Bil 212
Sonbahar 2022
Ödev2
7 Aralık 2022 23:59 max 110puan
15 Aralık 2022 23:59 max 80puan

Görev-Kaynak Çizelgeleme Algoritması(Job-Resource Scheduler)

Bu ödevde daha önceki scheduler sınıfının farklı bir versiyonunu hazırlamanız bekleniyor, JobScheduler. Bu sınıfın amacı girdi olarak gelen işlemleri varış zamanlarına(arrivalTime) göre min-heap veri yapısında depolayıp, uygun bir kaynağa(işlemci çekirdeği yada thread olarak düşünülebilir) yönlendirip çalışma zamanlarını planlamaktır.

Girdiler: **kolonları boşluk ile ayrılmış Jobs.txt ve Dependencies.txt dosyaları**

Jobs.txt // space seperated txt file
JobID duration

Bu dosyanin her satırında ya bir job bilgisi vardır, yada “no job” string i bulunmaktadır.
Id ler sıralı olmak zorunda değildir.

Örnek:
1 3
2 2
no job
3 4
4 1
no job
5 2
6 3
7 2

Dependencies.txt // space seperated txt file
JobID1 JobID2 //JobID2 bitmeden JobID1 başlayamaz.

Bu dosyada Job’lar arasındaki bağımlılıklar verilmiştir. Her bir Job için 0,1 yada birden fazla bağımlılık bulunabilir. Tüm bağımlı olduğu Job’lar bitmeden o Job çalışmaya başlayamaz.

Örnek:
4 3
5 3
7 4
7 5

İşleyiş

Birinci ödevde zaman simulasyonu ve çizelgeyi doldurma aşamaları birbirleri ile kesişmiyordu. Yani ilk önce çizelgeye gelecek bütün job lar alınıp sonra run() metodu ile zaman başlatılıyordu ve bundan sonra çizelgeye job eklenemiyordu. Bu ödevde çizelge çalışması ile ödev kodunuzun çalışması senkron olacaktır. Çalışma zamanı esnasında yeni bir job gelebilir, o job, min-heap içerisine uygun bir şekilde yerleştirilmelidir.

Bu sistemde bir job için iki farklı bloklanma sebebi vardır. Birincisi job ın bağımlılığından kaynaklanır (*dependencyBlocked)*, bağımlı olduğu job henüz bitmediği için çalışamaz. İkincisi kaynakların meşgul olmasından kaynaklanır(*resourceBlocked*), kendisini bağlayan bir şey yoktur fakat kaynakların hepsi meşgul olduğu için beklemektedir.

Size verilen sürücü kodunda ilk yapılacak şey, Dependency.txt dosyasını kullanarak bağımlılıklar için bir hashmap oluşturmaktır.

Ardından bir döngü bulunacaktır. Bu döngüde kontrol edilen koşul, job ların bulunduğu dosyada yeni bir satır bulunup bulunmadığını kontrol eder. Döngüye girildiği takdirde sistem şöyle ilerler:

1. Yeni bir satır bulunması halinde *insertJob( )* metodu **timer** değişkenini bir artırır. Bu yeni değer, gelen satırın varış zamanıdır. Gelen satır yeni bir job ise bu job varış zamanına göre min-heap e depolar. “no job” string i gelmiş ise bir işlem yapmaz. Böylece bir sonraki zaman dilimi için min heap yeniden düzenlenmiş olur
2. *run()* metodu çalıştığında, çalışma süresi bitmiş job ları uygun bir veri yapısına kaydeder. Eğer herhangi bir kaynak müsait olmamışsa bir işlem yapmasına gerek yoktur (burada resourceblocked job lar için ekstra bir şeyler yapmak isterseniz yapabilirsiniz). Fakat job çalıştıracak bir kaynak varsa min-heap ten bir job alır ve bu job ın bağlılıklarını kontrol eder. Eğer bağlı olduğu işlemler henüz bitmemiş ise bu job geçici bir veri yapısına eklenir. Bu işlem bağımlılık problemi kalmamış job gelene kadar devam eder. Böyle bir job geldiğinde gerekli kaynak ataması yapılır ve geçici veri yapısındaki job lar tekrar min heap e gönderilir.

Döngüye girilmemesi yeni bir job gelmeyeceği anlamına gelir. Bu durumda sistemde çalışmakta olan veya sırasını bekleyen job ların bitmesi için *runAllRemaining()* metodu çalışır (Not: timer değişkenin ayarlanması ve job lar için atama yapılması bu metodun en önemli iki bileşenidir). Sistemde herhangi bir job kalmadığında metoddan çıkılır.

Kullanılacak veri yapıları:

min heap: Sisteme gelen job lar, geliş süresine göre burada tutulur.
timer: zaman akışı bu değişken üzerinden gerçekleştirilecektir. Döngüde iken insertJob() ile, döngü dışında da runAllRemaining() ile timer değişkeni ayarlanır.
filePath: IO işlemleri sınıfın içerisinde gerçekleştirilecektir. Sınıfa ait bir nesne oluşturulurken bu değer parametre olarak alınır ve gerekli metotlarda kullanılır.
map: bağımlılıkların tutulacağı veri yapısıdır.

(JobScheduler.java dosyasındaki isimlendirmeleri kullanın)

Bu veri yapılarını kullanmak **zorunludur**. Bunlar haricinde lüzum olduğunu düşündüğünüzde, derste görülen kısımlardaki veri yapılarını kullanabilirsiniz.

Çizelgeleme arayüzü:

public boolean stillContinues( ) Jobs.txt dosyasının devamı olup olmadığını kontrol eder.

public void insertDependencies( ) dependency dosyasını okur ve bağımlılıkları kaydeder.

public void setResourcesCount(Integer) Sınıfın kullanması gereken toplam kaynak(çekirdek) sayısını ayarlar.

public void insertJob( ) Job dosyasından yeni bir satır okur. Bu satır bir job ve onun süresini içerebilr ya da herhangi bir job çizelgeye eklenmeyeceğine dair “no job” bilgisini içerebilir. Bu bilgiye göre ekleme yapar yada yapmaz.
Aynı zamanda timer değişkenini bir artırır.

public void run( ) Çalışma zamanı gelmiş ve bağımlılığı olmayan job ların kaynak atamasını gerçekleştirir. (İşleyiş kısmının ikinci maddesine bakınız)

public void completedJobs( ) O ana kadar çalıştırılıp bitirilen işlemleri ekrana yazdırır.

public void dependencyBlockedJobs( ) Bağımlılığı bulunan job/joblar henüz bitmediği için bekleyen job ları ekrana yazdırır.

public void resourceBlockedJobs( ) Boş kaynak olmadığı için bekleyen job ları ekrana yazdırır.

public void workingJobs( ) Çalışan job/jobları ekrana yazdırır. Örneğin timer değişkeni 5 olduğunda çağırılırsa, sadece 5 te aktif olarak kaynak kullanan job ları yazdırır.

public void allTimeLine( ) Çağırıldığı zaman birimine kadar, hangi zaman biriminde hangi kaynakta hangi job çalıştırıldığını ekrana yazdırır.

public void runAllRemaining( ) Yeni job gelmediği durumda döngüye girilemez. Fakat sistemde hala çalışması gereken job lar bulunabilir. Bu işlemlerin hepsini bitirir (Burada timer değişkenini ayarlayabilir, run() metodunu çağırabilirsiniz.)

Sürücü koddaki yorumlarda istenen ek bilgileri de ekrana bastırmalısınız. Çıktılarda da görebilirsiniz.

Kabuller:

- Bağımlı olunan job, o job ın kendisinden önce varmak zorunda değildir.
- *setResorcesCount() ve insertDependencies()* sürücü kodundaki ana döngü başlamadan önce çalıştırılacaktır.
- *runAllRemaining( )* sürücü kodundaki ana döngüden sonra çalıştırılacaktır.
- Kaynak atama önceliği küçük numaralı kaynağa göre ayarlanır. Örneğin 1 ve 2 numaralı kaynaklardan ikisi de boştayken bir iş geldiğinde kaynak 1 e atama yapılır.
- Min-heap hariç diğer sınıfları kütüphanelerden kullanabilirsiniz (Hashmap, Set gibi)
- Sürücü programı sonsuz döngüye götürecek şekilde bir bağımlılık verilmeyecektir.
- Bir job hem resourceblocked hem dependencyblocked ise, o job ı dependencyblocked a dahil etmelisiniz (Örnek program ve çıktıdaki job 7 gibi).

Örnek Program ve Çıktı:

Aşağıda, tasarlanan sınıfın örnek sürücü kodu gösterilmiştir. **Konsol çıktısı aşağıda verildiği gibi olmalıdır. Başka kelimeler de kullanmayın.**

Sürücü kod:

```
    public static void main(String[] args) {
       String path1 = "pathOfJobFile";
      String path2 = "pathOfDependencyFile";
      JobScheduler cizelge = new JobScheduler(path1);
      cizelge.setResourcesCount(2); //same as one in hw1
      cizelge.insertDependencies(path2);
     while(cizelge.stillContinues()){ // stillContinues metodu ile job dosyasında bir şey
bulunup bulunmadığına bakılır


      cizelge.insertJob(); //insertJob reads a new line from the inputfile, adds a job if
necessary
     System.out.println("min-heap\n" +cizelge); // a proper toString method for JobSchedular
class to print content of heap as tree
  // printing as a tree
    cizelge.run(); //different from one in hw1



    cizelge.completedJobs(); // prints completed jobs
    cizelge.dependencyBlockedJobs(); // prints jobs whose time is up but waits due to its
dependency, also prints its dependency
```

```
   cizelge.resourceBlockedJobs(); // prints jobs whose time is up but waits due to busy
resources
   cizelge.workingJobs(); // prints jobs working on this cycle and its resource
   System.out.println("---------- "+cizelge.timer+" ----------");
   }

   cizelge.runAllRemaining();
  cizelge.allTimeLine();
  }
```

Örnek çıktı: (baştaki örnek dosyası ile çalıştırıldığındaki çıktı)

```
min-heap
1
completed jobs
dependency blocked jobs
resource blocked jobs
working jobs (1,1)
---------- 1 ----------
min-heap
2
completed jobs
dependency blocked jobs
resource blocked jobs
working jobs (1,1) (2,2)
---------- 2 ----------
min-heap
completed jobs
dependency blocked jobs
resource blocked jobs
working jobs (1,1) (2,2)
---------- 3 ----------
min-heap
3
completed jobs 1, 2
dependency blocked jobs
resource blocked jobs
working jobs (3,1)
---------- 4 ----------
min-heap
4
completed jobs 1, 2
dependency blocked jobs (4,3)
resource blocked jobs
working jobs (3,1)
---------- 5 ----------
min-heap
```

4
completed jobs 1, 2
dependency blocked jobs (4,3)
resource blocked jobs
working jobs (3,1)
---------- *6* ----------
min-heap
	4
5
completed jobs 1,2
dependency blocked jobs (4,3) (5,3)
resource blocked jobs
working jobs (3,1)
---------- *7* ----------
min-heap
	4
5		6
completed jobs 1, 2, 3
dependency blocked jobs
resource blocked jobs 6
working jobs (4,1) (5,2)
---------- *8* ----------
min-heap
	6
7
completed jobs 1, 2, 3, 4
dependency blocked jobs (7,5)
resource blocked jobs
working jobs (6,1) (5,2)
---------- *9* ----------
alltimeline
	R1	R2
1	1
2	1	2
3	1	2
4	3
5	3
6	3
7	3
8	4	5
9	6	5
10	6	7
11	6	7

Gönderim:

- Yazdığınız bütün sınıfları *JobScheduler.java* içinde inner class olarak ekleyin. Uzak sistemine sadece *JobScheduler.java* dosyası yüklenecektir.
- Kodunuzun en üstüne yorum olarak isim, soyisim ve numaranızı ekleyin.
- Java 11 ile derleyin.
- Kodunuzun içine yorum ekleyerek anlaşılırlığını kolaylaştırın.
- Derlenmeyen veya yukarıdaki sürücü kodla hatalı çalışan kodlarınız değerlendirilmeyecektir.
- Burada belirtilen kısıtlamalara uymayan kodlardan puan düşülecek.
- Düzgün yazılmış kodlar bonus puan alacaklardır.