2. test DSA 11.5.2018 Jméno:

1. Příklad na rozptylování (hashing)

Uvažte rozptylovací tabulku rozměru **`M = 7`** a následující rozptylovací funkci **`h`**: **`h(k) = k mod M`**. Uvažte dále, že pro ukládání do tabulky používáte lineární prohledávání a odpovídající funkci je **`h1(k,i) = (h(k) + i) mod M`**, kde **`i`** je pořadí pokusu. Nakreslete obsah tabulky po vložení posloupnosti hodnot: 7, 5, 21, 10, 1, 14, 8.

Řešení:

Postup plnění tabulky lze zachytit následující řadou:

7						

Pokus č.1: ((7 mod 7) + 1) mod 7 = 1

7					5	

Pokus č.1: ((5 mod 7) + 1) mod 7 = 6

7	21				5	

Pokus č.1: ((21 mod 7) + 1) mod 7 = 1 – kolize
Pokus č.2: ((21 mod 7) + 2) mod 7 = 2

7	21		10		5	

Pokus č.1: ((10 mod 7) + 1) mod 7 = 4

7	21		10		5	

Pokus č.1: ((1 mod 7) + 1) mod 7 = 2 – kolize
Pokus č.2: ((1 mod 7) + 2) mod 7 = 3

7	21	1	10	14	5	

Pokus č.1: ((14 mod 7) + 1) mod 7 = 1 – kolize
Pokus č.2: ((14 mod 7) + 2) mod 7 = 2 – kolize
Pokus č.3: ((14 mod 7) + 3) mod 7 = 3 – kolize
Pokus č.4: ((14 mod 7) + 4) mod 7 = 4 – kolize
Pokus č.5: ((14 mod 7) + 5) mod 7 = 5

7	21	1	10	14	5	

Pokus č.1: ((14 mod 7) + 1) mod 7 = 1 – kolize
Pokus č.2: ((14 mod 7) + 2) mod 7 = 2 – kolize
Pokus č.3: ((14 mod 7) + 3) mod 7 = 3 – kolize
Pokus č.4: ((14 mod 7) + 4) mod 7 = 4 – kolize
Pokus č.5: ((14 mod 7) + 5) mod 7 = 5

7	21	1	10	14	5	8

Pokus č.1: ((8 mod 7) + 1) mod 7 = 2 – kolize
Pokus č.2: ((8 mod 7) + 2) mod 7 = 3 – kolize
Pokus č.3: ((8 mod 7) + 3) mod 7 = 4 – kolize
Pokus č.4: ((8 mod 7) + 4) mod 7 = 5 – kolize
Pokus č.5: ((8 mod 7) + 5) mod 7 = 6 – kolize
Pokus č.6: ((8 mod 7) + 6) mod 7 = 7

q.e.d.

2. Příklad na pravděpodobnost

Spočítejte střední hodnotu výstupu metody f pro dané n:

```
int f(int n) {
    int[] a = new int[n]; // pole inicializováno na samé nuly
    Random r = new Random();
    for (int i = 1; i < n; i++) {
        a[i] = r.nextInt(i + 1); // nextInt(x) generuje celá čísla 0, 1, ...,
                                 // x-1 včetně, rovnoměrné rozdělení
    }

    int count = 0;
    for (int i = 1; i < n; i++) {
        if (a[i - 1] < a[i]) {
            count++;
        }
    }
    return count;
}
```

Pomůcka (věta o linearitě střední hodnoty): $E[\sum_{i=1}^{n} Xi\ ] = \sum_{i=1}^{n} E[Xi]$

Řešení:

Náhodná veličina X je definována jako počet případů, kdy pro i ∈ {1, 2, ..., n-1} platí A[i-1] < A[i] (tj. počet případů, kdy je levý soused buňky ostře menší). Chceme vypočítat E[X]. Nadefinujeme indikátorové náhodné veličiny Xi = I{A[i-1] < A[i]}. Spočítáme střední hodnotu $E[Xi]$ = Pr{A[i-1] < A[i]}: Pr{A[i-1] < A[i]} = $\frac{\sum_{j=0}^{i} j}{i(i+1)}$ = 1/2, pro $E[X]$ pak platí (věta o linearitě střední hodnoty): $E[X] = E[\sum_{i=1}^{n-1} Xi\ ] = \sum_{i=1}^{n-1} E[Xi] = \frac{n-1}{2}$.

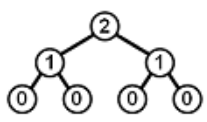
3. Příklad na rekurzi

Níže uvedená funkce **`ff`** je volána s parametrem **`n`**, tj: **`ff(n)`**. Odvoďte, kolikrát je pak volána funkce **`abc(x)`**.

```
void ff(int x) {
      if (x > 0) ff(x-1);
      abc(x);
      if (x > 0) ff(x-1);
}
```

Řešení:

Jedno volání funkce reprezentujeme uzlem ve stromu rekurzivního volání, hodnotou uzlu bude hodnota parametru funkce v daném volání. Kořen stromu bude mít hodnotu n, listy hodnotu 0. Protože při každém volání funkce **`ff`** se zavolá funkce **`abc()`** právě jednou, je počet volání funkce **`abc()`** roven právě počtu uzlů úplného binárního vyváženého stromu, tj. 2^n-1. Například pro n=2 je počet uzlů 7 – viz obrázek.



q.e.d.

4. Příklad na programování

Uvažte jednosměrně propojený spojový seznam, jehož definice je uvedena níže. Napište realizaci funkce reverse, která otočí pořadí prvků v seznamu a využije při to pouze konstantní paměť.

```
class Node {
    int key;
    Node nxt;

    public Node(int key, Node nxt) {
        this.key = key;
        this.nxt = nxt;
    }
}
class List {
    Node head;

    public List() { head = null; }
    public void print() {
        Node c = head;
        while (c != null) {
            System.out.print(c.key + " ");
            c = c.nxt;
        }
    }
    public void add(int key) {
        head = new Node(key, head);
    }
    public void reverse() { . . . }
}
```

Řešení:

```
class List {
    Node head;

    public List() { head = null; }

    public void print() {
        Node c = head;
        while (c != null) {
            System.out.print(c.key + " ");
            c = c.nxt;
        }
    }

    public void add(int key) {
        head = new Node(key, head);
    }

    public void reverse() {
        Node c = head;
        Node h = null;
        while(c != null) {
            Node n = c.nxt;
            c.nxt = h;
            h = c;
            c = n;
```

```
            }
            head = h;
        }
    }

    public class ReverseExample {
        public static void main(String[] args) {
            List lst = new List();
            lst.add(5);
            lst.add(4);
            lst.add(3);
            lst.add(2);
            lst.print();
            System.out.println();
            lst.reverse();
            lst.print();
        }
    }
```

q.e.d.

5. Příklad na datové struktury

Nadefinujte tzv. min-haldu. Formálně dokažte, že prvek v kořeni min-haldy je minimem všech vložených prvků. Počítejte s tím, že se v haldě může vyskytovat minimum vícekrát.

Řešení:

1) Definice viz učebnice, důležitá je vlastnost min-haldy: A[PARENT(i)] <= A[i].
2) Důkaz sporem: nechť se minimum vyskytuje v nekořenovém uzlu i. Pokud je v haldě minim více, bereme nejmenší i, pro které je min(A) = A[i], t.j. máme: A[1..i-1] > A[i]. Z toho ovšem plyne A[PARENT(i)] > A[i], což je ve sporu s vlastností haldy.

q.e.d.