Universidade Estadual de Montes Claros - UNIMONTES
Centro de Ciências Exatas e Tecnológicas - CCET
Engenharia de Sistemas

Algoritmos e Estruturas de Dados III

Éricles Miler Dias Barbosa
Hugo Edmar Dias Santos
3º Período

Montes Claros, 16 de dezembro de 2019

1 Introdução

O trabalho final da disciplina se constituiu na leitura do artigo Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach[1] e na implementação dos códigos apresentados no mesmo para indexação e busca em um grafo direcionado acíclico extenso.

O grafo foi implementado por sua lista de adjacências, da maneira vista em sala de aula. Seguem as estruturas utilizadas na implementação e as funções iniciais.

- Estruturas:

```
typedef struct node{
  int rotulo;
  struct node *prox;
}no;

typedef struct {
  int arestas, vertices;//Número de vértices e arestas no grafo
  int *visitado;//Indicação de quais vértices já foram visitados
  int *indeg, *outdeg;//Graus de entrada e saída dos vértices
  no **adj;//Lista de adjacências
}gr;
```

- Funções para inicializar o grafo, adicionar uma aresta, ordenação topológica e zerar os elementos do vetor visitado após alguma função alterar tais valores:

```
int inicializa(gr *g, int v){
  g->vertices=v;
  g->visitado=(int*) malloc(v*sizeof(int));
  g->indeg=(int*) malloc(v*sizeof(int));
  g->outdeg=(int*) malloc(v*sizeof(int));
  if(g->visitado==NULL || g->indeg==NULL || g->outdeg==NULL) return 0;
  g->adj=(no**)malloc(v*sizeof(no*));
  if(g->adj==NULL) return 0;
  int i;
  for(i=0; i<v; i++){
    x[i], y[i]=x2[i], y2[i]=d[i]=roots[i]=iny[i]=0;
    headsend[i]=-1;
    g->adj[i]=NULL;
    g->indeg[i]=g->outdeg[i]=g->visitado[i]=0;
  }
  g->arestas=0;
  return 1;
}


int novaaresta(gr *g, int a, int b){
  if (a<g->vertices && b<g->vertices){
    no* aux=(no*) malloc (sizeof(no));
    if (!aux) return 0;
    aux->rotulo=b;
    aux->prox=g->adj[a];
    g->adj[a]=aux;
    g->indeg[b]++;
    g->outdeg[a]++;
    g->arestas++;
    return 1;
  }
```

```
  return 0;
}

void top_visit(gr *g, int i, int *v){
  no *j;
  for(j=g->adj[i]; j!=NULL; j=j->prox){
    if (g->visitado[j->rotulo] == 0){
      g->visitado[j->rotulo]=1;
      top_visit(g, j->rotulo, v);
    }
  }
  v[st]=i;
  st++;
}

void top(gr *g, int *v){
  st=0;
  for (int i=0; i<g->vertices; i++){
    if (g->visitado[i]==0){
      g->visitado[i]=1;
      top_visit(g, i, v);
    }
  }
}

void zerar(gr *g){
  for(int i=0; i<g->vertices; i++){
    g->visitado[i]=0;
  }
}

void invertlist(int *l, int n){
  int i=0, f=n-1;
  while(i<f){
    int aux=l[i];
    l[i++]=l[f];
    l[f--]=aux;
  }
}
```

2 Desenvolvimento

Inicialmente foi feita a implementação do algoritmo de indexação do método FELINE, que tem como objetivo atribuir a cada vértice do grafo um par ordenado único de forma a estabelecer uma ordem parcial entre os vértices. Também foi implementado o algoritmo de consulta de alcance, cuja função é determinar se há um caminho entre dois vértices. Segue a implementação dos algoritmos na linguagem C.

- Indexação

```
void felineindex(gr *g){

  int *x,*y;
  x=(int*) malloc(g->vertices*sizeof(int));
  y=(int*) malloc(g->vertices*sizeof(int));
  heads=(int**) malloc(g->vertices*sizeof(int*));
```

```
for(int j=0; j<g->vertices; j++){
  heads[j]=(int*) malloc(g->vertices*sizeof(int));
}
int d[g->vertices], roots[g->vertices], iny[g->vertices];
int headsend[g->vertices], rootsend=-1, yend=-1;

for(int j=0; j<g->vertices; j++){
  headsend[j]=-1;
  d[j]=0;
  iny[j]=0;
}

top(g, x);// Vértices em ordem topológica
invertlist(x, g->vertices);
int rootsend=-1, yend=-1;//índices para o final das listas roots e y

// adiciona os vértices à lista heads e define os valores de d
for(int j=0; j<g->vertices; j++){
  for(no *k=g->adj[j]; k!=NULL; k=k->prox){
    heads[j][++headsend[j]]=k->rotulo;
    d[k->rotulo]++;
  }
}
for(int j=0; j<g->vertices; j++){
  if (d[j]==0){
    roots[++rootsend]=j;
  }
}

while(rootsend>-1){
  int u;
  //seleciona a raiz com maior índice de ordem topológica e adiciona ao vetor y
  for(int j=g->vertices-1; j>=0; j--){
    if (d[x[j]]==0 && iny[x[j]]==0){
      u=x[j];
      break;
    }
  }
  y[++yend]=u;
  iny[u]=1;

  //remove u da lista roots
  int roots2[g->vertices];
  for(int j=0, i=0; j<=rootsend; j++){
    if(roots[j]!=u){
      roots2[i]=roots[j];
      i++;
    }
  }
  rootsend--;
  for(int j=0; j<=rootsend; j++){
    roots[j]=roots2[j];
  }
  for(int j=0; j<=headsend[u]; j++){
    d[heads[u][j]]--;
    if(d[heads[u][j]]==0){
      roots[++rootsend]=heads[u][j];
    }
  }
}
// índices de cada vértice na ordenação topológica e no vetor y
for(int j=0; j<g->vertices; j++){
```

```
      x2[x[j]]=j+1;
      y2[y[j]]=j+1;
    }

  }

  int reachable(gr* g, int a, int b){
    if (a==b) return 0;
    if(g->x[a]<=g->x[b] && g->y[a]<=g->y[b]){
      for(no *j=g->adj[a]; j!=NULL; j=j->prox){
        if(reachable(g, j->rotulo, b)) return 1;
      }
    }
    return 0;
  }
```

A funcionalidade de cada função foi comprovada para grafos simples, entre eles um grafo dado como exemplo no artigo [1].

A partir de então tornou-se necessária a leitura do grafo de um arquivo. O seguinte código foi desenvolvido com esse propósito:

```
gr G;
inicializa(&G, 6000);
int *a;
FILE *q;
q=fopen("arq.txt", "rt");
if(q==NULL){
  printf("Não foi possível ler o arquivo");
}
char prim[1000];//string na primeira linha
fgets(prim, 999, q);
int ver;//numero de vertices
fscanf(q, "%d\n", &ver);
printf("Número de vértices: %s %d \n ",prim,ver);
char st[5*ver+6];
int size;// quantideade de elementos em verticess
int verticess[ver];// lista de vértices adjacentes
for(int j=0; j<ver; j++){
  fgets(st, 5*ver+6, q);
  char aux[5];
  int k=0, h=0;//k indice do vetor aux, h indice vetor vertice;
  for(int i=0; st[i]!='#'; i++){
    if(st[i]==' '){
      aux[k++]='\n';
      verticess[h++]=atoi(aux);
      k=0;
    }
    else{
      aux[k++]=st[i];
    }
  }
  for(int i=1; i<h; i++){
    novaaresta(&G, verticess[0], verticess[i]);
  }
}
fclose(q);
```

Nesse momento, tivemos grandes problemas relacionados ao tamanho do grafo. Muitos erros para os quais não encontramos causa apareceram e tivemos que encontrar novas maneiras de implementação. Por fim, decidimos estabelecer o escopo das listas como global.

Implementamos também um código para ler o arquivo de teste e verificar se havia ou não caminho entre os vértices nele contidos. Chegamos ao seguinte código-fonte:

```
#include<stdio.h>
#include<stdlib.h>
#define tst 100000

int st, ccount;
int heads[6000][180];
int x[6000], y[6000], roots[6000], headsend[6000], iny[6000];
int x2[6000], y2[6000], d[6000], roots2[6000];

struct node;
typedef struct gr;

int inicializa(gr *g, int v);

int novaaresta(gr *g, int a, int b);

void top_visit(gr *g, int i, int *v);

void top(gr *g, int *v);

int reachable(gr* g, int a, int b);

void invertlist(int *l, int n);

void zerar(gr *g);

void felineindex(gr *g){
  top(g, x);
  invertlist(x, g->vertices);
  int rootsend=-1, yend=-1;
  for(int j=0; j<g->vertices; j++){
    for(no *k=g->adj[j]; k!=NULL; k=k->prox){
      heads[j][++headsend[j]]=k->rotulo;
      d[k->rotulo]++;
    }
  }
  for(int j=0; j<g->vertices; j++){
    if (d[j]==0){
      roots[++rootsend]=j;
    }
  }
  while(rootsend>-1){
    int u;
    for(int j=g->vertices-1; j>=0; j--){
      if (d[x[j]]==0 && iny[x[j]]==0){
        u=x[j];
        break;
      }
    }
    y[++yend]=u;
    iny[u]=1;
    int roots2[g->vertices];
    for(int j=0, i=0; j<=rootsend; j++){
      if(roots[j]!=u){
```

```
        roots2[i]=roots[j];
        i++;
      }
    }
    rootsend--;
    for(int j=0; j<=rootsend; j++){
      roots[j]=roots2[j];
    }
    for(int j=0; j<=headsend[u]; j++){
      d[heads[u][j]]--;
      if(d[heads[u][j]]==0){
        roots[++rootsend]=heads[u][j];
      }
    }
  }
  for(int j=0; j<g->vertices; j++){
    x2[x[j]]=j+1;
    y2[y[j]]=j+1;
  }
}


int main(){
  setlocale(LC_ALL, "Portuguese");
  gr G;
  inicializa(&G, 6000);
  int *a;
  FILE *q;
  q=fopen("arq.txt", "rt");
  char prim[1000];
  fgets(prim, 999, q);
  int ver;
  fscanf(q, "%d\n", &ver);
  printf("Número de vértices: %s %d \n ",prim,ver);
  char st[5*ver+6];
  int size;
  int verticess[ver];
  for(int j=0; j<ver; j++){
    fgets(st, 5*ver+6, q);
    char aux[5];
    int k=0, h=0;
    for(int i=0; st[i]!='#'; i++){
      if(st[i]==' '){
        aux[k++]='\n';
        verticess[h++]=atoi(aux);
        k=0;
      }
      else{
        aux[k++]=st[i];
      }
    }
    for(int i=1; i<h; i++){
      novaaresta(&G, verticess[0], verticess[i]);
    }
  }
  fclose(q);

  felineindex(&G);

  FILE *test;
  test=fopen("test.txt", "rt");
  int auxx, cont=0, cont2=0;
  for(int a=0; a<tst; a++){
    fscanf(test, "%d %d %d\n", &teste[a][0], &teste[a][1], &auxx);
```

```
  }
  cout<<endl;
  for(int i=0; i<tst; i++){
    if (reachable(&G, teste[i][0], teste[i][1])) cont++;
    else cont2++;
    zerar(&G);
  }
  cout<<"#zeros: "<<cont2<<"; #uns: "<<cont<<endl;

  FILE *res;
  res=fopen("reachability query.txt", "wt");
  for(int i=0; i<tst; i++){
    fprintf(res, "%d %d %d\n", teste[i][0], teste[i][1],
 reachable(&G, teste[i][0], teste[i][1]));
  }
  fclose(res);

  return 0;
}
```

3 Conclusão

Por fim, após a indexação do grafo, realizamos as consultas de alcance e encontramos 15459 resultado positivos: havia um caminho entre os vértices e 84541 resultados negativos.

```
for(int i=0; i<tst; i++){
  if (reachable(&G, teste[i][0], teste[i][1])) cont++;
  else cont2++;
  zerar(&G);
}
```

Utilizando a biblioteca sys/time.h e sua função gettimeofday, estimamos o tempo de execução das partes principais do código:

- Inicialização do grafo e leitura do arquivo: 30000μs;
- Indexação: 100000μs;
- Leitura do arquivo teste e realização da consulta de alcance: 2700000μs;
- Todo o algoritmo: 2900000 μs;

4 Bibliografia

- [1] Reachability Queries in Very Large Graphs: A Fast Refined Online Search Approach. Renê R. Veloso, Loïc Cerf, Wagner Meira Jr, Mohammed J. Zaki.