33,5

Név: Kanyó Ádám **Neptun:** YOMPVP

Adatszerkezetek és algoritmusok KF 1. ZH

Figyelem! A zárthelyi dolgozat megírására 100 perc áll rendelkezésre. A ZH írásához nem használható segédeszköz. A zh teljes pontszáma 110 pont, maximális eredmény 100 ponttal érhető el.

1) Elméleti kérdések (28p) 17,5p

1. Mit jelent a statikusan illetve a dinamikusan megvalósított adatszerkezet? (4p) 1p

Dinamikusan megvalósított adatszerkezetnél a dinamikus memóriakezelést használunk, azaz pointert és referenciát. Ezeket általában egy Node osztály segítségével. Statikusan megvalósított adatszerkezetnél, pedig simplán csak int-eket. Statikus: static_stack, static_queue

Dinamikus: dynamic_stack, dynamic_queue stb.

2. Írd le a **kiegyensúlyozott bináris kereső fa** adatszerkezet definícióját! (6p) 2,5p

Egy bináris keresőfában ~~[illegible]~~ van egy root (gyökérelem), ami egy Node. A Node egy olyan osztály aminek van értéke (key), bal/jobb eleme (left/right) és egy szülője (parent). Ezek mind mutatók. A kiegyensúlyozott bináris keresőfa, másnéven AVL fa egy olyan fa melynek magassága sosem több mint $1,44 \cdot (\log_2(N))$, ahol N egy normális bináris fa magassága. Az üres fa magassága definíció szerint 0. Az AVL fa forgatásokkal éri el, hogy kiegyensúlyozott legyen. Az alsó sorosok magasságának a különbsége sose haladja meg a 1-et.

3. Add meg a kivételkezelés nyelvi elemeit C++-ban és ismertesd a funkciójukat! (3p) 2p

throw - ezzel "dobunk" egy hibaüzenetet

catch - ezzel "kapjuk" el az adott hibaüzenetet amit a throw dob.

print - ezzel iratunk ki

4. Írd fel a piros-fekete fa szabályokat! (5p) 3p

A gyökér mindig fekete. Egy piros csúcsnak csak fekete gyereke lehet.
A benűrt csúcs mindig piros. A levelek mind feketék, ezek AVL
fában NULL pointerek voltak, de itt léteznek, viszont nincs értékük.

Magasság: $2 \cdot (\log_2(n-1))$? Fekete csúcsnak nem 2 piros gyereke?
csúcs piros v. fekete
csúcs → levél összes fekete csúcs azonos számban

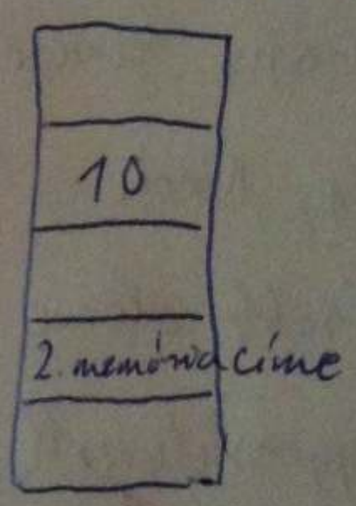
5. Hasonlítsd össze a pointer és a referencia alapú paraméterátadást! (4p) 0p

referencia: Ugyanarra memóriaterületre hivatkozik más névvel, mint a változó, amit referálunk. jele: &

pointer: Egy új memóriaterületet foglalunk le, viszont az értéke egy memóriacím lesz. jele: *

Pl.: int i = 10;
int *p; // pointer
p = &i;

P12: int i = 10;
~~int &r = i;~~
ahol &i az i memóriacíme



6. Mit jelent a statikus és a dinamikus típus egy változó esetén? Mi az a polimorfizmus? (6p) 3p

Statikus típusról beszélünk, amikor a változó típusát deklarációkor adjuk meg: int i = 10;

Dinamikus típusról beszélünk, hogyha létrehozunk egy változót egy adott osztállyal, viszont értéket úgy adunk neki, hogy az osztály leszármazottjával adjuk neki értéket.

Pl.: Auto a = Auto ("BMW"); // ahol a Taxi az Auto
a = Taxi ("Audi", 12); // leszármazottja

✓

2) Forráskód elemzés (12p) 1p

1. Mit ír ki az alábbi program a képernyőre? (5p) 1p

```
int& fv(int &p1, int *p2, int p3) {
        p1--;
        *p2 = p1 + p3;
        p3++;
        return p1;
}

int main() {
        int a = 8;
        int c = 3;
        int *b = &c;

        b = &(fv(a,b,c));
        (*b)--;

        std::cout << a << "," << *b << "," << c << std::endl;
        return 0;
}
```

// p1 értéke: 7
// c értéke: 10
// p3 értéke: 4

p2 pointer! C-re mutat

// b a 3-ra mutat (ami már 10

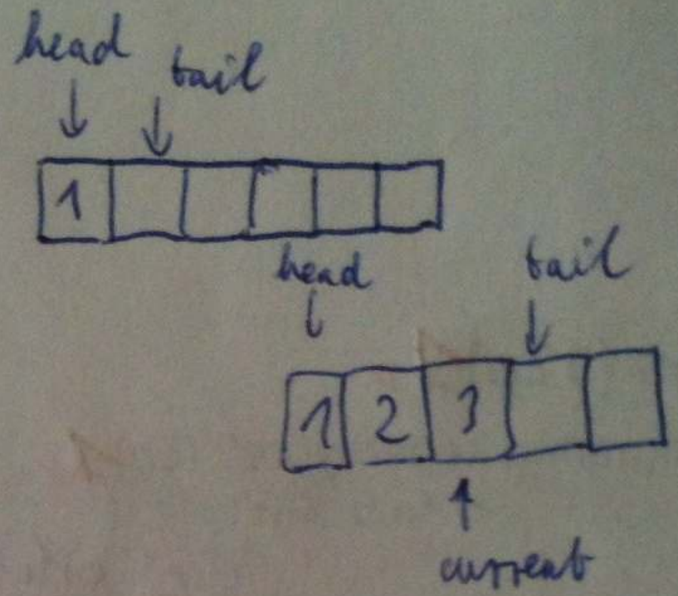
// megadja a b-nek a fv végeredményének a helyét

7, 6, 3

2. Milyen hibákat okoz a következő program? Hogyan lehetne javítani ezeket? (7p)

```
class resource_wrapper {
        int *resource_;
public:
        resource_wrapper(int init) {
                resource_ = new int(init);
        }
        ~resource_wrapper() {
                delete resource_;
        }
};

int main() {
        resource_wrapper a(10);
        resource_wrapper b(15);
        a = b;
        return 0;
}
```



3) Dinamikus láncolt lista (25p) 4p

1. Adott a következő kétszeresen láncolt lista:

```
template <class T>
class List<T> {
  private:
       struct Node {
              Node *prev;
              Node *next;
              T data;

              Node(Node *prev_, Node *next_, const T& data_) : prev(prev_),
                    next(next_), data(data_) { }
       };
       Node *head;
       Node *tail;
       Node *current;
};
```

Írd le a törlés adott (tetszőleges) pozícióból, **Remove(Node current)**, művelet kódját C++ vagy pszeudo nyelven! (Rajzold le magadnak az eseteket, ha nehezen megy!)

```
template <class T>
Remove :: List<T> (Node current) {

if (isEmpty)
    throw UnderflowException;
```

(munka lap)

6. 2.

~~postorder~~ inorder;

inorder ~~postorder~~ () { ← of(isEmpty) throw UnderflowException

while (! isEmpty) {

min (); // min keresés

Node * x = act; // aktuálisra állítom x-et

~~remove (x);~~ ~~// kitörlöm~~

cout << x -> key << " , " ;

remove (x); // kitörlöm

}

postorder; eljárás

postorder () { // ~~fa~~ eleje

if (isEmpty)

throw UnderflowException; ← Node * x = root;

~~while (! isEmpty) { while (x -> left~~

~~if~~ while (! isEmpty) { // 1. while eleje

while (x -> left != NULL && x -> right != NULL) { // 2. while eleje

if (x -> left != NULL) {

x = x -> left;

}

if (x -> right != NULL) {

x = x -> right;

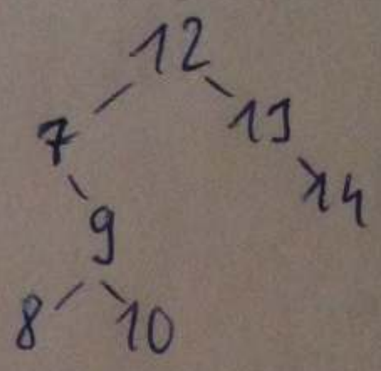
}

} ~~else~~ // 2. while vége

cout << x -> key << " , " ;

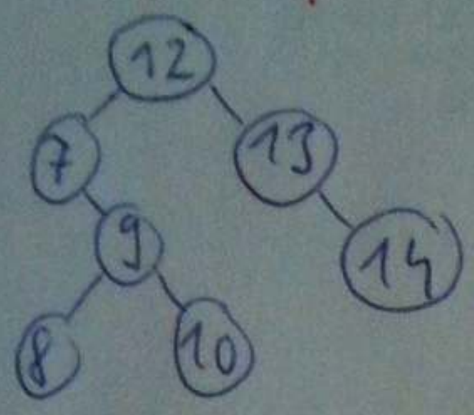
remove (x); // kitörlöm

} // 1. while vége



4) Bináris kereső fa (15p) 9p

1. Szúrd be egy bináris keresőfába az alábbi elemeket: 12, 7, 9, 8, 10, 13, 14. Rajzold le a keletkezett fát! (5p) 5p



2. Add meg a postorder és az inorder bejárás pszeudokódját bináris kereső fa esetén! (6p) 0p

postorder: 8, 10, 9, 7, 14, 13, 12 | inorder: 7, 8, 9, 10, 12, 13, 14

```
~~postorder~~() {
  ~~Node * x~~ root;
  while(!isEmpty()){
    man(); // minden
    Node * x = act;
    ~~cout~~ cout << x->key << " ";
  }
}
```

3. Írd le a globális maximumkeresés algoritmusát bináris keresőfában! (3p) 3p

```
max(){ if(isEmpty) throw UnderFlowException;
  Node * x = root;
  while(x->right != NULL){
    x = x->right;
  }
  return x->key;
}
```

4. Adj meg egy 6 elemű számsorozatot, melyet hagyományos módon beszúrva egy bináris keresőfába, a minimumkeresés művelete 6 csúcsot érint! (1p) 1p

6, 5, 4, 3, 2, 1

5) PF fa (30p) 8p

A következő már meglévő piros-fekete fába szúrd be lépésenként a 90 majd a 7 számot, ezt követően töröld a 20 számot. A forgatások esetén a forgatások irányait és helyét is ábrázold.