DM534 — Øvelser Uge 45

Introduktion til Datalogi, Efterår 2021

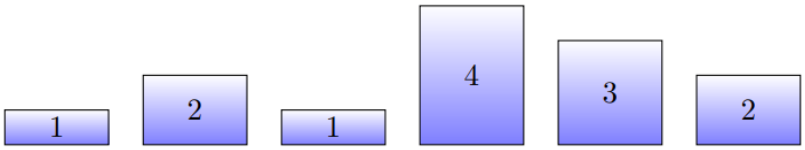
Jonas Vistrup og Rolf Fagerberg

---

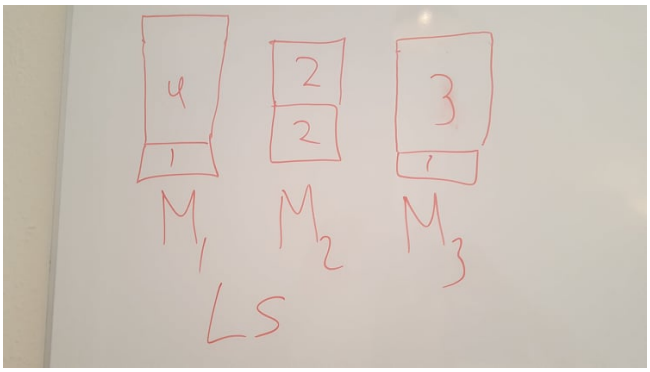
1 I

1.1

For $m = 3$, which schedule does the List Scheduling algorithm, Ls, produce on the following input sequence:



SVAR:



1.2

In the lecture, we proved that the machine scheduling algorithm, Ls, could not perform better than $2 - \frac{1}{m}$. We now consider only two machines. Thus, $m = 2$, and the ratio is then $\frac{3}{2}$. Just because Ls cannot perform better, it could be that some other algorithm could. Prove (for $m = 2$) that this is not the case. You must design an input, where no algorithm, no matter what decisions it makes, can do better than $\frac{3}{2}$ times Opt. You only need sequences with three jobs and a case analysis with only two cases, depending on what an algorithm does with the second job that is given.

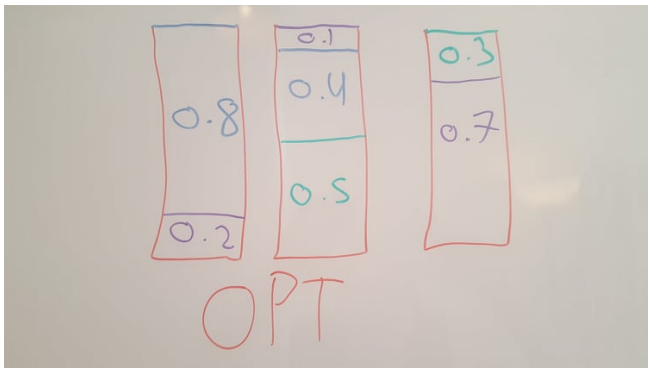
SVAR: Consider three jobs where the first two have sizes 3 and 3. When the second job arrives, the online algorithm has to make a decision on where to put it. A given algorithm can either put it on the same machine as the first job or on the other machine.

For an algorithm that puts it on the same machine, we give third job size 1. Then the ratio ALG/OPT is $\frac{6}{4} = \frac{3}{2}$, since the algorithm finishes at time 6, while it is possible to finish at time 4 (by putting the first two jobs on different machines), hence this is what OPT can achieve.

For an algorithm that puts it on the other machine, we give the third job size 6. Then the ratio ALG/OPT is $\frac{9}{6} = \frac{3}{2}$, since the algorithm finishes at time 9, while it is possible to finish at time 6 (by putting the first two jobs on the same machine), hence this is what OPT can achieve.

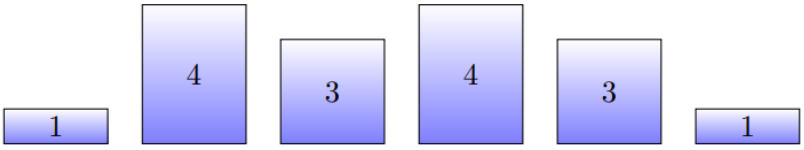
1.3

Consider the first bin packing example given in the lecture (slide 19), where the First-Fit algorithm, FF, uses four bins. Show that OPT only needs three.

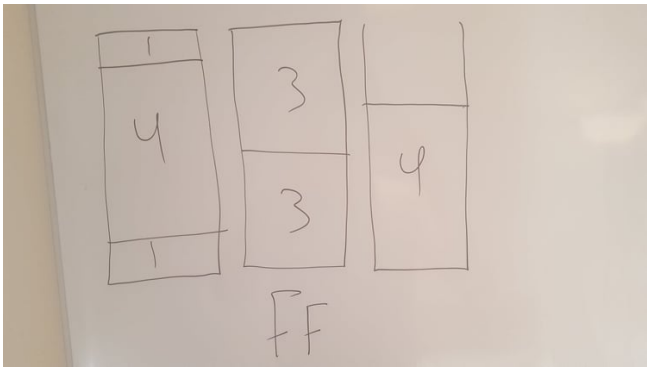
SVAR: 

1.4

How does the First-Fit algorithm, FF, behave on the input sequence below? Item sizes are given in multiples of $\frac{1}{6}$.

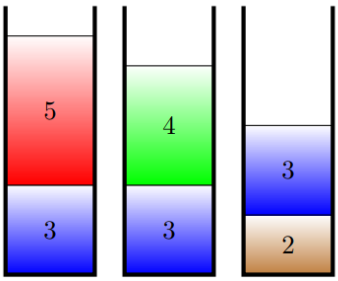


SVAR:



1.5

Why can the following configuration *not* have been produced by the bin packing algorithm FF? Item size are given in multiples of $\frac{1}{9}$.



SVAR: Here at the end, there is room for 2 in the second bin. Then 2 clearly could have been place in the second bin at the time that 2 arrived (since the second bin could only have been less filled at that time). Hence, the placement of 2 in the third bin is in contradiction with the definition of FF.

1.6

Prove that no matter which other algorithm than the one from the lecture slides we define for ski rental, the algorithm will perform worse, i.e., the competitive ratio will be strictly higher than $\frac{19}{10}$. (As explained in the lecture slides, any algorithm is of the form **Buy on day X**, where the choice of **X** determines the algorithm).

Start by analyzing the algorithms **Buy on day 5** and **Buy on day 15** to see what happens. The skis still cost 10 units to buy and 1 unit per day to rent.

SVAR: For the algorithm **Buy on day 5** we can try all possible inputs (all possible days 1, 2, 3,… for leaving) and then see that for this algorithm, the worst ratio arises for the input leave after 5 days, for which the ratio is ALG/OPT $= \frac{14}{5} = 2.8$.

In full details: For leaving after 1 day, the algorithm has cost 1 (one day of renting) and the optimal decision for this input is to have only rented (like the algorithm does, so it behaves optimally for this particular input). Hence, the ratio for this input is ALG/OPT $= \frac{1}{1} = 1$. For leaving after 2 days, the algorithm has cost 2 (two days of renting) and the optimal decision

for this input is to have only rented (like the algorithm does, so it behaves optimally also for this particular input). Hence, the ratio for this input is ALG/OPT $= \frac{2}{2} = 1$. This repeats up to and including the input of leaving after 4 days. For leaving after 5 days, the algorithm has cost $4 + 10 = 14$ (four days of renting and then buying) while the optimal decision for this input is to have only rented (now different from what the algorithm does), which has a cost of 5. Hence, the ratio for this input is ALG/OPT $= \frac{14}{5} = 2.8$. For leaving after 6 days, the algorithm still has cost $4 + 10 = 14$ while the optimal decision is to have only rented, which has a cost of 6. Hence, the ratio is ALG/OPT $= \frac{14}{6} = 2.333\ldots$. For leaving after 7 days, the algorithm has cost $4 + 10 = 14$ while the optimal decision is to have only rented, which has a cost of 7. Hence, the ratio is ALG/OPT $= \frac{14}{7} = 2$. This continues with decreasing ratios until the input is leaving after 10 days, for which the algorithm still has cost $4 + 10 = 14$, while the optimal decision now changes to buy at day 1, which has cost 10 (for this particular input, renting 10 days would also be optimal). Hence, the ratio is ALG/OPT $= \frac{14}{10} = 1.4$. For inputs of leaving after more than 10 days, nothing changes from the previous case (except that buying at day 1 is the only possible optimal decision), and the ratio stays at 1.4. In summary, for the algorithm **Buy on day 5**, the worst case ratio over all inputs is 2.8.

For the algorithm **Buy on day 15** we can also try all possible inputs (all possible days 1, 2, 3,… for leaving) and then see that for this algorithm, the worst ratio arises for the input leave after 15 days, for which the ratio is ALG/OPT $= \frac{24}{10} = 2.4$. Calculations are very similar to those above.

As the day that the algorithm buys (i.e., the value of X) approaches day 10 either from below or from above, the same type of analyses show that the worst case ratio approaches $\frac{19}{10} = 1.9$ from above, but never reaches it until the buying day X of the algorithm is 10.

Summing up, we have seen that for the ski rental problem with a cost structure of rent one day = 1, buy skis = 10, the best online algorithm (the one with the lowest competitive ratio) is **Buy on day 10**.

2 II

2.1

For bin packing, one can prove the upper bound that FF is 1.7-competitive. However, this is a quite hard proof. In this exercise, we will try to improve (raise) the lower bound.

In the lecture, we saw an example demonstrating that FF can be as bad as $\frac{3}{2} = 1.5$ times OPT.

Let that example inspire you, and try to use items of the following three sizes:

$$\frac{1}{7} + \frac{1}{1000},\ \frac{1}{3} + \frac{1}{1000},\ \frac{1}{2} + \frac{1}{1000}$$

Find a sequence where FF performs $\frac{5}{3}$ times worse than OPT.

SVAR: Similar to the example in the slides, we consider inputs starting with a number of items of the smallest size, then of the next size, and then of the largest size.

Specifically, we consider an input of n items of size $1/7 + 1/1000$, then n items of size $1/3 + 1/1000$, and finally n items of size $1/2 + 1/1000$. For simplicity of argument, we only consider n's which are multiples of $(2-1)\cdot(3-1)\cdot(7-1) = 12$.

There can be $7 - 1 = 6$ of the smallest elements in a bin, so algorithm FF will fit the first n elements into $n/(7-1) = n/6$ bins (exactly $n/6$ bins, since n is a multiple of 6). It will then fit the next n elements into $n/(3-1) = n/2$ bins, by a similar argument. Finally, it will fit the last n elements into n bins. Thus, the cost of FF on this sequence is $n(1/6 + 1/2 + 1) = n(1 + 3 + 6)/6 = 5n/3$.

However, one can collect items into n groups, with each group containing one item of each size (1/7+1/1000, 1/3+1/1000, and 1/2+1/1000). Since 1/7+1/3+1/2 = (6+14+21)/42 = 41/42 and $1/1000 + 1/1000 + 1/1000 < 1/42$, each group can be in a single bin. So the cost of OPT cannot be worse than n.

So on inputs of this type, the ratio FF/OPT is at least $(5n/3)/n = 5/3 = 1.666\ldots$. Hence, the competitive ratio of FF cannot be better than this.

Now try using

$$\frac{1}{43} + \frac{1}{10000},\ \frac{1}{7} + \frac{1}{10000},\ \frac{1}{3} + \frac{1}{10000},\ \frac{1}{2} + \frac{1}{10000}$$

to get a lower bound even closer to the 1.7 upper bound.

SVAR: We repeat the construction from above, just with one item size more.

Specifically, we consider an input of n items of size $1/43 + 1/10000$, then n items of size $1/7 + 1/10000$, then n items of size $1/3 + 1/10000$, and finally n items of size $1/2 + 1/10000$. For simplicity of argument, we only consider n's which are multiples of $(2-1) \cdot (3-1) \cdot (7-1) \cdot (43-1) = 504$.

There can be $43 - 1 = 42$ of the smallest elements in a bin, so algorithm FF will fit the first n elements into $n/(43-1) = n/42$ bins (exactly $n/42$ bins, since n is a multiple of 42). It will then fit the next n elements into $n/(7-1) = n/6$ bins, and then the next n elements into $n/(3-1) = n/2$ bins, by a similar argument. Finally, it will fit the last n elements into n bins. Thus, the cost of FF on this sequence is $n(1/42 + 1/6 + 1/2 + 1) = n(1 + 7 + 21 + 42)/42 = 71n/42$.

However, one can collect items into n groups, with each group containing one item of each size (1/43+1/10000 1/7+1/10000, 1/3+1/10000, and 1/2+1/10000). Since 1/43+1/7+1/3+1/2 = (42+258+602+903)/1806 = 1805/1806 and 1/10000+1/10000+1/10000+1/10000 < 1/1806, each group can be in a single bin. So the cost of OPT cannot be worse than n.

So on inputs of this type, the ratio FF/OPT is at least $(71n/42)/n = 71/42 = 1.6904\ldots$. Hence, the competitive ratio of FF cannot be better than this.

2.2

It is very easy to implement FF in Java, if there are no efficiency requirements: just use an array to hold the current level in the bins, and for each item, search for the first bin with enough space. If you make sure there are enough bins from the beginning, then there are no special cases. And you simple count the number of non-empty bins at the end to get the result.

Implement FF.

Try to define your own algorithm, from scratch or as a variant of FF.

Test your own algorithm up against FF and try to determine which one is best; for instance on uniformly distributed sequences.

In Java, one way to create pseudorandom floating point numbers uniformly distributed in the interval [0, 1[is via the class **java.util.Random** and its **NextFloat** method. You may want to look at the tutorial here: **http://www.functionx.com/java/Lesson18.htm**. If you want to generate the same sequence of pseudorandom numbers in different invocations of your program (for instance to compare two algorithms on the same input sequences), you must set the seed in the random number generator at the start of the program (otherwise it is automatically set to a different seed at each invocation). This is also described in the tutorial.