DATA STRUCTURES AND ALGORITHMS

LECTURE 14

Lect. PhD. Marian Zsuzsanna

Babeș - Bolyai University
Computer Science and Mathematics Faculty

2018 - 2019

In Lecture 13

- Binary Search Tree
- AVL Trees
- Problems solved with stacks and queues

Today

1. Problems solved with stacks and queues

Delimiter matching

- Given a sequence of round brackets (parentheses), (square) brackets and curly brackets, verify if the brackets are opened and closed correctly.
- For example:
  - The sequence ()([][](())]) - is correct
  - The sequence [()()()()] - is correct
  - The sequence [()]) - is not correct (one extra closed round bracket at the end)
  - The sequence [(]) - is not correct (brackets closed in wrong order)
  - The sequence {[[]] () - is not correct (curly bracket is not closed)

Bracket matching - Solution Idea

- Stacks are suitable for this problem, because the bracket that was opened last should be the first to be closed. This matches the LIFO property of the stack.
- The main idea of the solution:
  - Start parsing the sequence, element-by-element
  - If we encounter an open bracket, we push it to a stack
  - If we encounter a closed bracket, we pop the last open bracket from the stack and check if they match
  - If they don't match, the sequence is not correct
  - If they match, we continue
  - If the stack is empty when we finished parsing the sequence, it was correct

Bracket matching - Implementation

```
function bracketMatching(seq) is:
    init(st) //create a stack
    for elem in seq execute
        if @ elem is open bracket then
            push(st, elem)
        else //elem is a closed bracket
            if isEmpty(st) then
                bracketMatching ← False //no open bracket at all
            else
                lastOpenedBracket ←pop(st)
                if not @lastOpenedBracket matches elem then
                    bracketMatching ← False
                end-if
            end-if
        end-if
    end-for //continued on next slide...
```

Bracket matching - Implementation

```
    if isEmpty(st) then
        bracketMatching ← True
    else //we have extra open bracket(s)
        bracketMatching ← False
    end-if
end-function
```

- Complexity: $\Theta(n)$ - where n is the length of the sequence

Bracket matching - Extension

- How can we extend the previous implementation so that in case of an error we will also signal the position where the problem occurs?
- Remember, we have 3 types of errors:
  - Open brackets that are never closed
  - Closed brackets that were not opened
  - Mismatch

Bracket matching - Extension

- How can we extend the previous implementation so that in case of an error we will also signal the position where the problem occurs?
- Remember, we have 3 types of errors:
  - Open brackets that are never closed
  - Closed brackets that were not opened
  - Mismatch
- Keep count of the current position in the sequence, and push to the stack $< delimiter, position >$ pairs.

Red-Black Card Game

- Statement: Two players each receive $\frac{n}{2}$ cards, where each card can be red or black. The two players take turns; at every turn the current player puts the card from the upper part of his/her deck on the table. If a player puts a red card on the table, the other player has to take all cards from the table and place them at the bottom of his/her deck. The winner is the player that has all the cards.
- Requirement: Given the number n of cards, simulate the game and determine the winner.
- Hint: use stack(s) and queue(s)

Robot in a maze

- Robot in a maze:
  - Statement: There is a rectangular maze, composed of occupied cells (X) and free cells (*). There is a robot (R) in this maze and it can move in 4 directions: N, S, E, V.
  - Requirements:
    - Check whether the robot can get out of the maze (get to the first or last line or the first or last column).
    - Find a path that will take the robot out of the maze (if exists).

```
X * * X X X * *
X * X * * * * *
X * * * * * X *
X X X * * * X *
* X * * R X X *
* * * X X X X *
* * * * * * * X
X X X X X X X X
```

Robot in a maze

- Let T be the set of positions where the robot can get from the starting position.
- Let S be the set of positions to which the robot can get at a given moment and from which it could continue
- A possible way of determining the sets T and S could be the following:

```
T ← {initial position}
S ← {initial position}
while S ≠ ∅ execute
    Let p be one element of S
    S ← S \{p}
    for each valid position q where we can get from p and which is not in T do
        T ← T ∪ {q}
        S ← S ∪ {q}
    end-for
end-while
```

- T can be a list, a vector or a matrix associated to the maze
- S can be a stack or a queue (or even a priority queue, depending on what we want)

Conclusions

Conclusions

- While most programming languages have most containers already implemented, you need to know their specifics in order to use them.
- Example: Bachelor Degree Exam, 2016 June (http://www.cs.ubbcluj.ro/wp-content/uploads/subiect_licenta_informatica_iunie_2016_en.pdf)

- Understanding how these containers are implemented (by understanding the underlying data structure), will help you choose the right implementation when there are several possibilities.

Written exam

Group	Primary date	Secondary date
911	21.06	27.06
912	27.06	21.06
913	21.06	20.06
914	21.06	18.06
915	18.06	11.06
916	27.06	20.06
917	18.06	20.06
2nd, 3rd year Students	11.06	20.06

- Rooms and the starting hours are available at the faculty's webpage.

Written exam

- Every student has to participate in the exam on the primary date.
- Every student is expected to participate at the primary date by default. If someone wants to switch from the primary date D1 to a different date D2, this should be announced at the latest two days (48 hours) before MIN(D1, D2). Unless there are too many students already for D2 I am going to accept switches, even if D2 is not the secondary date of the group.
- Exam will take 2.5 - 3 hours - results will be given as soon as we can.
- You will need a grade of at least 5 for the written exam to be able to pass this course.

Written exam I

- Subjects for the written exam will be from everything we have covered this semester.
- You will have 5 problems:
  - Problem 1 implementation - either something on a simple data structure, or a problem where you use an ADT to solve a problem
  - Problem 2 - points a, b, c - mainly drawings, or short text answers (not code)
  - Problem 3 - "Pick the right answer from a, b, c, d and explain" - 6 questions
  - Problem 4 - "implement a given operation for a given ADT represented with a given DS" or "find the best SD to solve this problem with a given complexity and solve it"
  - Problem 5 - with binary trees

Written exam II

- The data structures for which we did not discuss implementation (skip lists, binomial heap, etc.) will appear only at problems 2 or 3.
- "Think about it" problems are good candidates for exam problems.