ΑΣΚΗΣΗ 1:

Για να υπολογίσουμε ένα έγκυρο υποσύνολο E' με το ελάχιστο συνολικό βάρος, μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο του Kruskal. Ο αλγόριθμος του Kruskal βρίσκει το minimum spanning tree ενός γράφου, το οποίο είναι ένα υποσύνολο των ακμών του γράφου που συνδέει όλους τους κόμβους χωρίς να περιέχει κύκλους.

Ο αλγόριθμος του Kruskal λειτουργεί ως εξής:

1. Ταξινομούμε τις ακμές του γράφου G σε αύξουσα σειρά βάσει του βάρους τους.
2. Αρχικοποιούμε το υποσύνολο E' ως κενό.
3. Για κάθε ακμή e από την ταξινομημένη λίστα ακμών:
    a. Προσθέτουμε την ακμή e στο υποσύνολο E'.
    b. Αν η προσθήκη της ακμής e δεν προκαλεί τη δημιουργία κύκλου στο γράφημα G=(V, E), τότε η ακμή e είναι έγκυρη και προστίθεται στο E'.
4. Επιστρέφουμε το υποσύνολο E'.

Η ορθότητα του αλγορίθμου του Kruskal εγγυάται από την ιδιότητα των minimum spanning tree. Αφού ταξινομήσουμε τις ακμές κατά αύξουσα σειρά βάρους, προσθέτουμε στο υποσύνολο E' μόνο τις ακμές που δεν προκαλούν τη δημιουργία κύκλου. Αφού τελειώσουμε την εκτέλεση του αλγορίθμου, το E' θα περιέχει το ελαχίστων βάρος υποσύνολο των ακμών που δεν περιέχουν κύκλους.

Όσον αφορά την πολυπλοκότητα, ο αλγόριθμος του Kruskal έχει πολυπλοκότητα O(log n), όπου n είναι ο αριθμός των κόμβων του γράφου. Επομένως, η συνολική πολυπλοκότητα του αλγορίθμου του Kruskal είναι O(|E| log |V|), όπου |E| είναι ο αριθμός των ακμών και |V| είναι ο αριθμός των κόμβων του γράφου.

ΑΣΚΗΣΗ 2:

Για την επίλυση του προβλήματος, μπορούμε να χρησιμοποιήσουμε έναν αλγόριθμο με γράφους.

Ας θεωρήσουμε ότι το σύνολο των εργαζομένων αναπαρίσταται από έναν γράφο G, όπου κάθε εργαζόμενος αναπαρίσταται από έναν κόμβο και μια ακμή (i, j).

Ο αλγόριθμος μας θα λειτουργήσει ως εξής:

1. Δημιουργούμε έναν κενό γράφο H.
2. Ξεκινάμε με μια κενή λίστα προσκεκλημένων και μια κενή στοίβα.
3. Επιλέγουμε έναν τυχαίο κόμβο u από τον γράφο G και τον προσθέτουμε στη λίστα προσκεκλημένων.
4. Όσο η λίστα προσκεκλημένων δεν είναι άδεια:
    a. Αφαιρούμε τον πρώτο κόμβο v από τη λίστα προσκεκλημένων.
    b. Αν ο κόμβος v έχει λιγότερους από 4 γείτονες στον γράφο H, τότε:
        i. Επιλέγουμε τους k γείτονές του στον γράφο G που δεν γνωρίζει ο v και τους προσθέτουμε στον γράφο H.
        ii. Προσθέτουμε τους κόμβους-γείτονες στη λίστα.

5. Ο γράφος H που προκύπτει είναι ο υπογράφος του G που περιέχει όλους τους προσκεκλημένους που πληρούν τις απαιτούμενες συνθήκες.

Ο αλγόριθμος μας εξασφαλίζει ότι κάθε προσκεκλημένος έχει τουλάχιστον 4 άλλους προσκεκλημένους που γνωρίζει και τουλάχιστον 4 προσκεκλημένους που δεν γνωρίζει. Αυτό συμβαίνει επειδή κάθε φορά που προσκαλούμε έναν κόμβο, ελέγχουμε τους γείτονές του στον γράφο H και προσκαλούμε μόνο τους γείτονές του που δεν γνωρίζει.

Η πολυπλοκότητα του αλγορίθμου εξαρτάται από το μέγεθος του γράφου G και τον αριθμό των ακμών του. Αν υπάρχουν n κόμβοι και m ακμές, η πολυπλοκότητα του αλγορίθμου είναι O(m), διότι κάθε ακμή επισκέπτεται μόνο μία φορά. Στην χειρότερη περίπτωση, όπου κάθε κόμβος έχει ακριβώς 4 γείτονες, ο αλγόριθμος θα εκτελεστεί σε γραμμικό χρόνο ως προς τον αριθμό των εργαζομένων.

Σημείωση: Αν ο γράφος δεν είναι συνεκτικός, τότε ο αλγόριθμος θα πρέπει να επαναληφθεί για κάθε συνεκτική συνιστώσα του γράφου.

ΑΣΚΗΣΗ 3:

Για την επίλυση αυτού του προβλήματος, μπορούμε να χρησιμοποιήσουμε τεχνικές δυναμικό προγραμματισμό.

Ορίζουμε έναν πίνακα dp μεγέθους n+1, όπου το κελί dp[i] θα αναπαριστά την ελάχιστη ποινή που μπορούμε να επιτύχουμε φτάνοντας στο ξενοδοχείο i. Αρχικά, θέτουμε όλες τις τιμές του πίνακα dp σε άπειρο.

Στη συνέχεια, αρχικοποιούμε το dp[0] με 0, διότι δεν υπάρχει ποινή για την αρχική θέση.

Για κάθε ξενοδοχείο i από το πρώτο μέχρι το τελευταίο, υπολογίζουμε την ελάχιστη ποινή dp[i] ως εξής:

- Ξεκινώντας από το πρώτο ξενοδοχείο, δοκιμάζουμε όλες τις δυνατές αποστάσεις x μεταξύ των δύο συνεχόμενων ξενοδοχείων i και j (όπου j > i).
- Υπολογίζουμε την ποινή για την απόσταση x ως (200 - x)^2.
- Ενημερώνουμε την τιμή dp[j] ως το ελάχιστο από τον τρέχοντα υπολογισμένο όρο και την τιμή dp[i] + ποινή.

Τέλος, η ελάχιστη ποινή για το σύνολο του ταξιδιού θα βρίσκεται στη θέση dp[n], όπου n είναι ο αριθμός των ξενοδοχείων.

Η πολυπλοκότητα του αλγορίθμου είναι O(n^2), διότι χρειαζόμαστε ένα εμφωλευμένο βρόγχο για να εξετάσουμε όλες τις δυνατές αποστάσεις μεταξύ των ξενοδοχείων. Ο πίνακας dp έχει μέγεθος n+1 και υπολογίζουμε την τιμή για κάθε θέση μία φορά, επομένως η συνολική πολυπλοκότητα είναι αποδεκτή για μεσαίους αριθμούς ξενοδοχείων.

Ο αλγόριθμος μας επιλέγει τη βέλτιστη ακολουθία ξενοδοχείων στην οποία πρέπει να σταθμεύσουμε, ώστε να ελαχιστοποιήσουμε τη συνολική ποινή.

ΑΣΚΗΣΗ 4:

Για και για την επίλυση αυτού του προβλήματος θα χρησιμοποιήσουμε τεχνικές δυναμικό προγραμματισμό.

Ορίζουμε έναν πίνακα dp μεγέθους (k+1) × (v+1), όπου το κελί dp[i][j] θα αναπαριστά αν είναι δυνατό να πραγματοποιηθεί ανταλλαγή με i νομίσματα και τιμή j. Αρχικά, θέτουμε όλες τις τιμές του πίνακα dp σε false.

Στη συνέχεια, αρχικοποιούμε το dp[0][0] με true, διότι με κανένα νόμισμα δεν μπορούμε να φτάσουμε στην τιμή 0.

Για κάθε νόμισμα x[i] από το πρώτο μέχρι το τελευταίο, και για κάθε δυνατή τιμή j από 0 έως v, υπολογίζουμε την τιμή του κελιού dp[i][j] ως εξής:

- Αν το νόμισμα x[i] είναι μικρότερο ή ίσο της τιμής j, τότε μπορούμε να πραγματοποιήσουμε ανταλλαγή χρησιμοποιώντας αυτό το νόμισμα, εφόσον μπορούμε να φτάσουμε στην τιμή j - x[i] χρησιμοποιώντας (i-1) νομίσματα.
- Αλλιώς, δεν μπορούμε να χρησιμοποιήσουμε το νόμισμα x[i] για την ανταλλαγή και πρέπει να εξετάσουμε το αντίστοιχο κελί στην προηγούμενη γραμμή dp[i-1][j].

Στο τέλος της διαδικασίας, αν το κελί dp[k][v] είναι true, τότε υπάρχει δυνατότητα να πραγματοποιηθεί ανταλλαγή με k ή λιγότερα νομίσματα για την τιμή v. Αντίστοιχα, μπορούμε να ανακτήσουμε τη λύση παρακολουθώντας την αλυσίδα από τα κελιά dp.

Η ορθότητα του αλγορίθμου προκύπτει από το γεγονός ότι εξετάζουμε όλες τις δυνατές συνδυασμούς των νομισμάτων και των τιμών. Η πολυπλοκότητα του αλγορίθμου είναι O(kv), διότι εξετάζουμε όλα τα κελιά του πίνακα dp μία φορά.