İSTANBUL TEKNİK ÜNİVERSİTESİ

PARALEL PROGRAMLAMA PROJE RAPORU

Feyza EKSEN

(504201522)

Lisansüstü Eğitim Enstitüsü

Bilgisayar Mühendisliği (YL)

Rapor Tarihi: 29 Mayıs 2022

İÇİNDEKİLER

Sayfa

ÇİZELGE LİSTESİ

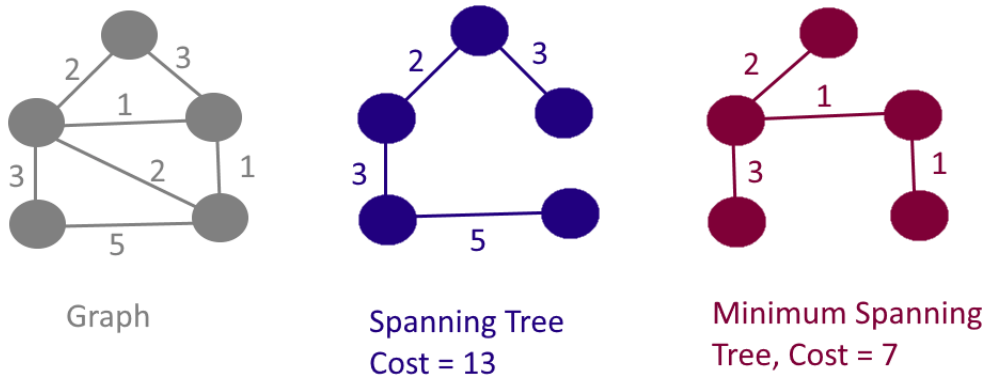
Sayfa

ŞEKİL LİSTESİ

Sayfa

1. Problemin Tanıtımı

En Küçük Kapsama Ağacı (Minimum Spanning Tree, MST) bir graf üzerinde tüm düğümleri kapsayan kenarların bir altkümesidir. MST, kapalı çevrim (çember) içermeyen (without any cycles), yönsüz (undirected), kenar ağırlıklandırılmış (edge weighted) ve bağlı (connected) grafta tanımlıdır. Bir graf üzerinde birden fazla kapsama ağacı olabilir. Bu kapsama ağaçlarından en düşük kenar değerleri toplamına sahip olan En Küçük Kapsama Ağacı (En Küçük Yol Ağacı, Minimum Spanning Tree, MST) olarak adlandırılır [2].



Şekil 1.1 : Graph, An Example of a Spannning Tree of a Graph, Minimum Spanning Tree of a Graph [1]

2. Ardışıl Algoritmanın Tanıtımı

En Küçük Kapsama Ağacı'nı (Minimum Spanning Tree, MST) hesaplamak için birçok algoritma geliştirilmiştir.

Kruskal'ın Algoritması: Daha az maliyetli kenarları tek tek değerlendirerek kapsama ağacını bulmaya çalışır. Ara işlemler birden çok ağaç oluşturabilir.

1. Graf üzerindeki düğümler, aralarında bağlantı olmayan N tane bağımsız küme gibi düşünülür.
2. Daha sonra bu kümeler tek tek maliyeti en az olan kenarlarla birleştirilir (çevrim/cycle oluşturmayacak şekilde).
3. Düğümler arasında bağlantı olan tek bir küme oluşturulmaya çalışılır.
4. Küme birleştirme işleminde en az maliyetli olan kenardan başlanılır; daha sonra kalan kenarlar arasından en az maliyetli olanlar seçilir.

Prim'in Algoritması: [3] En az maliyetli kenardan başlayıp onun uçlarından en az maliyetle genişleyecek kenarın seçilmesine dayanır. Bir tane ağaç oluşur.

- Greedy algoritmalardan biridir.
- Kruskal'ın algoritmasından tek farkı bir sonraki kenarı nasıl seçtiğidir.

1. Başlangıçta herhangi bir noktayı ağacı oluşturmaya başlamak için seçilir.
2. Oluşturulan ağaca eklemek için şu ana kadar oluşturulmuş ağaç üzerinden erişilebilen ve daha önceden ağaca katılmamış olan en küçük ağırlık kenar seçilir.
3. Eğer bu kenarın ağaca katılması, bir çember oluşmasına sebep olmuyorsa ağaca eklenir.
4. Ağaçtaki kenar sayısı (N-1)'e ulaşan kadar ikinci adıma geri dönülür.

Bu projede ardışıl algoritma olarak **Prim'in Algoritması** seçilmiştir.

3. Paralel Algoritmanın Tanıtımı

Boruvka'nın Algoritması: [4] Doğrudan paralel programlamaya yatkındır. Aynı anda birden çok ağaçla başlanır ve ilerleyen adımlarda ağaçlar birleşerek tek bir kapsama ağacına dönüşür.

1. Input is a connected, weighted and un-directed graph.
2. Initialize all vertices as individual components (or sets).
3. Initialize MST as empty.
4. While there are more than one components, do following for each component.
   (a) Find the closest weight edge that connects this component to any other component.
   (b) Add this closest edge to MST if not already added.
5. Return MST.

Time Complexity of Boruvka's algorithm is *O(E log V)* which is same as Kruskal's and Prim's algorithms.

Paralelleştirilmiş Boruvka'nın Algoritması: [5]

1. Track components via union-find data structure with union by rank and path compression
2. Parallelized search for minimum outgoing edge

4. Paralel Algoritmaya İlişkin Gerçekleme Ayrıntıları

MPI PACKAGE: Özetle, bu paket C programlama dilinde, her işlemin sıralamasını aldığı, yazdırdığı ve çıktığı bir iletişim süreçleri grubu kurar. MPI'de bu programın tüm makinelerde aynı anda başlayacağını anlamanız önemlidir. Örneğin, on makinemiz olsaydı, bu programı çalıştırmak, bu programın on ayrı örneğinin on farklı makinede birlikte çalışmaya başlayacağı anlamına gelir. Bu, sıradan C programlarından temel bir farktır, burada birisi "programı çalıştırın" dediğinde, programın yalnızca bir örneğinin çalıştığı varsayılır [6].

Nasıl Çalıştırılır?

- sudo apt-get install mpich *# ubuntu installation*

- sudo mpicc boruvka.c -o boruvka.o *# paralel boruvka algoritmasının derlenmesi*
- sudo mpirun -np 3 ./boruvka.o Graph1.csv *# paralel boruvka algoritmasının 3 process ile Graph1 üzerinde çalıştırılması*
- sudo mpirun -np 3 ./boruvka.o Graph2.csv *# paralel boruvka algoritmasının 3 process ile Graph2 üzerinde çalıştırılması*
- sudo mpirun -np 3 ./boruvka.o Graph3.csv *# paralel boruvka algoritmasının 3 process ile Graph3 üzerinde çalıştırılması*

- sudo mpicc prim.c -o prim.o *# ardışıl prim algoritmasının derlenmesi*
- sudo mpirun ./prim.o Graph1.csv *# ardışıl prim algoritmasının Graph1 üzerinde çalıştırılması*
- sudo mpirun ./prim.o Graph2.csv *# ardışıl prim algoritmasının Graph2 üzerinde çalıştırılması*
- sudo mpirun ./prim.o Graph3.csv *# ardışıl prim algoritmasının Graph3 üzerinde çalıştırılması*

5. Deneysel Sonuçlar ve Yorumlar

Çizelge 5.1 : Ardışıl Prim Algoritması ile Paralel Boruvka Algoritması'nın farklı graflarda çalışma sürelerinin karşılaştırılması

Graf	Düğüm Sayısı	Kenar Sayısı	Ardışıl Prim	Paralel Boruvka
Graf1	6	7	**0.000008 s**	0.000155 s
Graf2	21	29	**0.000031 s**	0.000184 s
Graf3	50	58	0.000288 s	**0.000164 s**

Yukarıdaki deneyler oluşturduğum farklı özellikteki grafların En Küçük Kapsama Ağacını bulmak için kullandığım ardışıl Prim algoritmasının ve Paralel Boruvka algoritmasının çalışma sürelerinin karşılaştırılmasını göstermektedir. Tüm graflar kapalı çevrim (çember) içermeyen (without any cycles), yönsüz (undirected), kenar ağırlıklandırılmış (edge weighted) ve bağlı (connected) graflardır. Graf ayrıntılarına csv dosyalarından bakılabilir. Sonuçlardan gözlemlediğim kadarıyla 3 process kullanarak gerçeklenen Paralel Boruvka algoritması graftaki düğüm sayısı ve kenar sayısı büyüdükçe Ardışıl Prim algoritmasına göre çalışma süresi açısından tercih edilebilir hale gelmektedir. Buna karşılık graf çok küçükken (düğüm ve kenar sayısına bağlı olarak) Ardışıl Prim algoritması'nın kullanılması daha mantıklıdır. Ancak gerçek dünya problemlerinde düğüm ve kenar sayısı çok büyük olduğundan paralelleştirilme kullanılması daha mantıklı olacaktır.

KAYNAKLAR

[1] Algorithms Tutoral Horizon MST, https://algorithms.tutorialhorizon.com/introduction-to-minimum-spanning-tree-mst/, accessed: 2022-05-28.

[2] **Pettie, S. and Ramachandran, V.** (2000). An Optimal Minimum Spanning Tree Algorithm, volume 49, pp.49–60.

[3] Prim Algorithm, https://www.programiz.com/dsa/prim-algorithm, accessed: 2022-05-28.

[4] Boruvka Algorithm, https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm, accessed: 2022-05-28.

[5] Parallel Boruvka & Kruskal, https://github.com/nikitawani07/MST-Parallel/, accessed: 2022-05-28.

[6] MPI Package, https://www.eecis.udel.edu/~pollock/367/manual/node18.html, accessed: 2022-05-28.