Θεμελιώδη Θέματα Επιστήμης Υπολογιστών

2η Σειρά Ασκήσεων

Χρήστος Τσούφης
ΣΗΜΜΥ

Άσκηση 1

α) $L_a = \{ 0^i 1^j \mid i,j \geq 1$ και $i \neq j \}$

Γνωρίζουμε ότι οι κανονικές γλώσσες παρουσιάζουν κλειστότητα ως προς την πράξη του συμπληρώματος. Δηλαδή, αν L_a είναι κανονική τότε και το συμπλήρωμά της $\overline{L_a}$ θα είναι κανονική γλώσσα. Υποθέτουμε ότι το συμπλήρωμα είναι κανονική γλώσσα και επομένως, εφαρμόζουμε PL. Με βάση το PL, $\exists n \notin \mathbb{N}$ ώστε να ισχύει το συμπέρασμά του. Επιλέγουμε $z = 0^n 1^n \in \overline{L}$ και $|z| = 2n > n$. Λόγω PL υπάρχει σπάσιμο της z σε u, v, w με $|uv| \leq n$ και $|u| \geq 1$. Τότε $v = 0^k$ για κάποιο $k \in \mathbb{N}$. Όμως πρέπει $uv^iw \in \overline{L_a}$ για κάθε $i \in \mathbb{N}$. Όμως, για οποιοδήποτε $i \geq 2$ ισχύει $uv^iw \notin \overline{L_a}$. Συνεπώς, η $\overline{L_a}$ δεν είναι κανονική και άρα η L_a δεν είναι κανονική.

β) $L_\beta = \{ a^n b^m \mid n$ είναι τέλειο τετράγωνο$\} = \{ a^{n^2} b^m \}$

Υποθέτουμε ότι η L_β είναι κανονική. Από PL θα υπάρχει $k \in \mathbb{N}$. Επιλέγουμε $z = a^{k^2} b^m$ όπου $z \in L$, $|z| = k+m \geq k$ και k τέλειο τετράγωνο. Λόγω PL η z γράφεται $z = uvw$ με $|uv| \leq k$, $|v| \geq 1$.

$$\underbrace{aa\ldots a}_{u=k^2-m}\ \underbrace{aa}_{m}\ \underbrace{bb\ldots b}_{w}$$

Επιλέγω $i=2 \Rightarrow uvvw = \underbrace{aa\ldots a}_{n^2+m} bb\ldots b \notin L_\beta$ ΑΤΟΠΟ

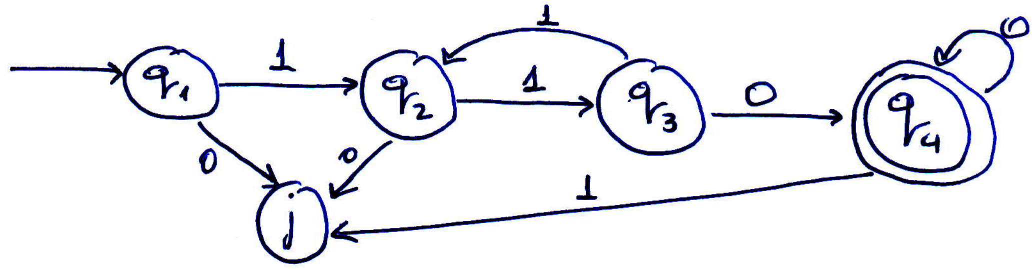
Άρα η L_β δεν είναι κανονική γλώσσα διότι $n^2 < n^2+m \leq n^2+n = n(n+1) < (n+1)^2$ και n^2+m : όχι τέλειο τετράγωνο γ) $L_\gamma = \{ 1^{2k} 0^{j+1} \mid k \geq 1 \text{ και } j \geq 0 \}$

Με χρήση PL, υποθέτουμε ότι η L_γ είναι κανονική. Από PL $\exists n \in \mathbb{N}$. Διαλέγουμε $z = 1^{2n} 0^{j+1}$. Λόγω PL η z γράφεται $z = uvw$ με $|uv| \leq n$, $|v| \geq 1$

$$\underbrace{\underbrace{11\ldots}_{2n-n}}_{u}\underbrace{\underbrace{11}_{n}}_{v}\underbrace{\underbrace{00\ldots0}_{j+1}}_{w}$$

Δέχεται να γίνει "pumped" αλλά με τη χρήση του PL δεν μπορεί να χαρακτηριστεί ως μη κανονική.

Η L_γ είναι γλώσσα που αποδέχεται κάθε αλφαριθμητικό η οποία έχει άρτιο πλήθος άσσων ακολουθούμενοι από μηδενικά. Επομένως αναπαρίσταται από το DFA:



δ) $L_\delta = \{ 0^{k+2} 1^{k} 0^{k-2} \mid k \geq 1 \}$

Υποθέτουμε ότι η L_δ είναι κανονική. Από PL $\exists n \in \mathbb{N}$. Διαλέγουμε $z = 0^{n+2} 1^{n} 0^{n-2}$. Λόγω PL η z γράφεται $z = uvw$ με $|uv| \leq n$, $|v| \geq 1$.

$$\underbrace{\underbrace{00\ldots0}_{n-2}\underbrace{00}_{2}}_{u}\;001\cdot\underbrace{1\ldots11}_{v\;n}\;\underbrace{00\ldots0}_{w\;n-2}$$

Επιλέγουμε $i=2$ οπότε $uv^2w = 0^{n+4} 1^{n} 0^{n-2} \notin L_\delta$ Άτοπο

Άρα, η L_δ δεν είναι κανονική.

Άσκηση 2

a) $G: S \longrightarrow aS \mid aSbS \mid \varepsilon$

Νδο $L(G) = \{ x \mid$ κάθε πρόθεμα του x έχει τουλ. τόσα a όσα $b \}$

Οι κανόνες παραγωγής της γραμματικής G είναι οι εξής:
$S \to aS \vee aSbS \vee \varepsilon$. Ο κανόνας $S \to aS$ δημιουργεί πρόθεμα από a, ο κανόνας $S \to \varepsilon$ ολοκληρώνει την παραγωγή της συμβολοσειράς και ο κανόνας $S \to aSbS$ παράγει και a και b. Όμως, το συνολικό πλήθος των a της παραγόμενης λέξης ισούται με τον αριθμό των παραγωγών, που υπακούουν στους κανόνες $S \to aS$ και $S \to aSbS$, ενώ το πλήθος των b ισούται με τον αριθμό των παραγωγών που ακολουθούν τον κανόνα $S \to aSbS$. Άρα, το συνολικό πλήθος των a είναι μεγαλύτερο ή ίσο του πλήθους των b. Τέλος, ο κανόνας $S \to aSbS$ αποτελεί βεβαίωση ότι τα b που παράγονται από αυτόν θα βρίσκονται δεξιότερα από τα a που παράγονται από αυτόν και συνεπώς, κάθε λέξη που παράγεται από αυτή τη γραμματική ικανοποιεί την ιδιότητα της γλώσσας L και άρα η γραμματική G παράγει την γλώσσα L.

β) $L = \{ a^{3n} b^{2n} \mid n \geq 1 \}$

$G: V = \{S\}, T = \{a, b\}, P = \{ S \to aaaSbb \vee aaabb \}$

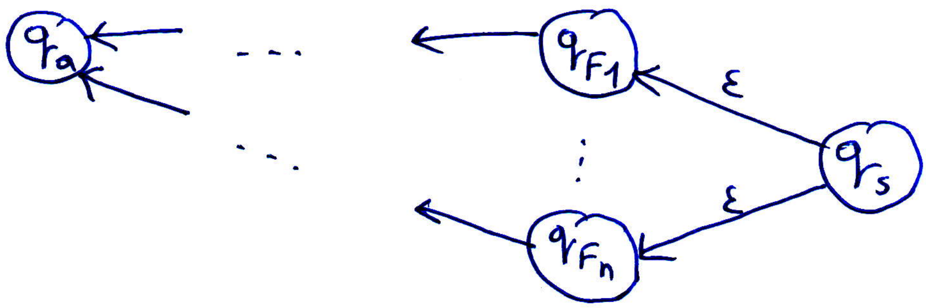
γ) $L = \{ w \in \{0,1\}^* \mid$ το w έχει περιττό μήκος και (ακριβώς) στη μέση $111 \}$

$G: V = \{S\}, T = \{0, 1\}, P = \{ S \to 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid 111 \}$

Άσκηση 3

α) Έστω L μια κανονική γλώσσα και $M = \{Q_M, \Sigma_M, \delta_M, q_a, F_M\}$ το αυτόματο που την αναγνωρίζει. Έστω η συνάρτηση μετάβασης δ_M για την οποία ισχύει: $p \in \delta_M(q, a) \Leftrightarrow q \in \delta_M(p, a)$ όπου a σύμβολο του Σ_M. Αυτή η συνάρτηση μετάβασης επιβεβαιώνει ότι εάν υπάρχει μονοπάτι w από μια κατάσταση P_1 σε μια άλλη q_1 στο M, τότε υπάρχει και μονοπάτι από την q_1 στην P_1, το οποίο μάλιστα είναι το αντίστροφο του w, δηλ. το w^R. Με αυτή τη συνάρτηση μετάβασης θα κατασκευάσουμε NF, το οποίο θα αποδέχεται την ανάστροφη της L. Το NFA$_\varepsilon$ είναι το εξής:

$M^R = \{Q_M \cup \{q_s\}, \Sigma_M, \delta_M, q_s, \{q_a\}\}$ όπου $q_s \notin Q_M$ και με ε-κίνηση μπορούμε να πάμε από την q_s σε όλες τις F_M.



β) Έστω δυο CF γλώσσες L_1, L_2 που παράγονται από τις γραμματικές $G_1 = \{V_1, T_1, P_1, S_1\}$ και $G_2 = \{V_2, T_2, P_2, S_2\}$, αντίστοιχα. Σχηματίζονται οι εξής γραμματικές:

- $G_3 = \{V_1 \cup V_2 \cup \{S\}, T_1 \cup T_2, \{S \to S_1 | S_2, P_1, P_2\}, \{S\}\}$ για την οποία ισχύει $L(G_3) = L(G_1) \cup L(G_2)$ και άρα ισχύει η κλειστότητα ως προς την ένωση.

- $G_4 = \{V_1 \cup \{S\}, T_1, \{S \to S S_1 | \varepsilon, P_1\}, \{S\}\}$ για την οποία ισχύει $L(G_4) = L(G_1)^*$ και άρα ισχύει η κλειστότητα ως προς το άστρο Kleene.

Γνωρίζουμε πως οι γλώσσες $L_5=\{a^nb^nc^m$, για $n,m\in\mathbb{N}\}$ και $L_6=\{a^nb^mc^m$, για $n,m\in\mathbb{N}\}$ είναι CF. Αν ισχύει η κλειστότητα ως προς το συμπλήρωμα για τις CF, έχ. οι $\overline{L_5}$ και $\overline{L_6}$ είναι CF και άρα και η ένωσή τους είναι CF, τότε θα έπρεπε και η $\overline{\overline{L_1}\cup\overline{L_6}} = L_5\cap L_6 = \{a^nb^nc^n, n\in\mathbb{N}\}$ να είναι CF, για την οποία όμως γνωρίζετε ότι δεν είναι CF. Άρα δεν ισχύει η κλειστότητα ως προς την πράξη του συμπληρώματος.

Έστω η γραμματική $G_5=\{V_1,T_1,P_1^R,S_1\}$, όπου για κάθε παραγωγή $X\to w$, που ανήκει στο P_1^R και $V\in S_1$ και το W είναι συμβολοσειρά, περιέχει τόσο τερματικά από το V_1 όσο και μη τερματικά από το T_1. Ισχύει $L(G_5)=L(G_5)^R$ και άρα ισχύει η κλειστότητα ως προς την αναστροφή.

Άσκηση 4:

a) Για να βρούμε κλειστό τύπο, δουλεύουμε ως εξής:

Ο ψευδοκώδικας του αναδρομικού αλγορίθμου είναι:

```
procedure move_anoi (n from X to Y using Z)
begin
  if n=1 then
     move top disk from X to Y ;
  else
     move_anoi (n-1 from X to Z using Y);
     move top disk from X to Y;
     move_anoi (n-1 from Z to Y using X);
end
```

Παρατηρούμε ότι όταν η διαδικασία move_anoi καλείται για $n>1$ τότε κάνει κατά σειρά κλήση της move_anoi για $n-1$ για μετακίνηση των $n-1$ δίσκων από το X στο Y, μετακίνηση του δίσκου από το X (αρχικό πύργο) στο Z (τελικό πύργο) και κλήση της move_anoi για $n-1$ για μετακίνηση των $n-1$ δίσκων από το Y στο X. Αν θεωρήσουμε ότι ο αριθμός των μετακινήσεων για n δίσκους είναι $T(n)$, τότε προκύπτει το αναδρομικό σχήμα:

$$T(n) = T(n-1) + 1 + T(n-1) = 2T(n-1) + 1$$

Λύνοντας το παραπάνω αναδρομικό σχήμα:

$$T(n) = 2T(n-1) + 1 = 2\left[2T(n-2) + 1\right] + 1 = 2T(n-2) + 2 + 1 = \dots =$$

$$= 2^{(n-1)}T(1) + 2^{(n-2)} + \dots + 2 + 1$$

Όμως $T(1) = 1$ και άρα η παραπάνω σχέση γίνεται:

$$T(n) = 2^{n-1} + 2^{n-2} + \dots + 2 + 1 = \sum_{k=0}^{n-1} 2^k = 2^n - 1$$

Άρα τελικά $T(n) = 2^n - 1$.

β) Ο επαναληπτικός αλγόριθμος επίλυσης των πύργων Hanoi εκτελεί πάντα τα εξής δύο βήματα μέχρι να φτάσουν όλοι οι δίσκοι στο στόχο: πρώτα, μετακινεί το μικρότερο δίσκο προς την ίδια κατεύθυνση κάθε φορά και μετά μετακινείται ο αμέσως μεγαλύτερος δίσκος για τον οποίον υπάρχει επιτρεπτή κίνηση.

Με χρήση μαθηματικής επαγωγής, θδο ο επαναληπτικός αλγόριθμος επίλυσης κάνει ακριβώς τα ίδια βήματα με τον αναδρομικό.

Για $n=1$ δίσκο, και οι δύο αλγόριθμοι κάνουν μια κίνηση (μετακίνηση του μικρότερου δίσκου).

Υποθέτουμε ότι ισχύει για n δίσκους. Θδο ισχύει για $n+1$ δίσκους.

Ο αναδρομικός αλγόριθμος για $n+1$ θα κάνει κλήση για να μετακινήσει τους n δίσκους από το X στο Z, θα μετακινήσει τον τελευταίο δίσκο από το X στο Y και τέλος θα μετακινήσει τους n δίσκους από το Z στο Y.

Χρησιμοποιώντας την επαγωγική υπόθεση ότι για την μετακίνηση των n δίσκων οι δυο αλγόριθμοι κάνουν τα ίδια βήματα, βλέπουμε ότι έπειτα από τη μετακίνησης των n πρώτων δίσκων από το X στο Z, η επόμενη κίνηση του επαναληπτικού αλγόριθμου θα είναι η μετακίνηση του n+1 δίσκου από το X στο Y ως η μόνη επιτρεπτή κίνηση και έπειτα πάλι με χρήση του επαγωγικού βήματος ο επαναληπτικός αλγόριθμος κάνει πάλι όσες μετακινήσεις και ο αναδρομικός για τη μετακίνηση των n δίσκων από το Z στο Y.

Άρα, αν ονομάσουμε A τον αριθμό μετακινήσεων που θεωρήσαμε στο επαγωγικό βήμα, και οι δύο αλγόριθμοι χρειάζονται 2A+1 βήματα για να λύσουν το πρόβλημα των n+1 δίσκων και άρα, τελικά, οι δυο αλγόριθμοι κάνουν τον ίδιο αριθμό μετακινήσεων.

γ) Με μαθηματική επαγωγή μπορούμε ~~να~~ εύκολα να αποδείξουμε πως ο ελάχιστος αριθμός κινήσεων για τους πύργους είναι από τον ίδιο τύπο που βρήκαμε για τον αναδρομικό αλγόριθμο, δηλαδή:
$M(n) = 2^n - 1$.

Για $n=1$, αρκεί μια κίνηση για την μετακίνηση του δίσκου από τον πάσσαλο X στον Z και επίσης $M(1)=1$.

Θ.δ.ο. ισχύει για n+1.

Για να γίνει η μετακίνηση του n+1 δίσκου πρέπει πρώτα να έχουν μετακινηθεί οι πρώτοι n δίσκοι και να έχουν σχηματίσει πύργο στον πάσσαλο Y, δηλ. χρειάζονται $M(n)$ κινήσεις. Έπειτα, μετακινούμε τον n+1 δίσκο από τον X στον Z, με κόστος κίνησης 1. Έπειτα, πρέπει να μετακινήσουμε τους n δίσκους από τον Y στον Z, με κόστος πάλι $M(n)$.

Άρα, συνολικά
$$M(n+1) = M(n) + 1 + M(n) = 2^n - 1 + 1 + 2^n - 1 =$$
$$= 2 \cdot 2^n - 1 = 2^{n+1} - 1$$
Συνεπώς ισχύει για n+1.

Άρα, ισχύει η επαγωγική υπόθεση και ο ελάχιστος αριθμός μετακινήσεων για τους πύργους Hanoi ταυτίζεται με τον αριθμό κινήσεων που δίνει ο αναδρομικός αλγόριθμος επίλυσης του προβλήματος και ισούται με: $M(n)=2^n-1$

Άσκηση 5

Ένα literal είναι μια ανεξάρτητη μεταβλητή. Ικανοποιείται με ανάθεση T αν δεν έχει τελεστή ¬, και με ανάθεση F, αν έχει τελεστή ¬.
Ένα clause είναι μια σύζευξη literals. Ικανοποιείται όταν δεν υπάρχει ένα literal και την άρνησή του.
Μια DNF είναι μια διάζευξη clauses. Ικανοποιείται αν υπάρχει ένα ικανοποιήσιμο clause.
Ένα πρόγραμμα που αποφαίνεται για την ικανοποιησιμότητα μιας DNF:

→ Είσοδος: Μια DNF
→ Διάβασε: ένα-ένα τα clauses
→ Αν ένα clause δεν περιέχει ένα literal και την άρνησή του, ανάθεσε T στα literals χωρίς τελεστή ¬, F στα literals με τελεστή ¬ και αυθαίρετες τιμές στα literals που δεν υπάρχουν στο clause, επίστρεψε "ΝΑΙ" και τις ανατεθειμένες τιμές και τερμάτισε το πρόγραμμα.
→ Αν τελειώσει η ανάγνωση της εισόδου πριν τερματίσει το πρόγραμμα επίστρεψε "όχι" και τερμάτισε το πρόγραμμα.

Άσκηση 6

Για την εύρεση της ελάχιστης τιμής προσβάσιμου κόμβου από οποιονδήποτε κόμβο ακολουθούμε τον ακόλουθο αλγόριθμο της c++:

```
int minDepth(int u) {
    if (ans[u] != 0) return ans[u];
    ans[u] = vari[u];
    for (int i=0; i<n; i++) {
        if (adj[u][i] == 1) {
            ans[u] = min(ans[u], minDepth(i));
        }
    }
    return ans[u];
}
```

a) Για άκυκλους κατευθυνόμενους γράφους καλούμε την minDepth ως εξής:

```
for (int i=0; i<n; i++) {
    if ( ans[i]==0)
        ans[i] = minDepth(i);
}
for (int i=0; i<n; i++)
    cout << ans[i]<<" ";
```

Εκ πρώτης όψεως φαίνεται να έχει πολυπλοκότητα $O(n^2)$ όμως ο πίνακας ans λειτουργεί ως πίνακας memoization. Δηλαδή όταν βρίσκουμε την ζητούμενη ελάχιστη τιμή για έναν κόμβο αυτή αποθηκεύεται στον πίνακα. Έτσι, όταν ξανακαλείται αναδρομικά η συνάρτηση για αυτό τον κόμβο δεν επαναλαμβάνει την διαδικασία, αλλά επιστρέφει την έτοιμη τιμή. Αυτό έχει ως αποτέλεσμα η διαδικασία της minDepth να εκτελείται μόνο μια φορά για κάθε κόμβο. Η πιο πάνω τακτική βασίζεται στο γεγονός ότι η ζητούμενη ελάχιστη τιμή για κάθε κόμβο εξαρτάται μόνο από τους "απογόνους κόμβους του" (για DAG).

β) Για γενικούς κατευθυνόμενους γράφους καλούμε την ίδια συνάρτηση minDepth με τον ακόλουθο τρόπο:

```
for(int i=0; i<n; i++) {
    cout << minDepth (i)<<" ";
    for (int j=0; j<n; j++)
        ans[j] = 0;
}
```

Αυτή τη φορά δεν ισχύει απόλυτα το γεγονός ότι η ζητούμενη ελάχιστη τιμή για κάθε κόμβο εξαρτάται μόνο από τους "απογόνους κόμβους του" διότι υπάρχει το ενδεχόμενο να γίνει κύκλος.

Σε εκείνη την περίπτωση πρέπει να ελεγχθούν όλοι οι επόμενοι κόμβοι μέχρι ο κύκλος να επιστρέψει στον κόμβο. Επειδή η συνάρτηση είναι αναδρομική και κάτι τέτοιο θα ήταν περίπλοκο μπορούμε να καλέσουμε την minDepth για κάθε κόμβο με άδειο τον πίνακα όλες κάθε φορά. Ο αλγόριθμος πλέον απαιτεί την n επαναλήψεων minDepth για κάθε κόμβο ανεβάζοντας την πολυπλοκότητα σε $O(n^2)$.

Άσκηση 7

a) Για $k=1$, είναι προφανές πως δεν υπάρχει περίπτωση να μην δοκιμαστούν γραμμικά όλα τα ύψη (σε εκατοστά) από $h=1$ ως $h=n-1$, άρα βέλτιστα, θα χρειαστούμε $n-1$ δοκιμές.

Για $k=2$, το ένα ποτήρι θα ορίζει ένα διάστημα εκατοστών, ξεκινώντας και πάλι από $h=1$, μέσα στο οποίο, το άλλο ποτήρι θα αναζητά (γραμμικά και πάλι) με ακρίβεια εκατοστού το ύψος που αναζητείται. Το πρώτο ποτήρι θα απορρίπτει σταδιακά τα διαστήματα αναζήτησης του δεύτερου ποτηριού, κάτι το οποίο είναι απαραίτητο αφού εφ' όσον μείνει μόνο το δεύτερο, η γραμμική αναζήτηση είναι μονόδρομος.

Η βέλτιστη λύση θα επιτευχθεί όταν οι δοκιμές που χρειαζόμαστε είναι σταθερές σε οποιαδήποτε περίπτωση για το ύψος που σπάνε τα ποτήρια, γιατί για παράδειγμα, αν δοκιμάζουμε με το ένα ποτήρι ανά δεκάδες, στη χειρότερη περίπτωση χρειάζονται $N=9+9=18$ δοκιμές (τα ποτήρια σπάνε για $h=n=100$). Όμως, αν τα ποτήρια σπάνε για $h \leq 10$, χρειάζονται στην χειρότερη περίπτωση (για $h=9$) $N=1+8=9$ δοκιμές, άρα επιδέχεται βελτίωση η μέθοδος. Το ζητούμενο επιτυγχάνεται όταν το ένα ποτήρι δοκιμάζει με διαστήματα τα οποία μειώνονται κατά 1 σε κάθε επόμενη δοκιμή, έτσι ώστε οι δοκιμές στην γραμμική αναζήτηση του δεύτερου ποτηριού + οι δοκιμές του πρώτου = σταθερό (πρέπει βέβαια και να ξεκινάμε με το ελάχιστο δυνατό διάστημα κατά την πρώτη δοκιμή.)

Για $n=100$ η βέλτιστη λύση είναι λοιπόν για αρχικό ύψος 14 εκατοστών..
Ο αριθμός δοκιμών στη χειρότερη περίπτωση που χρειάζονται είναι $N=14$.

β) Αναπαριστούμε ως $trials(n,k)$ τη λύση του προβλήματος (δοκιμές στην χειρότερη περίπτωση) για μέγιστο ύψος n και k διαθέσιμα ποτήρια. Θα συσχετίσουμε τη λύση $trials(n,k)$ με μικρότερα υποπροβλήματα (τα οποία έχουμε υπολογίσει). Αναζητούμε από ποιο εκατοστό να κάνουμε μια δοκιμή έτσι ώστε να έχουμε την ελάχιστη χειρότερη περίπτωση ως προς τις δοκιμές.

· Έστω ότι δοκιμάζουμε από ύψος h λοιπόν. Αν το ποτήρι σπάσει, κάνουμε μια δοκιμή + όσες δοκιμές χρειαστούν για να βρούμε ~~τη~~ λύση με ένα λιγότερο ποτήρι, ως ύψος $h-1$, δηλαδή $1+trials(h-1,k-1)$. Αν το ποτήρι δεν σπάσει, κάνουμε πάλι μια δοκιμή + όσες χρειαστούν για να βρούμε τη λύση για ύψος h': $h \le h' < n$, δηλαδή $1+trials(n-h,k)$. Επειδή χρειαζόμαστε την χειρότερη περίπτωση μεταξύ των δύο λοιπόν, είναι:

$$trials(n,k) = \min\left(\max\left(1+trials(h-1,k-1),\ 1+trials(n-h,k)\right)\right), \quad 1 \le h < n$$

και αρχικές συνθήκες $trials(n,1)=n$, $trials(1,k)=0$.

γ) Θα χρησιμοποιήσουμε memoization και θα χτίσουμε τις απαντήσεις σταδιακά ως εξής:

```
#include <iostream>
#include <cmath>
#include <climits>
using namespace std;
const unsigned N=100, K=2;
int main ()
{
    int dp[N+1][K+1];
    for (int i=1; i<=N; i++) dp[i][1] = i;
    for (int i=1; i<=K; i++) dp[1][i] = 0;
```

```
for (int n=2; n<=N; n++)
{
    for (int k=2; k<=K; k++)
    {
        int m= INT_MAX;
        for (int h=1; h<=n; h++)
        {
            int ma = max(dp[h-1][k-1], dp[n-h][k]);
            if (ma<m) m=ma;
        }
        dp[n][k] = 1+m;
    }
}
cout << dp[N][K] << endl;
return 0;
}
```

Άσκηση 8

Δεδομένα:

- ένας adjacency matrix με βάρη που δείχνει την απόσταση από κάθε πόλη σε κάθε πόλη. Στην περίπτωση που ο αρχικός adj matrix δεν περιέχει τις ελάχιστες αποστάσεις από κάθε πόλη προς κάθε πόλη τότε εφαρμόζουμε έναν αλγόριθμο. (N φορές τον dijkstra ή τον Bellman-Ford ή μία φορά τον Floyd-Worshall) για να βρούμε τις ελάχιστες αποστάσεις.
- Ένας αριθμός k που είναι ο μέγιστος αριθμός πυροσβεστικών σταθμών που μπορούν να τοποθετηθούν.
- Ένας αριθμός D που είναι η μέγιστη επιτρεπτή απόσταση μίας πόλης από τον πλησιέστερο πυροσβεστικό σταθμό.

Βήμα 1

Διατρέχουμε ολόκληρο τον adj matrix και αντικαθ. τις τιμές που έχουν τα βάρη με : (κόστος n^2)

1, αν $adj[i,j] \leq D$

0, αν $adj[i,j] > D$

Βήμα 2

Επιλύουμε το πρόβλημα Dominating Set για τον νέο adj matrix. Παίρνουμε σαν αποτέλεσμα έναν αριθμό dom όπου dom είναι ο ελάχιστος αριθμός κόμβων που δημιουργούν Dom. Set.

Βήμα 3

Ελέγχουμε αν $k \geq dom$, τότε υπάρχει λύση για το firestation Problem

$k < dom$, τότε δεν υπάρχει λύση

Συμπέρασμα:

Με την πιο πάνω διαδικασία έχουμε αναγωγή το πρόβλημα Dom. Set στο πρόβλημα firestation Placement με κόστος n^2.

Αφού γνωρίζουμε ότι το πρόβλημα Dom. Set είναι NP- πλήρες τότε προκύπτει ότι και το firestation placement είναι NP- πλήρες.