Scheduling jobs on two machines

Table of Contents

This task is about finding an optimal schedule for n jobs on two machines. Every item must first be processed on the first machine, and afterwards on the second one. The i-th job takes a_i time on the first machine, and b_i time on the second machine. Each machine can only process one job at a time.

We want to find the optimal order of the jobs, so that the final processing time is the minimum possible.

This solution that is discussed here is called Johnson's rule (named after S. M. Johnson).

It is worth noting, that the task becomes NP-complete, if we have more than two machines.

Construction of the algorithm

Note first, that we can assume that the order of jobs for the first and the second machine have to coincide. In fact, since the jobs for the second machine become available after processing them at the first, and if there are several jobs available for the second machine, than the processing time will be equal to the sum of their b_i, regardless of their order. Therefore it is only advantageous to send the jobs to the second machine in the same order as we sent them to the first machine.

Consider the order of the jobs, which coincides with their input order $1, 2, \ldots, n$.

We denote by x_i the **idle time** of the second machine immediately before processing i. Our goal is to **minimize the total idle time**:

$$F(x) = \sum x_i \ \rightarrow \min$$

For the first job we have $x_1 = a_1$. For the second job, since it gets sent to the machine at the time $a_1 + a_2$, and the second machine gets free at $x_1 + b_1$, we have $x_2 = \max\left((a_1 + a_2) - (x_1 + b_1), 0\right)$. In general we get the equation:

$$x_k = \max\left(\sum_{i=1}^{k} a_i - \sum_{i=1}^{k-1} b_i - \sum_{i=1}^{k-1} x_i, 0\right)$$

We can now calculate the **total idle time** $F(x)$. It is claimed that it has the form

$$F(x) = \max_{k=1\dots n} K_i,$$

where

$$K_i = \sum_{i=1}^{k} a_i - \sum_{i=1}^{k-1} b_i.$$

This can be easily verified using induction.

We now use the **permutation method**: we will exchange two neighboring jobs j and $j+1$ and see how this will change the total idle time.

By the form of the expression of K_i, it is clear that only K_j and K_{j+1} change, we denote their new values with K'_j and K'_{j+1}.

If this change from of the jobs j and $j+1$ increased the total idle time, it has to be the case that:

$$\max(K_j, K_{j+1}) \le \max(K'_j, K'_{j+1})$$

(Switching two jobs might also have no impact at all. The above condition is only a sufficient one, but not a necessary one.)

After removing $\sum_{i=1}^{j+1} a_i - \sum_{i=1}^{j-1} b_i$ from both sides of the inequality, we get:

$$\max(-a_{j+1}, -b_j) \le \max(-b_{j+1}, -a_j)$$

And after getting rid of the negative signs:

$$\min(a_j, b_{j+1}) \le \min(b_j, a_{j+1})$$

Thus we obtained a **comparator**: by sorting the jobs on it, we obtain an optimal order of the jobs, in which no two jobs can be switched with an improvement of the final time.

However you can further **simplify** the sorting, if you look at the comparator from a different angle. The comparator can be interpreted in the following way: If we have the four times $(a_j, a_{j+1}, b_j, b_{j+1})$, and the minimum of them is a time corresponding to the first machine, then the corresponding job should be done first. If the minimum time is a time from the second machine, then it should go later. Thus we can sort the jobs by $\min(a_i, b_i)$, and if the processing time of the current job on the first machine is less then the processing time on the second machine, then this job must be done before all the remaining jobs, and otherwise after all remaining tasks.

One way or another, it turns out that by Johnson's rule we can solve the problem by sorting the jobs, and thus receive a time complexity of $O(n \log n)$.

Implementation

Here we implement the second variation of the described algorithm.

```
struct Job {
    int a, b, idx;
    bool operator<(Job o) const {
        return min(a, b) < min(o.a, o.b);
    }
};

vector<Job> johnsons_rule(vector<Job> jobs) {
    sort(jobs.begin(), jobs.end());
    vector<Job> a, b;
    for (Job j : jobs) {
        if (j.a < j.b)
            a.push_back(j);
        else
            b.push_back(j);
    }
    a.insert(a.end(), b.rbegin(), b.rend());
    return a;
}

pair<int, int> finish_times(vector<Job> const&
    int t1 = 0, t2 = 0;
    for (Job j : jobs) {
        t1 += j.a;
        t2 = max(t2, t1) + j.b;
    }
    return make_pair(t1, t2);
}
```

All the information about each job is store in struct. The first function sorts all jobs and computes the optimal schedule. The second function computes the finish times of both machines given a schedule.