TRABALHO 2 DE ESTRUTURAS DE DADOS AVANÇADAS – 2014.2

Nome do grupo:
Ian Albuquerque Raymundo da Silva - 1310451
Gustavo Marques Martins – 1310630
Victor Macedo Sabino - 1312035

1 - Encontrar o Click

(a) Inicie com S, um conjunto vazio
(b) Seleciona um vertice de V de grau maximo em G = (V,E)
(c) Adicione a S um vertice que seja vizinho a todos os vertices ja em S dentre estes, escolha o vertice de maior grau.
(d) Se nenhum vertice puder ser adicionado a S, Pare! S e uma clique.

Nesta primeira questão incializamos o grafo como uma set de inteiros da biblioteca “set” de tamanho máximo *MAX_VERTICES*.

```
set<int> graph[MAX_VERTICES];
```

Então foi feito a função *int vertexWithMaximumDegree()* que percorre todo os vértices comparando para ver qual tem maior grau.

```
for (set<int>::iterator it=vertices->begin(); it!=vertices->end(); it++)
  {
   if(vertex_degree[*it] > max_degree)
        {
                max_degree = vertex_degree[*it];
                 vertex_with_max_degree = *it;
         }
}
  return vertex_with_max_degree;
```

Para achar o *click* fizemos a função *void findCliqueIncludingSoloVertex(set<int>* vertices)* que acha o vértice de maior grau e o click do mesmo.

Para encontrar o click no qual o vertice de maior grau está, criamos um vetor de vertices adjacentes vector<int> v_adj, que corresponde aos vertices adjacentes e candidatos a participarem do clique do vertice encontrado pela função vertexWithmaximumDegree().

Para isso fizemos um vetor de vértices adjacentes *vector<int> v_adj* que adiciona todos os vértice que não fazem parte do click e são adjacente ao mesmo.

```
while(true)
```

```
{
    adjacent_vertices.clear();
    for (set<int>::iterator i=vertices->begin(); i!=vertices->end(); i++)
    {
        if(!isVertexInTheClique(*i,&clique))
        {
            if(hasEdgesToEveryoneInClique(*i,&clique))
            {
                adjacent_vertices.push_back(*i);
            }
        }
    }
```

Se não houver vetores adjacentes ao click não existe mais membros a serem adicionados, o próximo passo foi checar nos vizinhos que não são partes do click qual entre eles tem o maior grau. Esse será adicionado e o processo repetido até não existir mais irmãos.

```
        if(!num_adjacent_vertices)
        {
            break;
        }


        for(int i=0; i<num_adjacent_vertices; i++)
        {
            if(vertex_degree[adjacent_vertices[i]] > max_degree)
            {
                max_degree = vertex_degree[adjacent_vertices[i]];
                vertex_with_max_degree = adjacent_vertices[i];
            }
        }
```

Para finalizar, a função void findClique(set<int>* vertices) verifica se o clique é maior do que 1 para seguir a definição de cliques.

2 - Cobertura Minimal

(a) Inicie com V’ = V . Faca q = 1.
(b) Encontre uma clique em G(V’) correspondente ao conjunto S0q
.
(c) Faca V’ = V’\ S’q
.
(d) Se V’ != 0. Faca q = q + 1 e volte ao passo (b)
(e) Caso contrario, para p de 1 a q faca:
i. Adicione a S’p
um vertice que seja vizinho a todos os vertices ja em S0p
dentre estes,
escolha o vertice de maior grau.
ii. Se nenhum vertice puder ser adicionado a S’p
, Pare! S’p
e uma clique.
iii. Faca Sp = S’p
.
(f) O conjunto S1;....., Sq e uma cobertura minimal em cliques.

O primeiro passo foi iniciar dois objetos do tipo set (presentes na biblioteca “set”) *vertices* e *candidates*. Com isso, usamos as funções criadas no exercício anterior para achar a clique G(v’), e com um loop que vai extraindo o clique atual V’ (aqui chamado de vertices) de S.

```
do
   {
        component_index++;

        findCliqueIncludingSoloVertex(&vertices);
        clique_cover_components[component_index] = clique;

        for(set<int>::iterator it=clique_cover_components[component_index].begin();
it!=clique_cover_components[component_index].end(); it++)
        {
              vertices.erase(*it);
        }
```

```
  }
while(!vertices.empty());
```

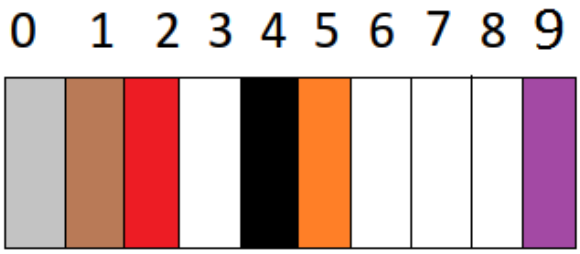
Então, é inserido no set<int> candidates os elementos que tem todos os vértices vizinhos com o clique, se este cumprir essa regra será adicionado ao clique. Após esses passos, fazemos uma varredura sobre todos os cliques encontrados e caso encontremos algum elemento sem clique, o inserimos em algum clique.

3 - Encontra Coloração

(a) Selecione um vertice v pertence a V .
(b) Execute uma Busca em Profundidade em G = (V,E). A partir de v. Utilize a marcacão
dos vertices para atribuir as cores. Faca c(v) = 1. Marque cada vertice visitado com a cor de menor índice possvel.
(c) Ao final indique o maior indice utilizado (o K da coloracão).

Para esta etapa, fizemos o vector *int vertex_color[MAX_VERTICES]* que armazena as cores de todos os vértices, em seguida inicializamos as cores com -1 e é chamado a função *void color(int v)* que é responsável por colorir cada vértice.

Também usamos o array de vetores *bool adjacent_used_colors[MAX_VERTICES]* que é preenchido com as cores dos vértices adjacentes, para ser usado a menor cor.



*espaços não coloridos são representados pela cor branca

A figura, que representa o array *used_colors[MAX_VERTICES]*, mostra como seria um possível vetor de cores. Nele, iriamos color o vértice atual com o vetor 3 por ser a menor cor possível.

Tendo isso em mente, preenchemos todas as cores e depois, para colorir, procuramos até que um vértice não-colorido seja achado, então o vértice para de ser preenchido.
É chamada a função *void setAdjacentUsedColors(int vertex, int num_adjacent_vertices, bool* adjacent_used_colors)* que é responsável por colorir as cores adjacentes nesse array:

```
for(int i=0;i<num_adjacent_vertices;i++)
  {
      if(vertex_color[adjacency_list[vertex][i]] != -1)
      {
          adjacent_used_colors[vertex_color[adjacency_list[vertex][i]]] = true;
      }
  }
```

O próximo passo é analisar qual é o valor não-colorido e colorir o vértice desta cor, e então, analisar todos os vértices adjacentes procurando por vértices sem cor, quando achar, chamar recursivamente a função *void colorVertex(int vertex)* com o valor achado.

```
for(int i=0;i<num_adjacent_vertices;i++)
  {
      if(vertex_color[adjacency_list[vertex][i]] == -1)
                                                  {
          colorVertex(adjacency_list[vertex][i]);
      }
  }
```

Tempo de Execução dos Algorítimos

Para cada arquivo disponibilizado, executamos o algorítmo de encontrar cliques, de encontrar coberturas e de colorir gráfos. Medimos o tempo de execução para comparar os resultados de execução desses algoritmos entre os arquivos. O número de vértices no grafo e sua densidade de arestas variavam de acordo com cada teste. Segue, na tabela abaixo, os resultados obtidos.

Nome do Arquivo de Teste	Número de Vértices	Densidade de Arestas	Tempo Médio para "Encontra Clique"	Tempo Médio para "Encontra Cobertura"	"Tempo Médio para "Colore Grafo"
brock200_2.clq	200	50%	0.006552 s	0.434846 s	0.001667 s
brock200_4.clq	200	65%	0.010589 s	0.527091 s	0.001752 s
brock400_2.clq	400	75%	0.056256 s	10.119000 s	0.005107 s
brock400_4.clq	400	75%	0.061756 s	7.077500 s	0.004761 s
brock800_2.clq	800	65%	0.178586 s	72.221000 s	0.014855 s
brock800_4.clq	800	65%	0.180379 s	74.505000 s	0.014873 s
C1000.9.clq	1000	90%	0.225333 s	237.379000 s	0.020415 s

Os tempos encontrados estão de acordo com o previsto. Quanto maior o número de vértices e a densidade de arestas, maior o tempo de execução, conforme o esperado.

Nestas execuções obtemos também o tamanho do clique encontrado pelo algoritmo "encontra clique", a quantidade de cores utilizadas para colorir o grafo pelo algoritmo "colore grafos" (uma aproximação para o número cromático do grafo) e o tamanho do maior, menor e tamanho médio dos cliques utilizados pela cobertura encontrada pelo algoritmo "encontra cobertura".

IMPORTANTE: É importante notar que os casos em que existem vértices isolados no grafo são contabilizados pelo algoritmo como um clique de tamanho um, justificando médias de clique de tamanho menor que dois (que por definição, é o tamanho mínimo de um clique).

Os resultados encontrados encontram-se na tabela a seguir:

Nome do Arquivo de Teste	Número de Vértices	Densidade de Arestas	Tamanho do Clique Encontrado pelo “Encontra Clique”	Tamanho do Maior Clique da Cobertura	Tamanho do Menor Clique da Cobertura	Média dos Temanhos dos Cliques	Quantidade de Cores Utilizadas
brock200_2.clq	200	50%	8	8	2	1.90	35
brock200_4.clq	200	65%	13	13	2	2.08	49
brock400_2.clq	400	75%	19	19	2	2.01	100
brock400_4.clq	400	75%	18	18	2	2.13	100
brock800_2.clq	800	65%	15	15	2	2.09	147
brock800_4.clq	800	65%	14	16	2	2.05	145
C1000.9.clq	1000	90%	4	38	2	1.40	183

Primeiramente podemos observar que a quantidade de cores utilizadas é maior (ou igual) que o tamanho dos cliques encontrados tanto pelo algoritmo “Encontra Clique” quanto pelo algoritmo “Encontra Coloração”, uma vez que o número de cores necessárias para colorir um clique de tamanho N é N.

Quanto maior o grafo e maior a densidade de arestas, maior o tamanho da maior cobertura encontrada.

Como podemos ver, nos exemplos abaixo, o numero de cliques com menos número de vértices, normalmente é maior que o numero de cliques com maior número de vértices. Um grafo com muitos cliques de muitos vertices, é um grafo muito denso.

Com a maior densidade é esperado um maior clique da cobertura, no brock200_2.clq a densidade é de 50%, já no C1000.9.clq com densidade de 90%, logo é plausível o maior click ser no com maior densidade.

brock200_2.clq

Number of cliques with size 2 = 10
Number of cliques with size 3 = 6
Number of cliques with size 4 = 8
Number of cliques with size 5 = 3
Number of cliques with size 6 = 1
Number of cliques with size 7 = 3
Number of cliques with size 8 = 2

C1000.9.clq

Number of cliques with size 2 = 33
Number of cliques with size 3 = 23
Number of cliques with size 4 = 8
Number of cliques with size 5 = 4
Number of cliques with size 6 = 2
Number of cliques with size 7 = 2
Number of cliques with size 8 = 1
Number of cliques with size 9 = 5
Number of cliques with size 10 = 3
Number of cliques with size 12 = 2
Number of cliques with size 14 = 1
Number of cliques with size 38 = 1