UNIVERSIDADE FEDERAL DE MINAS GERAIS
ALGORITMOS E ESTRUTURAS DE DADOS III

TP 4
Paralelização

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.

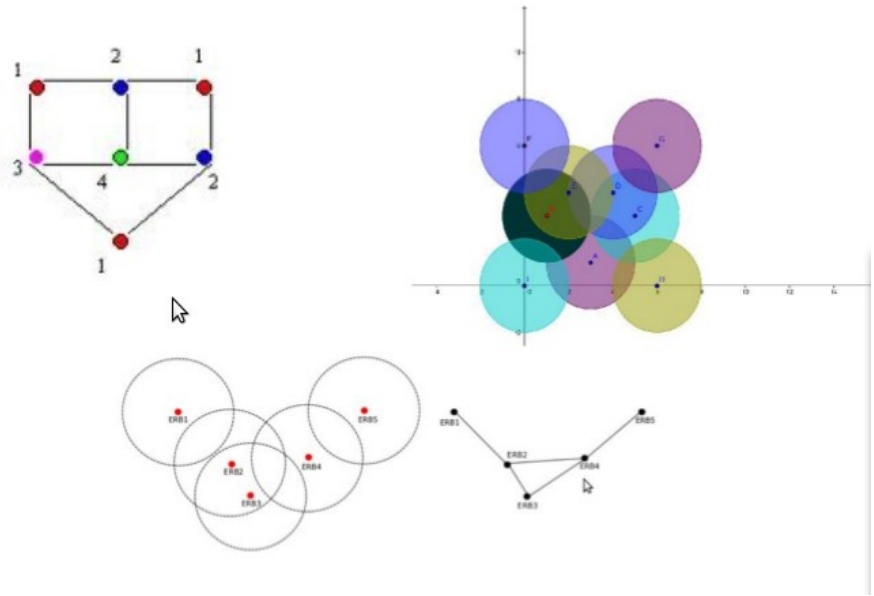
Este problema resume-se em descobrir o menor número de movimentos que são necessários para a partir de um cadeado com n letras em que todas iniciam-se com a letra a , chegar a uma senha previamente informada.

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.

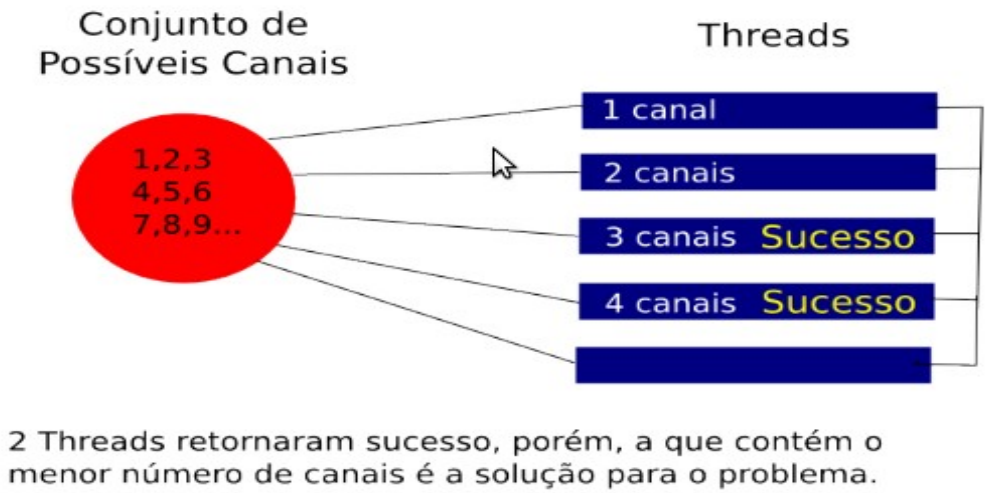
Para efeito de orientação as cores do grafo serão representadas por números, de forma que o resultado final apresentará o índice da ultima cor inserida no grafo da solução . O algoritmo de coloração de vértice colorirá sempre um vértice com o menor id de cor disponível, buscando esta informação nas arestas ligadas ao vértice. Um vértice é considerado não colorido quando seu id de cor for igual a zero.

3 Paralelização

A ideia da paralelização neste problema consiste em que no algoritmo da força bruta explicitado acima , serão disparadas N threads de forma que todas as threads tentarão o mesmo algoritmo com a única diferença de que cada thread possuirá o número que esta thread está tentando colorir, com isto ao conseguir colorir uma combinação com sua cor respectiva , esta thread termina a sua permutação.

Existem as váriaveis globais numCores, corAtual e ultimoColorido que são alteradas onde numCores é o número máximo de cores que o grafo pode ter, corAtual faz referencia a próxima cor que uma thread livre pegará e ultimoColorido é o valor mínimo que uma thread conseguiu colorir.

Ao terminar todas as tentativas , uma thread busca em o ultimoColorido+1 e passa este valor para esta thread tentar colorir. Todas as threads verificam se o valor mínimo colorido é menor que o seu valor de limite, caso seja esta thread é abortada.



2 Threads retornaram sucesso, porém, a que contém o menor número de canais é a solução para o problema.

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 mínimas para colori grafo

5 Algoritmo da Solução

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 .

7 Pseudo-códigos e descrições das principais 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 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 . O(1)

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

int coloreGrafo(LinkVertice *vertices, int tam)
      Dado um vetor de ponteiros de vértices colore o grafo na ordem que eles
estão posicionados neste vetor. O(v)para v como o número de vértices do grafo
void coloreVertice(LinkVertice vertice)
      Seta o valor da cor com o retorno da menor cor em que é possível colorir
este vértice. O(1) utililizando a verificaCorVizinho
```

```
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. O(VXA) para pior caso.

```
void permute(LinkVertice *a, int i, int n, int numLimite);
```
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. Esta função altera a variável global ultimoColorido, pois caso o valor conseguido para esta thread,identificada pelo numLimite, seja menor que o menor global altera este valor.

8 Análise de Espaço e Complexidade

Complexidade O(V!), pois apesar de utilzar as funções verificaCorMinimaLivreVizinho e colore grafo que a tornaria O(V*(V*A)) e esta é dominada pela complexidade exponencial da permutação.

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

Paralelização

Permuta com coloração disponível

Possui alguma coloração disponível menor que a menorColorida ?

Passa este valor para a thread

senão

mata a thread

10 Execução

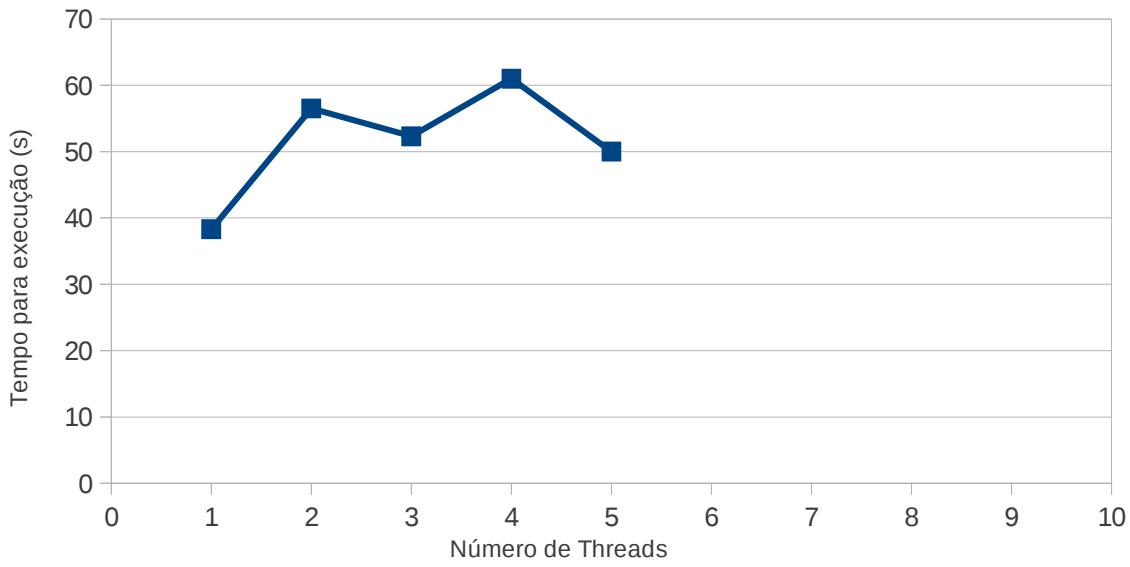
Para a compilar este trabalho deve-se executar o comando make de dentro do diretório raiz do projeto em seguida os comandos para heurístico e ótimo respectivamente:

./tp4 [arquivodeentrada.txt] [arquivodesaida.txt]

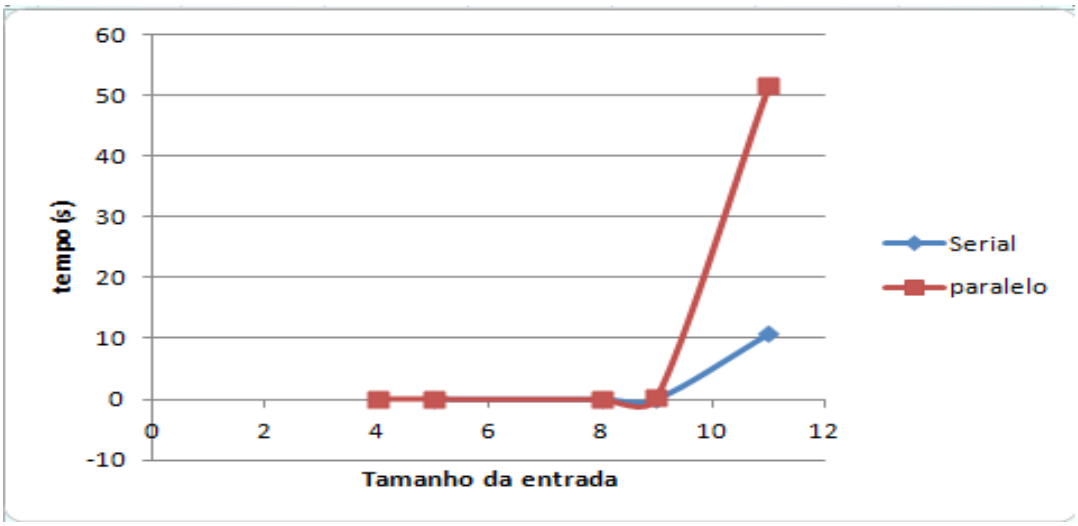
O arquivo de entrada deverá seguir a especificação dada em tópico anterior e a saída do programa será impressa no arquivo de saida desde que o mesmo possua permissão de escrita.

11 Avaliação Experimental

Mantendo a entrada com um número fixo vértices temos:



Comparando o tempo do algoritmo serial com o algoritmo paralelizado para 3 threads



12 Conclusão

O propósito do trabalho foi atingido com sucesso, contudo a paralelização deste algoritmo acabou mostrando-se ineficiente devido a uma boa parte serial do algoritmo e a computação adicional utilizada para gerenciar as threads. Outro problema associado é que de acordo com a entrada a thread com menor-1 valor executará quase que todo o algoritmo e as outras menores chegarão a um valor próximo também.

Referências

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

http://www.geeksforgeeks.org/