Informatik II (D-ITET)

Tutorial 8

TA: Anwar Hithnawi, E-mail: hithnawi@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Outlook

- Exercise 7: Solution discussion

- Exercise 8: Overview (Binary Search, Backtracking, Reversi)
Generics

- **Parametrization**
  - We implemented: StackOfStrings.
  - We also want: StackOfURLs, StackOfInts, StackOfVans, ...

- **Attempt 1**: Implement a separate stack class for each type.
  - Rewriting code is tedious and error-prone.
  - Maintaining cut-and-pasted code is tedious and error-prone.

- **Attempt 2**: Implement a stack with items of type Object.
  - Casting is required in client.
  - Casting is error-prone: run-time error if types mismatch

```java
StackOfObjects s = new StackOfObjects();
Apple a = new Apple();
Orange b = new Orange();
s.push(a);
s.push(b);
a = (Apple) (s.pop());
```
Generics

- **Parametrization**
  - We implemented: StackOfStrings.
  - We also want: StackOfURLs, StackOfInts, StackOfVans, ... .

- **Attempt 1**: Implement a separate stack class for each type.
- **Attempt 2**: Implement a stack with items of type Object.
- **Attempt 3**: Java generics.
  - Avoid casting in client.
  - Discover type mismatch errors at compile-time instead of run-time.

```java
Stack<Apple> s = new Stack<Apple>();
Apple a = new Apple();
Orange b = new Orange();
s.push(a);
s.push(b);
a = s.pop();
```
Solution Ex7.Q1 – IFilter

FilterRaw

```java
public ArrayList filterRaw( ArrayList groups )
{
    ArrayList result = new ArrayList();
    for( int i = 0; i < groups.size(); i++ )
    {
        ArrayList group = (ArrayList) groups.get(i);
        for( int j = 0; j < group.size(); j++ )
        {
            Student student = (Student) group.get(j);
            if( filter( student ) )
                result.add( student );
        }
    }
    return result;
}
```
Lösung Ex7.Q1 – IFilter

FilterGeneric

```java
public ArrayList<Student> filterGeneric(ArrayList<ArrayList<Student>> groups) {
    ArrayList<Student> result = new ArrayList<Student>();
    for (int i = 0; i < groups.size(); i++) {
        ArrayList<Student> group = groups.elementAt(i);
        for (int j = 0; j < group.size(); j++) {
            Student student = group.elementAt(j);
            if (filter(student))
                result.add(student);
        }
    }
    return result;
}
```
Solution Ex7.Q1 – Tic-Tac-Toe

MAX strategy is demonstrated through a game tree, where MAX aims to maximize the outcome, and MIN aims to minimize it. The terminal nodes are marked with outcomes 0, 1, and -1, indicating the states of the game.
Solution Ex7.Q2 – IBinarySearchTreeUtils<T>

(a) Löschen: 15
(b) Löschen: 12
(c) Löschen: 20
(d) Resultierender Baum
Confusion because of Generics?

The type of T does not matter for us.

```java
public class TreeUtils<T> extends IBinarySearchTreeUtils<T>
{
    //...

    public int height(BinarySearchTree<T> tree)
    {
        if (tree == null) return 0;
        return 1 + Math.max(height(tree.left),height(tree.right));
    }

    //...
}
```
public boolean isLeaf(BinarySearchTree<T> tree) {
    return (tree.left == null && tree.right == null);
}

public boolean hasOneChild(BinarySearchTree<T> tree) {
    return (tree.left != null) ^ (tree.right != null);
}

/* Post-order: L-R-W */
private ArrayList<T> postOrder(BinarySearchTree<T> tree, ArrayList<T> arrayList) {
    if (tree == null) return arrayList;
    arrayList = postOrder(tree.left, arrayList);
    arrayList = postOrder(tree.right, arrayList);
    arrayList.add(tree.thing);
    return arrayList;
}

public ArrayList<T> postOrder(BinarySearchTree<T> tree) {
    return postOrder(tree, new ArrayList<T>());
}
Solution Ex7.Q2 – IBinarySearchTreeUtils<T>

```java
public BinarySearchTree<T> insert(BinarySearchTree<T> tree, int key, T thing)
{
    if (tree == null) {
        return new BinarySearchTree<T>(key, thing);
    }

    if (tree.key == key) {
        tree.thing = thing;
    } else if (key < tree.key) {
        tree.left = insert(tree.left, key, thing);
    } else {
        tree.right = insert(tree.right, key, thing);
    }
    return tree;
}

public T find(BinarySearchTree<T> tree, int key)
{
    if (tree == null) return null;
    if (key == tree.key) {
        return tree.thing;
    }
    if (key < tree.key) {
        return find(tree.left, key);
    }
    return find(tree.right, key);
}
```
public BinarySearchTree<T> remove(BinarySearchTree<T> tree, int key) {
    if (tree == null) return null;
    if (tree.key == key) {
        if (isLeaf(tree)) {
            return null;
        }
        if (hasOneChild(tree)) {
            if (tree.left != null) {
                return tree.left;
            }
            return tree.right;
        }
        BinarySearchTree<T> smallest = getSmallest(tree.right);
        smallest.left = tree.left;
        smallest.right = tree.right;
        return smallest;
    }
    if (key < tree.key) {
        tree.left = remove(tree.left, key);
    } else {
        tree.right = remove(tree.right, key);
    }
    return tree;
}
private Random rand = new Random();  //outside nextMove!!!

public Coordinates nextMove( GameBoard gb ){
    Coordinates coord = null;

    ArrayList<Coordinates> validMoves =
        new ArrayList<Coordinates>( gb.getSize() * gb.getSize() );

    System.out.print( "RandomPlayer" );
    for( int row = 1; row <= gb.getSize(); row++ ){
        for( int col = 1; col <= gb.getSize(); col++ ){
            coord = new Coordinates(row, col);
            if( gb.checkMove( color, coord ) )
                validMoves.add( coord );
        }
    }

    if( validMoves.isEmpty() ) return null;

    int randIndex = rand.nextInt( validMoves.size() );
    return validMoves.elementAt( randIndex );
}
Outlook

- Exercise 7: Solution discussion

- Exercise 8: Overview (Binary Search, Backtracking, Reversi)
Hints to Sheet 8

- Binary Search
- Knapsack problem and Backtracking
- Reversi (Part 2)
Hints to Ex8.Q1 – Binary Search

- Binary Search Algorithm (illustrated)

- Decision Tree

---

**Query:** 16  
**Sub-array length:** 16  
**Return:** null

---

**Query:** 23  
**Sub-array length:** 1  
**Return:** "23"
Hints to Ex8.Q1 – Binary Search

- Draw the decision tree and make some thoughts
  - Superposition, factors

- Implementation:
  - `find(List<Unit<Key, Value>> haystack, Key needle)`
  - `setFactor(int factor)`
    - Generalize the search \(\rightarrow\) unbalanced search trees
  - `getNumberOfCalls()`
    - Benchmarking with various factors
    - Average # of recursive calls to various factors
Ex8.Q2 – Knapsack problem and Backtracking

Which items to take?

Maximum Capacity
10kg
Hints Ex8.Q2

The general Knapsack problem

- k items $x_1, \ldots, x_k$ and each has known value and weight
- Choice of items, such that total weight is not exceeded
- Optimization problem: Maximize the value of the chosen items

a) Theory
b) Bruteforce approach
c) Backtracking approach
d) Comparison of Bruteforce und Backtracking approaches
Ex8.Q2 – Backtracking

- What does "Backtracking" mean?
  - Principle: "trial and error"

- Example: Looking for a maze exit
  - Decide upon a direction
  - Continue in this direction
  - If eventually unsuccessful
    - Return and choose another direction
  - If eventually successful
    - Done…

In case all directions were tried → keep going back.
Ex8.Q2 – Modelling Subset

- How to formalize an optimization task as a mathematical model

- How many different possibilities does our thief have?
  - S = Set of items at our disposal, each item \( i \in S \) characterized by weight \( (w_i) \) and its value \( (v_i) \)
  - The thief can only take a subset home
  - The thief can also choose the empty subset \( \phi \) (lazy thief) or the whole set \( S \) (strong thief with big bag)!

- \#items:= \#elements in the power set of \( S \)

- Example:
  - \( S = \{x_1, x_2\}, |S| = K = 2 \)
  - 4 Subsets: \( \phi, \{x_1\}, \{x_2\}, \{x_1, x_2\} \)
Ex8.Q2 – Modelling Subset

- **Decision variables**
  - $x_i$ denotes whether item $i$ is selected in the solution
    - $x_i = 1$ means the item is selected
    - $x_i = 0$ means that it is not selected

- **Problem constraint**
  - The selected item cannot exceed the capacity of the knapsack

- **Objective function**
  - Captures the total value of the selected items
    
    \[
    \begin{align*}
    \text{maximize} & \quad \sum_{i \in I} v_i x_i \\
    \text{subject to} & \quad \sum_{i \in I} w_i x_i \leq \text{Capacity} \\
    & \quad x_i \in \{0, 1\} \quad (i \in I)
    \end{align*}
    \]
Ex8.Q2 – A simple strategy for the thief

- How many possible configurations?
  - \((0,0,0,...,0), (0,0,0,...,1), ..., (1,1,1,...,1)\)

- Not all of them are feasible
  - They cannot exceed the capacity of the knapsack

- How many are they?
  - A set \(S\) with \(|S|=k\) contains \(2^k\) subsets
Ex8.Q2 – A simple strategy for the thief

- Apply the implementation in pseudo-code

1. Initialization
2. Take the next configuration (how exactly?)
3. Compute the overall weight
   if (overall weight < W)
     Compute overall value
     if (new overall value > overall value of current optimal solution)
       Current configuration becomes optimal solution
4. If more configurations remain ,
   Go to point 2
   else
     Computation done
Ex8.Q2.a – A simple strategy for the thief

- Does the thief’s strategy always yield the optimal results?
  - Yes/No
  - Why?...

- Is there always exactly one optimal solution?
  - Yes/No
  - Why?...
Ex8.Q2b,c – Bit value

- Configuration as a sequence of bits: `class Selection`
- The bit value describes the value of single bits, which depends on their position in the overall binary form of a number.

```
 7  0
1 0 0 1 0 1 1 0
```
Ex8.Q2b,c – Hints for the implementation

- **class Selection** is well documented
- Beware: If you increase the configuration (if you put a new item in the bag, A1c), the new status has to be updated
- Example of selections for $S$

$$M = \{x_1, x_2, x_3, x_4\}, \quad |M| = K = 4$$

$Teilmengen: \emptyset, \{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_1, x_2\}, \{x_1, x_3\},...$

$$(b_1, b_2, b_3, b_4) = (0, 0, 0, 0)$$

$$(b_1, b_2, b_3, b_4) = (1, 0, 1, 0)$$

$$(b_1, b_2, b_3, b_4) = (0, 1, 0, 0)$$

$$(b_1, b_2, b_3, b_4) = (1, 0, 0, 0)$$
Ex8.Q2b,c – Hints for the implementation

Bruteforce approach

```java
public Selection findBest(ArrayList<Integer> values,
                          ArrayList<Integer> weights,
                          int maxWeight)
{
    ...

    int last = java.Math.pow(2, values.size()); //Number of subsets
    for( int i = 0; i < last; i++ )
    {
        new Selection(values.size(), i); //Selection bit field with value i
        ...
    }
    ...
}
```
Ex8.Q2b,c – Hints for the implementation

- Backtracking approach:
  - FindResult Class (Selection and Value together)
  - Recursive method:
  
  ```java
  FindResult fr = find(currSelection, currWeight, values, weights, maxWeight);
  ```
  - End condition: `selection.size()==values.size();` //all things considered
  - In the method: two directions to continue
    - //Leave item
    
    ```java
    Selection without = new Selection(...); //Increase by one, set bit to 0
    //then continue down the tree
    ```
    - //Check whether weight is OK, then take item
      
      ```java
      ... 
      Selection with = new Selection(...); //Increase by one, set bit to 1
      //then continue down the tree
      ```
Hints to Ex8.Q3 – Reversi (Part 2)

HumanPlayer
nextMove()
Wait for input from The command line

RandomPlayer
nextMove()
Random selection (but valid move!)
Next move

GreedyPlayer
nextMove()
Select the next move using a simple, None-recursive Evaluation function

Download
Excercise 7
Excercise 8
Hints: Reversi (Part 2)

a. Implementing `ICheckMove` without Framework-Function.
   Ideas?
   Learn from GameBoard class. What methods available what could be useful?

b. Implement a player that selects the best move among all possible moves
   
   **Best move:** Move, after it's execution one owns max. more stones than the opponent: «Depth = 1»
   
   **Given:** Game Tree is not needed!

   Determining the best move: Copy Board (clone), run the move, count...
Hints Ex8.Q3a – checkMove()

```java
boolean checkMove(GameBoard gb, int player, Coordinates c) {
    //Check all directions

    //Unless at least one direction is valid
    //...

    //GameBoard.checkMove is not allowed to be used!
}
```
Hints Ex8.Q3a – greedyPlayer()

- Simple computer opponents
  - Move selection: Best first
  - Search depth: 1 (my move)
  - Evaluation Function: The difference of stone numbers after the move

- Tips
  - A GameBoard can be copied with gb.clone()
  - You are allowed to use GameBoard.checkMove() here
Have Fun!