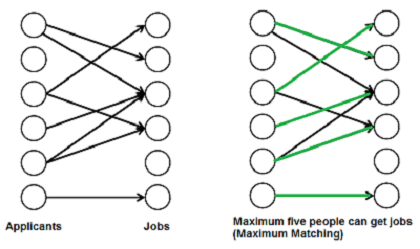


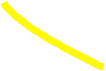
**Program 1: Matching**

In this programming assignment, we will explore the difference between an agent-based solution and a centralized solution.

Motivating Example: Consider matching partners.

Which is more important making as many matches as possible (but may not be stable) or making the happiest matches?





.

**Centralized Matching**

Suppose, for example, that we allowed a centralized broker to make the decisions (given the relative preferences). The marriage r wants to make the partners happy, but also wants as many pairs as possible.

So for example

Applicant’s Preferences

W: b,c,a  
X: b,a,c  
V: b,a

Employer’s Preferences

a: W,X,V  
b: W,X,V  
c: X,W,V

A multiagent solution would be

a2 is paired with X2

b1 is paired with W1

c is paired with None

The broker might prefer:

a3 is paired with V2

b1 is paired with W1

c1 is paired with X3

We will solve this version of the matching problem using a min-cost max flow solution to the network flow.   You need to write  your own min-cost max flow algorithm (not copy from the web or use a library).

Create a flow graph. Connect a source node to each of the men (with cost 0). Connect each of the women to the sink (with cost 0). Create an edge between (man,woman) if woman is on man’s preference list and woman is on man’s preference list. The weight of the edge is the sum of the rank man has for woman plus the rank woman has for man.

A picture containing air, different, device, several

Description automatically generated

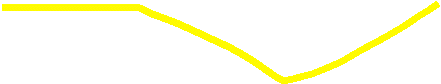
We are using the cost of an edge to be the sum of the ranks for that pairing. For example, the edge from cX is 4 as c ranks X as 1 and X ranks C as 3. Edges from the source or to the sink are of weight 0. All edges have capacity 1.

Paths found in order

0 b W 7 (flow 1) totalCost 2

A picture containing air, different, device, several

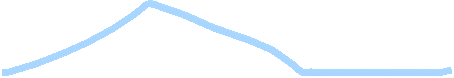
Description automatically generated



0 a X 7 (flow 1) totalCost 4

A picture containing air, different, device, several

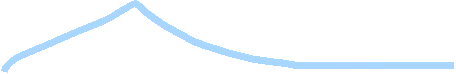
Description automatically generated



0 c X a V 7 (flow 1) totalCost 5

A picture containing air, different, device, several

Description automatically generated



Output:

a. For files Employers1.txt, Applicants1.txt,(and Employers2.txt, Applicants2.txt)

Using stable matching, show the matching provided with Employers proposing.

Using stable matching, show the matching provided with Applicants proposing.

Using min cost max flow, show the matching provided.

b. For data files you create

Using stable matching, show the matching provided with Employers proposing.

Using stable matching, show the matching provided with Applicants proposing.

Using min cost max flow, show the matching provided.

**Hints:**

1. Do read the code that has been provided. It creates adjacency, residual, and cost matrices.
2. For the min cost max flow algorithm, store edges in an adjacency matrix. Use a residual matrix to store the remaining flow.
3. Store costs in another matrix where the cost of an edge is the sum of the preferences.
4. You will want to use Ford Fulkerson algorithm.
5. You will want to use Bellman Ford as a shortest path algorithm as it accommodates negative edge weights. This has already been implemented for you in the starter code.

**boolean bellmanFord(G, source, sink){**

**for each vertex V in G{**

**distance[V] = infinite**

**previous[V] = -1**

**}**

**distance[source]= 0**

**for each vertex V in G**

**for each edge (U,V) in G{**

**tempDistance = distance[U] + edge\_weight(U, V)**

**if tempDistance < distance[V]{**

**distance[V] =tempDistance**

**previous[V] = U**

**}**

**}**

**return previous[sink] >=0**

**}**

**//Finds how much flow is on a path**

**public FordFulkerson(FlowNetwork G, int s, int t) {**

**totalFlow= 0;**

**while (BellmanFord(G, s, t)) {**

**// compute bottleneck capacity starting at terminal node**

**double availFlow = Double.POSITIVE\_INFINITY;**

**for (int v = t; v != s; v=prev){**

**prev = pred[v];**

**availFlow = Math.min(availFlow, residual[prev][v])**

**}**

**// augment flow edges from residual[prev][v]**

**for (int v = t; v != s; v=prev){**

**prev = pred[v];**

**update residual by availFlow on forward AND backward edges.**

**}**

**totalFlow += availFlow;**

**}**

**}**