Analiza Algoritmilor
Tema 1

Iulia Pașov, Tudor Berariu

6 noiembrie 2012

1 Problema 1: Asasinul

1.1 Descrierea problemei

Agentul 007 merge la un dineu la palatul Buckingham pentru a prinde un asasin periculos. El știe cu siguranță că la acest banchet sunt, în afară de el, n persoane. Una dintre acestea este, cu siguranță, asasinul pe care îl caută. Asasinul este un individ extrem de bine informat, cunoaște detalii despre fiecare invitat de la petrecere. Cu toate acestea, el nu este cunoscut de absolut nimeni dintre celelalte $n-1$ persoane. Agentul 007 trebuie să afle cine este asasinul și va discuta cu participanții încercând să afle cine pe cine cunoaște (punând întrebări de tipul: *X, îl cunoști pe Y ?*).

1.2 Cerințe

Scrieți un algoritm cât mai eficient prin care agentul 007 poate afla cine este asasinul de la banchet, doar punând întrebări de tipul *X, îl cunoști pe Y ?* la care i se răspunde cu *Da* sau *Nu* (`cunoaste(x,y)` $\in \{0, 1\}$). Se știe cu siguranță că asasinul se află la dineu, cunoaște pe toată lumea, însă nimeni nu îl cunoaște pe el.

Calculați complexitatea algoritmului propus pentru cazul cel mai defavorabil, numărând doar operațiile `cunoaste`. Atenție, apelurile funcției `cunoaste` trebuie să fie operații critice ale algoritmului. Se presupune că operația `cunoaste` se execută în $\mathcal{O}(1)$.

1.3 Observații

- Pseudocodul algoritmului și analiza de complexitate vor fi predate într-un document pdf: `nume_prenume_grupa_P1.pdf`.
- Problema valorează 2.5 puncte.
- Pentru punctaj maxim, algoritmul trebuie să folosească $\mathcal{O}(n)$ întrebări. Un algoritm corect, dar mai puțin eficient va primi minim 60% din punctaj.

2 Problema 2: Meteorologul

2.1 Descrierea problemei

Un meteorolog trebuie să instaleze niște echipamente la altitudinea h pe versantul unui munte. Acest versant reprezintă o arie pătrată și se prezintă sub forma unor platouri pătrate de dimensiuni egale, aliniate ca într-o matrice. Dacă nu există un punct cu exact altitudinea h, meteorologul nu va instala echipamentele. Creasta muntelui se află în extremitatea sud-estică a zonei de interes și se știe că orice platou din această arie are o altitudine mai mică atât decât platoul din estul lui, cât și decât platoul din sudul lui.

10	11	12	14	15	18
11	12	15	16	19	20
12	13	17	18	20	21
13	14	18	19	21	22
15	16	19	20	22	23
16	19	20	21	23	24

Figura 1: Exemplu de hartă a înălțimilor

Meteorologul nostru este parașutat deasupra oricărui platou dorește și are asupra lui echipamentele pe care trebuie să le instaleze și un altimetru care-i poate spune în orice moment la ce altitudine se află. Acesta se poate deplasa către unul dintre platourile de la nord, sud, est sau vest, fără a ieși însă din hartă. Ajutați-l să găsească platoul de lungime h.

2.2 Cerințe

Construiți un algoritm eficient care pentru harta altitudinilor unui versant dat sub forma unei matrice $A \in \mathcal{R}^{n \cdot n}$ de numere reale care are proprietatea că toate liniile și toate coloanele sunt sortate crescător (vezi Figura 1) și un număr real h, determină dacă există platou la altitudinea h în zona dată.

Matricea A nu va fi accesată direct, ci printr-o funcție
`float read_altitutde(int i, int j) { return A[i][j]; }`.
Atenție: două citiri consecutive ale altimetrului trebuie făcute din platouri învecinate.

Calculați complexitatea algoritmului propus pentru cazul cel mai defavorabil, considerând toate operațiile.

Scrieți un program C care implementează algoritmul propus.

2.3 Detalii de implementare

Programul va primi 2 argumente: numele fișierului de intrare și numele fișierului de ieșire. Fișierul de intrare va avea pe prima linie numărul real h, pe cea de-a doua linie numărul întreg n, iar pe liniile $3 \dots n+2$ liniile matricei A.

De exemplu, pentru $h = 21$, $n = 4$ și $A = \begin{bmatrix} 1 & 5 & 19 & 20 \\ 3 & 6 & 21 & 22 \\ 11 & 22 & 23 & 24 \\ 12 & 23 & 24 & 25 \end{bmatrix}$, fișierul de intrare va arăta așa:

```
5
4
1  5  19  29
3  6  21  22
11  22  23  24
12  23  24  25
```

Fișierul de ieșire va avea pe prima linie o singură valoare care va fi 0 dacă elementul h nu se află în A sau 1 dacă da. Pentru exemplul de mai sus, fișierul de ieșire va conține:

```
1
```

2.4 Observații

- Pseudocodul algoritmului și analiza de complexitate vor fi predate într-un document pdf: `nume_prenume_grupa_P2.pdf`.
- Problema valorează 2.5 puncte.
- Pentru punctaj maxim, algoritmul trebuie să fie în clasa de complexitate $\mathcal{O}(n)$. Pentru un algoritm corect, dar mai puțin eficient, se va obține minim 60% din punctajul maxim (în funcție de cât de ineficient este față de un algoritm în $\mathcal{O}(n)$).
- Codul se va scrie în fișierul `meteo.c`. Construiți un fișier `Makefile` care compilează programul în executabilul `meteo`.
- Verificați înainte de a trimite tema că secvența următoare funcționează corect.

  ```
  make
  ./cautare matrix1.in matrix2.out
  ```

  Pentru testarea automată se va folosi un script asemănător.

- Nu accesați direct matricea, ci scrieți o funcție ca cea dată mai sus. La testare o vom înlocui cu una care verifică și pozițiile din care se fac citirile.

3 Problema 3: Ocean's N

3.1 Descrierea problemei

Danny Ocean reunește încă o dată echipa pentru a patra lovitură. De data aceasta, ținta este o expoziție de artă ce contine n piese. Toate obiectele sunt protejate de seifuri speciale, diferite și foarte complexe. Din acest motiv, Ocean are nevoie de câte un expert pentru a sparge fiecare seif.

Toate obiectele sunt expuse în aceeași sală, dar ordinea lor se schimba periodic, asa ca nu se poate ști unde este asezat fiecare obiect decât dupa intrarea în sală. Sala are n uși, toate pe peretele opus celui cu seifurile și fiecare ușă se află în fața unui seif. Prin fiecare ușă poate trece un singur om, doar de doua ori, dupa care aceasta este blocată pentru o perioadă îndelungată de timp, deci doi oameni nu pot intra sau ieși din sală pe aceeași ușă. Mai mult, sala dispune de un sistem de securitate special: alarma se declanșează atunci cand orice perete sau podeaua, este atinsă o singură dată.

Mai mult, sala dispune de un sistem de securitate foarte bun: alarma se declanșează la orice mișcare. Pentru a putea ajunge la seifuri, hoții vor utiliza un sistem special care le permite să traverseze sala, cu condiția ca fiecare să meargă în linie dreaptă, iar drumurile lor să nu se intersecteze. De exemplu, în Figura 2, fie hoțul 1 (roz), fie hoții 2 și 3 (gri și rosu) pot accesa seifurile, dar nu toți trei în acelasi timp. Cum toate obiectele au aceeași valoare, Ocean dorește să colecteze cât mai multe dintre acestea, fără a declanșa nicio alarmă.

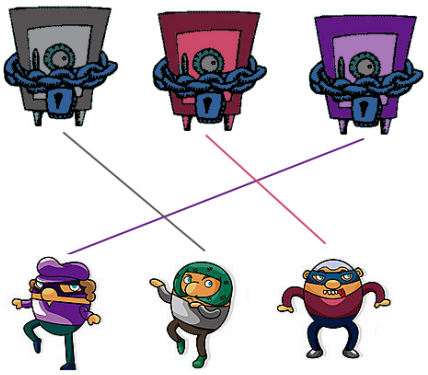

Figura 2: Ocean's n

3.2 Cerințe

Scrieți un algoritm cât mai eficient prin care Ocean poate alege membrii echipei care să acționeze, astfel încât să colecteze cât mai multe obiecte. Considerăm că membrii echipei au fost numerotați cu $\{1, 2, ..., n\}$ în funcție de ușa pe care

intră în sala cu seifurile. Odata ce intră, fiecare persoană îl va identifica pe cel pe care îl poate deschide. Cei care nu sunt selectați să acționeze, vor pregăti plecarea.

Motivați alegerea pentru algoritmul propus și calculați complexitatea pentru cazul cel mai defavorabil, considerând toate operațiile.

Scrieți un program C care să implementeze algoritmul propus.

3.3 Detalii de implementare

Programul va primi doua argumente: numele fișierului de intrare și numele fișierului de ieșire. Fișierul de intrare va conține pe prima linie numărul întreg n, iar pe urmatoarele n linii numere întregi. Astfel, linia i va conține identificatorul persoanei ce poate sparge seiful i. Un exemplu de fișier de intrare este:

```
5
2
4
1
5
3
```

Fișierul de ieșire va conține doar o linie cu identificatorii persoanelor alese să participe la jaf.

```
2 4 5
```

3.4 Observații

- O scurtă descriere a algoritmului și calculul de complexitate vor fi predate într-un document pdf: `nume_prenume_grupa_P3.pdf`.
- Problema valorează 2.5p.
- Codul se va scrie în fisierul`ocean.c`. Construiți un fișier `Makefile` care compilează programul în executabilul `ocean`. Verificați înainte de a trimite tema dacă urmatoarea secvență funcționează corect:

  ```
  ./ocean date.in date.out
  ```

  Pentru testarea automata se va folosi un script asemanator.

4 Problema 4: Arbori-Liste

4.1 Descrierea problemei

Pentru rezolvarea acestei probleme se vor folosi două structuri:

- arborele binar de căutare,
- lista dublu înlățuită circulară.

4.1.1 Arborele binar de căutare

Un arbore binar de căutare este o structură de date înlănțuită în care fiecare nod conține o *cheie* (valoarea nodului) și două câmpuri `mici`, `mari`, pentru cei 2 fii ai nodului curent. Arborele binar are proprietatea că pentru orice nod, cheia acestuia este mai mare decât toate cheile aflate în subarborele corespunzător câmpului `mici` și este mai mică decât toate cheile aflate în subarborele corespunzător câmpului `mari`.

Inserarea unui element într-un arbore de căutare se face astfel încât să se conserve proprietatea fundamentală a acestuia. Se pleacă de la rădăcină și se compară succesiv valoarea noului element cu cheile din arbore. Dacă valoarea este mai mică se va insera în subarborele `mici`, altfel se va insera în subarborele `mari`. Atunci când subarborele în care trebuie adăugat noul element este `NIL`, se creează un nou nod ce va avea cheia egală cu noua valoare.

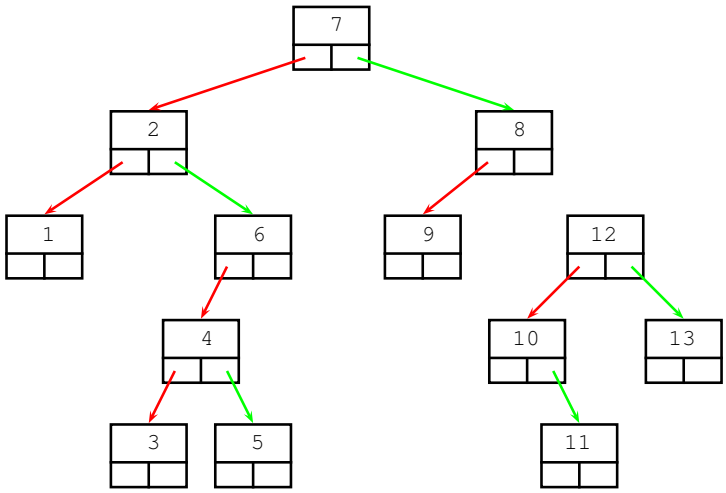

Figura 3: Arbore binar de căutare

Pentru mai multe detalii despre arborele binar de căutare citiți capitolul dedicat din [CSRL01].

4.1.2 Lista dublu înlănțuită circulară

O listă dublu înlănțuită este formată din obiecte ce au un câmp `cheie` și alte două câmpuri: `urmator` și `precedent`. Câmpul `urmator` indică către succesorul elementului curent, iar câmpul `precedent` indică către predecesorul elementului curent. într-o listă circulară, referința `precedent` a capului listei referă coada, iar referința `următor` a cozii liste referă capul ei. Lista dublu înșănțuit circulară poate fi privită ca un inel de elemente.

4.2 Cerințe

Scrieți un algoritm recursiv eficient care rearanjează legăturile interne ale unui arbore binar de căutare pentru a-l transforma într-o listă dublu înlănțuită circulară, care să păstreze ordinea elementelor. (vezi Figura 4)

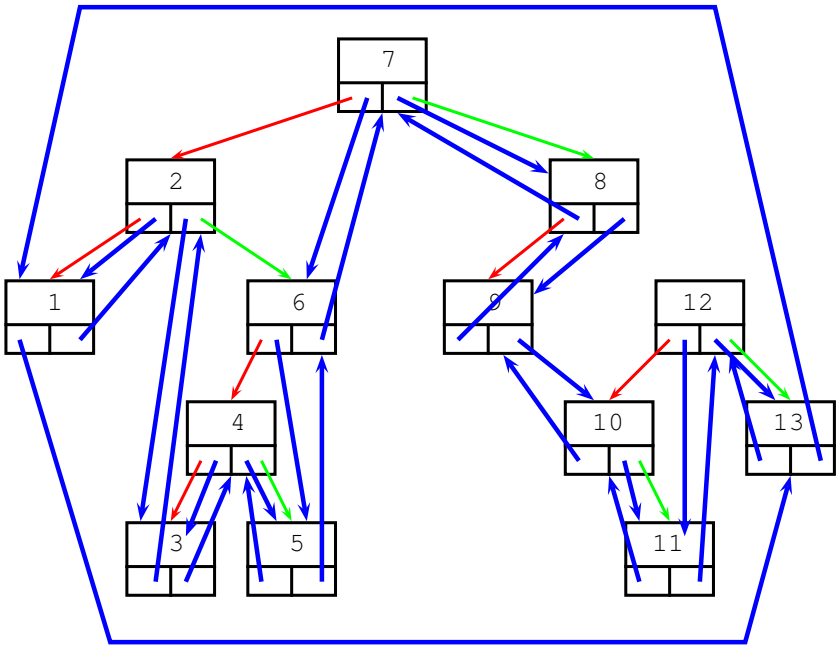

Figura 4: Cu albastru sunt legăturile ce trebuie create pentru a transforma arborele binar de căutare într-o listă dublu înlănțuită circulară, păstrând ordinea.

Scrieți un program C care:

1. Implementează arborele binar de căutare și operația de inserare pentru acesta.
2. Implementează o funcție `arbToList(arbore)` care primește un arbore binar de căutare și reface legăturile interne ale nodurilor pentru a-l transforma într-o listă dublu înlănțuită circulară.

4.3 Detalii de implementare

Programul va primi 2 argumente: numele fișierului de intrare și numele fișierului de ieșire. Fișierul de intrare va avea conține n linii cu numere întregi.

Programul va pleca de la un arbore binar de căutare gol și va insera pe rând elementele date în fișierul de intrare.

Un exemplu de fișier de intrare este:

```
7
8
2
12
13
9
6
4
3
10
5
1
11
```

Arborele din fișierul de intrare dat ca exemplu corespunde figurii 3. Fișierul de ieșire va conține o linie cu toate elementele listei.

```
1 2 3 4 5 6 7 8 9 10 11 12 13
```

(vezi figura 4).

4.4 Observații

- O scurtă descriere a algoritmului folosit pentru transformarea arborelui binar de căutare în listă dublu înlănțuită circulară va fi scrisă într-un document pdf: `nume_prenume_grupa_P4.pdf`.
- Problema valorează 2.5 puncte.
- Codul se va scrie în fișierul `arborelista.c`. Construiți un fișier `Makefile` care compilează programul în executabilul `arborelista`.
- Verificați înainte de a trimite tema că secvența următoare funcționează corect.

  ```
  make
  ./arborelista arb.in list.out
  ```

  Pentru testarea automată se va folosi un script asemănător.

5 Trimiterea temei

Construiți o arhivă cu următorul nume: `prenume_nume_grupa.zip` care conține:

```
nume\_prenume\_grupa.zip
| P1
| | nume\_prenume\_grupa\_P1.pdf
| P2
| | nume\_prenume\_grupa\_P2.pdf
| | meteo.c
| | Makefile
| P3
| | nume\_prenume\_grupa\_P3.pdf
| | ocean.c
| | Makefile
| P4
| | nume\_prenume\_grupa\_P4.pdf
| | arborelista.c
| | Makefile
```

Bibliografie

[CSRL01] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.