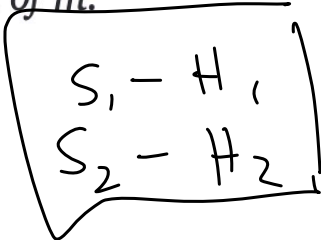


True or false? In every instance of the Stable Matching Problem, there is a stable matching containing a pair (m, w) such that m is ranked first on the preference list of w and w is ranked first on the preference list of m .

False



is this matching stable?
is there a pair:

- ① not in matching
- ② want to switch



is it stable?

there is no pair s.t. m is w 's #1 and w is m 's #1

$$S_1 = \{x, y, z\}$$

$$S_2 = \{a, b, c\}$$

$$(x, a), (y, c), (z, b)$$

Quiz 1 — end at 10:10

problem 2 should say:

Suppose we have $2n$ double rooms and $2n$ people to assign to them...

This is called the stable roommate problem

Here is the full text:

2. There are many other settings in which we can ask questions related to some type of "stability" principle. Here's one, involving assigning roommates.

Suppose we have n double rooms and $2n$ people to assign to them. Every person has a preference list indicating their ordered preference of rooming with every other person. For example, here is a preference list for four people who need to be assigned to two rooms:

Allie: Brenda, Cat, Diane

Brenda: Cat, Allie, Diane

Cat: Allie, Brenda, Diane

Diane: Allie, Brenda, Cat

In the *stable roommate problem*, we would like to find an assignment of roommates that is stable; that is, an assignment of roommates where no pair of people who are *not* assigned to be roommates both prefer one another over their currently assigned roommate.

Closing out stable matching basics

Stable roommate

A: B, C, D

B: (C), A, D

C: A, (B), D

D: A, B, C

A-B, C-D

B-C is
not in
this
matching

A: B, C, D

B: C, A, D

C: A, B, D

D: A, B, C

A-C, B-D

A-B

A: B, (C), D

B: C, A, D

C: (A), B, D

D: A, B, C

A-D, B-C

A-C

How do we know Hosp-Student problem always has a stable matching? ₂

How do we know Hosp-Student always has a stable matching?

Proved that Gale-Shapley alg:

① terminates

② returns a stable matching

Overall goals for CSCI 332

- * Take real-world problems and precisely define computational problems
- * Design correct and efficient algorithms for these problems
- * Last week, we saw a proof that Gale-Shapley returns a stable matching for any input.

Correctness

- * This week, we focus on defining what it means for algorithms to be **efficient**

big picture

refine



A definition of efficiency?

An algorithm is efficient if, when implemented, it runs quickly on real input instances.

↓
depends on
software
hardware

bugs

↓
what
does this
mean?

↓
- infinite?
- bad
assumptions

input size?

A definition of efficiency?

An alg. is efficient if it achieves qualitatively better performance than brute force search.

Brute force

Brute force. For many nontrivial problems, there is a natural brute-force search algorithm that checks every possible solution.

- usually 2^n or worse
- $n!$ for stable match



A brute force sorting algorithm

Input: an array A of integers *n integers*

Output: a sorted version of A

For each ordering A' of A :

→ n!

If A' is sorted:

Return A

What is the runtime of this algorithm?

1. n
2. $n \log n$
3. n^2
4. $\log n \cdot 2^n$
5. $n \cdot n!$

Polynomial running time

an

Desirable scaling property. When the input size doubles, the algorithm should slow down by at most some multiplicative constant factor C .

Alg 1

	n	time	n	time	C
Test 1	50	4 sec	100	8 sec	2
Test 2	70	5 sec	140	14 sec	3
Test 3	90	7 sec	180	24 sec	3.5

2^n

Alg 2

	n	time	n	time	C
Test 1	40	5 sec	80	20 sec	4
Test 2	10	0.3 sec	20	1.2 sec	4
Test 3	75	17.5 sec	150	70 sec	4

n^2

Why it matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Polynomial running time

We say that an algorithm is **efficient** if it has a polynomial running time.

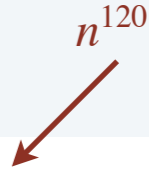
Theory. Definition is insensitive to model of computation.

Practice. It really works!

- The poly-time algorithms that people develop have both small constants and small exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions.

- Some exponential algorithms are fast in practice because worst-case inputs don't occur.
- “Galactic” poly-time algorithms.



Map graphs in polynomial time

Mikkel Thorup*
Department of Computer Science, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen East, Denmark
mthorup@diku.dk

Abstract

Chen, Grigni, and Papadimitriou (WADS'97 and STOC'98) have introduced a modified notion of planarity, where two faces are considered adjacent if they share at least one point. The corresponding abstract graphs are called map graphs. Chen et.al. raised the question of whether map graphs can be recognized in polynomial time. They showed that the decision problem is in NP and presented a polynomial time algorithm for the special case where we allow at most 4 faces to intersect in any point — if only 3 are allowed to intersect in a point, we get the usual planar graphs.

Chen et.al. conjectured that map graphs can be recognized in polynomial time, and in this paper, their conjecture is settled affirmatively.