Decision Making
DM.A.4: Production scheduling exercises

Msc in Logistics & Supply Chain Management

Submitted by: Jamia Begum

NIU: 1676891

Exercise-1: Code in OPL the scheduling problem stated as exercise 3 in the DM.P.4 list. This model should be named as jobScheduling.mod. The required input data is provided in the enclosed assemblyRules.dat data file.

OPL model:

```
using CP;

{string} ComputerTypes = ...;
{string} ActivityTypes = ...;
{string} ResourceTypes = ...;

// Production work orders
int requiredQuantities[ComputerTypes] = ...;


/*****************************************************
 * An activity consists of
 *   - an activity type,
 *   - a duration,
 *   - a unary resource requirement, and
 *   - a list of precedences.
```

```
*********************************************************/
tuple ActivityData {
  key string  activity;
        int     duration;
        string  requirement; //assigned resoruce
     {string} precedences;
};



{ActivityData} activities[ComputerTypes] = ...;

/*****************************************************************
 * Each particular activity for each computer consist of:
 *   - a defined activity
 *   - for a computer type
 *   - for each computer to be manufatured
 *****************************************************************/
tuple ComputerActivityMatch {
  ActivityData activity;
```

```
  string      computerType;
  int         computer;
};
// All activities that must get scheduled
{ComputerActivityMatch} allActivities = {<a,c,j> | c in ComputerTypes,
                                    a in activities[c],
                                    j in 1..requiredQuantities[c]};
// The activities which must precede activity a
{ComputerActivityMatch} precedences[a in allActivities] = { b | b in allActivities :
                                a.computerType == b.computerType &&
                                a.computer == b.computer &&
                                b.activity.activity in a.activity.precedences };
//decision variables
dvar interval activity[a in allActivities] size a.activity.duration;

dvar sequence resource[r in ResourceTypes] in
  all(a in allActivities: a.activity.requirement==r) activity[a];

// Constraints labels
constraint Precedence[allActivities,allActivities];
```

```
execute {
            cp.param.FailLimit = 1000;
}

dexpr int makespan =max(a in allActivities) endOf(activity[a]);
//the completion time of the last job


minimize makespan;
subject to {
  // Remove symmetries
forall(a1,a2 in allActivities:(a1.activity == a2.activity &&
      a1.computerType == a2.computerType && a1.computer < a2.computer) )

     endBeforeStart(activity[a1], activity[a2]);//end(a1)+z <= start(a2)

  // Resource Requirements
  forall(r in ResourceTypes)
   //no overlapping in the use of each resource
      noOverlap(resource[r]);
```

```
  // Precedences
  //b is the precedence of a, after b finishes a starts
  forall( a in allActivities)
    forall( b in precedences[a])
      Precedence[a,b]: endBeforeStart(activity[b], activity[a]);

};
```

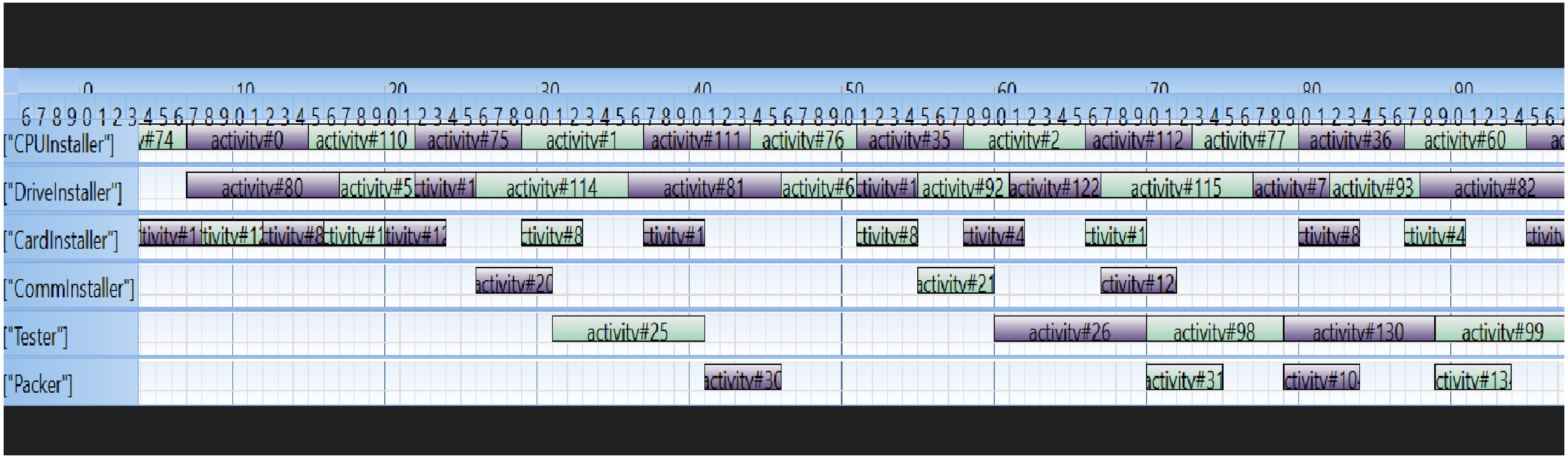
Solution:

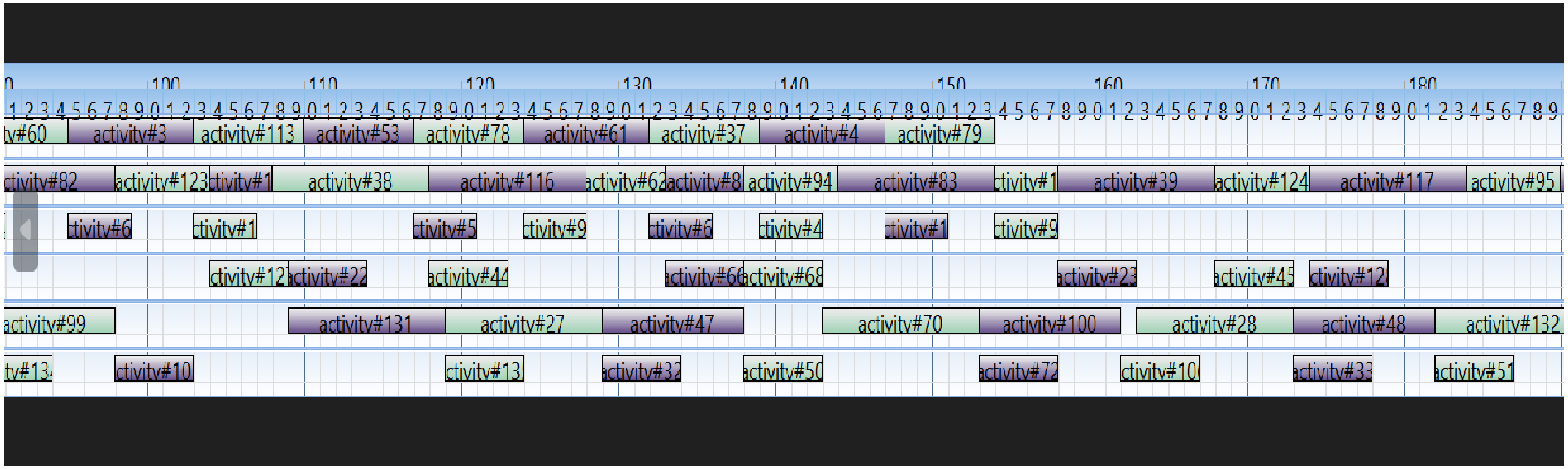
The optimal make span is 282 minutes.

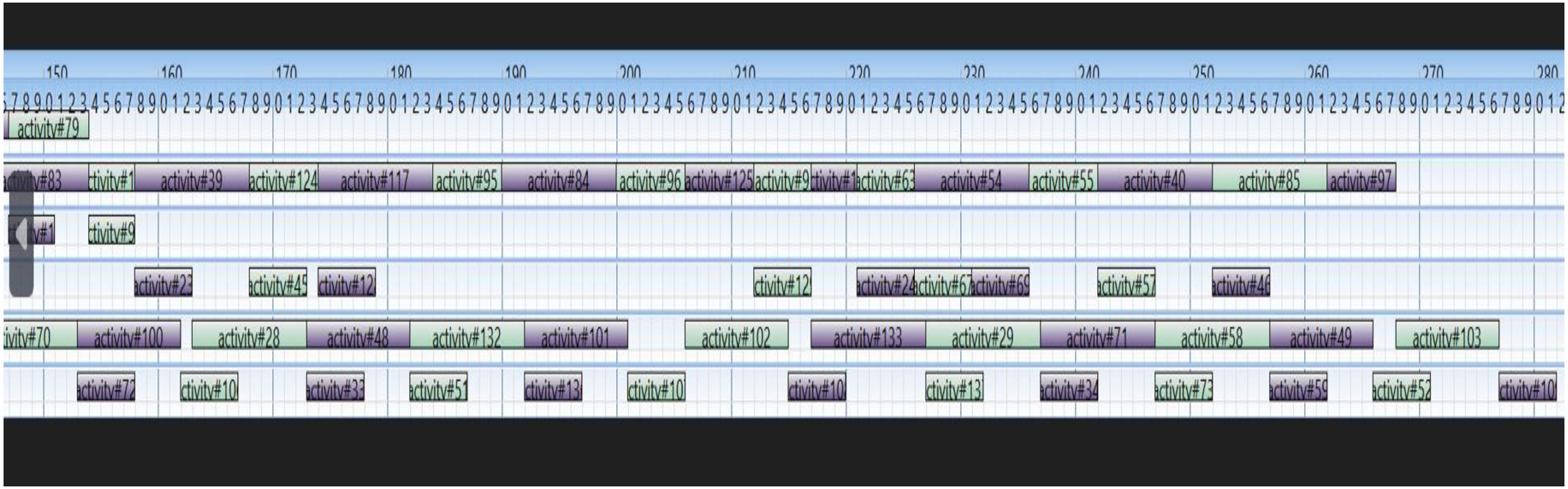
Exercise-2:

Question 2.1: **Analyze the results obtained from Exercise 1. If any, where do you think the bottlenecks are? Justify your answer.**

Ans: All activities are distributed over the resources in the following way:


["CPUInstaller"]
["DriveInstaller"]
["CardInstaller"]
["CommInstaller"]
["Tester"]
["Packer"]
activity#0
activity#110
activity#75
activity#1
activity#111
activity#76
activity#35
activity#2
activity#112
activity#77
activity#36
activity#60
activity#80
activity#114
activity#81
activity#92
activity#122
activity#115
activity#93
activity#82
activity#20
activity#21
activity#25
activity#26
activity#98
activity#130
activity#99


activity#3
activity#113
activity#53
activity#78
activity#61
activity#37
activity#4
activity#79
activity#123
activity#38
activity#116
activity#94
activity#83
activity#39
activity#124
activity#117
activity#95
activity#22
activity#44
activity#68
activity#23
activity#45
activity#99
activity#131
activity#27
activity#47
activity#70
activity#100
activity#28
activity#48
activity#132
activity#32
activity#50
activity#72
activity#33
activity#51



A bottleneck is any resource whose capacity is less than the demand placed on it. Here, resource-2(DriveInstaller) is a bottleneck in this problem since it's densely allocated.

Question 2.2: **Extend your model from Exercise 1 to enable several machine units of the same resource type. The new required data set and structures are given to you in the assemblyRules.v2 data file and jobScheduling.v2 model template respectively. Neither data sets nor data structures should be modified. You must complete the decision variable declarations**

and adapt the constraints from Exercise 1 and add the new constraint as indicated in the model template.

OPL Model:

```
using CP;

{string} ComputerTypes = ...;
{string} ActivityTypes = ...;
{string} ResourceTypes = ...;

// Production work orders
int requiredQuantities[ComputerTypes] = ...;


/**********************************************
 * An activity consists of
 *   - an activity type,
 *   - a duration,
 *   - a unary resource requirement, and
 *   - a list of precedences.
```

```
 ************************************************/
tuple ActivityData {
  key string  name;
        int     duration;
        string  requirement;
     {string} precedences;
};

{ActivityData} activities[ComputerTypes] = ...;

/*******************************************************
 * Each particular activity for each computer consist of:
 *   - a defined activity
 *   - for a computer type
 *   - for each computer to be manufatured
 *******************************************************/
tuple ComputerActivityMatch {
  ActivityData activity;
  string      computerType;
  int         computer;
};
```

```
// All activities that must get scheduled
{ComputerActivityMatch} allActivities = {<a,c,j> | c in ComputerTypes,
                                          a in activities[c],
                                          j in 1..requiredQuantities[c]};
// The activities which must precede an activity
{ComputerActivityMatch} precedences[a in allActivities] = { b | b in allActivities :
                                    a.computerType == b.computerType &&
                                    a.computer == b.computer &&
                                    b.activity.name in a.activity.precedences };

/*****************************************************
 * Resource data consists of:
 *  - ResourceType
 *  - Number of available resorces of this type
 *****************************************************/
tuple ResourceData {
      key string resourceType;
      int available;
}

// for describing each resource (machine)
```

```
tuple ResourceUnit{
      string resourceType;
      int unit;
}

/***********************************
 * Reads the defined resource units
 * for each resource type
 ***********************************/
{ResourceData} resources = ...;

{ResourceUnit} availableResources = {<r.resourceType, i> | r in resources, i in 1..r.available};

/*******************************************
 * Represent the pairs activity<->resouce unit
 *******************************************/
tuple JobAllocation {
       ComputerActivityMatch job;
       int machineId;
}
```

```
/****************************************
 * Define the domain for the different
 * choices when assigning a job to a machine
 ****************************************/
{JobAllocation} jobAllocations = {<job, unit> |
                                  job in allActivities, r in resources, unit in 1..r.available :
                                   job.activity.requirement == r.resourceType};

dvar interval activity[a in allActivities] size a.activity.duration;
dvar interval jobAllocation[j in jobAllocations]optional size j.job.activity.duration ; // complete the
dvar declaration accordingly
dvar sequence resource[r in availableResources] in all(j in jobAllocations:
 j.job.activity.requirement == r.resourceType && j.machineId == r.unit)jobAllocation[j]; // complete
the dvar declaration accordingly


// Constraints labels
constraint Precedence[allActivities,allActivities];

execute {
        cp.param.FailLimit = 100000;
```

```
}

dexpr int makespan = max(a in allActivities) endOf(activity[a]);
//the completion time of the last job
// Complete makespan expression;

minimize makespan;
subject to {
  // Remove symmetry
  //avoiding symmetric solutions
       forall(a1,a2 in allActivities:(a1.activity == a2.activity &&
       a1.computerType == a2.computerType && a1.computer < a2.computer) )

     endBeforeStart(activity[a1], activity[a2]);//end(a1)+z <= start(a2)

  // Each activity is performed only once
  forall(a in allActivities)
     //using Alternative constraint for allocating job to activity
     //then one activity is performed exactly once
  alternative(activity[a], all(j in jobAllocations:j.job.activity == a.activity &&
 j.job.computerType ==  a.computerType &&
```

```
            j.job.computer == a.computer) jobAllocation[j]);


  // Resource Requirements
  forall(r in availableResources)
   //no overlapping in the use of each resource
       noOverlap(resource[r]);

  // Precedences
   forall( a in allActivities)
    forall( b in precedences[a])
     //b is the precedence of a, after b finishes a starts
     Precedence[a,b]: endBeforeStart(activity[b], activity[a]);

};
```

Question 2.3: **Determine the increase of the selected resources that is required for reducing the make span to less than 190 minutes for the given work orders.**

Ans:

Increasing the resource units for DriveInstaller and Tester by 1, the makespan becomes less than 190 minutes.

```
resources = {
  <CPUInstaller, 1>,
  <DriveInstaller, 2>,
  <CardInstaller, 1>,
  <CommInstaller, 1>,
  <Tester, 2>,
  <Packer, 1>
```