TRABALHO PRÁTICO 4:
Problemas NP-Completo e Programação paralela

Sandro Miccoli - 2009052409 - smiccoli@dcc.ufmg.br

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)

31 de dezembro de 2012

Resumo. *Este relatório descreve como foi modelado e implementado um problema de alto custo computacional. Será descrito como foi modelado o problema, as estruturas utilizadas e porquê não é possível implementar uma solução que consiga resolver entradas não triviais de maneira ótima. Finalmente será detalhado a análise de complexidade dos algoritmos e, por último, uma breve conclusão do trabalho implementado.*

1. INTRODUÇÃO

O trabalho propõe resolver um problema de uma rede social exclusiva para cozinheiros, o Facecook; para isso foi utilizado a teoria dos grafos para modelar o problema. Neste grafo, cada vértice representa um membro da rede, e as arestas representam uma conexão de amizade entre dois membros.

O que devemos encontrar nessa lista de membros, é o maior grupo de amigos registrados na rede social. Podemos perceber que isso se trata de um problema conhecido na teoria dos grafos, o **problema da clique**. Esse problema refere-se a qualquer problema que possui como objetivo encontrar subgrafos completos (”cliques”) em um grafo. Um subgrafo é completo se existir uma aresta para todos os pares de vértices. Como exemplo, o problema de encontrar conjuntos de nós em que todos os elementos estão conectados entre si. [Wikipedia b]

Para resolver a questão da clique utilizamos o algorimo **Bron-Kerbosch**, que é utilizado para encontrar cliques máximos em grafos não direcionados. Iremos detalhar mais sobre seu funcionamento na Seção 3.

Esse problema é **NP-Completo**, pois não pode ser resolvido de forma ótima para entradas grandes. Ou seja, não existe um algorimo de tempo polinomial para resolver esse problema, e isso que iremos mostrar neste trabalho.

Além disso, para entradas muito grandes o algoritmo sequencial não resolveria em tempo hábil, então foi proposto que construíssimos uma versão paralelizada do algoritmo.

O restante deste relatório é organizado da seguinte forma. A Seção 2 descreve como foi feita a modelagem do problema e o armazenamento das conexões do grafo. A Seção 3 descreve as estruturas e como resolvido o problema do clique sequencialmente. A Seção 4 descreve como seria feito a paralelização do problema. A Seção 5 trata de detalhes específicos da implementação do trabalho: quais os arquivos utilizados; como é feita a compilação e execução; além de detalhar o formato dos arquivos de entrada e saída. A Seção 6 contém a análise de desempenho do problema para entradas triviais e não-triviais. A Seção 7 conclui o trabalho.

2. MODELAGEM

Modelamos o trabalho com grafos implementados utilizando matriz de adjacência, além disso, para trabalhar com o problema da clique, foi necessário implementar um tipo de conjunto, para trabalhar melhor com os nós do grafo.

```
typedef struct Matriz{
    int col, lin;
    int ** matriz;
} Matriz;

typedef struct grafo {
    Matriz matrizAdj;
    int N; // Quantidade de vertices no grafo
} Grafo;

typedef struct conjunto {
    int * elementos;
    int tam;
    int qtde;
} Conjunto;
```

Essas três estruturas foram necessárias para implementar todo o trabalho. Podemos dizer que a complexidade espacial da estrutura é de $O(n^2)$, sendo n o número de vértices que o grafo tem. A estrutura de conjunto não passa de $O(n)$, pois apenas armazena quais vértices estão contidos ou não nos conjuntos, não necessariamente precisando armazenar as arestas que conectam cada um.

3. SOLUÇÃO SEQUENCIAL PROPOSTA

Para nossa solução sequencial, utilizamos uma forma básica do **Bron-Kerbosch**, que se trata de um algoritmo recursivo que utiliza *backtracking* para encontrar todos os cliques máximos de um grafo. A base do algoritmo é a busca exaustiva, mas utiliza-se um conhecimento maior sobre a natureza do problema. Ele gera apenas cliques maximais, evitando assim que cada conjunto gerado tenha que ser comparada com os previamente testados. [Bron and Kerbosch]

A estratégia de backtracking constrói incrementalmente candidatos para solução do problema, e na medida que determina que não tem possibilidade do candiato fazer parte da solução válida ele o abandona. [Wikipedia a]

A estrutura básica do algoritmo utiliza três conjuntos: um para armazenar os vértices que já foram definidos como parte da clique; outro que armazena os vértices que são candidatos a fazer parte da clique; e o último armazena os vértices que já foram analisados e que não irão fazer parte da clique.

Após o algoritmo fazer os cálculos para cada vértice, no final é possível saber qual o clique maximal do grafo. Esse clique máximo representa o maior grupo de usuários da rede social que são amigos.

No pior caso, o algoritmo apresenta complexidade exponencial de $O(2^n)$, sendo n o número de vértices no grafo.

4. SOLUÇÃO PARALELIZADA PROPOSTA

Aqui será detalhado nossa proposta de solução paralelizada, apesar dela não ter sido implementada.

O algortimo do Bron-Kerbosch é recursivo e faz uma pesquisa com backtracking para cada um dos vértices do grafo. Uma possibilidade de paralelização seria dividir o número de vértices do grafo pela quantidade de processadores, e atribuir para cada um dos processadores um conjunto de vértices a serem análisados. O resultado seria armazenado em um conjunto de solução que seria sincronizado no final da execução de cada um dos processadores.

Seria utilizado o conceito de semáforo para controlar essa sincronização dos cliques encontrados no grafo.

4.1. Algoritmos implementados

Foram implementadas e reutilizadas inúmeras funções/procedimentos que não serão detalhadas nesssa seção, pois não são necessárias para compreensão do trabalho como um todo.

- *void intersecaoVizinhos(Conjunto * P, Grafo * G, int vertice)*

  Descrição: Função que calcula a interseção entre o conjunto de vértices P com os vizinhos do vértice em questão.
  Parâmetros: Estrutura de conjunto P, estrutura de grafo G e um inteiro representando o vértice.
  Complexidade: $O(n)$, onde n é a quantidade de vértices do grafo G.

- *void uniaoVizinhos(Conjunto * P, Grafo * G, int vertice)*

  Descrição: Função que calcula a união encontre o conjunto de vértices P com os vizinhos do vértice em questão.
  Parâmetros: Estrutura de conjunto P, estrutura de grafo G e um inteiro representando o vértice.
  Complexidade: $O(n)$, onde n é a quantidade de vértices do grafo G.

- *int BK(Conjunto * C, Conjunto * P, Conjunto * S, Grafo * G)*

  Descrição: Função que faz o cálculo do Bron-Kerbosch.
  Parâmetros: Três conjuntos C, P e S e um grafo G. Dos três conjuntos, um é utilizado para armazenar os vértices que já foram definidos como parte da clique; outro que armazena os vértices que são candidatos a fazer parte da clique; e o último armazena os vértices que já foram analisados e que não irão fazer parte da clique.
  Complexidade: $O(2^n)$, onde n é o número de vértices no grafo.

4.2. COMPLEXIDADE GERAL

A complexidade geral do programa reside basicamente na complexidade do algoritmo Bron-Kerbosch, que é $O(2^n)$. Porém, como o algoritmo possui complexidade exponencial podemos supor alguns resultados.

Para cada vértice a mais no grafo, o tempo de execução do programa iria dobrar; e quando os vértices dobrassem, o tempo de execução iria elevar ao quadrado.

5. IMPLEMENTAÇÃO

5.1. Código

5.1.1. Arquivos .c

- **tp4-seq.c** Arquivo principal do programa. Lê os arquivos de entrada, calcula o maior clique de cada instância e escreve resultado em um arquivo de saída.
- **grafos_matriz.c** Contém todas as funções de manipulação, leitura e escrita de grafos. Utiliza o TAD de matriz como implementação da matriz de adjacências.
- **clique.c** Contém as funções necessárias para o cálculo do clique.
- **arquivos.c** Um tipo abstrato de dados de manipulação de arquivos, contendo funções de abertura, leitura, escrita e fechamento.

5.1.2. Arquivos .h

- **grafos_matriz.h** Além de definir a estrutura de grafos, contém todas as funções de manipulação, leitura e escrita de grafos. Utiliza o TAD de matriz como implementação da matriz de adjacências.
- **clique.h** Contém as definições das funções necessárias para o cálculo do clique. Além disso, contém a definição da estrutura de conjuntos.
- **arquivos.h** Definição da das funções utilizadas para ler, escrever e fechar corretamente um arquivo.

5.2. Compilação

O programa deve ser compilado através do compilador GCC através dos seguintes comandos

Para programação dinâmica:

```
gcc -Wall -Lsrc src/tp4-seq.c src/grafos\_matriz.c src/arquivos.c src/clique.c -o tp4-seq
```

Ou através do comando $make$.

5.3. Execução

A execução do programa tem como parâmetros:

- Um arquivo de entrada contendo várias instâncias a serem simuladas.
- Um arquivo de saída que irá receber a quantidade de nós do maior clique de cada instância.

O comando para a execução do programa é da forma:

```
./tp4-seq <arquivo_de_entrada> <arquivo_de_saída>
```

5.3.1. Formato da entrada

A primeira linha do arquivo de entrada contém o valor *k* de instâncias que o arquivo contém. As k instâncias são definidas da seguinte forma. A primeira linha contém um inteiro n indicando quantos nós (que representa o número de usuários da rede) o grafo possui. As n linhas seguintes contém a matriz de adjacência.

A seguir um arquivo de entrada de exemplo:

```
1
7
0 1 1 0 0 1 0
1 0 1 1 1 0 0
1 1 0 1 1 0 0
0 1 1 0 1 0 0
0 1 1 1 0 0 1
1 0 0 0 0 0 1
0 0 0 0 1 1 0
```

5.3.2. Formato da saída

O arquivo de saída consiste em k linhas, cada uma representando o resultado de uma instância. Cada linha contém um inteiro que representa o número de convidados para a festa, ou, em outras palavras, o maior clique encontrado no grafo.

```
4
```

6. AVALIAÇÃO EXPERIMENTAL

Não foi possível finalizar o trabalho prático para que fosse possível realizar análises experimentais decentes, por isso essa seção não irá conter nenhuma análise concreta.

6.1. Máquina utilizada

Segue especificação da máquina utilizada para os testes:

```
model name:     Intel(R) Core(TM) i3 CPU       M 330  @ 2.13GHz
cpu MHz:        933.000
cache size:     3072 KB
MemTotal:       3980124 kB
```

7. CONCLUSÃO

Infelizmente não foi possível completar o trabalho com êxito. Os conceitos de grafos e NP-Completo foram bem executados na parte de codificação, porém não produziram um resultado interessante para que fosse possível fazer análises.

Seria interessante ter tido mais tempo para pode codificar a parte de programação paralela, que talvez seja a mais complexa da disciplina.

Acredito que o resultado deste trabalho poderia ter sido melhor se ele tivesse sido postergado. Tanto pela complexidade das matérias envolvidas (NP-Completo e Paralelismo), quanto pelo tempo reduzido no fim do ano.

Referências

Bron, C. and Kerbosch, J. *Algorithm 457: finding all cliques of an undirected graph.*

Wikipedia. Backtracking. `http://en.wikipedia.org/wiki/Backtracking`.

Wikipedia. Problema do clique. `http://pt.wikipedia.org/wiki/Problema_do_clique`.