Module 7 Homework - Divide and Conquer

Job scheduling in the gig economy

In the gig economy, companies rely on independent contractors rather than on traditional employees. We are going to consider the situation from the contractor's point of view - what jobs should we select to make the most money?

A job offer has three components:

- start time
- finish time
- hourly wage

A contractor is allowed to accept an offer in its entirety, or for parts of the duration, even going back and forth between jobs. That is, we can switch between jobs at will.

Accepting one or more job offers defines a **planning**, which is a list of tuples. Each tuple contains two components - the time at which we start the job, and the wage for that job.

```
job1 = [(1, 5, 10)] # a job that runs from 1-5, and pays $10/hr
job2 = [(2, 3, 20)] # a job that runs from 2-3, and pays $20/hr
opt_plan = [(1,10), (2, 20), (3, 10), (5, 0)] # optimal planning
```

Given a list of job offers, our goal is to determine the optimal planning.

Divide-and-conquer algorithm

A first idea is to add the jobs to the planning one-by-one. However, as a job can potentially overlap with all the other jobs, this would lead to a quadratic running time.

A better idea is to use a divide-and-conquer strategy, inspired by mergesort. We will divide the set of job offers in two, recursively contruct the plannings for each half, and finally merge the two. If two plannings can be merged in linear time, and we have a total of log2(n) levels of splitting jobs in two, this would lead to an algorithm with *O*(n log n) running time.

Part 1 - Merge plannings

For the divide-and-conquer strategy to be efficient, we need to merge two plannings in linear time.

Requirements

A planning is a list of time-wage pairs at which the wages change, ordered by ascending times.

```
[(2, 3), (1, 5), (5, 4), (7, 0)]    # invalid: time stamps not in order
[(1, 5), (2, 3), (5, 4), (7, 0)]    # valid
```

A planning must not contain any redundancy: No two adjacent pairs can have the same hourly wages.

```
[(1, 6), (2, 3), (5, 3), (7, 0)]    # invalid: redundancy
[(1, 6), (2, 3), (7, 0)]            # valid
```

Deliverables

Write a function that takes two plannings and merges them in linear time.

- `merge_plannings(p1, p2)`

  ```
  >>> p1 = [(2, 4), (5, 0), (6, 3), (10, 0)]
  >>> p2 = [(3, 5), (7, 0), (9, 2), (13, 0)]
  >>> merge_plannings(p1, p2)
  [(2, 4), (3, 5), (7, 3), (10, 2), (13, 0)]
  ```

Remember, the goal is to maximize our total wage.

Part 2 - Construct an optimal planning

We will now use the linear merge to implement the divide-and-conquer algorithm that computes the optimal planning.

A single job offer defines a planning as follows:

```
[(1, 6, 4)]       # job offer
[(1, 4), (6, 0)]  # corresponding planning
```

A list of `n` jobs will be divided in two. The problem will be solved recursively for the two halves and the results will be merged using the linear function defined above. The job offers cannot be assumed to be sorted in any particular order.

Deliverables

Write a recursive function that takes a list of job offers and returns an optimal planning in $O(n \log n)$ time.

- `optimal_planning(jobs)`

  ```
  >>> jobs = [(2, 8, 3), (5, 7, 5)]
  >>> optimal_planning(jobs)
  [(2, 3), (5, 5), (7, 3), (8, 0)]
  ```

Submission

At a minimum, submit the following file with the functions noted:

- `optimal_planning.py`
  - `merge_plannings()` - function
  - `optimal_planning()` - function

Students must submit to Mimir **individually** by the due date (typically, the second Wednesday after this module opens at 11:59 pm EST) to receive credit.

Grading

- 50 - `merge_plannings`
  - 20 - linear running time
  - 30 - functionality
- 50 - `optimal_planning`
  - 10 - uses recursion
  - 10 - n log n running time
  - 30 - functionality

Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly. It has resulted in:

- A simplified, clear **Submitting** section on all assignments
- A simplified, clear **Grading** section on all assignments
- Clearer instructions on assignments