BLG 336E

Analysis of Algorithms II

Lecture 2:

Introduction, Stable Matching, and Gale-Shapley Algorithm

Announcement

- Single Section
- Online Courses
- In-class Exams

Syllabus and Grading

Grading

- 3 Programming Projects (30%)
- 1 Midterm (30%)
- Final (40%)

Please note:

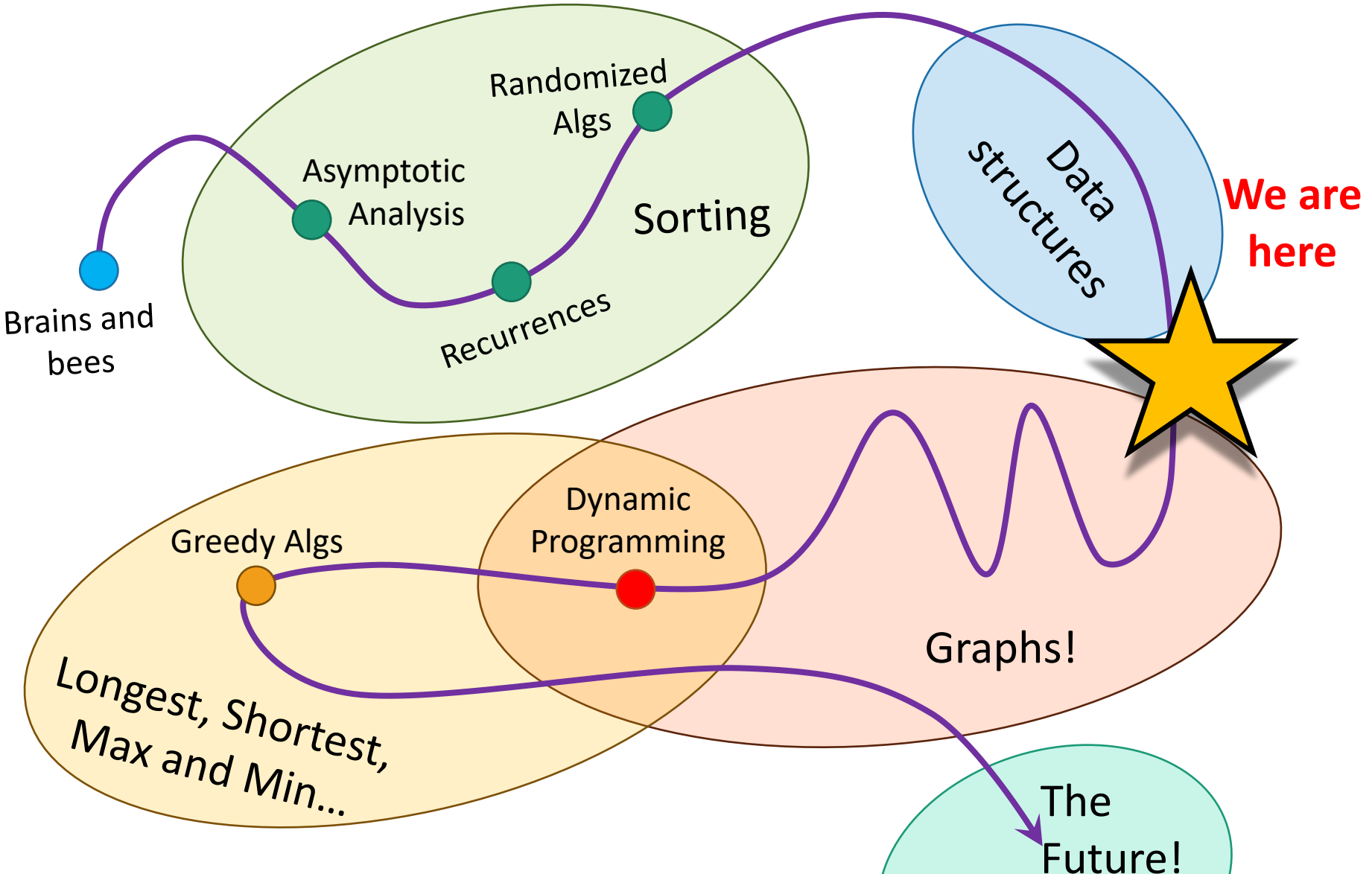
- VF Condition **15/50 (First 2 Hws+Midterm)**

Week	Date	Topics
1	22 Feb	Introduction. Some representative problems
2	1 March	Stable Matching
3	8 March	Basics of algorithm analysis.
4	15 March	Graphs (Project 1 announced)
5	22 March	Greedy algorithms I
6	29 March	Greedy algorithms II (Project 2 announced)
7	5 April	Divide and conquer
8	12 April	Midterm
9	19 April	Dynamic Programming I
10	26 April	Dynamic Programming II (Project 3 announced)
11	3 May	BREAK
12	10 May	Network Flow-I
13	17 May	Network Flow II
14	24 May	NP and computational intractability I
15	31 May	NP and computational intractability II

Homework!

- 3 Homeworks each %10 (Total %30)
- Use C++ and object oriented approach in your assignments.
- The goal is to practice your implementation skills, so copy-pasting is not encouraged.
- There will be explicit instructions on which files to submit, how it should be compiled, example cases etc.
- Some examples will be provided so that you can test your program yourselves.
- However, your program will be graded based on how good it is in solving the given problem.

Roadmap


Randomized Algs
Asymptotic Analysis
Sorting
Brains and bees
Recurrences
Data structures
We are here
Dynamic Programming
Greedy Algs
Graphs!
Longest, Shortest, Max and Min…
The Future!

Let’s start at the beginning

Muhammad ibn Mūsā al-Khwārizmī 780-850

The words **'algorithm**' and **'algorism**' come from the name **al-Khwārizmī**. Al-Khwārizmī (Persian: خوارزمی, 8th century) was a Persian *mathematician, astronomer, geographer, and scholar* in the House of Wisdom in Baghdad.

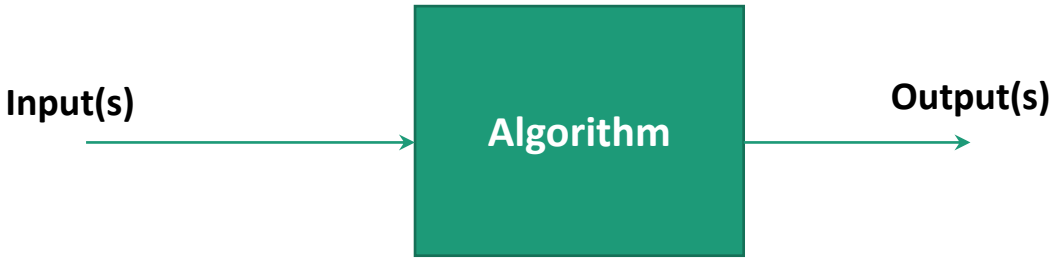
About 825, he wrote a treatise in the Arabic language, which was translated into Latin in the 12th century under the title Algoritmi de numero Indorum. This title means "Algoritmi on the numbers of the Indians", where **"Algoritmi" was the translator's Latinization of Al-Khwarizmi's name**. Al-Khwarizmi was the most widely read mathematician in Europe in the late Middle Ages, primarily through his book, the Algebra.

His name has taken on a special significance in computer science, where the word “algorithm” has come to refer to a method that can be used by a computer for the solution of a problem.

In Class discussion: what is an algorithm?

- Program?
- Function?

A Picture of an Algorithm



First Algorithm

- It is in human nature to think algorithmically!
- Long before computers were invented, people could create precise algorithms that could compute. One of the first algorithms (computing the greatest common divisor) is 300 years older than the Bible! It was written by Euclid (~300 BC).

- Here is an idea:

Gcd(m,n) = n, if m **mod** n = 0 or

Gcd(n, (m **mod** n)) otherwise.

Gcd (54,24)

Use Euclid's algorithm to determine the greatest common divisor of 54 and 24?

Gcd(m,n) = n, if m **mod** n = 0

or Gcd(n, (m **mod** n)) otherwise

Gcd (54,24)

Use Euclid's algorithm to determine the greatest common divisor of 54 and 24?

Gcd(m,n) = n, if m **mod** n = 0

or Gcd(n, (m **mod** n)) otherwise

Solution:

Gcd(54,24) = ?
54 mod 24 = 6 (=54 – 2 x 24)
Gcd(24,6) = 6 (since 24 **mod** 6 = 0)

First Algorithm

What is the greatest common divisor of 54 and 24?

The number 54 can be expressed as a product of two integers in several different ways:

$$54 \times 1 = 27 \times 2 = 18 \times 3 = 9 \times 6.$$

Thus the **divisors of 54** are: $1, 2, 3, 6, 9, 18, 27, 54.$

Similarly, **the divisors of 24** are: $1, 2, 3, 4, 6, 8, 12, 24.$

The numbers that these two lists share in common are the **common divisors** of 54 and 24:

$$1, 2, 3, 6.$$

The greatest of these is 6. That is, the **greatest common divisor** of 54 and 24. One writes:

$$\gcd(54, 24) = 6.$$

"Homework": implement the Euclidean algorithm in the language of your choice.

First Algorithm (Euclid's) = **function**

$$f(x) = y$$

input(s) output(s)

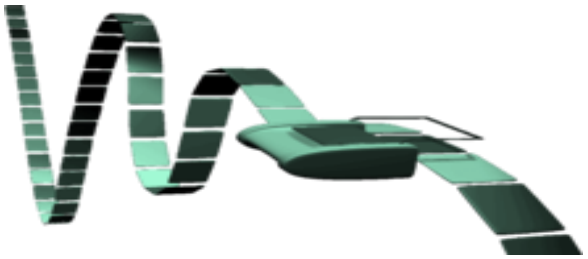
$$GCD(x_1, x_2) = y$$

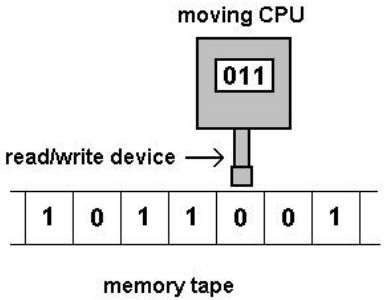
How to think about an algorithm differently?

Alan Turing. 1912 - 1954

Alan Turing was an English **mathematician, logician, cryptanalyst and computer scientist**. He was influential in the development of computer science and providing a formalization of the concept of the algorithm and computation with the Turing machine, playing a significant role in the creation of the modern computer

Turing Machine





A Turing machine is a kind of a *state machine*. At any time the machine is in any one of a finite number of states.

A Turing machine has an infinite one-dimensional *tape* divided into cells. Traditionally we think of the tape as being horizontal with the cells arranged in a left-right orientation. The tape has one end, at the left say, and stretches infinitely far to the right. Each cell is able to contain one symbol, '0' or '1'.

The machine has a *read-write head*, which scans a single cell on the tape. This read-write head can move left and right along the tape to scan successive cells.

The **action of a Turing machine** is determined completely by (1) the current state of the machine (2) the symbol in the cell currently being scanned by the head and (3) a table of transition rules, which serve as the "program" for the machine.

Turing Machine

Each transition rule is a 4-tuple:

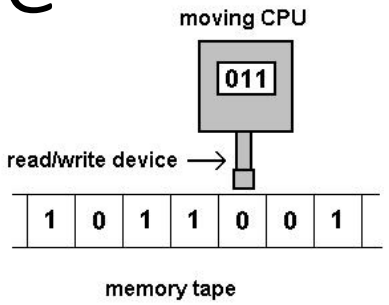
$\langle\ State_0, Symbol, State_{next}, Action\ \rangle$

which can be read as saying "if the machine is in state $State_0$ and the current cell contains *Symbol* then move into state $State_{next}$ taking *Action*".

The actions available to a Turing machine are either to **write a symbol on the tape in the current cell** (which we will denote with the symbol in question), or to **move the head one cell to the left or right**.

If the machine reaches a situation in which there is not exactly one transition rule specified, i.e., none or more than one, then the machine halts.

In modern terms, the tape serves as the memory of the machine, while the read-write head is the memory bus through which data is accessed (and updated) by the machine.



A **function** will be **Turing-computable** if there exists a set of instructions that will result in the machine computing the function, regardless of the amount of time it takes.

Although the device looks **primitive**,
even very complex modern computers can perform *only* Turing-computable tasks!

ARTIFICIAL INTELLIGENCE

Deep neural reasoning

The human brain can solve highly abstract reasoning problems using a neural network that is entirely physical. The underlying mechanisms are only partially understood, but an artificial network provides valuable insight. SEE ARTICLE P.471

HERBERT JAEGER

A classic example of logical reasoning is the syllogism, "All men are mortal. Socrates is a man. Therefore, Socrates is mortal." According to both ancient and modern views[1], reasoning amounts to a rule-based mental manipulation of symbols — in this example, the words 'All', 'men', and so on. But human brains are made of neurons that operate by exchanging jittery electrical pulses, rather than word-like symbols. This difference encapsulates a notorious scientific and philosophical enigma, sometimes referred to as the neural–symbolic integration problem[2], which remains unsolved. On page 471, Graves *et al.*[3] use the machine-learning methods of 'deep learning' to impart some crucial symbolic-reasoning mechanisms to an artificial neural system. Their system can solve complex tasks by learning symbolic-reasoning rules from examples, an achievement that has potential implications for the neural–symbolic integration problem.

A key requirement for reasoning is a working memory. In digital computers, this role is served by the random-access memory (RAM). When a computer reasons — when it executes a program — information is bundled together in working memory in ever-changing combinations. Comparing human reasoning to the running of computer programs is not a far-fetched metaphor. In fact, a venerable historical alley leads from Aristotle's definition of syllogisms to the modern model of a programmable computer (the Turing machine). Alan Turing himself used 'mind' language in his groundbreaking work[4]: "The behaviour of the computer at any moment is determined by the symbols which he is observing and his 'state of mind' at that moment."

The authors' neural system cannot and need not be programmed — instead, it is trained.

Although there are clear parallels between human reasoning and the running of computer programs, we lack an understanding of how either of them could be implemented in biological or artificial neural networks. Graves

Church-Turing thesis

Everything computable is computable by a Turing Machine.

Alonzo Church
(1903–1995)

Turing's Algorithm = **program**

"a series of instructions that can be put into a computer in order to make it perform an operation"

Is an algorithm a function or a program?

- It can be both:

1. **Declarative/functional** (Euclid's way of thinking)
2. **Imperative/ via machine instructions** (Turing's way of thinking)

What is an Algorithm?

All algorithms must satisfy the following criteria:

1. Input. Zero or more quantities are externally supplied.
2. Output. At least one quantity is produced.
3. Definiteness. Each instruction is clear and unambiguous. E.g., “add 6 or 7 to x” is not permitted.
4. Finiteness. If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps. (Termination!)

We can also add “effectiveness”. Every instruction must be very basic so that it can be carried out, in principle, by a person using only pencil and paper; it must be feasible.

4 questions to ask about algorithms

- **How to devise algorithms?**

Creating an algorithm is an art which will never be fully automated.

During your studies, you can learn useful techniques.

- **How to validate/verify algorithms?**

Once an algorithm is devised, it is necessary to show that it computes a correct answer for all possible inputs —this is called algorithm validation.

Once the validity of the method is shown, a program can be written. Then a program verification is needed.

- **How to analyze algorithms?**

Performance analysis determines how much computing time and storage an algorithm requires [in best case, in worst case, and in average].

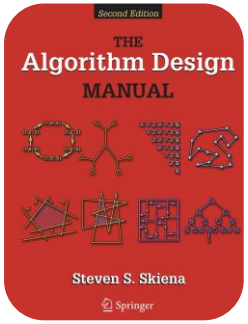
- **How to test a program?** Testing a program has two phases – debugging and profiling.

"Debugging can only point to the presence of errors, and not to their absence!" A proof of correctness is much more certain.

Profiling is the process of executing a correct program on data sets and measuring the time and space required.

How do we know that an algorithm is correct?

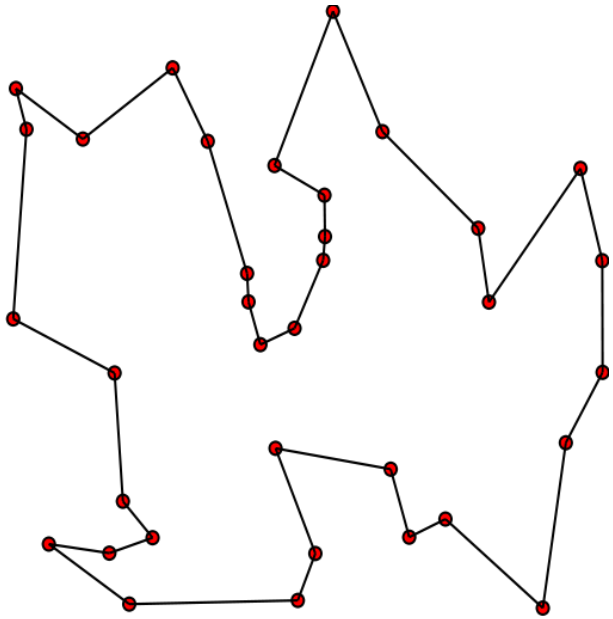
There are three desirable properties for a good algorithm. We seek algorithms that are *correct* and *efficient*, while being *easy to implement*. These goals may not be simultaneously achievable. In industrial settings, any program that seems to give good enough answers without slowing the application down is often acceptable, regardless of whether a better algorithm exists. The issue of finding the best possible answer or achieving maximum efficiency usually arises in industry only after serious performance or legal troubles.



algorithm correctly solves a given problem. Correct algorithms usually come with a proof of correctness, which is an explanation of *why* we know that the algorithm must take every instance of the problem to the desired result. However, before we go further we demonstrate why *"it's obvious"* never suffices as a proof of correctness, and is usually flat-out wrong.

Shortest travel distance in a graph

The **travelling salesman problem** (**TSP**) asks the following question: *"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"*



Stop right now and think up and algorithm to solve this problem (make the travelling salesman happy!).

Nearest Neighbour Tour

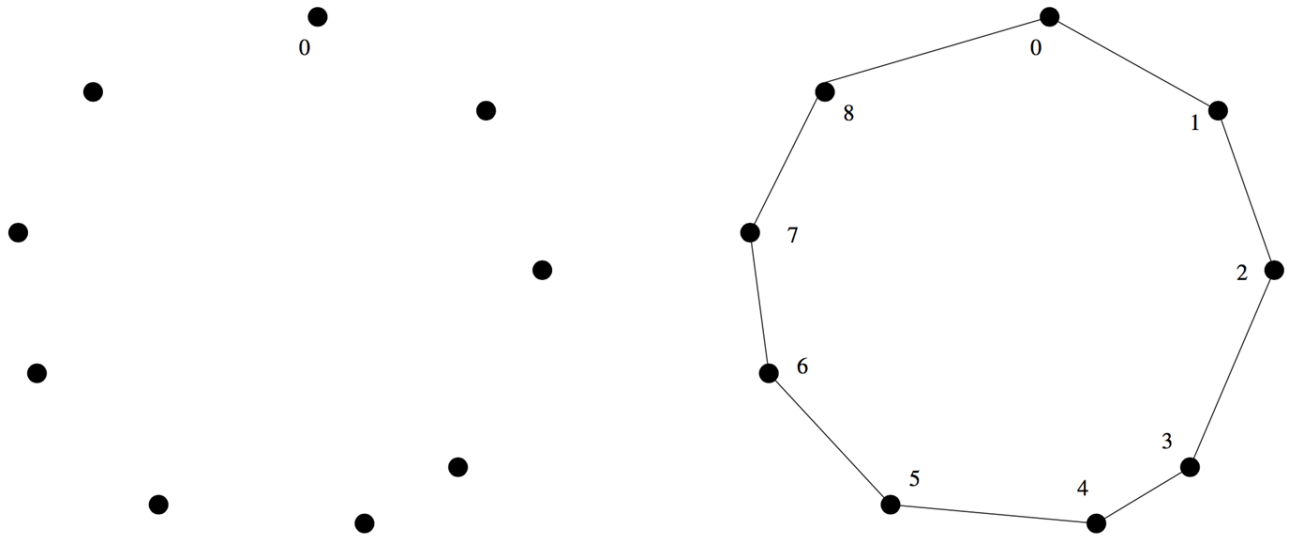
- A popular solution starts at some point p0 and then walks to its nearest unvisited neighbor p1, then repeats from p1, etc. Until done.

NearestNeighbor(P)
    Pick and visit an initial point p_0 from P
    $p = p_0$
    $i = 0$
    While there are still unvisited points
        $i = i + 1$
        Select p_i to be the closest unvisited point to p_{i-1}
        Visit p_i
    Return to p_0 from p_{n-1}

Is this solution correct?

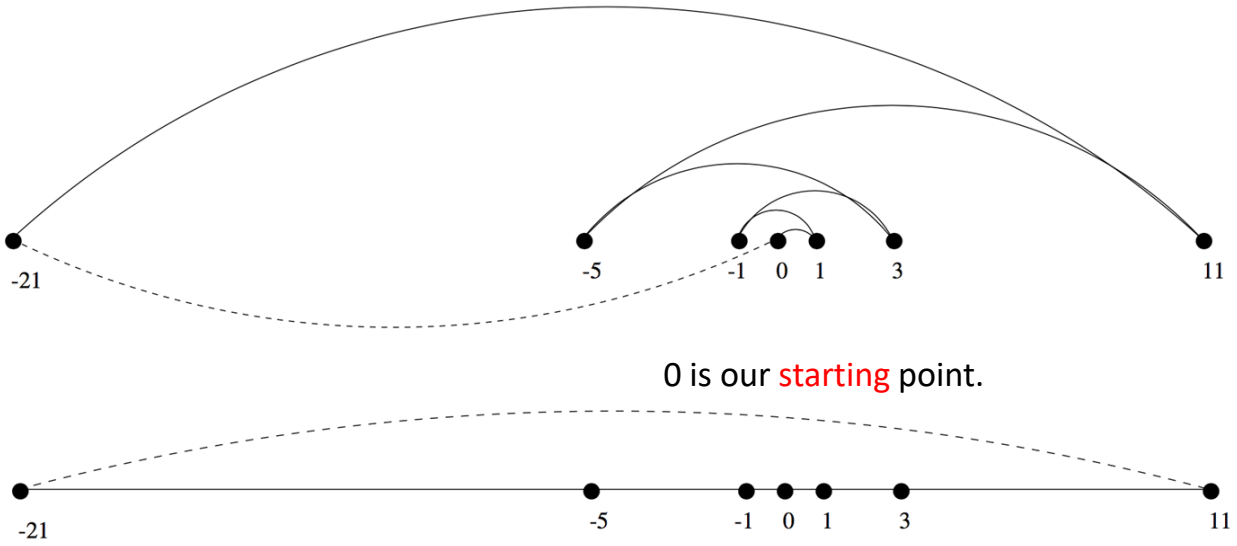
Let us try another example …

Nearest Neighbor Algorithm works perfectly for this case too! 😊



To make sure that NN algorithm is a correct solution to the TSP problem, we need to keep testing it on different examples.

What about this example?



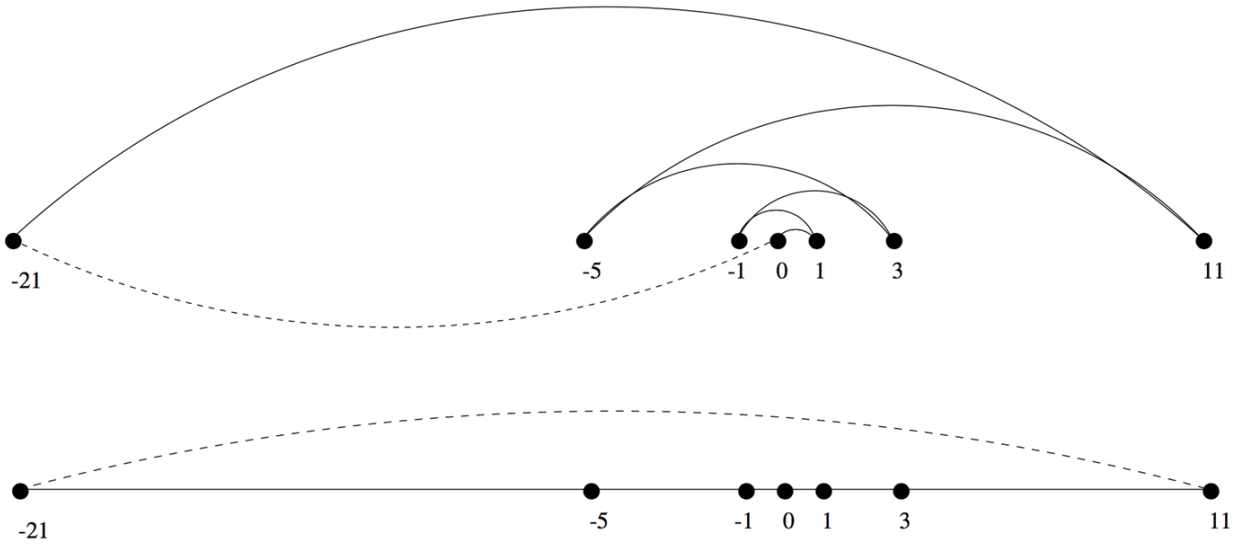
Closest Pair Tour

- Another idea is to repeatedly connect the closest pair of points whose connection will not cause a cycle or a three-way branch, until all points are in one tour.

ClosestPair(P)

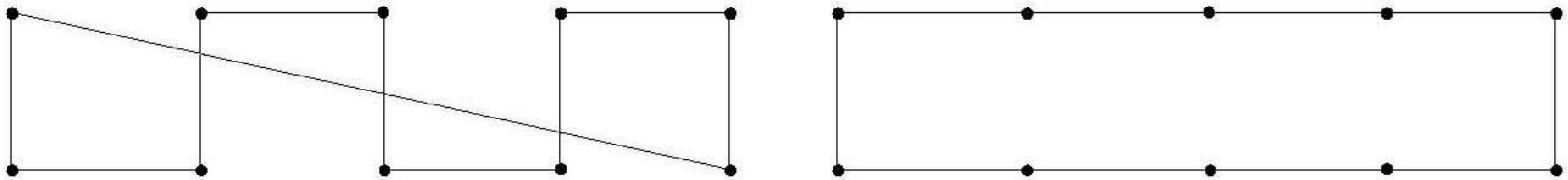
Let n be the number of points in set P.
For $i = 1$ to $n - 1$ do
$d = \infty$
For each pair of endpoints (s, t) from distinct vertex chains
if $dist(s, t) \leq d$ then $s_m = s$, $t_m = t$, and $d = dist(s, t)$
Connect (s_m, t_m) by an edge
Connect the two endpoints by an edge

What about this example?



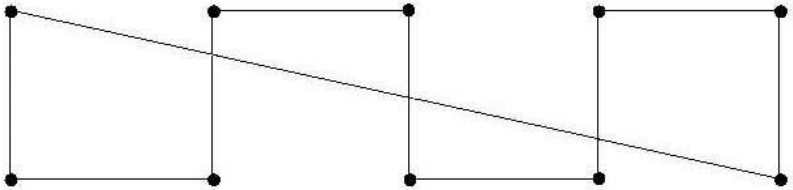
A good solution

What about this example? Does the Closest Pair Tour work on this?

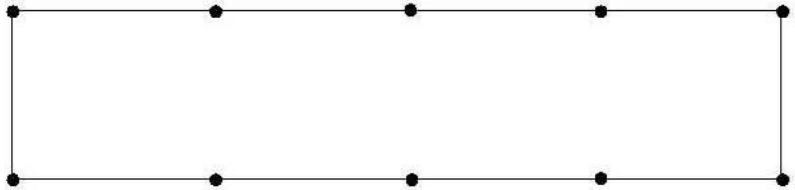


Although it works correctly on the previous example, other data causes trouble.

What about this example? Does the Closest Pair Tour work on this?



A good solution



Although it works correctly on the previous example, other data causes trouble.

Does a GOOD algorithm for the problem exist?

A Correct Algorithm: Exhaustive Search

- We could try all possible orderings of the points, then select the one which minimizes the total length:

OptimalTSP(P)
$d = \infty$
For each of the $n!$ permutations P_i of point set P
If $(cost(P_i) \leq d)$ then $d = cost(P_i)$ and $P_{min} = P_i$
Return P_{min}

- Since all possible orderings are considered, **we are guaranteed to end up with the shortest possible tour**.

Factorial function:
n! = 1*2*3*...*n (e.g. 3! = 1*2*3 = 6)

Exhaustive Search is Slow!

Because it tries all **n!** permutations, it is extremely slow to use when there are more than 10-20 points.

The fastest computer in the world couldn't hope to enumerate all the

20! =2,432,902,008,176,640,000 orderings of 20 points within a day.

[For n=1000, will not be achieved in your lifetime!!!]

No efficient and correct algorithm exists for the *travelling salesman problem.*

William Rowan Hamilton
(1805–1865)
Irish mathematician

*The **problem of joining dots** is also known as "**Hamiltonian cycle**"*

Take-Home Lesson: There is a fundamental difference between *algorithms*, which always produce a correct result, and *heuristics*, which may usually do a good job but without providing any guarantee.

Take-Home Lesson: An important and honorable technique in algorithm design is to narrow the set of allowable instances until there *is* a correct and efficient algorithm. For example, we can restrict a graph problem from general graphs down to trees, or a geometric problem from two dimensions down to one.

Take-Home Lesson: Searching for counterexamples is the best way to disprove the correctness of a heuristic.

Searching for **counterexamples** is the best way to disprove the correctness of an algorithm.

If the algorithm works some cases and fails in others, it is generally called a *heuristic*

"In which case my algorithm might fail?"

Think algorithmically 😊
Top interview questions on algorithms

- Sorting (plus searching binary search)
- Divide-and-conquer
- Dynamic programming / memoization
- Greediness
- Recursion
- Algorithms associated with a specific data structure

A sample of interview questions

- How to find middle element in a linked list through a single pass?
- Write a Java program to sort an array using Bubble Sort algorithm?
- How to reverse a linked list using recursion and iteration?
- Design an algorithm to find all the common elements in two sorted lists of numbers.

1.1 A First Problem: Stable Matching

Matching Residents to Hospitals

Goal. Given a set of preferences among hospitals and medical school students, design a self-reinforcing admissions process.

Unstable pair: applicant x and hospital y are unstable if:

- x prefers y to its assigned hospital.
- y prefers x to one of its admitted students.

Stable assignment. Assignment with no unstable pairs.

- Natural and desirable condition.
- Individual self-interest will prevent any applicant/hospital deal from being made.

Stable Matching Problem

Perfect matching: everyone is matched monogamously.

- Each man gets exactly one woman.
- Each woman gets exactly one man.

Stability: no incentive for some pair of participants to undermine assignment by joint action.

- In matching M, an unmatched pair m-w is unstable if man m and woman w prefer each other to current partners.
- Unstable pair m-w could each improve by eloping.

Stable matching: perfect matching with no unstable pairs.

Stable matching problem. Given the preference lists of n men and n women, find a stable matching if one exists.

Stable Matching Problem

Goal. Given n men and n women, find a "suitable" matching.

- Participants rate members of opposite sex.
- Each man lists women in order of preference from best to worst.
- Each woman lists men in order of preference from best to worst.

favorite ↓ least favorite ↓

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

favorite ↓ least favorite ↓

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

Women's Preference Profile

Stable Matching Problem

Q. Is assignment X-C, Y-B, Z-A stable?

favorite ↓ least favorite ↓

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

favorite ↓ least favorite ↓

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

Women's Preference Profile

Stable Matching Problem

Q. Is assignment X-C, Y-B, Z-A stable?

A. No. Bertha and Xavier will hook up.

favorite ↓ least favorite ↓

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

favorite ↓ least favorite ↓

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

Women's Preference Profile

Stable Matching Problem

Q. Is assignment X-A, Y-B, Z-C stable?

A. Yes.

favorite ↓ least favorite ↓

	1st	2nd	3rd
Xavier	Amy	Bertha	Clare
Yancey	Bertha	Amy	Clare
Zeus	Amy	Bertha	Clare

Men's Preference Profile

favorite ↓ least favorite ↓

	1st	2nd	3rd
Amy	Yancey	Xavier	Zeus
Bertha	Xavier	Yancey	Zeus
Clare	Xavier	Yancey	Zeus

Women's Preference Profile

Stable Roommate Problem

Q. Do stable matchings always exist?

A. Not obvious a priori.

is core of market *(a housing term)* nonempty?

Stable roommate problem.

- 2n people; each person ranks others from 1 to 2n-1.
- Assign roommate pairs so that no unstable pairs.

	1st	2nd	3rd
Adam	B	C	D
Bob	C	A	D
Chris	A	B	D
Doofus	A	B	C

A-B, C-D ⇒ B-C unstable
A-C, B-D ⇒ A-B unstable
A-D, B-C ⇒ A-C unstable

Observation. Stable matchings do not always exist for stable roommate problem.

Propose-And-Reject Algorithm

Propose-and-reject algorithm. [Gale-Shapley 1962] Intuitive method that guarantees to find a stable matching.

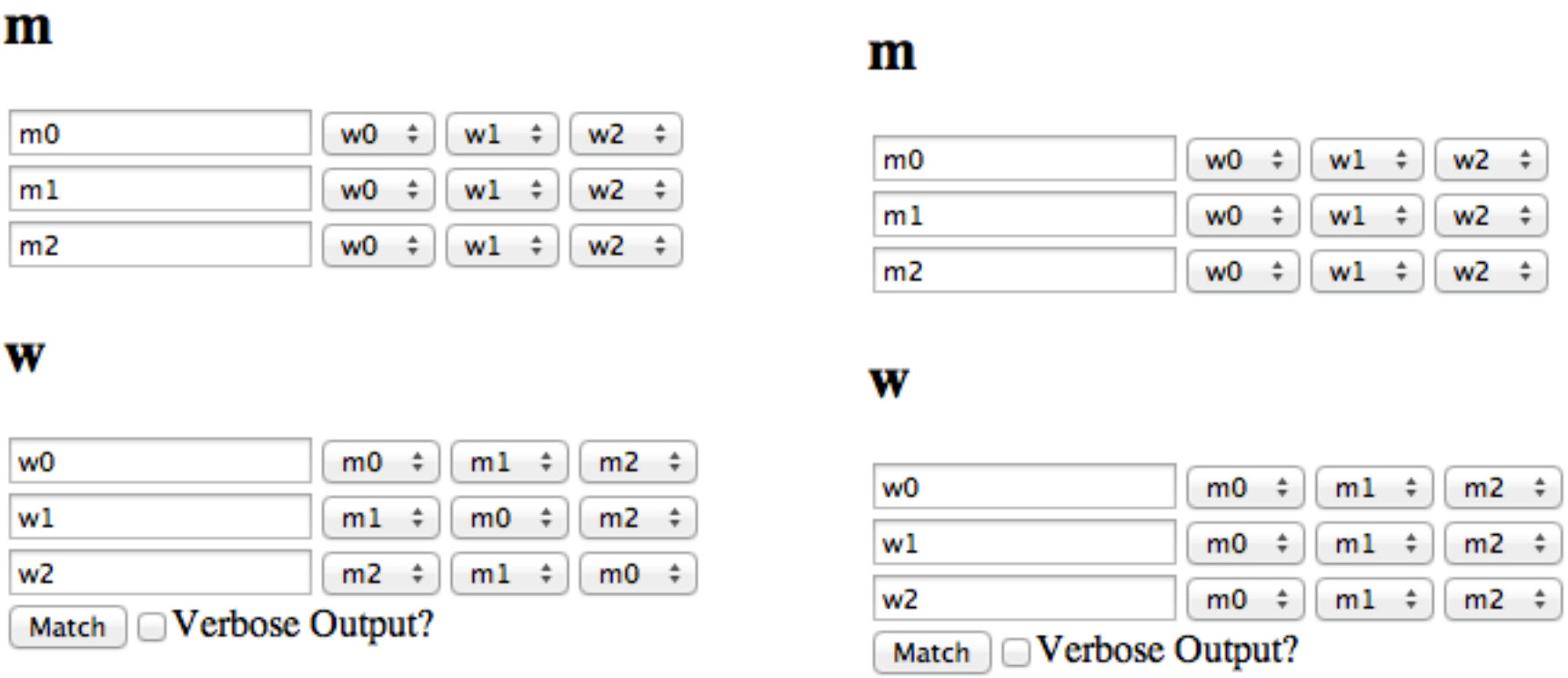
```
Initialize each person to be free.
while (some man is free and hasn't proposed to every woman) {
    Choose such a man m
    w = 1st woman on m's list to whom m has not yet proposed
    if (w is free)
        assign m and w to be engaged
    else if (w prefers m to her fiancé m')
        assign m and w to be engaged, and m' to be free
    else
        w rejects m
}
```

Algorithm Flow

The Algorithm (Loop):

3. One individual from the proposing group who is not already engaged will propose to their most preferable option who has not already rejected them.
4. The person being proposed to will:
    - Accept if this is their first offer.
    - Reject if this is worse than their current offer.
    - Accept if this is better than their current offer. In this case they will jilt their previous offer.

Demo (from: http://sephlietz.com/gale-shapley/)



Proof of Correctness: Termination

Observation 1. Men propose to women in decreasing order of preference.

Observation 2. Once a woman is matched, she never becomes unmatched; she only "trades up."

Claim. Algorithm terminates after at most n^2 iterations of while loop.
Pf. Each time through the while loop a man proposes to a new woman. There are only n^2 possible proposals. ·

	1st	2nd	3rd	4th	5th
Victor	A	B	C	D	E
Wyatt	B	C	D	A	E
Xavier	C	D	A	B	E
Yancey	D	A	B	C	E
Zeus	A	B	C	D	E

	1st	2nd	3rd	4th	5th
Amy	W	X	Y	Z	V
Bertha	X	Y	Z	V	W
Clare	Y	Z	V	W	X
Diane	Z	V	W	X	Y
Erika	V	W	X	Y	Z

n(n-1) + 1 proposals required

Proof of Correctness: Perfection

Claim. All men and women get matched.

Pf. (by contradiction)

- Suppose, for sake of contradiction, that Zeus is not matched upon termination of algorithm.
- Then some woman, say Amy, is not matched upon termination.
- By Observation 2, Amy was never proposed to.
- But, Zeus proposes to everyone, since he ends up unmatched. ▪

	1st	2nd	3rd	4th	5th
Victor	A	B	C	D	E
Wyatt	B	C	D	A	E
Xavier	C	D	A	B	E
Yancey	D	A	B	C	E
Zeus	A	B	C	D	E

	1st	2nd	3rd	4th	5th
Amy	W	X	Y	Z	V
Bertha	X	Y	Z	V	W
Clare	Y	Z	V	W	X
Diane	Z	V	W	X	Y
Erika	V	W	X	Y	Z

- Note: Please read the following to review proof of contradiction:
- http://zimmer.csufresno.edu/~larryc/proofs/proofs.contradict.html
- Proofs in general: http://zimmer.csufresno.edu/~larryc/proofs/proofs.html

Proof of Correctness: Stability

Claim. No unstable pairs.

Pf. (by contradiction)

- Suppose A-Z is an unstable pair: each prefers each other to partner in Gale-Shapley matching S*.

- Case 1: Z never proposed to A.
  - ⇒ Z prefers his GS partner to A.
  - ⇒ A-Z is stable.

men propose in decreasing order of preference

S*

Amy-Yancey
Bertha-Zeus
. . .

- Case 2: Z proposed to A.
  - ⇒ A rejected Z (right away or later)
  - ⇒ A prefers her GS partner to Z.
  - ⇒ A-Z is stable.

← women only trade up

- In either case A-Z is stable, a contradiction. ▪

Summary

Stable matching problem. Given n men and n women, and their preferences, find a stable matching if one exists.

Gale-Shapley algorithm. Guarantees to find a stable matching for any problem instance.

Q. How to implement GS algorithm efficiently?

Q. If there are multiple stable matchings, which one does GS find?

Efficient Implementation

Efficient implementation. We describe $O(n^2)$ time implementation.

Representing men and women.

- Assume men are named 1, …, n.
- Assume women are named 1', …, n'.

Engagements.

- Maintain a list of free men, e.g., in a queue.
- Maintain two arrays `wife[m]`, and `husband[w]`.
  - set entry to `0` if unmatched
  - if m matched to w then `wife[m]=w` and `husband[w]=m`

Men proposing.

- For each man, maintain a list of women, ordered by preference.
- Maintain an array `count[m]` that counts the number of proposals made by man `m`.

Efficient Implementation

Women rejecting/accepting.

- Does woman `w` prefer man `m` to man `m'`?
- For each woman, create inverse of preference list of men.
- Constant time access for each query after O(n) preprocessing.

Amy	1st	2nd	3rd	4th	5th	6th	7th	8th
Pref	8	3	7	1	4	5	6	2

Amy	1	2	3	4	5	6	7	8
Inverse	4th	8th	2nd	5th	6th	7th	3rd	1st

Amy prefers man 3 to 6 since `inverse[3] < inverse[6]`

2 7

```
for i = 1 to n
    inverse[pref[i]] = i
```

Understanding the Solution

Q. For a given problem instance, there may be several stable matchings. Do all executions of Gale-Shapley yield the same stable matching? If so, which one?

	1st	2nd	3rd
Xavier	A	B	C
Yancey	B	A	C
Zeus	A	B	C

	1st	2nd	3rd
Amy	Y	X	Z
Bertha	X	Y	Z
Clare	X	Y	Z

Understanding the Solution

Q. For a given problem instance, there may be several stable matchings. Do all executions of Gale-Shapley yield the same stable matching? If so, which one?

An instance with two stable matchings.

- X-A, Y-B, Z-C.
- Y-A, X-B, Z-C.

	1st	2nd	3rd
Xavier	A	B	C
Yancey	B	A	C
Zeus	A	B	C

	1st	2nd	3rd
Amy	Y	X	Z
Bertha	X	Y	Z
Clare	X	Y	Z

Understanding the Solution

Q. For a given problem instance, there may be several stable matchings. Do all executions of Gale-Shapley yield the same stable matching? If so, which one?

Def. Man m is a valid partner of woman w if there exists some stable matching in which they are matched.

Man-optimal assignment. Each man receives best valid partner.

Claim. All executions of GS yield man-optimal assignment, which is a stable matching!

- No reason a priori to believe that man-optimal assignment is perfect, let alone stable.
- Simultaneously best for each and every man.

Define: S* ={(m,best(m)): m in M} where best(m) is the best valid partner of m

Examples for S* = {(m,best(m)): m in M}

An instance with two stable matchings.

- X-A, Y-B, Z-C.
- Y-A, X-B, Z-C.

S*={(X,A),(Y,B),(Z,C)}

	1st	2nd	3rd
Xavier	A	B	C
Yancey	B	A	C
Zeus	A	B	C

	1st	2nd	3rd
Amy	Y	X	Z
Bertha	X	Y	Z
Clare	X	Y	Z

- X-A, Y-B, Z-C

S*={(X,A),(Y,B),(Z,C)}

	1st	2nd	3rd
Xavier	A	B	C
Yancey	A	B	C
Zeus	A	B	C

	1st	2nd	3rd
Amy	X	Y	Z
Bertha	X	Y	Z
Clare	X	Y	Z

man optimal matching

woman optimal matching

Man Optimality

Claim. Every execution of the GS algorithm results in the (man-optimal) set S* !

Pf. (by contradiction)

- Suppose an execution E of GS resulted in some man paired with someone who is not his best valid partner.
- Since men propose in decreasing order of preference, then there must be some man who is rejected by a valid partner during E.
- Consider the first moment during E in which some man (m) is rejected by a valid partner (w).
- men propose in decreasing order of preference AND this is the first time such a rejection occurred
  - Therefore it must be that best(m)=w
- w may have rejected m,
  - either because m proposed and w turned it down because she was already engaged with someone she prefers more, or
  - w broke her engagement to m in favor of a better proposal.
  - Let m' be the man whom w prefers to compared to m.

	1st	2nd	3rd	4th	5th
m	w				

	1st	2nd	3rd	4th	5th
w			m'		m

Man Optimality

- Since w is a valid partner of m, there exists a stable matching S' containing the pair (m,w).
- Let m' be matched with some w' ≠w in that matching S'.
  - S'={(m,w),(m',w'),...}
- Rejection of m by w was the first rejection in THEREFORE m' had not been rejected by any valid partner at the point in E when he became engaged to w.
- Since m' proposed in decreasing order of preference AND w' is a valid partner of m' THEREFORE m' prefers w to w'.
- But we have already seen that w prefers m' to m, because in E she rejected m in favor of m'.
- Since (m',w) is not in S', then (m',w) is an unstable pair in S' (because both m' and w are willing to leave their current partners and get engaged, see below)!
- This contradicts our claim that S' is stable, hence it contradicts our initial assumption.

Matching S' in which (m,w) happens

	1st	2nd	3rd	4th	5th
m	w				
m'			w	w'	

	1st	2nd	3rd	4th	5th
w			m'		m
w'					

Stable Matching Summary

Stable matching problem. Given preference profiles of n men and n women, find a stable matching.

no man and woman prefer to be with each other than assigned partner

Gale-Shapley algorithm. Finds a stable matching in $O(n^2)$ time.

Man-optimality. In version of GS where men propose, each man receives best valid partner.

w is a valid partner of m if there exist some stable matching where m and w are paired

Q. Does man-optimality come at the expense of the women?

Woman Pessimality

Woman-pessimal assignment. Each woman receives worst valid partner.

Claim. GS finds woman-pessimal stable matching S*.

Pf.

- Suppose A-Z matched in S*, but Z is not worst valid partner for A.
- There exists stable matching S in which A is paired with a man, say Y, whom she likes less than Z. ← man-optimality
- Let B be Z's partner in S.
- Z prefers A to B.
- Thus, Z-A is an unstable pair in S. ▪

Extensions: Matching Residents to Hospitals

Ex: Men ≈ hospitals, Women ≈ med school residents.

Variant 1. Some participants declare others as unacceptable.

Variant 2. Unequal number of men and women.

resident A unwilling to work in Cleveland

Variant 3. Limited polygamy.

hospital X wants to hire 3 residents

Def. Matching S unstable if there is a hospital h and resident r such that:

- h and r are acceptable to each other; and
- either r is unmatched, or r prefers h to her assigned hospital; and
- either h does not have all its places filled, or h prefers r to at least one of its assigned residents.

Application: Matching Residents to Hospitals

NRMP. (National Resident Matching Program)

- Original use just after WWII. ← predates computer usage
- Ides of March, 23,000+ residents.

Rural hospital dilemma.

- Certain hospitals (mainly in rural areas) were unpopular and declared unacceptable by many residents.
- Rural hospitals were under-subscribed in NRMP matching.
- How can we find stable matching that benefits "rural hospitals"?

Rural Hospital Theorem. Rural hospitals get exactly same residents in every stable matching!

Stable Marriage Interesting Notes

other stable marriages possible?

-can be many

More questions, rich theory

do better by lying? boys -No! girls -Yes!

CC Huang, How Hard is it to Cheat in the Gale-Shapley **…**

To our knowledge, ours is the first attempt in proposing men-*lying*

Stable Matching Publications:

- **Local search algorithms on the stable marriage problem: Experimental studies**
- Gelain, Pini, Rossi, Venable… - 2010

- **Stable marriage with ties and bounded length preference lists**
- Irving, Manlove… - Journal of Discrete Algorithms, 2009

- **Approximation algorithms for hard variants of the stable marriage and hospitals/residents problems**
- RW Irving… - Journal of Combinatorial Optimization, 2008

- **A 1.875: approximation algorithm for the stable marriage problem**
- Iwama, S Miyazaki… - Proceedings of the eighteenth …, 2007

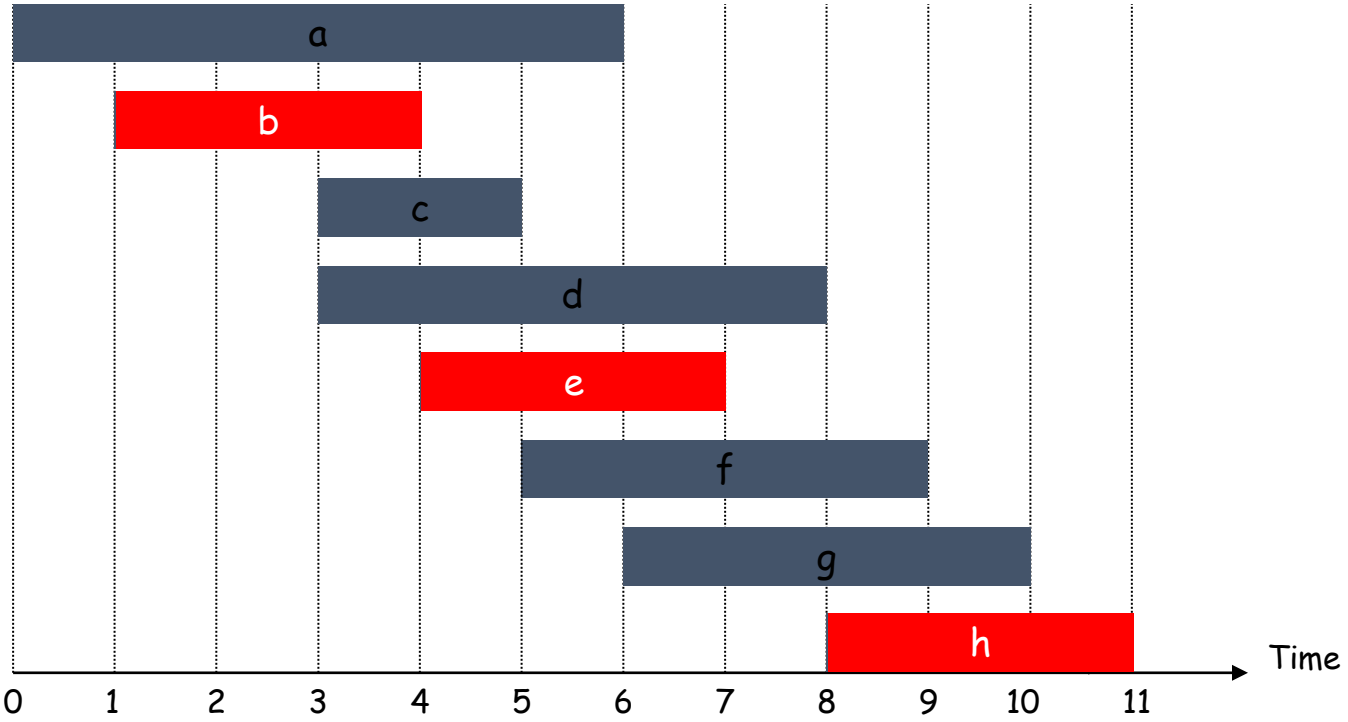
1.2 Five Representative Problems

1. Interval Scheduling

Input. Set of jobs with start times and finish times.

Goal. Find maximum cardinality subset of mutually compatible jobs.

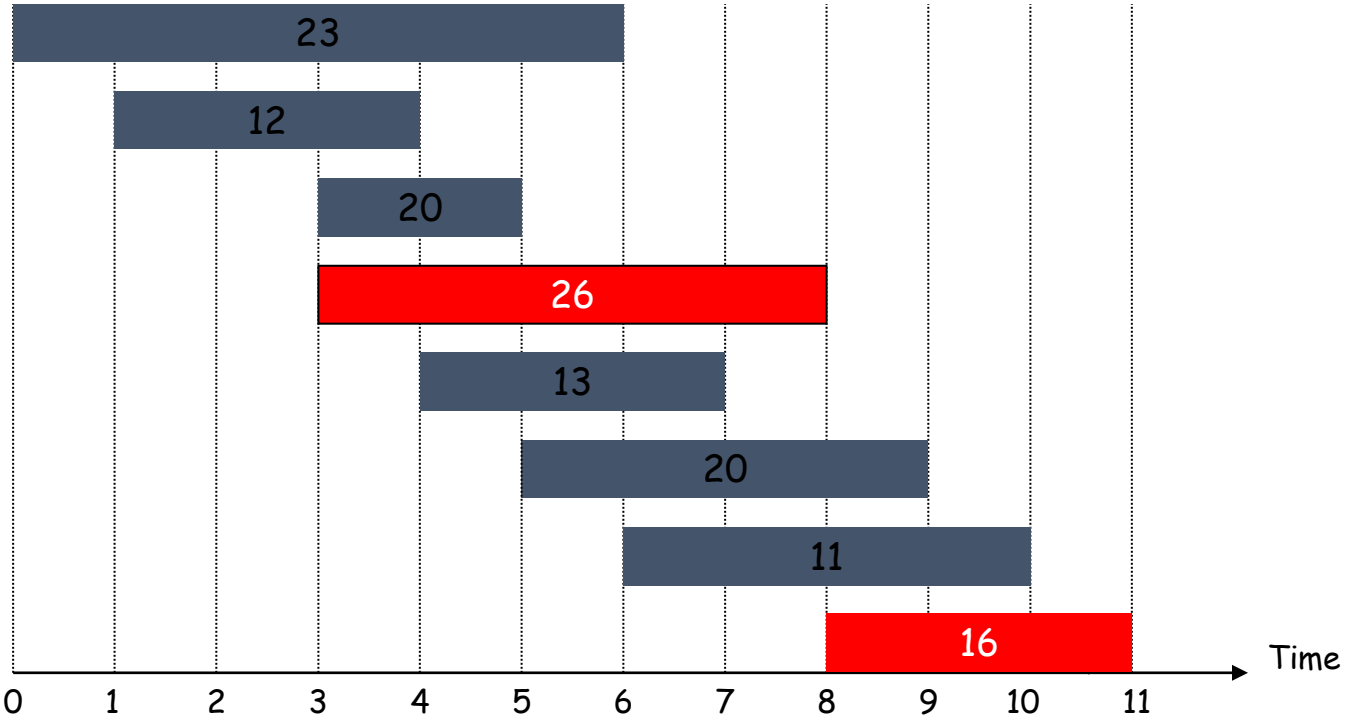
jobs don't overlap



2. Weighted Interval Scheduling

Input. Set of jobs with start times, finish times, and weights.

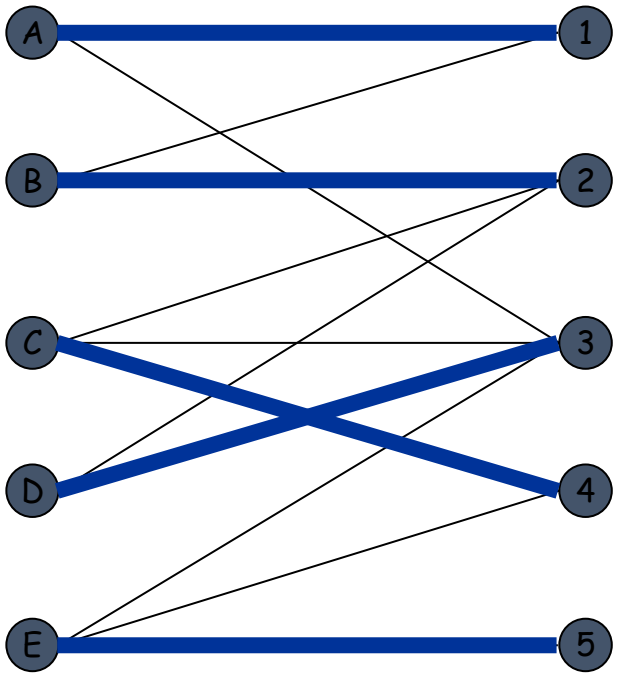
Goal. Find maximum weight subset of mutually compatible jobs.



3. Bipartite Matching

Input. Bipartite graph.

Goal. Find maximum cardinality matching.



Why do we care?

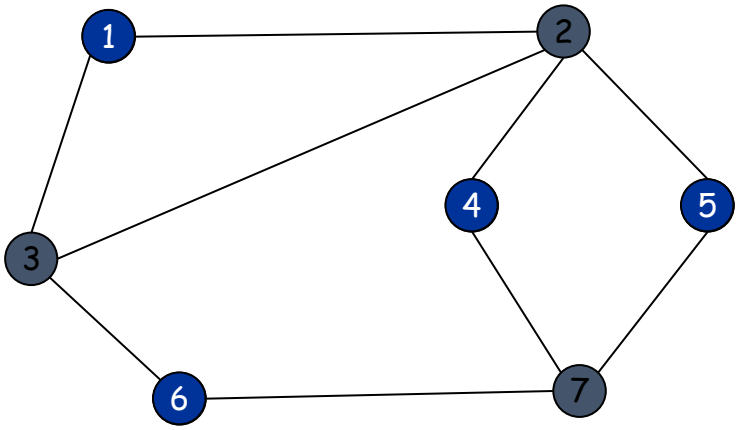
- *There are M job applicants and N jobs.*
- *Each applicant has a subset of jobs that he/she is interested in.*
- *Each job opening can only accept one applicant and a job applicant can be appointed for only one job.*
- *Find an assignment of jobs to applicants in such that as many applicants as possible get jobs.*

4. Independent Set

Input. Graph.

Goal. Find maximum cardinality independent set.

subset of nodes such that no two joined by an edge

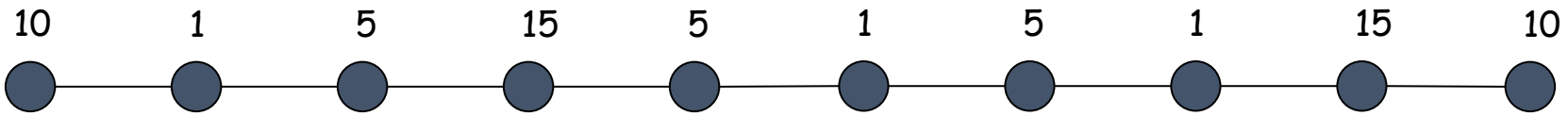


5. Competitive Facility Location

Input. Graph with weight on each node.

Game. Two competing players alternate in selecting nodes. Not allowed to select a node if any of its neighbors have been selected.

Goal. Select a maximum weight subset of nodes.



Second player can guarantee 20, but not 25.

Five Representative Problems

Variations on a theme: independent set.

Interval scheduling: n log n greedy algorithm.

Weighted interval scheduling: n log n dynamic programming algorithm.

Bipartite matching: n^k max-flow based algorithm.

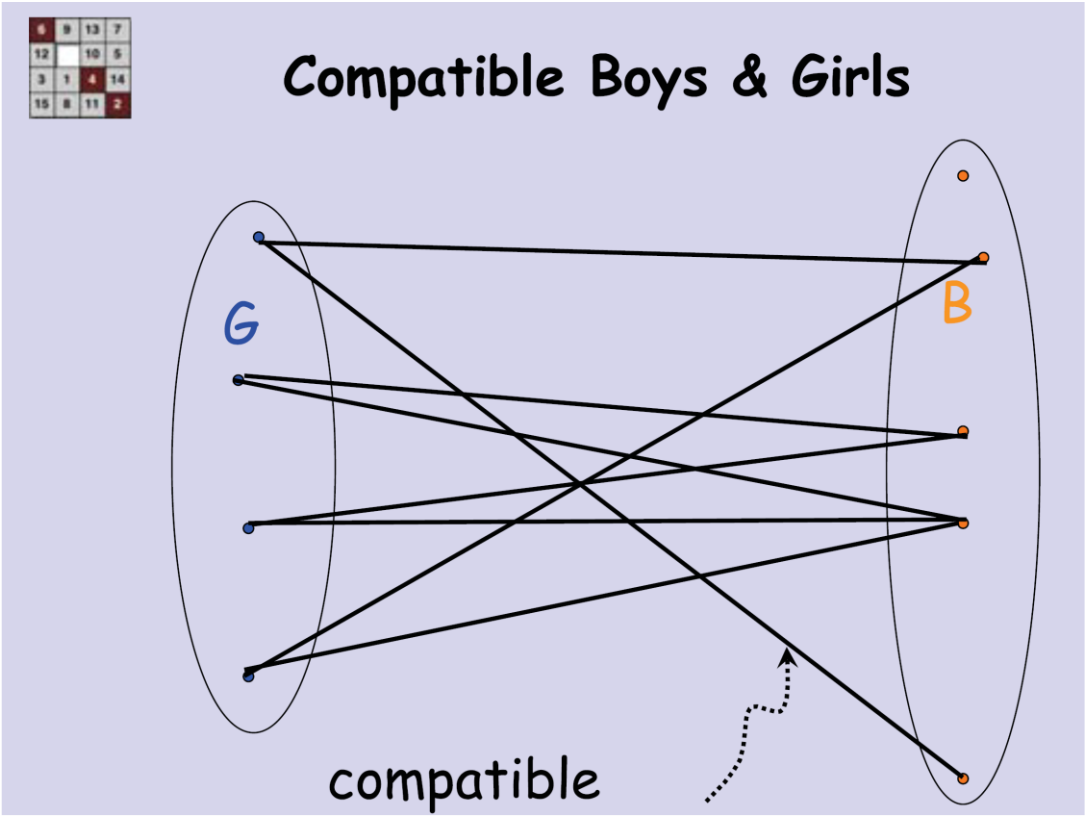
Independent set: NP-complete.

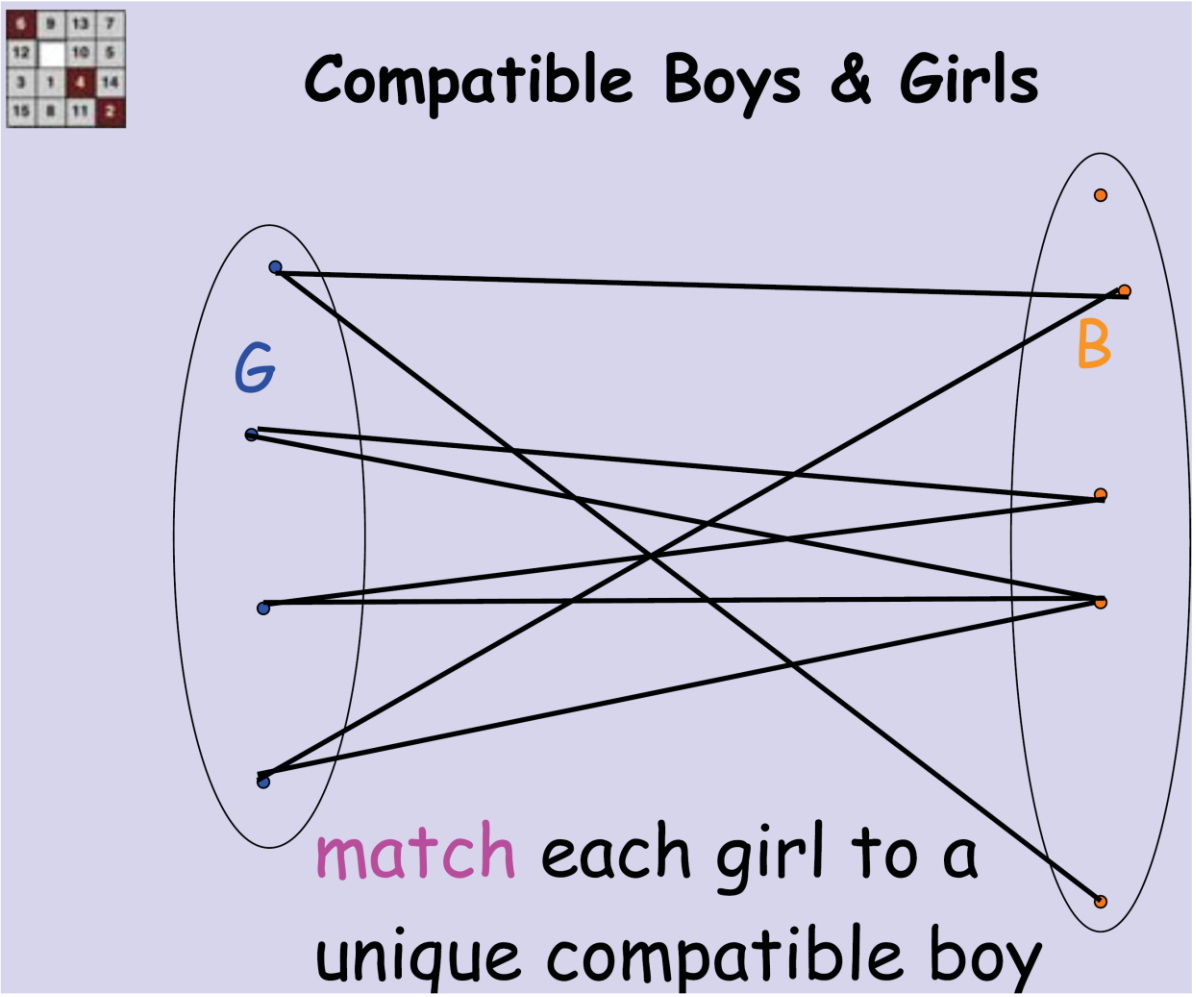
Competitive facility location: PSPACE-complete.

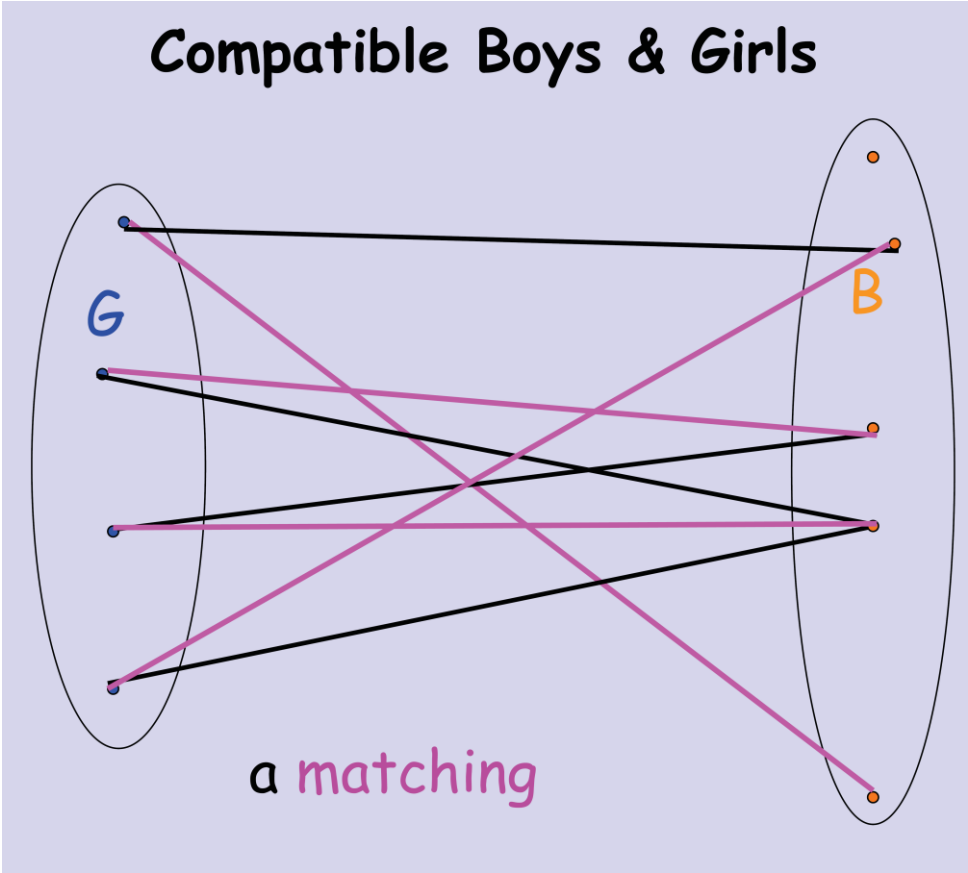
PSPACE: The set of all problems that can be solved by an algorithm with polynomial space complexity (Chapter 9).
P $\subseteq$ PSPACE (in poly time an algorithm can only consume poly space.)
NP $\subseteq$ PSPACE (There is an algorithm that can solve 3-SAT using only a polynomial amount of space.

Bipartite Matching

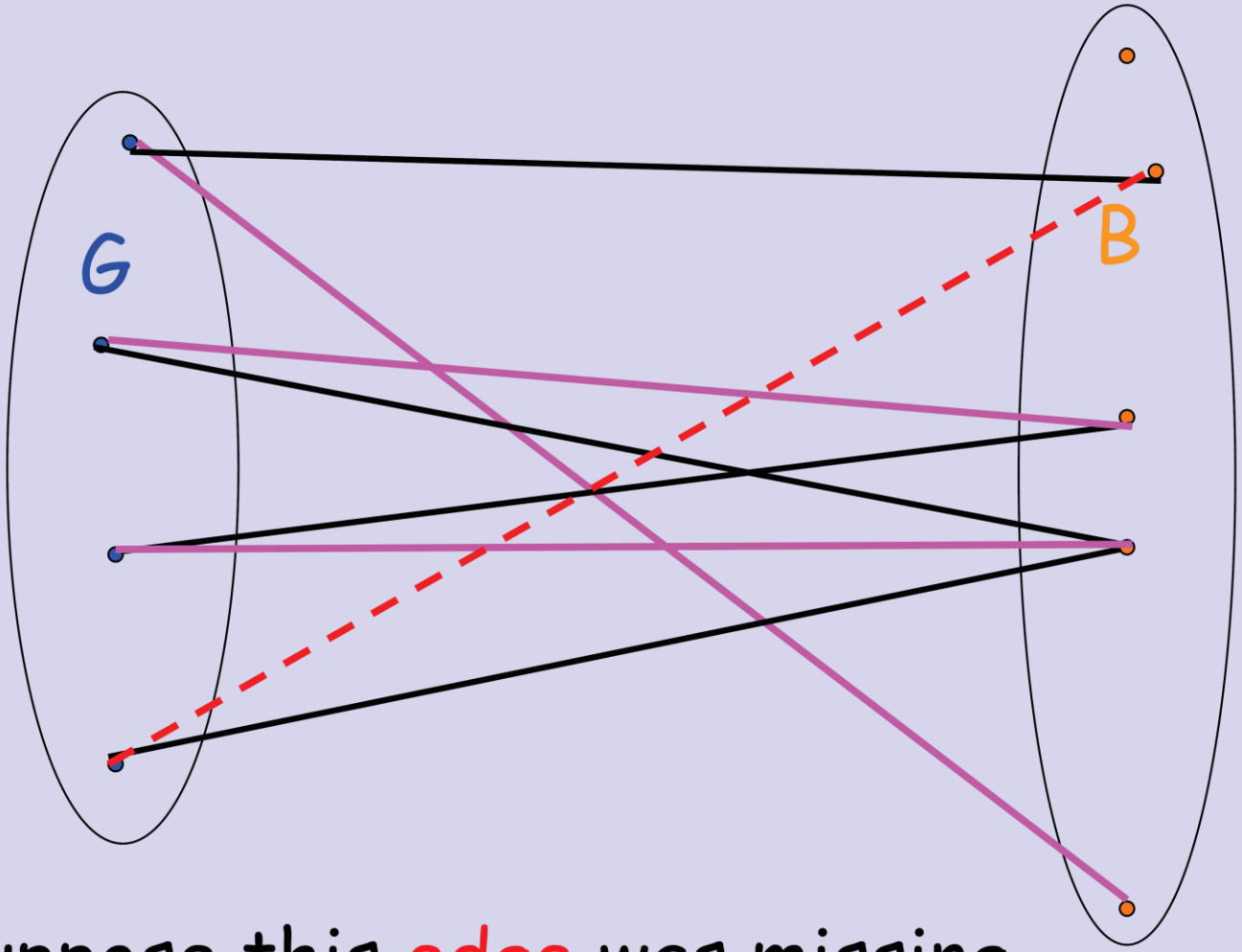
A matching in a Bipartite Graph is a set of the edges chosen in such a way that no two edges share an endpoint.



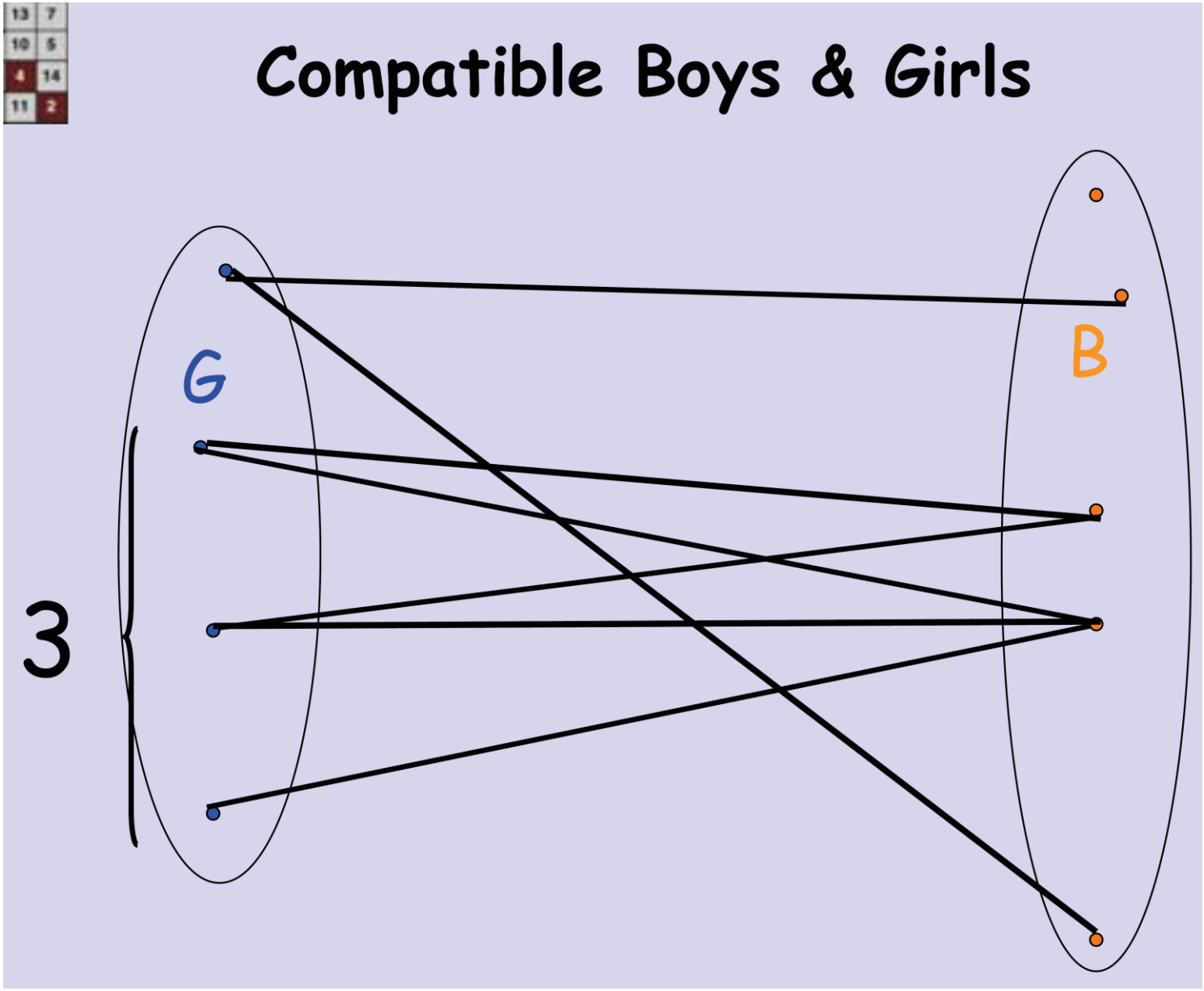

Compatible Boys & Girls
G
B
match each girl to a
unique compatible boy

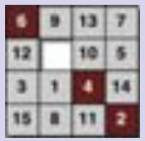

Compatible Boys & Girls
G
B
a matching

Compatible Boys & Girls

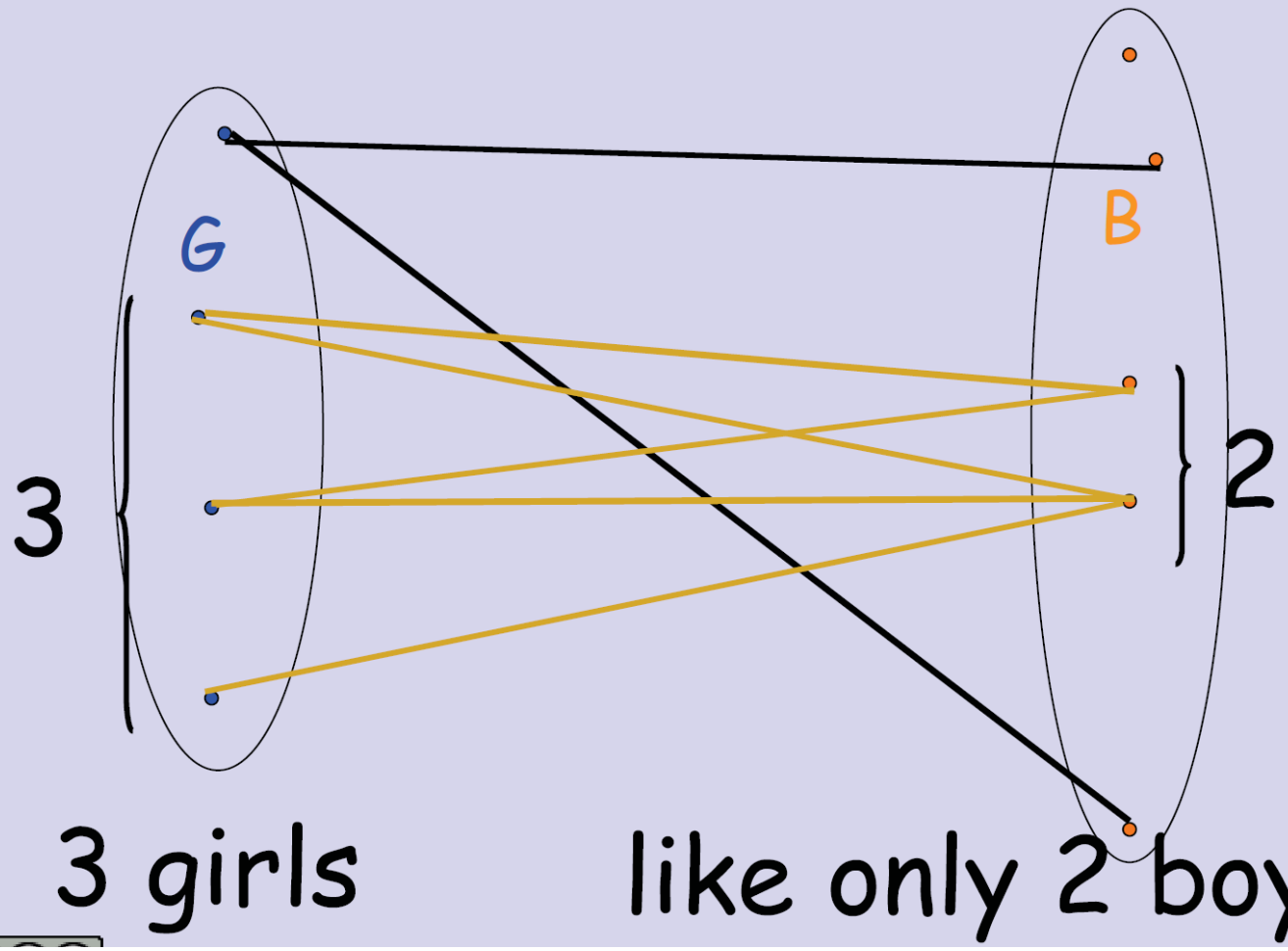


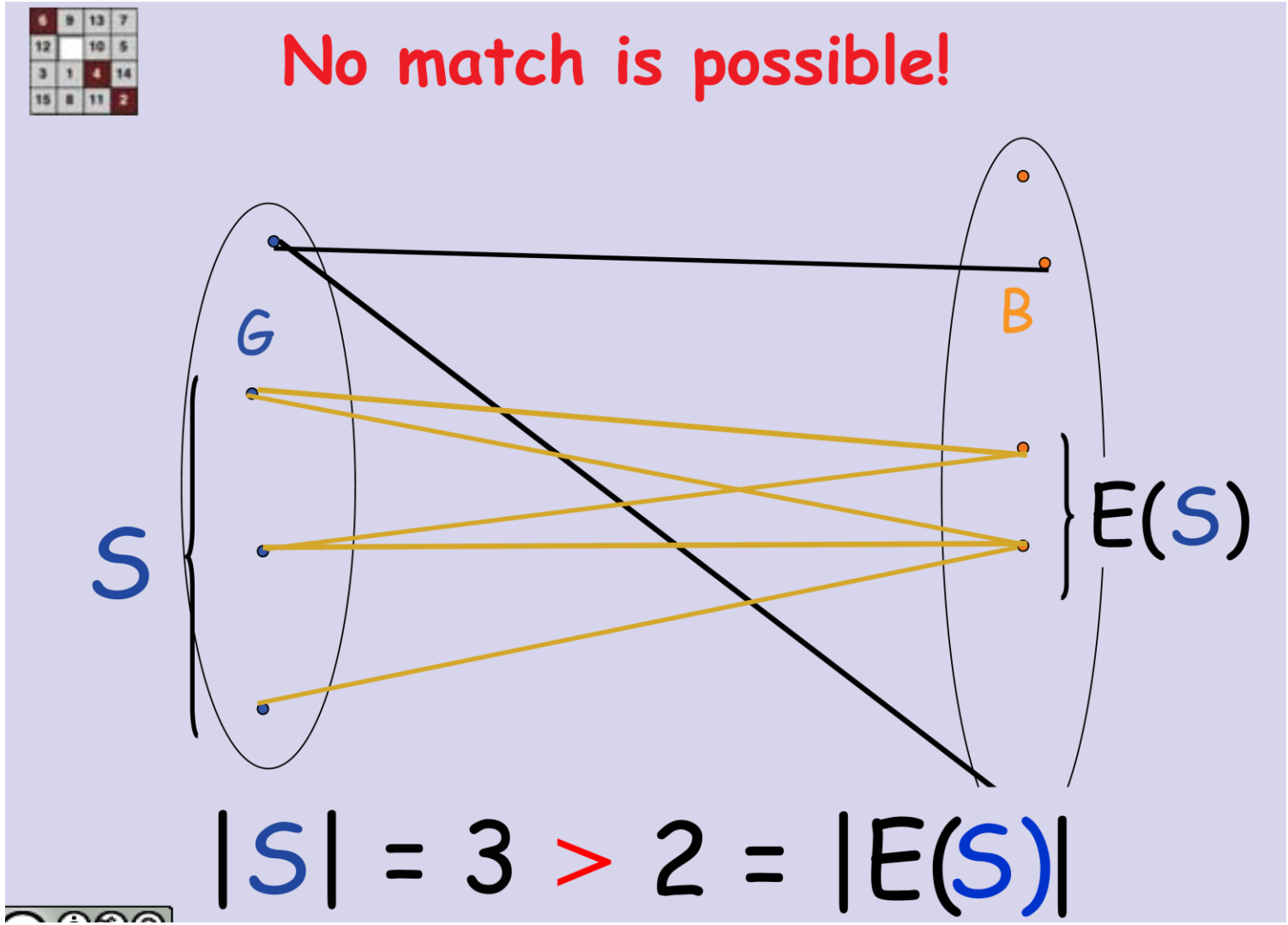
suppose this edge was missing

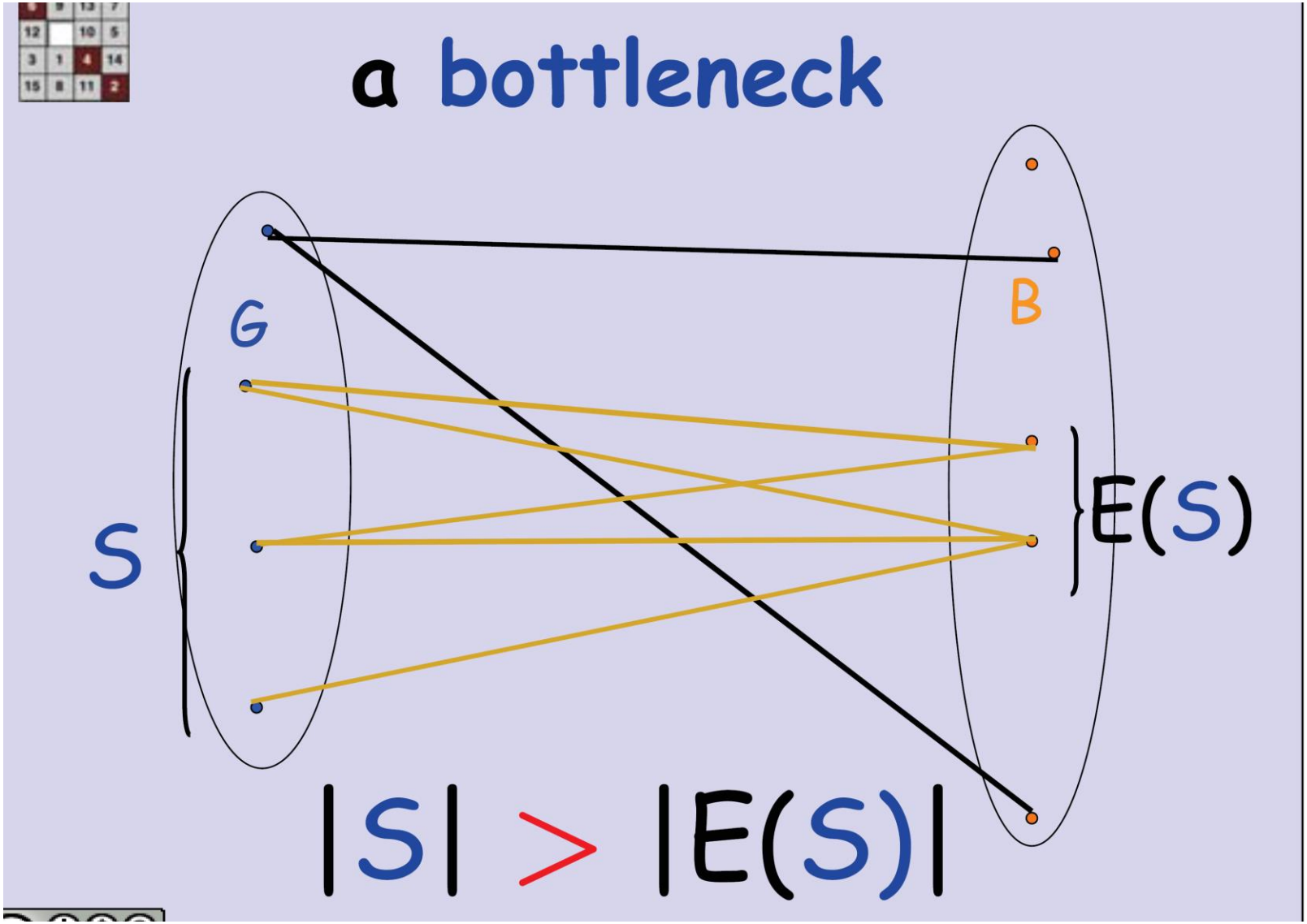

13 7
10 5
4 14
11 2
Compatible Boys & Girls
G
B
3



Not enough boys for these girls!




No match is possible!
G
B
S
E(S)
|S| = 3 > 2 = |E(S)|


a bottleneck
G
B
S
E(S)
|S| > |E(S)|

Bottleneck Lemma

Bottleneck: a set S of girls without enough boys.

E(S) ::= boys adjacent to at least one girl in S.

$$|S| > |E(S)|$$

Hall's Theorem

Conversely, if there are no bottlenecks, then there is a match.

If there is a bottleneck, then no match is possible obviously

Next Lecture

- Asymptotically Tight Bounds
- Big-O Notation
- Big Theta and Omega
- A Survey of runtimes

Week	Date	Topics
1	22 Feb	Introduction. Some representative problems
2	1 March	Stable Matching
3	8 March	Basics of algorithm analysis.
4	15 March	Graphs (Project 1 announced)
5	22 March	Greedy algorithms I
6	29 March	Greedy algorithms II (Project 2 announced)
7	5 April	Divide and conquer
8	12 April	Midterm
9	19 April	Dynamic Programming I
10	26 April	Dynamic Programming II (Project 3 announced)
11	3 May	BREAK
12	10 May	Network Flow-I
13	17 May	Network Flow II
14	24 May	NP and computational intractability I
15	31 May	NP and computational intractability II