Universitatea Al.I.Cuza Iasi **Nume:**
Facultatea de Informatică **Grupa:**

Proiectarea algoritmilor – Test scris 08.04.2016, A

Observaţii:
1. Nu este permisă consultarea bibliografiei.
2. Toate întrebările sunt obligatorii.
3. Fiecare întrebare/item este notată cu un număr de puncte indicat în paranteză. Descrieți conceptele utilizate în răspunsuri.
4. Algorimii vor fi descrişi în limbajul Alk(cel utilizat la curs).
5. Nu este permisă utilizarea de foi suplimentare.
6. Timp de răspuns: 1 oră.

1. Se considera problema testării dacă o lista de numere întregi include un număr ce este divizibil cu 3 dar nu este divizibil cu 5, notată cu P35.

a) [1p] Să se formuleze problema P35 ca pereche (input, output). Se vor da formulări cât mai precise.

b) [2p] Să se descrie un algoritm determinist care rezolvă P35.

c) [2p] Să se arate ca algoritmul de la b) rezolvă corect problema descrisă la a) și să se precizeze timpul de execuție pentru cazul cel mai nefavorabil.

d) [2p] Să se descrie un algoritm nedeterminist care rezolvă P35.

e) [2p] Să se arate ca algoritmul de la d) rezolvă corect problema descrisă la a) și să se precizeze timpul de execuție pentru cazul cel mai nefavorabil.

Răspuns.

a) *Input*: $L = \langle x_0, x_1, \ldots, x_{n-1} \rangle$, $x_i \in Z$, i=1,...n

Output: true dacă $(\exists\ i)\ 0 \le i < n$ și 3 divide x_i (x_i % 3 == 0) și 5 nu divide x_i (x_i % 5 != 0)

false dacă $(\forall\ i)\ 0 \le i < n$ implică 3 nu divide x_i (x_i % 3 != 0) sau 5 divide x_i (x_i % 5 == 0)

b)
```
p35(L) {
  for (i = 0; i < L.size(); ++i)
    if (L.at(i) % 3 == 0 && L.at(i) % 5 != 0) return true;
  return false;
}
```
c) L.at(i) = x_i, , i=1,...n

Corectitudinea obținerii valorii true rezultă din semantica lui if.

Corectitudinea obținerii valorii false:

invariantul pentru for: pentru orice j cu $0 \le j < i$, L.at(j) % 3 != 0 sau L.at(j) % 5 == 0.

la terminarea lui for are loc invariantul și i == L.size(), care implica pentru orice j cu $0 \le j <$ L.size(), L.at(j) % 3 != 0 sau L.at(j) % 5== 0.

Dimensiunea unei instanțe: n = L.size().

Cazul cel mai nefavorabil: când returnează false.

Bucla for se execută de n ori în cazul cel mai nefavorabil, de unde rezultă T(n) = O(n).

d)
```
p35nedet(L) {
  choose i in 0..L.size() – 1;
  if (L.at(i) % 3 == 0 && L.at(i) % 5 != 0) success;
  else failure;
}
```
sau
```
p35nedet(L) {
  choose i in 0..L.size() – 1s.t. if (L.at(i) % 3 == 0 && L.at(i) % 5 != 0);
  return true;
}
```
e) Dacă $(\exists\ i)\ 0 \le i < n$ și 3 divide L.at(i) și 5 nu divide L.at(i), atunci există un calcul care se termină cu succes și întoarce *true*, cel în care instrucțiunea choose atribuie lui i o valoare care satisface condiția.

Deoarece alegerea lui i se face în timpul O(1) și oricare din ramurile lui if necesită timpul O(1), rezultă T(n) = O(1).

2. Se consideră linia poligonală simplă închisă L cu următoarele puncte în plan:

$$L[0]=(1,1), L[1]=(3,4), L[2]=(5,0), L[3]=(7,4), L[4]=(9,2), L[5]=(7,6), L[6]=(5,2), L[7]=(3,5)$$

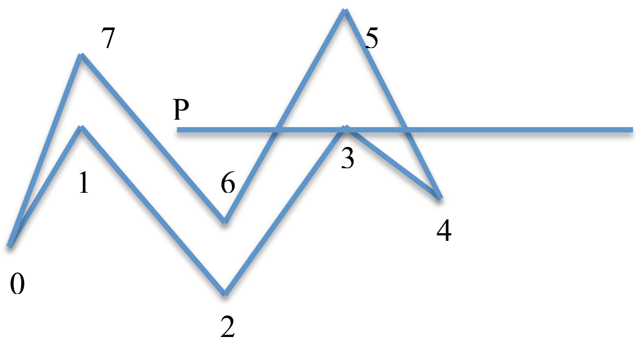
Scopul exercițiului este de a descrie cum se aplică algoritmul de localizare a unui punct față de o astfel de linie pentru $P = (4,4)$.

a) [5p] Descrieți cum se determină numărul de intersecții. Se va preciza cum se determină fiecare intersecție.

b) [2p] Descrieți invariantul menținut de algoritmul de numărare.

c) [2p] Descrieți cum se decide dacă P este interiorul sau exteriorul lui L.

Răspuns.



a) Fie d dreapta orizonatală care trece prin P.

lineIntersection(line(L[0], L[1]), d) = L[1] și L[1] < P.x => count() întoarce 0;

lineIntersection(line(L[1], L[2]), d) = L[1] și L[1] < P.x => count() întoarce 0;

lineIntersection(line(L[2], L[3]), d) = L[3] ; primul punct care nu e pe d este L[4]; L[2] și L[4] de aceeași parte a lui d => count() întoarce 0;

lineIntersection(line(L[4], L[5]), d) = Q și Q între L[4], L[5] => count() întoarce 1;

lineIntersection(line(L[5], L[6]), d) = Q și Q între L[4], L[5] => count() întoarce 1;

lineIntersection(line(L[6], L[7]), d) = Q și Q.x < P.x => count() întoarce 0;

lineIntersection(line(L[7], L[0]), d) = Q și Q.x < P.x => count() întoarce 0;

b)

```
  for (i = 0; i < L.size()-1; i = j)
    counter = counter + count(i, j);
```

`counter` = numărul de intersecții ale lui d cu `L[0..i]` care schimbă starea intermediara de interior sau exterior a lui P

c) dacă valoarea finală a lui counter este pară, P este exterior (aceeași stare cu punctele de la extremitatea semidreptei, care sunt exterioare);

dacă valoarea finală a lui counter este impară, P este intterior (stare diferită față de punctele de la extremitatea semidreptei, care sunt exterioare)

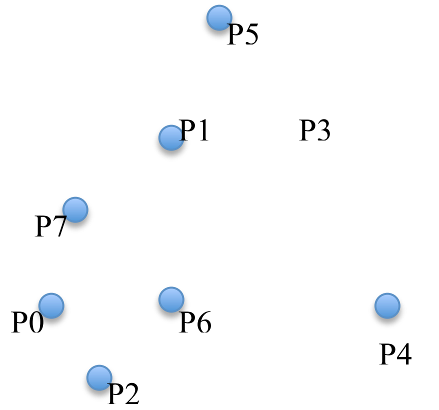
Pentru S avem counter = 2, rezultă că P este exterior.

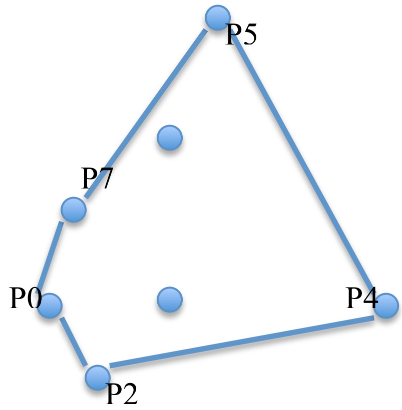
3. a) [2p] Descrieți problema determinării diametrului unei mulțimi.
Se consideră mulțimea

$$S = \{(1,1),(3,4),(2,0),(5,4),(6,2),(4,6),(3,1),(2,3)\}$$

b) [3p] Să se descrie cum se calculează punctele antipodale ale lui S.
c) [2p] Să se arate cum se calculează diametrul lui S.
d) [2p] Să se precizeze timpul de execuție pentru cazul cel mai nefavorabil a algoritmului aplicat la pașii precedenți și dacă există algoritmi mai performanți pentru a calcula diametrul unei mulțimi.

Răspuns.





a) *Input*: O mulțime finită de puncte în plan S.

Output: o pereche de puncte din S aflate la distanța cea mai mare.

b) se caculează înfășuratoarea convexă a lui S, CH(S)= ⟨P0, P2, P4, P5, P7⟩.

Se pleacă din P7P0.

- punctele antipodale cu P0: P4 (primul si singurul aflat la distanța cea mai mare față de P7P0)
- punctele antipodale cu P2: P5 (primul si singurul aflat la distanța cea mai mare față de P0P2)
- punctele antipodale cu P4: P0 (primul si singurul aflat la distanța cea mai mare față de P2P4)
- punctele antipodale cu P5: P0 (primul si singurul aflat la distanța cea mai mare față de P4P5)
- punctele antipodale cu P7: P4 (primul si singurul aflat la distanța cea mai mare față de P5P7)

Lista de perechi antipodale: AP(S) = ⟨(P0, P4), (P2, P5), (P4, P0), (P5, P2), (P7, P4)⟩

c) Se caută perechea aflată la distanța cea mai mare din AP(S): (P2, P5)

d) Dacă se utilizează pentru CH(S) algoritmul lui Jarvis, atunci complexitatea timp este $O(n^2)$. Dacă se utilizează scanarea Graham, atunci complexitatea timp este O(n log n). La curs nu s-au făacut algoritmi mai performanți de O(n log n) (de fapt nici nu există în general, dar rezultatul nu a fost demonstrat la curs).