2η Γραπτή σειρά ασκήσεων
Μάνος Αραπίδης, Α.Μ.: 03116071

<u>1η Άσκηση</u>

1)

a)

Το κριτήριο να στοιβάζουμε τα πακέτα σε φθίνουσα σειρά ανάλογα με το βάρος τους είναι λάθος. Ένα αντιπαράδειγμα που αν ακολουθούσαμε το κριτήριο θα οδηγούμασταν σε λάθος λύση είναι το εξής:

Έστω ότι έχουμε 2 πακέτα με βάρος 5 και 4 και αντοχή 1 και 5. Τα αναπαριστούμε με αυτόν τον τρόπο : (5,1) (4,5).

Σύμφωνα με το κριτήριο θα βάζαμε στη βάση το πακέτο (5,1) και θα ελέγχαμε αν μπορούμε να βάλουμε από πάνω το (4,5). Οδηγούμαστε ότι δεν υπάρχει λύση αν ακολουθήσουμε αυτό το κριτήριο, ενώ υπάρχει και είναι να βάλουμε πρώτα το (4,5) και το (5,1).

Άρα το κριτήριο είναι λανθασμένο.

a)

Το κριτήριο να στοιβάζουμε τα πακέτα σε φθίνουσα σειρά ανάλογα με την αντοχή τους είναι λάθος. Ένα αντιπαράδειγμα που αν ακολουθούσαμε το κριτήριο θα οδηγηθούμε σε λάθος λύση είναι το εξής:

Έστω ότι έχουμε 2 πακέτα (100,2) και (1,99). Αν ακολουθήσουμε το κριτήριο θα προσπαθήσουμε να τα στοιβάξουμε με αυτόν τον τρόπο : (1,99) (100,2) και θα συμπεράνουμε ότι δεν υπάρχει λύση. Το συμπέρασμα είναι λάθος καθώς υπάρχει λύση με την εξής στοίβαξη : (100,2) (1,99).

Άρα το κριτήριο είναι λανθασμένο.

b)

Το κριτήριο να στοιβάζουμε τα πακέτα σε φθίνουσα σειρά ανάλογα με άθροισμα του βάρους και της αντοχής τους είναι σωστό. Το κριτήριο μπορούμε να το μεταφράσουμε με το εξής τρόπο, αν υπάρχει λύση στο πρόβλημα τότε ένας επιτρεπτός τρόπος στοίβαξης είναι με βάση το άθροισμα του βάρους και της αντοχής .Δηλαδή για κάθε υποστοίβα της παραπάνω στοίβαξης μπορούμε να χρησιμοποιήσουμε ως βάση το πακέτο με το μεγαλύτερο άθροισμα βάρους και αντοχής.

Έστω 2 πακέτα (α,β) και (γ,δ), για τα οποία ισχύει η σχέση $\alpha+\beta\geq\gamma+\delta \Leftrightarrow \beta-\gamma\geq\delta-\alpha$(1)
Αν υπάρχει λύση θα πρέπει $\beta\geq\gamma$ ή $\delta\geq\alpha$,για να μπορούνε τα πακέτα να στοιβαχτούν με τον εξής τρόπο : (α,β) (γ,δ) ή (γ,δ) (α,β).

- Αν $\beta-\gamma<0 \Leftrightarrow \beta<\gamma$ τότε δεν μπορεί να γίνει η στοίβαξη (α,β) (γ,δ)

Λόγω της (1) $\delta-\alpha<0 \Leftrightarrow \delta<\alpha$

Άρα ούτε η στοίβαξη (γ,δ) (α,β) μπορεί να γίνει. Όποτε συμπεραίνουμε ότι αν ισχύει η (1) και το στοιχείο με το μεγαλύτερο άθροισμα δεν μπορεί να χρησιμοποιηθεί ως βάση τότε δεν υπάρχει λύση στο πρόβλημα.

- Αν $\beta-\gamma\geq 0 \Leftrightarrow \beta\geq\gamma$, τότε η στοίβαξη (α,β) (γ,δ) μπορεί να γίνει. Άρα υπάρχει λύση.

  Λόγω της (1) έχουμε :
  $\delta-\alpha\geq 0 \Leftrightarrow \delta\geq\alpha$, άρα μπορεί να γίνει η στοίβαξη (γ,δ) (α,β) και υπάρχει λύση.
  ή
  $\delta-\alpha<0 \Leftrightarrow \delta<\alpha$, άρα δεν μπορεί να γίνει η στοίβαξη (γ,δ) (α,β) και δεν υπάρχει λύση για αυτήν.

Όποτε συμπεραίνουμε ότι, εφόσον υπάρχει τουλάχιστον μια δυνατή στοίβαξη που να επιλύει το πρόβλημα τότε πάντοτε τα πακέτα θα μπορούν να στοιβαχτούν σε φθίνουσα σειρά ανάλογα με το άθροισμα του βάρους και της αντοχής τους.
Το παραπάνω το αποδείξαμε για δυο πακέτα, επαγωγικά αποδεικνύεται ότι ισχύει για οποιοδήποτε αριθμό πακέτων.

2)

Για να βρούμε το μέγιστο κέρδος που μπορούμε να αποκομίσουμε από την στοίβαξη i=1,2...n πακέτων (w_i,d_i) με κέρδος p_i το καθένα ορίζουμε την παρακάτω αναδρομική σχέση:

$$Profit(i,w)=\begin{cases} 0, \alpha\nu\ w\leq 0\ \eta\ i=0 \\ Profit(i-1,w), \alpha\nu\ i\geq 1\ \kappa\alpha\iota\ w<wi \\ max\{Profit(i-1,w),\ pi+Profit(i-1,min(w-wi,di))\} \\ ,\gamma\iota\alpha\ i=1,2,\ldots,n\ \kappa\alpha\iota\ w=wi,\ldots,W+D \end{cases}$$

Το πρώτο όρισμα της σχέσης αντιπροσωπεύει το πακέτο που εξετάζουμε κάθε φόρα και το δεύτερο το βάρος που μπορεί να αντέξει η στοίβα με τα πακέτα. Ο πίνακας που δημιουργείται είναι μεγέθους n*(W+D), όπου n είναι το πλήθος των πακέτων και W+D αντιστοιχεί στο μέγιστο άθροισμα βάρους και αντοχής από όλα τα πακέτα. Δηλαδή το μέγιστο βάρος που μπορεί να αντέξει μια άδεια στοίβα είναι W+D.
Ακόμα, τα πακέτα τα εξετάζουμε με αύξουσα σειρά του αθροίσματος αντοχής και βάρους. Ο λόγος που το κάνουμε αυτό είναι, ότι παραπάνω αποδείξαμε αν μια ακολουθία πακέτων είναι έγκυρη, τότε τα πακέτα μπορούν να στοιβαχτούν σε φθίνουσα σειρά βάρους και αντοχής. Αν και στη προκειμένη περίπτωση τα πακέτα τα εξετάζουμε με αύξοντα τρόπο επειδή και το βάρος που αντέχει μια στοίβα, που αντιπροσωπεύεται από τον αριθμό της στήλης, το ελέγχουμε με τον ίδιο τρόπο.
Ο πίνακας μας ενημερώνεται από τα αριστερά προς τα δεξιά και από πάνω προς τα κάτω. Από την αναδρομική σχέση φαίνεται ότι η μέγιστη τιμή κέρδους που μπορούμε να αποκομίσουμε, καθώς αυτή ενημερώνεται ανά τις επαναλήψεις του αλγόριθμου, διατηρείται στη δεξιά στήλη, πράγμα λογικό καθώς αυτή αντιπροσωπεύει την στοίβα με το μέγιστο

δυνατό βάρος. Οπότε μετά το τέλος του αλγορίθμου το μέγιστο κέρδος βρίσκεται στην θέση του πίνακα P[n,W+D].
Παρακάτω φαίνεται ο ψευδοκώδικας της παραπάνω αναδρομικής σχέσης
$Profit(\{(w1,d1),...,(wn,dn)\},W+D)$

```
for i← 0 to n do Profit[i,0] ← 0
for w← 1 to W+D do Profit[0,w] ← 0
for i← 1 to n do
        for w←1 to W+D do
                if w-w_i ≥ 0 then
                        t← Profit[i-1, min{w-w_i, d_i}] + p_i
                else t←0
                if Profit[i-1,w] ≥ t then
                        Profit[i,w] ← Profit[i-1,w]
                else Profit[i,w] ← t
return Profit[n,W+D]
```

Όπως φαίνεται και από πάνω ο αλγόριθμος μας χρειάζεται n*(W+D) μνήμη και έχει πολυπλοκότητα O(n*(W+D)).

2η Άσκηση

Από κάθε χώρα i έχουμε να διαλέξουμε ανάμεσα σε K_i αντικείμενα με συναισθηματική αξία p_k και κόστος c_k , κ = 1,2..., K_i. Έχουμε συνολικά C χρηματικό ποσό να διαθέσουμε, για να αγοράσουμε ένα αναμνηστικό από κάθε χώρα.

Προκειμένου να λύσουμε το πρόβλημα βελτιστοποίησης, να βρούμε δηλαδή την συλλογή των αντικειμένων με τη μέγιστη συναισθηματική αξία αλλά με το ελάχιστο κόστος αναπτύσσουμε την παρακάτω αναδρομική σχέση.

.

$$Souvenirs(i,c)=\begin{cases} 0, \alpha v\ i=0 \\ -\infty, \alpha v\ c\le 0\ \kappa\alpha\iota\ i\ge 1 \\ max\{pk+Souvenirs(i-1,c-ck)\} \\ ,\gamma\iota\alpha\ 1\le c\le C\ \kappa\alpha\iota\ i\ge 1\ \kappa\alpha\iota\ 1\le k\le Ki \end{cases}$$

Το πρώτο όρισμα της αναδρομικής σχέσης μας δείχνει ποια χώρα i=1,2,...n εξετάζουμε και το δεύτερο το χρηματικό ποσό c=1,2,...,C που μπορούμε να διαθέσουμε. Για την υλοποίηση του παραπάνω αλγόριθμου χρειάζεται ένας πίνακας n*C, όπου στα κελιά του βάζουμε την συνολική συναισθηματική αξία που μπορούμε να έχουμε για i χώρες με c χρήματα. Σε κάθε επανάληψη της αναδρομικής σχέσης ελέγχουμε αν για την χώρα i που εξετάζουμε υπάρχει ένα τουλάχιστον αναμνηστικό, το οποίο να μπορούμε να αγοράσουμε με χρήματα c και να μας επιτρέπει να έχουμε ένα αντικείμενο από τις i-1 χώρες. Το παραπάνω το πετυχαίνουμε μέσω της σχέσης $max\{pk+Souvenirs(i-1,c-ck)\}$, σύμφωνα με την οποία υπολογίζουμε την συναισθηματική αξία που θα είχαμε για κάθε αναμνηστικό από τα K_i ,αν επιλέγαμε να το αγοράσουμε και από αυτές τις τιμές κρατάμε την μεγαλύτερη. Αν ένα αναμνηστικό k έχει κόστος $c_k \ge c$ <=> $c-c_k \le 0$ τότε η αξία του είναι -∞, για να δείξουμε ότι δεν μπορούμε να το αγοράσουμε με c χρήματα. Ο λόγος που συγκρίνουμε και με ισότητα είναι για να δείξουμε ότι, πέραν της περίπτωσης i=1, αν η τιμή του αναμνηστικού ισούται με το ποσό c ,δεδομένου ότι το επιλέγαμε, δεν μας

επιτρέπει να αγοράσουμε τίποτα από τις i-1 χώρες, γι' αυτό και το απορρίπτουμε.
Στην αντίθετη περίπτωση όπου c_k>c ,προσθέτουμε την αξία του p_k σ' αυτήν που είχαμε υπολογίσει στη θέση Souvenirs[i-1,c-c_k]. Αν Souvenirs[i-1,c-c_k] = -∞, δηλαδή δεν μπορούμε να αγοράσουμε αντικείμενα από i-1 χώρες με c-c_k χρήματα, τότε το αποτέλεσμα της παραπάνω πράξης είναι -∞, καθιστώντας την επιλογή του συγκεκριμένου αναμνηστικού λανθασμένη. Αλλιώς, αν το αποτέλεσμα είναι >0 τότε η επιλογή μας είναι έγκυρη. Αφού έχουμε εξετάσει και τα K_i αναμνηστικά, συγκρίνουμε την αξία που μας επιστρέφουν, διαλέγουμε αυτή με την μεγαλύτερη τιμή και την τοποθετούμε στη θέση του πίνακα Souvenirs[i,c].
Για i=1 , αν και ακολουθούμε απλά την αναδρομική σχέση, αυτό που κάνουμε στην πραγματικότητα φαίνεται λίγο διαφορετικό. Αν c-c_k<o, τότε το αποτέλεσμα της πράξης $pk+Souvenirs(0,c-ck)$ισούται με -∞ για τους λόγους που εξηγήσαμε πριν. Όμως αν c_k-c≥0, τότε $pk+Souvenirs(0,c-ck)=pk$ επειδή $Souvenirs(0,c-ck)=0$. Ο λόγος που επιτρέπεται να κάνουμε το παραπάνω και για αναμνηστικά με c_k=c είναι γιατί για i=1 δεν υπάρχουν άλλες i-1 χώρες που πρέπει να διαλέξουμε αντικείμενα.
Στο τέλος της αναδρομής, θα έχουμε γεμίσει τον πίνακα και στα κελιά της γραμμής n θα έχουμε την συνολική συναισθηματική αξία για αναμνηστικά και από τις n χώρες με χρηματικό ποσό από c = 1 έως C. Οπότε, αρκεί να βρούμε την μέγιστη τιμή οπού και την επιστρέφουμε ως λύση στο πρόβλημα μας.
Παρακάτω φαίνεται και ο ψευδοκώδικας που περιγράφει τον αλγόριθμο.
$Souvenirs(\{1,2,..,n\},C)$

```
for c←0 to C do Souvenirs[0,c]← 0
for i←1 to n do
        for c←0 to C do
                if i=1 then
                        for k←1 to Ki do
                                if c-ck < 0 then
                                        value[k]← -∞
                                else value[k]← pk
                        Souvenirs[1,c] ← max[value[0 to Ki-1]]
                else then
                        for k←1 to Ki do
                                if c-ck ≤ 0 then
                                        value[k]← -∞
                                else value[k]← pk + Souvenirs[i-1, c-ck]
                        Souvenirs[i,c] ← max[value[0 to Ki-1]]
result ← max[Souvenirs[n, 0 to C]]
return result
```

Όπως γίνεται ξεκάθαρο και από το ψευδοκώδικα ο αλγόριθμος έχει πολυπλοκότητα $O(\sum_{i=1}^{n} Ki*C)$και χρειάζεται μνήμη $\Theta(n*C+Kmax)$για έναν πίνακα μεγέθους n*C και έναν K_{max}, όπου K_{max} είναι το μέγιστο πλήθος αναμνηστικών από τις n χώρες.

<u>3η Άσκηση</u>

Έχουμε να διαλέξουμε από i = 1,2,…,n κουτιά που το καθένα έχει μέσα q_i σοκολατάκια με τύπο type_i. Εμείς πρέπει να φάμε αρκετά, ώστε ο συνολικός αριθμός να είναι μεγαλύτερος ή ίσος με Q. Κάθε κουτί j που καταναλώνουμε πρέπει σε σύγκριση με το προηγούμενο i να ισχύουν τα εξής : q_j>q_i και type_j≠type_i. Η μετακίνηση από το κουτί i στο j

χρειάζεται χρόνο όσο η μεταξύ τους απόσταση, $dis(i,j)$. Όποτε θέλουμε τον ελάχιστο χρόνο που απαιτείται, ξεκινώντας από το κουτί p για να φάμε τουλάχιστον Q σοκολατάκια.
Για να λύσουμε το πρόβλημα αναπτύσσουμε την παρακάτω αναδρομική σχέση.

$$Chocolate(i,q)=\begin{cases} 0, \alpha\nu\ Q\le qi+q \\ +\infty, \alpha\nu\ Q>qi+q\ \kappa\alpha\iota \\ \nexists\, j\in N: i<j\le n\ \kappa\alpha\iota\ qj>qi\ \kappa\alpha\iota\ type_i\neq type_j \\ min\left(dis(i,j)+Chocolate(j,q+qi)\right), \\ \gamma\iota\alpha\ \forall\, j\in N: i<j\le n\ \kappa\alpha\iota\ qj>qi\ \kappa\alpha\iota\ type_i\neq type_i \end{cases}$$

Η σχέση αυτή υπολογίζει τον χρόνο που χρειάζεται να κάνουμε για να φάμε τουλάχιστον Q σοκολατάκια, αν βρισκόμαστε στο κουτί i : $1\le i\le n$ και έχουμε φάει ήδη q:$0\le q\le Q$. Για την υλοποίηση της χρειαζόμαστε ένα πίνακα μεγέθους n*Q.
Αρχικά, ταξινομούμε τα κουτιά με αύξοντα τρόπο σε σχέση με την χωρητικότητα τους σε σοκολατάκια, q. Ο λόγος που το κάνουμε αυτό είναι γιατί για ένα κουτί i θέλουμε κάθε φορά να εξετάζουμε μόνο εκείνα τα κουτιά j που ισχύει $qj>qi\ \kappa\alpha\iota\ type_i\neq type_i$. Οπότε, με την ταξινόμηση ελέγχουμε κάθε φορά μόνο τις γραμμές του πίνακα που είναι μεγαλύτερες του i. Φαίνεται λοιπόν ότι για μια γραμμή i χρειάζεται να έχουμε γεμίσει και τις γραμμές μεγαλύτερες της i. Γι' αυτό λοιπόν, ξεκινάμε την αναδρομή από την γραμμή n, δηλαδή ο πίνακας μας γεμίζει από κάτω προς τα πάνω. Άρα, ξεκινώντας από την γραμμή n γεμίζουμε διαδοχικά τα κελιά από την στήλη 0 έως την Q και επαναλαμβάνουμε το ίδιο αναδρομικά για τις επάνω γραμμές.
Η σχέση ελέγχει, αν ανοίγοντας το κουτί i με q ήδη φαγωμένα υπάρχει τρόπος φτάσουμε σε τουλάχιστον Q. Αν το παραπάνω γίνεται, βάζουμε στο κελί του πίνακα Chocolate[I,q] τον ελάχιστο χρόνο που απαιτείται, ο υπολογισμός του οποίου αναλύεται παρακάτω, αλλιώς βάζουμε $+\infty$ για να δείξουμε ότι δεν υπάρχει χρόνος που να μας επιτρέπει να φάμε Q σοκολατάκια. Πιο συγκεκριμένα, αρχικά ελέγχουμε αν με q φαγωμένα σοκολατάκια και τα qi του κουτιού i μπορούμε να πετύχουμε τον στόχο μας, δηλαδή αν $Q\le q+qi$, τότε δηλώνουμε ότι δεν χρειάζεται να κινηθούμε σε άλλο κουτί και επιστρέφουμε στον πίνακα 0 λεπτά, Chocolate[i][q]=0. Αν το τελευταίο δεν γίνεται,$Q>q+qi$, τότε ψάχνουμε όλα τα κουτιά j τα οποία είναι προσβάσιμα σε μας, δηλαδή να ισχύει $qj>qi\ \kappa\alpha\iota\ type_i\neq type_j$. Στην περίπτωση που δεν υπάρχουν τέτοια κουτιά τότε είναι αδύνατο από το κουτί i και q σοκολατάκια προηγουμένως φαγωμένα να φτάσουμε σε Q, άρα επιστρέφουμε στον πίνακα Chocolate[i][q]=$+\infty$. Αντίθετα αν υπάρχουν j κουτιά , $j: i<j\le n$, τότε μέσω της σχέσης υπολογίζουμε για κάθε j τον χρόνο dis(i,j) κάνουμε για να μετακινηθούμε από κουτί i στο j και τον προσθέτουμε στον χρόνο $Chocolate(j,q+qi)$. Όπως είπαμε προηγουμένως ξεκινάμε με $i=n$ για $q=1\ \acute{\epsilon}\omega\varsigma\ Q$και έπειτα το i μειώνουμε κατά ένα και επειδή στην παραπάνω σχέση ισχύει $j>i$ ο χρόνος $Chocolate(j,q+qi)$ έχει υπολογιστεί νωρίτερα. Οπότε από τους χρόνους που επιστρέφει η σχέση $dis(i,j)+Chocolate(j,q+qi)$ κρατάμε πάντα τον μικρότερο και τον τοποθετούμε στην θέση Chocolate[i][q].
Στο τέλος της αναδρομής θα έχουμε γεμίσει όλο πίνακα και για λύσουμε το πρόβλημα αρκεί να ελέγξουμε την θέση Chocolate[p][0], η οποία δηλώνει τον χρόνο που κάνουμε αν ανοίγαμε πρώτα το κουτί p. Αν η τιμή

της είναι Chocolate[p][0] $\neq+\infty$, τότε την επιστρέφουμε ως την λύση μας. Στην αντίθετη περίπτωση ψάχνουμε στην στήλη 0 για όλα τα κουτιά $i: p<i\leq n \kappa\alpha\iota qi>qp \kappa\alpha\iota type_p \neq type_i$ την ελάχιστη τιμή της παράστασης $dis(i,p)+Chocolate[i][0]$ και την επιστρέφουμε ως την λύση.
Η πολυπλοκότητα του αλγορίθμου είναι $O(n^2*Q)$ και το αποδεικνύουμε παρακάτω.
Στην αναδρομική σχέση $Chocolate(i,q)$ για κάθε (i,q) με $1\leq i\leq n \kappa\alpha\iota 0\leq q\leq Q$ έχουμε στην χειρότερη περίπτωση να συγκρίνουμε $n-i$ τιμές. Οπότε για κάθε κελί της γραμμής i έχουμε να κάνουμε το πολύ $n-i$ συγκρίσεις και κάθε γραμμή έχει Q κελιά. Άρα η πολυπλοκότητα προκύπτει

$$O\left(\sum_{i=1}^{n}(n-i)*Q\right)=O\left(Q*\sum_{i=1}^{n}n-i\right)=O(Q*n^2).$$

Παρακάτω φαίνεται και ψευδοκώδικας του αλγορίθμου. Θεωρώ ότι η ταξινόμηση των κουτιών με βάση την χωρητικότητα σε σοκολατάκια έχει ήδη γίνει.

```
Chocolate({(1,q1,type_1),...,(n,qn,type_n)}, Q, p)
    for i← n to 1 do
        for q←0 to Q do
            if i=n then
                if Q>q+qi then Chocolate[i][q]←+∞
                else Chocolate[i][q] ← 0
            else
                if Q≤q+qi then
                    Chocolate[i][q] ← 0
                else
                    flag←false
                    k←0
                    for j←i+1 to n do
                        if type_j≠type_i and qj > qj then
                            value[k] ←dis(i,j)
+Chocolate[j][q+qi]
                            flag ← true
                        k++
                    if flag = false then
                        Chocolate[i][q]←+∞
                    else
                        Chocolate[i][q]←min(value[0 to k])
    if Chocolate[p][0] ≠+∞ then return Chocolate[p][0]
    else
        k=0
        for j←p+1 to n do

        value[k] ←dis(p,j)+Chocolate[j][0]
        k++
    return min(value[0 to k])
```

5η Άσκηση

a)

Έχουμε ένα δυαδικό δέντρο T=(V,E), με |V|=n κορυφές και ρίζα r. Θέλουμε να βρούμε ένα υποσύνολο κορυφών του T το K, τέτοιο ώστε |K| =κ όπου $1\leq\kappa\leq n$ και η απόσταση των κορυφών του δέντρου από τον κοντινότερο πρόγονο τους στο K να είναι η ελάχιστη. Οπότε , πρέπει η τιμή της παράστασης $cost(K)=max\{cost(u,K)\}$ να είναι ελάχιστη, όπου

$cost(u,K)$ η απόσταση της κορυφής $u \in V$ από τον κοντινότερο πρόγονο της που ανήκει στο K.
Παρακάτω φαίνεται ορισμός της συνάρτησης cost.

$$cost(u,K)=\begin{cases} 0, αν\, u \in K \\ \infty, αν\, u=r\, και\, r\, δεν\, ανήκει\, στο\, K \\ cost(par(u),K)+1, διαφορετικά \end{cases}$$

Για να βρούμε το υποσύνολο K με |K|=κ αναπτύσσουμε την παρακάτω αναδρομική σχέση.

$$Tree(i,j,K)=\begin{cases} 0, αν\, i \in K \\ +\infty, αν\, i=0\, και\, i\, δεν\, ανήκει\, στο\, K \\ tree(parent(i),j,K)+1, διαφορετικά \end{cases}$$

Για την υλοποίηση της αναδρομής χρειαζόμαστε ένα πίνακα Tree μεγέθους n^2. Το πρώτο όρισμα $i: 1 \le i \le n$ δηλώνει την κορυφή που θέλουμε να υπολογίσουμε την απόσταση, το δεύτερο $j: 1 \le j \le n$ την κορυφή που προσθέτουμε προσωρινά, για εκείνη μόνο της επανάληψη, στο σύνολο K και το τρίτο το σύνολο K για το οποίο εκτελούμε την αναδρομή.
Οι κορυφές i του δέντρου T πρέπει να ελέγχονται με τρόπο preorder(root,left,right). Αρχικά, χρειάζεται να κάνουμε ένα πέρασμα στο δέντρο προκειμένου να βρούμε τον άμεσο πρόγονο κάθε κορυφής και να αποθηκεύσουμε το αποτέλεσμα σε ένα πίνακα parent, για να μπορούμε να έχουμε την θέση του πατέρα σε σταθερό χρόνο, σε κάθε επανάληψη. Όπως είπαμε παραπάνω, ξεκινάμε πρώτα από την ρίζα r του δέντρου και έπειτα συνεχίζουμε στα παιδιά της, άρα ο πίνακας Tree γεμίζει από πάνω προς τα κάτω. Οπότε, η διάταξη του πίνακα είναι τέτοια ώστε για κάθε γραμμή i , όπου το i αντιστοιχεί στην υπό εξέταση κορυφή, πρέπει η γραμμή $parent[i]$, δηλαδή αυτή του πατέρα του i , να έχει συμπληρωθεί. Εξαίρεση σ' αυτό αποτελεί η ρίζα $i=0$ αφου $parent[0]=0$.
Για κάθε i υπολογίζουμε την απόσταση αν προσθέταμε προσωρινά το j στο K. Αρχικά ισχύει $K=\varnothing$, οπότε εκτελούμε την αναδρομική σχέση Tree ψάχνοντας να βρούμε K με |K|=1. Ο τρόπος που γεμίζουμε τα κελιά Tree[i][j] είναι ο παρακάτω. Αν προσθέτοντας το j στο K, η υπό εξέταση κορυφή $i \in K$, τότε επιστρέφουμε κόστος 0. Αν εξετάζουμε την ρίζα, $i=0$ και εκείνη, παρά την προσθήκη του j, δεν ανήκει στο K τότε επιστρέφουμε $+\infty$. Σε αντίθετη περίπτωση από τις παραπάνω, επιστρέφουμε το κόστος για τον πατέρα του i , όπου το είχαμε υπολογίσει προηγουμένως, αυξημένο κατά ένα, δηλαδή $Tree[i][j]=Tree[parent[i]][j]+1$. Μετά την ολοκλήρωση της αναδρομής, σε κάθε κελί $Tree[i][j] i,j: 1 \le i \le n,\ 1 \le j \le n$ έχουμε το κόστος για την κορυφή i αν μόνο η κορυφή $j \in K, |K|=1$. Οπότε, διατρέχοντας κάθε στήλη j μπορούμε να βρούμε το μέγιστο κόστος $max\{Tree[0 έως n-1][j]\}$ αν επιλέγαμε να βάλουμε το j στο K. Ο χρόνος εύρεσης του μεγίστου για κάθε στήλη είναι $O(n^2)$. Αφού κάνουμε το παραπάνω , επιλέγουμε να συμπεριλάβουμε στο K το j που έχουμε το ελάχιστο μέγιστο κόστος. Για να βρούμε το ελάχιστο χρειάζεται $O(n)$, οπότε συνολικά χρειάζεται χρόνος $O(n^2+n)=O(n^2)$.
Μετά από μία ολοκλήρωσης της Tree έχουμε βρει σύνολο K με |K|=1. Άρα, για βρούμε K με |K|=κ θα χρειαστεί να εκτελέσουμε την αναδρομική σχέση κ φορές και μετά από κάθε ολοκλήρωση της, προσθέτουμε στο σύνολο K την κορυφή που ελαχιστοποιεί το μέγιστο κόστος .
Η πολυπλοκότητα του αλγορίθμου είναι $O(n^2 * k^2)$ και η απόδειξη είναι η εξής:

Ο χρόνος που χρειάζεται για τον υπολογισμό της τιμής $Tree(i,j,K)$ $i,j:1\le i\le n,\ 1\le j\le n$ και $|K|=m,0\le m\le k-1$εξαρτάται απο το μέγεθος του συνόλου K. Σε κάθε επανάληψη της αναδρομικής σχέσης, πρέπει να ελέγχουμε αν προσθέτοντας την j στο K, η κορυφή $i\in K$ και ανάλογα από πόρισμα του ελέγχου, να επιστρέψουμε την κατάλληλη τιμή σε χρόνο $O(1)$. Ο χρόνος που απαιτεί ο έλεγχος ισούται με μέγεθος του συνόλου K ,δηλαδή $O(m+1)$ επειδή το i μπορεί να συγκριθεί το πολύ με $m+1$ κορυφές. Για μια επανάληψη της $Tree(i,j,K)$ χρειάζεται χρόνος $O(m+1)$ και συνολικά θα εκτελέσουμε n^2 φορές την αναδρομή. Άρα, για συγκεκριμένο σύνολο K, ο χρόνος εκτέλεσης της Tree είναι $O(n^2*(m+1))$.
Όπως είπαμε και προηγουμένως, θα χρειαστεί να εκτελέσουμε την Tree κ φορές και κάθε φορά το μέγεθος του συνόλου K θα αυξάνει κατά ένα, με $|K|=m,0\le m\le k-1$. Ακόμα, μετά από κάθε ολοκλήρωση της αναδρομής χρειάζεται να βρούμε την κορυφή που μας δίνει το ελάχιστο μέγιστο κόστος και να την προσθέσουμε στο K, $O(n^2)$. Οπότε, προκύπτει πολυπλοκότητα $O\left(n^2*\sum_{m=0}^{k-1}(m+1)+k*n^2\right)=O$¿
¿$O(n^2*k^2+k*n^2)=O(n^2*k^2)$.
Παρακάτω φαίνεται ο ψευδοκώδικας του αλγορίθμου. Θεωρούμε ότι τα στοιχεία έχουν τοποθετηθεί στον πίνακα Tree σύμφωνα με την ταξινόμηση που περιγράψαμε παραπάνω και ο πίνακας parent έχει συμπληρωθεί.

```
Tree(T=(V,E),r,k)
    K←∅
    for z←0 to k do
        for i←0 to n do
            for j←0 to n do
                add j to K
                if i=0 then
                    if i∈K then Tree[0][j]←0
                    else Tree[0][j]←+∞
                else
                    if i∈K then Tree[i][j]←0
                    else Tree[i][j] ← Tree[parent[i][j]]+1
                remove j from K
        for j←0 to n do
            max_value[j]←-∞
            for i←0 to n-1 do
                max_value[j]←max(max_value[j],Tree[i][j])
        min_value←max_value[0]
        for j←0 to n do
            min_value=min(min_value,max_value[j])
        add element with max_value=min_value to K
    return K
```

b)
Έχουμε ένα δυαδικό δέντρο T=(V,E), με |V|=n κορυφές και ρίζα r. Θέλουμε να βρούμε ένα ελάχιστο υποσύνολο κορυφών K τέτοιο ώστε το κόστος $cost(K)=max\ \{cost(u,K)\}\le z$ για κάθε $u\in V$ και $z\le n$.
Για να λύσουμε το πρόβλημα περιγράφω τον παρακάτω άπληστο αλγόριθμο. Αρχικά ξεκινάμε να ελέγχουμε κορυφές με postorder τρόπο, δηλαδή αρχίζουμε από τα φύλλα του δέντρου και κινούμαστε προς τα

πάνω. Θέλουμε $\forall u \in V$ να ισχύει $cost(u,K) \leq z$. Για να συμβαίνει το παραπάνω πρέπει για κάθε κορυφή να υπάρχει ένας τουλάστον πρόγονος σε απόσταση $\leq z$ που να ανήκει στο K και η ρίζα να ανήκει υποχρεωτικά σ' αυτό.
Το άπληστο κριτήριο σύμφωνα με το οποίο βάζουμε κόμβους στο σύνολο K είναι το εξής:
Πρώτα συμπεριλαμβάνουμε απαραίτητα στο K την ρίζα, καθώς σε διαφορετική περίπτωση το κόστος $cost(K) = +\infty$. Έπειτα, ξεκινώντας από τα φύλλα του δέντρου, επιλέγουμε εκείνα που έχουν διαφορετικό πατέρα, βρίσκουμε τον πρόγονο τους που μπορούμε να φτάσουμε με ακριβώς z βήματα και τον προσθέτουμε στο K. Αν για κάποια φύλλα ακολουθώντας την πορεία προς τα πάνω φτάσουμε στον ίδιο πρόγονο τότε τον προσθέτουμε μια μόνο φορά. Όταν βρούμε όλους τους κόμβους που μπορούμε να φτάσουμε από τα φύλλα με ακριβώς z βήματα τότε επαναλαμβάνουμε την ίδια διαδικασία θεωρώντας αυτή την φορά τους κόμβους αυτούς ως φύλλα του δέντρου. Σταματάμε τον αλγόριθμο όταν φτάσουμε από όλα τα φύλλα, τα καινούργια που ορίζουμε κάθε φορά, στην ρίζα όπου και προσθέτουμε τα φύλλα στο K, αποφεύγοντας τα διπλότυπα. Με τον παραπάνω τρόπο εξασφαλίζουμε ότι ισχύει $cost(K) \leq z$ και ότι το K έχει το ελάχιστο μέγεθος.
Το επιχείρημα ανταλλαγής είναι το παρακάτω.
Αν καθώς κινούμασταν από τα φύλλα του δέντρου προς τα πάνω, βάζαμε στο K κόμβους που βρίσκονται τυχαία βήματα $m : m < z$ μακριά από τα φύλλα τότε το κόστος θα ήταν $cost(K) < z$ ωστόσο το μέγεθος του K θα αυξανόταν. Ο λόγος που γίνεται αυτό είναι ότι για $m < k$ διανύουμε κάθε φορά μικρότερη απόσταση από ότι θα μπορούσαμε για z βήματα. Οπότε, χρειαζόμαστε περισσότερους ενδιάμεσους κόμβους, ξεκινώντας κάθε φορά από τα νέα φύλλα, για να φτάσουμε στην ρίζα του δέντρου και αυξάνεται το μέγεθος K. Αν αντί για m βήματα κάναμε z τότε πάλι θα ισχύει $cost(K) \leq z$ αλλά η απόσταση που διανύουμε είναι η μέγιστη επιτρεπτή και το μέγεθος του K το ελάχιστο δυνατό. Οπότε με την αλλαγή στο βήμα ίσο με z καταφέρνουμε λύση ίση με αυτή του optimal αλγόριθμου.
Άρα, το άπληστο κριτήριο μας είναι σωστό.
Η πολυπλοκότητα του αλγορίθμου μας είναι $O(n)$ καθώς διασχίζουμε κάθε κόμβο του δέντρου και απλά ελέγχουμε αν απέχει απόσταση z από τα φύλλα, $O(1)$. Ένας τρόπος για να κάνουμε τον ελέγχο είναι καθώς αναβαίνουμε από τα φύλλα προς τα πάνω να μειώνουμε τον αριμθό z κατά 1 για κάθε πρόγονο που περνάμε. Έτσι όταν φτάσουμε σε τιμή 0 προσθέτουμε στο K τους κόμβους εκείνους και συνεχίζουμε ανανεώνοντας την τιμή ελέγχου πάλι σε z και θέτοντας ως νέα φύλλα τους παραπάνω κόμβους.