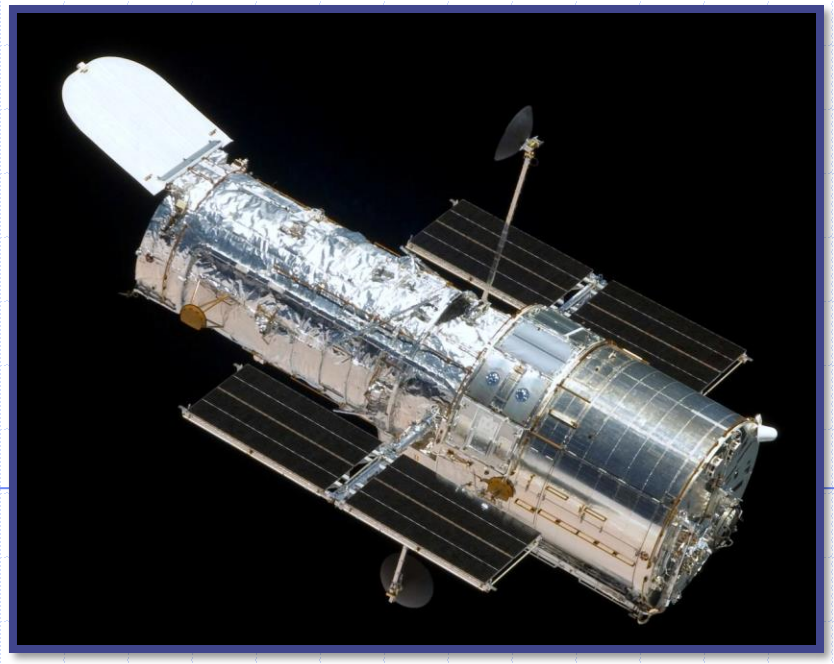
Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Δυναμικός Προγραμματισμός: Προγραμματισμός Τηλεσκοπίων



Hubble Space Telescope. Public domain image, NASA, 2009.

Παραδείγμα κίνητρο

- Τα μεγάλα, ισχυρά τηλεσκόπια αποτελούν πολύτιμους πόρους για τους οποίους υπάρχει μεγάλη ζήτηση από αστρονόμους που επιθυμούν να τα χρησιμοποιήσουν.
- Αυτή η υψηλή ζήτηση εκδηλώνεται με λήψη χιλιάδων αιτημάτων ανά μήνα για χρονικά διαστήματα παρατηρήσεων σε διαστημικά τηλεσκόπια.

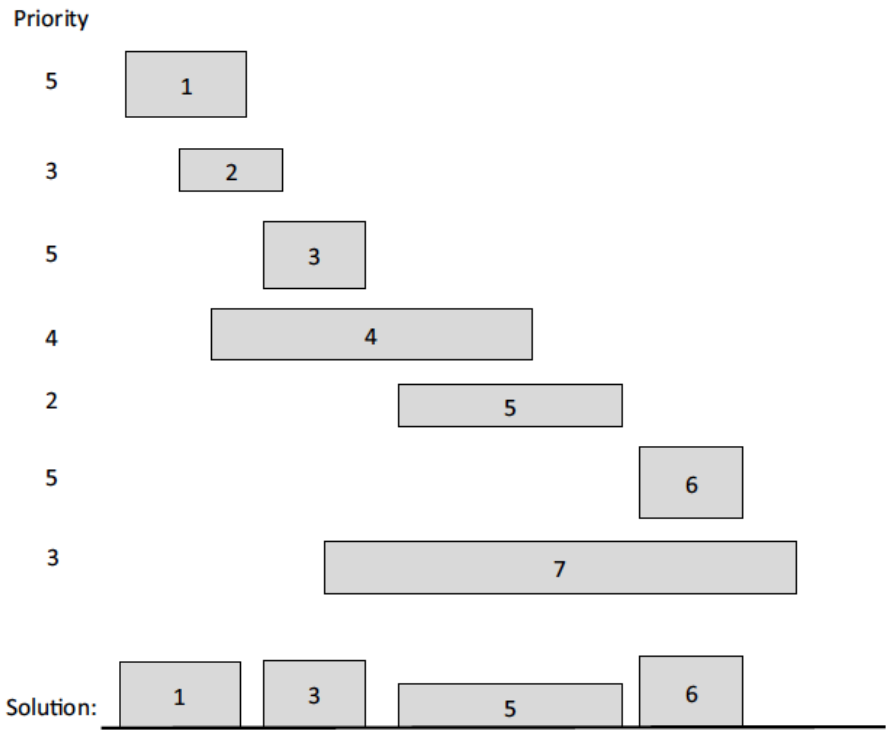
Πρόβλημα Προγραμματισμού Τηλεσκοπίων

- Η είσοδος στο πρόβλημα προγραμματισμού τηλεσκοπίων είναι μια λίστα, **L**, από αιτήματα παρατηρήσεων, όπου κάθε αίτημα, **i**, αποτελείται από τα ακόλουθα στοιχεία:
  - μια **αιτούμενη ώρα έναρξης**, s_i, που είναι η στιγμή που η αιτούμενη παρατήρηση θα πρέπει να ξεκινήσει
  - μια **ώρα τερματισμού**, f_i, που είναι η στιγμή που η αιτούμενη παρατήρηση θα πρέπει να τερματίσει (υποθέτοντας ότι ξεκίνησε στην ώρα έναρξής της)
  - μια θετική αριθμητική τιμή **όφελος**, b_i, που είναι δείκτης του επιστημονικού κέρδους που προσδοκάται από την πραγματοποίηση της αιτούμενης παρατήρησης.
- Οι χρόνοι έναρξης και λήξης του αιτήματος παρακολούθησης καθορίζονται από τον αστρονόμο που αιτείται την παρακολούθηση - το δε όφελος του αιτήματος καθορίζεται από έναν διαχειριστή ή μια επιτροπή αξιολόγησης αιτημάτων.

Πρόβλημα Προγραμματισμού Τηλεσκοπίων

- Για να ληφθεί το όφελος, $\mathbf{b_i}$, ενός αιτήματος παρατήρησης, **i**, η παρατήρηση θα πρέπει να πραγματοποιηθεί για το συνολικό διάστημα από την ώρα έναρξης, $\mathbf{s_i}$, μέχρι την ώρα τερματισμού, $\mathbf{f_i}$.
- Έτσι, δύο αιτήματα, **i** και **j**, **συγκρούονται** αν το χρονικό διάστημα $\mathbf{[s_i, f_i]}$, τέμνει το χρονικό διάστημα, $\mathbf{[s_j, f_j]}$.
- Δεδομένης μιας λίστας, **L**, με αιτήματα παρακολουθήσεων, το πρόβλημα βελτιστοποίησης αφορά τον προγραμματισμό των αιτημάτων παρακολούθησης με τρόπο που να μη δημιουργούνται συγκρούσεις και να μεγιστοποιείται το συνολικό όφελος από τις παρατηρήσεις που θα συμπεριληφθούν στο πρόγραμμα.

Παράδειγμα



Το αριστερό και το δεξιό όριο κάθε ορθογωνίου αναπαριστά τους χρόνους έναρξης και τερματισμού ενός αιτήματος παρακολούθησης. Το ύψος κάθε ορθογωνίου αναπαριστά το όφελος. Το όφελος κάθε αιτήματος (priority=προτεραιότητα) παρουσιάζεται στα αριστερά του σχήματος. Η βέλτιστη λύση έχει συνολικό όφελος 17=5+5+2+5.

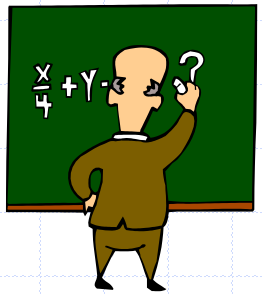
Απόπειρα 1: Ωμή Δύναμη

- Υπάρχει ένας προφανής εκθετικός αλγόριθμος για την επίλυση του προβλήματος, που εξετάζει όλα τα πιθανά υποσύνολα του L και επιλέγει εκείνο που έχει το υψηλότερο συνολικό όφελος χωρίς να προκαλεί συγκρούσεις.
- Η υλοποίηση του αλγορίθμου αυτού ωμής δύναμης απαιτεί χρόνο $O(n2^n)$, όπου n είναι το πλήθος των αιτημάτων παρατήρησης.
- Ωστόσο, υπάρχει πολύ καλύτερη λύση με χρήση του Δυναμικού Προγραμματισμού.

Απόπειρα 2: Άπληστη Μέθοδος

- Μια φυσική άπληστη στρατηγική θα ήταν να εξεταστούν τα αιτήματα παρατήρησης ταξινομημένα σε μη-αύξουσα τιμή οφέλους, και να συμπεριλαμβάνεται στο πρόγραμμα εκείνο το αίτημα που δεν βρίσκεται σε σύγκρουση με άλλο αίτημα που έχει επιλεγεί πριν από αυτό.
  - Ωστόσο, η στρατηγική αυτή δεν οδηγεί στη βέλτιστη λύση.
- Για παράδειγμα, έστω μια λίστα με 3 αιτήματα - ένα με όφελος 100 που συγκρούεται με δύο άλλα μη συγκρουόμενα μεταξύ τους αιτήματα με όφελος 75 το καθένα.
  - Η άπληστη μέθοδος θα επέλεγε την παρατήρηση με όφελος 100, ενώ μπορεί να επιτευχθεί συνολικό όφελος 150 επιλέγοντας τα αιτήματα με όφελος 75 το καθένα.
  - Συνεπώς, η άπληστη στρατηγική που επιλέγει επαναληπτικά το αίτημα με το μεγαλύτερο όφελος που δεν δημιουργεί σύγκρουση δεν επιστρέφει τη βέλτιστη λύση.

Η γενική τεχνική δυναμικού προγραμματισμού



- Εφαρμόζεται σε προβλήματα που αρχικά φαίνεται να απαιτούν πολύ χρόνο (πιθανότατα εκθετικό), αρκεί να έχουμε:
  - **Απλά υπό-προβλήματα:** τα υπό-προβλήματα να μπορούν να οριστούν χρησιμοποιώντας λίγες μόνο μεταβλητές, όπως j, k, l, m, κ.ο.κ.
  - **Βελτιστότητα υπό-προβλημάτων:** η καθολικά βέλτιστη λύση να μπορεί να οριστεί με όρους βέλτιστων λύσεων υπό-προβλημάτων.
  - **Επικάλυψη υπό-προβλημάτων:** τα υπό-προβλήματα να μην είναι ανεξάρτητα, αλλά να επικαλύπτονται (οπότε να πρέπει να κατασκευαστούν από κάτω προς τα πάνω).

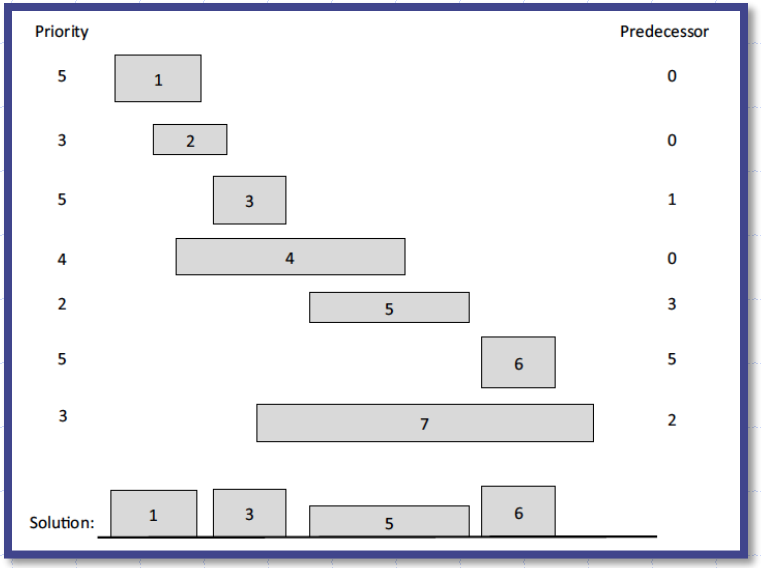
Ορισμός Απλών Υποπροβλημάτων

- Ένας φυσικός τρόπος ορισμού υποπροβλημάτων είναι να εξεταστούν τα αιτήματα παρατήρησης σύμφωνα με κάποια διάταξη, όπως ταξινόμηση κατά χρόνο έναρξης, ταξινόμηση κατά χρόνο τερματισμού, ή ταξινόμηση κατά όφελος.
  - Ήδη είδαμε ότι η ταξινόμηση κατά όφελος δεν οδηγεί σε βέλτιστες λύσεις.
  - Οι χρόνοι έναρξης και οι χρόνοι τερματισμού είναι ουσιαστικά συμμετρικοί, οπότε ας ταξινομήσουμε τις παρατηρήσεις κατά χρόνους τερματισμού.

B_i = το μέγιστο όφελος που μπορεί να επιτευχθεί από τα πρώτα i αιτήματα παρατηρήσεων της L
Ως οριακή συνθήκη, ορίζουμε $B_0=0$

Προκάτοχοι

- Για κάθε αίτημα **i**, το σύνολο των άλλων αιτημάτων που συγκρούονται με το **i** σχηματίζουν ένα συνεχές διάστημα αιτημάτων στο **L**.
- Ορίζεται ως **προκάτοχος (predecessor)**, pred(**i**), για κάθε αίτημα, **i**, ο μεγαλύτερος δείκτης, **j < i**, τέτοιος ώστε τα αιτήματα **i** και **j** να μη συγκρούονται. Αν δεν υπάρχει τέτοιο **j**, τότε ορίζεται ως προκάτοχος του **i** το 0.



Η λίστα αιτημάτων διατεταγμένη κατά χρόνο λήξης αιτημάτων L: 1→(0, **5**), 2→(2, **7**), 3→(6, **11**), 4→(4, **17**), 5→(13, **23**), 6→(24, **28**), 7→(9, **30**)

Για να υπολογιστεί ο προκάτοχος κάθε αιτήματος i, μπορούμε να αναζητήσουμε τη θέση του s_i (χρόνος έναρξης του i) στο L με δυαδική αναζήτηση στους χρόνους λήξης.

π.χ. Ο προκάτοχος του αιτήματος 7→(9,30) εντοπίζεται αναζητώντας τη θέση του s_i=9, στη λίστα 5,7,11,17,23,28,30. Η θέση είναι αμέσως μετά το **7** και συνεπώς ο προκάτοχος του αιτήματος 7 είναι το αίτημα 2→(2,**7**)

Βελτιστότητα Υπο-προβλήματος

- Ένα πρόγραμμα που επιτυγχάνει τη βέλτιστη τιμή, $\mathbf{B_i}$, είτε περιέχει την παρατήρηση **i** είτε δεν την περιέχει.

- Αν το βέλτιστο πρόγραμμα που πετυχαίνει όφελος B_i περιέχει την παρατήρηση i τότε $B_i = B_{pred(i)} + b_i$
- Αν το βέλτιστο πρόγραμμα που πετυχαίνει όφελος B_i δεν περιέχει την παρατήρηση i τότε $B_i = B_{i-1}$

Συνεπώς, οδηγούμαστε στον ακόλουθο αναδρομικό ορισμό:

$$B_i = \max\{B_{i-1}, B_{pred(i)} + b_i\}$$

Επικάλυψη Υποπροβλημάτων

- Ο παραπάνω ορισμός εμπεριέχει επικάλυψη υποπροβλημάτων.
- Έτσι, είναι αποδοτικότερο να χρησιμοποιείται memoization όταν υπολογίζονται οι τιμές $\mathbf{B_i}$, με αποθήκευση σε έναν πίνακα, **B**, ο οποίος δεικτοδοτείται από το 0 μέχρι το **n**.
- Ταξινομώντας τα αιτήματα παρατηρήσεων κατά χρόνο τερματισμού και χρησιμοποιώντας έναν πίνακα, **P**, τέτοιον ώστε **P**[**i**] = pred(**i**), μπορούμε να συμπληρώσουμε τον πίνακα, **B**, χρησιμοποιώντας τον ακόλουθο απλό αλγόριθμο:

$B[0] \leftarrow 0$
for $i = 1$ to n **do**
    $B[i] \leftarrow \max\{B[i-1],\ B[P[i]] + b_i\}$

After this algorithm completes, the benefit of the optimal solution will be $B[n]$

Ανάλυση του αλγόριθμου

- Ο χρόνος εκτέλεσης του αλγόριθμου είναι **O(n)**, υποθέτοντας ότι η λίστα **L** είναι διατεταγμένη κατά χρόνους τερματισμού και ότι δίνεται ο προκάτοχος κάθε αιτήματος παρατήρησης **i**.
- Βέβαια, μπορούμε εύκολα να ταξινομήσουμε τη λίστα **L** κατά χρόνους τερματισμού αν η λίστα δεν έχει ήδη δοθεί κατάλληλα ταξινομημένη.
- Για τον υπολογισμό του προκατόχου κάθε αιτήματος, παρατηρήστε ότι αρκεί επιπλέον να έχουμε τα αιτήματα του **L** ταξινομημένα κατά χρόνο έναρξης.
  - Ειδικότερα, δεδομένης μιας λίστας **L** ταξινομημένης κατά χρόνους τερματισμού και μιας άλλης λίστας, **L'**, ταξινομημένης κατά χρόνους έναρξης, η συγχώνευση των δύο λιστών, όπως στον αλγόριθμο merge-sort, επιτρέπει τον εντοπισμό της ζητούμενης τιμής.
  - Ο προκάτοχος του αιτήματος **i** είναι ο δείκτης του προκατόχου στο **L** της τιμής, $\mathbf{s_i}$, στο **L'**.