This problem is a combination of 0/1 Knapsack and topological sorting.
My first approach was to apply brute force and generate all possible valid blocks.
A valid block is one which in which:

- All parent tasks are performed before dependent children tasks
- The total transaction weight does not exceed 4000000

The brute force algorithm recursively tries to keep adding tasks such that all parent tasks of the new task being added have been performed and the addition of the weight of this task to the present weight does not make the present weight exceed 4000000.

Brute Force algorithm
Parameters : list of tasks already performed -> tlist
Current fee -> f
Current weight -> w
Indegrees of tasks denoting how many of the parent tasks of a task have been performed
Current task to add -> task

```
If indegree[task] == 0
    If w+weight[task]<=4000000
        F = f+fee[task]
        Update indegrees of child tasks of task
        Recurse till w>4000000 or no task left
```

TRADE OFF:
The brute force approach has an exponential time complexity which makes it impractical to use for present big volumes of data although it guarantees the best result.

Optimised approach
In my optimised approach I do the following:

- I generate a list of valid tasks that are tasks who have no parents or all of whose parents have been already performed.
- I perform a 0/1 knapsack to get the best possible fee from this set of valid tasks.
- From the knapsack computation above I get a list of jobs that will be performed, the maximum fee obtained from these jobs, and the total transaction weight needed to perform these jobs. This is a classical 0/1 knapsack problem where weight is optimised to get the best possible fee.
- I add the jobs performed to my final block and set their indegrees to -1
- I add the fee obtained to my final fee.
- I update my max transaction weight as max transaction weight - transaction weight of performing the above jobs.
- I then pick up the jobs performed one by one and update the indegrees of their child jobs by decrementing independence by 1.
- I then iterate through the list of jobs and add all jobs that have an indegree (updated) of 0 to the next set of valid jobs.
- I then again perform knapsack on this set of updated valid jobs.
- I continue the process till the max transaction weight == 0 or till there are no jobs left to perform.

Worst Time Complexity : O(m*n*n)
Where m = maximum number of parents, n = number of jobs

As m<=(n-1) the max time complexity is O(n^3) as compared to exponential time complexity of brute force approach

Trade Off :
It may be that a certain job with an indegree of 0 but a lesser advantage in terms of fee and transaction weight gets performed over another job with an indegree not equal to 0 but greater fee advantage. This is because the algorithm performs tasks that have an indegree of 0 before tasks that have indegree>0. So a task can be performed only once all its parents have been performed. Thus it may be that this task is a better pick than another task but gets pushed into the list of valid jobs later than another less advantageous job with indegree=0.
But given the ratio of total number of tasks to the maximum number of parents that a task can have (~5200/4 = 1300) this is an acceptable tradeoff as the disadvantage of this algorithm in terms of fee calculation is negligible while the time gain is very significant.