Informatik II (D-ITET)

Tutorial 12

TA: Hông-Ân Cao, E-mail: hong-an.cao@inf.ethz.ch
Distributed Systems Group, ETH Zürich
Outlook

- **Exercise 11: Solution discussion**

- Exercise 12: Overview (HeapSort, Parallelized MergeSort, Recursive Problem Solving)
Solution Q11 – Time Complexity

- Bachmann-Landau notations
- Estimation by Analysis
  - Grows with order of magnitude

- $\mathcal{O}$-Notation
  - Upper bound
- $\Omega$-Notation
  - Lower bound

<table>
<thead>
<tr>
<th>Notation</th>
<th>Intuitive Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f \in \mathcal{O}(g)$</td>
<td>$f$ does not grow faster than $g$</td>
</tr>
<tr>
<td>$f \in \omega(g)$</td>
<td>$f$ grows faster than $g$</td>
</tr>
<tr>
<td>$f \in \Omega(g)$</td>
<td>$f$ does not grow much slower than $g$</td>
</tr>
<tr>
<td>$f \in \Theta(g)$</td>
<td>$f$ grows exactly as quickly as $g$</td>
</tr>
</tbody>
</table>

Solution Ex11.Q1 – Sorting by Search Trees

- Inserting all numbers and then reading in-order
  - In the best case, the values in the list are well-mixed → balanced tree
    \[ \mathcal{O}(\log(n)) \]
  - In the worst case, the values in the list are sorted in ascending or descending order → degenerate tree
    \[ \mathcal{O}(n) \]

- Sorting complexity
  - In best case: \[ O(n \cdot \log n) \]
  - In average case: \[ O(n \cdot \log n) \]
  - In worst case: \[ O(n^2) \]
Solution Ex11.Q2 – Complexity Analysis

// Fragment 1
for (int i = 0; i < n; i++)
    a++;

// Fragment 2
for (int i = 0; i < 2*n; i++)
    a++;
for (int j = 0; j < n; j++)
    a++;

// Fragment 3
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        a++;

- **Computation:**
  - `a++` is executed \( n \) times
  - \( \rightarrow \) Total executions: \( n \sim \mathcal{O}(n) \)

- **Computation:**
  - `a++` is executed \( 2n \) times
  - `a++` is executed \( n \) times
  - \( \rightarrow \) Total executions:
    - \( 2n + n = 3n \sim \mathcal{O}(n) \)

- **Computation:**
  - Outer loop is executed \( n \) times
  - Inner loop executes `a++` \( n \) times
  - \( \rightarrow \) Total executions:
    - \( n \times n = n^2 \sim \mathcal{O}(n^2) \)
Solution Ex11.Q2 – Complexity Analysis

// Fragment 4
for (int i = 0; i < n; i++)
    for (int j = 0; j < i; j++)
        a++;

- Computation:
  - Outer loop is executed $n$ times
  - Inner loop executes $a++$ $i$ times

→ Total executions:
  - $i = 1 + 2 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \sim O(n^2)$
Solution Ex11.Q2 – Complexity Analysis

// Fragment 5
while (n >= 1)
    n = n / 2;

- Computation:
  - while-loop executes $\frac{n}{2}$ times $n = n/2$ until $\frac{n}{2 \cdot 2 \cdot \cdots \cdot 2} = 1 \Rightarrow n = 2^x$
  - → Total executions:
    - $x = \log_2 n \sim 0(\log_2 n)$
Solution Ex11.Q2 – Complexity Analysis

// Fragment 6
for (int i = 0; i < n; i++)
    for (int j = 0; j < n*n; j++)
        for (int k = 0; k < j; k++)
            a++;

- Computation:
  - Outer loop is executed \( n \) times
  - Following loop is executed \( n^2 \) times
  - Inner loop executes \( a++ \) \( n \) times
  - \( \rightarrow \) Total executions:
    - \( n \times j = n \times (1 + 2 + \cdots + n^2) = n \times \sum_{j=1}^{n^2} j = n \times \frac{n^2(n^2+1)}{2} \sim O(n^5) \)
Hints Ex11.Q3 – Complexity (I)

\[ t_{op} = \frac{1}{3} t_{op} \]

- **Time per Operation**
- **Input Size**
- **Total run time**
### Solution Ex11.Q3 – Complexity (II)

<table>
<thead>
<tr>
<th>O(...)</th>
<th>$T_{tot}$</th>
<th>$T'_{tot}$</th>
<th>$T'<em>{tot} = T</em>{tot}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n)$</td>
<td>$T_{tot} = t_{op} \cdot M_1$</td>
<td>$T'<em>{tot} = t'</em>{op} \cdot M'_1$</td>
<td>$t'<em>{op} \cdot M'<em>1 = t</em>{op} \cdot M_1$ $\Rightarrow \frac{1}{3} t'</em>{op} \cdot M'<em>1 = t</em>{op} \cdot M_1$ $\Rightarrow M'_1 = 3M_1$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>$T_{tot} = t_{op} \cdot M_2^2$</td>
<td>$T'<em>{tot} = t'</em>{op} \cdot M'_2^2$</td>
<td>$t'<em>{op} \cdot M'<em>2^2 = t</em>{op} \cdot M_2^2$ $\Rightarrow \frac{1}{3} t'</em>{op} \cdot M'<em>2^2 = t</em>{op} \cdot M_2^2$ $\Rightarrow M'_2 = 3M_2$ $\Rightarrow M'_2 = \sqrt{3M_2}$</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>$T_{tot} = t_{op} \cdot 2^{M_3}$</td>
<td>$T'<em>{tot} = t'</em>{op} \cdot 2^{M'_3}$</td>
<td>$t'<em>{op} \cdot 2^{M'<em>3} = t</em>{op} \cdot 2^{M_3}$ $\Rightarrow \frac{1}{3} t'</em>{op} \cdot 2^{M'<em>3} = t</em>{op} \cdot 2^{M_3}$ $\Rightarrow 2^{M'_3} = 2^{M_3} \cdot 3$ $\Rightarrow M'_3 = M_3 + \log_2 3$ $\approx 1,7$</td>
</tr>
<tr>
<td>$O(\log_2 n)$</td>
<td>$T_{tot} = t_{op} \cdot \log_2 M_4$</td>
<td>$T'<em>{tot} = t'</em>{op} \cdot \log_2 M'_4$</td>
<td>$t'<em>{op} \cdot \log_2 M'<em>4 = t</em>{op} \cdot \log_2 M_4$ $\Rightarrow \frac{1}{3} t'</em>{op} \cdot \log_2 M'<em>4 = t</em>{op} \cdot \log_2 M_4$ $\Rightarrow \log_2 M'_4 = 3 \cdot \log_2 M_4$ $\Rightarrow M'_4 = 2^{3 \cdot \log_2 M_4}$ $\Rightarrow M'_4 = \left(2^{\log_2 M_4}\right)^3$ $\Rightarrow M'_4 = (M_4)^3$</td>
</tr>
</tbody>
</table>
Solution Ex11.Q4 – A knight on a chess board
Solution Ex11.Q4 – Numbers...

<table>
<thead>
<tr>
<th>Board</th>
<th>Number of knight’s tours</th>
<th>Number of closed knight’s tours</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x 4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5x5</td>
<td>1728</td>
<td>0</td>
</tr>
<tr>
<td>6x6</td>
<td>9862</td>
<td>$\geq 5$</td>
</tr>
<tr>
<td>8x8</td>
<td>?</td>
<td>$1.6 \cdot 10^{15}$</td>
</tr>
</tbody>
</table>

Source: [http://en.wikipedia.org/wiki/Knight%27s_tour](http://en.wikipedia.org/wiki/Knight%27s_tour)
Solution Ex11.Q4b – Backtracking

- Find a path...
  - which goes over all fields
  - and visits each field only once

- Early termination
  - There is no maximum depth: relatively simple
  - Backtracking when the next move already in the path

- Search efficiency
  - Code static values in a static manner
  - Linear search(\texttt{ArrayList}) replace with your own position-set-query
public class Knight implements IKnight {
    private ArrayList<Position> possibleMoves;
    public Knight();

    //Functions imposed by the interface
    public ArrayList<Position> getReachableSet(Position pos, int numberOfMoves) {};

    public ArrayList<Position> findCompletePath(Position pos) {};

    //Helper functions
    private boolean check(Position pos) {};

    private void visit(Position pos, int maxDepth, int depth, ArrayList<Position> visited) {};

    private boolean explore(Position pos, ArrayList<Position> path) {};
}
//>Loading all L-shaped moves for the knight
public Knight() {
    possibleMoves = new ArrayList<>(8);
    possibleMoves.add(new Position(1, 2));
    possibleMoves.add(new Position(2, 1));
    possibleMoves.add(new Position(2, -1));
    possibleMoves.add(new Position(1, -2));
    possibleMoves.add(new Position(-1, -2));
    possibleMoves.add(new Position(-2, -1));
    possibleMoves.add(new Position(-2, 1));
    possibleMoves.add(new Position(-1, 2));
}

//Still within the gameboard?
private boolean check(Position pos) {
    return pos.x >= 0 &&
           pos.x < IKnight.boardSize &&
           pos.y >= 0 &&
           pos.y < IKnight.boardSize;
}

public ArrayList<Position> getReachableSet(Position pos, int numberOfMoves) {
    ArrayList<Position> visited = new ArrayList<Position>();
    visit(pos, numberOfMoves, 0, visited);
    return visited;
}

private void visit(Position pos, int maxDepth, int depth, ArrayList<Position> visited) {
    if (!visited.contains(pos)) {
        visited.add(pos);
        if (depth == maxDepth) { return; }
    }
    for (Position possibleMove : possibleMoves) {
        Position newPos = pos.add(possibleMove);
        if (check(newPos)) {
            visit(newPos, maxDepth, depth + 1, visited);
        }
    }
}
public ArrayList<Position> findCompletePath(Position pos) {
    ArrayList<Position> path = new ArrayList<Position>();
    if (explore(pos, path)) {
        return path;
    } else {
        return null;
    }
}

private boolean explore(Position pos, ArrayList<Position> path) {
    if (path.contains(pos)) { return false; }
    path.add(pos);
    if (path.size() == IKnight.boardSize * IKnight.boardSize) { return true; }
    // Trying all possible moves from pos
    for (Position possibleMove : possibleMoves) {
        Position newPos = pos.add(possibleMove);
        if (check(newPos)) {
            if (explore(newPos, path)) {
                return true;
            }
        }
    }
    // Nothing worked → backtrack
    path.remove(path.size() - 1);
    return false;
}
Outlook

- Exercise 11: Solution discussion

- Exercise 12: Overview (HeapSort, Parallelized MergeSort, Recursive Problem Solving)
Hints Ex12.Q1 – Heap

A heap is a binary tree in which:

- All levels (except possibly the last) are completely filled
- The last level is filled from the left
- For all k nodes (except the root):
  - value (previous (k)) ≤ value (k) in a MIN-Heap
  - Or ≥ in a MAX-Heap

Properties (MIN-Heap):

- Root has the smallest value
- All paths from the root to a leaf are monotonically increasing
Hints Ex12.Q1 – Heap

Heap as tree

insert

Heap as Array
Hints Ex12.Q1a,b – Properties of Heaps

- How many elements are in a heap of height \( h \) containing minimum and maximum?

- Is a sorted array a heap (if it is interpreted as a binary tree)? And vice versa?
Hints Ex12.Q1c – HeapSort

Phase 1
Array converted to Heap

Phase 2
Read sorted Heap: remove from the root

6 5 3 1 8 7 2 4

Hints Ex12.Q1d – Implementation

2-phases

As in A1c
Take care of requirements for sort (copy!)
Note: all HeapSort operations are 'in-place'
Hints Ex12.Q2 – Parallelized Merge Sort

a) Much is up to you
   - u10a1.ISort you still (hopefully) have
   - ISort.sort: returns a sorted copy of the vector
   - Your MergeSort class should provide a way to select the number of parallel threads
   - Read: http://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html

b) 1'000'000 Integers
   - A main class to perform the measurements
   - Here also Ex10.Q1 offers a reference
   - An important indication of your measurements is the number of available CPU cores on your system (Google helps)
   - Don't forget the explanation!
Hints Ex12.Q3 – Recursive Problem Solving

- The company Springli intends to bring some new chocolate to the market.

- Acceptance of all combinations (that remain rectangular) of the basis rectangular piece with a maximum of n bits must be tested.

- How many combinations in terms of n must the company Springli test?

- Hint:
  - For n = 1, 2, 3, 4, 5, 6 exists 1, 3, 5, 8, 10, 14 formats.
Hints Ex12.Q3 – Springli Formats (I)

\[ n = 1 \rightarrow 1 \text{ Format} \]
Hints Ex12.Q3 – Springli Formats (II)

\[ n = 2 \rightarrow 3 \text{ Formats} \]
Hints Ex12.Q3 – Springli Formats (III)

\[ n = 3 \rightarrow 5 \text{ Formats} \]
Hints Ex12.Q3 – Springli Formats (IV)

\[ n = 4 \rightarrow 8 \text{ Formats} \]
Hints Ex12.Q3 – Springli Formats (V)

\[ n = 5 \rightarrow 10 \text{ Formats} \]
Hints Ex12.Q3 – Springli Formats

Recursive Solution:

\[ \text{Formats}(n) = \text{Formats}(n-1) + \ldots \]
That's all folks!

Source: [https://xkcd.com/557/](https://xkcd.com/557/)