Lab04-MemoizationGoneWildcard

Darwin Jacob Groskleg

February 28, 2019

Contents

1 src/main.cpp

```
/* main.cpp
 * --------
 * CSCI 355 Algorithm Analysis
 * Lab 4    Memoization Gone Wildcard!
 *
 * Authors: Darwin Jacob Groskleg
 * Date:    Tuesday, February 26, 2019
 *
 * CONSOLE SAMPLE
 * $> make
 * $> ./bin/lab04 < test/given_tests.in
 * Test Case 1
 *   Phrase:  "abba"
 *   Pattern: "a*a"
 * Match
 *
 * Test Case 2
 *   Phrase:  "abba"
 *   Pattern: "a?a*"
 * No match
 *
 * Test Case 3
 *   Phrase:  "abba"
 *   Pattern: "a**a"
 * Match
 *
 * Test Case 4
 *   Phrase:  "abcba"
 *   Pattern: "*ab?ba*"
 * Match
 *
 * $> ./bin/lab04 < test/given_tests.in 2>/dev/null \
 *      | diff -y - test/given_expected.out
 * Match                         Match
 * No match                         No match
 * Match                         Match
 * Match                         Match
 */
```

```cpp
#include "Glob.hpp"
#include <iostream>

using namespace std;

bool match(string pattern, int m, string phrase, int n);

int main() {
    int test_cases;
    cin >> test_cases;

    for (int t=1; t<=test_cases; t++) {
        string phrase, pattern;
        cin >> phrase >> pattern;
        clog << "Test Case " << t << '\n'
            << "  Phrase:  \"" << phrase  << "\"\n"
            << "  Pattern: \"" << pattern << "\"\n";
        match(pattern, pattern.size()-1, phrase, phrase.size()-1);
        clog << '\n';
    }

    return 0;
}

/// Prints whether the given pattern matches the phrase.
///     Is just a procedural interface for Glob.
///
/// T(n) = O(n)     where n = size_of_pattern x size_of_phrase
///     See implementation of Glob for analysis.
bool match(string pattern, int m, string phrase, int n) {
    Glob globber(pattern, phrase);                  // T(n) = O(n)
    bool is_match = globber.match(m,n);             // T(n) = O(n)
    if (is_match)
        cout << "Match\n";
    else
        cout << "No match\n";
    return is_match;
}
```

2 src/Glob.hpp

```
/* Glob.hpp
 * --------
 * CSCI 355 Algorithm Analysis
 * Lab 4    Memoization Gone Wildcard!
 *
 * Authors: Darwin Jacob Groskleg
 * Date:    Tuesday, February 26, 2019
 *
 * Purpose: an interface to the Glob object.
 */
#ifndef GLOB_HPP_INCLUDED
#define GLOB_HPP_INCLUDED

#include "MaybeMatrix.hpp"
#include <string>

/// Glob
///
/// Serves to determine if a pattern containing wildcard characters (*, ?)
/// can be expanded, or `globbed`, to match the given phrase.
///
class Glob {
    const std::string pattern;
    const std::string phrase;

public:
    /// constructor
    ///
    /// T(n) = O(n)     where n = size_of_pattern x size_of_phrase
    ///
    Glob(std::string pattern_, std::string phrase_) :
        pattern{ pattern_ },
        phrase{ phrase_ },
        memos(pattern.size(), phrase.size())                // T(n) = O(n)
    {}
```

```
39      /// match
40      ///
41      /// Returns whether the given pattern matches to the phrase.
42      ///
43      /// T(n) = O(n)     where n = size_of_pattern x size_of_phrase
44      ///
45      bool match();
46      bool match(int m, int n);
47
48
49  private:
50      /// An MxN matrix for storing memoized subproblems for the match(m, n).
51      MaybeMatrix<bool> memos;
52
53      /// memoize
54      ///
55      /// Returns the given value after ensuring something has been memoized
56      /// for (m, n).
57      ///
58      /// T(n) = O(1)
59      ///
60      bool memoize(int m, int n, bool return_value);
61  };
62
63  #endif // GLOB_HPP_INCLUDED
```

3 src/Glob.cpp

```cpp
/* Glob.cpp
 * --------
 * CSCI 355 Algorithm Analysis
 * Lab 4    Memoization Gone Wildcard!
 *
 * Authors: Darwin Jacob Groskleg
 * Date:    Tuesday, February 26, 2019
 *
 * Purpose: the implementation of Glob.
 */
#include "Glob.hpp"
#include <cassert>

bool Glob::match() {
    return Glob::match(pattern.size()-1, phrase.size()-1);
}

/// match
///
/// ANALYSIS
///     where n' = size_of_pattern x size_of_phrase
///              = m x n
///
///   TRYING THE MASTER METHOD
///     Worst case for dividing and combining, f(n') is the last of the base
///     cases with a large m: f(n') = W(m).
///     Since we use memoization this would only ever happen once in the entire
///     greater problem, thus f(n) = O(1).
///
///     The problem may divide in to 2 subproblems, so a = 2.
///     In that case (and the others, mostly) the subproblem size is n-1.
///     So  n' - 1    = n'/b
///         b(n' - 1) = n'
///         b = n'/(n' - 1)
///         ...so b is not constant,
///         CAN'T USE MASTER METHOD.
///
///
```

```
39  ///   THINKING ABOUT TREES & SUBPROBLEM OVERLAP
40  ///     Any base case is solved in constant time.
41  ///     Each recursize call case has f(n') = O(1),
42  ///     and will recurse again until it either:
43  ///         1. hits a base case,
44  ///         2. reaches a memoized result.
45  ///     Then it memoizes all values going back up.
46  ///     Thus, no subproblem is ever computed more than once.
47  ///     Since there are m x n subproblems we have the follwing:
48  ///
49  ///     T(n') =     O(m * n)    can't do worse than this
50  ///           = OMEGA(m * n)    can't do asymptotically better than this
51  ///           = THETA(m * b)    since big-O == big-omega
52  ///
53  bool Glob::match(const int m, const int n) {
54      // Exit early with a memo
55      if (memos.has(m, n))
56          return memos.value(m, n);
57
58  // BASE CASES (not memoizable)
59
60      // If m < 0 and n < 0,
61      //      return true  both reached the end
62      if (m < 0 && n < 0)
63          return true;
64      // If m < 0 and n >= 0,
65      //      return false  only pattern reached end
66      if (m < 0 && n >=0)
67          return false;
68      // If n < 0 and m >= 0,
69      //      return true only if remaining characters in pattern are all *
70      if (n < 0 && m >= 0) {
71          for (int i=0; i<=m; i++)
72              if (pattern.at(i) != '*')
73                  return false;
74          return true;
75      }
76
77
78
```

```
79      // Closure to be concise
80      auto memoize = [this, &m, &n] (bool return_value) {
81          return this->memoize(m, n, return_value);
82      };
83
84  // RECURSIVE CASES
85
86      // If pattern[m] == *, return true only if either:
87      //      pattern[0..m] matches phrase[0..n-1],
88      //      OR pattern[0..m-1] matches phrase[0..n]
89      if (pattern.at(m) == '*')
90          return memoize( match(m, n-1) || match(m-1, n) );
91
92      // If pattern[m] == ?, return true only if
93      //      pattern[0..m-1] matches phrase[0..n-1]
94      if (pattern.at(m) == '?')
95          return memoize( match(m-1, n-1) );
96
97      // Otherwise return true only if both
98      //      pattern[m] == phrase[n]
99      //      AND pattern[0..m-1] matches phrase[0..n-1]
100     else
101         return memoize( (pattern.at(m) == phrase.at(n)) && match(m-1, n-1) );
102 }
103
104 /// memoize
105 ///
106 /// This method IS idempotent but not in respect to the value of return_value,
107 /// only that there is a guarantee something must then be at any valid position
108 /// matric(m, n).
109 bool Glob::memoize(int m, int n, bool return_value) {
110     if (!memos.has(m, n))
111         memos.emplace(m, n, return_value);
112     // Must Fail: The matrix is mutable, our memos should not be.
113     assert( (memos.value(m, n) == return_value)
114         && "Return value differs from memoed value.");
115     return return_value;
116 }
```

4 src/MaybeMatrix.hpp

```
/* MaybeMatrix.hpp
 * ---------------
 * CSCI 355 Algorithm Analysis
 * Lab 4    Memoization Gone Wildcard!
 *
 * Authors: Darwin Jacob Groskleg
 * Date:    Tuesday, February 26, 2019
 *
 * Purpose: template (interface and implementation) of MaybeMatrix.
 */
#ifndef MAYBEMATRIX_HPP_INCLUDED
#define MAYBEMATRIX_HPP_INCLUDED

#include <vector>
#include <cassert>

/// MaybeMatrix
///
/// An m-by-n matrix (A_mn) where each position maybe contains a value.
/// Position access is indexed starting from 0 by the same form as 'a_ij'
/// where a is the element located in
///     - the i'th row and
///     - the j'th column.
///
/// Uses the Maybe Monad pattern from Haskell in its interface to simplify
/// having to manage 2 matrices and get the assurances from the type system.
///
/// SPACE COMPLEXITY (at construction and thereafter)
///     S(n) = theta( n )   where n = rows x columns
///
///     The implementation relies on the space-efficient specialization of
///     std::vector<bool>, which uses std::bitset in some implementation.
///     However, this does not bring an asymptotic improvement.
///
/// TIME COMPLEXITY (at construction)
///     T(n) = theta( n )   where n = rows x columns (x construction of type T)
///
///     The implement relies on std::vector, which is linear in this case.
```

```cpp
template<typename T>
class MaybeMatrix {
    std::vector< std::vector<bool>> _maybes; // uses a bit-array internally
    std::vector< std::vector<T> >   _matrix;

public:
    /// T(n) = O(m*n)
    MaybeMatrix(int m_rows, int n_columns);

    /// has
    ///     Returns whether the matrix has a value at a given coordinate.
    ///     Out of range queries don't break the semantics so they must be
    ///     defined behaviour thus returning false.
    /// T(n) = O(1)
    /// Usage:
    ///      MaybeMatrix<bool> m(3,3);
    ///      if( m.has(1,1)) ...
    bool has(int row_i, int column_j) const noexcept;

    /// value
    ///     Returns the value stored at the given coordinates.
    ///     Fails if query is out of range or element is empty,
    ///     so always query using has() first.
    /// T(n) = O(1)
    T value(int row_i, int column_j) const;

    /// emplace
    ///     Mutates the matrix at the given coordinates by the inserting the
    ///     given value.
    ///     Fails for impossible negative coordinates.
    ///     Does nothing for flat matrices but fails for out of range access
    ///     for all others.
    /// T(n) = O(1)
    void emplace(int row_i, int column_j, T value);

private:
    // This makes it obvious that there's a bug in the constructor,
    // where there's >0 columns and 0 rows.
    bool is_flat() const;
};
```

```
79
80  template<typename T> MaybeMatrix<T>::
81  MaybeMatrix(int m_rows, int n_columns) :
82      _maybes(m_rows, std::vector<bool>(n_columns, false) ),
83      _matrix(m_rows, std::vector<T>(n_columns) )
84  {
85      assert( (m_rows >= 0 && n_columns >= 0)
86                  && "m_rows and n_columns must be positive");
87  }
88
89  template<typename T> auto MaybeMatrix<T>::
90  has(int row_i, int column_j) const noexcept -> bool
91  {
92      // Out of Range: negative should be false
93      if (row_i < 0 || column_j < 0)
94          return false;
95      // Out of Range: greater than should be false
96      if (static_cast<std::size_t>(row_i) >= _matrix.size()
97              || static_cast<std::size_t>(column_j) >= _matrix.at(0).size())
98          return false;
99      bool b = _maybes.at(row_i).at(column_j);
100     return b;
101 }
102
103 template<typename T> auto MaybeMatrix<T>::
104 value(int row_i, int column_j) const -> T
105 {
106     return _matrix.at(row_i).at(column_j);
107 }
108
109 template<typename T> auto MaybeMatrix<T>::
110 emplace(int row_i, int column_j, T value) -> void
111 {
112     assert((row_i >= 0 && column_j >= 0)
113             && "i and j must be positive");
114     if (!is_flat()) {
115         _matrix.at(row_i).at(column_j) = value;
116         _maybes.at(row_i).at(column_j) = true;
117     }
118 }
```

```
119 
120 template<typename T> auto MaybeMatrix<T>::
121 is_flat() const -> bool
122 {
123     return (_matrix.size() == 0 || _matrix.at(0).size() == 0);
124 }
125 
126 #endif // MAYBEMATRIX_HPP_INCLUDED
```