν $p(s_{j'}) = 0 = f_{i-1}$ τότε αυτό μπορεί να έχει προέλθει μόνο από το $i = 1$ άρα το i δεν έχει αυξηθεί καθόλου. Συνεπώς, λήξη$(f_1) >$ έναρξη$(s_{j'})$ που σημαίνει ότι το $s_{j'}$ δεν είναι συμβατό με κανένα αίτημα.

Αν $p(s_{j'}) > 0$ τότε $p(s_{j'}) = f_{i'-1}$ για κάποιο $i' > 1$. Όταν ανατέθηκε αυτή η τιμή το i ήταν ίσο με το i'. Έστω t η χρονική στιγμή που το $i' - 1$ έγινε i'. Εκείνη τη χρονική στιγμή η λήξη$(f_{i'-1})$ ήταν μικρότερη ή ίση από την έναρξη$(s_{j''})$ (όπου πάλι $j'' \leq j'$).

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$ (ορθότητα)

Η παραπάνω διαδικασία επιστρέφει σωστά την τιμή $p(k)$ για κάθε $k \in \{1, 2, \ldots, n\}$.

Έστω ότι $k = 1$ και έστω ότι βρίσκεται στη j'-οστή θέση με βάση το χρόνο έναρξής, δηλαδή $s_{j'} = k$.

Αν $p(s_{j'}) = 0 = f_{i-1}$ τότε αυτό μπορεί να έχει προέλθει μόνο από το $i = 1$ άρα το i δεν έχει αυξηθεί καθόλου. Συνεπώς, λήξη$(f_1) >$ έναρξη$(s_{j'})$ που σημαίνει ότι το $s_{j'}$ δεν είναι συμβατό με κανένα αίτημα.

Αν $p(s_{j'}) > 0$ τότε $p(s_{j'}) = f_{i'-1}$ για κάποιο $i' > 1$. Όταν ανατέθηκε αυτή η τιμή το i ήταν ίσο με το i'. Έστω t η χρονική στιγμή που το $i' - 1$ έγινε i'. Εκείνη τη χρονική στιγμή η λήξη$(f_{i'-1})$ ήταν μικρότερη ή ίση από την έναρξη$(s_{j''})$ (όπου πάλι $j'' \leq j'$).

Άρα λήξη$(f_{i'-1}) \leq$ έναρξη$(s_{j''}) \leq$ έναρξη$(s_{j'})$.

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$ (ορθότητα)

Η παραπάνω διαδικασία επιστρέφει σωστά την τιμή $p(k)$ για κάθε $k \in \{1, 2, \ldots, n\}$.

Έστω ότι $k = 1$ και έστω ότι βρίσκεται στη j'-οστή θέση με βάση το χρόνο έναρξής, δηλαδή $s_{j'} = k$.

Αν $p(s_{j'}) = 0 = f_{i-1}$ τότε αυτό μπορεί να έχει προέλθει μόνο από το $i = 1$ άρα το i δεν έχει αυξηθεί καθόλου. Συνεπώς, λήξη$(f_1) >$ έναρξη$(s_{j'})$ που σημαίνει ότι το $s_{j'}$ δεν είναι συμβατό με κανένα αίτημα.

Αν $p(s_{j'}) > 0$ τότε $p(s_{j'}) = f_{i'-1}$ για κάποιο $i' > 1$. Όταν ανατέθηκε αυτή η τιμή το i ήταν ίσο με το i'. Έστω t η χρονική στιγμή που το $i' - 1$ έγινε i'. Εκείνη τη χρονική στιγμή η λήξη$(f_{i'-1})$ ήταν μικρότερη ή ίση από την έναρξη$(s_{j''})$ (όπου πάλι $j'' \leq j'$).

Άρα λήξη$(f_{i'-1}) \leq$ έναρξη$(s_{j''}) \leq$ έναρξη$(s_{j'})$.

Επιπλέον, αφού $p(s_{j'})$ πήρε τιμή f_{i-1} ίσχυε ότι λήξη$(f_{i'}) >$ έναρξη$(s_{j'})$.

Υπολογισμός $p(j)$ σε χρόνο $\mathcal{O}(n)$ (ορθότητα)

Η παραπάνω διαδικασία επιστρέφει σωστά την τιμή $p(k)$ για κάθε $k \in \{1, 2, \ldots, n\}$.

Έστω ότι $k = 1$ και έστω ότι βρίσκεται στη j'-οστή θέση με βάση το χρόνο έναρξής, δηλαδή $s_{j'} = k$.

Αν $p(s_{j'}) = 0 = f_{i-1}$ τότε αυτό μπορεί να έχει προέλθει μόνο από το $i = 1$ άρα το i δεν έχει αυξηθεί καθόλου. Συνεπώς, λήξη$(f_1) >$ έναρξη$(s_{j'})$ που σημαίνει ότι το $s_{j'}$ δεν είναι συμβατό με κανένα αίτημα.

Αν $p(s_{j'}) > 0$ τότε $p(s_{j'}) = f_{i'-1}$ για κάποιο $i' > 1$. Όταν ανατέθηκε αυτή η τιμή το i ήταν ίσο με το i'. Έστω t η χρονική στιγμή που το $i' - 1$ έγινε i'. Εκείνη τη χρονική στιγμή η λήξη$(f_{i'-1})$ ήταν μικρότερη ή ίση από την έναρξη$(s_{j''})$ (όπου πάλι $j'' \leq j'$).

Άρα λήξη$(f_{i'-1}) \leq$ έναρξη$(s_{j''}) \leq$ έναρξη$(s_{j'})$.

Επιπλέον, αφού $p(s_{j'})$ πήρε τιμή f_{i-1} ίσχυε ότι λήξη$(f_{i'}) >$ έναρξη$(s_{j'})$.

Συνεπώς, λήξη$(f_{i'-1}) \leq$ έναρξη$(s_{j''}) \leq$ έναρξη$(s_{j'}) <$ λήξη$(f_{i'})$.

Σύνοψη

- Χαρακτηρισμός δομής του προβλήματος.
- Αναδρομικός ορισμός της τιμής της βέλτιστης λύσης.
- Υπολογισμός της τιμής της βέλτιστης λύσης.
- Κατασκευή της βέλτιστης λύσης από την υπολογισμένη πληροφορία.