Graph Project

Charlie

June 23, 2019

1 Introduction to code

The package includes three code files: main.c, func.h and func.c. In main.c, three ways are available to initialize the adjacent matrix. One way is generating a graph randomly by this code:

generate_graph(Mat, 0.7, time(NULL));

Or you can set the seed for random function as a constant by:

generate_graph(Mat, 0.7, 20);

In addition, you can input an adjacent matrix manually by calling this function:

input_graph(Mat);

More over, it can read a file with name graph.txt as input. (A template file graph.txt is contained in the package.)

read_graph(Mat);

If you want to change size of graph as well as adjacent matrix, you could change it in func.h where #define NUM_NODE n. (n should be a positive integer) For more details about how to use these functions, you could read comments in func.h.

2 Exact Algorithm

We compare the performance of exhaustion algorithm (méthode brutale) and Zykov algorithm on different graph size.

NUM_NODE	9	10	11	12
Exhaustion	0.173 s	0.390 s	7.947s	>300 s
Zykov	<1 ms	<1 ms	< 1 ms	<1 ms

We notice that the calculating time for exhaustion algorithm explosives with the increasing of graph size. Since the time complexity of this algorithm is $O(n^n)$.

3 Approximate Algorithm

3.1 Counter-Example

One counter-example for both Leighton and Brelaz algorithms could not give the best result is shown in figure 1.

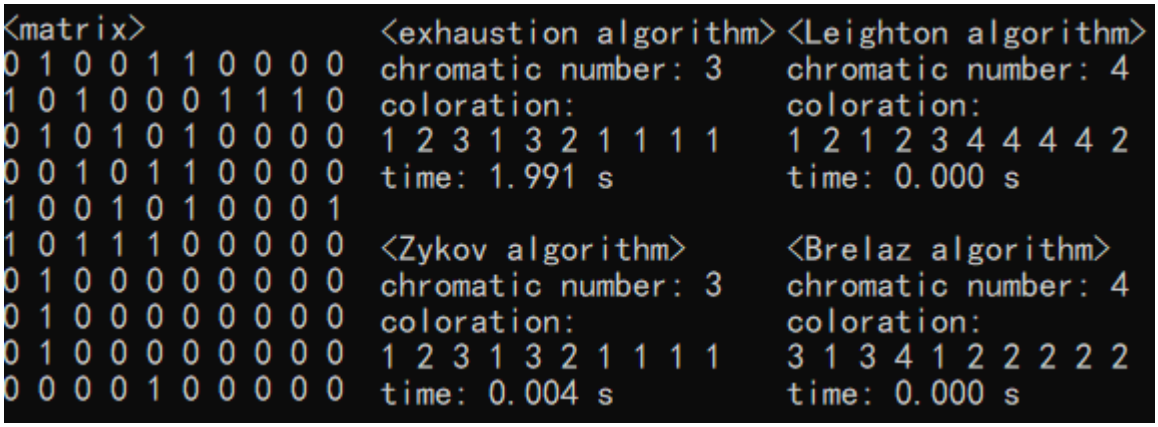

Figure 1: counter-example

3.2 Performance

Results are shown in the following table. Clearly, approximate algorithms calculate solution much faster than Zykov algorithm and for most cases, they will give the best solution.

NUM_NODE	12	14	16	18	20	22
Zykov	<1 ms	0.012 s	0.063 s	1.644 s	6.827 s	55.554 s
Leighton	<1 ms	<1 ms	< 1 ms	<1 ms	<1 ms	<1 ms
Brelaz	<1 ms	<1 ms	< 1 ms	<1 ms	<1 ms	<1 ms