1. O expresie aritmetica complet parantezata este definita astfel:

0, 1, x, [e1+e2], [e1*e2], [-e2]

2 constructori nulari, 1 constructor extern, 3 constructori interni. Sa notam cu E acest tip de date.

Se definesc operatorii:

eval(e, n): E x N -> N
(E1) eval(0, n) = 0
(E2) eval(1, n) = 1
(E3) eval(x, n) = n
(E4) eval([e1+e2], n) = eval(e1, n) + eval(e2, n)
(E5) eval([e1*e2], n) = eval(e1, n) * eval(e2, n)
(E6) eval([-e1], n) = -eval(e1, n)

subst(e, f): E x E -> E
(S1) subst(0, f) = 0
(S2) subst(1, f) = 1
(S3) subst(x, f) = f
(S4) subst([e1+e2], f) = [subst(e1,f) + subst(e2, f)]
(S5) subst([e1*e2], f) = [subst(e1,f) * subst(e2, f)]
(S6) subst([-e], f) = [-subst(e, f)]

Sa se demonstreze prin inductie structurala ca pentru orice 2 expresii aritmetice *e*, *f* din *E* si *n* din *N*, proprietatea urmatoare este adevarata:

eval(subst(e, f), n)= eval(e, eval(f, n))

2. O formula bine formatata este definita recursiv de urmatoarele reguli:

- Baza: B = {p| p este o propozitie} U {T, F}
- Daca f este bine formatata atunci (¬f) este bine formatata
- Daca f1 si f2 sunt bine formatate, atunci (f1 ^ f2) este bine formatata
- Daca f1 si f2 sunt bine formatate, atunci (f1 v f2) este bine formatata

Spunem ca o formula este *„friendly”* [1] daca ea nu contine T sau F si negatia in formula este aplicata numai propozitiei si nu la o sub-formula mai complicata.

Demonstrati prin inductie structurala ca orice formula bine formatata este echivalenta logic cu o formula bine formatata *„friendly”*.

[1] Acesta nu este un termen standard, a fost introdus de catre noi

Exemplu:
formula bine formatata:

$$(\neg((p \wedge q) \vee \neg r)) \wedge (\neg(p \vee \neg r) \wedge q)$$

este logic echivalenta cu formula bine formatata *„friendly”* urmatoare:
$(((\neg p \vee \neg q) \wedge r)) \wedge ((\neg p \wedge r) \wedge q)$

3. Din cauza problemelor cu Tom, Jerry isi va face o casuta de vacanta afara. El vrea sa-si construiasca in copac o casuta din betisoare. In urma proiectarii, el decide ca modelul nu trebuie sa contina unghiuri drepte. Jerry strange N betisoare, fiecare betisor i avand lungimea xi. Pentru a se apuca de treaba, el trebuie sa afle daca poate construi un triunghi dreptunghic cu oricare 3 betisoare.

Gasiti un algoritm eficient pentru a determina daca exista 3 betisoare a, b, c astfel incat triunghiul format de ele sa fie dreptunghic. Demonstrati ca acest algoritm este corect pentru problema data folosind invarianti la ciclare.

4. a. Argumentati corectitudinea pentru algoritmul *Binomial-Heap-Union* (reuniunea a doua heap-uri binomiale intr-un singur heap binomial) utilizand invarianti la ciclare.

b. Se considera structura de date coada de prioritati minimala (min-priority queue) implementata folosind un heap binar, ce are asociata operatia *decrease-key(S, i, key)* – scade valoare cheii indicate de indicele i la key. Argumentati ca algorimul ce implementeaza aceasta operatie este corect folosind invarianti la ciclare.

La urmatorul link se pot gasi cei doi algoritmi:
http://www.cse.yorku.ca/~aaw/Sotirios/BinomialHeapAlgorithm.html

5. Se considera un contor binar pe k biti care implicit are doar operatia INCREMENT (incrementeaza valoarea sa plecand de la 0).

a. Presupunand ca am adaugat operatia DECREMENT la un contor binar (pe *k* biti), care este costul mediu pe operatie pentru n operatii succesive?

b. Presupunand ca am adauga operatia RESET (resetare la 0 - initializam toti bitii cu 0) la un contor binar (pe *k* biti), care este costul mediu pe operatie pentru n operatii succesive?

Proiectati o structura de date astfel incat pentru n operatii de INCREMENT si RESET sa obtineti timp in O(n).

6. Se considera o stiva implementata folosind un vector. Operatiile definite pentru stiva sunt push si pop. Totusi, cand se adauga un element in stiva (push) si vectorul este plin, trebuie sa se aloce un nou vector si sa se copieze elementele din vectorul vechi in cel nou, iar apoi se va folosi vectorul nou pe post de stiva.

a. Care este complexitatea operatiilor push si pop in cazul cel mai defavorabil? Justificati.

b. Daca dimensiunea vectorului se incrementeaza cu 1 atunci cand este necesar un vector mai mare, care este costul total pentru n operatii si care este costul mediu (amortizat) al unei operatii pe stiva, folosind metoda agregarii ?

c. Daca dimensiunea vectorului se dubleaza atunci cand este necesar un vector mai mare, care este costul total pentru n operatii si care este costul mediu (amortizat) al unei operatii pe stiva, folosind metoda agregarii ?

7. Considerand un min-heap binar, cu operatiile insert si delete_min, se cere:

a. Care este complexitatea celor doua operatii in cazul cel mai defavorabil? Justificati.

b. Definiti o functie potential pentru min-heap astfel incat costurile amortizate ale celor doua operatii sa fie cat mai mici (delete_min sa fie O(1)).

c. Demonstrati prin metoda potentialului care sunt costurile amortizate ale celor doua operatii.

8. O structura de date coada (first-in first-out) are operatiile dequeue(Q) si enqueue(Q, x) care extrage un element de la inceputul cozii, respectiv adauga un element la sfarsitul cozii. Aratati ca este posibil sa implementati aceasta structura de date folosind doua stive astfel incat pentru aceste doua operatii costurile amortizate sa fie constante.

9. Se considera un arbore de cautare obisnuit, care are atasata in plus, pentru fiecare nod x, valoarea dim[x] care memoreaza numarul de chei din subaorborele de radacina x. Fie α o constanta cu $1/2 <= \alpha < 1$. Un nod x se numeste α-echilibrat daca:

dim[stanga[x]] <= α* dim[x] si dim[dreapta[x]] <= α*dim[x]

Intregul arbore este α-echilibrat daca fiecare nod din arbore este α-echilibrat.

Presupunem ca operatiile INSEREAZA SI STERGERE sunt implementate in modul uzual pentru un arbore cautare binar avand n noduri, cu urmatoare exceptie: dupa fiecare astfel de operatie, daca toate nodurile din arbore nu mai sunt α-echilibrate, atunci subarborele avand radacina in nodul de acest tip, situat pe cel mai de sus nivel in arbore, este reconstituit astfel incat sa devina 1/2-echilibrat.

Vom analiza schema de reconstituire folosind metoda de potential. Pentru un nod x oarecare, din arborele de cautare binara T, definim:

$\Delta(x) = |$ dim[stanga[x]] - dim[dreapta[x]]$|$

si definim potentialul lui T prin :

$\Phi(T) = c^* \sum_{x \in T, \Delta(x) >= 2} \Delta(x)$, unde c este o constanta suficient de mare, care depinde de α

a. Fiind dat un nod x, intr-un arbore binar de cautare oarecare, aratati cum poate fi reconstituit subarborele de radacina x astfel incat sa devina 1/2-echilibrat. Algoritmul trebuie sa se incadreze in timpul Θ(dim[x]) si poate folosi o memorie auxiliara de ordinul O(dim[x]).

b. Aratati ca orice arbore de cautare binar are potentialul nenegativ si ca un arbore 1/2-echilibrat are potentialul 0

c. Sa presupunem ca m unitati de potential sunt suficiente pentru a plati reconstituirea unui subarbore avand m noduri. Cat de mare trebuie sa fie c, in functie de α, pentru ca timpul amortizat necesar reconstituirii unui subarbore care nu este α-echilibrat sa fie O(1)

d. Aratati ca, pentru un arbore α-echilibrat avand n noduri, inserarea si stergerea unui nod α-echilibrat necesita un timp amortizat de ordinul O(lg n)

Notare: Fiecare exercitiu valoreaza 1p => 9 * 1p + 1p (oficiu) = 10p