GB13624 - Maths for Computer Science

Lecture 5 – Graphs Part II

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2020-11-04

Last updated November 1, 2022

Graphs – Lectures 4 and 5

Lecture I: Chapter 9

- Graphs and Relations
- Directed Graphs and Walks
- Scheduling and Partial Orders

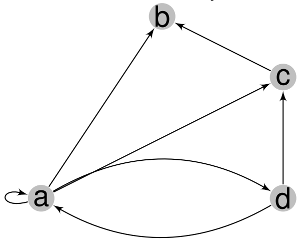
Lecture II: Chapter 11

- Using Isomorphism
- Coloring and Connectivity
- Spanning Trees
- Matching

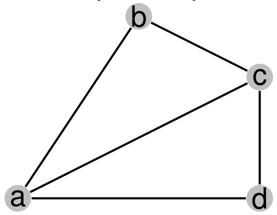
Part 1: Graph Isomorphism

Directed Graphs and Simple Graphs

Directed Graph:



Simple Graph:



- No double edges allowed;
- No self-loop allowed;

Simple Graphs

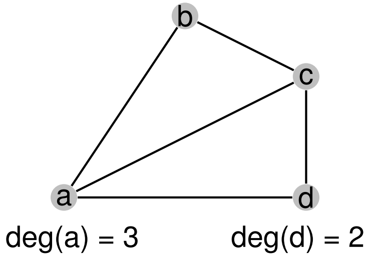
Some definitions

A Simple Graph G consists of:

- A *non-empty* set V of vertices;
- A set E of edges so that:
    - Each edge has two endpoints in V: (not an **start** and an **end**)
    - The order of the vertices in an edge does not matter: $e_1 = \{v_1, v_2\} = \{v_2, v_1\}$
    - Two vertices with an edge between them are adjacent
    - An edge that connects two vertices is incident to them. Ex: e_1 is incident to v_1 and v_2

Vertice Degrees

The degree of a vertex is the number of incident edges.



Quiz: Can you build a graph with following vertice degrees?

- 3, 2, 2, 1 (four vertices)
- 3, 2, 2, 2 (four vertices)

Verdice Degrees

The Handshaking Lemma

Lemma: The sum of vertice degrees in a graph is 2x the number of edges.

$$2|E| = \sum_{v \in V} \deg(v) \tag{1}$$

Proof.

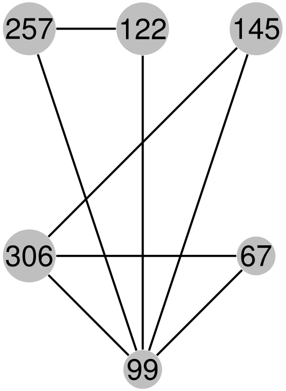
- Every edge in a graph connects two vertices;
- If we begin with a graph with 0 edges, for every edge (v_i, v_j) that we add to the graph, we add 2 vertice degrees (one for v_i, one for v_j).
- So the total of vertices is 2 times the total of edges.

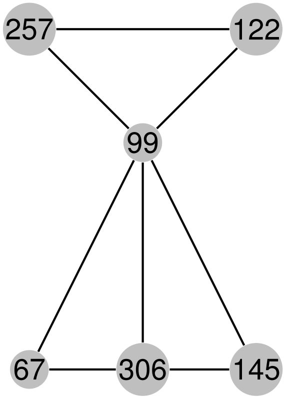
□

Because of the lemma, it is impossible to make a graph with vertice degrees 3, 2, 2, 2.

Review: Isomorphism in graphs

Remember: an isomorphism is an edge preserving bijection

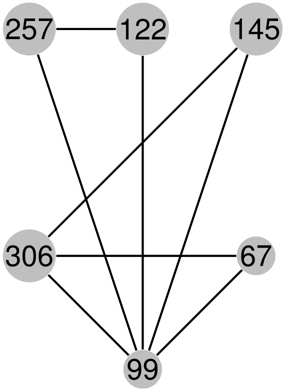


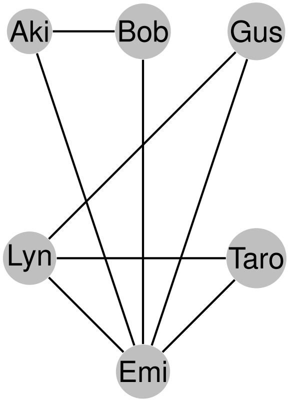


The left and the right are the same graph, but with different positions for the vertices.

Review: Isomorphism in graphs

Remember: Isomorphism is an edge preserving vertex bijection





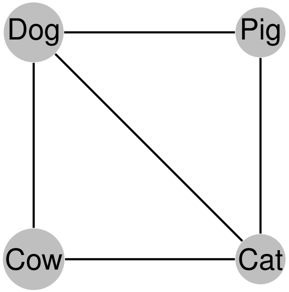
The left and the right are the same graph, but with different **labels** for the vertices.

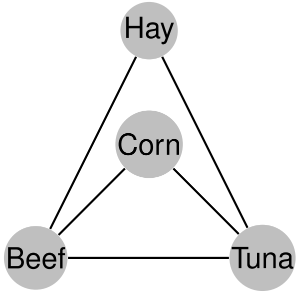
Isomorphism

- Graph Isomorphism is determined solely by the edges between vertices;
- Two graphs with the same edge connections are isomorphic;
- Formally, wwo graphs are isomorphic if there is an Edge Preserving Matching Relation between their vertices;

Isomorphism

Are these graphs Isomorphic?





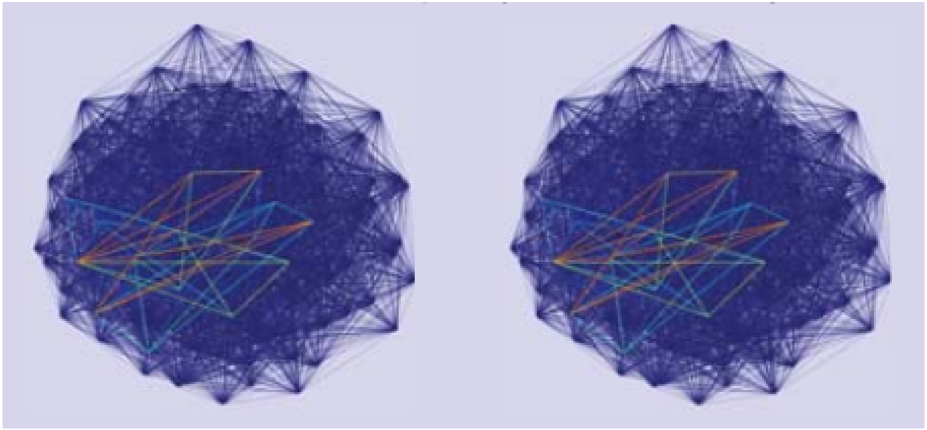
Edge Preserving Bijection:

f(dog) = Beef; f(cow) = Hay

f(cat) = Tuna; f(pig) = Corn

Isomorphism

Are THESE graphics Isomorphic? (2)



Graph Isomorphism

Edge Preserving Bijection

G_1 isomorphic to G_2 means that $\exists$ Edge Preserving Vertex Matching:

$$\exists f : V_1 \to V_2, (u, v) \in E_1 \iff (f(u), f(v)) \in E_2$$

It is easy to quickly identify **non-isomorphic** graphs:

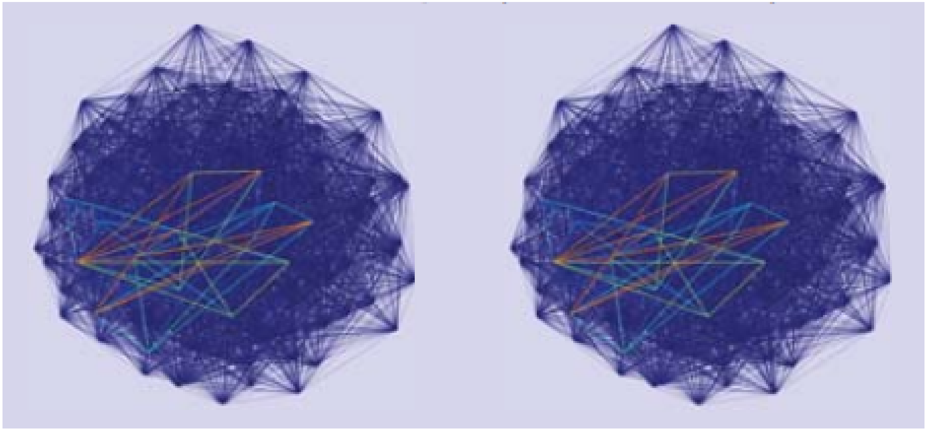
- Not the same number of vertices;
- Not the same number of edges;
- Not the same degree distribution;
- Differences in Paths, Distances, etc...

How to find Graph Isomorphism?

- Finding the bijection is very hard:
  - Total Number of possible bijections: permutation on $|V|$
- If the graph is "small", can check the permutations by hand;
- If the graph is "large", create random matchings $f : V_1 \rightarrow V_2$, and check:
  - Quickly prune matchings that are **not** isomorphic:
  - Vertices in the bijection must have the same degree. (ex: a vertice with edge 4 must match to another vertice with edge 4)
  - *Adjacent vertices* must match degree as well. (ex: A vertice with degree 3, and neighbors with degree 4, 2, 1)

How to find Graph Isomorphism?

Finding an isomorphism for two graphs is a very expensive, and important, problem. In theory, there is no algorithm that is better than just checking every possible bijection.



Part 2: Graph Isomorphism

Graph Coloring: Airplanes and boarding gates

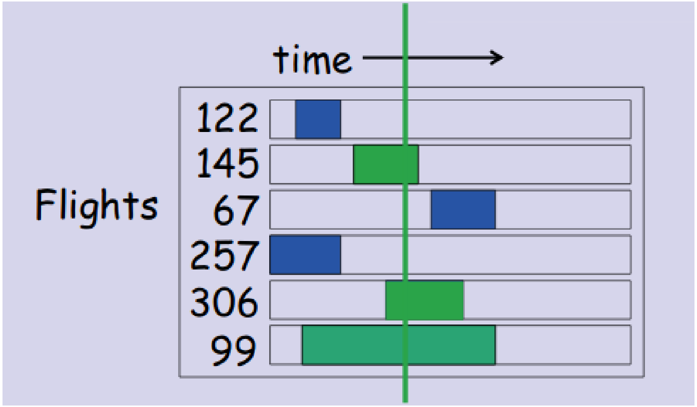
Graph coloring is a problem with several applications, such as scheduling problems. Let's look at Gate scheduling:

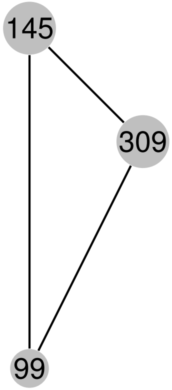
Example:

- Every flight requires a gate for embark/disembark;
- Sometimes flight times overlap, so multiple gates are necessary;
- How many gates do we need to satisfy a flight schedule?

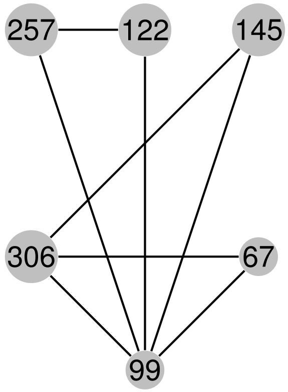
Boarding Gate Scheduling Graph

Let's define a Gate Scheduling Graph, where each flight is a vertice, and an edge indicates that two flights are on the ground at the same time.



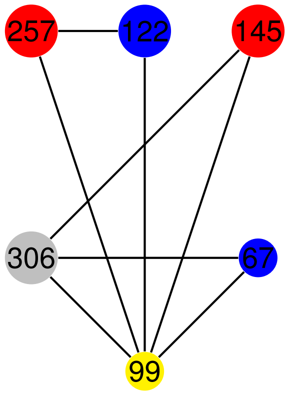


Gate Scheduling Graph and Coloring



- If each flight is a vertice, and each edge is a conflict, we can use graph coloring to solve the problem.
- Each color is a new gate.
- If two vertices have an edge between them, the flights are in conflict, and their colors must be different.
- The minimum number of colors to color all vertices is the same as the minimum number of boarding gates.

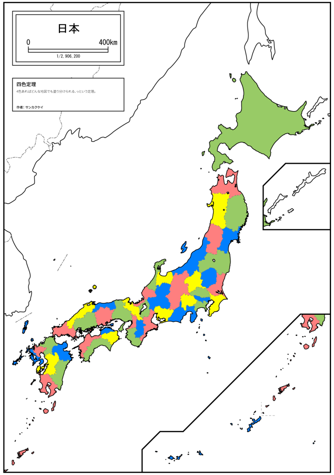
Gate Scheduling Graph and Coloring



- We select colors for each vertex so that no adjacent vertex has the same color.
- Each color = One new Gate
- Final gate assignment:
  - Blue Gate: Flight 122 and 67
  - Red Gate: Flight 145 and 257
  - Yellow Gate: Flight 99
  - Gray Gate: Flight 306
- Can you find a better coloring using only 3 gates?

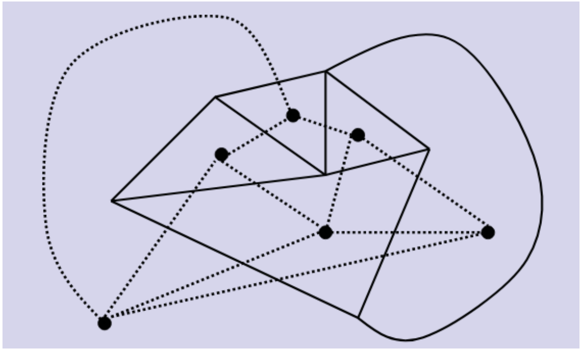
More Graph Coloring Problems

- Allocate classrooms for courses. Some courses can be at the same time.
- Allocate cages for animals. Some animals can't live in the same cage.
- Different Frequencies for radio stations. Some frequencies interfere with each other
- Color a map so that it look pretty!



Vertex Coloring and Face Coloring

Graphs (Vertices) to Maps (Faces) are equivalent when coloring!



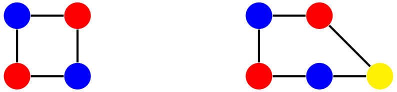
Theorem: Maps can always be colored with 4 colors

- 1970: “Proof” with computers (automatically checks 1000’s of maps)
- 1990: Better mathematical proof. (still needs programmed testing)

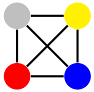
Chromatic Number

The Chromatic Number $\chi(G)$ is the minimum number of colors needed for a graph G.
Examples: There are several rules for certain kinds of graphs:

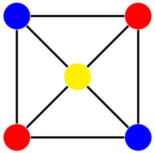
- Cycle Graphs: $\chi(C_{\text{even}}) = 2$, $\chi(C_{\text{odd}}) = 3$



- Complete Graph with n vertices: $\chi(K_n) = n$



- Wheel Graph: $\chi(W_{\text{odd}}) = 4, \chi(W_{\text{even}} = 3)$



Bounding Chromatic Numbers

What is the maximum Chromatic Number?

- If all vertex degrees are $\leq k \implies \chi(G) \leq k+1$

  (Proof by Greedy coloring algorithm).

- Is a graph 2-colorable?

  (**easy** to check: do a Breath First Search and mark as you go)

- Is a graph 3-colorable?

  (**very hard** to check: NP complete!)

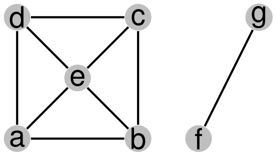
- Is $\chi(G) = k$?

  (in theory not harder than 3 color, harder in practice).

Connectivity

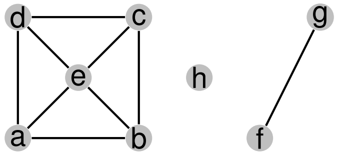
Definition

- Two vertices are connected **iff** there is a path between the two.
- Every vertice is connected to itself. (even if it does not have a self-edge)
- A whole graph is connected if every vertice is connected to each other.



Connected Components

Vertex Connectivity



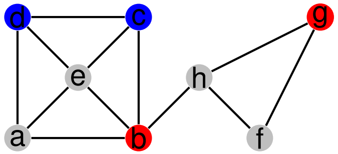
- Every Graph is composed of connected subgraphs called connected components
- connected component of v ::= {w | w connected to v}.
- connected component of $v = E^*(v)$ (walk relation of v)

- A graph is connected **iff** it has exactly 1 connected component.

Connected Components

Edge connectivity

- vertices v, w are k-edge connected if they remain connected even if fewer than k are deleted.



- In this graph, the blue vertices are 3-edge connected, and the red vertices are 1-edge connected;
- A **Graph** is k-edge connected if all pairs of vertices are at least k-edge connected.

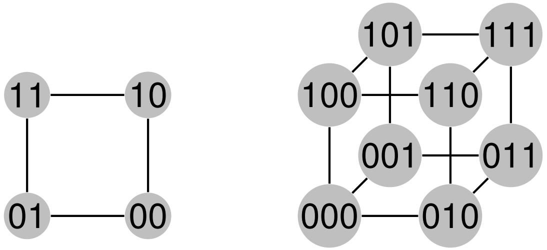
Connected Components

Edge Connectivity

- **Edge Connectivity** represents the degree of **fault tolerance** in a graph.
- **Example:** In a communication network, how many channels can fail before communication is disrupted?

- Related Concept: k-vertice connectivity
  - k-vertice connected graph $\implies$ k-edge connected;
  - BUT! k-edge connected $\not\Longrightarrow$ k-vertice connected.
- The complete graph K_n is n-1 connected.

Connectivity and Hypercubes



- Consider the n-dimensional hypercube H_n
- $V(H_n) ::= \{0,1\}^n$
- $E(H_n) ::= \{(u,v)$ **iff** u and v differ in 1 bit $\}$
- H_n is n vertex connected. (H_n has n^2 vertices)

Part 3: Trees

Trees and Connectivity

- Trees are connected Graphs with no cycles.
- Every tree 1-edge connectivity, 1-vertex connectivity.
- Chromatic Number = 2 (trees can always be bi-colored)

- Trees come up all the time:

- Family Trees;
- Search Trees;
- Game Trees;
- Parse Trees;
- Spanning Trees;
- Rooted Trees;
- Ordered Trees;
- Binary Trees;
- etc...

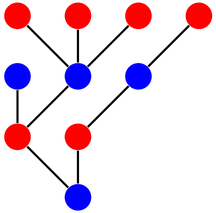
Trees and Connectivity

- Cut Edge: An edge is a cut edge if removing it makes two vertices disconnected.
- **Lemma:** An edge is not a cut edge if it is on a cycle.
- A tree is a connected graph where every edge is a cut edge
- This implies that a tree is a connected graph which is **Edge Minimal**
    - A tree has the minimum number of edges necessary to connect a set of vertices.

Tree Coloring

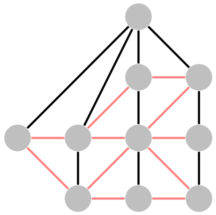
- A tree is a graph with a unique path between every pair of vertices.
- As a consequence, $\chi(\text{tree}) = 2$
- **Constructive Demonstration**

- Pick any node in the tree to be the **root**, color it “blue”.
- Color nodes “odd” length from the root as “red”
- Color nodes “even” length from the root as “blue”
- This is the algorithm for 2-coloring on general graphs



Spanning Trees

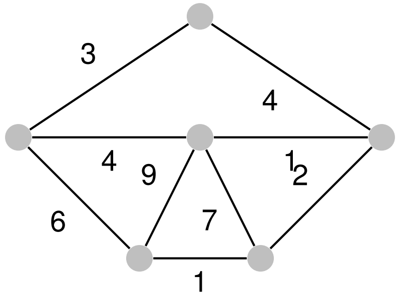
- A Spanning Subgraph of G is a subgraph of *G* that has all vertices of *G* (and some of the edges).
- A Spanning Tree of G is a spanning graph of *G* that is also a tree.



- One graph can have multiple spanning trees.
- Every connected graph has a spanning tree.

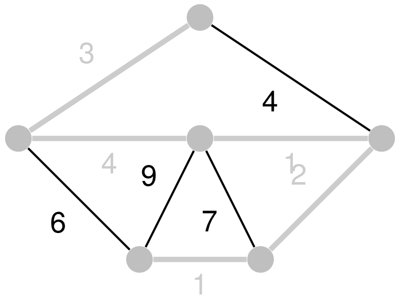
Weighted Spanning Trees

The Spanning Tree problem becomes more interesting when we consider weighted edges.



What is the minimal cost structure that allows me to connect everything?

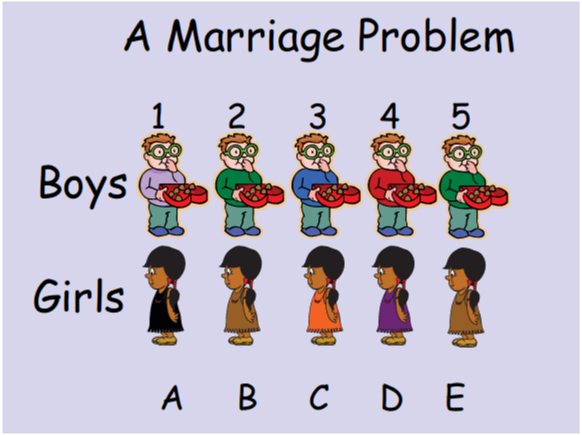
Minimum Spanning Tree Algorithm



1. Start with one arbitrary vertex and add it to the MST.
2. From all edges connected with the MST, select one with minimum weight;
3. Add the edge, and vertex, to the MST;
4. Return to (2)

Part 4: Stable Matching

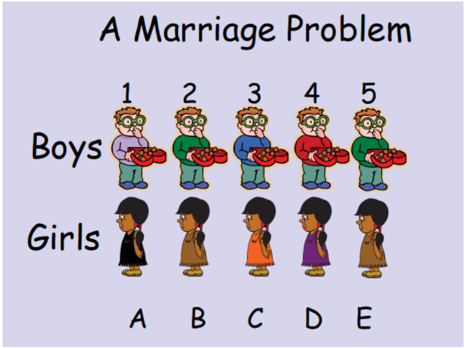
The Stable Marriage Problem

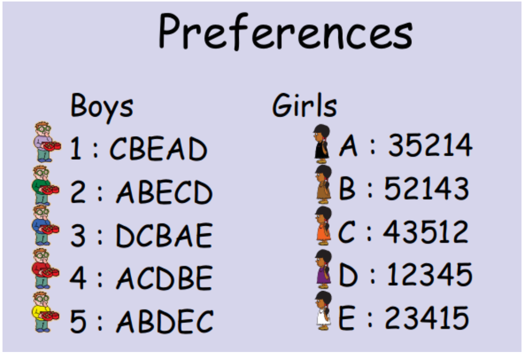


Which boy should marry with which girl?

The Stable Marriage Problem

Each boy and girl has a preference list





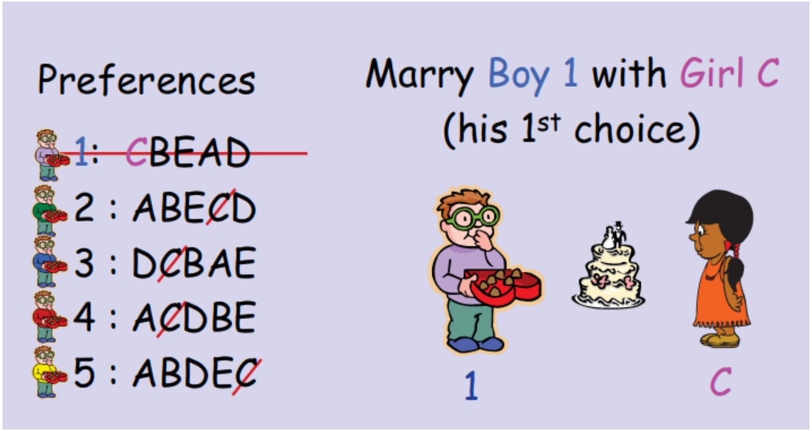
Preferences

Boys	Girls
1 : CBEAD	A : 35214
2 : ABECD	B : 52143
3 : DCBAE	C : 43512
4 : ACDBE	D : 12345
5 : ABDEC	E : 23415

Which algorithm do you use to match them?

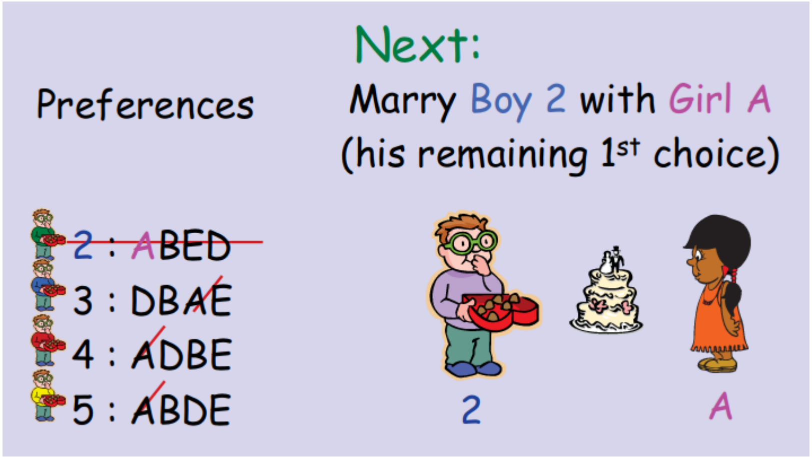
The "Boy-greedy" algorithm

Boy-greedy algorithm: Each boy, in order, marries to favorite girl:



The "Boy-greedy" Algorithm

Boy-greedy algorithm: Each boy, in order, marries to favorite girl:



Next:

Marry Boy 2 with Girl A (his remaining 1st choice)

Preferences

~~2 : ABED~~

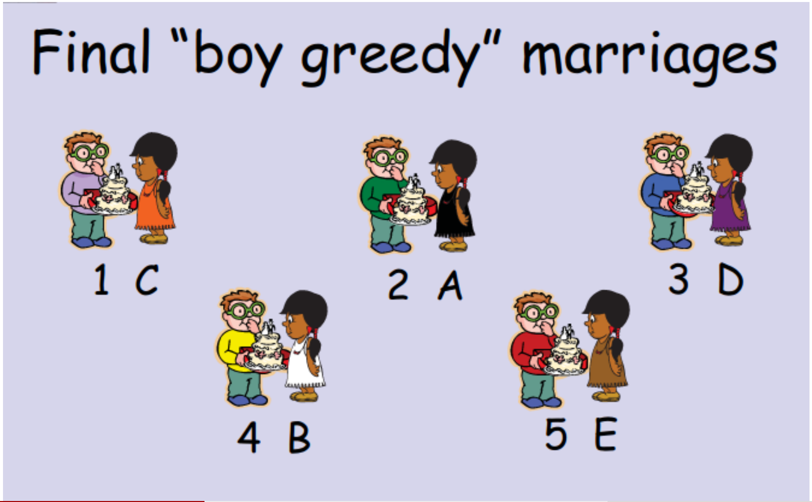
3 : DB~~A~~E

4 : ~~A~~DBE

5 : ~~A~~BDE

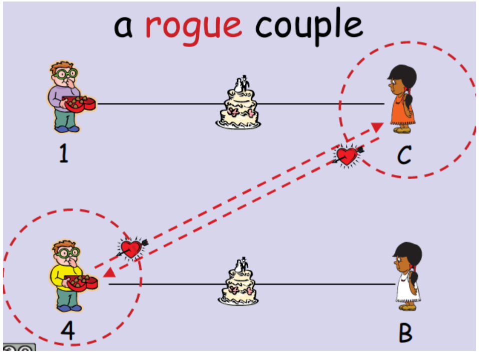
The "Boy-greedy" Algorithm

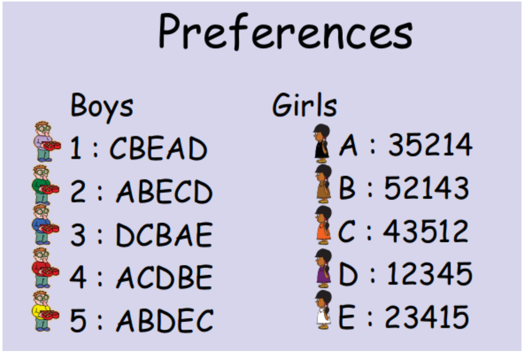
Final Pairings



The "Boy-greedy" Algorithm

Rogue Couples





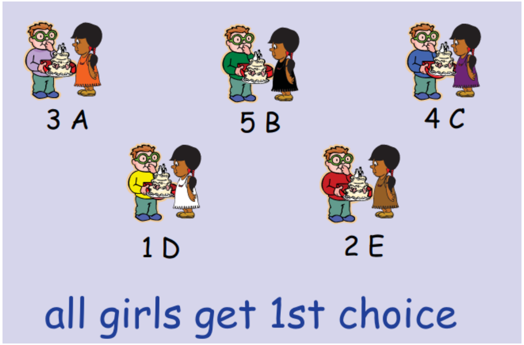
Preferences

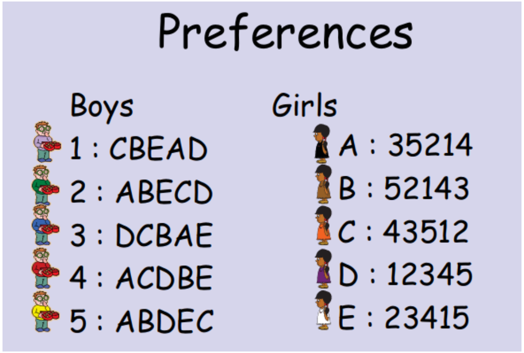
Boys	Girls
1 : CBEAD	A : 35214
2 : ABECD	B : 52143
3 : DCBAE	C : 43512
4 : ACDBE	D : 12345
5 : ABDEC	E : 23415

Girl C likes Boy 4 better than Boy 1. Boy 4 likes Girl C better than Girl B.
Can we find a pairing without rogue couples?

A stable matching

Using a Girl Greedy algorithm





Preferences

Boys	Girls
1 : CBEAD	A : 35214
2 : ABECD	B : 52143
3 : DCBAE	C : 43512
4 : ACDBE	D : 12345
5 : ABDEC	E : 23415

Why is the Stable Marriage Problem Important?

- School Admissions in the US
  - Matching school preference and student preference
- Server/Client Request Matching
  - In large webpages, multiple HTTP servers serve the same page for multiple clients;
  - Servers are matched to clients by geolocation, etc;
- Etc...

The "Mating Ritual" Algorithm

Let us describe an algorithm to **always** find a stable matching:

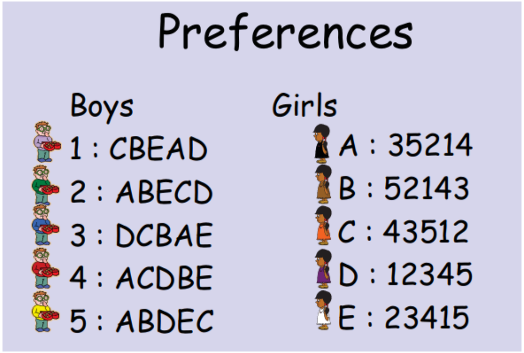
- States:
  - Each boy is proposing to some girl.
  - Each girl has a list of proposers.
- **Start State**: Every boy is proposing to their favorite girl.

Algorithm:

1. If all girls have ≤ 1 proposers in their list, they are paired and the algorithm ends;
2. Any girl with > 1 proposers in their list reject all except their favorite proposer;
3. If a boy is rejected, they propose to the next girl in their list;
4. Return to (1).

The Mating Ritual Algorithm

Example



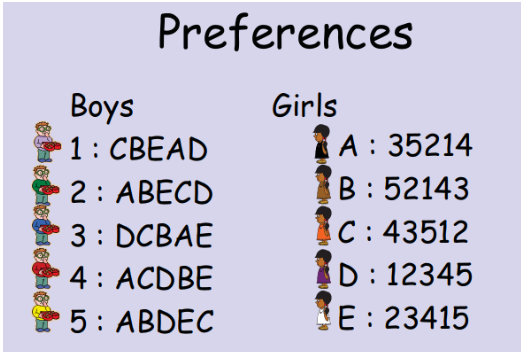
Preferences

Boys	Girls
1 : CBEAD	A : 35214
2 : ABECD	B : 52143
3 : DCBAE	C : 43512
4 : ACDBE	D : 12345
5 : ABDEC	E : 23415

- **iter 1**: No rejections. Proposals:
  - A: 2, 4, 5
  - B:
  - C: 1
  - D: 3
  - E:
- **iter 2**: A rejects 2 and 4. Proposals:
  - A: 5
  - B: 2
  - C: 1, 4
  - D: 3
  - E:

The Mating Ritual Algorithm

Example



Preferences

Boys	Girls
1 : CBEAD	A : 35214
2 : ABECD	B : 52143
3 : DCBAE	C : 43512
4 : ACDBE	D : 12345
5 : ABDEC	E : 23415

- **iter 3**: C rejects 1. Proposals:
  - A: 5
  - B: 1, 2
  - C: 4
  - D: 3
  - E:
- **iter 4**: B rejects 1. Proposals:
  - A: 5
  - B: 2
  - C: 4
  - D: 3
  - E: 1

The "Mating Ritual" Algorithm

Proof of Correctness

To proof the correctness of an algorithm, requires that you demonstrate two facts:

- The algorithm stops at some point after the start state;
- The algorithm is correct when it stops;

Proof of Correctness

The algorithm stops

Every day, the total number of girls in the boy's lists is reduced.

- Every day, **At least one boy** is rejected by **at least one girl**
  - If no boy is rejected, it means that all girls have ≤ 1 boys in their list
  - If all girls have ≤ 1 boys in their list, the algorithm stops;
- At some point, every boy's list will have **no girls**:
  - A boy with no girls in their list will propose to no one.
  - If no boys propose, then all girl's lists are empty.
  - Then the algorithm stops.

The total size of "Boy's Lists" is strictly decreasing, so the algorithm is guaranteed to stop.

Proof of Correctness: No rogue couples

- **Lemma 1**: The rank of a girl's favorite is **weakly increasing**
  Every iteration, the girl rejects a favorite **iff** she finds a better one.
- **Lemma 2**: The rank of a boy's favorite is **weakly decreasing**
  Every iteration, the boy stays with current favorite, or is rejected and goes to the next lower one.

Proof of Correctness: No rogue couples

Invariant: If G_i is not on B_j list, she has a better curent favorite.

- At the beginning of the algorithm, G_i is on B_j list;
- G_i will reject a boy proposing to her only if a better favorite is also proposing to her;
- This implies that a girl's favorite never get worse (lemma 1)

Proof of Correctness: No rogue couples

Lemma: When boy B_i is paired, he cannot form a rogue couple.

Proof by Cases:

- **Case 1:** B_i tries to form a rogue couple with someone not on his list. However, by Invariant, any girl not on his list has a better favorite, and no rogue couple is possible.
- **Case 2:** B_i tries to form a rogue couple with someone on his list. However, by Lemma 2, B_i always propose to the best girl in his list, and no rogue couple is possible.

Therefore, no rogue couple is possible.

Slide Credits

Image Credits I

1. Isomorphism image from MIT OCW materials
2. Isomorphism image from MIT OCW materials