UNIVERSIDADE FEDERAL DE MINAS GERAIS
ALGORITMOS E ESTRUTURAS DE DADOS III

TP 2
Problemas NP-Completo

Aluno: Rafael Ramon de Oliveira Egídio
Email: nomar22@gmail.com
Matricula: 2009052360

1-Introdução

Este documento irá detalhar a solução de um algoritmo , por meio de um programa de computador, do problema de alocação de canais em redes de telefonia Celular.

O objetivo do trabalho prático é encontrar o menor número possível de frequencias necessárias para atender a cidade de Belo Horizonte sem que haja interferencias entre os sinais ,dada a localização das antenas.

Existe interferencia entre duas ERBs(Estação de rádio base) caso exista um ponto em que esteja dentro do raio de cobertura de duas ou mais ERBs e estas estejam usando o mesmo canal de comunicação.

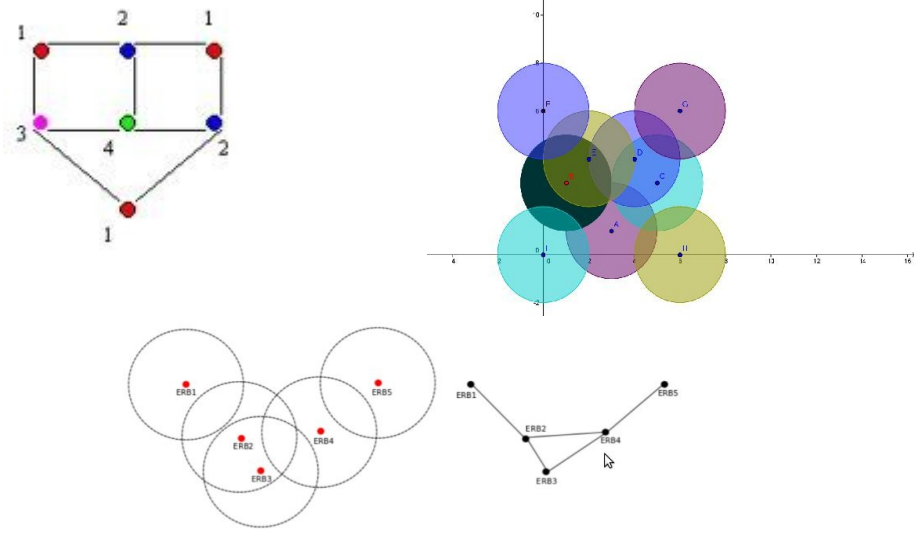
Esta empresa solicita que dado uma gama de localizações de ERBs distribuídas pela cidade , qual a quantidade mínima de canais a empresa precisará assinar junto ao órgão federal responsável para que consiga atender a cidade sem problemas com interferências.

2-Modelagem e solução proposta

O problema de alocação de frequência será modelado como um problema de coloração de grafos, no qual o objetivo é colorir um certo grafo com o mínimo de cores distintas possíveis sem que hajam dois vértices com a mesma cor ligados por alguma aresta.

Este problema de coloração de grafos é sabidamente um problema da classe Np-completo e não se conhece uma solução em tempo polinomial para tal.

Cada vértice deste grafo será a representação de uma ERB e cada aresta representará uma possível interferencia entre duas ERBs.



A solução do problema passa pela instanciação da estrutura grafo seguida da inserção de vértices, de maneira que a cada vértice inserido é verificada a existencia de interferencia em seu raio de atuação. Para cada dois vértices que se encontre interferência uma nova aresta é inserida ligando estes dois vértices.

Para efeito de orientação as cores do grafo serão representadas por números, de forma que o resultado final apresentará o índice da ultima cor inserida no grafo da solução . O algoritmo de coloração de vértice colorirá sempre um vértice com o menor id de cor disponível,

buscando esta informação nas arestas ligadas ao vértice. Um vértice é considerado não colorido quando seu id de cor for igual a zero.

Com o grafo pronto em memória existem dois algoritmos para a solução do problema e a decisão de qual algorimo deve-se seguir será tomada em tempo de chamada do programa. Para a solução heurística ou ótima é preciso chamar o programa obedecendo-se a seguinte sintaxe respectivamente:

tp2h input.txt output.txt;

tp2o input.txt output.txt;

Definições

SOLUÇÃO HEURÍSTICA- Tenta a partir de alguma lógica chegar um resultado satisfatório que pode ser equiparado ao ótimo, mas não garantidamente é o próprio, gerando muitas das vezes uma solução aproximada, mas com um custo de execução viável.

SOLUÇÃO ÓTIMA: Garante que a solução encontrada é a melhor possível, porém para isto, para um problema da classe NP-Completo possui um custo que na maioria das vezes a torna inviável para entradas minimamente consideráveis.

3 Ambiente de Teste

Este trabalho foi testado em um computador Intel I3 2.4 GHz 4GB com sistema operacional Linux Ubuntu 12.04 32bits.

4 Requisitos

A solução do trabalho deverá ser implementada na linguagem de programação C. O programa receberá um arquivo de entrada,passado como parametro na chamada do programa e retornará uma saída conforme informada na entrada do programa.

Estrutura do arquivo de entrada:

1 //Número de Instancias

3 //Número de Erbs

17 36 7 //Coordenadas x e y e Raio de cobertura da ERB

11 26 6

25 28 6

Estrutura do arquivo de saída:

O programa deverá imprimir no arquivo de saída seguindo o modelo:

2 //Número de cores

5 Algoritmo da Solução

Algoritmo HEURÍSTICO

Será utilizado como heurística para o problema de coloração de grafos a heurística chamada MaiorPrimeiro, que consiste na ideia de que um vértice com uma alta incidencia de arestas tende a ser um vértice mais “dificil” de se colorir, com isto os vértices que possuem um mais alto grau serão coloridos primeiro, como não foi implementado um critério de desempate a qualidade do resultado apresentado por este algoritmo ótimo é comprometida em ambientes que possuam um alto número de Herbs com o mesmo grau de interferência. Fica em aberta a oportunidade de melhoria deste algoritmo através de algum critério de desempate.

Algoritmo ÓTIMO:

A solução ótima apresentada neste trabalho consiste na implementação de um algoritmo de permutação que percorre todo o vetor de adjacencias do grafo e tenta colorir o grafo vértice a vértice, sempre com a menor cor possível, de forma que a cada linha do resultado da permutação é contabilizado o número de cores necessário para colorir o grafo naquela ordem, ao final de todas as permutações é possível indicar o menor número de cores possível para se colorir este grafo e a ordem que se deve colorir.

Dado que este algoritmo de força bruta é exageradamente custoso e inviável para entradas maiores que 10, obtido por experimentação, foi implementado uma política de poda de árvores que antes de permutar cada galho da árvore verifica se o número de cores encontrada até este caminho já atingiu o mínimo encontrado em algum outro galho, caso já tenha atingido este galho é podado e assim otimizando o algoritmo em algumas ordens de grandezas .

6 Estruturas de dados utilizadas e descrições

Para um melhor entendimento do código e estruturas convencionou-se que neste trabalho prático estruturas de dados recebem um nome e apontadores para estas estruturas recebem o prefixo **Link** que define o tipo como apontador.

Principais Estruturas:

```
typedef struct coord{
      double x;
      double y;

}Coordenadas;
```

Coordenadas x e y do plano cartesiano;

```
typedef struct {
  LinkCelula primeiro;
  LinkCelula ultimo;
  int len;
}Lista;
```

Lista encadeada comum, na estrutura célula existirá um vértice.

```
typedef struct vertice{
      Coordenadas coord;
      float raio;
      Lista arestas;
      int cor;

}Vertice;
```

O vértice possui as suas coordenadas de localização, o tamanho de seu raio , sua cor atual e uma lista de apontadores para os vértices que existem arestas com este vértice.

```
typedef struct grafo{
	//Array dinâmico de ponteiros
	LinkVertice *vertices;
	int len;
	int inseridos;

}Grafo;
```

Trata-se de um lista de Adjacencias para os vértices do grafo e seu tamanho.

7 Pseudo-códigos e descrições de funções

Grafo.c

É o objeto direto da modelagem do mapa de ERBs para uma cidade em grafo.

7.1Funções públicas da Biblioteca

São as funções que podem ser utilizadas por qualquer programador que queira utiliza-la sem alterar o funcionamento básico da biblioteca chamadas de funções públicas.

```
void incializaGrafo(LinkGrafo grafo, int tam);
	Instancia o grafo somente.

LinkVertice criaVertice(Coordenadas coord, float raio);
	Instancia um vértice e sua lista de arestas, retorna um ponteiro para este vértice novo.

void insereVertice (LinkVertice vertice, LinkGrafo grafo,int pos);
	Insere um vértice um vértice em um posição pos no grafo, em seguida insere todas as arestas deste vértice .

void carregaGrafo(LinkGrafo grafo, FILE * p);
	Recebe um ponteiro do grafo e o arquivo de entrada para a construção do grafo.

void permute(LinkVertice *a, int i, int n,int *numCores);
	Função que permuta o vetor de adjacencias, colore o grafo em cada uma das combinações de posição e guarda sempre o menor número de cores possível na variável numCores, realiza além disto a poda verificando se existe a necessidade de caminhamento na sub-árvore.

void maiorPrimeiro(LinkVertice *a,  int n,int *numCores);
	Algoritmo heurístico que ordena o vetor de adjacencias e colore os vértices do grafo por ordem de maior grau.
```

7.2 Funções internas da biblioteca

Funções de uso restrito a biblioteca, são invisíveis para o usuário e tem a função de manter o funcionamento básico da biblioteca.

```
float distancia(Coordenadas a, Coordenadas b);
      Calcula a distancia entre dois pontos através do cáculo da hipotenusa.

int possuiInterferencia(LinkVertice a, LinkVertice b);
      Retorna 1 se a distancia entre os pontos for menor que a soma dos raios
dos vértices.

void criaAresta(LinkVertice a, LinkVertice b);
      Insere um ponteiro de a para b e um ponteiro de b para a.

void insereArestas(LinkVertice vertice,LinkGrafo grafo);
      Verifica no grafo a existencia de arestas para a inserção de um novo
vértice e insere estas arestas.

int verificaCorMinimaLivreVizinho(Lista vertices,int menor);
      Deve ser chamada com o valor de menor sendo o número de vértices do grafo,
itera de 0 até o número de vértices(número de cores no pior caso) se caso a cor
menor for maior que a do vértice atual ,este passa ser o menor e faz o mesmo
processo,  caso não continua percorrer a lista.

void coloreVertice(LinkVertice vertice);
      Seta o valor da cor com o retorno da menor cor em que é possível colorir
este vértice.

int coloreGrafo(LinkVertice *vertices, int tam);
      Dado um vetor de ponteiros de vértices colore o grafo na ordem que eles
estão posicionados neste vetor.

void swap (LinkVertice *x, LinkVertice *y);
      Troca a posição de dois ponteiros.

int compareGrau(LinkCelula a, LinkCelula b);
      retorna 1 se o grau de a>b e -1 caso não seja.

void ordenaLista(LinkVertice *vetor,int tamanho);
      Utiliza a função qsort da biblioteca do c que recebe um ponteiro para a
função de comparação , o vetor e seu tamanho.
```

9 Algoritmo principal

Ótimo

Enquanto houverem instancias do problema

Inicia o grafo

Carrega o grafo

permuta o grafo comparando a menor cor em cada linha da permutação

escreve no arquivo o número de mínimo de cores

fim Enquanto

Heurístico

Enquanto houverem instancias do problema

Inicia o grafo

Carrega o grafo

ordena os vértices do grafo por ordem de grau

colore o grafo seguindo a ordem

fim Equanto

10 Execução

Para a compilar este trabalho deve-se executar o comando make de dentro do diretório raiz do projeto em seguida os comandos para heurístico e ótimo respectivamente:

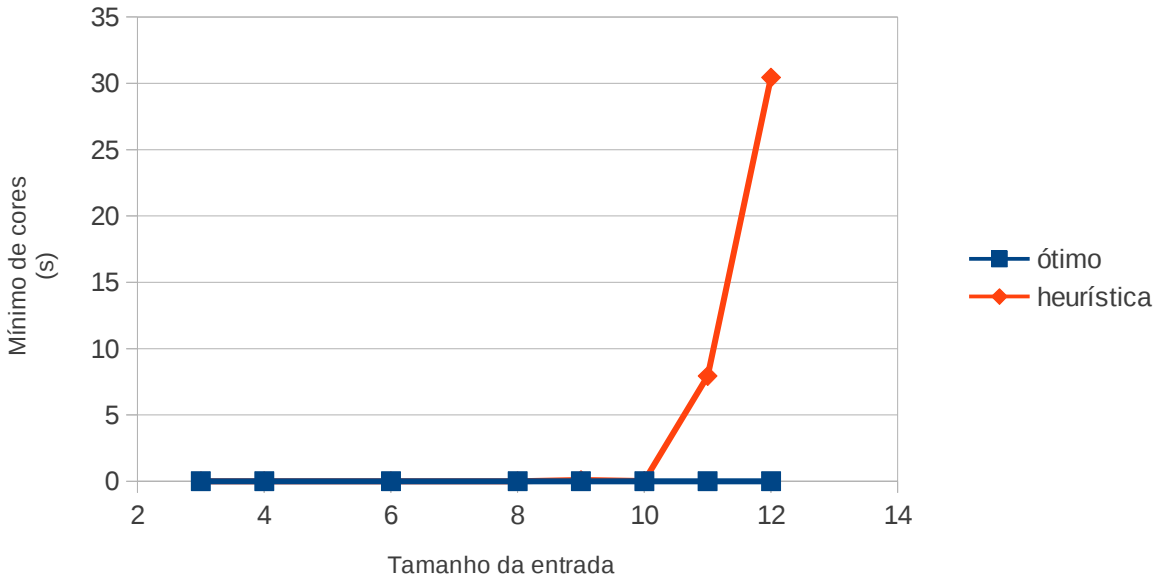
./tp2h [arquivodeentrada.txt} [arquivodesaida.txt]

./tp2o [arquivodeentrada.txt} [arquivodesaida.txt]

O arquivo de entrada deverá seguir a especificação dada em tópico anterior e a saída do programa será impressa no arquivo de saida desde que o mesmo possua permissão de escrita.

11 Avaliação Experimental

Para a realização dos testes foram geradas entradas com número de ERB's e comparado o desempenho do algoritmo frente ao ótimo.



12 Conclusão

Mais importante do que entender o funcionamento deste problema , muitas das vezes o maior dos problemas é ,diante de uma situação real perceber e conseguir provar que um problema trata-se de um problema da classe NP-completo , onde deve ser aceitável uma solução aproximada ao invés da solução ótima. O aprendizado deste trabalho prático foi importante na parte do entendimento de como se modelar um problema real e usar-se de técnicas para auxiliar na solução de um problema.

Referências

Ziviani, Nivio (2011) “Projeto de Algoritmos com implementações em Pascal e C”, Cengage Learning, 3ª Edição.

http://www.geeksforgeeks.org/