**Benchmark**

Para realizar foi utilizada a IDE code block com compilador GCC no windows 10, foram feito três caso de testes para cada entrada de arestas:

Melhor caso: quando nenhum vértice é conectado a alguma aresta, assim só haverá uma cor.

Pior caso: quando cada vértice do grafo é conectado com cada um dos outros vértice, assim haverá uma cor diferente para cada vértice.

.

Caso médio: foi gerado um grafo que conecta os vértices aleatoriamente, e com a metade da quantidade de arestas de cada do pior caso.

Para gerar as arestas aleatoriamente foi usado as seguintes funções para cada algoritmo:

Guloso

srand(time(NULL));

for (int j=1; j<=NUMEDGE; j++)

{

int a = rand() % NUM;

int b = rand() % NUM;

g.addEdge(a,b);

}

BackTracking

for(int i = 0; i < E; i++)

{

int a = rand() % (V - 1);

int b = rand() % (V - 1);

graph[a][b] = 1;

graph[b][a] = 1;

}

E para cada caso de teste, foi definido um tempo limite de 20 minutos

Tabela de testes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Teste N° | Vértices | Arestas  (pior/ médio) | Algoritmo Guloso: Tempo - N° Cores  (melhor/ pior/ médio) | Algoritmo BackTracking: Tempo - N° Cores  (melhor/ pior/ médio) |
| 1 | 4 | 16/8 | 0.071 s - 1 cor/ 0.078 s - 4 cores/ 0.085 s - 3 cores | 0.079 s - 1 cor/ 0.076 s - 4 cores/ 0.075 s - 2 cores |
| 2 | 10 | 100/50 | 0.082 s-1 cor/0.072 s - 10 cores/ 0.078 s - 5 cores | 0.078 s - 1 cor/2.913 s - 10 cores/ 2.592 s - 5 cores |
| 3 | 12 | 144/72 | 0.071 s-1 cor/0.081 s - 12 cores/ 0.067 s - 5 cores | 0.071 s - 1 cor/ timelimitout/ timelimitout |
| 4 | 100 | 10^4/5x10^3 | 0.063 s-1cor/0.074s - 100 cores/ 0.080 s - 27 cores | 0.077 s - 1 cor/ timelimitout/ timelimitout |
| 5 | 10000 | 10^8/10^7 | 0.081 s-1 cor/ alloc error/ 18.677s - 294 cores | execution error/ execution error/ execution error |

**Análise dos resultados**

Observamos que o BackTracking, aumenta drasticamente o seu tempo de execução, conforme maior o número de vértices do grafo, não sendo possível identificar o tempo no seu tempo limite determinado para grande número de arestas, tal comportamento pode ser devido a sua natureza recursivo, que aumenta o número de recursos conforme maiores os números nas entradas, e não funciona para mil dezenas de entrada.

Possui um tempo de execução estável desde pequenas, entradas, até às demais grandes entradas. No caso do teste 5,, ele gera alloc error também para o caso de 5x10^7 arestas, por isso foi adaptado para um caso que ainda funciona(10^7), para mostrar que é capaz de gerar ainda um grande número de cores.

REFERÊNCIAS:

https://www.geeksforgeeks.org/test-case-generation-set-4-random-directed-undirected-weighted-and-unweighted-graphs/