**Project 2 documentation**

**CS 421**

**Yanzhou Huang and Yi Liu**

**11/19/2017**

**Graph Coloring Problem**

We have been given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with same color. And final represent the maximum color sequence of the given graph.

V: vertices

M: color amount

Colorsum[]:one array containing the color sequence

Color[]: One array containing color

Graph[][]: adjacency matrix of the graph .

**QuickSort**

Using QuickSort to sort the color sequences with descending order. To select an element from a sequence called a "pivot", reordering the sequence, all elements bigger than the pivot are placed in front of the pivot, and all elements larger than the pivot are placed behind the pivot. After this, recursively sort the subarray of elements bigger than pivot and the subarray of elements smaller than the pivot. When you recursive to the bottom, the number line is zero or one, which is already sorted.

In my function I pick first element as pivot. The quick sort produces the most effective and widely used method of sorting a list of any item size.

The time complexity for QuickSort will be 0(n log(n))

**Pseudo-code:**

Quicksort(array, high, low)

While (high<low)// //using recursive algr low>=high

while(high<low and pivot >array[low])

low--

if (high < low)

swap

high++

while (high < low and pivot < array[high])

high++

if (high < low)

swap

low--

array[high] = pivot;//put pivot to its position

if (high < pivot-location)

quickSort(array, high, pivot-location - 1)// Do the recursive to the left side array

if (pivot-location < low)

quickSort(array, pivot-location + 1, low)//Do the recursive to the right side array

**Backtracking**

Backtracking is an optimal search method, which is searched on the optimal condition to achieve the target. But when discover a step, find the original choice is not optimal or targets, just take a step back to choose, this is no return to walk again technology for backtracking, and meet back in the condition of a certain state point is called the "back point".

Using backtracking to solve the color problem, the idea is to give the colors one by one to different vertices, starting at 0. Before assigning colors, we examine “OK” by considering the assigned colors to adjacent vertices. If we find a “OK” color allocation, we mark color allocation as part of the solution. If we don't find color in the conflict, then we go back to the previous one.

The time complexity for backtracking is = nm(m^n-1)/(m-1)=O( (mV)^V), O(mv)if check only one.

**Pseudo-code:**

graphColor([]color, [][]graph, v, m, []Colorsum)

While (k>1)

color[k] += 1; // get the color

while (color[k] <= m and k <= v)

if (OK(k, color, graph)) //if it is ok

break

else

color[k] += 1 // //if have conflict, add one color

if (color[k] <= m and k == v) // if search to the bottom vertices, output one color combo

print()

if (color[k] <= m and k < v) //keep search in deep

k = k + 1

color[k] = 0

k = k – 1 // //when color[k]>m or k>n， backtrack

**Adjacency matrix**

A array graph[i][j] where V is the number of vertices in graph and graph[i][j] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

The time complexity for Adjacency matrix is 0(V^2). With adjacency matrix, the time required to find all outgoing edges is O(n) because all n columns in the row for a node must be inspected. Summing up across all n nodes, this works out to O(n^2)

**Boolean ”ok” method**

A utility function to check whether the current color assignment can be used for vertex v or not.

**Pseudo-code:**

Boolean OK(k, []color, [][]graph)

For(i=1 to k)

If graph[k][i]=1 and color[k]=color[i]

Return false

Return true

**Data structures**

One array containing color, array color[V] that should have numbers from 1 to m.

Another array Colorsum[] containing the color sequence. The index will be the color, the value will be the amount of this color.

The time complexity for array; add O(n), search O(n).