COSC1285/2123: Algorithms & Analysis

Iterative Improvment

Jeffrey Chan

RMIT University
Email : jeffrey.chan@rmit.edu.au

Lecture 11

Overview

Levitin – The design and analysis of algorithms

This week we will be covering the material from Chapters 10.

Learning outcomes:

- Understand the paradigm of iterative improvement
- Understand and apply examples of iterative improvement:
  - Maximum-flow problem
  - Stable marriage problem (Gale-Shapeley algorithm)

Outline

Overview

Iterative Improvement

Previously, we looked at greedy approaches to solving optimisation problems. It constructs a solution piece by piece, in a greedy fashion.

Iterative Improvement

Previously, we looked at greedy approaches to solving optimisation problems. It constructs a solution piece by piece, in a greedy fashion.

In contrast, iternative improvement starts with a feasible solution (one that satisfies all constraints), then proceed to improve it by repeated application of simple steps.

Overview

Maximum-Flow Problem

Imagine you are given this problem:

Problem

Yarra Valley Waters needs to move water from a dam to a local water reserviour. There is a network of pipes and junctions that they can transport water over. Assume there is no loss within the network. How do we maximise the amount of water transported each day?

Maximum-Flow Problem

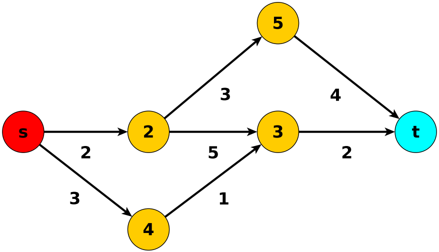
Imagine you are given this problem:

Problem

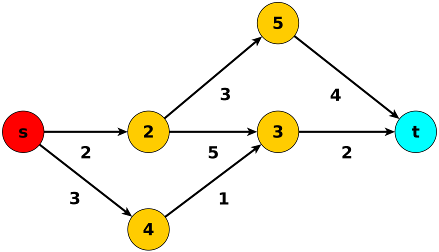
Yarra Valley Waters needs to move water from a dam to a local water reserviour. There is a network of pipes and junctions that they can transport water over. Assume there is no loss within the network. How do we maximise the amount of water transported each day?

This is an instance of a maximum-flow problem.

Flow Network

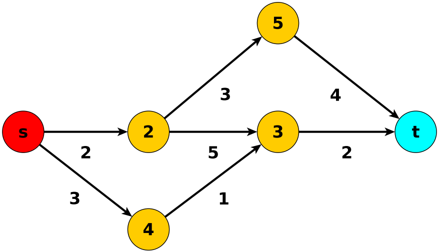


Flow Network



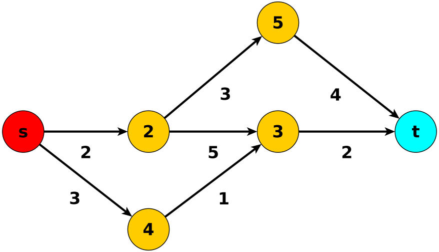
- Source: vertex which has no incoming edges.

Flow Network



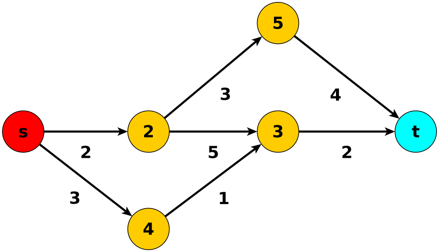
- Source: vertex which has no incoming edges.
- Sink: vertex with no outgoing edges.

Flow Network



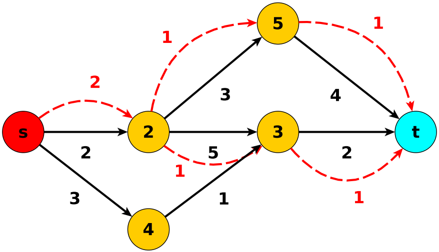
- Source: vertex which has no incoming edges.
- Sink: vertex with no outgoing edges.
- Edge has a weight representing its capacity.

Flow Network



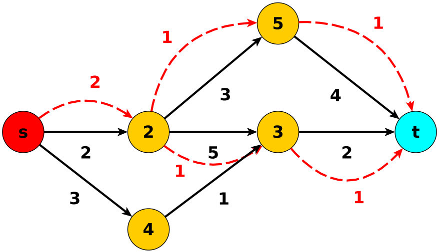
- Source: vertex which has no incoming edges.
- Sink: vertex with no outgoing edges.
- Edge has a weight representing its capacity.
- Graphs satisfying above properties called flow network.

Maximum-Flow Problem



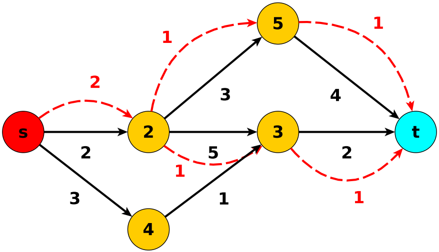
- Source is the origin of all “material” into the flow network.

Maximum-Flow Problem



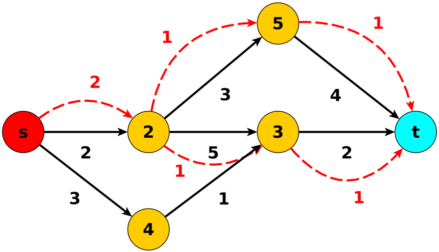
- Source is the origin of all "material" into the flow network.
- Sink is the final destination of all "material" in the network.

Maximum-Flow Problem



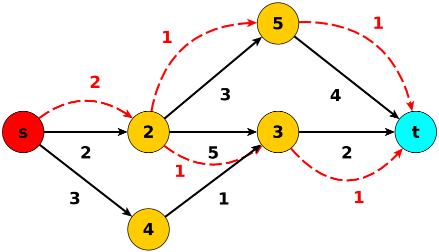
- Source is the origin of all “material” into the flow network.
- Sink is the final destination of all “material” in the network.
- Maximum amount of flow on an edge cannot exceed its capacity; capacity constraint.

Maximum-Flow Problem



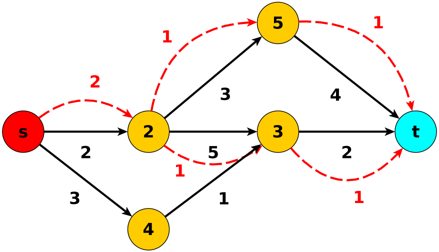
- Source is the origin of all "material" into the flow network.
- Sink is the final destination of all "material" in the network.
- Maximum amount of flow on an edge cannot exceed its capacity; capacity constraint.
- All other vertices are transit points – flow in = flow out; called flow-conservation.

Maximum-Flow Problem



- Source is the origin of all “material” into the flow network.
- Sink is the final destination of all “material” in the network.
- Maximum amount of flow on an edge cannot exceed its capacity; capacity constraint.
- All other vertices are transit points – flow in = flow out; called flow-conservation.
- Total material leaving source = total material flowing into sink; called value of the flow.

Maximum-Flow Problem



Problem

Given a flow network, the maximum-flow problem is find a flow of maximum value, subject to (edge) capacity constraints and flow-conservation.

Ford Fulkerson Method – Sketch

Idea:

- Given an initial flow network, find an initial feasible flow.

Ford Fulkerson Method – Sketch

Idea:

- Given an initial flow network, find an initial feasible flow.
- Find a path from source to sink that can increase the total flow.

Ford Fulkerson Method – Sketch

Idea:

- Given an initial flow network, find an initial feasible flow.
- Find a path from source to sink that can increase the total flow.
- Increase the flow along that path.

Ford Fulkerson Method – Sketch

Idea:

- Given an initial flow network, find an initial feasible flow.
- Find a path from source to sink that can increase the total flow.
- Increase the flow along that path.
- Repeat until no more such paths.

Preliminaries: Residual Network

Purpose: Given a flow, the residual network shows which edges in the flow network can admit more flow.

Purpose: Given a flow, the residual network shows which edges in the flow network can admit more flow.

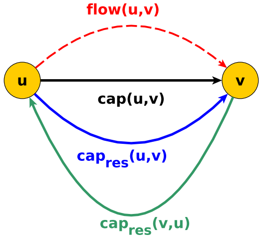
For each edge (u,v) in flow network, we have two edges in the residual network with following residual capacity:

Preliminaries: Residual Network

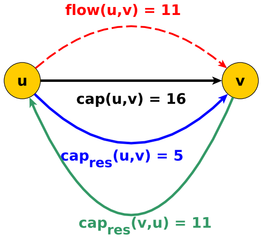
Purpose: Given a flow, the residual network shows which edges in the flow network can admit more flow.

For each edge (u,v) in flow network, we have two edges in the residual network with following residual capacity:

$cap_{res}(u,v) = cap(u,v) - flow(u,v)$, $cap_{res}(u,v) > 0$ (forward edge)
$cap_{res}(v,u) = flow(u,v)$, $cap_{res}(v,u) > 0$ (backward edge)



Example: cap(u,v) = 16, flow(u,v) = 11, then can still increase the flow by cap_{res}(u,v) = 5 in the (u,v) direction, but can also send up to cap_{res}(v,u) = 11 units in the other (v,u) direction to cancel out flow(u,v).



Preliminaries: Augmenting Path

Given a residual network, an augmenting path is a path from s to t in the residual graph.

Preliminaries: Augmenting Path

Given a residual network, an augmenting path is a path from s to t in the residual graph.

From the definition of a residual network, we know each edge in the augmenting path can admit additional positive flow without violating the capacity of the edge.

Preliminaries: Augmenting Path

Given a residual network, an augmenting path is a path from s to t in the residual graph.

From the definition of a residual network, we know each edge in the augmenting path can admit additional positive flow without violating the capacity of the edge.

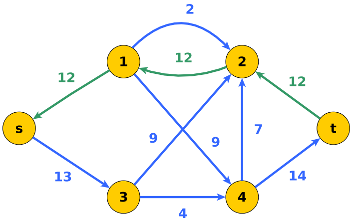
This path basically tells us which individual flows to increase (when traversing forward edge), and which to decrease (when traversing backward edge) to increase the total flow/value, while satisfying capacity and flow conservation constraints.

Preliminaries: Augmenting Path

Given a residual network, an augmenting path is a path from s to t in the residual graph.

From the definition of a residual network, we know each edge in the augmenting path can admit additional positive flow without violating the capacity of the edge.

This path basically tells us which individual flows to increase (when traversing forward edge), and which to decrease (when traversing backward edge) to increase the total flow/value, while satisfying capacity and flow conservation constraints.



Ford-Fulkerson Method

1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.

Ford-Fulkerson Method

1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.
2. Construct residual network from flow network + current flow.

Ford-Fulkerson Method

1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.
2. Construct residual network from flow network + current flow.
3. If there is a (augmenting) path* *p* from *s* to *t* in the residual network:

Ford-Fulkerson Method

1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.
2. Construct residual network from flow network + current flow.
3. If there is a (augmenting) path* *p* from *s* to *t* in the residual network:
    a. For all the edges in path *p* in residual network, find the one with minimum residual capacity (cap_{minres})

Ford-Fulkerson Method

1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.
2. Construct residual network from flow network + current flow.
3. If there is a (augmenting) path* p from s to t in the residual network:
   a. For all the edges in path p in residual network, find the one with minimum residual capacity (cap_{minres})
   b. For each edge (u,v) in path p, update the flows on flow network:
      if ((u,v) is forward) then flow(u,v) += cap_{minres};
      if ((u,v) is backward) then flow(u,v) -= cap_{minres}

Ford-Fulkerson Method

1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.
2. Construct residual network from flow network + current flow.
3. If there is a (augmenting) path* p from s to t in the residual network:
   a. For all the edges in path p in residual network, find the one with minimum residual capacity (cap_{minres})
   b. For each edge (u,v) in path p, update the flows on flow network:
      if ((u,v) is forward) then flow(u,v) += cap_{minres};
      if ((u,v) is backward) then flow(u,v) -= cap_{minres}
   c. Update residual network

Ford-Fulkerson Method

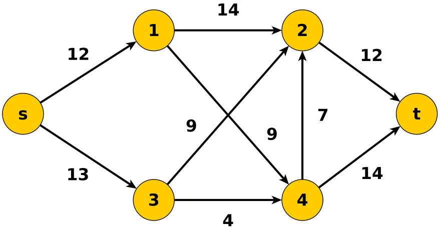
1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.
2. Construct residual network from flow network + current flow.
3. If there is a (augmenting) path* *p* from *s* to *t* in the residual network:
   a. For all the edges in path *p* in residual network, find the one with minimum residual capacity (cap_{minres})
   b. For each edge (u,v) in path *p*, update the flows on flow network:
      if ((u,v) is forward) then flow(u,v) += cap_{minres};
      if ((u,v) is backward) then flow(u,v) -= cap_{minres}
   c. Update residual network
4. repeat step 3 until no more paths from *s* to *t* in residual network

Ford-Fulkerson Method

1. For each edge (u,v) in flow network, set flow(u,v) = 0 and flow(v,u) = 0.
2. Construct residual network from flow network + current flow.
3. If there is a (augmenting) path* *p* from *s* to *t* in the residual network:
   a. For all the edges in path *p* in residual network, find the one with minimum residual capacity (cap_{minres})
   b. For each edge (u,v) in path *p*, update the flows on flow network:
      if ((u,v) is forward) then flow(u,v) += cap_{minres};
      if ((u,v) is backward) then flow(u,v) -= cap_{minres}
   c. Update residual network
4. repeat step 3 until no more paths from *s* to *t* in residual network

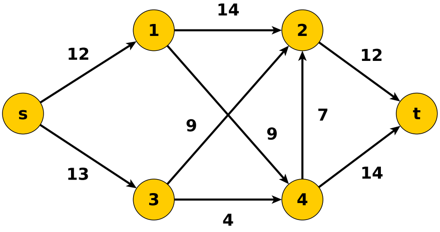
* There are generally many possible augmenting path. We select the shortest one (in terms of number of edges) for step 3.

Ford-Fulkerson Method – Example

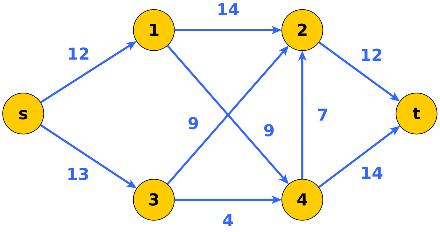


(a) Flow network

Ford-Fulkerson Method – Example

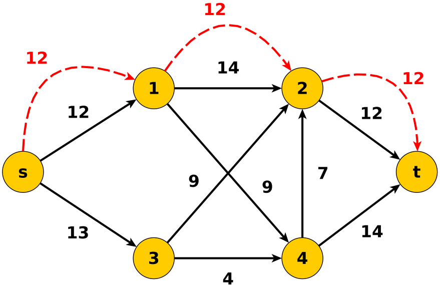


(c) Flow network



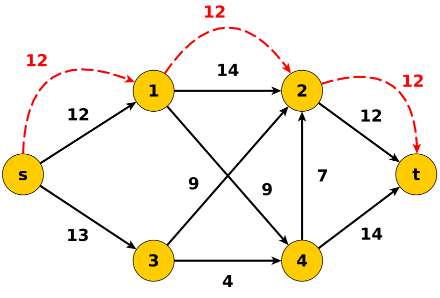
(d) Residual network

Ford-Fulkerson Method – Example

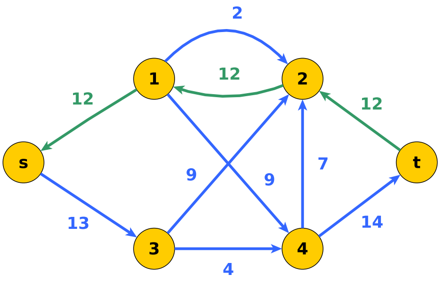


(e) Flow network

Ford-Fulkerson Method – Example

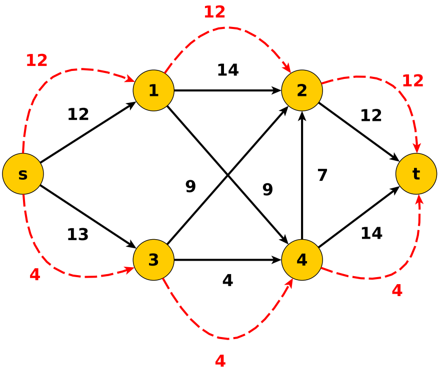


(g) Flow network



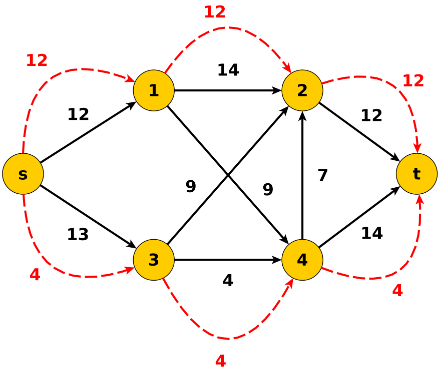
(h) Residual network

Ford-Fulkerson Method – Example

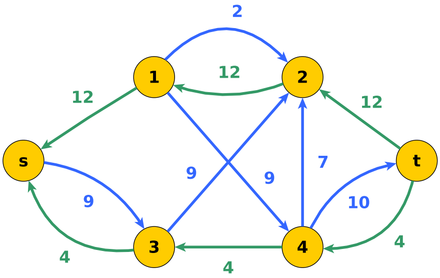


(i) Flow network

Ford-Fulkerson Method – Example

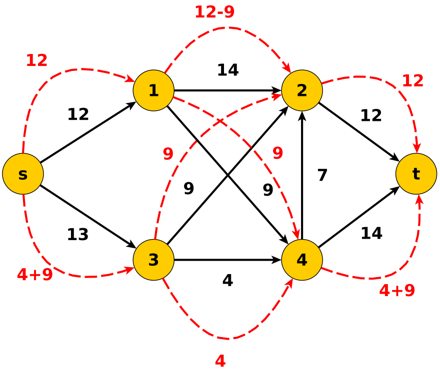


(k) Flow network



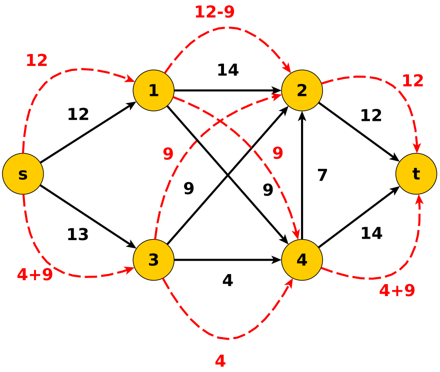
(l) Residual network

Ford-Fulkerson Method – Example

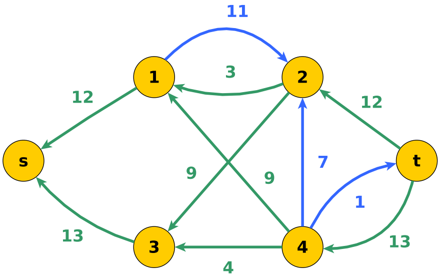


(m) Flow network

Ford-Fulkerson Method – Example

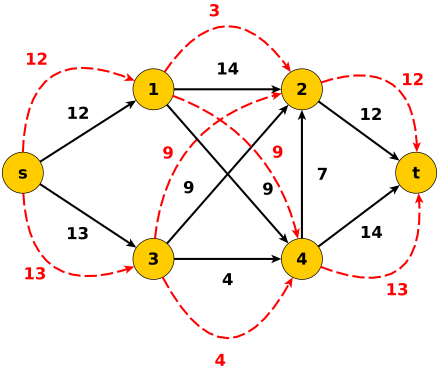


(o) Flow network



(p) Residual network

Ford-Fulkerson Method – Example



Value of flow?

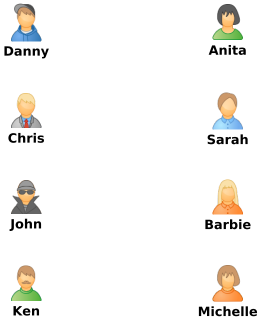
Maximum-Flow Problem

Applications of maximum-flow problem:

`https://www.youtube.com/watch?v=D36MJCXT4Qk`

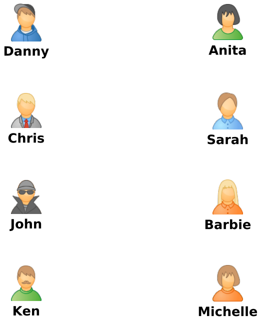
Overview

Stable Marriage Problem

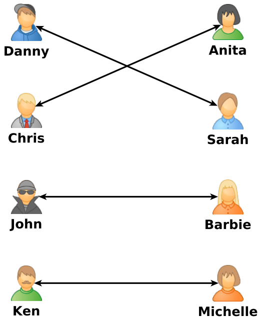


(q) Males and Females.

Stable Marriage Problem



(s) Males and Females.



(t) Matched couples!

Stable Marriage Problem

Stable Marriage Problem

Given a set of n men and n women, who has a list of preferences to the other sex (in terms of a ranking), the problem is how to find a matching between them such that the matching is *stable*?

A matching is stable if:

- No matched pair of man and woman can find other partners and both do better, i.e., both man and woman prefer other partners over their existing match.

Stable Marriage Problem

Stable Marriage Problem

Given a set of n men and n women, who has a list of preferences to the other sex (in terms of a ranking), the problem is how to find a matching between them such that the matching is *stable*?

A matching is stable if:

- No matched pair of man and woman can find other partners and both do better, i.e., both man and woman prefer other partners over their existing match.

Is a stable marriage (matching) always possible?

- Yes, if equal number of men and women.

Gale-Shapley Algorithm

Idea: Female proposing variant:

1. Over a number of rounds, each unmatched female proposes to their remaining highest male preferences.

Gale-Shapley Algorithm

Idea: Female proposing variant:

1. Over a number of rounds, each unmatched female proposes to their remaining highest male preferences.
2. Each round, males reply "yes" to proposal from their highest female proposer and "no" to all other proposers.

Gale-Shapley Algorithm

Idea: Female proposing variant:

1. Over a number of rounds, each unmatched female proposes to their remaining highest male preferences.
2. Each round, males reply “yes” to proposal from their highest female proposer and “no” to all other proposers.
3. This continues until all females (and males) are matched.

Gale-Shapley Algorithm

Female Proposing variant:

1. Round 1: Each female proposes to their first male preferences. Each male receives 0 or more proposals. They reply “maybe” to the female they most prefer and “no” to all other proposals. For each “maybe” reply, the corresponding female-male are tentatively matched.

Gale-Shapley Algorithm

Female Proposing variant:

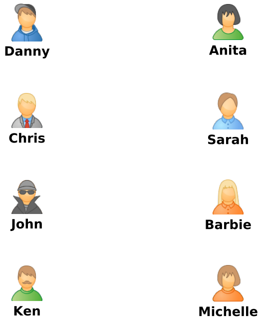
1. Round 1: Each female proposes to their first male preferences. Each male receives 0 or more proposals. They reply “maybe” to the female they most prefer and “no” to all other proposals. For each “maybe” reply, the corresponding female-male are tentatively matched.
2. Round 2: Each unmatched female proposes to their next preferred male (2nd ranked), even if the male is tentatively matched already. Each male evaluates their proposals, and again reply “maybe” to the female they most prefer (this can be their existing matched partner) and “no” to all other proposals. For each “maybe” reply, the corresponding female-male are tentatively matched.

Gale-Shapley Algorithm

Female Proposing variant:

1. Round 1: Each female proposes to their first male preferences. Each male receives 0 or more proposals. They reply “maybe” to the female they most prefer and “no” to all other proposals. For each “maybe” reply, the corresponding female-male are tentatively matched.
2. Round 2: Each unmatched female proposes to their next preferred male (2nd ranked), even if the male is tentatively matched already. Each male evaluates their proposals, and again reply “maybe” to the female they most prefer (this can be their existing matched partner) and “no” to all other proposals. For each “maybe” reply, the corresponding female-male are tentatively matched.
3. Round 3 onwards: We continue with this process until all females and males are matched.

Gale-Shapley Algorithm Example



Gale-Shapley Algorithm Example

Anita	Sarah	Barbie	Michelle
Chris	Chris	Ken	Chris
Ken	Ken	Danny	Ken
Danny	Danny	John	Danny
John	John	Chris	John

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 1 (proposing):

Anita	Sarah	Barbie	Michelle
Chris	Chris	Ken	Chris

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 1 (end):

Anita	Sarah	Barbie	Michelle
Chris	~~Chris~~	Ken	~~Chris~~

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 2 (proposing):

Anita	**Sarah**	**Barbie**	**Michelle**
Chris	~~Chris~~	Ken	~~Chris~~
	Ken		Ken

Table: Female preferences

Danny	**Chris**	**John**	**Ken**
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 2 (end):

Anita	Sarah	Barbie	Michelle
Chris	~~Chris~~	~~Ken~~	~~Chris~~
	~~Ken~~		Ken

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 3 (proposing):

Anita	Sarah	Barbie	Michelle
Chris	~~Chris~~	~~Ken~~	~~Chris~~
	~~Ken~~	Danny	Ken
	Danny		

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 3 (end):

Anita	Sarah	Barbie	Michelle
Chris	~~Chris~~	~~Ken~~	~~Chris~~
	~~Ken~~	~~Danny~~	Ken
	Danny		

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 4 (proposing):

Anita	Sarah	Barbie	Michelle
Chris	~~Chris~~	~~Ken~~	~~Chris~~
	~~Ken~~	~~Danny~~	Ken
	Danny	John	

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example

Round 4 (end):

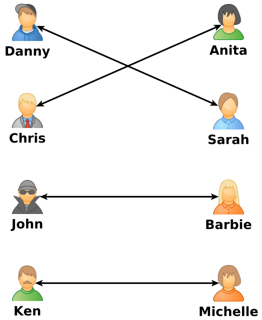
Anita	Sarah	Barbie	Michelle
Chris	~~Chris~~	~~Ken~~	~~Chris~~
	~~Ken~~	~~Danny~~	Ken
	Danny	John	

Table: Female preferences

Danny	Chris	John	Ken
Michelle	Anita	Barbie	Michelle
Sarah	Sarah	Anita	Barbie
Barbie	Michelle	Michelle	Sarah
Anita	Barbie	Sarah	Anita

Table: Male preferences

Gale-Shapley Algorithm Example



Gale-Shapley Algorithm

Properties of algorithm:

- It will always find a stable marriage configuration.
- All males and females will be matched.

Gale-Shapley Algorithm

Properties of algorithm:

- It will always find a stable marriage configuration.
- All males and females will be matched.
- For the female proposing variant, the females are matched with the best male partners under any stable marriage, but the males can be matched with their worst female partners under any stable marriage.

Gale-Shapley Algorithm

Properties of algorithm:

- It will always find a stable marriage configuration.
- All males and females will be matched.
- For the female proposing variant, the females are matched with the best male partners under any stable marriage, but the males can be matched with their worst female partners under any stable marriage.
- Stable matchings may not be unique, e.g., the male proposing variant could come up with a different stable matching.

Gale-Shapley Algorithm

Properties of algorithm:

- It will always find a stable marriage configuration.
- All males and females will be matched.
- For the female proposing variant, the females are matched with the best male partners under any stable marriage, but the males can be matched with their worst female partners under any stable marriage.
- Stable matchings may not be unique, e.g., the male proposing variant could come up with a different stable matching.

Time complexity?

Stable Marriage Video

https://www.youtube.com/watch?v=fudb8DuzQlM

Overview

Summary

- Maximum flow problem (Ford-Fulkerson method)
- Stable marriage problem (Gale-Shapley algorithm)