CS472 Module 8 Part B - Graph Coloring

Athens State University

March 4, 2014

Outline

Contents

1 Introduction

What is the compiler doing to my code?

- Modern compilers perform extensive optimization of code
  - To the point where the compiler is far better at writing assembly language than a human can ever aspire to doing
- One of the prime problems the compiler must solve is how to allocate variables to a small set of registers
  - Variables that cannot be assigned to register must be *spilled* to memory and loaded in/out for access
  - Optimizing compilers aim to assign as many variables to registers as possible

Relationship to graphs

- Compilers have the ability to determine what sets of variables are *live* at the same time
- This information is used to built a graph such that each vertex of the graph represents a unique variable in a program
- Edges of this graph are organized into two sets:

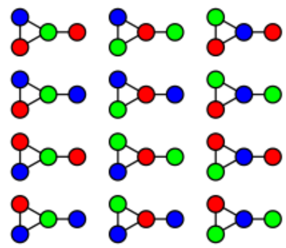
  Interference edges connect pairs of vertexes which are live at the same time

  Preference edges connect pairs of vertexes which are involved in `move` instructions
- We can determine how to allocate registers by picking n colors, where n is the number of registers in the processor, and assigning colors to vertexes
  - No two vertexes connected by an interference edge can have the same color
  - Two vertexes sharing a preference edge should have the same color

Graph coloring

- We have now reduced the problem of register allocation to the problem of assigning a set of colors to the vertexes of a graph
- There are a number of other real world problems that can be reduced to graph coloring: map coloring, flight scheduling, pattern matching
- For example, we can reduce completing a Sudoku puzzle to attempting to assign 9 colors to a specific graph with 81 vertexes

Examples and terms



The 12 ways we can 3-color this graph

- A **k-coloring** is a coloring of a graph using at most k colors
- The smallest number of colors required to color a graph G is called its **chromatic number**
- A graph is **k-chromatic** if it is k-colorable and its chromatic number is exactly k
- A k-coloring of a graph is the same as partitioning the vertex set into k independent sets. Thus k-colorable and k-parite has the same meaning

Chromatic polynommial

- The **chromatic polynomial** $P(G, t)$ counts the number of ways a graph can be colored using no more than a given number of colorings
- The chromatic number of a graph is the smallest positive integer that is not a root of the chromatic polynomial of that graph
- Study of the properties of these polynomials has been important to algebraic graph theory in mathematics and computational geometry and computational complexity in computer science

2 Graph coloring Algorithms

Exact Algorithms

- Brute-force search algorithms for k-coloring
  - Do each of k^n assignments of k colors to n vertexes and check to see if they are legal
  - For finding the chromatic number and chromatic polynomial, do this for every $k = 1, \ldots, n-1$
- Dynamic programming algorithms exist that can find decide k-colorability with worst-case performance of $O(2.445^n)$
- Other algorithms have been found that can k-coloring in $O(2^n * n)$

A Backtracking Algorithm for k-coloring

Algorithm 1: k_color(): A Backtracking Algorithm for k-coloring

---

Input: An adjacency matrix W, k for number of colors, and n for number of vertexes
Output: For each coloring, an array *vcolor* where i-th entry in the array is color assigned to node i

```
if promising(i) then
    if i = n then
        Output vcolor array;
    else
        for color ∈ [1..k] do
            vcolor[i + 1] = color;
            k_color(i+1);
```

A Backtracking Algorithm for k-coloring

Algorithm 2: promising(i): backtracking reject function for k-coloring

---

Input: An adjacency matrix W, k for number of colors, and n for number of vertexes, and an index i
Output: True if path in state space leads to solution, false otherwise

```
switch ← true;
j ← 1;
while (j < 1) and switch is true do
    if W[i][j] and vcolor[i] = vcolor[j] then
        switch ← false;
        j ← j + 1 ;
return switch
```
