AA de Grafos e Algoritmos

Elaine Tady[1] **e Juliane Marinho**[2]
Orientador: Leandro G. M. Alvim[3]

[1]Departamento de Ciência da Computação –
Universidade Federal Rural do Rio de Janeiro (UFRRJ)
R. Governador Roberto Silveira S/N – Nova Iguaçu –
Rio de Janeiro – RJ – Brasil

elainetady@gmail.com, julianeufrrj@hotmail.com

alvim.lgm@gmail.com

Abstract. *This report describes the study on how to implement an algorithm to separate a graph valued G = (V, E) in k-groups closest vertices. To solve this problem the algorithm must find a minimum spanning tree in the graph and after that eliminate k-1 edges. To find the minimum spanning tree will be used Algorithm of Prim and to complement the in-depth search algorithm.*

Resumo. *Este relatório descreve o estudo sobre como implementar um algoritmo capaz de separar um grafo valorado G=(V,E) em k-grupos de vértíces mais próximo. Para resolver este problema o algoritmo deve encontrar uma árvore geradora mínima no grafo e após isso eliminar as k-1 arestas. Para achar a árvore geradora mínima será usado o algortimo de Prim e como complemento o algoritmo de busca em profundidade.*

1. Introdução

Uma solução para problemas de agrupamentos consiste em pegar o todo de um grupo e fazer a separação em k-grupos. O trabalho proposto visa fazer essa separação,obtendo uma árvore geradora mínima que obtemoscom o algoritmo de Prim para que as k-1 maiores arestas sejam eliminadas de modo que o agrupamento fique bem dividido.

Será discutida neste artigo a eficiência do algoritmo através do teste de corretude, análise algorítmica e testes computacionais realizados com diferentes tipos de grafos. Na seção 2, faremos uma breve descrição do problema. Na seção 3, descreveremos o objetivo principal do problema. Na seção 4,será apresentado o algoritmo, a demonstração da corretude e a análise de sua complexidade. Na seção 5, são apresentados testes computacionais a fim de demonstrar seu custo com diferentes tipos de grafo. Por fim, na seção 6, será feita uma pequena análise dos resultados obtidos na seção anterior e uma conclusão.

2. Problema e Motivação

Resolver problemas de agrupamento, ou seja dividir um grupo e k parte e eliminar as k-1 maiores partes utilizando árvore geradora mínima que pode ser encontrada com o algoritmo de Kruskal ou com o algoritmo Prim. Para desenvolver este trabalho o algoritmo escolhido foi o de Prim não afetando o desempenho pois as duas soluuções possuem complexidades iguais quando implementados de forma simples.

3. Objetivo

O trabalho visa separar um grafo G=(V,E) com pesos nas arestas em K partições de vértices mais próximos. A separação é feita removendo as K-1 arestas com maior peso da Árvore Geradora Mínima(MST).

4. Proposta

Para a realização desse trabalho, utilizei, além das estruturas envolvidas para a representação da lista, heap, o algoritmo de busca em profundidade e o algoritmo de Prim.

4.1. Utilizando fila e pilha

As pilhas e filas são conjuntos dinâmicos nos quais o elemento removido do conjunto pela operação DELETE é especificado previamente.Em uma pilha, o elemento eliminado do conjunto é o mais recentemente inserido: a pilha implementa uma norma de último a entrar, primeiro a sair, ou LIFO (last-in, first-out). De modo semelhante, em uma fila, o elemento eliminado é sempre o que esteve no conjunto pelo tempo mais longo, a fila implementa uma norma de primeiro a entrar, primeiro a sair, ou FIFO (first-in, first-out).[Thomas H. Cormen 2009]

4.2. O algoritmo de busca em profundidade

Na busca em profundidade, as arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas inexploradas saindo dele. Quando todas as arestas de v são exploradas, a busca ”regressa”para explorar as arestas que deixam o vértice a partir do qual v foi descoberto. Esse processo continua até descobrirmos todos os vértices acessíveis a partir do vértice de origem inicial. Se restarem quaisquer vértices não descobertos, então um deles será selecionado como uma nova origem, e a busca se repetirá a partir daquela origem. Esse processo inteiro será repetido até que todos os vértices sejam descobertos. [Thomas H. Cormen 2009]

```
DFS(G)
1 for cada vértice u ← V[G]
2     do cor[u] ← BRANCO
3        π[u] ← NIL
4 tempo ← 0
5 for cada vértice u ∈ V[G]
6     do if cor[u] = BRANCO
7           then DFS-VISIT(u)
```

```
DFS-VISIT(u)
1 cor[u] ← CINZA          ▷ Branco, o vértice u acabou de ser descoberto.
2 tempo ← tempo + 1
3 d[u] ← tempo
4 for cada v ∈ Adj[u]     ▷ Explora a aresta (u, v).
5     do if cor[u] = BRANCO
6           then π[v] ← u
7                DFS-VISIT(v)
8 cor[u] ← PRETO          ▷ Enegrece u; terminado.
9 f[u] ← tempo ← tempo + 1
```

4.3. O Algoritmo de Prim

Quando o algoritmo termina, a fila de prioridades Q está vazia, a árvore espalhada mínima A para G é portanto: $A = (v, \pi[v]) : v \in V - r$. As linhas 1 a 5 definem a chave de cada vértice como inf (com exceção da raiz r, cuja chave é definida como 0, de forma que será o primeiro vértice processado), inicializa o pai de cada vértice como NIL e incializa a fila de prioridade mínima Q para conter todos os vértices. O algoritmo mantém o seguinte loop invariante de três partes:

Antes de cada iteração do loop while das linhas 6 a 11
1 . A = (v, π[v]: $v \in V - r - Q$
2. Os vértices já colocados na árvore espalhada mínima são aqueles em V-Q.
3. Para todos os vértices v pertencente a Q, se π[v] != NIL, então chave[v] ¡ inf e chave[v] é o peso de uma aresta leve (v, π[v]) que conecta c à algum vértice já inserido na árvore espalhada mínima.

A linha 7 identifica um vértice $u \in Q$ incidente em uma aresta leve que cruza o corte (V-Q, Q), com exceção da primeira iteração, na qual u = r devido à linhas 4. A remoção de u do conjunto Q o acrescenta ao conjunto V-Q de vértices na árvore, adicionando assim (u, π[u]) a A. O loop for das linhas 8 a 11 utiliza os campos chave e pi de cada vértice v adjacente a u, mas não na árvore. A atualização mantém a terceira parte do loop invariante.[Thomas H. Cormen 2009]

```
MST-PRIM(G, w, r)
1  for cada u ∈ V[G]
2       do chave[u] ← ∞
3          π[u] ← NIL
4  chave[r] ← 0
5  Q ← V[G]
6  while Q ≠ 0
7       do u ← EXTRACT-MIN(Q)
8          for cada v ∈ Adj[u]
9          do if v ∈ Q  e w(u, v) < chave[v]
10               then π[v] ← u
11                    chave[v] ← w(u, v)
```

4.4. Prova de Corretude do Algoritmo de Prim

Uma excelente análise da corretude do algoritmo explicado acima encontra-se no livro ”Introduction to Algorithms”de Cormen et. al, descrevemos a seguir uma versão resumida. O algoritmo de Prim começa com uma árvore contendo somente um único nó (logo mínima, porém ainda não geradora) e, a cada passo, mantém a invariante de que a árvore sendo criada é mínima, através do teorema (cuja prova encontra-se no livro mencionado acima):

Dessa maneira, adicionando-se a cada passo a aresta mínima que liga um elemento que está na árvore a um elemento que não está na árvore, ao adicionarmos V-1 arestas teremos somente uma única árvore geradora, que será mínima

Dada uma árvore mínima A = (S,E'), subconjunto da árvore geradora mínima T, podemos adicionar ao conjunto A a aresta leve (u,v) e manter A contido em T. Uma aresta leve é a aresta de menor peso que liga um elemento $u \in S$ a um elemento $v \in (V - S)$.[Thomas H. Cormen 2009]

4.5. Análise de complexidade do algoritmo

Para cada u in V[G] : O(V), while Q != 0 : O(V), do u = EXTRACT-MIN(Q) : O(VlogV) ao todo, para cada v in Adj[u] : O(E) ao todo, chave[v] = w(u,v) : O(logV) para atualizar a heap e O(ElogV) ao todo. Com implementação min-heap eficiente: O(Vlog(V) + Elog(V)) = O(ElogV)

5. Experimentos

O trabalho foi implementado na linguagem C e utilizou para representar os grafos um vetor de listas encadeadas já com seus métodos básicos (inicializar, criar, excluir e etc.) também implementados. O grafo em si, fica armazenado num arquivo ".txt"e guarda o número de vértices e cada vértice com seu vizinho. Foi implementado uma busca em profundidade voltada para o problema. A estrutura da heap foi usada para que a fila de prioridades fosse ordenada pelo peso dos vértices. A Fila de prioridades foi implementada usando vetores.

Os resultados obtidos em cada execução são descritos na tabela abaixo.Os testes foram feitos em um computador com processador Dual Core 1.8Ghz. Foi contabilizado o tempo para execução do programa completo.

Tabela 1. Tabela de Experimentos

Instância	Vértices	Arestas	Tempo
1	6	7	0,016s
2	3	2	0,016s
3	10	13	0,016s

6. Conclusões

Conforme proposto a implementação é capaz de separar os vértices em grupos, utilizando uma árvore mínima através do Algoritmo de Prim em conjunto com a busca em profundidade, resolvendo assim o problema de agrupamento

O programa obtendo o grafo de um arquivo, utiliza a representação de lista de adjacências e realiza a busca em profundidade neste.Como após a remoção cada grupo se torna uma componente conexa, a busca em profundidade já pode determinar quem são os grupos próximos. Utiliza-se as informações obtidas no algoritmo de Prim(key e pred) para representar a arvore geradora minima como uma lista de arestas.

Os resultados obtidos pelo programa nos testes realizados apresentaram sempre resultados corretos e tempos de execução relativamente pequenos, mesmo aumentando o número de vértices e arestas.

Referências

Thomas H. Cormen, Charles E. Leiserson, R. L. R. C. S. (2009). In *Introduction to Algorithms*. The MIT Press.