Structuri de Date și Algoritmi
Recapitulare

Mihai Nan

Departamentul de Calculatoare
Facultatea de Automatică și Calculatoare
Universitatea POLITEHNICA din București

Anul Universitar 2022–2023

Conținutul cursului

Structura examenului

- Examenul va conține **6 probleme**
- Fiecare problemă valorează **20 puncte**
- Pentru cele **5 puncte** din nota finală trebuie să acumulați **100 puncte**.
- Punctajul la examen nu se trunchiază. Astfel, puteți recupera din punctajul pierdut în timpul semestrului.
- Examenul va dura **2 ore**.
- Este examen de tipul **Closed Book**.

Materia examenului

Materia de la **curs** și **laborator** pentru capitolele

1. Colecții și mulțimi
2. Liste simplu și dublu înlănțuite
3. Stive și cozi
4. Tipuri particulare de liste
5. Arbori (toate tipurile studiate)
6. Heap binar
7. Grafuri (toate tipurile + algoritmi)
8. Tabele de dispersie
9. Algoritmi de sortare
10. Căutări în șiruri de caractere

Tipul exercițiilor de la examen

- Exercițiu de teorie aplicată (de prezentat avantaje / dezavantaje pentru folosirea unei structuri într-o anumită situație, de comparat două structuri de date, de oferit variante de implementare pentru o structură și realizată o comparație între acestea, de comparat doi algoritmi etc.).
- Exercițiu de prezentare a înțelegerii unui algoritm / unor operații pentru o structură de date (de aplicat un algoritm studiat pentru o intrare furnizată, de specificat cum arată o structură de date după aplicarea unor operații etc.).
- Exercițiu de implementare a unei funcții ce este foarte similară cu ceva implementat în cadrul laboratorului sau prezentat la curs (determinarea elementului de pe o anumită poziție dintr-o listă, ștergerea unei valori dintr-un arbore binar, schimbarea priorității unui nod într-un heap etc.).

Tipul exercițiilor de la examen

- Exercițiu pentru care trebuie propusă o implementare *eficientă* (trebuie identificată structura de date potrivită și propusă o implementare considerând structura deja implementată).
- Exercițiu la care este furnizată o implementare ce trebuie înțeleasă / analizată și apoi actualizată conform unor criterii sau optimizată. Sau furnizarea unui algoritm descris prin pașii acestuia care trebuie să fie implementat.
- Exercițiu general de implementare cu un grad mai ridicat de dificultate (nu va presupune o operație foarte similară cu cele implementate în laborator / curs).

Model examen – Problema 1

1. Realizarea unei comparații între două structuri de date care să urmărească planul de idei:
   1. Definiția structurilor de date
   2. Specificarea succintă a operațiilor de bază care există
   3. Metode de comparație
      - Enumeră și descrie cel puțin trei metode posibile de comparare între cele două structuri de date.
      - Argumentează avantajele și dezavantajele fiecărei metode.
   4. Evaluarea performanței
      - Compară eficiența celor două structuri de date în ceea ce privește operațiile specifice comparației.
      - Discută despre complexitatea timp și spațiu a fiecărei structuri de date în contextul comparației.
   5. Studiu de caz
      - Propune un scenariu sau o problemă specifică în care este necesară utilizarea celor două structuri de date.

Model examen – Problema 1

1. Alegem să realizăm comparația între **arbore binar de căutare** și **heap**.

Rezolvare

1. Definiția structurilor

Model examen – Problema 1

1. Alegem să realizăm comparația între **arbore binar de căutare** și **heap**.

Rezolvare

1. Definiția structurilor
   - **Arborele binar de căutare**: Este o structură de date în care fiecare nod are cel mult doi copii, iar elementele sunt organizate într-un mod specific, astfel încât fiecare nod din subarborele stâng conține valori mai mici decât nodul părinte, iar nodurile din subarborele drept conțin valori mai mari.
   - **Heap**: Este o structură de date arborescentă în care fiecare nod are o valoare asociată și are proprietatea că valoarea din fiecare nod este mai mare (în cazul unui heap maxim) sau mai mică (în cazul unui heap minim) decât valorile copiilor săi.

Model examen – Problema 1

1. Alegem să realizăm comparația între **arbore binar de căutare** și **heap**.

Rezolvare

2. Specificarea succintă a operațiilor de bază

Model examen – Problema 1

1. Alegem să realizăm comparația între **arbore binar de căutare** și **heap**.

Rezolvare

2. Specificarea succintă a operațiilor de bază
   - **Arborele binar de căutare**
     - *Inserare*: Adăugarea unui element în arbore în conformitate cu regulile de ordonare.
     - *Căutare*: Găsirea unui element în arbore și returnarea informațiilor asociate cu acesta.
     - *Ștergere*: Eliminarea unui element din arbore.
   - **Heap**
     - *Inserare*: Adăugarea unui element în heap în conformitate cu proprietățile de ordonare ale heap-ului.
     - *Extracția valorii maxime/minime*: Obținerea și eliminarea valorii maxime/minime din heap.

Model examen – Problema 1

1. Alegem să realizăm comparația între **arbore binar de căutare** și **heap**.

Rezolvare

3. Metode de comparație

Model examen – Problema 1

1. Alegem să realizăm comparația între **arbore binar de căutare** și **heap**.

Rezolvare

3. Evaluarea performanței

Model examen – Problema 1

1. Alegem să realizăm comparația între **arbore binar de căutare** și **heap**.

Rezolvare

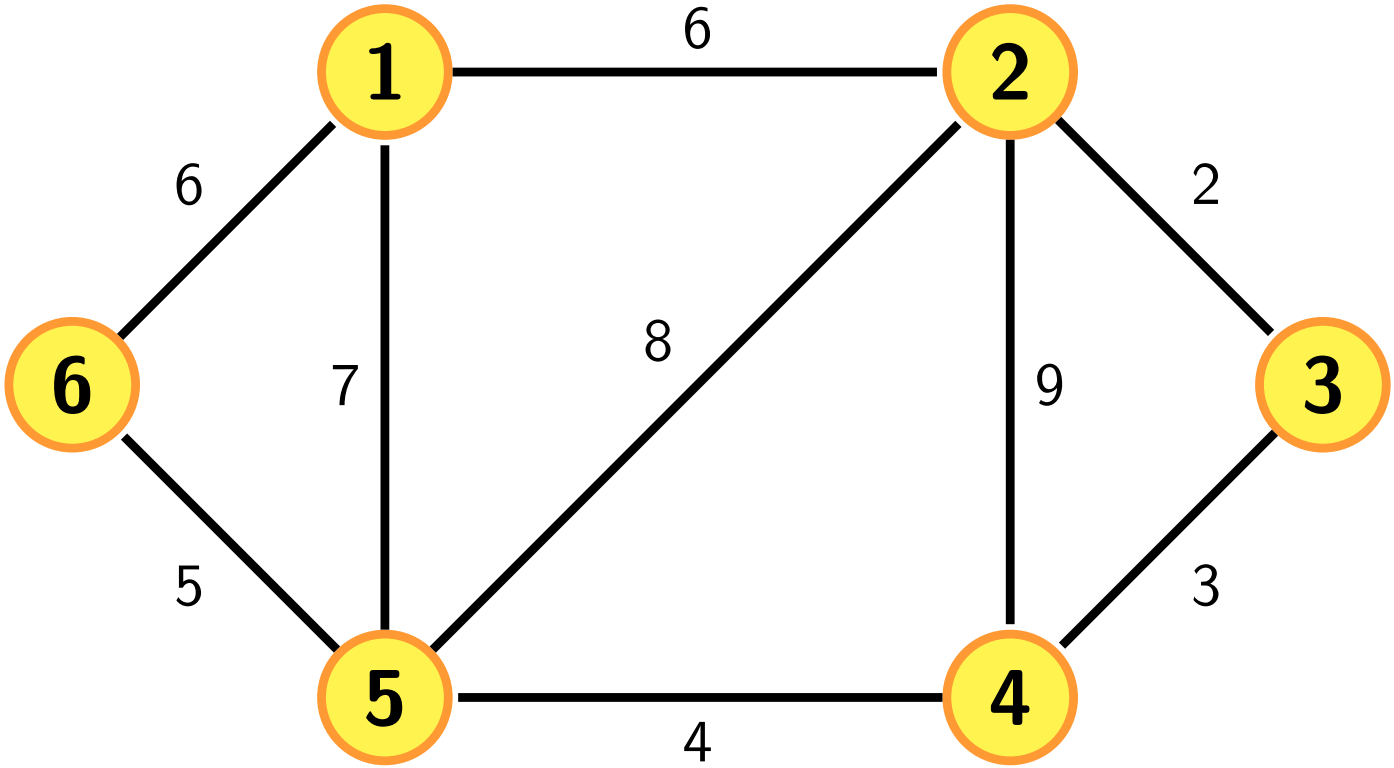
3. Studiu de caz

Model examen – Problema 1

1. Să presupunem că trebuie să dezvolți o aplicație de gestionare a unei biblioteci și trebuie să alegi între două structuri de date pentru a stoca cărțile din bibliotecă: un arbore binar de căutare și o listă simplu înlănțuită.
   Ce avantaje și dezavantaje ar putea apărea atunci când aplici fiecare dintre aceste două structuri de date pentru stocarea cărților din bibliotecă?

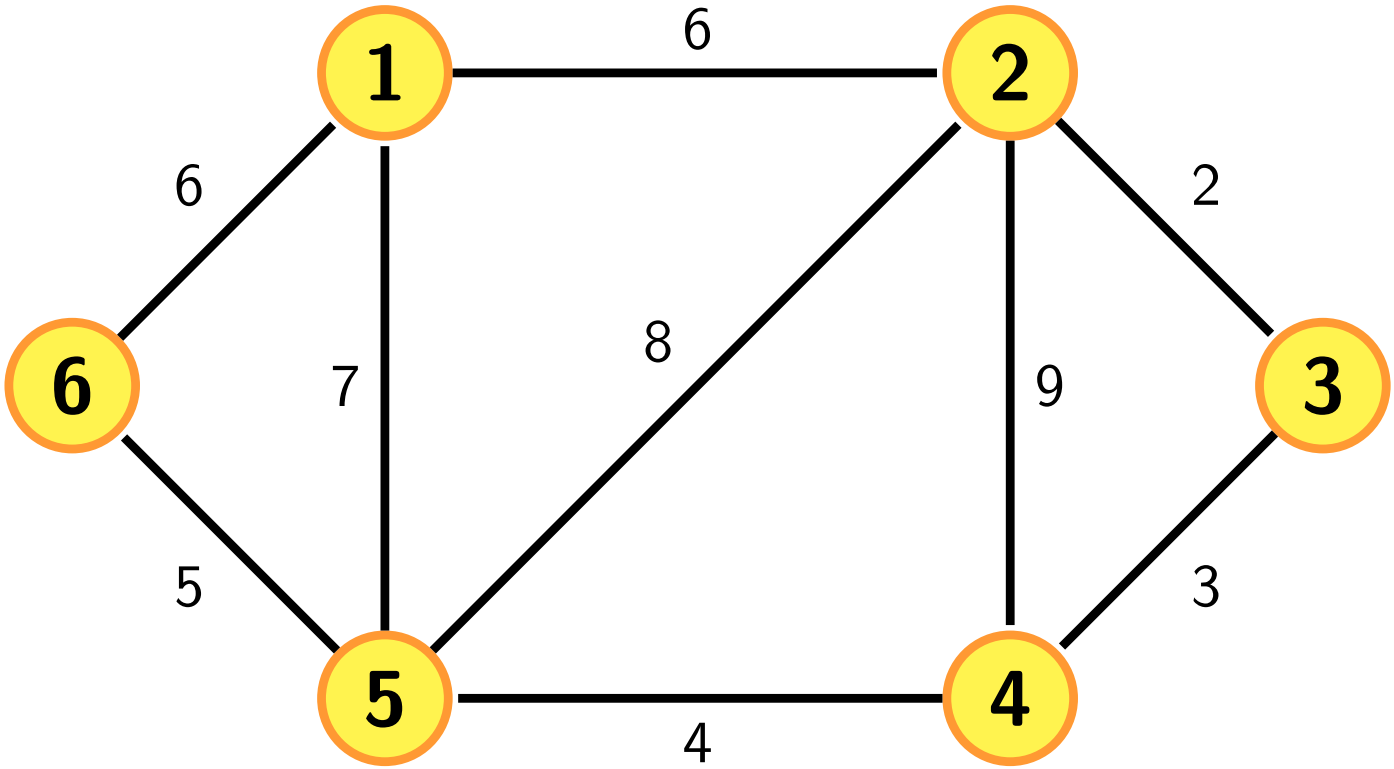
Model examen – Problema 2

2. Aplicați algoritmul lui Dijkstra, pas cu pas, pentru următorul graf, folosind ca nod de start nodul 1:



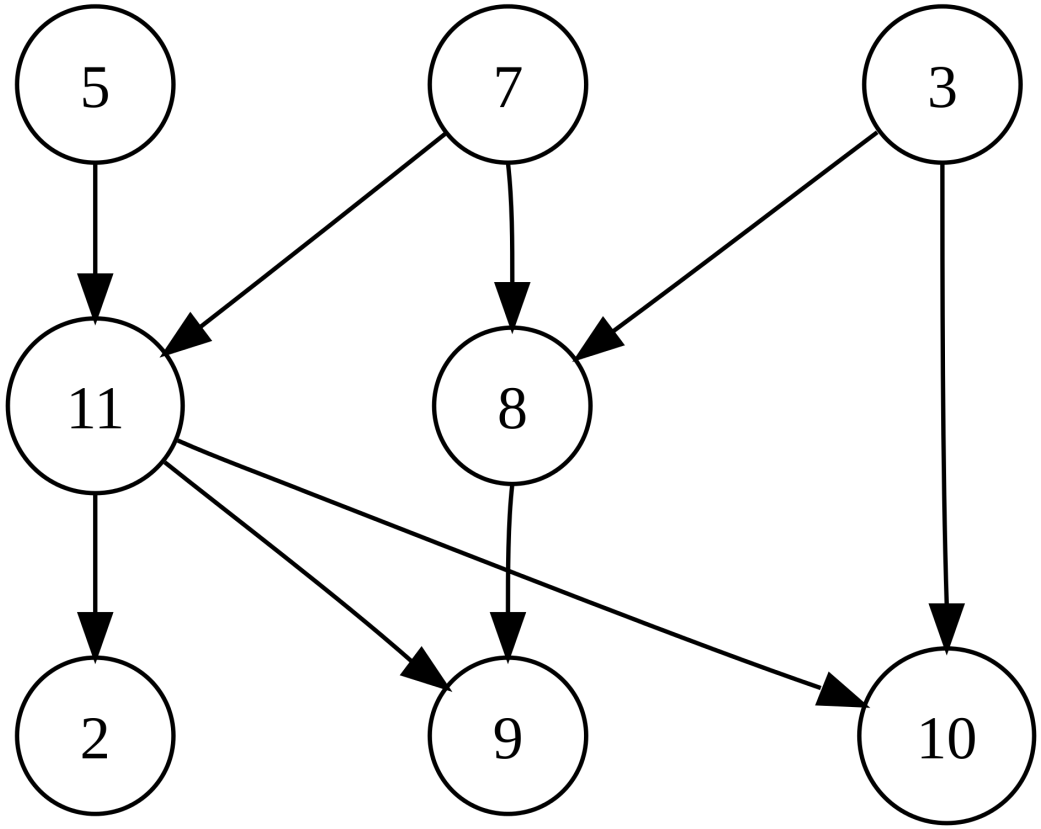
Model examen – Problema 2

2. Determinați arborele minim de acoperire pentru următorul graf, specificând ce algoritm alegeți să utilizați și evidențiind pas cu pas rularea algoritmului:



Model examen – Problema 2

2. Determinați sortarea topologică pentru următorul graf orientat (specificați metoda utilizată):



Model examen – Problema 2

2. Se dă o tabelă de dispersie de dimensiune $M = 7$ cu următoarea funcție de hash: $h(x) = x \bmod 10$. Știind că se inserează următoarele valori în tabela de dispersie: 4322, 1334, 1471, 9679, 1989, 6171, 6173, 4199 și că pentru a trata coliziunile se folosește înlănțuirea separată, indicați cum va arăta tabela de dispersie după aceste inserări. Descrieți sumar o altă modalitate de tratare a coliziunilor studiată.

Model examen – Problema 2

2. Care dintre următorii vectori poate reprezenta un heap? Desenați heap-ul sub forma unui arbore binar. Motivați răspunsul pentru fiecare exemplu în parte și specificați cum va arăta vectorul după inserarea valorii 23 dacă el reprezintă un heap.

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11
v1	1	12	23	10	15	38	45	15	18	20	21
v2	1	8	27	10	45	83	81	31	12	52	51
v3	1	13	20	21	65	54	67	41	30	83	52

Model examen – Problema 3

3. Implementați o funcție **recursivă** care primește ca argumente o listă simplu înlănțuită și un număr natural și determină numărul de apariții ale acelui element în listă.

Model examen – Problema 3

3. Implementați o funcție **recursivă** care primește ca argumente o listă simplu înlănțuită și un număr natural și determină numărul de apariții ale acelui element în listă.

```
int countOccurrences(List l, int x) {
    if (l == NULL)
        return 0;
    if (x == l->data)
        return 1 + countOccurences(l->next, x);
    rerturn countOccurences(l->next, x);
}
```

Model examen – Problema 3

3. Implementați o funcție **recursivă** (cu recusivitate pe coadă) care primește ca argumente o listă simplu înlănțuită și un număr natural și determină numărul de apariții ale acelui element în listă.

```
int countOccurrences(List l, int x, int acc) {
    if (l == NULL)
        return acc;
    if (x == l->data)
        return countOccurences(l->next, x, acc + 1);
    rerturn countOccurences(l->next, x, acc);
}
```

Model examen – Problema 3

3. Implementați o funcție care primește ca argumente o listă dublu înlănțuită, o valoare, un index și inserează valoarea la indexul specificat.

```
//antetul funcției
void insertNth(List *l, int value, int index);
```

Model examen – Problema 4

4. Gigel este pasionat de numere. El dorește să scrie numere din ce în ce mai mari și mai interesante. În ultima vreme, a descoperit un joc foarte fascinant pe care îl joacă împreună cu sora lui. Ea îi va da o secvență cu N operatori de comparație ($<$ sau $>$) și el va trebuie să concateneze numerele naturale din intervalul $[1, N+1]$ astfel încât ordinea în care concatenează numerele să respecte condiția impusă de sora lui. Primind o astfel de secvență, implementați o funcție care determină numărul rezultat. Argumentați alegerea structurii de date folosite.
Input: <<<< *Output:* 12345 (**Explicație:** $1 < 2 < 3 < 4 < 5$)
Input: >>>> *Output:* 54321 (**Explicație:** $5 > 4 > 3 > 2 > 1$)
Input: <<>><<>> *Output:* 125437698 (**Explicație:** $1 < 2 < 5 > 4 > 3 < 7 > 6 < 9 > 8$)

Model examen – Problema 4

```
1  void interestingNumbers(const char *string) {
2      int i, start = 1;
3      Stack s = NULL;
4      s = push(s, start++);
5      for (i = 0; i < strlen(string); i++) {
6          if (string[i] == '<') {
7              while (!isEmptyStack(s)) {
8                  printf("%d", top(s));
9                  s = pop(s);
10             }
11         }
12         s = push(s, start++);
13     }
14     while (!isEmptyStack(s)) {
15         printf("%d", top(s));
16         s = pop(s);
17     }
18     printf("\n");
19 }
```

Model examen – Problema 5

5. Fie următoarea funcție aplicată pentru un arbore binar definit prin:

```
typedef struct node {
    int value, index;
    struct node *left, *right;
}*Tree;

void functie(Tree root, int *index) {
    if (root == NULL)
        return;
    functie(root->left, index);
    root->index = *index;
    *index = *index + 1;
    functie(root->right, index);
}
```

Precizați ce realizează această funcție și propuneți o funcție echivalentă, dar în varianta **iterativă**.

Model examen – Problema 5

```
void inorder(Tree root) {
    Tree current = root;
    Stack s = NULL;
    int index = 0;
    while (current != NULL || !isEmptyStack(s)) {
        while (current != NULL) {
            s = push(s, current);
            current = current->left;
        }
        current = (Tree) top(s);
        s = pop(s);
        current->index = index;
        index++;
        current = current->right;
    }
}
```

Model examen – Problema 5

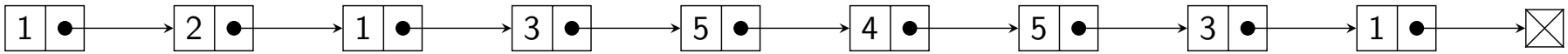
5. Algoritmul de parcurgere în lățime poate fi folosit pentru a determina dacă un graf este bipartit. Vom marca alternativ nodurile vizitate ale grafului astfel:
   - dacă primul nod din coadă are valoarea 1, atunci toate nodurile adiacente cu el și nevizitate vor fi marcate cu valoarea 2;
   - dacă primul nod din coadă are valoarea 2, atunci toate nodurile adiacente cu el și nevizitate vor fi marcate cu 1.

   La final, dacă între noduri marcate cu același număr există muchie, graful nu este bipartit. Pe baza algoritmului descris, implementați o funcție care determină dacă graful este bipartit.

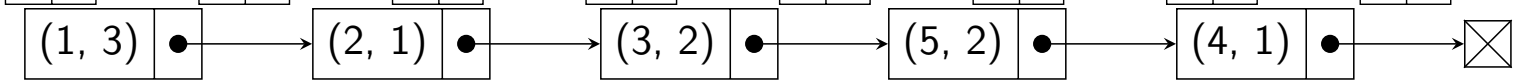
Model examen – Problema 6

6. Se dă o listă cu elemente având ca tip numere întregi. Scrieți o funcție care convertește această listă într-o listă de perechi de forma (element, număr de apariții).

Input:



Output:



Vă mulțumesc pentru atenție!