Informatik II (D-ITET)

Tutorial 12

TA: Anwar Hithnawi, E-mail: hithnawi@inf.ethz.ch
Distributed Systems Group, ETH Zürich
L11.A1 – Sorting by Search Trees

1. Build a binary search tree out of the elements
2. Traverse the tree in-order and copy the elements back into the array

- In the best case, the values in the list are well-mixed → balanced tree
- In the worst case, the values in the list are sorted in ascending or descending order → degenerate tree

- Complexity
  - In best case: $O(n \cdot \log n)$
  - In average case: $O(n \cdot \log n)$
  - In worst case: $O(n^2)$
L11.A2 – Complexity Analysis

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

Computation:
• a++ is executed \( n \) times
→ Total executions: \( n \sim O(n) \)

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

Computation:
• a++ is executed \( 2n \) times
• a++ is executed \( n \) times
→ Total executions:
    \( 2n + n = 3n \sim O(n) \)
// Fragment 3
for (int i=0; i<n; i++)
    for (int j=0; j<n; j++)
        a++;

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

→ Total executions:

$$n \times n = n^2 \sim O(n^2)$$
```c
// 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

$$\text{Gesamtaufwand} = i = 1 + 2 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n-1)}{2} \sim O(n^2)$$
// Fragment 5
while (n >= 1)
    n = n/2;

Berechnung:  
- while-Schleife ruft $\frac{n}{2}$ Mal die Operation $n = n/2$ auf

\[
\underbrace{\frac{n}{2 \cdot 2 \cdots 2}}_{x \text{ mal}} = 1 \iff n = 2^x
\]

\[\Rightarrow \text{Gesamtaufwand} = x = \log_2(n) \sim O(\log_2(n))\]
// 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

\[
\begin{align*}
\text{Gesamtaufwand} &= n \cdot j = n \cdot (1 + 2 + \cdots + n^2) \\
&= n \cdot \sum_{j=1}^{n^2} j = n \cdot \frac{n^2(n^2 - 1)}{2} \\
&\sim O(n^5)
\end{align*}
\]
L11.A3 – Complexity (I)

\[ t'_{op} = \frac{1}{3} t_{op} \]

- Time per operation
- Input size
- Total running time
# L11.A3 – 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>$t'<em>{op} \cdot M'<em>1 = t</em>{op} \cdot M_1 \Rightarrow \frac{1}{3} t</em>{op} \cdot M'<em>1 = t</em>{op} \cdot M_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M'_1 = 3M_1$</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>$t'<em>{op} \cdot M'<em>2^2 = t</em>{op} \cdot M_2^2 \Rightarrow \frac{1}{3} t</em>{op} \cdot M'<em>2^2 = t</em>{op} \cdot M_2^2$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M'_2^2 = 3M_2^2 \Rightarrow M'_2 = \sqrt{3}M_2$</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>$t'<em>{op} \cdot 2^{M'<em>3} = t</em>{op} \cdot 2^{M_3} \Rightarrow \frac{1}{3} t</em>{op} \cdot 2^{M'<em>3} = t</em>{op} \cdot 2^{M_3}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$2^{M'_3} = 2^{M_3} \cdot 3 \Rightarrow M'_3 = M_3 + \log_2 3$</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>$t'_{op} \cdot \log_2 M'<em>4 = t</em>{op} \cdot \log_2 M_4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\frac{1}{3} t_{op} \cdot \log_2 M'<em>4 = t</em>{op} \cdot \log_2 M_4$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\log_2 M'_4 = 3 \log_2 M_4 \Rightarrow M'_4 = 2^{3 \log_2 M_4}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$M'_4 = (2^{\log_2 M_4})^3 \Rightarrow M'_4 = (M_4)^3$</td>
</tr>
</tbody>
</table>
L11.A4 – A knight on a chess board
L11.A4 – 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>≥ 5</td>
</tr>
<tr>
<td>8x8</td>
<td>?</td>
<td>1.6 \cdot 10^{15}</td>
</tr>
</tbody>
</table>

Check out: http://en.wikipedia.org/wiki/Knight%27s_tour
L11.A4b – 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 field already in the path

- Search efficiency
  - Code static values in a static manner
  - Linear search(ArrayList) replace with your own position-set-query
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);
        }
    }
}

private ArrayList<Position> possibleMoves;

Knight() {
    possibleMoves = new ArrayList<Position>(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));
}
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;
    }

    for (Position possibleMove: possibleMoves) {
        Position newPos = pos.add(possibleMove);
        if (check(newPos)) {
            if (explore(newPos, path)) {
                return true;
            }
        }
    }

    path.remove(path.size()-1);
    return false;
}
HINTS ON U12

A1 – Heapsort
A2 – Parallelized Mergesort (Threads)
A3 – Recursive Problem Solving
A4 – Master solution (philosophical question!)
U12.A1 – 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):
  - \( \text{value (previous } (k)) \leq \text{value } (k) \) in a MIN-Heap
  - Or \( \geq \) in a MAX-Heap

Properties (MIN-Heap):

- Root has the smallest value
- All paths from the root to a leaf are monotonically increasing
U12.A1 – Heap

Heap as tree

Heap as Array

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>6</th>
<th>10</th>
<th>12</th>
<th>13</th>
<th>11</th>
<th>17</th>
<th>18</th>
<th>16</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>
U12.A1a,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?
U12.A1c – HeapSort

Phase 1
Array converted to Heap

Phase 2
Read sorted Heap: remove from the root


2-phases

As in A1c
Take care of requirements for sort (copy!)
Note: all HeapSort operations are 'in-place'
U12.A2 - Parallel Mergesort

- Implement Mergesort using threads
  - Number of threads configurable
  - Create new thread whenever array is split into two
  - Then merge results of individual threads

- Create measurements to identify the best number of threads

Discuss Threads
(start(), run(), join())
U12.A2 – 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

b) 1'000'000 Integers
   - A main class to perform the measurements
   - Here also U10.A1 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!
U12.A3 – Recursive Problem Solving

- The company Springli intends to bring a new chocolate on the market

- Acceptance of all rectangular formats with a maximum of n pieces must be tested

- How many rectangular Schokis can a fictionary company („Springli“) produce with n pieces?

- Hint:
  - For n = 1, 2, 3, 4, 5, 6 exists 1, 3, 5, 8, 10, 14 formats.
U12.A3 – Springli Formats (I)

\[ n = 1 \rightarrow 1 \text{ Format} \]
U12.A3 – Springli Formats (II)

\[ n = 2 \rightarrow 3 \text{ Formats} \]
U12.A3 – Springli Formats (III)

\[ n = 3 \rightarrow 5 \text{ Formats} \]
U12.A3 – Springli Formats (IV)

n = 4 → 8 Formats
n = 5 → 10 Formats

U12.A3 – Springli Formats (V)
U12.A3 – Springli Formats

Recursive Solution:

\[ \text{Formats}(n) = \text{Formats}(n-1) + \ldots \]

<table>
<thead>
<tr>
<th>Number</th>
<th>Divisors</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1, 3</td>
</tr>
<tr>
<td>4</td>
<td>1, 2, 4</td>
</tr>
<tr>
<td>5</td>
<td>1, 5</td>
</tr>
<tr>
<td>6</td>
<td>1, 2, 3, 6</td>
</tr>
</tbody>
</table>