Intro to Programming for Public Policy Week 4
Matching Problems

Eric Potash

April 17, 2018

Matching Problem

Market design

- Most markets studied in ecoomics are decentralized
- Recently there has been interest in centralized markets:
  - placing students in schools
  - workers to firms
  - patients to organ donors

Hospital Residency

- After graduating from medical school, students seek residencies (internships) at teaching hospitals
- For a long time, hospitals were in fierce competition for residents
  - They were pushing their offers back earlier and earlier

N.R.M.P.

- In the 50s, the National Residency Matching Program (N.M.R.P.) was created

N.R.M.P.

- In the 50s, the National Residency Matching Program (N.M.R.P.) was created
- Students and hospitals submit their ordered preference lists

N.R.M.P.

- In the 50s, the National Residency Matching Program (N.M.R.P.) was created
- Students and hospitals submit their ordered preference lists
- A computer program matches students to hospitals, using a variant of the Gale-Shapely algorithm that we'll cover later

Input

- n students (e.g. A, B, and C) and n hospitals (X, Y, and Z)
- Each student ranks each hospital

Student	1st	2nd	3rd
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

- Each hospital ranks each student

Hospital	1st	2nd	3rd
X	B	A	C
Y	A	B	C
Z	A	B	C

Matching

A *matching* is set of pairs of students and hospitals, e.g. $(A, Z), (B, Y), (C, X)$

Student	1st	2nd	3rd
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

Hospital	1st	2nd	3rd
X	B	A	C
Y	A	B	C
Z	A	B	C

Unstable pair

- In a given matching, a student s and hospital h are an unstable pair if:
  - h prefers s to its current match
  - s prefers h to its current match
- E.g. (A, Y) are an unstable pair in the matching above:

Student	1st	2nd	3rd
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

Hospital	1st	2nd	3rd
X	B	A	C
Y	A	B	C
Z	A	B	C

Stable matching

A *stable matching* is one in which there are no unstable pairs

Student	1st	2nd	3rd
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

Hospital	1st	2nd	3rd
X	B	A	C
Y	A	B	C
Z	A	B	C

Gale-Shapely Algorithm

Propose and reject

Gale and Shapely proposed a solution to the matching problem in their paper *College Admissions and the Stability of Marriage* (1962). The algorithm has many iterations. In each iteration, we consider a student s and:

- `for` each hospital h that s has not yet proposed to
  - `if` h is free
    - match s to h
  - `else if` h prefers s to current match s'
    - free s' and match s to h

Iterations

- The algorithm continues `while` a student is free and has a hospital they have not yet proposed to.

Properties of algorithm

- There are at most n^2 (one for each possible student-hospital pair) proposals and so the algorithm will terminate.
- Every student and hospital gets a matched
- The algorithm produces a match with no unstable pairs

Non-uniqueness

- For a given set of preferences, there can be more than one stable matching
- The order in which the students are iterated over can affect *which* stable matching is selected

Optimality

- Favors residents over residents
- (The original algorithm has hospitals proposing and so preferred them. That changed in the 1990s.)

Variations

- Married couples
- One hospital admits multiple residents

2012 Nobel Memorial Prize in Economics

Shapely and Alvin Roth, who applied the algorithm to hospitals, schools, and organ donors, shared the 2012 Nobel Memorial Prize in Economics for this work.

Implementation

Students and hospitals lists

```
students = ['A', 'B', 'C']
hospitals = ['X', 'Y', 'Z']
```

Preferences table

We can store the preferences

Student	1st	2nd	3rd
A	X	Y	Z
B	Y	X	Z
C	X	Y	Z

Hospital	1st	2nd	3rd
X	B	A	C
Y	A	B	C
Z	A	B	C

as a python dictionary.

Preferences dictionary

```
prefs = {
    'X': ['B', 'A', 'C'],
    'Y': ['A', 'B', 'C'],
    'Z': ['A', 'B', 'C'],
    'A': ['X', 'Y', 'Z'],
    'B': ['Y', 'X', 'Z'],
    'C': ['X', 'Y', 'Z']
}
```

Pseudo-code

```
initialize M to be an empty matching
while there exists a free student:
   h = s's top preference to which s hasn't proposed
   if h free:
     add (s, h) to the match
   else
     some pair (s2, h) already exists
     if h prefers s to s2
        s2 becomes free
       (s, h) become matched
     else
       (s2, h) remain engaged
```

Matching dictionary

```
initialize M to be an empty matching
```

- We can also use a dictionary to store the matching:

```
M = {}
```

Matching dictionary

```
initialize M to be an empty matching
```

- We can also use a dictionary to store the matching:

```
M = {}
```

- We can either use hospitals or students as keys (since the mapping is one-to-one). We'll use hospitals, so a possible matching would be:

```
M = {'X':'C', 'Y':'B', 'Z':'A'}
```

Free students

```
M = {}
while there exists a free student:
    ...
```

Free students list

```
free_students = ['A', 'B', 'C']
```

```
M = {}
while len(free_students) > 0:
```

Proposals

```
M = {}
while len(free_students) > 0:
```

```
M = {}
while len(free_students) > 0:
    h = s's top preference to which s hasn't proposed
```

List pop

- Note that `list.pop()` without a parameter removes and returns the *last* element in the list.

List pop

- Note that `list.pop()` without a parameter removes and returns the *last* element in the list.
- Recall

```
>>> prefs['A']
['X', 'Y', 'Z']
```

List pop

- Note that `list.pop()` without a parameter removes and returns the *last* element in the list.
- Recall

```
>>> prefs['A']
['X', 'Y', 'Z']
```

- So pop would remove and return the *least* preferred hospital

List pop

- Note that `list.pop()` without a parameter removes and returns the *last* element in the list.
- Recall

```
>>> prefs['A']
['X', 'Y', 'Z']
```

- So pop would remove and return the *least* preferred hospital
- We simply *reverse* our preference lists so that the most preferred hospital is last:

```
for x in prefs:
    prefs[x].reverse()
```

So far

```python
while len(free_students) > 0:
    s = free_students.pop()
    # if s hasn't proposed to everyone
    if len(prefs[s]) > 0:
        # get the top remaining preference
        h = prefs[s].pop()
```

if h is free

We can translate:

```
if h free:
     add (s, h) to the match
```

to

```
if h not in M:
    M[h] = s
```

So far

```python
while len(free_students) > 0:
    s = free_students.pop()
    # if s has not proposed to everyone
    if len(prefs[s]) > 0:
        # get the top remaining preference
        h = prefs[s].pop()
        if h not in M:
            M[h] = s
```

if h not free

```
else
  some pair (s2, h) already exists
  if h prefers s to s2
     s2 becomes free
    (s, h) become matched
  else
    (s2, h) remain engaged
```

Current match

To find the current match $(s2, h)$:

```
s2 = M[h]
```

Compare rankings

- To compare h's ranking of s and $s2$ we will use an auxillary data structure:

```
rank = {
    'X': {'A':2, 'B':1, 'C':3},
    'Y': {'A':1, 'B':2, 'C':3},
    'Z': {'A':1, 'B':2, 'C':3}
}
```

- Then we can compare rankings:

```
rank[h][s] < rank[h][s2]
```

Conditional

- Finally we can translate:

```
if h prefers s to s2
   s2 becomes free
   (s, h) become matched
else
   (s2, h) remain engaged
```

- Into

```
if rank[h][s] < rank[h][s2]:
    M[h] = s
    free_students.append(s2)
else:
    # s is still free
    free_students.append(s)
```

Putting it together

```python
while len(free_students) > 0:
    s = free_students.pop()
    # if s has not proposed to everyone
    if len(prefs[s]) > 0:
        # get the top remaining preference
        h = prefs[s].pop()
        if h not in M:
            M[h] = s
        else:
            s2 = M[h]
            if rank[h][s] < rank[h][s2]:
                M[h] = s
                free_students.append(s2)
            else:
                # s is still free
                free_students.append(s)

print(M)
```