Modeling with graphs, solving with Sage

Authors: • Thierry Monteil

In this series of worksheets, we will see how graph theory can help us to represent and solve some concrete problems. For each problem,

- try to model it as a problem on graphs,
- translate the input data of the concrete problem into an input of the graph problem,
- let Sage solve it effectively,
- translate the obtained result back as a solution to the concrete problem,
- check the consistency of the result.

Task assignment

Problem

Twelve robots have to perform twelve tasks. To each robot is associated a subset of the tasks corresponding to its abilities. Both robots and tasks are identified by an ID from 0 to 11.

To fix the ideas, let us assume that the association is given by the following Python dictionary that associates to each robot the list of its abilities:

```
sage: abilities = {0: {2,3,4},
....:              1: {1,7,11},
....:              2: {0,9},
....:              3: {4},
....:              4: {3,5,11},
....:              5: {0,2,4},
....:              6: {1,6,9},
....:              7: {10},
....:              8: {3,5,7,9},
....:              9: {1,2,3},
....:              10: {2,4,6,8},
....:              11: {5,10}}
```

Could you distribute the tasks among the robots so that each robot gets one task corresponding to its abilities and such that every task is accomplished ?

Modeling

A *graph* G is a pair (V,E) such that E is is a subset of pairs of elements of V. The elements of V are called *vertices*, and the elements of E are called *edges*.

For example, $G = (\{0,1,2,3\},\{\{0,1\},\{1,2\},\{1,3\},\{2,3\}\})$ is a graph that can be represented as:

```
sage: Graph([[0,1,2,3],[(0,1),(1,2),(1,3),(2,3)]]).plot()
```

Here, we will represent the abilites with a graph `G` whose vertices are robots and tasks, and we put an edge between a robot with ID `i` and a task with ID `j` if the robot with ID `i` is able to do the task with ID `j`.

As a baby example, if we have three robots an three tasks, where the robot with ID `0` is able to do tasks with ID `1` and `2`, the robot with ID `1` is only able to do the task with ID `2` and the robot with ID `2` is able to do tasks with ID `0` and `2`, we obtain the following graph:

```
sage: Graph([['robot 0','robot 1','robot 2','task 0','task 1','task 2'],[('robot 0',
```

In this framework, a correct assignment consists in chosing a set of edges that will link each robot to its assigned task. Each robot will be adjacent to exactly one edge and each task will be adjacent to exactly one edge. Since the task assigned to each robot should correspond to its abilities, the edges must be chosen among the edges of the graph we constructed above.

A *perfect matching* of a graph G is a subset M of E such that every element of V belongs to exactly one element of M.

$\forall v \in V, \exists! e \in M, v \in e$

Hence, we are looking for a perfect matching of `G`.

Translate the input data to fit in the model

While this problem could perhaps be solved by hand, the input data could be much bigger, so that all the process, including translating the data to fit into the model, should be done automatically.

Let us construct the graph `G` from the input data (that is, the `abilities` dictionary).

First, we have to be careful that the robots and the tasks share the same set of IDs, while they do not correspond to the same vertices in our graph. Hence, we have to separate them: the usual way to transform two sets that have nonempty intersection into two disjoint sets is by *cartesian product*, here $\{0,1\} \times$ IDset , where the robot with ID i is mapped to $(0,i)$ and the task with ID j is mapped to $(1,j)$.

Now, we have to construct a graph from a dictionary mapping integers to sets of integers. Let us look at the different ways to construct a graph and chose the most convenient for our purpose:

```
sage: Graph?
```

We see that the fifth method allows to build a graph from a dictionary mapping vertices to a *list* of its neighbors (not the *set*).

Hence, we have to transform the `abilities` dictionary into a dictionary that maps, for each key `i` of the `abilities` dictionary (corresponding to robots IDs), the vertex `(0,i)` to the list of vertices `(1,j)` where `j` belongs to the set `abilities[i]` (corresponding to the set of tasks that the robot `i` is able to perform).

We can achieve this with list and dictionary comprehensions:

```
sage: d = {(0,i):[(1,j) for j in abilities[i]] for i in abilities} ; d
{(0, 0): [(1, 2), (1, 3), (1, 4)],
 (0, 1): [(1, 1), (1, 11), (1, 7)],
 (0, 2): [(1, 0), (1, 9)],
 (0, 3): [(1, 4)],
 (0, 4): [(1, 3), (1, 11), (1, 5)],
 (0, 5): [(1, 0), (1, 2), (1, 4)],
 (0, 6): [(1, 9), (1, 6), (1, 1)],
 (0, 7): [(1, 10)],
 (0, 8): [(1, 9), (1, 3), (1, 5), (1, 7)],
 (0, 9): [(1, 1), (1, 2), (1, 3)],
 (0, 10): [(1, 8), (1, 2), (1, 4), (1, 6)],
 (0, 11): [(1, 10), (1, 5)]}

sage: G = Graph(d)
sage: G.plot(pos={(a,b):(10*a,-b) for a,b in G.vertices()}, vertex_size=1000)
```

Let Sage solve the graph problem

By tab completion, we see that there exists a `perfect_matchings` method which iterates over all perfect matchings of the graph `G`. So, we can get the first perfect matching found:

```
sage: M = G.perfect_matchings().next() ; M
[((0, 1), (1, 1)),
 ((0, 0), (1, 3)),
 ((0, 3), (1, 4)),
 ((0, 8), (1, 7)),
 ((0, 7), (1, 10)),
 ((0, 9), (1, 2)),
 ((0, 6), (1, 6)),
 ((0, 10), (1, 8)),
 ((0, 11), (1, 5)),
 ((0, 5), (1, 0)),
 ((0, 4), (1, 11)),
 ((0, 2), (1, 9))]
```

We can see that our problem has five solutions:

```
sage: len(list(G.perfect_matchings()))
5
```

However, if we look at the source code of the method, we see that the recursive algorithm tries all possibiles choices of disjoint edges, which may take an exponential time to find the first matching:

```
sage: G.perfect_matchings??
```

A *matching* of a graph G is a set of disjoint edges of G. Equivalently, a matching of a graph G is a subset M of E such that every element of V belongs to at most one element of M.

$$\forall (e,e') \in M^2 (e \cap e' \neq \emptyset \Rightarrow e = e')$$

Hence, a perfect matching is a matching of size $|V|/2$.

There is a polynomial-time algorithm (the first one being due to Edmonds) that, given a graph G, returns a matching of G with maximal cardinality, and such an algorithm is implemented in the `matching` method:

```
sage: M = G.matching() ; M
[((0, 7), (1, 10), None),
 ((0, 10), (1, 8), None),
 ((0, 3), (1, 4), None),
 ((0, 11), (1, 5), None),
 ((0, 4), (1, 11), None),
 ((0, 0), (1, 3), None),
 ((0, 5), (1, 2), None),
 ((0, 8), (1, 9), None),
 ((0, 1), (1, 7), None),
 ((0, 6), (1, 6), None),
 ((0, 9), (1, 1), None),
 ((0, 2), (1, 0), None)]
```

And then check that this maximal matching found by this method is actually a perfect matching:

```
sage: 2*len(M) == len(G)
True
```

This method is much faster (and consumes much less memory) than the previous one for large graphs. For example, you can compare them with a random example of 1000 robots, 1000 tasks and around 10 abilities per robot:

```
sage: R = graphs.RandomBipartite(1000, 1000, 1/100)
sage: %time R.matching()

sage: %time R.perfect_matchings().next()
```

Translate the result into a solution of the original problem

We have to map each robot to its assigned task. Let us do it as a dictionary mapping a robot ID to its assigned task's ID.

We can notice that each element of the list `M` is an edge of the form `((0, robot_ID), (1, task_ID), None)`.

Indeed, the vertices of each edge returned by Sage in a graph are sorted with respect to the lexicographical order, see:

```
sage: G.edges?
```

Hence, since for every i and j, $(0,i) < (1,j)$ we can be a bit lazy and assume that the first vertex of an edge corresponds to a robot and the second one corresponds to its assigned task. So, se can construct the afectation dictionary as follows:

```
sage: assignment = {a[1]:b[1] for (a,b,label) in M} ; assignment
{0: 3, 1: 7, 2: 0, 3: 4, 4: 11, 5: 2, 6: 6, 7: 10, 8: 9, 9: 1, 10: 8, 11: 5}
```

Check that the provided solution is correct

# Post-checking is also ... certificate, not only gives a right to be dirty, tous les coups sont permis, needle dans une botte de foin, random sampling methods,...

Let us check that every robot has an assignment:

```
sage: set(assignment.keys()) == set(abilities.keys())
True
```

Let us check that every task is assigned:

```
sage: set(assignment.values()) == set().union(*[abilities[i] for i in abilities])
True
```

Note that, while in our case it seems useless to check that the obtained

Automatize

Since we wrote our commands in a generic way (for example, the number of robots is not hardcoded), we can merge the whole process into a single function, which also deals with corner cases:

```
sage: def task_assignment(abilities, check=True):
....:     r"""
....:     Return...
....:
....:     INPUT:
....:
....:     - abilities : a dictionary mapping each robot ID to the set
....:       of the ID of the tasks it is able to perform.
....:
....:
....:
```

```
....:
....:     - check (default: True) : if set to ``True``, the result is
....:       checked w.r.t the definition of a correct assignment.
....:
....:
....:     OUTPUT:
....:
....:       A dictionary mapping each robot ID to its assigned task.
....:     """
....:     G = Graph({(0,i):[(1,j) for j in abilities[i]] for i in abilities})
....:     M = G.matching()
....:     if 2*len(M) != len(G):
....:         raise ValueError('There does not exist a correct assignment')
....:     assignment = {a[1]:b[1] for (a,b,label) in M}
....:     if check:
....:         assert set(assignment.keys()) == set(abilities.keys()), 'Check failed'
....:         assert set(assignment.values()) == set().union(*[abilities[i] for i in
....:     return assignment
```

Test

Let us add some tests

```
sage: task_assignment({1:{1},2:{3},3:{1,2,3}})
{1: 1, 2: 3, 3: 2}
```

```
sage: task_assignment({1:{1},2:{1},3:{1,2,3}})
ValueError: There does not exist a correct assignment
```

Note

There is a combinatorially simple (but algorithmically inefficient) necessary and sufficient condition for such an abilities dictionary to admit a correct task assignment, known as the Hall's marriage theorem, see the following references:
[1] van Lint and Wilson: A Course in Combinatorics, Chapter 5, "Systems of distinct representatives".
[2] Aigner and Ziegler: Proofs from THE BOOK, Chapter 29, "Three famous theorems on finite sets".

Group exercise assignment

Problem

A list of 11 exercises is presented to a class of 44 students. Each student can chose some exercises in the list and the teachers have to assign a exsecise to each student so that each exercise is studied by a group of 4 students. Here is the dictionary `student -> set of exercises` representing students choices:

```
sage: student_choices = {0: {3, 7},
....:                    1: {1, 4, 8, 9, 10},
....:                    2: {0, 4, 6, 10},
....:                    3: {2, 3, 5},
....:                    4: {2, 3, 4, 9},
....:                    5: {0, 4},
....:                    6: {6, 9, 10},
....:                    7: {4, 5, 6, 9},
....:                    8: {0, 10},
....:                   9: {3, 5, 8},
....:                   10: {1, 9},
....:                   11: {5, 6, 9},
....:                   12: {0, 6, 7},
....:                   13: {2, 6, 10},
....:                   14: {3, 4, 6, 9},
....:                   15: {0, 3, 4, 5, 9},
....:                   16: {0, 1, 9},
....:                   17: {0, 6, 7},
....:                   18: {1, 3, 9, 10},
....:                   19: {1, 8},
....:                   20: {2, 3, 6, 8},
....:                   21: {2, 3, 7, 10},
....:                   22: {2, 7},
....:                   23: {2, 5, 6},
....:                   24: {0, 1, 2, 5},
....:                   25: {6, 7, 9},
....:                   26: {1, 3, 7},
....:                   27: {6, 7},
....:                   28: {1, 3, 6, 9},
....:                   29: {0, 1, 2, 7},
....:                   30: {0, 1, 4},
....:                   31: {3, 8},
....:                   32: {4, 8},
....:                   33: {0, 2, 7, 8},
....:                   34: {5, 7},
....:                   35: {0, 3, 6, 7, 8},
....:                   36: {0, 4, 8, 10},
....:                   37: {2, 6, 10},
....:                   38: {0, 1, 3, 10},
....:                   39: {0, 4},
....:                   40: {0, 6},
....:                   41: {0, 5, 7, 10},
....:                   42: {1, 6},
....:                   43: {3, 8, 9}}
```

Could you help the teachers to correctly assign the exercises to the students ?

Hint: find a way to reduce this problem to the previous one.