Lab06-Coloring

Darwin Jacob Groskleg

Friday, March 15, 2019

Contents

1 src/main.cpp

```cpp
/* main.cpp
 * --------
 * CSCI 355 Algorithm Analysis
 * Lab 6    m-Coloring Problem
 *
 * Authors: Darwin Jacob Groskleg
 * Date:    Wednesday, March 13, 2019
 *
 * QUESTIONS: are answered in the comments of main().
 *
 * CONSOLE:
 * > make
 * > ./bin/Lab06-Coloring
 *  [1]:
 *  [2]:
 *  [3]: 2 1 2 1 1 3 1 1 3 3 3 1 2 2 2 1 2 2 2 1 2 1 1 1 1 3 1 3 3 3 1 3 2 2 1
 *  2 3 1 3 3 1 1 2 2 2 3 2 2 1 1 2 1 3 3 3 2 1 1 2 1 3 2 3 3 2 1 1 1 3 2 3 3 2
 *  2 1 2 2 2 3 3 1 1 3 1 2 3 2 2 1 3 1 1 1 3 1 3 3 3 3 2 3 2 2 2 3 2 2 2 3 1 1
 *  1 3 3 1 3 3 2 3 2
 *
 *  Nodes visited for m=3 : 538 in a search tree of 88572 nodes.
 *  Thus 88034 nodes have been pruned!
 *
 *  IMPROVED algorithm:
 *  Nodes visited for m=3 : 10 in a search tree of 29524 nodes.
 *  Thus 29514 nodes have been pruned!
 */
#include "graph.hpp"

#include <iostream>
#include <cmath>

int m_color_visits(int i, int n, int m);
namespace improved {
    int m_color_visits(int i, int n, int m);
}; // namespace my

using namespace std;
```

```
39
40  int main() {
41      // 1. Determine the minimum number of colors required to color this graph.
42      //
43      //      The minimum number of colours is 3.
44      //      Since there is no output from m < 3.
45      for (int m=1; m<=3; m++) {
46          cout << "[" << m << "]: ";
47          m_color(0, GRAPH_NODES, m);
48          cout << '\n';
49      }
50      cout << '\n';
51
52      // 2. How many nodes are in the search tree?
53      //
54      //      Summation(n=1..GRAPH_NODES, m^n)
55      //        = Summation(n=1..10, 3^n)
56      //        = 3^1 + ... + 3^10
57      //        = 88,572 nodes in the search tree
58      //      for the smallest number of colours m=3.
59      int search_tree_nodes = 0;
60      for (int n=1; n<=GRAPH_NODES; n++)
61          search_tree_nodes += pow(3, n);
62
63      // 3. How many nodes are pruned by the promising algorithm above?
64      //
65      //      88,034 nodes have been pruned!
66      int node_count = m_color_visits(0, GRAPH_NODES, 3);
67      cout << "Nodes visited for m=3 : " << node_count
68          << " in a search tree of " << search_tree_nodes << " nodes.\n"
69          << "Thus " << search_tree_nodes - node_count
70          << " nodes have been pruned!\n\n";
71
72      // 4. With symmetries, many colorings are the same. What are a couple of
73      //    quick enhancements that can be made to further prune the search space
74      //    to not create the same colorings?
75      //
76      //    We can do a couple of things:
77      //      1. Recognize the colour number patterns as more significant than the
78      //         colour numbers themselves. These are cyclic groups.
```

```
79      //
80      //      2. Always start with colour 1 for the first vertex then for
81      //         subsequent vertices only attempt to match colours that are a
82      //         lesser number than the vertex since.
83      //
84      //    Search tree size:
85      //      Summation(n=0..GRAPH_NODES-1, m^n)
86      //        = Summation(n=0..9, 3^n)
87      //        = 3^0 + ... + 3^9
88      //        = 29,524 nodes for the smallest number of colours m=3.
89      search_tree_nodes = 0;
90      for (int n=0; n<GRAPH_NODES; n++)
91          search_tree_nodes += pow(3, n);
92
93      // 5. Now how many nodes are visited by the algorithm?
94      //
95      //      The algorithm visits 10 nodes!
96      node_count = improved::m_color_visits(0, GRAPH_NODES, 3);
97      cout << "IMPROVED algorithm:\n"
98          << "Nodes visited for m=3 : " << node_count
99          << " in a search tree of " << search_tree_nodes << " nodes.\n"
100         << "Thus " << search_tree_nodes - node_count
101         << " nodes have been pruned!\n";
102
103
104     return 0;
105 }
106
107 /// m_color_visits
108 ///
109 /// Returns the number of nodes visited in the same search tree generated by
110 /// m_color() in graph.hpp
111 /// Arguments:
112 ///     i - some vertex in the graph
113 ///     n - number of vertices in the graph
114 ///     m - maximum number of colours to colour the graph with
115 int m_color_visits(int i, int n, int m) {
116     int visits = 0;
117     if (i == n)
118         return visits;
```

```
119     else {
120         visits++;
121         for (int c=1; c <= m; c++)
122             if (promising(i,c)) {
123                 color[i] = c;
124                 visits += m_color_visits(i+1, n, m);
125             }
126     }
127     return visits;
128 }
129 
130 /// improved::m_color_visits
131 ///
132 /// Returns the number of nodes visited using a more sophisticated pruning
133 /// method than what's used in m_color() in graph.hpp
134 /// Arguments:
135 ///     i - some vertex in the graph
136 ///     n - number of vertices in the graph
137 ///     m - maximum number of colours to colour the graph with
138 int improved::m_color_visits(int i, int n, int m) {
139     int visits = 0;
140     if (i == n)
141         return visits;
142     else {
143         visits++;
144         for (int c=1; c <= m; c++)
145             if (i<c && promising(i,c)) {
146                 color[i] = c;
147                 visits += m_color_visits(i+1, n, m);
148             }
149     }
150     return visits;
151 }
```

2 include/graph.hpp

```cpp
/* graph.hpp
 * ---------
 * CSCI 355 Algorithm Analysis
 * Lab 6    m-Coloring Problem
 *
 * Authors: Martin van Bommel, Darwin Jacob Groskleg
 * Date:    Wednesday, March 13, 2019
 *
 * Purpose: data structures and functions given as part of the lab explanation.
 */
#ifndef GRAPH_HPP_INCLUDED
#define GRAPH_HPP_INCLUDED

#include <iostream>

// There are 10 vertices
#define GRAPH_NODES 10

// Adjacency matrix representing an undirected graph:
//      true iff vertex i and vertex j are adjacent.
const bool W[GRAPH_NODES][GRAPH_NODES] = {
    { 0, 1, 0, 0, 1, 1, 0, 0, 0, 0 },
    { 1, 0, 1, 0, 0, 0, 1, 0, 0, 0 },
    { 0, 1, 0, 1, 0, 0, 0, 1, 0, 0 },
    { 0, 0, 1, 0, 1, 0, 0, 0, 1, 0 },
    { 1, 0, 0, 1, 0, 0, 0, 0, 0, 1 },
    { 1, 0, 0, 0, 0, 0, 0, 1, 1, 0 },
    { 0, 1, 0, 0, 0, 0, 0, 0, 1, 1 },
    { 0, 0, 1, 0, 0, 1, 0, 0, 0, 1 },
    { 0, 0, 0, 1, 0, 1, 1, 0, 0, 0 },
    { 0, 0, 0, 0, 1, 0, 1, 1, 0, 0 }
};

// Colors (ints) starting from 1.
int color[GRAPH_NODES * GRAPH_NODES];
```

```
39 /// promising
40 ///
41 /// Determines for some vertex i whether all its adjacent vertices are not the
42 /// colour c, the same colour as i.
43 /// Arguments:
44 ///     i - some vertex in the graph
45 ///     c - some color value (>=1)
46 bool promising(int i, int c) {
47     for (int j=0; j<i; j++)
48         if (W[i][j] && color[j] == c)
49             return false;
50     return true;
51 }
52
53 /// m_color
54 ///
55 /// Outputs the maximum number of colour used by each possible configuration
56 /// for the graph W using only m colours.
57 /// No two adjacent vertices can be the same colour.
58 /// Arguments:
59 ///     i - some vertex in the graph
60 ///     n - number of vertices in the graph
61 ///     m - maximum number of colours to colour the graph with
62 void m_color(int i, int n, int m) {
63     if (i == n)
64         std::cout << color[i-1] << " ";
65     else
66         for (int c=1; c <= m; c++)
67             if (promising(i,c)) {
68                 color[i] = c;
69                 m_color(i+1, n, m);
70             }
71 }
72
73 #endif // GRAPH_HPP_INCLUDED
```