CS 2051: Honors Discrete Mathematics
Spring 2023 Homework 5 Supplement

Sarthak Mohanty

Overview

Note: This is a minimal working copy. It's missing a lot of context, and is only meant to help you get started with the homework early. I'll be updating it a lot more tomorrow with detailed instructions, more examples, and better formatting.

Traditionally, computer software has been written for serial computation: instructions corresponding to a tasks are executed on one processing unit at a time. However, nowadays many computational tasks consist of many elementary operations, some of which had to be computed sequentially, while others could be computed in parallel.

In this supplement, you learn about a generalized form of functions known as **relations**, and explore the connection between functions and relations. You'll learn about equivalence relations and partial orders. You'll also explore some of the applications of these concepts

Part 1: Relations and Graphs

We begin with (binary) relations, a mathematical representation for associating elements of sets.

Definition a *relation* over sets A, B is a subset of $A \times B$. The notation $a\mathcal{R}b$ or $a \sim b$ is often used to denote that $(a, b) \in \mathcal{R}$.

We denote relations by $\mathcal{R}$. We write $a\mathcal{R}b$ to indicate that $(a, b) \in \mathcal{R}$ (i.e.: in the subset denoted by $\mathcal{R}$). When $A = B$ we said that $\mathcal{R}$ is a relation on A.

Examples.
Let $\mathcal{R}_1, \ldots, \mathcal{R}_4$ be a relation on $A = \{1, 2, 3, 4\}$.

- $\mathcal{R}_1 = \{(a, b) \mid a \leq b)\}$
- $\mathcal{R}_2 = \{(a, b) \mid a = b)\}$
- $\mathcal{R}_3 = \{(a, b) \mid a + b \leq 2022)\}$
- $\mathcal{R}_4 = \{(a, b) \mid a \text{ divides } b\}$

So why relations? They are more general and allow us to study more complex sets. Elaborate.

Properties

- Reflexive: $(\forall a \in A)(a\mathcal{R}a)$
- Symmetric: $(\forall a, b \in A)(a\mathcal{R}b \iff b\mathcal{R}a)$
- Antisymmetric: $(\forall a, b \in A)(a\mathcal{R}b \wedge b\mathcal{R}a \to a = b)$
- Transitive: $(\forall a, b, c \in A)(a\mathcal{R}b \wedge b\mathcal{R}c \to a\mathcal{R}c)$

When a relation is reflexive, antisymmetric, and transitive, we call it a *partial order*.
When a relation is reflexive, symmetric, and transitive, we call it a *equivalence relation*.

In this part, you'll implement the functions `isPartialOrder(elements, relation)` and `isEquivalenceRelation(elements, relation)`. These functions takes in a relation (represented as a list of tuples) and returns whether or not the relation (taken over the set of elements) is a valid partial order or equivalence relation, respectively. You must use the following helper methods:

- `isReflexive(elements, relation)`
- `isSymmetric(elements, relation)`
- `isAntisymmetric(elements, relation)`
- `isTransitive(elements, relation)`

All methods (and all helper methods) must be implemented in one line. Hint: use `all` method

Part 2: Partitioning with Equivalence Relations

In the world of computer science, there are two main applications for relations: partitioning and scheduling. In this part, we'll cover partitioning, which is essentially just a reframing of our knowledge about equivalence relations.
If any of the concepts introduced in this section feel rather hand-wavy, feel free to consult the textbook, which has rigorous proofs for the theorems.

Definition: Given some relation $\mathcal{R}$ over the set A, the *equivalence class* of an element $x \in A$ is

$$[x] = \{y : x\mathcal{R}y\}$$

There is a very powerful theorem related to this concept:

Important Theorem: The equivalence classes of an equivalence relation on a set A *partition* A into a collection of disjoint, nonempty subsets $A_1, A_2, \ldots, A_n$ such that (and this is the important part) $\bigcup_{i=1}^{n} = A$.

Example 1: Congruence Relations

Our first example delves into *number theory*, a field you will become more intimate with in the next few supplements. Informally, define the relation $a\mathcal{R}_N b$ over $\mathbb{Z} \times \mathbb{Z}$ if a and b have the same remainder when divided by some number n. Another way to say this is

$$a \equiv b \pmod{n}.$$

All such relations $\mathcal{R}_n$ are equivalence relations, and partition the set of integers into n equivalence classes. For example, the relation $\mathcal{R}_3$ partitions the integers like so

$$\{\ldots, -8, -5, -2, 1, 4, 7, \ldots\} \tag{1}$$
$$\{\ldots, -7, -4, -1, 2, 5, 8, \ldots\} \tag{2}$$
$$\{\ldots, -6, -3, 0, 3, 6, 9, \ldots\} \tag{3}$$

In this part, you'll implement the following function:

- `partition(elements, relation)`: This function takes a equivalence relation (this time represented as a boolean function), and returns a partition of the elements into equivalence classes. For example, given the congruence relation described above, the function should return

```
>>> partition([i for i in range(-8, 8)], lambda x, y: (x - y) % 3 == 0)
[{-6, -3, 0, 3, 6}, {-8, -5, -2, 1, 4, 7}, {-7, -4, -1, 2, 5, 8}]
```

Part 3: Single Processor Job Scheduling

Job Scheduling

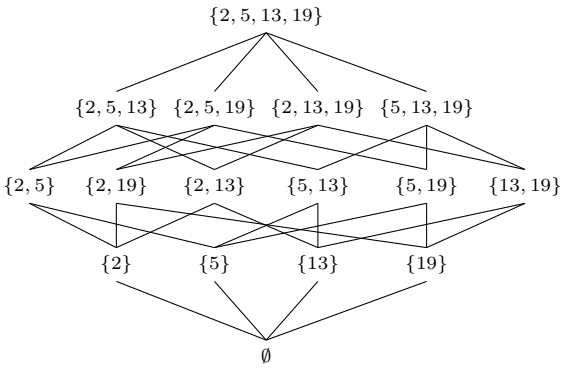

Figure 1: Dependency graph of poset $(\mathcal{P}(\{2, 5, 19, 13\}), \subseteq)$.

Using the formalism of posets, we can now define the problem of computing an optimal schedule. We do so in the language of job scheduling, as it's the most common form.

Assume that some task T can be deocmposed into sub-tasks $T = \{T_1, T_2, \ldots, T_n\}$. In general, we can encode the relationships between the various T_i as a poset, where $T_i \prec T_j$ if (sub)task T_i needs to be performed before T_j. **Assume each task T_i takes 1 unit of time to complete.** Given k processors P_1, P_2, ..., P_k a **schedule** for T assigns each sub-task T_i a processor P_j as well as a time t_i at which P_j shoudl start task T_i. Observe that if a processor starts T_i at times t_i, then the task will complete at time $t_i + w_i$, where w_i is the weight of T_i. A schedule is **feasible** if:

1 no single processor is performing multiple tasks at the same time (i.e., if a processor is assigned T_i at time t_i and T_j at time $t_j > t_i$, then $t_j \geq t_i + w_i$), and

2 for every pair of tasks T_i and T_j with $T_i \prec T_j$, T_j is schedule to start some time after (or at the same time) T_i completes (i.e., $t_j \geq t_i + w_i$).

If we have only one processor, then the answer is fairly simple, just create a topological sorting: Definition: A topological sorting is a total ordering of a partially ordered set.

Algorithm 1 TOPOLOGICAL_SORT(*poset*)

T = empty list
S = all minimal elements in *poset*
while S is not empty **do**
    remove some node u from S and add it to T.
    uv_dependencies set of all dependencies of the form (u, v) for some v.
    for each (u, v) in *uv_dependencies* **do**
        if v is minimal **then** Insert v in T
return T

In this part, you'll implement the following function:
`topological_sort(poset)`: This method takes in a partially ordered set in the form of a dependency list and returns a valid topological sort.
I'll be manually checking that you used some semblance of Kahn's algorithm in your solution, so don't use other approaches like a modified DFS (besides, it'll be more useful for the next part).
Tip: The use of external data structures such as queues or stacks may be helpful (but not required) in completing this task.

Part 4: Multi-Processor Job Scheduling

Now suppose we have multiple processors. Modify your algorithm from the last part to accomodate this change

Dual Dilworth's Theorem

`generate_schedule(poset, num_processors)`: This method takes in a partially ordered set in the form of a dependency list and returns a valid schedule in the form of a list of lists, where the i-th element in the list represents the jobs we should schedule at time $t = i$.

Submission Instructions (10 pts)

After you fill the appropriate functions, submit the following files to Gradescope and make sure you pass all test cases:

- `relations.py`
- `scheduler.py`

Notes

- The autograder may not reflect your final grade on the assignment. We reserve the right to run additional tests during grading.
- Do not import additional packages, as your submission may not pass the test cases or manual review.