Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Θεμελιώδη Θέματα Επιστήμης Υπολογιστών, 2020-21

1η σειρά γραπτών ασκήσεων

(αυτόματα – τυπικές γλώσσες – γραμματικές
λογική – υπολογισιμότητα – πολυπλοκότητα)

Άσκηση 1.

Σχεδιάστε πεπερασμένο αυτόματο με αλφάβητο $\{0, ..., 9\}$ που να διαβάζει τα ψηφία ενός ακεραίου αριθμού n (τα περισσότερο σημαντικά πρώτα) και να αποδέχεται αν $n \bmod 5 = 3$.

Άσκηση 2.

Δίνονται οι παρακάτω γλώσσες:

$L_1 = \{w \in \{a, b, c\}^* \mid$ η w περιέχει την συμβολοσειρά $'bac'$ και όχι τη συμβολοσειρά $'bacc'\}$.

$L_2 = \{w \in \{0, 1\}^* \mid$ η w είναι δυαδική αναπαράσταση ακεραίου της μορφής $4^k + 1$, για $k \geq 1\}$

$L_3 = \{w \in \{a, b\}^* \mid$ η w έχει μήκος τουλάχιστον 4 και στην προ-προτελευταία θέση περιέχει $'b'\}$

(i) Κατασκευάστε DFA με όσο το δυνατόν λιγότερες καταστάσεις για τις γλώσσες L_1 και L_2:

Υπόδειξη: όπου βοηθάει, σχεδιάστε και συνδυάστε αυτόματα για απλούστερες γλώσσες.

(ii) Δώστε κανονική παράσταση για τις γλώσσες L_1 και L_2.

(iii) Κατασκευάστε NFA για την γλώσσα L_3 και βρείτε το ισοδύναμο DFA. Είναι το ελάχιστο; Αποδείξτε. Αν όχι, βρείτε το.

Άσκηση 3.

Είναι κανονικές οι παρακάτω γλώσσες; Αν μια γλώσσα δεν είναι κανονική, να το αποδείξετε χρησιμοποιώντας είτε το Λήμμα Άντλησης είτε κάποια ιδιότητα κλειστότητας. Αν μια γλώσσα είναι κανονική, να το αιτιολογήσετε κατάλληλα.

α) $\{0^n 1^+ 0^m : n, m \geq 1, n \neq m\}$.

β) $\{0^n 1^+ 0^m : n, m \geq 1, n = m\}$.

γ) $\{0^+ 1^+ 0^+\}$.

δ) $\{ww : w \in \{0, 1\}^*$ το μήκος της w είναι $\leq 2^{2^{1000}}\}$.

Σημείωση: 1^+ *είναι η συμβολοσειρά* 11^**, και* 0^+ *είναι η συμβολοσειρά* 00^*.

Άσκηση 4.

(α) Έστω $G : S \to aaA, A \to aa \mid aaA \mid B, B \to b \mid bB$. Περιγράψτε σε φυσική γλώσσα τη γλώσσα που παράγει η G .

(β) Δώστε γραμματική για τη γλώσσα $\{w \in \{0, 1\}^* \mid$ το w δεν περιέχει δύο συνεχόμενα $0\}$.

(γ) Δώστε γραμματική για τη γλώσσα $\{w \in \{0, 1\}^* \mid$ το πλήθος των 1 στο w είναι πολλαπλάσιο του 3 $\}$.

Άσκηση 5.

Εξετάστε αν η κλάση των γλωσσών χωρίς συμφραζόμενα είναι κλειστή ως προς τις πράξεις **ένωση** και **παράθεση**.

Εξάσκηση σε αυτόματα

Εξασκηθείτε στο σχεδιασμό και κατανόηση λειτουργίας των DFA, NFA και NFA$_\varepsilon$ χρησιμοποιώντας το εργαλείο που θα βρείτε στη σελίδα `http://automata.discrete.gr/` (Ευχαριστίες στους δημιουργούς, απόφοιτους της ΣΗΜΜΥ, Μανόλη Ζαμπετάκη και Διονύση Ζήνδρο).

Επαληθεύστε την ορθή λειτουργία των αυτομάτων που σχεδιάσατε στις προηγούμενες ασκήσεις (όπου γίνεται) με χρήση του εργαλείου αυτού.

Προαιρετικά: ελάτε σε επαφή με τους δημιουργούς της εφαρμογής για να συμβάλετε στην ανάπτυξη νέων λειτουργιών ή/και βελτίωση του interface.

Άσκηση 6. (Υπολογισιμότητα)

Αποδείξτε ότι το πρόβλημα του ελέγχου αν ένα πρόγραμμα τερματίζει με είσοδο την κενή συμβολοσειρά είναι μη επιλύσιμο.

Υπόδειξη: Δείξτε ότι αν υπάρχει πρόγραμμα Π που να παίρνει σαν είσοδο ένα οποιοδήποτε πρόγραμμα Π' και να αποφαίνεται αν το Π' τερματίζει με είσοδο την κενή συμβολοσειρά, τότε θα μπορούσαμε να λύσουμε το Πρόβλημα Τερματισμού τροφοδοτώντας το Π με κατάλληλη είσοδο Π'.

Άσκηση 7. (Λογική και Αλγόριθμοι)

Διατυπώστε αποδοτικό αλγόριθμο που να δέχεται σαν είσοδο οποιονδήποτε τύπο της προτασιακής λογικής σε μορφή Horn και να αποφαίνεται αν είναι ικανοποιήσιμος. Σε περίπτωση που είναι θα πρέπει να επιστρέφει μία ανάθεση αληθοτιμών που ικανοποιεί τον τύπο.

Π.χ. με είσοδο $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_4 \vee x_5 \vee \neg x_6) \wedge (x_6)$ θα πρέπει να επιστρέφει 'Ναι' και μία από τις αναθέσεις αληθοτιμών στις $(x_1, \ldots, x_6)$ που ικανοποιούν τον τύπο, π.χ. την ανάθεση $(\mathsf{True}, \mathsf{False}, \mathsf{True}, \mathsf{False}, \mathsf{True}, \mathsf{True})$ ενώ με είσοδο $(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2)$ θα πρέπει να επιστρέφει 'Όχι'.

Θεωρήστε ότι όλες οι μεταβλητές ενός τύπου δίνονται στη μορφή x_n, όπου n ένας φυσικός αριθμός.

Άσκηση 8. (Πολυπλοκότητα: αναγωγές προς απόδειξη δυσκολίας)

Έστω ένα μη κατευθυνόμενο γράφημα $G = (V, E)$. **Κλίκα** λέγεται ένα υπογράφημα του G το οποίο είναι πλήρες (δηλ. όλες οι κορυφές του ενώνονται με ακμή). Ένα σύνολο κορυφών του G λέγεται **ανεξάρτητο σύνολο** αν κάθε δύο κορυφές του δε συνδέονται με ακμή. Θεωρήστε τα προβλήματα απόφασης: (i) **Independent Set**, το οποίο δεδομένης εισόδου (G, k) έχει θετική απάντηση αν και μόνο αν το γράφημα G περιέχει κάποιο ανεξάρτητο σύνολο μεγέθους τουλάχιστον k, (ii) **Clique**, το οποίο δεδομένης εισόδου (G, k) έχει θετική απάντηση μεγέθους τουλάχιστον k και (iii) **Dense Subgraph**, το οποίο δεδομένου ενός γραφήματος G και δύο θετικών ακέραιων a, b έχει θετική απάντηση αν και μόνο αν το G περιέχει ένα σύνολο κορυφών μεγέθους a έτσι ώστε να υπάρχουν τουλάχιστον b ακμές μεταξύ τους.

(α) Δείξτε ότι αν είναι γνωστό ότι το πρόβλημα **Clique** είναι NP-πλήρες, τότε και το πρόβλημα **Independent Set** είναι NP-πλήρες.

(β) Δείξτε επίσης ότι το πρόβλημα **Dense Subgraph** είναι NP-πλήρες.

Προθεσμία υποβολής και οδηγίες. Οι απαντήσεις θα πρέπει να υποβληθούν έως τις 12/11/2020, στις 22:00, σε ηλεκτρονική μορφή, στο mycourses (φροντίστε το τελικό αρχείο να είναι μεγέθους <2MB συνολικά).

Συνιστάται *θερμά* να αφιερώσετε ικανό χρόνο για να λύσετε τις ασκήσεις μόνοι σας προτού καταφύγετε σε οποιαδήποτε *θεμιτή* βοήθεια (διαδίκτυο, βιβλιογραφία, συζήτηση με συμφοιτητές). Σε κάθε περίπτωση, οι απαντήσεις θα πρέπει να είναι *αυστηρά* ατομικές.

Για να βαθμολογηθείτε θα πρέπει να παρουσιάσετε σύντομα τις λύσεις σας (διαδικτυακά) σε ημέρα και ώρα που θα ανακοινωθεί αργότερα.

Για απορίες / διευκρινίσεις: στείλτε μήνυμα στη διεύθυνση `focs@corelab.ntua.gr`.

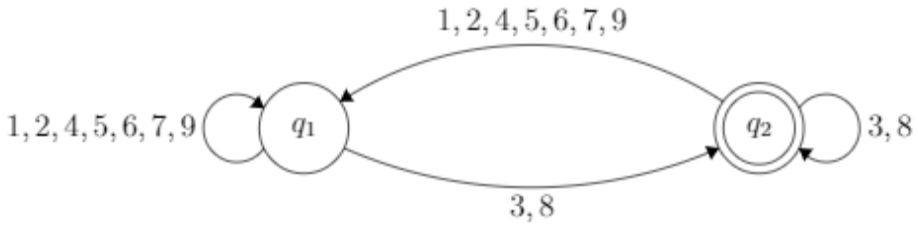
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Μάθημα 3ου εξαμήνου: Θεμελιώδη Θέματα Επιστήμης Υπολογιστών
Ονοματεπώνυμο Φοιτητή: ;
Α.Μ: 00000000

1η σειρά ασκήσεων

αυτόματα – τυπικές γλώσσες – γραμματικές – λογική – υπολογισιμότητα – πολυπλοκότητα

Άσκηση 1.

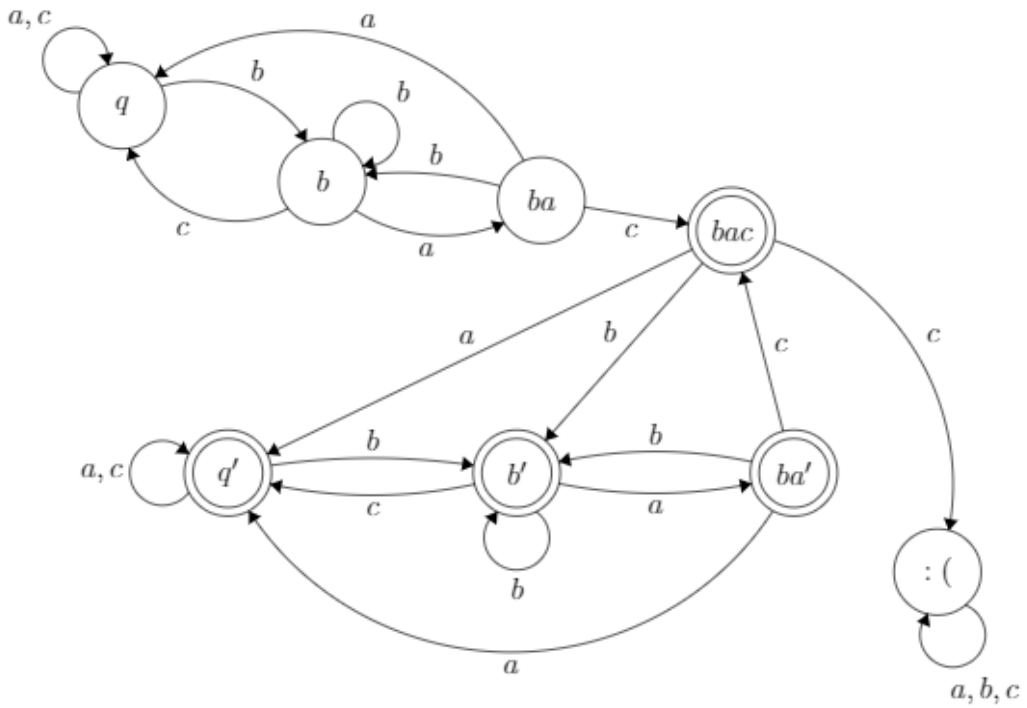
Μας ενδιαφέρει μόνο το τελευταίο ψηφίο να λήγει σε 3 ή 8. Ξεκινάμε από την q_1.



Σχήμα 1: DFA άσκησης 1

Άσκηση 2.

(i) Για την L_1: Οι καταστάσεις χάριν ευαναγνωσίας ονομάστηκαν κατά τα τελευταία σύμβολα της εισόδου που μας ενδιαφέρουν. Έχω το εξής DFA:

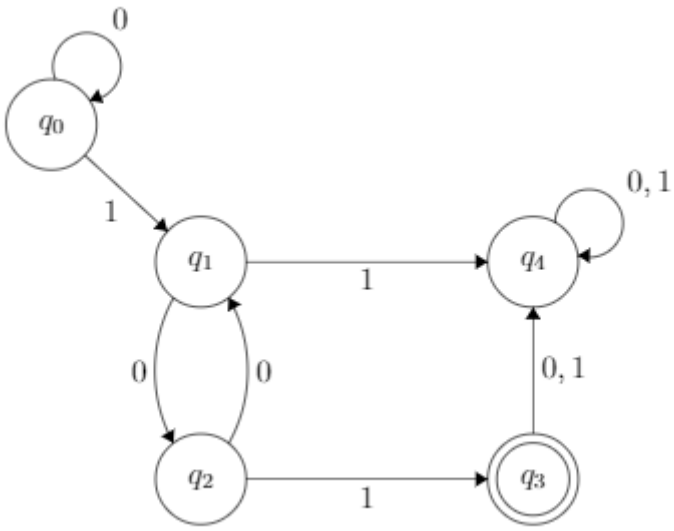


Σχήμα 2: DFA της L_1

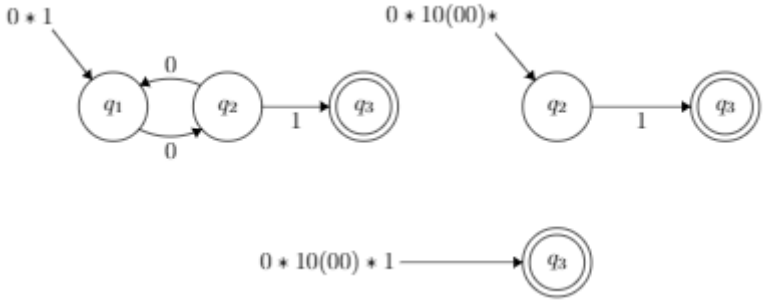
θέτω w=$(a+c+(b^{+}(ab)*c))+bb*(ab)*aa$, που αποτελεί όλες τις πιθανές "διαδρομές" που μπορώ να ακολουθήσω από την q ή την q' για να φτάσω πάλι στον εαυτό της. Οπότε η κανονική έκφραση της L_1 είναι:

$$w*bac(aw*+bb*(ab)*cw*+bb*(ab)*aaw*)*(b+ba)*(c+\varepsilon)$$

(ii) Για την L_2, έχω το εξής DFA:



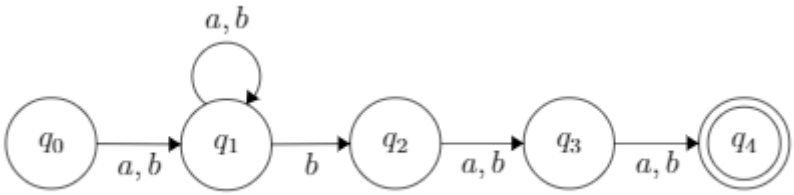
Σχήμα 3: DFA της L_2



Σχήμα 4: Εξαγωγή r.e. από DFA της L_2

Κανονική έκφραση της L_2: $0^*10(00)^*1$

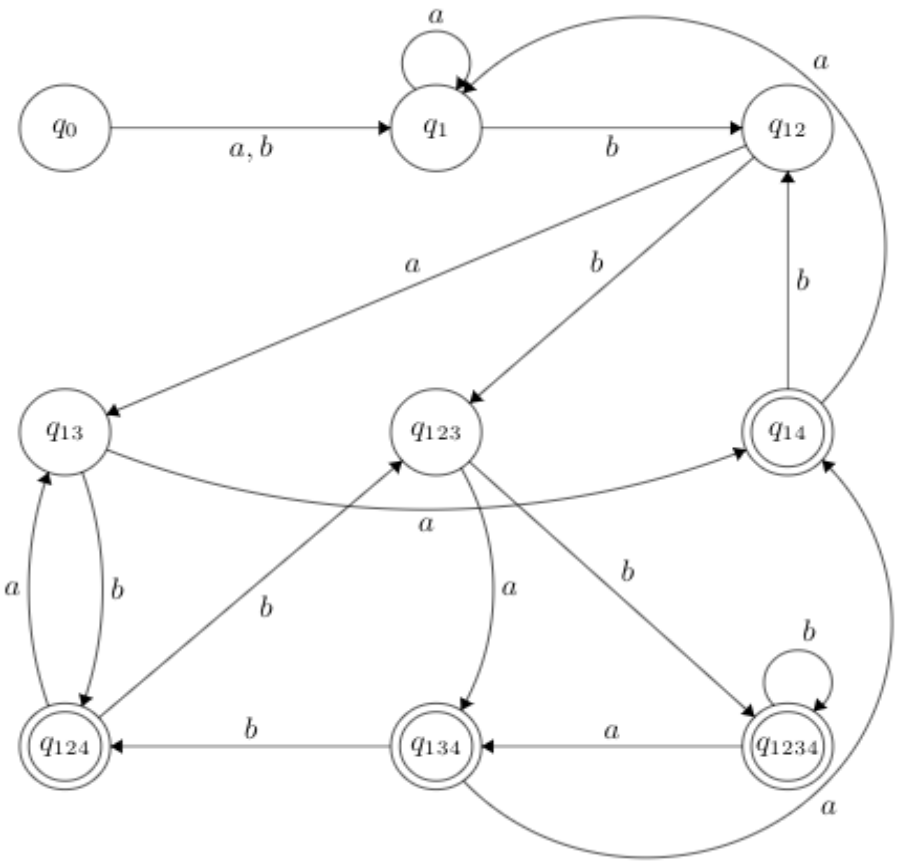
(ii) Για την L_3 έχω το εξής NFA:



Σχήμα 5: NFA της L_3

Για τη μετατροπή του NFA σε DFA εξετάζω τις καταστάσεις προσβάσιμες από την q_0 ως εξής με αυτόν τον πίνακα:

Q'/Σ	a	b
$\{q_0\}$	$\{q_1\}$	$\{q_1\}$
$\{q_1\}$	$\{q_1\}$	$\{q_1,q_2\}$
$\{q_1,q_2\}$	$\{q_1,q_3\}$	$\{q_1,q_2,q_3\}$
$\{q_1,q_3\}$	$\{q_1,q_4\}$	$\{q_1,q_2,q_4\}$
$\{q_1,q_2,q_3\}$	$\{q_1,q_3,q_4\}$	$\{q_1,q_2,q_3,q_4\}$
$\{q_1,q_4\}$	$\{q_1\}$	$\{q_1,q_2\}$
$\{q_1,q_2,q_4\}$	$\{q_1,q_3\}$	$\{q_1,q_2,q_3\}$
$\{q_1,q_3,q_4\}$	$\{q_1,q_4\}$	$\{q_1,q_2,q_4\}$
$\{q_1,q_2,q_3,q_4\}$	$\{q_1,q_3,q_4\}$	$\{q_1,q_2,q_3,q_4\}$



Σχήμα 6: DFA της L_3 (Για οικονομία χώρου οι καταστάσεις q_iq_j συμβολίζονται ως q_{ij})

Για να ελέγξω αν το DFA είναι ελάχιστο θα ελέγξω την i-διακρισιμότητα των καταστάσεων, για να δω αν υπάρχουν καταστάσεις που δεν είναι i-διακρίσιμες. Αυτό μπορώ να το κάνω με τον εξής πίνακα (Οι κόκκινες καταστάσεις είναι τερματικές):

q_1	X_3							
q_1q_2	X_2	X_2						
q_1q_3	X_1	X_1	X_1					
$q_1q_2q_3$	X_1	X_1	X_1	X_2				
q_1q_4	X_0	X_0	X_0	X_0	X_0			
$q_1q_2q_4$	X_0	X_0	X_0	X_0	X_0	X_2		
$q_1q_3q_4$	X_0	X_0	X_0	X_0	X_0	X_1	X_1	
$q_1q_2q_3q_4$	X_0	X_0	X_0	X_0	X_0	X_1	X_1	X_2
	q_0	q_1	q_1q_2	q_1q_3	$q_1q_2q_3$	q_1q_4	$q_1q_2q_4$	$q_1q_3q_4$

Μετά από 4 "γύρους" ελέγχου, παρατηρώ ότι όλες οι καταστάσεις είναι διακρίσιμες μεταξύ τους. Άρα το DFA που προαναφέρθηκε για την L_3 είναι ελάχιστο.

Άσκηση 3.

α) Θα χρησιμοποιήσω το λήμμα άντλησης (Pumping Lemma - PL) για να δείξω ότι η γλώσσα $\{0^n1^+0^m: n, m \geq 1, n \neq m\}$ δεν είναι κανονική. Σύμφωνα με αυτό, για μία κανονική γλώσσα L υπάρχει φυσικός αριθμός *n* τέτοιος ώστε για κάθε $z \in L$ με μήκος $|z| \geq n$ υπάρχει "σπάσιμο" του z σε uvw, με $|uv| \leq n$ και $|v| \geq 0$, ώστε για κάθε $i = 0,1,2,...$: $uv^iw \in L$. Δουλεύω με εις άτοπον απαγωγή: Έστω ότι η $L = \{0^n1^+0^m: n, m \geq 1, n \neq m\}$ είναι κανονική. Τότε:

~ Για κάθε φυσικό n επιλέγουμε $z = 0^n10^{n!+n} \in L$, με $|z| \geq n$.

~ Τότε αναγκαστικά : $u = 0^l$, $v = 0^k$, k, $l \leq n$, άρα θα είναι $uv^iw = 0^{n+(i-1)k}10^{n!+n}$ με $i = 0,1,2...$, οπότε πάντα για $i = n!/k + 1$ (Το οποίο θα είναι σίγουρα ακέραιος για κάθε $k \leq n$, κάτι που μάλιστα καθόρισε την επιλογή του συγκεκριμένου z) θα είναι $uv^iw \notin L$ γιατί θα ισχύει $n+(i-1)k = n!+n$. Καταλήγουμε λοιπόν σε άτοπο, άρα η L δεν είναι κανονική.

β) Θα χρησιμοποιήσω πάλι το PL. Έστω ότι η $\{0^n1^+0^m: n, m \geq 1, n = m\}$ είναι κανονική.
~ Για κάθε φυσικό n επιλέγουμε $z = 0^n10^n \in L'$, με $|z| \geq n$.
~ Τότε αναγκαστικά : $u = 0^l$, $v = 0^k$, $k, l \leq n$, άρα θα είναι $uv^iw = 0^{n+ik-k}10^n$, με $i = 0,1,2...$, οπότε γενικά $uv^iw \notin L$. Καταλήγουμε λοιπόν σε άτοπο, άρα η L δεν είναι κανονική.

γ) Η $\{0^+1^+0^+\}$ είναι κανονική, αφού αντιστοιχεί στην κανονική παράσταση $0^+1^+0^+$ (και εξάλλου αποτελεί παράθεση των κανονικών γλωσσών $\{0^+\}$, $\{1^+\}$, $\{0^+\}$, και οι κανονικές γλώσσες είναι σύνολο κλειστό ως προς την παράθεση)

δ) Η γλώσσα $\{ww : w \in \{0, 1\}^* \text{ το μήκος της } w \text{ είναι} \leq 2^{2^{\wedge}1000}\}$ είναι κανονική. Εφόσον το μήκος της w είναι πεπερασμένο, το σύνολο των λέξεων από την οποία αποτελείται αυτή η γλώσσα είναι επίσης πεπερασμένο. Επομένως, θα μπορούσαμε να θεωρήσουμε ότι η γλώσσα εκφράζεται από ένα DFA με τελικές καταστάσεις το σύνολο των αποδεκτών λέξεων, και κάποιο junk state που θα αφορά τις λέξεις με μήκος μεγαλύτερο του $2*2^{2^{\wedge}1000}$

Άσκηση 4.
α) Η γλώσσα που περιγράφει η G είναι η $\{aa(aa)^*b^*\}$ ή η $\{a^{2n}b^m, n \geq 1, m \geq 0\}$
β) $G : S \rightarrow AB \mid 0AB$, A → 1A | 101A| ε, B → 0 | ε
γ) $G : S \rightarrow AAAS \mid$ ε, A → 0A | 1

Άσκηση 5.
Για την ένωση γλωσσών χωρίς συμφραζόμενα: Έστω δύο γλώσσες L_1, L_2, οι οποίες παράγονται από τις γραμματικές $G_1=(V_1, T_1, P_1, S_1)$ και $G_2=(V_2, T_2, P_2, S_2)$. Η γραμματική G που ορίζεται από την ένωση των L_1, L_2 είναι η ακόλουθη:

$$G=(V_1 \cup V_2 \cup S, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}, S)$$

Η παραγωγή μιας λέξης με αυτή τη γραμματική θα ξεκινάει με τον κανόνα $S \rightarrow S_1 \mid S_2$, οπότε η διαδικασία που θα ακολουθήσει θα είναι αυτή της παραγωγής μιας λέξης της L_1 ή της L_2. Συνεπώς, οποιαδήποτε λέξη της $L_1 \cup L_2$ θα ανήκει είτε στην L_1, είτε στην L_2, είτε και στις δύο. Άρα η $L_1 \cup L_2$ θα αποτελεί CF γλώσσα. Άρα οι CF γλώσσες είναι γλώσσες κλειστές ως προς την πράξη ένωση.

Για την παράθεση CF γλωσσών: Έστω δύο γλώσσες L_1, L_2, οι οποίες παράγονται από τις γραμματικές $G_1=(V_1, T_1, P_1, S_1)$ και $G_2=(V_2, T_2, P_2, S_2)$. Η γραμματική G που ορίζεται από την ένωση των L_1, L_2 είναι η ακόλουθη:

$$G=(V_1 \cup V_2 \cup S, T_1 \cup T_2, P_1 \cup P_2 \cup \{S \rightarrow S_1S_2\}, S)$$

Η παραγωγή μιας λέξης με αυτή τη γραμματική θα ξεκινάει με τον κανόνα $S \rightarrow S_1S_2$, οπότε εύκολα συνάγεται ότι κάθε λέξη της L_1L_2 θα ξεκινά με μία λέξη της L_1 και θα λήγει με μια

λέξη της L_2. Αλλιώς, μπορούμε να θεωρήσουμε ότι η γλώσσα L_1L_2 εκφράζεται από τα PDA της L_1 και της L_2 συνδεδεμένα σε σειρά.

Άσκηση 6.

Έστω ότι το πρόβλημα ελέγχου του τερματισμού ενός προβλήματος με κενή συμβολοσειρά είναι επιλύσιμο, δηλαδή υπάρχει πρόγραμμα A που με είσοδο ένα πρόγραμμα Π και την κενή συμβολοσειρά ελέγχει αν το Π θα τερματίσει ή όχι, δηλαδή:

$$A\text{: } \Pi, \varepsilon \rightarrow \text{X} \in \texttt{\{halt, infinite loop\}}$$

Θεωρώ λοιπόν πρόγραμμα B που δέχεται στην εισόδο ένα πρόγραμμα Π και μία συμβολοσειρά n, και το πρόγραμμα εξόδου με είσοδο n, $\Pi^*(n)$, το κατευθύνω στην είσοδο του A, μαζί με την κενή συμβολοσειρά, ως εξής:

$$\Pi, n \xrightarrow{B} \Pi^*(n)$$

$$\Pi^*(n)\ , \varepsilon \xrightarrow{A} \text{X} \in \texttt{\{halt, infinite loop\}}$$

Έτσι, όμως, μπορούμε να αποφανθούμε αν το πρόβλημα Π με είσοδο n μπορεί να τερματίσει, κάτι που είναι άτοπο, εφόσον γνωρίζουμε ότι το Πρόβλημα Τερματισμού (Halting Problem) είναι μη επιλύσιμο. Άρα πρόβλημα ελέγχου του τερματισμού ενός προβλήματος με κενή συμβολοσειρά δεν είναι επιλύσιμο.

Άσκηση 7. (Καλύτερα συμβουλευτείτε από αλλού για αυτή)

Έστω $C_m = x_1 \vee x_2 \vee \ldots \vee x_k$, $P = C_1 \wedge C_2 \wedge \ldots \wedge C_n$. Ακολουθεί αλγόριθμος ελέγχου του αν η P ικανοποιείται, που βασίζεται στον αλγόριθμο των Davis-Putnam, με τον οποίο "κυνηγάμε" κάθε x_i στην πρόταση P (θέτοντας την πρόταση C_m που βρίσκεται σε αληθή, αν υπάρχει η x_i εκεί ή "αφαιρώντας" την $\neg x_i$ από την C_m αν τη βρούμε εκεί) ώστε να καταλήξουμε είτε σε ταυτολογία είτε σε αντίφαση. Επαναλαμβάνουμε μέχρι να βρεθεί η πρώτη περίσταση που ικανοποιεί την πρόταση ή να εξαντληθούν όλες οι περιπτώσεις, οπότε η πρόταση δεν ικανοποιείται.

```
for x_i in X = {set(C_m)}
  x_i ← TRUE
  for C_m in P
    if x_i in C_m
      C_m ← TRUE
      if P == TRUE
        for all uninitialised x_i in P x_i ← TRUE
        print Truth_Table={x_1, x_2, x_3...x_k}
        print "satisfiable"
        flag ← TRUE
    else if ¬ x_i in C_m
      x_i ← FALSE
      if C_m == FALSE
        mark this current state of (P, x_i) as unavailable
        restore P, x_i to previous available state and continue
```

repeat this loop for the next available literal
if not flag
print “unsatisfiable”

Άσκηση 8.

Έστω ότι το πρόβλημα Independent Set δεν είναι NP-πλήρες. Θεωρώ ένα γράφημα $G = (V, E)$, το οποίο έχει μέγιστο ανεξάρτητο σύνολο μεγέθους *m*. Επειδή κανένας κόμβος του ανεξάρτητου συνόλου δεν είναι συνδεδεμένος με ακμή με κάποιον άλλον σε αυτό το σύνολο, αν θεωρήσουμε το αντίστροφο γράφημα G', τότε όλοι αυτοί οι κόμβοι θα είναι συνδεδεμένοι μεταξύ τους, οπότε θα αποτελούν μια κλίκα, και μάλιστα τη μεγαλύτερου μεγέθους κλίκα ίδιου μεγέθους *m* με το μέγιστο ανεξάρτητο σύνολο του G'. Έτσι λοιπόν μπορούμε, με τη βοήθεια του αντίστροφου γραφήματος, να ελέγξουμε το μέγεθος του μέγιστου ανεξάρτητου συνόλου και της μέγιστης κλίκας σε παρόμοιο χρόνο. Αυτό όμως είναι άτοπο, εφόσον υποθέσαμε ότι το πρόβλημα Independent Set δεν είναι NP-πλήρες, και γνωρίζουμε ότι το πρόβλημα Clique είναι NP-πλήρες. Άρα το Independent Set είναι NP-πλήρες.

Έστω ότι το Dense Subgraph δεν είναι NP-πλήρες. Τότε για είσοδο (a, a*(a-1)/2) θα μπορούμε να βρούμε αν το γράφημα G περιέχει κλίκα με a κόμβους, εφόσον μία κλίκα a κόμβων έχει πάντα $a+a\text{-}1+...+2+1 = a(a\text{-}1)/2$ ακμές. Άρα το πρόβλημα Clique λύνεται σε παρόμοιο χρόνο, κάτι το οποίο όμως είναι άτοπο, γιατί ήδη ξέρουμε ότι είναι NP-πλήρες. Άρα και το Dense Subgraph είναι NP-πλήρες.

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Θεμελιώδη Θέματα Επιστήμης Υπολογιστών, 2020-21

2η σειρά γραπτών ασκήσεων

(αλγοριθμικές τεχνικές – αριθμητικοί αλγόριθμοι
αλγόριθμοι γράφων – δυναμικός προγραμματισμός)

Άσκηση 1. (Αναδρομή – Επανάληψη – Επαγωγή)

(α) Εκφράστε τον αριθμό κινήσεων δίσκων που κάνει ο αναδρομικός αλγόριθμος για τους πύργους του Hanoi, σαν συνάρτηση του αριθμού των δίσκων n.

(β) Δείξτε ότι ο αριθμός κινήσεων του αναδρομικού ισούται με τον αριθμό μετακινήσεων του επαναληπτικού αλγορίθμου.

(γ) Δείξτε ότι ο αριθμός των κινήσεων των παραπάνω αλγορίθμων είναι ο ελάχιστος μεταξύ όλων των δυνατών αλγορίθμων για το πρόβλημα αυτό.

(δ*) Θεωρήστε το πρόβλημα των πύργων του Hanoi με 4 αντί για 3 πασσάλους. Σχεδιάστε αλγόριθμο μετακίνησης n δίσκων από τον πάσσαλο 1 στον πάσσαλο 4 ώστε το πλήθος των βημάτων να είναι σημαντικά μικρότερο από το πλήθος των βημάτων που απαιτούνται όταν υπάρχουν μόνο 3 πάσσαλοι. Εκφράστε τον αριθμό των απαιτούμενων βημάτων σαν συνάρτηση του n.

Άσκηση 2. (Επαναλαμβανόμενος Τετραγωνισμός – Κρυπτογραφία)

(α) Γράψτε πρόγραμμα σε γλώσσα της επιλογής σας (θα πρέπει να υποστηρίζει πράξεις με αριθμούς 100δων ψηφίων) που να ελέγχει αν ένας αριθμός είναι πρώτος με τον έλεγχο (test) του Fermat:

Αν n πρώτος τότε για κάθε a τ.ώ. $1 < a < n-1$, ισχύει

$$a^{n-1} \bmod n = 1$$

Αν λοιπόν, δεδομένου ενός n βρεθεί a ώστε να μην ισχύει η παραπάνω ισότητα τότε ο αριθμός n είναι οπωσδήποτε σύνθετος. Αν η ισότητα ισχύει, τότε το n είναι πρώτος με αρκετά μεγάλη πιθανότητα (για τους περισσότερους αριθμούς $\geq 1/2$). Για να αυξήσουμε σημαντικά την πιθανότητα μπορούμε να επαναλάβουμε μερικές φορές (τυπικά 30 φορές) με διαφορετικό a. Αν όλες τις φορές βρεθεί να ισχύει η παραπάνω ισότητα τότε λέμε ότι το n "περνάει το test" και ανακηρύσσουμε το n πρώτο αριθμό·αν έστω και μία φορά αποτύχει ο έλεγχος, τότε ο αριθμός είναι σύνθετος.

Η συνάρτησή σας θα πρέπει να δουλεύει σωστά για αριθμούς χιλιάδων ψηφίων. Δοκιμάστε την με τους αριθμούς:

$67280421310721, 170141183460469231731687303715884105721, 2^{2281}-1, 2^{9941}-1, 2^{19939}-1$

Σημείωση 1: το $a^{2^{19939}-2}$ έχει 'αστρονομικά' μεγάλο πλήθος ψηφίων (δεν χωράει να γραφτεί σε ολόκληρο το σύμπαν!), ενώ το $a^{2^{19939}-2} \bmod (2^{19939}-1)$ είναι σχετικά "μικρό" (έχει μερικές χιλιάδες δεκαδικά ψηφία *μόνο* :-)).

Σημείωση 2: Υπάρχουν (λίγοι) σύνθετοι που έχουν την ιδιότητα να περνούν τον έλεγχο Fermat για κάθε a που είναι σχετικά πρώτο με το n, οπότε για αυτούς το test θα αποτύχει όσες δοκιμές και αν γίνουν (εκτός αν πετύχουμε κατά τύχη a που δεν είναι σχετικά πρώτο με το n, πράγμα αρκετά απίθανο). Αυτοί οι αριθμοί λέγονται *Carmichael* – δείτε και `http://en.wikipedia.org/wiki/Carmichael_`

number. Ελέγξτε τη συνάρτησή σας με *αρκετά μεγάλους* αριθμούς Carmichael που θα βρείτε π.χ. στη σελίδα http://de.wikibooks.org/wiki/Pseudoprimzahlen:_Tabelle_Carmichael-Zahlen. Τι παρατηρείτε;

(β) Μελετήστε και υλοποιήστε τον έλεγχο Miller-Rabin που αποτελεί βελτίωση του ελέγχου του Fermat και δίνει σωστή απάντηση με πιθανότητα τουλάχιστον 1/2 για *κάθε* φυσικό αριθμό (οπότε με 30 επαναλήψεις έχουμε αμελητέα πιθανότητα λάθους για κάθε αριθμό εισόδου). Δοκιμάστε τον με διάφορους αριθμούς Carmichael. Τι παρατηρείτε;

(γ*) Γράψτε πρόγραμμα που να βρίσκει όλους τους πρώτους αριθμούς Mersenne, δηλαδή της μορφής $n = 2^x - 1$ με $100 < x < 1000$ (σημειώστε ότι αν το x δεν είναι πρώτος, ούτε το $2^x - 1$ είναι πρώτος – μπορείτε να το αποδείξετε;). Αντιπαραβάλετε με όσα αναφέρονται στην ιστοσελίδα https://www.mersenne.org/primes/.

Άσκηση 3. (Αριθμοί Fibonacci)

(α) Υλοποιήστε και συγκρίνετε τους εξής αλγορίθμους για υπολογισμό του n-οστού αριθμού Fibonacci: αναδρομικό με memoization, επαναληπτικό, και με πίνακα.

Υλοποιήστε τους αλγορίθμους σε γλώσσα που να υποστηρίζει πολύ μεγάλους ακεραίους (100δων ψηφίων), π.χ. σε Python. Χρησιμοποιήστε τον πολλαπλασιασμό ακεραίων που παρέχει η γλώσσα. Τι συμπεραίνετε;

(β*) Δοκιμάστε να λύσετε το παραπάνω πρόβλημα με ύψωση σε δύναμη, χρησιμοποιώντας τη σχέση του F_n με το ϕ (χρυσή τομή). Τι παρατηρείτε;

(γ) Υπολογίστε και συγκρίνετε την πολυπλοκότητα ψηφιοπράξεων (bit complexity) του επαναληπτικού αλγορίθμου για υπολογισμό αριθμών Fibonacci και του αλγορίθμου που χρησιμοποιεί πίνακα. Για τον αλγόριθμο με πίνακα θεωρήστε (α) απλό πολλαπλασιασμό ακεραίων και (β) πολλαπλασιασμό Gauss-Karatsuba. Τι παρατηρείτε σε σχέση και με το ερώτημα (α);

(δ) Υλοποιήστε συνάρτηση που να δέχεται σαν είσοδο δύο θετικούς ακεραίους n, k και να υπολογίζει τα k λιγότερο σημαντικά ψηφία του n-οστού αριθμού Fibonacci.

(ε*) Αναζητήστε και εξετάστε τη μέθοδο Fast Doubling σε σχέση με τα παραπάνω ερωτήματα. Συγκρίνετέ την με τη μέθοδο του πίνακα.

Άσκηση 4. (Λιγότερα Διόδια)

Θεωρήστε το εξής πρόβλημα σε οδικά δίκτυα: κάθε κόμβος έχει διόδια (μια θετική ακέραια τιμή) και θέλουμε να βρεθούν οι διαδρομές με το ελάχιστο κόστος έναν αρχικό κόμβο s προς κάθε άλλο κόμβο. Θεωρήστε ότι στον αρχικό κόμβο δεν πληρώνουμε διόδια (ενώ στον τελικό πληρώνουμε, όπως και σε κάθε ενδιάμεσο) και ότι το δίκτυο παριστάνεται σαν κατευθυνόμενος γράφος. Περιγράψτε όσο το δυνατόν πιο αποδοτικούς αλγόριθμους για το πρόβλημα αυτό στις εξής περιπτώσεις:

(α) Το δίκτυο δεν έχει κύκλους (κατευθυνόμενους).

(β) Το δίκτυο μπορεί να έχει κύκλους.

(γ) Σε κάποιους κόμβους δίνονται προσφορές που μπορεί να είναι μεγαλύτερες από το κόστος διέλευσης (αλλά δεν υπάρχει κύκλος όπου συνολικά οι προσφορές να υπερβαίνουν το κόστος).

Άσκηση 5. (Επιβεβαίωση Αποστάσεων - Δέντρο Συντομότερων Μονοπατιών)

Θεωρούμε ένα κατευθυνόμενο γράφημα $G(V, E, \ell)$ με n κορυφές, m ακμές και (ενδεχομένως αρνητικά) μήκος $\ell(e)$ σε κάθε ακμή του $e \in E$. Συμβολίζουμε με $d(u, v)$ την απόσταση των κορυφών u

και v (δηλ. το $d(u, v)$ είναι ίσο με το μήκος της συντομότερης $u - v$ διαδρομής) στο G. Δίνονται n αριθμοί $\delta_1, \ldots, \delta_n$, όπου κάθε δ_k (υποτίθεται ότι) ισούται με την απόσταση $d(v_1, v_k)$ στο G. Να διατυπώσετε αλγόριθμο που σε χρόνο $\Theta(n + m)$, δηλ. γραμμικό στο μέγεθος του γραφήματος, ελέγχει αν τα $\delta_1, \ldots, \delta_n$ πράγματι ανταποκρίνονται στις αποστάσεις των κορυφών από την v_1, δηλαδή αν για κάθε $v_k \in V$, ισχύει ότι $\delta_k = d(v_1, v_k)$. Αν αυτό αληθεύει, ο αλγόριθμός σας πρέπει να υπολογίζει και να επιστρέφει ένα Δέντρο Συντομότερων Μονοπατιών με ρίζα τη v_1 (χωρίς ο χρόνος εκτέλεσης να ξεπεράσει το $\Theta(n + m)$).

Άσκηση 6. (Προγραμματισμός Διακοπών)

Μπορούμε να αναπαραστήσουμε το οδικό δίκτυο μιας χώρας ως ένα συνεκτικό μη κατευθυνόμενο γράφημα $G(V, E, \ell)$ με n κορυφές και m ακμές. Κάθε πόλη αντιστοιχεί σε μια κορυφή του γραφήματος και κάθε οδική αρτηρία σε μία ακμή. Κάθε οδική αρτηρία $e \in E$ συνδέει δύο πόλεις και έχει μήκος $\ell(e)$ χιλιόμετρα. Η ιδιαιτερότητα της συγκεκριμένης χώρας είναι ότι έχουμε σταθμούς ανεφοδιασμού σε καύσιμα μόνο στις πόλεις / κορυφές του γραφήματος, όχι στις οδικές αρτηρίες / ακμές.

Θέλουμε να ταξιδέψουμε από την πρωτεύουσα s σε ένα ορεινό θέρετρο t για διακοπές, για να αλλάξουμε παραστάσεις, μετά την άρση του lockdown. Θα χρησιμοποιήσουμε το αυτοκίνητό μας που διαθέτει αυτονομία καυσίμου για L χιλιόμετρα.

(α) Να διατυπώσετε έναν αλγόριθμο, με όσο το δυνατόν μικρότερη χρονική πολυπλοκότητα, που υπολογίζει αν κάτι τέτοιο είναι εφικτό. Ποια είναι η χρονική πολυπλοκότητα του αλγορίθμου σας στη χειρότερη περίπτωση;

(β) Να διατυπώσετε αλγόριθμο με χρονική πολυπλοκότητα $(m \log m)$ που υπολογίζει την ελάχιστη αυτονομία καυσίμου (σε χιλιόμετρα) που απαιτείται για το ταξίδι από την πόλη s στην πόλη t.

(γ*) Να διατυπώσετε αλγόριθμο με *γραμμική* χρονική πολυπλοκότητα που υπολογίζει την ελάχιστη αυτονομία καυσίμου για το ταξίδι από την πόλη s στην πόλη t. *Υπόδειξη:* Εδώ μπορεί να σας φανεί χρήσιμο ότι (με λογική αντίστοιχη με αυτή της Quicksort) μπορούμε να υπολογίσουμε τον median ενός μη ταξινομημένου πίνακα σε γραμμικό χρόνο.

Άσκηση 7. (Αγορά Εισιτηρίων)

Εκτός από τις διακοπές σας, έχετε προγραμματίσει προσεκτικά και την παρουσία σας στη Σχολή, μετά την άρση του lockdown. Συγκεκριμένα, έχετε σημειώσει ποιες από τις T ημέρες, που θα ακολουθήσουν την άρση του lockdown, θα έρθετε στο ΕΜΠ για να παρακολουθήσετε μαθήματα και εργαστήρια και να δώσετε εξετάσεις. Το πρόγραμμά σας έχει τη μορφή ενός πίνακα S με T θέσεις, όπου για κάθε ημέρα $t = 1, \ldots, T$, $S[t] = 1$, αν θα έρθετε στο ΕΜΠ, και $S[t] = 0$, διαφορετικά. Το πρόγραμμά σας δεν έχει κανονικότητα και επηρεάζεται από διάφορες υποχρεώσεις και γεγονότα.

Για κάθε μέρα που θα έρθετε στο ΕΜΠ, πρέπει να αγοράσετε εισιτήριο που να επιτρέπει τη μετακίνησή σας με τις αστικές συγκοινωνίες. Υπάρχουν συνολικά k διαφορετικοί τύποι εισιτηρίων (π.χ., ημερήσιο, τριών ημερών, εβδομαδιαίο, μηναίο, εξαμηνιαίο, ετήσιο). Ο τύπος εισιτηρίου i σας επιτρέπει να μετακινηθείτε για c_i διαδοχικές ημέρες (μπορεί βέβαια κάποιες από αυτές να μην χρειάζεται να έρθετε στο ΕΜΠ) και κοστίζει p_i ευρώ. Για τους τύπους των εισιτηρίων, ισχύει ότι $1 = c_1 < c_2 < \cdots < c_k \leq T$, $p_1 < p_2 < \cdots < p_k$, και $p_1/c_1 > p_2/c_2 > \cdots > p_k/c_k$ (δηλαδή, η τιμή αυξάνεται με τη διάρκεια του εισιτηρίου, αλλά η τιμή ανά ημέρα μειώνεται).

Να διατυπώσετε αλγόριθμο που υπολογίζει τον συνδυασμό τύπων εισιτηρίων με ελάχιστο συνολικό κόστος που καλύπτουν όλες τις ημέρες που θα έρθετε στο ΕΜΠ. Ποια είναι η χρονική πολυπλοκότητα του αλγορίθμου σας στη χειρότερη περίπτωση;

Άσκηση 8. (Αντιπροσωπεία Φορητών Υπολογιστών)

Επιθυμούμε να βελτιστοποιήσουμε την λειτουργία μιας νέας αντιπροσωπείας φορητών υπολογιστών για τις επόμενες n ημέρες. Για κάθε ημέρα i, $1 \leq i \leq n$, υπάρχει μια (απόλυτα ασφαλής) πρόβλεψη d_i του αριθμού των πωλήσεων. Όλες οι πωλήσεις λαμβάνουν χώρα το μεσημέρι, και όσοι φορητοί υπολογιστές δεν πωληθούν, αποθηκεύονται. Υπάρχει δυνατότητα αποθήκευσης μέχρι S υπολογιστών, και το κόστος είναι C για κάθε υπολογιστή που αποθηκεύεται και για κάθε ημέρα αποθήκευσης. Το κόστος μεταφοράς για την προμήθεια νέων υπολογιστών είναι K ευρώ, ανεξάρτητα από το πλήθος των υπολογιστών που προμηθευόμαστε, και οι νέοι υπολογιστές φθάνουν λίγο πριν το μεσημέρι (άρα αν πωληθούν αυθημερόν, δεν χρειάζονται αποθήκευση). Αρχικά δεν υπάρχουν καθόλου υπολογιστές στην αντιπροσωπεία.

Το ζητούμενο είναι να προσδιορισθούν οι παραγγελίες (δηλ. πόσους υπολογιστές θα παραγγείλουμε και πότε) ώστε να ικανοποιηθούν οι προβλεπόμενες πωλήσεις με το ελάχιστο δυνατό συνολικό κόστος (αποθήκευσης και μεταφοράς). Να διατυπώσετε αλγόριθμο δυναμικού προγραμματισμού με χρόνο εκτέλεσης πολυωνυμικό στο nS για τη βελτιστοποίηση της λειτουργίας της αντιπροσωπείας.

Προθεσμία υποβολής και οδηγίες. Οι απαντήσεις θα πρέπει να υποβληθούν έως τις 03/12/2020, και ώρα 22:00, σε ηλεκτρονική μορφή, στο mycourses (προσπαθήστε το τελικό αρχείο να είναι μεγέθους <2MB συνολικά).

Τα ερωτήματα με (*) είναι προαιρετικά. Εφ'όσον τα επιλύσετε μπορούν να προσμετρηθούν στη θέση ερωτημάτων που δεν απαντήσατε.

Συνιστάται *θερμά* να αφιερώσετε ικανό χρόνο για να λύσετε τις ασκήσεις μόνοι σας προτού καταφύγετε σε οποιαδήποτε *θεμιτή* βοήθεια (διαδίκτυο, βιβλιογραφία, συζήτηση με συμφοιτητές). Σε κάθε περίπτωση, οι απαντήσεις θα πρέπει να είναι *αυστηρά* ατομικές και να περιλαμβάνουν αναφορές σε κάθε πηγή που χρησιμοποιήσατε.

Για να βαθμολογηθείτε θα πρέπει να παρουσιάσετε σύντομα τις λύσεις σας σε ημέρα και ώρα που θα ανακοινωθεί αργότερα.

Για απορίες / διευκρινίσεις: στείλτε μήνυμα στη διεύθυνση `focs@corelab.ntua.gr`.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Μάθημα 3ου εξαμήνου: Θεμελιώδη Θεμέλια Θεμελιωδών Θεμελίων
Ονοματεπώνυμο Φοιτητή: Ονοματεπώνυμο Φοιτητή
Α.Μ:

2η σειρά γραπτών ασκήσεων

(Αλγοριθμικές τεχνικές - αριθμητικοί αλγόριθμοι
αλγόριθμοι γράφων - δυναμικός προγραμματισμός)

Άσκηση 1

(α) Ο αναδρομικός αλγόριθμος για τους πύργους του Hanoi είναι ο εξής:

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

Παρατηρούμε ότι σε κάθε αναδρομική κλήση της move_hanoi() με $n \neq 1$ καλείται η ίδια συνάρτηση άλλες δύο φορές και γίνεται μία κίνηση, κάτι που θα γίνεται για τις τιμές του n από n έως 1. Συνολικά λοιπόν οι κινήσεις θα είναι:

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

(β) Ο επαναληπτικός αλγόριθμος για τους πύργους του Hanoi είναι ο εξής:

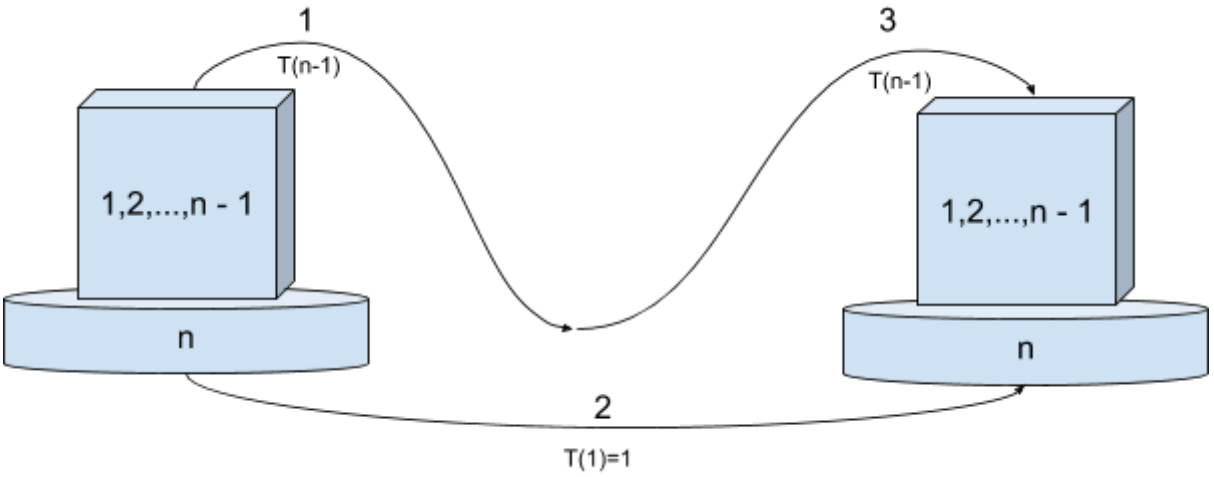
Επανάλαβε (μέχρι να επιτευχθεί η μετακίνηση):

- *Μετακίνησε κατά τη θετική φορά τον μικρότερο δίσκο*
- *Κάνε την μοναδική επιτρεπτή κίνηση που δεν αφορά τον μικρότερο δίσκο*

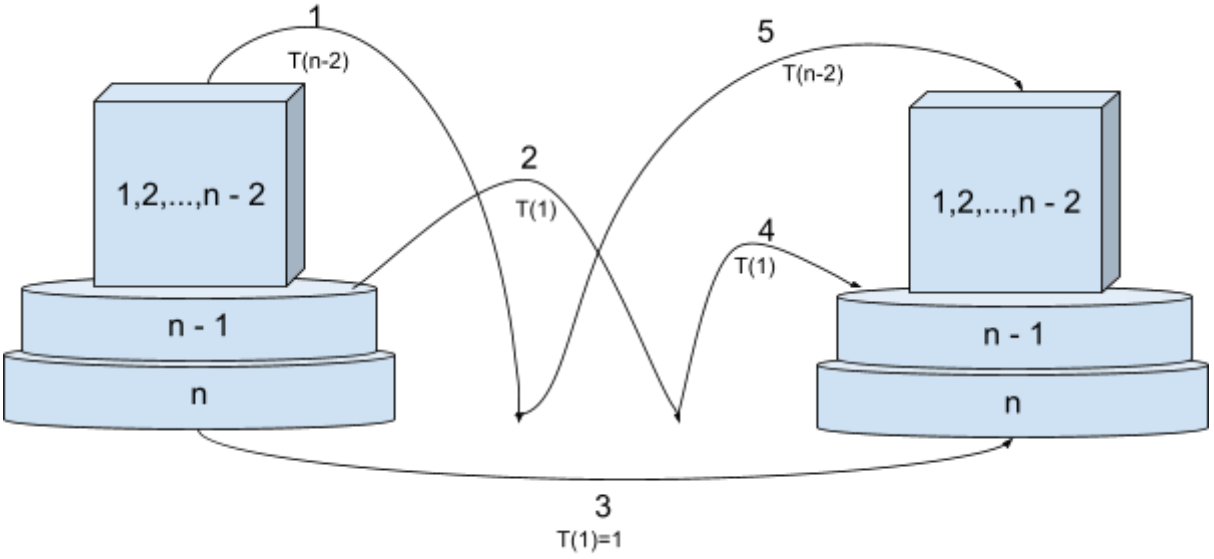
Έστω ότι πράγματι οι απαιτούμενες κινήσεις για αυτόν είναι $2^n - 1$. Αυτό εύκολα επαληθεύεται για $n = 1$. Για $n + 1$ δίσκους θα δείξω ότι χρειάζονται $2^{n+1} - 1$ κινήσεις. Παρατηρώ ότι σε κάθε βήμα του αλγορίθμου ο μόνος τρόπος να μετακινηθεί ο δίσκος στην κορυφή του αρχικού πύργου είναι αν υπάρχει ένας άδειος στύλος, που σημαίνει ότι οι υπόλοιποι δίσκοι βρίσκονται διατεταγμένοι, οπότε ουσιαστικά αποτελούν έναν 'υποπύργο' μετακινημένο σε άλλο στύλο:

Έτσι, για να μετακινήσουμε τον δίσκο $n + 1$ θα χρειαστεί πρώτα όλοι οι υπόλοιποι δίσκοι να έχουν μετακινηθεί και να έχουν σχηματίσει έναν πύργο ύψους n, και μάλιστα στον στύλο Y αν ονομάσουμε X, Y, Z τους τρεις στύλους. Όμως αυτό σύμφωνα με την υπόθεση απαιτεί $2^n - 1$ κινήσεις. Μετά από αυτό απαιτείται η μετακίνηση του δίσκου $n + 1$ στον στύλο Z, (1 κίνηση) και η μετακίνηση του 'υποπύργου' ύψους n στον στύλο Z πάνω στον μεγαλύτερο

δίσκο, που απαιτεί πάλι $2^n - 1$ κινήσεις. Δηλαδή συνολικά απαιτούνται $2^n - 1 + 1 + 2^n - 1 = 2(2^n) - 1 = 2^{n+1} - 1$ κινήσεις. Οπότε με μαθηματική επαγωγή αποδείχθηκε ότι με τον επαναληπτικό αλγόριθμο εκτελούνται $2^{n+1} - 1$ κινήσεις.
(γ) Σε κάθε περίπτωση, είναι απαραίτητο να μετακινηθούν οι $n - 1$ δίσκοι στον βοηθητικό στύλο, έπειτα ο δίσκος n στον τελικό στύλο, και έπειτα οι $n - 1$ δίσκοι στον τελικό στύλο, πάνω στον δίσκο n. Αυτή η διαδικασία απαιτεί $T(n) = 2T(n-1) + 1 = 2^n - 1$ κινήσεις. Εφόσον αυτές οι κινήσεις αποτελούν μέρος εκτέλεσης μιας απαραίτητης διαδικασίας -ο δίσκος n είναι αδύνατο να μετακινηθεί χωρίς να μετακινηθούν όλοι οι υπόλοιποι-, αυτό το πλήθος των κινήσεων είναι ο ελάχιστος μεταξύ όλων των δυνατών αλγορίθμων.



(δ*) Στο πρόβλημα των πύργων Hanoi με 4 στύλους, θα ακολουθήσω την ίδια λογική με πριν, μόνο που τώρα θα μπορώ να χρησιμοποιήσω έναν επιπλέον βοηθητικό στύλο, οπότε θα μπορώ να μετακινώ δύο δίσκους τη φορά, καταλήγοντας στους n - 1 και n, όπως φαίνεται και στο παρακάτω σχήμα, που περιγράφει τη διαδικασία που θα ακολουθήσω.



Μπορώ λοιπόν με βάση αυτό να γράψω τον εξής αλγόριθμο:

```
procedure move_hanoi(n from X to Y using Z, W)
  begin
     if n = 1 then
       move top disk from X to Y
     if n = 0 then
       return
     else
       move_hanoi(n-2 from X to Z using Y, W);
       move top disk from X to W;
       move top disk from X to Y;
       move top disk from W to Y;
       move_hanoi(n-2 from Z to Y using X, W);
  end
```

Τόσο από το σχήμα, όσο και από τον αλγόριθμο, μπορούμε να συμπεράνουμε ότι θα γίνουν $n/2$ αναδρομικές κλήσεις, οπότε οι συνολικές κινήσεις θα είναι $T(n) = 2T(n-2) + 3 = 3 \cdot 2^{\lfloor n/2-1 \rfloor} + 2^{\lfloor n/2-1 \rfloor} \cdot (n \bmod 2) = O(2^{n/2})$. Αν και αυτός δεν είναι ο βέλτιστος αλγόριθμος (υπάρχει π.χ ο αλγόριθμος των Frame-Stewart σε πολυπλοκότητα $O(2^{\sqrt{n}})$, για 4 στύλους, που γενικεύεται και για k στύλους, με την πολυπλοκότητα να μειώνεται περαιτέρω με την αύξηση των στύλων: https://en.wikipedia.org/wiki/Tower_of_Hanoi#Frame%E2%80%93Stewart_algorithm), είναι φανερό ότι τα βήματα που απαιτούνται είναι αισθητά λιγότερα από αυτά στην περίπτωση των τριών μόνο στύλων.

Άσκηση 2

(α) Γράφτηκε το εξής πρόγραμμα σε Python, το οποίο εκτελεί τον "μικρό" έλεγχο του Fermat με 30 τυχαίους αριθμούς μικρότερους του 1000, που δουλεύει ικανοποιητικά και για τα δοσμένα παραδείγματα (με μικρές τροποποιήσεις για να δοθούν αριθμοί της μορφής $2^x - 1$ στην είσοδο):

```
import random

def fermat(n):
  n = int(n)
  for i in range(1, 30):
    a = random.randint(2, n - 2) % 1000
    y = pow(a, n - 1, n)
    if y % n != 1:
      return False
  return True

n = int(input("Give a number: "))
if fermat(n):
```

```
  print("Probably prime!")
else:
  print("Composite!")
```

Το πρόγραμμα χρησιμοποιεί την εντολή `pow(a, n - 1, n)` για τη χρήση του modular exponentiation (για να μειωθεί δραματικά το μέγεθος των αριθμών). Υλοποιείται όπως η κάτωθι συνάρτηση:

```
def fastmodpower(a,n,p):
  res=1
  while (n>0):
    if (n%2==1):
      res=res*a % p
    print (n, a, res)
    n=n//2
    a=a*a % p
  return res
```

Σημείωση: Κάποιοι αριθμοί Carmichael που δοκιμάστηκαν βγάζουν θετική έξοδο, ενώ είναι σύνθετοι, π.χ:

```
Give a number: 1348964401
Probably prime!
```

Όμως: 1.348.964.401 = 401 * 1201 * 2801

(γ) Γράφτηκε η παρακάτω συνάρτηση σε Python για τον έλεγχο Miller-Robin:

```
import random

def FMR(n):
 n = int(n)
 for i in range(1, 30):
   a = random.randint(2, n - 2)
   x = pow(a, n - 1, n)
   if x % n != 1:
     return False
   else:
     t = n - 1
     s = 0
     while t % 2 == 0:
       t //= 2
       s += 1
     y = pow(a, t, n)
     z = pow(y, 2, n)
     while y % n != 1 % n and y % n != -1 % n and z % n != 1 % n and z % n
!= -1 % n:
       y = z
       z = pow(y, 2, n)
```

```
      if  y % n == -1 % n and z % n == 1 % n:
        return False
 return True
```

Τώρα η συνάρτηση βγάζει αρνητικό αποτέλεσμα για τους αριθμούς Carmichael που δοκιμάστηκαν, που είναι αναμενόμενο εφόσον αυτή αποτελεί βελτίωση του ελέγχου του Fermat.
(γ*) Γράφτηκε το εξής πρόγραμμα σε Python, που χρησιμοποιεί τη συνάρτηση του προηγούμενου ερωτήματος:

```
primes = []
for i in range(100, 1000):
  if FMR(i):
    primes.append(i)

for prime in primes:
  q = 2**prime - 1
  if FMR(q):
    print("2^" + str(prime) + " -1 is prime!!")
```

οπότε η έξοδος είναι η εξής:

```
2^107 -1 is prime!!
2^127 -1 is prime!!
2^521 -1 is prime!!
2^607 -1 is prime!!
```

Το αποτέλεσμα λοιπόν βρίσκεται σε πλήρη συμφωνία με όσα λέγονται στη σελίδα https://www.mersenne.org/primes/. Το πρόγραμμα ελέγχει αν το $2^x - 1$ είναι πρώτος μόνο για τους πρώτους αριθμούς x, εφόσον αν αυτός είναι σύνθετος, δηλ. $x = pq$, τότε $2^x - 1 = (2^p)^q - 1 = (2^p - 1)(1 + 2^p + 2^{2p} + \ldots + 2^{p(q-1)})$, που είναι σύνθετος εφόσον $2^p - 1 \neq 1$.

Άσκηση 3
(α, β*)Γράφτηκε το εξής πρόγραμμα σε Python για τις διάφορες μεθόδους υπολογισμού των αριθμών Fibonacci, μαζί και για τον χρόνο εκτέλεσής τους:

```
import time
import sys

sys.setrecursionlimit(10**6)

def fib_memoization(n, dictionary):
  if n in dictionary.keys():
      return dictionary[n]
  else:
```

```
      dictionary[n] = fib_memoization(n - 1, dictionary) +
fib_memoization(n - 2, dictionary)
      return dictionary[n]

def fib_iterative(n):
    if n == 0:
        return 0
    a = 0
    b = 1
    for i in range(2, n + 1):
        c = b
        b = a + b
        a = c
    return b

def matrix_pow(A, n):
    #for 2x2 matrices
    R = [[1, 0], [0, 1]]
    while n > 0:
        if n % 2 == 1:
            a = R[0][0] * A[0][0] + R[0][1] * A[1][0]
            b = R[0][0] * A[0][1] + R[0][1] * A[1][1]
            c = R[1][0] * A[0][0] + R[1][1] * A[1][0]
            d = R[1][0] * A[0][1] + R[1][1] * A[1][1]
            R = [[a, b], [c, d]]
        n //= 2
        a = A[0][0] * A[0][0] + A[0][1] * A[1][0]
        b = A[0][0] * A[0][1] + A[0][1] * A[1][1]
        c = A[1][0] * A[0][0] + A[1][1] * A[1][0]
        d = A[1][0] * A[0][1] + A[1][1] * A[1][1]
        A[0][0] = a
        A[0][1] = b
        A[1][0] = c
        A[1][1] = d
    return R


def fib_matrix(n):
    if n == 0:
        return 0
    F = [[1, 1], [1, 0]]
    M = matrix_pow(F, n - 2)
```

```
    return M[0][0] + M[0][1]

def fib_GoldenRatio(n):
    phi = (1 + 5**0.5)/2
    F = (phi**n - (1 - phi)**n)/(5**0.5)
    return F

d = {0 : 0, 1 : 1}
n = 0
while n != -1:
    time_report = ""
    n = int(input("Give the number of the nth term of the
Fibonacci Sequence you want to find (input -1 to exit): "))
    if n < 0:
        break
    time_a = time.time()
    print("Calculation with iterative method:
{}".format(fib_iterative(n)))
    time_b = time.time()
    time_report += "Calculation time with iterative method was {}
seconds\n".format(time_b - time_a)
    print("Calculation with matrix method:
{}".format(fib_matrix(n)))
    time_c = time.time()
    time_report += "Calculation time with matrix method was {}
seconds\n".format(time_c - time_b)
    try:
        print("Calculation with memoization method:
{}".format(fib_memoization(n, d)))
        time_d = time.time()
        time_report += "Calculation time with memoization method
was {} seconds\n".format(time_d - time_c)
    except:
        print("Number too large to use recursive method")
    try:
        time_e = time.time()
        print("Approximation with Golden Ratio relationship:
{}".format(fib_GoldenRatio(n)))
        time_f = time.time()
        time_report += "Calculation time with Golden Ratio method
was {} seconds\n".format(time_f - time_e)
    except:
```

```
        print("Golden Ratio Method resulted in overflow")

    print(time_report)
```

```
Give the number of the nth term of the Fibonacci Sequence you want
to find (input -1 to exit): 1000
Calculation with iterative method:
43466557686937456435688527675040625802564660517371780402481729089536555417949051890403879840079255169295922593080322634775209689623239873322471161642996440906533187938298969649928516003704476137795166849228875
Calculation with matrix method:
43466557686937456435688527675040625802564660517371780402481729089536555417949051890403879840079255169295922593080322634775209689623239873322471161642996440906533187938298969649928516003704476137795166849228875
Calculation with memoization method:
43466557686937456435688527675040625802564660517371780402481729089536555417949051890403879840079255169295922593080322634775209689623239873322471161642996440906533187938298969649928516003704476137795166849228875
Approximation with Golden Ratio relationship:
4.34665576869389 15e+208
Calculation time with iterative method was 0.0010020732879638672
seconds
Calculation time with matrix method was 0.0 seconds
Calculation time with memoization method was 0.010577917098999023
seconds
Calculation time with Golden Ratio method was 0.0 seconds
```

Καταρχάς σημειώνεται ότι και οι τρεις αλγόριθμοι βγάζουν το ίδιο αποτέλεσμα, ως πρέπει, και ο αλγόριθμος της σχέσης με την χρυσή τομή βγάζει σε πολύ γρήγορο χρόνο μια αρκετά ακριβή προσέγγιση. Φαίνεται επίσης ότι ο αλγόριθμος με τη χρήση πίνακα είναι ο πιο γρήγορος, ακολουθούμενος από τον επαναληπτικό αλγόριθμο και τέλος από τον αναδρομικό με memoization. Αυτή η διαφορά μπορεί να μην είναι αισθητή σε μικρά νούμερα, αν όμως δοκιμάσουμε π.χ. να βρούμε τον F(1.000.000) προκύπτει ο εξής χρόνος:

```
Calculation time with iterative method was 144.22999167442322 seconds
Calculation time with matrix method was 11.864635467529297 seconds
```

Ο υπολογισμός με αναδρομή δεν ήταν εφικτός (θα μπορούσε εύλογα να υποτεθεί ότι θα χρειαζόταν πολύ περισσότερο χρόνο), ενώ ο εκατομμυριοστός όρος της ακολουθίας Fibonacci παραλείφθηκε για ευνόητους λόγους.

(γ) Για την πρόσθεση δύο αριθμών a, b το bit complexity είναι $O(max\{||a||, ||b||\})$. Προς διευκόλυνσή μας θα θεωρήσουμε ότι εφόσον ένας αριθμός $F(n) \propto \phi^n$ (περίπου), έχει ~n bits στη δυαδική του αναπαράσταση. Στον επαναληπτικό αλγόριθμο λοιπόν οι ψηφιοπράξεις θα είναι $O(1+2+...+n-1) = O(n^2)$, εφόσον γίνονται n-1 προσθέσεις.

Στον πολλαπλασιασμό πινάκων χρησιμοποιείται ο επαναλαμβανόμενος τετραγωνισμός, στον οποίο ο "μεγαλύτερος" πολλαπλασιασμός θα γίνει κατά την εκτέλεση του πολ/μου πινάκων:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{\frac{n-2}{2}} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{\frac{n-2}{2}}$$

$$= \begin{bmatrix} F\left(\frac{n}{2}\right) & F\left(\frac{n}{2}-1\right) \\ F\left(\frac{n}{2}-1\right) & F\left(\frac{n}{2}-2\right) \end{bmatrix} \begin{bmatrix} F\left(\frac{n}{2}\right) & F\left(\frac{n}{2}-1\right) \\ F\left(\frac{n}{2}-1\right) & F\left(\frac{n}{2}-2\right) \end{bmatrix}$$

Θα εκτελεστούν λοιπόν οι σχετικοί πολ/μοί των εντολών σε αυτήν την -τελευταία- επανάληψη:

```
a = A[0][0] * A[0][0] + A[0][1] * A[1][0]
b = A[0][0] * A[0][1] + A[0][1] * A[1][1]
c = A[1][0] * A[0][0] + A[1][1] * A[1][0]
d = A[1][0] * A[0][1] + A[1][1] * A[1][1]
```

και αν λάβουμε υπόψη τον απλό πολ/μο ακεραίων το κόστος τους θα είναι ανάλογο του $O(||F(\frac{n}{2})||^2) \approx O(\frac{n^2}{4}) = O(n^2)$, ενώ αν λάβουμε υπόψη τον πολ/μο Gauss-Karatsuba, το κόστος θα είναι $O(||F(\frac{n}{2})||^{log3}) = O(n^{log3})$. Επειδή το κόστος των πολ/μων στις προηγούμενες επαναλήψεις μειώνεται απότομα, θα θεωρήσουμε ότι η ασυμπτωτική πολυπλοκότητα του αλγορίθμου είναι ίση με το κόστος της τελευταίας επανάληψης, όπως υπολογίστηκε πριν. Παρατηρώ ότι στον αλγόριθμο με χρήση πίνακα (με απλό πολ/μο) ακόμα και αν οι επαναλήψεις είναι αισθητά λιγότερες από αυτές του επαναληπτικού αλγορίθμου (logn vs. n), η πολυπλοκότητα ψηφιοπράξεων είναι ίδια, ενώ ελαττώνεται σχετικά με τη χρήση του πολλαπλασιασμού Gauss-Karatsuba.

(δ) Γράφτηκε το ακόλουθο πρόγραμμα σε Python, το οποίο περιλαμβάνει τροποποίηση της `fib_matrix(n)` και εκμεταλλεύεται τις ιδιότητες της ισοτιμίας αριθμών.

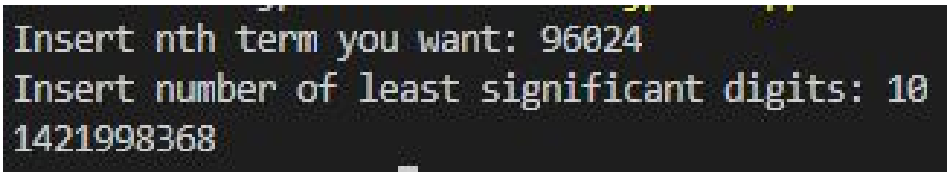
```
def matrix_modpow(A, n, k):
    #for 2x2 matrices
    R = [[1, 0], [0, 1]]
    while n > 0:
        if n % 2 == 1:
```

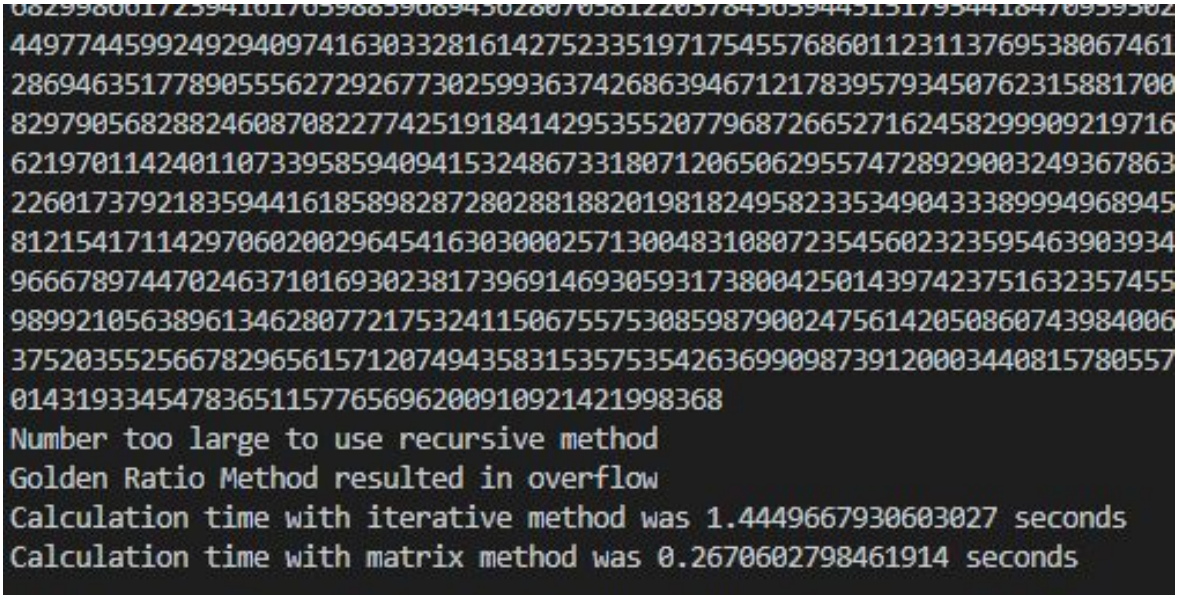
```
            a = R[0][0] * A[0][0] % k + R[0][1] * A[1][0] % k
            b = R[0][0] * A[0][1] % k + R[0][1] * A[1][1] % k
            c = R[1][0] * A[0][0] % k + R[1][1] * A[1][0] % k
            d = R[1][0] * A[0][1] % k + R[1][1] * A[1][1] % k
            R = [[a, b], [c, d]]
        n //= 2
        a = A[0][0] * A[0][0] % k + A[0][1] * A[1][0] % k
        b = A[0][0] * A[0][1] % k + A[0][1] * A[1][1] % k
        c = A[1][0] * A[0][0] % k + A[1][1] * A[1][0] % k
        d = A[1][0] * A[0][1] % k + A[1][1] * A[1][1] % k
        A[0][0] = a % k
        A[0][1] = b % k
        A[1][0] = c % k
        A[1][1] = d % k
    return R


def fib_matrix_mod(n, k):
    if n == 0:
        return 0
    F = [[1, 1], [1, 0]]
    M = matrix_modpow(F, n - 2, 10**k)
    result = (M[0][0] + M[0][1]) % 10**k
    if result != 0:
        return result
    else:
        return k*'0'

n = int(input("Insert nth term you want: "))
m = int(input("Insert number of least significant digits: "))
print(fib_matrix_mod(n, m))
```

Η συνάρτηση για να μην επιστρέψει 0 αντί για τα k τελευταία ψηφία (που είναι 0) επιστρέφει ένα string με k φορές το 0. Θα δοκιμάσουμε το πρόγραμμα για να βρούμε τα 10 τελευταία από τα χιλιάδες ψηφία ενός μεγάλου όρου Fibonacci και θα αντιπαραβάλλουμε το αποτέλεσμα με το προαναφερθέν πρόγραμμα για τους αριθμούς Fibonacci.





Τα δέκα τελευταία ψηφία είναι πράγματι ίσα!

(δ*) Η μέθοδος Fast Doubling βασίζεται στον αλγόριθμο πολ/μού πινάκων. Στην εξίσωση

$$\begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

μπορώ να θέσω όπου n το $2n$, οπότε:

$$\begin{bmatrix} F(2n+1) & F(2n) \\ F(2n) & F(2n-1) \end{bmatrix} = \left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \right)^2$$

$$\begin{bmatrix} F(2n+1) & F(2n) \\ F(2n) & F(2n-1) \end{bmatrix} = \begin{bmatrix} F^2(n+1) + F^2(n) & F(n+1)F(n) + F(n)F(n-1) \\ F(n)F(n+1) + F(n-1)F(n) & F(n)2 + F(n-1)2 \end{bmatrix}$$

Στην τελευταία εξίσωση βασίζεται και ο αλγόριθμος, και ουσιαστικά λειτουργεί όπως ο αλγόριθμος των πολ/μών πινάκων, όπου μόνο παραλείπονται κάποιες παραπανίσιες πράξεις. Η μέθοδος Fast Doubling είναι γρηγορότερη από την πρώτη κατά μία σταθερά, ωστόσο ασυμπτωτικά η πολυπλοκότητα (αριθμητική, ψηφιοπραξεων) είναι ίδια με αυτή του πολ/μού πινάκων. Παρόλα αυτά η διαφορά δεν είναι αμελητέα.
Πηγή: https://www.nayuki.io/page/fast-fibonacci-algorithms

Άσκηση 4

(α) Εφόσον το δίκτυο δεν περιέχει κατευθυνόμενους κύκλους, μπορεί να αναπαραστεί με ένα κατευθυνόμενο ακυκλικό γράφημα (DAG) $G = (V, E, C)$. Χρησιμοποιώντας λοιπόν μία τοπολογική ταξινόμηση με αρχή τον αρχικό μας κόμβο, μπορώ να βρω το συντομότερο μονοπάτι, ελέγχοντας για κάθε κόμβο στην σχετική τοπολογική διάταξη αν το μονοπάτι που οδηγεί σε αυτόν είναι το συντομότερο ξεκινώντας από τους γειτονικούς του αρχικού, ως εξής:

- for all vertices v in V
    - d[v] := inf
- d[S] := 0
- path := empty
- repeat for all topologically arranged vertices v starting from S:
    - for all edges connecting v to each v_next
        - if d[v] + C[Edge to v_next] < d[v_next]
            - d[v_next] = d[v] + C[Edge to v_next]
            - prev[v_next] := v
        - go to the next available vertex
- return prev

Επειδή έχουμε DAG, μπορώ να "χαλαρώσω" κάθε ακμή, ελέγχοντας σε κάθε κόμβο της τοπολογικής ταξινόμησης αν η απόσταση του από τον αρχικό κόμβο S συν το μήκος της ακμής που συνδέει τον τρέχων κόμβο με τους γειτονικούς του είναι μικρότερη από την ήδη ορισμένη απόσταση από τον επόμενο κόμβο. Η πολυπλοκότητα αυτού του αλγορίθμου είναι $\Theta(|E|+|V|)$, η χαμηλότερη δυνατή για το πρόβλημα.

(β) Αν το δίκτυο είχε κύκλους, τότε θα έπρεπε να χρησιμοποιήσουμε τον αλγόριθμο του Dijkstra. Θα λειτουργούσε όμοια με τον παραπάνω αλγόριθμο για το (α), όμως εφόσον δεν υπάρχει τοπολογική ταξινόμηση για τις κορυφές του γραφήματος, στο τέλος κάθε επανάληψης θα επιλέγω να συνεχίσω στην κορυφή που απέχει μικρότερη απόσταση από την τρέχουσα κορυφή και δεν έχω ήδη επισκεφτεί. Η πολυπλοκότητα αυτού του αλγορίθμου όταν υλοποιείται με σωρό Fibonacci είναι $O(|E|+|V|log|V|)$.

(γ) Αν το δίκτυο έχει αρνητικά βάρη, τότε δεν μπορεί να χρησιμοποιηθεί ο αλγόριθμος του Dijkstra. Θα χρησιμοποιηθεί ο αλγόριθμος των Bellman-Ford. Σύμφωνα με αυτόν, θα πρέπει να βρω χρησιμοποιώντας όλες τις ακμές την απόσταση των κορυφών από την αρχική κορυφή, επικαιροποιώντας το αποτέλεσμα κάθε φορά που η απόσταση που προκύπτει είναι μικρότερη από την αποθηκευμένη (ίδια λογική με τους προηγούμενους αλγορίθμους). Για να βρεθούν οι συντομότεροι δρόμοι, αυτή η διαδικασία θα πρέπει να επαναληφθεί $|V-1|$ φορές, οπότε η πολυπλοκότητα του αλγορίθμου θα είναι $O(|E||V|)$.

Άσκηση 5

Αναπτύχθηκε ο εξής αλγόριθμος, για έναν γράφο $G=(V,E,l)$ και έναν πίνακα δ με τις υποτιθέμενες αποστάσεις. Έστω S ο αρχικός κόμβος:

- for each vertex v in V
    - check[v] := False
- shortest_path_tree := {}
- enforce δ(S) == 0
- check[S] = True
- for each edge (u, v) in E
    - enforce δ(v) <= δ(u) + l(u, v)
    - if δ(v) == δ(u) + l(u, v)
        - if !check(v)
            - add edge (u, v) to shortest_path_tree

            - check(v) = True
    - for each vertex v in V
        - enforce check[v] == True
    - return shortest_path_tree

Αυτός ο αλγόριθμος επιβεβαιώνει (μέσω των ελέγχων με τις εντολές `enforce`, που αν αποτύχουν ο αλγόριθμος επιστρέφει αρνητικό αποτέλεσμα) ότι για κάθε ακμή από τον κόμβο u στον v, η -υποτιθέμενη- απόσταση της v από τον αρχικό κόμβο σε σύγκριση με την αντίστοιχη υποτιθέμενη απόσταση της u συν την απόσταση `l(u, v),` είναι είτε ίση, είτε μικρότερη (αν υπάρχει κάποιο συντομότερο μονοπάτι από τον u στον v). Αν αυτές οι δύο αποστάσεις είναι ίσες, τότε η ακμή (u, v) αποτελεί μέρος της συντομότερης διαδρομής (κάτι που δεν οδηγεί σε κύκλο, καθώς για μια ακμή που συμπληρώνει έναν θετικό κύκλο δε θα ισχύει η ισότητα, ενώ αν ο κύκλος είναι αρνητικός ο παραπάνω έλεγχος θα αποτύχει). Στο τέλος ελέγχεται ότι για κάθε κόμβο υπάρχει τουλάχιστον μία ακμή για την οποία ίσχυε η συνθήκη `δ(v) == δ(u) + l(u, v).` Αν ικανοποιούνται όλες αυτές οι συνθήκες, τότε επιστρέφεται το δέντρο συντομότερων διαδρομών, όπως έχει αυτό διαμορφωθεί. Είναι φανερό ότι εκτελούνται $\Theta(n+m)$ εντολές, όπως φαίνεται από τις επαναλήψεις για όλες τις ακμές και όλους τους κόμβους, μέσα στις οποίες εκτελούνται εντολές κόστους $O(1)$.

Άσκηση 6

(α) Μπορούμε να χρησιμοποιήσουμε τον εξής αλγόριθμο για το γράφημα $G=(V,E,l)$ που αναπαριστά το οδικό δίκτυο:

- for each edge e in E
    - if l(e) > L
        - remove edge e from graph
- if connected(s, t)
    - return True
- else
    - return False

Ο έλεγχος `connected(s, t)` που ελέγχει αν οι κορυφές s και t είναι συνδεδεμένες με κάποιο μονοπάτι, μπορεί να υλοποιηθεί με μία απλή DFS ή BFS, που έχει πολυπλοκότητα $O(n+m)$. Η συνολική πολυπλοκότητα λοιπόν του αλγορίθμου θα είναι $O(n+m+m) = O(n+2m) = O(n+m) = O(m)$, εφόσον σε έναν συνεκτικό γράφο που αναπαριστά ένα οδικό δίκτυο μπορούμε εύλογα να υποθέσουμε ότι $n \leq m$

(β) Για τη λύση αυτή του προβλήματος αρκεί να βρω το δέντρο ελαχίστων διαδρομών, π.χ. με τον αλγόριθμο του Prim, καθώς μετά ο δρόμος που θα προκύψει στο δέντρο ανάμεσα στο s και στο t (που μπορώ να βρω με μία DFS) θα έχει εγγυημένα την ελάχιστη μέγιστη ακμή από όλους τους δρόμους ανάμεσα στο s και t, οπότε αυτός ο δρόμος θα έχει και τη μικρότερη αυτονομία L. Η πολυπλοκότητα είναι $O(mlogn+m+n) = O(mlogm)$, εφόσον σε έναν συνεκτικό γράφο που αναπαριστά ένα οδικό δίκτυο μπορούμε εύλογα να υποθέσουμε ότι $n \leq m$ (Σίγουρα ισχύει ότι $n - 1 \leq m$).

(γ*) Ο Camerini πρότεινε έναν αλγόριθμο για την εύρεση ενός Minimum Bottleneck

Spanning Tree (MSBT) σε ένα γράφο, δηλαδή ένα συνδετικό δέντρο του οποίου η πιο βαριά ακμή είναι όσο το δυνατόν πιο ελαφριά. Αυτός ο αλγόριθμος διαιρεί το σύνολο των ακμών του γράφου σε δύο υποσύνολα, στα οποία τα βάρη των ακμών του πρώτου δεν ξεπερνούν τα βάρη των ακμών του δεύτερου. Αυτός ο διαχωρισμός δηλαδή γίνεται με τη βοήθεια της διαμέσου-median των βαρών των ακμών, που βρίσκεται σε χρόνο O(m). Αν σχηματιστεί ένα συνδετικό δέντρο με τις ακμές από το σύνολο με τις ελαφριές ακμές, τότε υπολογίζεται ένα MBST στο υπογράφημα, που θα αποτελεί και MBST και του αρχικού γραφήματος. Αν δεν υπάρχει συνδετικό δέντρο, κάθε ανεξάρτητη κορυφή συγχωνεύεται σε έναν "υπέρ-κόμβο" (supervertex), και υπολογίζεται ένα MBST στον γράφο που αποτελείται από αυτούς τους υπέρ-κόμβους και τις ακμές στο υποσύνολο με τις πιο βαριές ακμές, εφόσον ένα δάσος σε κάθε ανεξάρτητο μέρος αποτελεί κομμάτι από MBST στο αρχικό γράφημα. Αυτή η διαδικασία επαναλαμβάνεται μέχρι δύο (υπέρ)κόμβοι να μείνουν και η ακμή που έχει μείνει θα προστεθεί. Βρίσκεται λοιπόν ένα MSBT που αποτελείται από όλες τις ακμές που βρέθηκαν πρότερα. Ο αλγόριθμος του Camerini:

```
function MBST(graph G, weights w)
E := the set of the graph's edges
if |E| == 1
    return E
else
    A := half edges with weights greater than the median
    B := E - A
    F := forest of G(B) // graph with edges of B
    if F is a spanning tree
        return MSBT(G(B), w)
    else
        return MBST(G(A)*, w)∪F // G(A)* is G(A) with
                                 // supervertices
```

Ο αλγόριθμος βρίσκει τον διάμεσο και διαχωρίζει τις ακμές του γράφου σε $O(m)$, βρίσκει το δάσος F σε $O(m)$, και εφόσον σε κάθε αναδρομή το πλήθος των ακμών υποδιπλασιάζεται, το συνολικό κόστος θα είναι $O(m + m/2 + m/4 + ... + 1) = O(m)$. Για να βρω τη διαδρομή με τη χαμηλότερη αυτονομία λοιπόν, δε χρειάζεται παρά να βρω τη διαδρομή από το s στο t στο MSBT του οδικού δικτύου, που γίνεται με μία DFS, οπότε ο χρόνος παραμένει γραμμικός. Αξιοσημείωτο είναι ότι και το MST που βρέθηκε στο προηγούμενο ερώτημα π.χ. με τον αλγόριθμο του Prim αποτελεί MSBT (ένα MSBT δεν αποτελεί απαραίτητα MST, ωστόσο).
Πηγή:
https://en.wikipedia.org/wiki/Minimum_bottleneck_spanning_tree#Camerini's_algorithm_for_undirected_graphs

Άσκηση 7

Θα χρησιμοποιήσω Δυναμικό Προγραμματισμό για την επίλυση του προβλήματος.
Αναπτύσσω τον κάτωθι αλγόριθμο:

```
cost[0] := 0
for j from 1 to T
```

```
        ○ if S[j] == 0
            ■ cost[j] = cost[j-1]
            ■ continue
        ○ cost[j] = cost[j - 1] + p[1] // start with 1-day ticket
        ○ ticket[j] = 1
        ○ for i from 2 to k
            ■ if j >= i
                ● cost[j] = min(cost[j], cost[j-c[i]] + p[i])
                ● if i was chosen ticket[j] := i
            ■ else
                ● cost[j] = min(cost[j], p[i])
    ● result = {} // works like a Python dictionary
    ● i := T
    ● while i != 0
        ○ result[i] = ticket[i]
        ○ i = i - c[ticket[i]]
    ● return cost[T], result
```

Για να βρω το ελάχιστο κόστος για τα εισιτήρια ακολουθώ παρόμοια τακτική με το πρόβλημα του σακιδίου. Σπάω το πρόβλημά μου σε μικρότερα υποπροβλήματα, και συγκεκριμένα βρίσκω τι θα έκανα αν είχα ένα διάστημα με λιγότερες από T μέρες. Αποθηκεύω το αποτέλεσμα για το κάθε κόστος και αντίστοιχο τύπο εισιτηρίου σε σχετικό πίνακα, και το χρησιμοποιώ στις επόμενες επαναλήψεις, μέχρι να βρω το ζητούμενο αποτέλεσμα, όταν φτάσω στην τελευταία μέρα. Μετά βρίσκω την ακολουθία των εισιτηρίων που θα αγοραστούν. Για τη μέρα i το κόστος του εισιτηρίου που οφείλω να αγοράσω θα είναι ίσο με:

- $cost(i-1),\ \text{αν}\ S[i]=0$
- $min\{cost(i-c[j]+p[j])\}, \text{για}\ j\ =\ 1,2,\ 3,...i$

Η πολυπλοκότητα του αλγορίθμου είναι $O(Tk)$.

Άσκηση 8

Θα χρησιμοποιήσω πάλι Δυναμικό προγραμματισμό, με λογική παρόμοια με αυτή του προηγούμενου προβλήματος, όμως αυτή τη φορά θα πρέπει να λαμβάνουμε υπόψη περισσότερες παραμέτρους για τον υπολογισμό της κατάλληλης παραγγελίας. Για την n-οστή μέρα, η κατάλληλη παραγγελία p_n μπορεί να είναι ύψους:

- ίσου με d_n, αν $d_n > S$, οπότε όλοι οι υπολογιστές πωλούνται αυθημερόν με κόστος μεταφοράς K. Αν αυτό ισχύει τη προηγούμενη μέρα κατά τη βέλτιστη τακτική δεν θα είχαν μείνει υπολογιστές αποθηκευμένοι.
- ίσου με d_n, αν $d_n \le S$, και συμφέρει η μεταφορά τους εκείνη τη μέρα από την διατήρησή τους στην αποθήκη για (ακόμα) μία μέρα, δηλαδή όταν $K \le d_n C$
- ίσου με 0, αν $d_n \le S$, και συμφέρει η διατήρηση τους στην αποθήκη για μία μέρα από τη μεταφορά τους, δηλαδή όταν $K \ge d_n C$. Αυτό σημαίνει ότι την (n-1)-οστή μέρα θα υπάρχουν d_n υπολογιστές διαθέσιμοι, που θα αποθηκευτούν προς κόστος $d_n C$.

Σύμφωνα με αυτά αποφαίνομαι και για το κόστος των παραγγελιών για κάθε ημέρα i, που λειτουργεί με ακριβώς την ίδια λογική με την παραγγελία της n-οστής μέρας, μόνο που τώρα πρέπει να λάβω υπόψιν και το απόθεμα που θα οφείλω να κρατάω για τις επόμενες μέρες. Αναπτύσσω των κάτωθι αλγόριθμο, που αναδρομικά υπολογίζει το ελάχιστο κόστος:

- `function cost[i, stock]`
  - `if i == 1`
    - `order[i] = stock + d[i]`
    - `return K`
  - `if d[i] + stock > S`
    - `order[i] = stock + d[i]`
    - `return cost[i-1, 0] + K`
  - `else if cost(i-1, s+d[i]) + (stock + d[i])*C >= K + cost(i-1, 0)`
    - `order[i] = stock + d[i]`
    - `return cost[i-1, 0] + K`
  - `else`
    - `order[i] = 0`
    - `return cost[i-1, stock + d[i]] + (stock + d[i])*C`
- `return cost[n, 0], order`

Αναφορικά με την ακολουθία των παραγγελιών, όταν συμφέρει η αποθήκευση της παραγγελίας d_i την (i-1)-οστή μέρα, τότε την i μέρα δεν υπάρχει κάποια παραγγελία, και η d_i προστίθεται στην παραγγελία της (i-1)-οστής μέρας (που μπορεί με τη σειρά της αν ισχύει η ίδια συνθήκη να είναι 0 και να προστεθεί στη παραγγελία της (i-2)-οστής μέρας, κ.ο.κ). Έτσι, όταν φτάσουμε είτε σε μία μέρα που θα συμφέρει το κόστος μεταφοράς από το κόστος διατήρησης του "συσσωρευμένου αποθέματος", είτε σε μια μέρα που το απόθεμα μαζί με την παραγγελία d_i υπερβεί το ποσό S, τότε αυτή τη μέρα θα πρέπει να γίνει η παραγγελία i, η οποία θα καλύπτει την ημερήσια πώληση d_i, καθώς και το απόθεμα το οποίο θα καλύψει τις επόμενες μέρες, στη περίπτωση που χρειαστεί. Αυτή η διαδικασία θα επαναληφθεί αναδρομικά μέχρι να φτάσουμε στην πρώτη μέρα, όπου θα γίνει μια παραγγελία κόστους K και ύψους d_1 συν το αναγκαίο απόθεμα για τις επόμενες μέρες. Η πολυπλοκότητα του αλγορίθμου είναι $O(nS)$, όσο και ο μέγιστος αριθμός των συνδυασμών `cost[i, j]` που μπορούν να υπολογιστούν.

Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

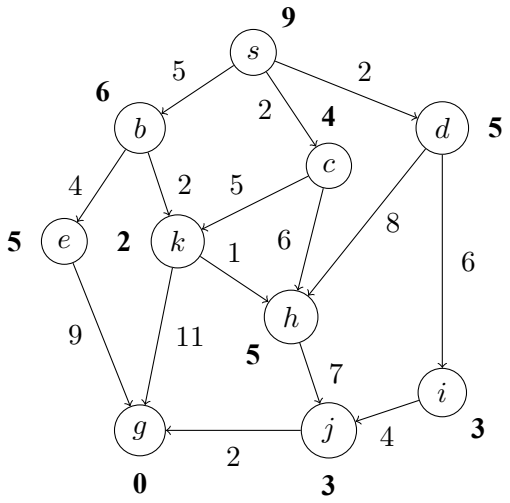
Θεμελιώδη Θέματα Επιστήμης Υπολογιστών, 2020-21

3η σειρά γραπτών ασκήσεων

(τεχνητή νοημοσύνη και βάσεις δεδομένων)

Άσκηση 1. (Αλγόριθμοι αναζήτησης λύσης - Άσκηση)

Δίνεται ο παρακάτω χώρος αναζήτησης, όπου s είναι η αρχική και g η τελική κατάσταση. Οι αριθμοί δίπλα σε κάθε ακμή αντιπροσωπεύουν την πραγματική απόσταση των κόμβων που συνδέει η ακμή, και οι αριθμοί δίπλα σε κάθε κατάσταση (με έντονα γράμματα) αντιπροσωπεύουν την τιμή της ευριστικής εκτίμησης της απόστασης μέχρι την τελική κατάσταση.



1. Εκτελέστε τον αλγόριθμο αναρρίχησης λόφων και τον αλγόριθμο A* για το παραπάνω πρόβλημα.

2. Πόσες λύσεις έχει το πρόβλημα και ποια είναι η βέλτιστη λύση του προβλήματος; Βρίσκουν τη βέλτιστη λύση οι παραπάνω αλγόριθμοι; Για αυτούς που τη βρίσκουν μπορούμε να είμαστε βέβαιοι εκ των προτέρων ότι θα τη βρουν με βάση τα χαρακτηριστικά του προβλήματος;

Άσκηση 2. (Ευφυείς Πράκτορες - Θέμα)

Καλείστε να σχεδιάσετε το μηχανισμό επιλογής ενέργειας ενός συστήματος τεχνητής νοημοσύνης, το οποίο έχει ως λειτουργικό στόχο την αυτόματη πλοήγηση οχημάτων σε δρόμο ταχείας κυκλοφορίας (αυτοκινητόδρομο), συμπεριλαμβανομένης της δυνατότητας να διαχειρίζεται επείγουσες καταστάσεις. Υποθέτουμε ότι το σύστημα αυτόματης πλοήγησης έχει όλους τους απαραίτητους αισθητήρες (σε συνεργασία και με αισθητήρες του αυτοκινητοδρόμου) για να αναγνωρίζει τη θέση στην οποία βρίσκεται, τα αντικείμενα του χώρου, το είδος και τις ιδιότητές τους, καθώς και τη θέση στην οποία βρίσκονται τα αντικείμενα αυτά. Επομένως, μπορεί να αναγνωρίζει την κατάσταση του κόσμου. Επιπλέον, γνωρίζει την κατάσταση στόχο, δηλαδή τον προορισμό του οχήματος (μπορείτε να υποθέσετε

ότι ο προορισμός είναι μία έξοδος από τον αυτοκινητόδρομο). Τέλος, υποθέτουμε ότι το σύστημα πλοήγησης είναι ικανό να εκτελέσει τις απαραίτητες ενέργειες που αλλάζουν τη θέση και τις ιδιότητες του αυτοκινήτου, δεδομένων των φυσικών περιορισμών, καθώς και να αναγνωρίσει ποιες από τις ενέργειες αυτές είναι δυνατές και επιτρεπτές. Προφανώς, οι επιτρεπτές ενέργειες καθορίζουν τους τελεστές μετάβασης από μία κατάσταση σε μία άλλη. Στο πλαίσιο της μελέτης και του σχεδιασμού του μηχανισμού επιλογής ενέργειας του συστήματος πλοήγησης, υποθέτοντας ότι ο χώρος του αυτοκινητοδρόμου δεν περιέχει περισσότερα από 5 διαφορετικά είδη αντικειμένων και δεν υπάρχει άλλος μηχανισμός αλλαγής της κατάστασης του οχήματος, εκτός από τις ενέργειες του συστήματος πλοήγησης, καλείστε να κάνετε τα εξής:

1. Να καθορίσετε το περιβάλλον, τους αισθητήρες, τις δράσεις και τους δείκτες επίδοσης, κάνοντας τις απαραίτητες αφαιρέσεις.

2. Να καθορίσετε τον κόσμο του προβλήματος και να δώσετε ένα παράδειγμα μίας κατάστασης του κόσμου. Να καθορίσετε τους τελεστές μετάβασης από μία κατάσταση σε μία άλλη, και να δώσετε μερικά παραδείγματα τελεστών.

3. Να σχεδιάσετε ευριστικές συναρτήσεις που εκτιμούν τόσο το κόστος μετάβασης από μία κατάσταση σε μία άλλη, όσο και το υπολοιπόμενο κόστος μέχρι την τελική κατάσταση.

4. Να δώσετε την αναπαράσταση, σε Προτασιακή Λογική, μίας κατάστασης του κόσμου (του παραδείγματος που έχετε δώσει στο Ερώτημα 2).

Άσκηση 3. (Ταξινομητές Naive Bayes)

Θέλουμε να αναπτύξουμε ένα σύστημα που προβλέπει αν ένα μήνυμα ηλεκτρονικού ταχυδρομείου είναι διαφημιστικό ή όχι, ανάλογα με το αν περιέχει τις λέξεις «ευκαιρία» και «μοναδικός». Δίνεται ο Πίνακας 1 που συνοψίζει τα δεδομένα, δηλαδή τα μέχρι τώρα στοιχεία που έχουμε από τα μηνύματα ηλεκτρονικού ταχυδρομείου που έχουμε λάβει (έχουμε εξετάσει αν είναι διαφημιστικά ή όχι). Να σχεδιάσετε για το σκοπό αυτό ένα ταξινομητή Naive Bayes και να εξηγήσετε τη λειτουργία του με ένα παράδειγμα ταξινόμησης ενός μηνύματος ηλεκτρονικού ταχυδρομείου (πρόβλεψης αν είναι διαφημιστικό ή όχι) που περιέχει τη λέξη «ευκαιρία» και τη λέξη «μοναδικός».

Πίνακας 1: Συγκεντρωτικός πίνακας μηνυμάτων ηλεκτρονικού ταχυδρομείου

Σύνολο μηνυμάτων που	Είναι διαφημιστικά	Δεν είναι διαφημιστικά
Περιέχουν τη λέξη «ευκαιρία»	285	225
Δεν περιέχουν τη λέξη «ευκαιρία»	15	1275
Περιέχουν τη λέξη «μοναδικός»	240	600
Δεν περιέχουν τη λέξη «μοναδικός»	60	900

Άσκηση 4. (Entity-Relationship Model —Μοντέλο Οντοτήτων-Συσχετίσεων)

Ένας συνεταιρισμός γαλακτοκόμων παράγει και πουλάει γαλακτοκομικά προϊόντα. Στα πλαίσια μιας επέκτασης αποφασίζει να αποθηκεύσει τα δεδομένα του σε μια βάση δεδομένων. Στην βάση αυτήν θα

αποθηκεύονται τα στοιχεία των γαλακτοκόμων, δηλαδή το ονοματεπώνυμό τους, η διεύθυνσή τους και τα τηλέφωνα επικοινωνίας μαζί τους. Θα αποθηκεύονται επίσης στοιχεία για τα προϊόντα τα οποία παράγουν, δηλαδή το είδος του προϊόντος, η ποσότητα και το κόστος του. Τέλος θα αποθηκεύονται τα στοιχεία των πελατών, οι οποίοι αγοράζουν τα προϊόντα. Μας ενδιαφέρει επίσης η ημερομηνία που κάποιος πελάτης αγόρασε ένα προϊόν. Για τους πελάτες θα αποθηκεύεται το ονοματεπώνυμό τους, η διεύθυνσή τους, και το ΑΦΜ τους.

Καλείστε να σχεδιάσετε το ER διάγραμμα (διάγραμμα οντοτήτων-συσχετίσεων) έτσι ώστε να αποθηκεύονται τα δεδομένα της φάρμας, έτσι όπως αυτά περιγράφονται παραπάνω. Μην κάνετε υποθέσεις που δεν περιγράφονται. Ως κλειδιά σημειώστε γνωρίσματα (ή συνδυασμούς γνωρισμάτων) με μοναδικές τιμές, αν υπάρχουν. Αν δεν υπάρχουν τέτοια γνωρίσματα, προσθέστε δικά σας κλειδιά.

Προθεσμία υποβολής και οδηγίες. Η σειρά αυτή θα συμπληρωθεί σύντομα με κάποιες ασκήσεις ακόμη (στην περιοχή των βάσεων δεδομένων).

Οι απαντήσεις θα πρέπει να υποβληθούν έως τις 10/1/2021, στις 23:59, σε ηλεκτρονική μορφή, στο mycourses (φροντίστε το τελικό αρχείο να είναι μεγέθους <2MB συνολικά).

Συνιστάται *θερμά* να αφιερώσετε ικανό χρόνο για να λύσετε τις ασκήσεις μόνοι σας προτού καταφύγετε σε οποιαδήποτε *θεμιτή* βοήθεια (διαδίκτυο, βιβλιογραφία, συζήτηση με συμφοιτητές). Σε κάθε περίπτωση, οι απαντήσεις θα πρέπει να είναι *αυστηρά* ατομικές.

Για να βαθμολογηθείτε θα πρέπει να παρουσιάσετε σύντομα τις λύσεις σας σε ημέρα και ώρα που θα ανακοινωθεί αργότερα.

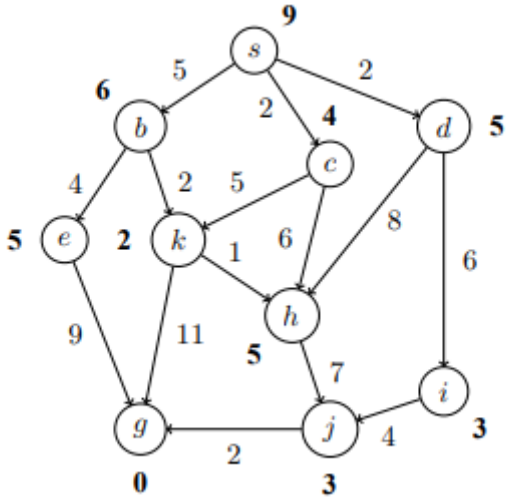
Για απορίες / διευκρινίσεις: στείλτε μήνυμα στη διεύθυνση `focs_course@ails.ece.ntua.gr`.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Μάθημα 3ου εξαμήνου: ΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘΘ
Ονοματεπώνυμο Φοιτητή: ναι και να τρέχω στα φοιτητοδικεία μετά
Α.Μ: ^^^^

3η σειρά γραπτών ασκήσεων

Τεχνητή Νοημοσύνη - Βάσεις Δεδομένων

Άσκηση 1



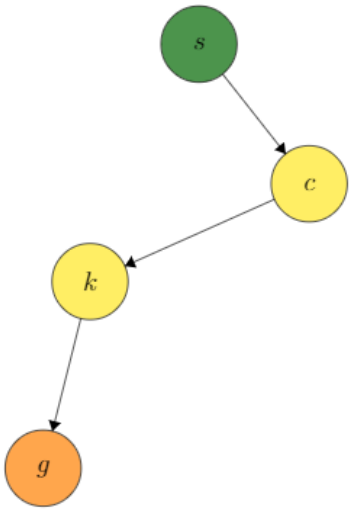
Ο αλγόριθμος αναρρίχησης λόφων ορίζεται με τα εξής βήματα:

1) Όρισε τον τρέχοντα κόμβο ως τη ρίζα του δέντρου
2) Μέχρι που ο τρέχων κόμβος δεν είναι κόμβος στόχος, εκτέλεσε:
    a) Βρες τα παιδιά του τρέχοντος κόμβου, και στη συνέχεια βρες αυτό με την ελάχιστη υπολογιζόμενη υπόλοιπη απόσταση από το στόχο
    b) Εάν ο τρέχων κόμβος δεν έχει παιδιά ή το παιδί που βρέθηκε στο βήμα 2a) έχει μεγαλύτερη τιμή ευριστικής από αυτόν πήγαινε στο Βήμα 3.
    c) Όρισε τον κόμβο που βρέθηκε στο Βήμα 2a) ως τρέχων κόμβο.
3) Εάν βρήκαμε ένα κόμβο στόχο τότε ανακοινώνουμε επιτυχία αλλιώς ανακοινώνουμε αποτυχία

Τον εκτελούμε για τον τρέχοντα χώρο αναζήτησης, ξεκινώντας από τον κόμβο s:

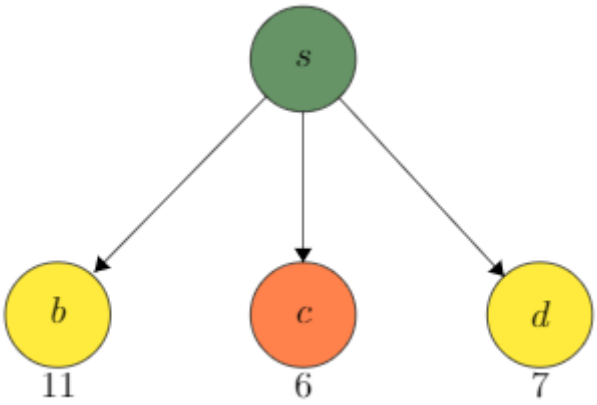
- Για τρέχοντα κόμβο τον s,το παιδί με την ελάχιστη ευριστική απόσταση από το στόχο είναι ο c,και αφού αυτή είναι μικρότερη από την ευριστική απόσταση του κόμβου s ($4 < 9$), θα ορίσουμε τον c ως τρέχοντα κόμβο.
- Για τρέχοντα κόμβο τον c,το παιδί με την ελάχιστη ευριστική απόσταση από το στόχο είναι ο k,και αφού αυτή είναι μικρότερη από την ευριστική απόσταση του κόμβου c ($2 < 4$), θα ορίσουμε τον k ως τρέχοντα κόμβο.
- Για τρέχοντα κόμβο τον k,το παιδί με την ελάχιστη ευριστική απόσταση από το στόχο είναι ο g,και αφού αυτή είναι μικρότερη από την ευριστική απόσταση του κόμβου k ($0 < 2$), θα ορίσουμε τον g ως τρέχοντα κόμβο.
- Ο κόμβος g δεν έχει παιδιά. Ωστόσο, ο κόμβος gπρόκειται για τον κόμβο στόχο. Συνεπώς, επιστρέφεται επιτυχές αποτέλεσμα.

Η διαδρομή που ακολουθήθηκε λοιπόν είναι η $s \rightarrow c \rightarrow k \rightarrow g$, που φαίνεται και στο κάτωθι σχήμα, και έχει συνολικό μήκος ίσο με 18:

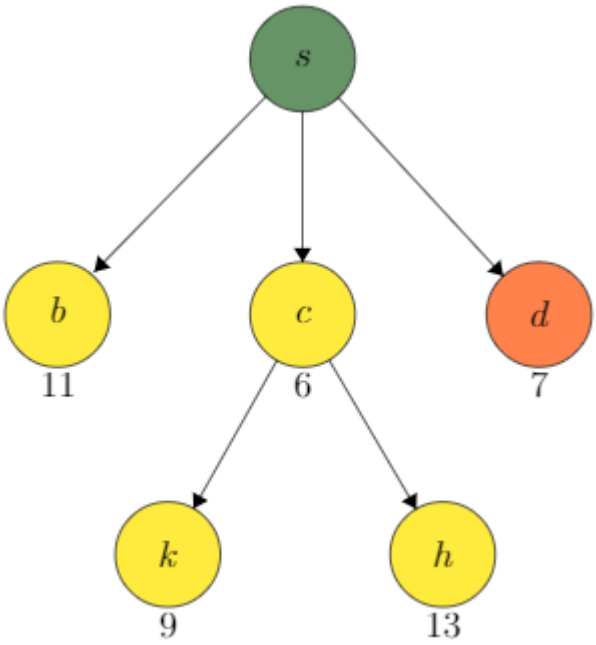


Ο αλγόριθμος A* συνδυάζει ουσιαστικά τη λογική των αλγορίθμων Best First και Branch and Bound. Η διαδικασία που ακολουθούμε είναι να επεκτείνουμε το μονοπάτι με τον καλύτερο από όλους τους κόμβους στο μέτωπο αναζήτησης του δέντρου. Για αυτόν το σκοπό χρησιμοποιούμε τη σύνθετη ευριστική συνάρτηση $f(k) = g(k) + h(k)$, όπου $g(k)$ η απόσταση της k από την αρχική κατάσταση (γνωστή και πραγματική), και $h(k)$ η εκτίμηση της απόστασης της k από το στόχο, που υπολογίζεται μέσω μιας ευριστικής συνάρτησης. Στην παρούσα περίπτωση θα ακολουθηθούν τα εξής βήματα:

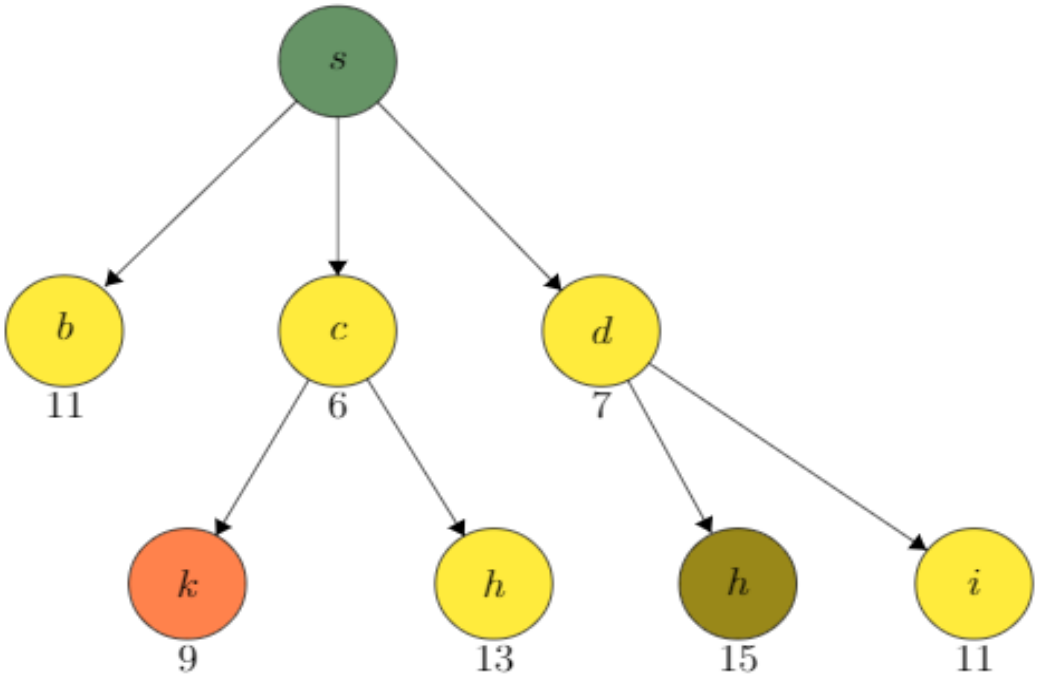
Βήμα 1ο: Επεκτείνω τον αρχικό κόμβο s, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης f που περιγράφεται ανωτέρω. Είναι $f(b) = 5 + 6 = 11,\ f(c) = 2 + 4 = 6,\ f(d) = 2 + 5 = 7$. Ο κόμβος λοιπόν στο μέτωπο αναζήτησης με τη μικρότερη τιμή της ευριστικής συνάρτησης είναι ο c, και αυτός θα επεκταθεί στο επόμενο βήμα.



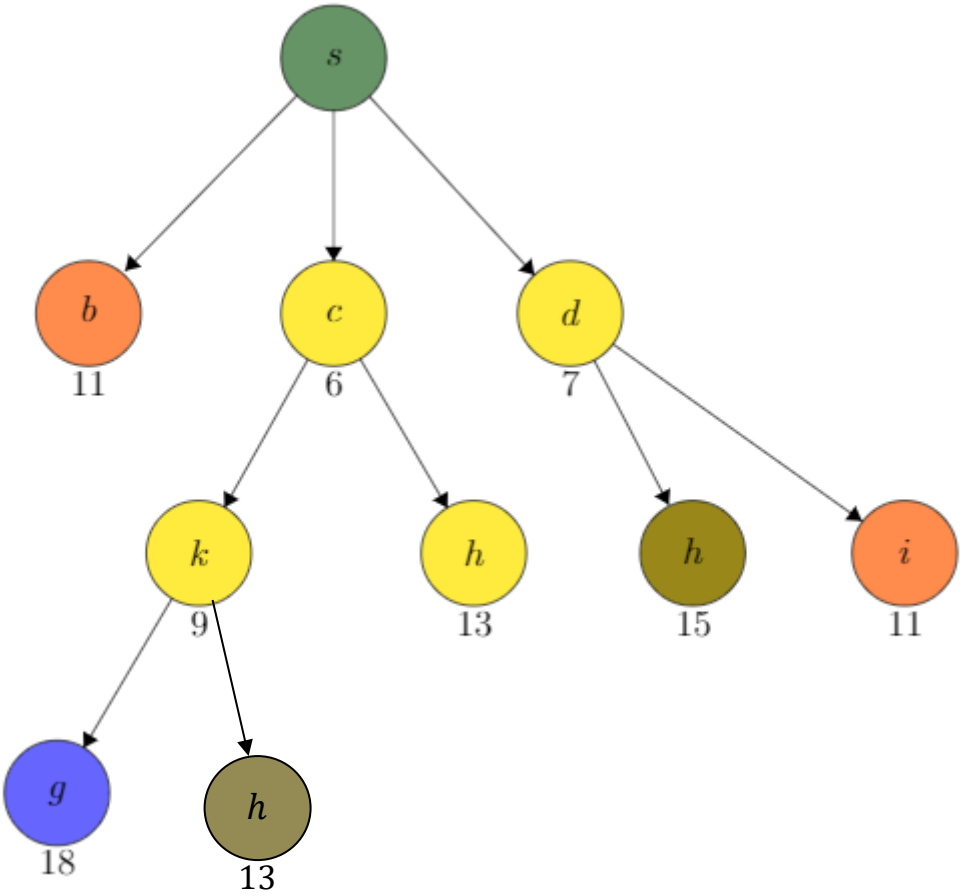
Βήμα 2ο: Επεκτείνω τον κόμβο c, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης f. Είναι $f(k) = 7 + 2 = 9,\ f(h) = 8 + 5 = 13$. Άρα ο «καλύτερος» κόμβος στο μέτωπο αναζήτησης είναι τώρα ο d, και θα επεκταθεί στο επόμενο βήμα.



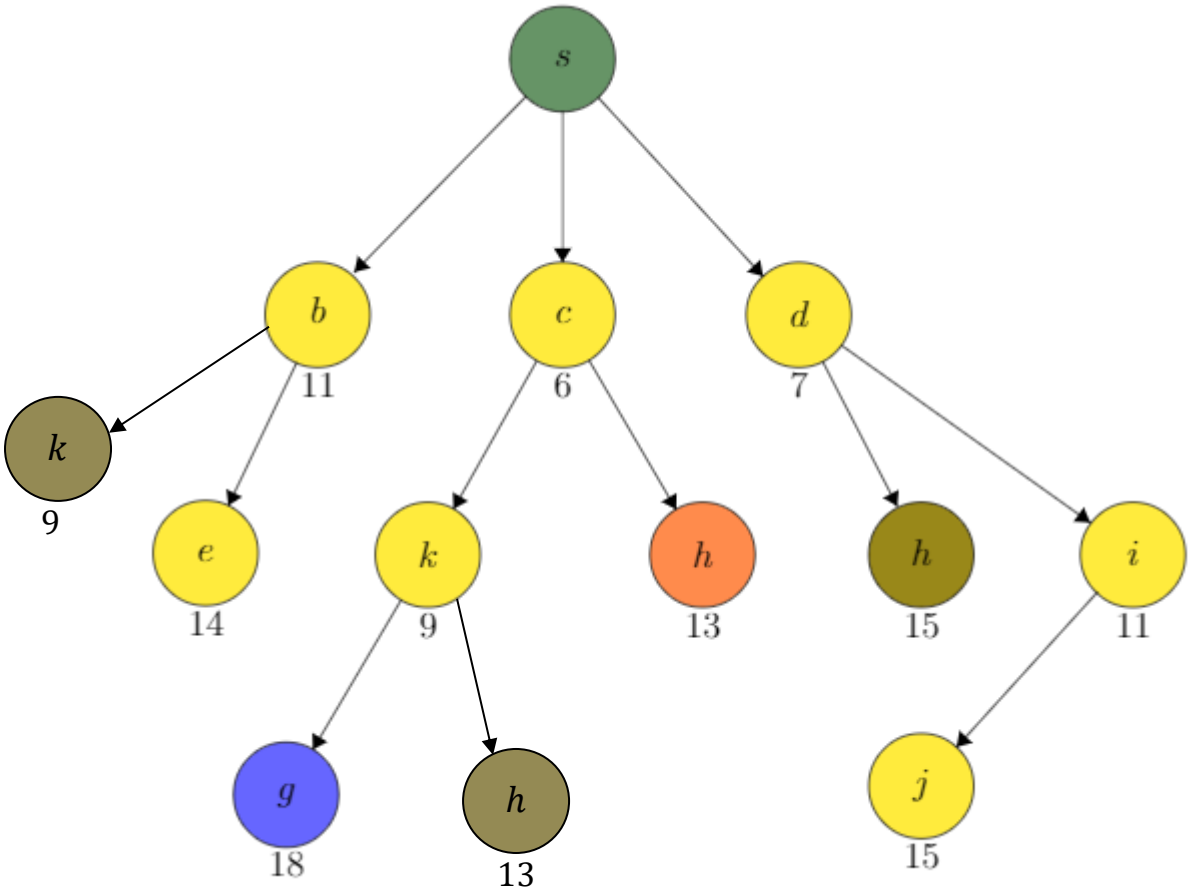
Βήμα 3ο: Επεκτείνω τον κόμβο d, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης f. Είναι $f(h) = 10 + 5 = 15$, $f(i) = 8 + 3 = 11$. Όμως για τον κόμβο h έχουμε υπολογίσει ήδη την τιμή 13, που είναι μικρότερη από την παρούσα, οπότε θα «κλαδέψουμε» τον κόμβο με την τιμή 15, δηλαδή θα τον αφαιρέσουμε από το μέτωπο αναζήτησης (φαίνεται στο σχήμα με καφέ χρώμα). Τελικά ο «καλύτερος» κόμβος στο μέτωπο αναζήτησης είναι τώρα ο k, και θα επεκταθεί στο επόμενο βήμα.



Βήμα 4ο: Επεκτείνω τον κόμβο k, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης f. Είναι $f(g) = 18 + 0 = 18$, $f(h) = 8 + 5 = 13$. Όμως για τον κόμβο h έχουμε υπολογίσει ήδη την τιμή 13, που είναι ίση με την παρούσα, οπότε θα «κλαδέψουμε» τον κόμβο με την τιμή 13, εφόσον δεν υπάρχει νόημα στο να συνεχίσουμε την αναζήτηση από δύο ίδιους κόμβους με την ίδια τιμή. Ο κόμβος g είναι ο τελικός, όμως η αναζήτηση θα συνεχιστεί εφόσον υπάρχουν ακόμα μονοπάτια που δεν έχουν ολοκληρωθεί και έχουν μέχρι τώρα μικρότερη τιμή από την απόσταση που βρέθηκε μόλις για το μονοπάτι (18). Τώρα οι «καλύτεροι» κόμβοι στο μέτωπο αναζήτησης είναι οι b και i, και θα επεκταθούν στο επόμενο βήμα.

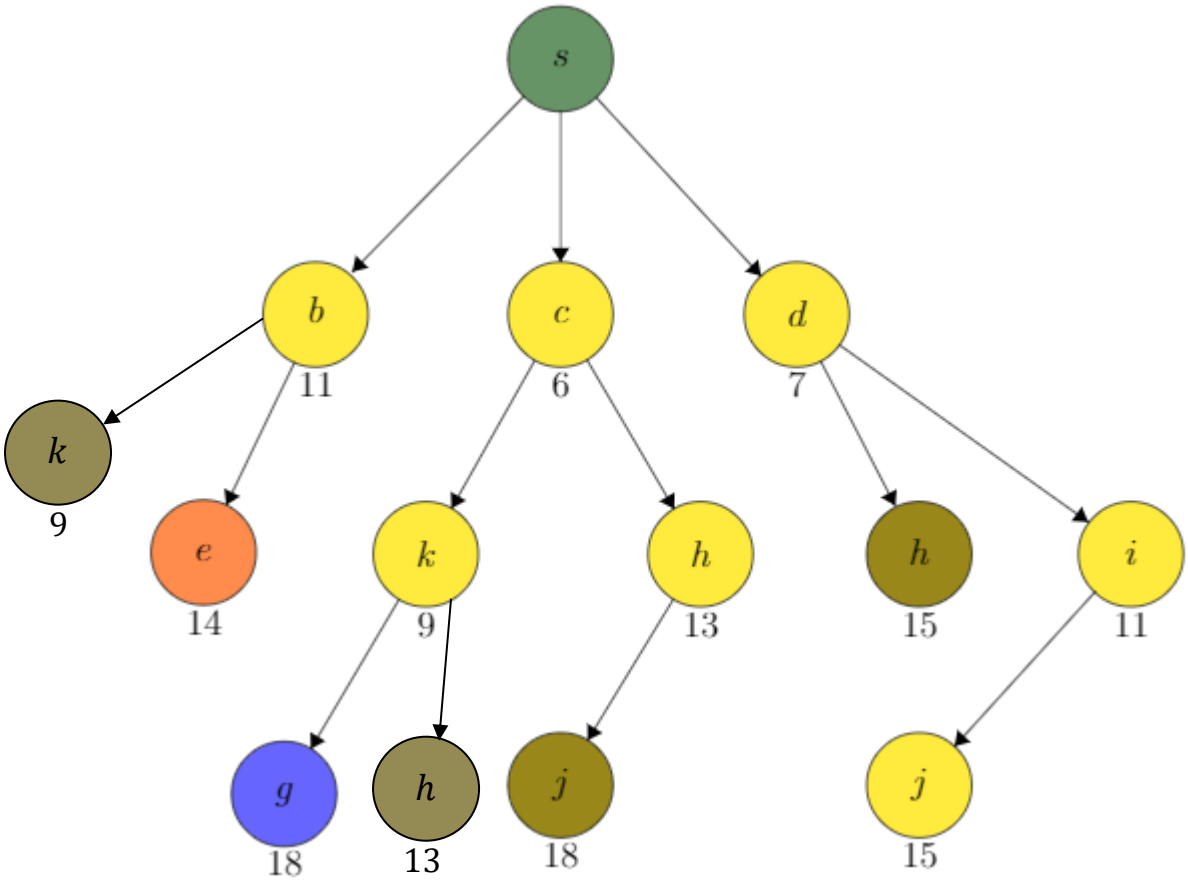


Βήμα 5ο: Επεκτείνω τους κόμβους b και i, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης k. Είναι $f(k) = 7 + 2 = 9$, $f(e) = 9 + 5 = 14$, $f(j) = 12 + 3 = 15$. Όμως για τον κόμβο k έχουμε υπολογίσει ήδη την τιμή 9, που είναι ίση με την παρούσα, οπότε θα «κλαδέψουμε» τον κόμβο με την τιμή 13, εφόσον δεν υπάρχει νόημα στο να συνεχίσουμε την αναζήτηση από δύο ίδιους κόμβους με την ίδια τιμή. Τώρα ο «καλύτερος» κόμβος στο μέτωπο αναζήτησης είναι ο h, και θα επεκταθεί στο επόμενο βήμα. (Κανονικά η επέκταση των κόμβων θα γινόταν σειριακά, αλλά στη συγκεκριμένη περίπτωση δεν εμφανίζεται κάποια διαφορά αφού έτσι κι αλλιώς αμέσως μετά τον b θα επεκτεινόταν ο i)

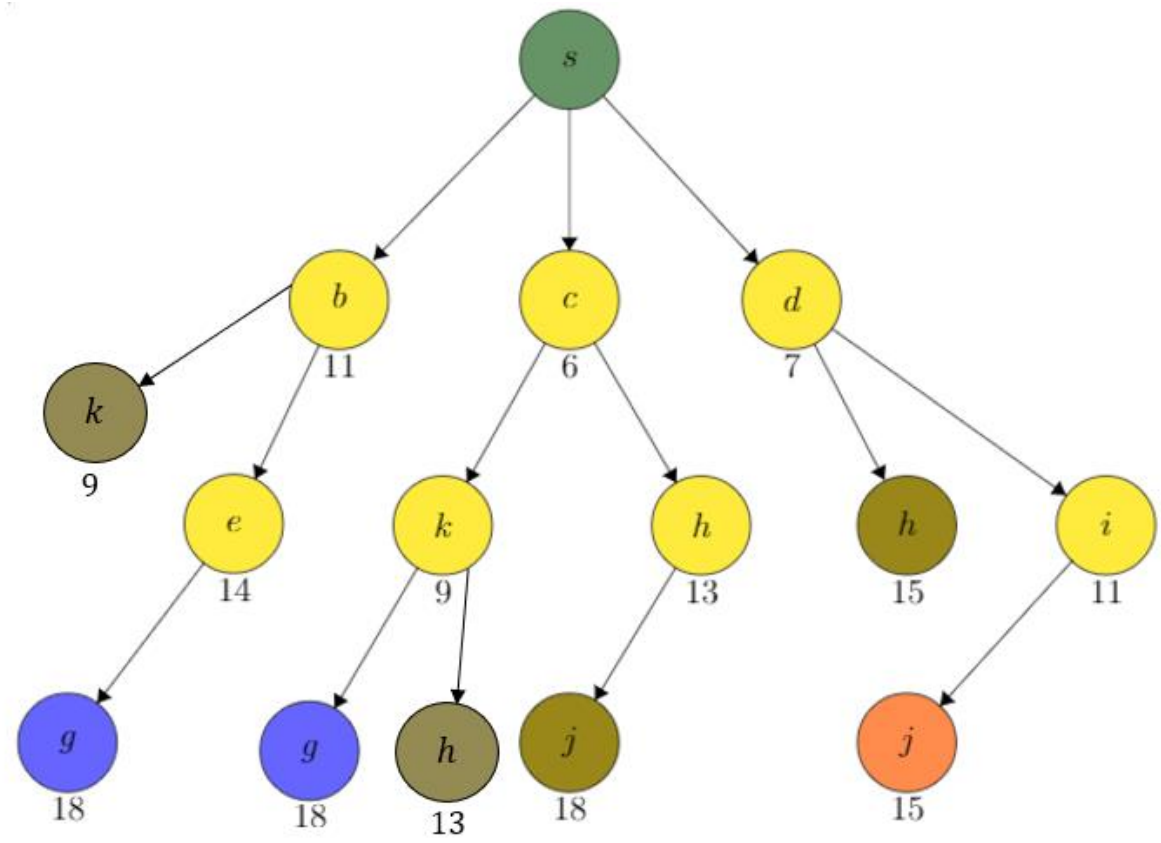


Βήμα 6ο: Επεκτείνω τον κόμβο h, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης f. Είναι $f(j) = 15 + 3 = 18$. Όμως, για τον κόμβο j έχουμε υπολογίσει ήδη την τιμή 15, που είναι μικρότερη από την παρούσα, οπότε θα «κλαδέψουμε» τον κόμβο με

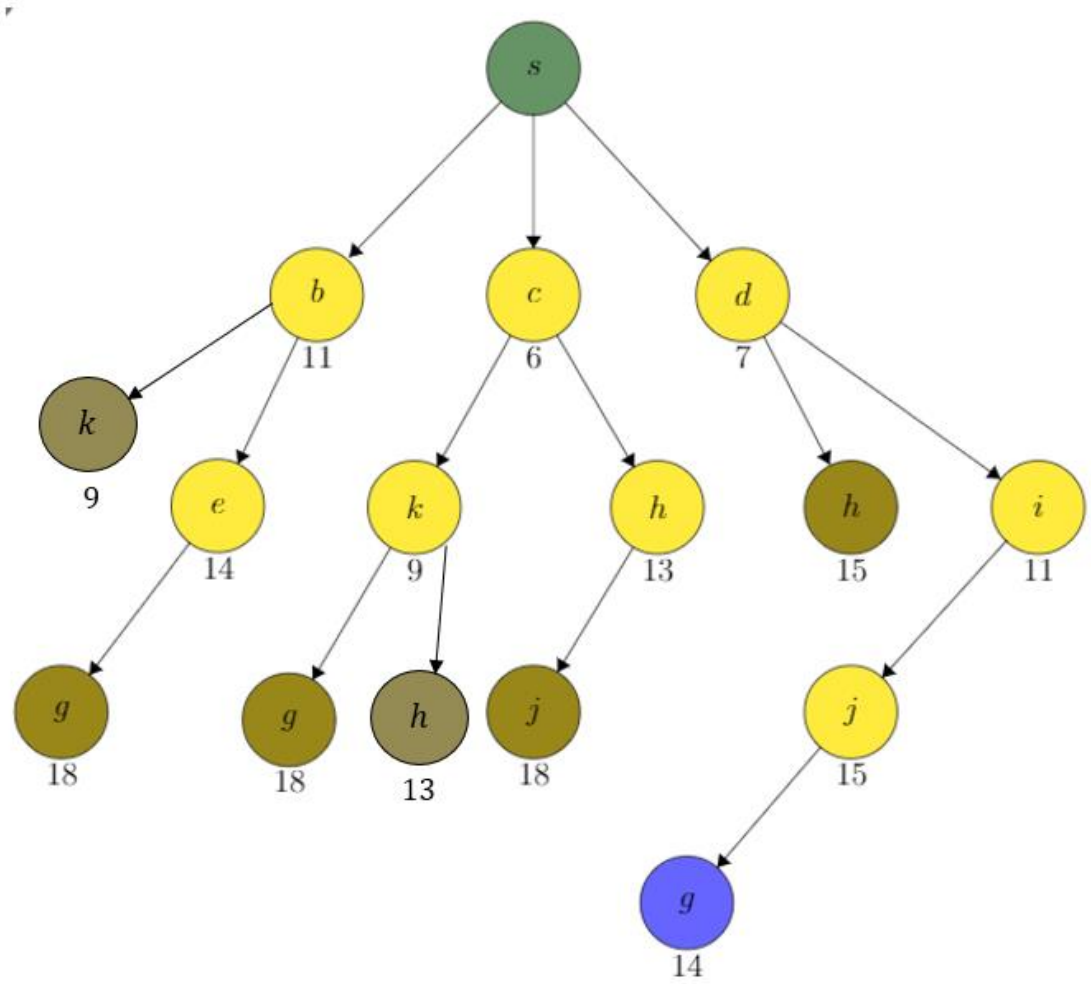
την τιμή 18. Τώρα ο «καλύτερος» κόμβος στο μέτωπο αναζήτησης είναι ο e, και θα επεκταθεί στο επόμενο βήμα



Βήμα 7ο: Επεκτείνω τον κόμβο h, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης f. Είναι $f(g) = 18 + 0 = 18$. Βρέθηκε λοιπόν άλλο ένα μονοπάτι μέχρι τον κόμβο, όμως ο αλγόριθμος θα συνεχίσει την εκτέλεσή του εφόσον υπάρχει άλλο μονοπάτι που δεν έχει φτάσει στο πέρας του και έχει μήκος ως τώρα μικρότερο από αυτό που έχει βρεθεί στα μονοπάτια μέχρι το στόχο. Ο "καλύτερος κόμβος" είναι τώρα ο j, ο οποίος θα επεκταθεί στο επόμενο βήμα.



Βήμα 8ο: Επεκτείνω τον κόμβο h, και υπολογίζω για τα παιδιά του τις τιμές της ευριστικής συνάρτησης f. Είναι $f(g) = 14 + 0 = 14$. Βρέθηκε λοιπόν ένα μονοπάτι μέχρι τον κόμβο στόχο με μικρότερο μήκος από τα προηγούμενα μονοπάτια (Ουσιαστικά τώρα κλαδεύονται οι υπόλοιποι κόμβοι g με απόσταση 18). Επίσης τώρα δεν υπάρχουν άλλες διαδρομές που μπορούν να επεκταθούν, οπότε τελικά από τον αλγόριθμο A* επιστρέφεται το μονοπάτι $s \rightarrow d \rightarrow i \rightarrow j \rightarrow g$, με μήκος 14.



Το πρόβλημα έχει τις εξής πιθανές λύσεις:

Διαδρομή	**Κόστος**
$s \to d \to i \to j \to g$	14
$s \to c \to h \to j \to g$	17
$s \to c \to k \to h \to j \to g$	17
$s \to c \to k \to g$	18
$s \to b \to e \to g$	18
$s \to d \to h \to j \to g$	19
$s \to b \to k \to h \to j \to g$	17
$s \to b \to k \to g$	18

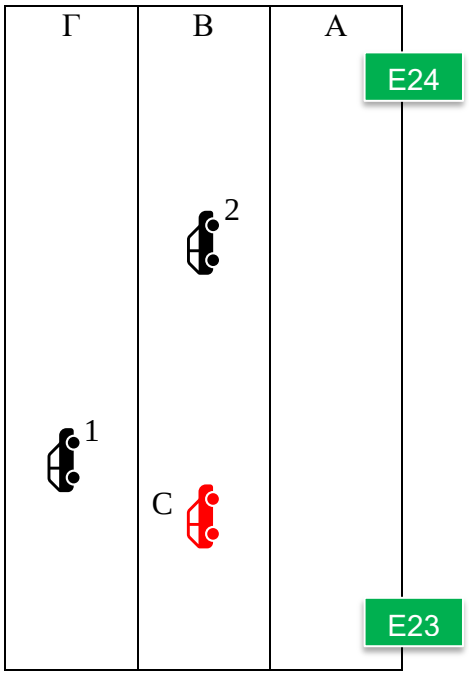
Παρατηρούμε λοιπόν ότι ο αλγόριθμος αναρρίχησης λόφων δε βρίσκει τη βέλτιστη λύση (αλλά τουλάχιστον επιστρέφει λύση). Ο αλγόριθμος A* από την άλλη βρίσκει τη βέλτιστη λύση, με κόστος 14. Ωστόσο, δε θα μπορούσαμε να είμαστε σίγουροι εκ των προτέρων για την εύρεση της βέλτιστης λύσης, εφόσον παρατηρούμε ότι η ευριστική απόσταση του κόμβου j είναι $h(j) = 3$, ενώ η πραγματική του απόσταση είναι 2, δηλαδή μεγαλύτερη της ευριστικής. Δηλαδή, ο ευριστικός μηχανισμός δεν είναι αποδεκτός (admissible).

Άσκηση 2

1. Για το σύστημα αυτόματης πλοήγησης σε αυτοκινητόδρομο έχουμε:

- Περιβάλλον: Δρόμος, λωρίδες, άλλα οχήματα, έξοδοι, εξωτερική κατάσταση (φωτεινότητα, καιρός)
- Αισθητήρες: Κάμερες, αισθητήρες υπερήχων, GPS, αισθητήρες λειτουργίας αυτοκινήτου, εγκέφαλος αυτοκινήτου
- Δράσεις: Γκάζι, φρένο, χειρισμός τιμονιού, χειρισμός φώτων, φλας
- Δείκτες Επίδοσης: Ασφάλεια, χρόνος, άνεση, τήρηση του ΚΟΚ, μεταφορά στον προορισμό, χειρισμός επειγόντων περιστατικών

2. Ιδού μία κατάσταση του κόσμου:



Έχουμε:

- Αντικείμενα: Αυτοκίνητο C (αυτόματης πλοήγησης), οχήματα 1,2, λωρίδες Α, Β, Γ, έξοδοι Ε24, Ε23, εξωτερική κατάσταση (καιρός, φωτεινότητα)
- Ιδιότητες: Λωρίδα Α ελεύθερη (μπορεί να γίνει αλλαγή), Β τρέχουσα, Γ πιασμένη – Ε24 έξοδος στόχος – καιρός αίθριος και φωτεινό περιβάλλον
- Σχέσεις: όχημα 1 αριστερά του C, όχημα 2 σε σχετικά κοντινή απόσταση μπροστά, Ε23 περασμένη, Ε24 επόμενη έξοδος

Κάποιοι τελεστές μετάβασης στον κόσμο του αυτοκινητόδρομου είναι:

- Άλλαξε λωρίδα προς τα αριστερά
- Άνοιξε δεξί φλας
- Άνοιξε τα φώτα διασταύρωσης
- Πάτα φρένο/γκάζι
- Πάρε την έξοδο στα δεξιά σου

3.Για την ευριστική συνάρτηση που θα χρησιμοποιηθεί για το κόστος της μετάβασης από μία κατάσταση σε μια άλλη (που θα αφορά την κατεύθυνση του αμαξιού – διατήρηση ή αλλαγή

στην αριστερή/δεξιά λωρίδα, εφόσον είναι δυνατή), θα πρέπει να ληφθεί υπόψη η ταχύτητα που εκτιμάται ότι μπορεί να αναπτυχθεί σε κάθε λωρίδα (ανάλογα με την ελεύθερη απόσταση που αντιλαμβάνονται οι αισθητήρες υπερήχων σε κάθε ελεύθερη λωρίδα), με μέγιστη δυνατή το όριο ταχύτητας (διαθέσιμο από το GPS). Πρέπει επίσης να λάβουμε υπόψη κάποιο κόστος για την αλλαγή λωρίδας (σταθερό ας θεωρήσουμε για την αλλαγή σε κάθε λωρίδα), καθώς και να συνυπολογίσουμε την απόσταση από την επιθυμητή έξοδο, εφόσον όσο πλησιάζουμε σε αυτή θα πρέπει να βρισκόμαστε σε όλο και δεξιότερη λωρίδα, προκειμένου στο τέλος να μπορούμε να πάρουμε την έξοδο. Μια τέτοια συνάρτηση λοιπόν για τη μετάβαση από την κατάσταση p στην κατάσταση q θα μπορούσε να είναι η

$$c(p,q) = K_1(\frac{d_q}{v_q} - \frac{d_p}{v_p}) + K_2 \frac{1 - right}{d_p(E_0 - E)} + c_0 \cdot change$$

Και εξηγώ:

- v_i είναι η εκτιμώμενη ταχύτητα στην κατάσταση i (η πραγματική στην τρέχουσα, η υπολογισμένη από τους αισθητήρες στις υποψήφιες επόμενες)
- d_i η απόσταση από την επιθυμητή έξοδο στον αυτοκινητόδρομο στην κατάσταση i
- $E_0 - E$ οι έξοδοι που απομένουν μέχρι την τελική (προκειμένου να παίζει ρόλο εκτός και από την απόσταση και το αν απομένουν πολλές έξοδοι)
- $right = 1$ αν η q αφορά την αλλαγή σε δεξιότερη λωρίδα, 0 αν αφορά τη διατήρηση λωρίδας, -1 αν αφορά την αλλαγή σε αριστερότερη λωρίδα
- K_j σταθερά προς ρύθμιση της βαρύτητας της σχετικής μεταβλητής στο κόστος
- $change = 1$ αν ή q αφορά αλλαγή κατεύθυνσης, 0 ειδάλλως
- c_0 σταθερά, σχετικά μικρή εφόσον η αλλαγή λωρίδας δεν κοστίζει πολύ χρονικά, όμως αν είναι παρόμοιες οι καταστάσεις γενικά θα ήταν προτιμητέα η διατήρηση της κατεύθυνσης.

Για την ευριστική συνάρτηση για την εκτίμηση της υπολειπόμενης απόστασης μέχρι την τελική κατάσταση θα λάβω υπόψη κυρίως την απόσταση μέχρι την έξοδο στον αυτοκινητόδρομο, την κίνηση που υπολογίζεται από την υπηρεσία GPS, και το πλήθος των λωρίδων μακριά από την δεξιότερη (που θα παίζει περισσότερο ρόλο πλησιάζοντας την έξοδο):

$$h(p) = T \cdot d_p + \Delta L \cdot (d_0 - d_p)$$

όπου:

- d_0 η αρχική απόσταση
- T μια παράμετρος που αφορά την κίνηση στον αυτοκινητόδρομο
- ΔL οι λωρίδες που το αμάξι απέχει από την δεξιότερη

4. Θα επιστρέψουμε στην κατάσταση του κόσμου που παρουσιάστηκε στο 2ο ερώτημα, όπως και στην περιγραφή της κατάστασης του κόσμου, η οποία γίνεται σε φυσική γλώσσα. Μπορούμε για αρχή να διατυπώσουμε αυτήν την περιγραφή σε συμβολική γλώσσα:

- current_lane(B)
- available_lanes(A)
- forward_near(2)
- at_left(1)

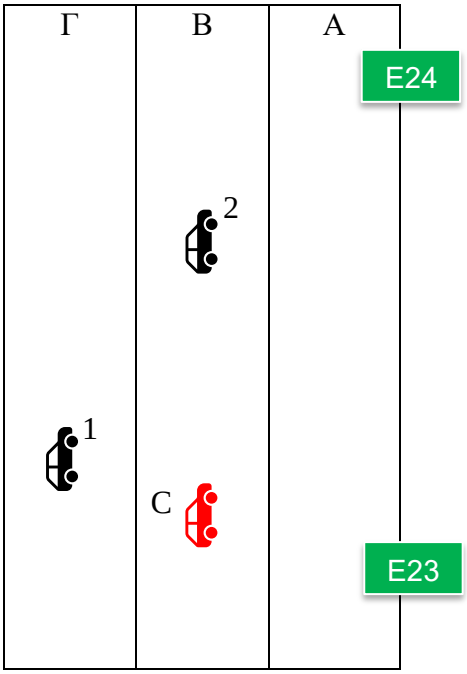
- exit_queue(E24)
- passed_exits(E23)

Αυτή η αναπαράσταση με Προτασιακή Λογική, θα μπορούσε να γίνει ως εξής:

$$C_B \mid \neg E_{24} \wedge E_{23} \mid \neg A_\Gamma \wedge A_A \mid L_1 \mid F_2$$

Όπου:

- Το κατηγόρημα C αφορά την τρέχουσα λωρίδα του αυτοκινήτου
- Το κατηγόρημα E αφορά το αν μία έξοδος έχει ήδη περαστεί
- Το κατηγόρημα A αφορά το αν μία λωρίδα είναι διαθέσιμη
- Το κατηγόρημα L αφορά το αν κάποιο όχημα βρίσκεται αριστερά του αμαξιού
- Το κατηγόρημα F αφορά το αν κάποιο όχημα βρίσκεται μπροστά του αμαξιού



Άσκηση 3

Έχω τα εξής δεδομένα που αφορούν τα μηνύματα ηλεκτρονικού ταχυδρομείου, που παρουσιάζονται στον κάτωθι πίνακα:

Σύνολο μηνυμάτων που	Είναι διαφημιστικά	Δεν είναι διαφημιστικά
Περιέχουν τη λέξη «ευκαιρία»	285	225
Δεν περιέχουν τη λέξη «ευκαιρία»	15	1275
Περιέχουν τη λέξη «μοναδικός»	240	600
Δεν περιέχουν τη λέξη «μοναδικός»	60	900

Έχουμε λοιπόν 1800 μηνύματα, τα οποία χωρίζονται σε δύο κλάσεις ανάλογα με το αν είναι διαφημιστικά (ορίζω ενδεχόμενο Δ) ή όχι (Δ'), και τα χαρακτηριστικά που μπορεί να έχουν είναι να περιέχουν τη λέξη «ευκαιρία» (E), ή να μην τη περιέχουν (E'), και να περιέχουν τη λέξη «μοναδικός» (M), ή να μην τη περιέχουν (M').

Υπολογίζουμε τις πιθανότητες:

$$P(\Delta) = \frac{300}{1800} = \frac{1}{6}, \qquad P(\Delta') = \frac{1500}{1800} = \frac{5}{6}$$

$$P(E|\Delta) = \frac{285}{300} = \frac{29}{30}, \qquad P(E'|\Delta) = \frac{15}{300} = \frac{1}{30}$$
$$P(E|\Delta') = \frac{225}{1500} = \frac{3}{20}, \qquad P(E'|\Delta') = \frac{1275}{300} = \frac{17}{20}$$
$$P(M|\Delta) = \frac{240}{300} = \frac{4}{5}, \qquad P(M'|\Delta) = \frac{60}{300} = \frac{1}{5}$$
$$P(M|\Delta') = \frac{600}{1500} = \frac{2}{5}, \qquad P(M|\Delta') = \frac{900}{1500} = \frac{3}{5}$$

Ο τύπος του Bayes για τον υπολογισμό της πιθανότητας ενός αντικειμένου με συγκεκριμένα χαρακτηριστικά x να ανήκει στη κλάση i είναι ο εξής:

$$P(i|x) = \frac{P(i)\prod_{k=1}^{p} P(x^{(k)}|i)}{\sum_{j=1}^{c} P(j) \cdot \prod_{k=1}^{p} P(x^{(k)}|j)}$$

Αυτός ο τύπος, μαζί με τις πιθανότητες που υπολογίστηκαν ανωτέρω, χρησιμοποιούνται προς τη λειτουργία του ταξινομητή Naive-Bayes ως εξής: όταν δίνεται ένα ηλεκτρονικό μήνυμα που περιέχει τα χαρακτηριστικά E ή E' και M ή M' υπολογίζεται η πιθανότητά του να ανήκει στη κλάση Δ και στη Δ' και ανάλογα με το ποια πιθανότητα είναι μεγαλύτερη ταξινομείται αντίστοιχα. Για παράδειγμα, για ένα μήνυμα που περιέχει τη λέξη «ευκαιρία» και τη λέξη «μοναδικός» έχουμε:

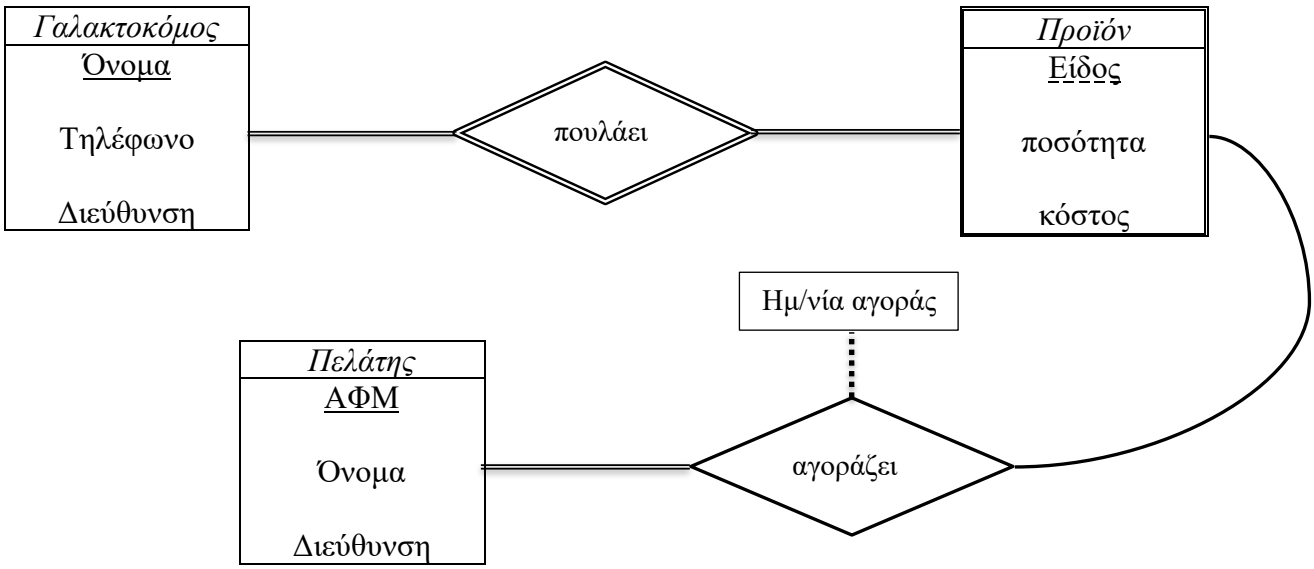
$$P(\Delta|E,M) = \frac{P(\Delta)\cdot P(E|\Delta)\cdot P(M|\Delta)}{P(\Delta)\cdot P(E|\Delta)\cdot P(M|\Delta) + P(\Delta')\cdot P(E|\Delta')\cdot P(M|\Delta')}$$
$$= \frac{\frac{1}{6}\times\frac{29}{30}\times\frac{4}{5}}{\frac{1}{6}\times\frac{29}{30}\times\frac{4}{5} + \frac{5}{6}\times\frac{3}{20}\times\frac{2}{5}} = \frac{\frac{116}{900}}{\frac{116}{900}+\frac{30}{600}} = \frac{116}{161} \approx 0.72$$

Με τον ίδιο τύπο, ή επειδή ισχύει ότι $P(\Delta'|E,M) = 1 - P(\Delta|E,M)$ αφού πρόκειται για συμπληρωματικά ενδεχόμενα, υπολογίζεται ότι $P(\Delta'|E,M) = 0.28$. Άρα μεγαλύτερη είναι η πιθανότητα $P(\Delta|E,M)$, και το μήνυμα ταξινομείται στην κατηγορία των διαφημιστικών.

Άσκηση 4

Για τη κατασκευή του διαγράμματος συσχετίσεων-οντοτήτων λαμβάνω υπόψη τα εξής:

- Όλοι οι γαλακτοκόμοι πωλούν κάποιο προϊόν, και όλα τα προϊόντα πωλούνται από κάποιον γαλακτοκόμο
- Τα χαρακτηριστικά των προϊόντων δεν αρκούν για τον προσδιορισμό τους, ενώ η ίδια η ύπαρξή τους εξαρτάται από την οντότητα των γαλακτοκόμων και τη μεταξύ τους σχέση, άρα θα αντιμετωπίσουμε τα προϊόντα ως weak entity.
- Δεν αγοράζονται απαραίτητα όλα τα προϊόντα, όμως όλοι οι πελάτες αγοράζουν κάποιο προϊόν


Γαλακτοκόμος
Όνομα
Τηλέφωνο
Διεύθυνση
πουλάει
Προϊόν
Είδος
ποσότητα
κόστος
Ημ/νία αγοράς
Πελάτης
ΑΦΜ
Όνομα
Διεύθυνση
αγοράζει