Introduction

Data Structures and Algorithms

Andrei Bulatov

Course Info

- **Instructor: Andrei Bulatov**
  - Email: abulatov@cs.sfu.ca
  - Room: TASC 8013
  - Office hours (tentative):
    Thursday 15:30 – 17:00

- **Teaching Assistants:**
  - Mohammad Tayebi, email: mohammad.tayebi@gmail.com
  - Pariya Raoufi, email: pariya.raoufi@sfu.ca

- **Course webpage**
  - http://www.cs.sfu.ca/CC/307/abulatov

Course Info

- **Course objective:**

  To introduce concepts and problem-solving techniques that are used in the design and analysis of efficient algorithms. This is done by studying various algorithms and a variety of data structures.

- **Syllabus:**
  - Preliminaries: asymptotic notation, models of computation, basic probability theory.
  - Sorting and Order Statistics
  - Simple Data Structures: lists, stacks, queues
  - Dictionaries: hashings, height-balanced trees, B-trees
  - Priority Queues: Heaps, Binomial Heaps, Fibonacci Heaps
  - Dynamic programming, greedy algorithms
  - Graph Algorithms: BFS, DFS, shortest-paths, etc.

Course Info

- **Textbook:**
  - Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms,* McGraw Hill, MIT Press.

- It is impossible to finish studying all the contents of the textbook in one semester. The contents not covered in lectures/slides are not required, unless explicitly indicated as required.
- The content and order of topics, as presented in the class, do not one-to-one correspond to any part of the book. Use of Subject Index and Recommended Text is advised.

Course Info

- **References:**
  - Jon Kleinberg and Eva Tardos, *Algorithm Design*, Addison Wesley, 2005.
  - D. E. Knuth, *The Art of Computer Programming. Vol. 1,2,3,4,* Addison-Wesley
  - R. L. Graham; D. E. Knuth; and O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1994

Course Info

- **Grading:**
  - 8 Assignments (8 × 3%)
  - 1 Midterm 26%
  - 1 Final Exam 50%

Prerequisites

- Not much of specific knowledge
- Some general knowledge is needed, as there will be examples
- Basic math erudition
- Some experience in programming is very helpful

Stable Matching Problem

Problem (first try)

There are n men and m women. Every men has a ranked list of some women, and every woman has a ranked list of men. Is it possible to match men and women so that there won't be any difficulties later?

What is a difficulty?

Some men m and m' are matched to women w and w', respectively. However, m prefers w' to w, and w' prefers m to m'

Stable Matching Problem: Examples

Example 1

Two men m, m’, and two women w, w’.

m prefers w to w’	w prefers m to m’
m’ prefers w to w’	w’ prefers m to m’

Only one stable matching: (m,w), (m’,w’)

Example 2

Two men m, m’, and two women w, w’.

m prefers w to w’	w prefers m’ to m
m’ prefers w’ to w	w’ prefers m to m’

Two stable matchings: (m,w), (m’,w’); and (m’,w), (m,w’)

Stable Matching Problem: Mathematical Formalism

Consider a set of men $M = \{m_1, \ldots, m_n\}$ and a set of women $W = \{w_1, \ldots, w_n\}$

Let $M \times W$ denote the set of ordered pairs (m,w) where $m \in M$, $w \in W$

A matching S is a subset of $M \times W$ such that each member of M and each member of W appears in at most one pair from S

A perfect matching S' is a matching such that each member of M and W appears in exactly one pair from S'

Each man ranks all the women. We say that m prefers w to w' if m ranks w higher than w'. Ordered ranking of m will be called his preference list

Women rank men analogously

No ties!!

Stable Matching Problem: Formalism (cntd)

A pair (m,w’) is an instability with respect to a matching S if (m,w’) does not belong to S, but both m and w’ prefers each other to their current matches.

A matching is stable if it is (i) perfect, and (ii) has no instabilities

Problem (bare-bone, formal)

There are n men and n women with their preference lists.

(a) Does there exist a stable matching?

(b) If a stable matching exists, how can we find it?

Stable Matching Problem: Gale-Shapley Algorithm

Input: sets M and W of men and women with preference lists

Output: a stable matching

```
initially all m∈M and w∈W are free
while there is a free man do
  let w be the highest ranked woman for m to whom he
   hasn't yet proposed
  if w is free then (m,w) become engaged
  else if w is currently engaged to m'
     if w prefers m' to m then m remains free
     else w prefers m to m'
         (m,w) become engaged
         m' becomes free
     endif
  endif
endwhile
Return the set S of engaged pairs
```

Stable Matching Problem: Analysis, Running Time

Lemma 1

w remains engaged from the point at which she receives her first proposal; and the sequence of matches gets better and better

Lemma 2

The sequence of women to whom m proposes gets worse and worse

Lemma 3

The G-S algorithm terminates after at most n^2 iterations of the `while` loop.

Stable Matching Problem: Running Time

Proof of Lemma 3

The main idea is to find a measure of progress, so that every step of the algorithm increases this measure and brings it closer to termination

Our measure is the set P(t) (for each point of time t) that contains all pairs (m,w) such that m has proposed to w by the time t.

Since someone proposes to someone at each step, P(t+1) is strictly greater than P(t).

Therefore the number of iterations does not exceed the maximal size of P(t), that is n^2

QED

Stable Matching Problem: Analysis

Lemma 4

If m is free then there is a woman to whom he hasn't yet proposed

Proof

Suppose at some point m is free an he has proposed to all women.

By Lemma 1 every woman is engaged at this point. But there are only n men, so some woman should be engaged to m

A contradiction

QED

Stable Matching Problem: Analysis (cntd)

Lemma 5

The set S returned at termination is a perfect matching

Proof

The set of engaged pairs always forms a matching.

Suppose that the algorithm terminates with a free man m.

At termination m has proposed to all the women.

A contradiction with Lemma 4.

QED

Stable Matching Problem: Soundness

Lemma 6

The set S returned at termination is a stable matching

Proof

By Lemma 5, S is a perfect matching

Suppose there is an instability w.r.t. S.

There are 2 pairs (m,w) and (m',w') in S such that

m prefers w' to w, and

w' prefers m to m'

Clearly, the last proposal of m was w. Did m propose to w'?

If not, then m prefers w to w', a contradiction

If yes, then he was rejected in favor of m'', who was rejected in favor of m. This means w' prefers m' to m, a contradiction.

QED

More Questions

If there is more than one stable matching, which one is returned by G.-S. algorithm?

Example 2

Two men m, m’, and two women w, w’.

m prefers w to w’	w prefers m’ to m
m’ prefers w’ to w	w’ prefers m to m’

Two stable matchings: (m,w), (m’,w’); and (m’,w), (m,w’)

In a sense, G.-S. is biased toward the proposing gender

Even More Questions

We do not specify how we choose a free man to propose. In this sense the algorithm is non-deterministic. Given the same input there are many possible executions.

Do different executions produce different results?

If yes which one is `better'?

One way to answer negatively is to describe the matching produced by *any* execution

w is a valid partner of m if there is a stable matching containing (m,w)

w is the best valid partner of m if w is a valid partner of m and no woman – valid partner – ranked higher than w

The best valid partner of m will be denoted $best(m)$

Even More Questions (cntd)

Let S^* denote the set of pairs $\{ (m,best(m)) : m \in M \}$.

Lemma 7

Every execution of the G.-S. algorithm returns S^*

Proof

Suppose some execution returns a stable matching in which some man is paired with a woman who is not the best valid partner.

There is a man m first rejected by his valid partner $w = best(m)$

Therefore w got engaged with m' whom she prefers to m

Even More Questions (cntd)

Proof (cntd)

There is a stable matching S’ in which m is paired with best(m)

Who is m’ paired with? Suppose it is $w' \neq w$.

Since the rejection of m by w is the first rejection by a valid partner during the execution, m’ has not been rejected by any his valid partner at the point he proposed to w

Therefore m’ prefers w to w’

On the other hand w prefers m’ to m

Thus the pair (m’,w) does not belong to S’ and is an instability w.r.t. S’

QED

Even More Questions (cntd)

In a similar way we can define the worst valid partner

Lemma 7

Every execution of the G.-S. algorithm returns the stable matching in which every woman is paired with her worst valid partner.

Efficient Algorithms

What algorithm we call efficient?

Intuition:

One that works reasonable time on reasonable (practical) instances

Not quite good:

works where?

what is reasonable time?

what are reasonable instances?

does it matter how the algorithm scales?

Efficient Algorithms (cntd)

Need to make it more precise:

- `platform-independent’
- `instance-independent’
- clear indication of scaling

Instance size:

A parameter showing how big an instance is

depends on the problem

For the Stable Matching Problem it may be the number n of men and women

Then we can evaluate the running time mathematically $N = n^2$

Worst Case Running Time

We will analyze the worst case running time:

bound on the largest possible running time that achieved on instances on a given size n

So, it will be a function of n

Worst case vs. average case

Brute Force

For many problems there is a very simple algorithm

For example, for the Stable Matching Problem we could try all perfect matchings

Difficulty: there are too many of them, $n!$

Such an algorithm is called a brute force algorithm:

enumerate all possible configurations and choose the right one

An efficient algorithm should outperform the brute force algorithm

- substantially
- provably / analytically

Polynomial Time

Good criterion of scaling: If the instance size is doubled the running time increases by a constant factor

Natural example: polynomials

Let the running time is $f(n) = 3n^d + 2n^2 + n$

When doubling the instance size

$$f(2n) = 3(2n)^d + 2(2n)^2 + 2n \le 2^d (3n^d + 2n^2 + n)$$

An algorithm has polynomial running time, or is a polynomial time algorithm if its running time is bounded from above by a polynomial

Is it good?

Polynomial Time (cntd)

Contras:

- there are bad polynomial time algorithms
- there are good non-poly time algorithms
- it does not capture the `practical’ complexity of algorithms

Pros:

- usually if there is a poly time algorithm, there is a good one
- it captures something

Running time	Size n					
	10	20	30	40	50	60
n	.00001 seconds	.00002 seconds	.00003 seconds	.00004 seconds	.00005 seconds	.00006 seconds
n^2	.0001 seconds	.0004 seconds	.0009 seconds	.0016 seconds	.0025 seconds	.0036 seconds
n^3	.001 seconds	.008 seconds	.027 seconds	.064 seconds	.125 seconds	.216 seconds
n^5	.1 seconds	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 seconds	1.0 seconds	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	.059 seconds	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Asymptotics

We don’t want to compute the exact running time

Why?

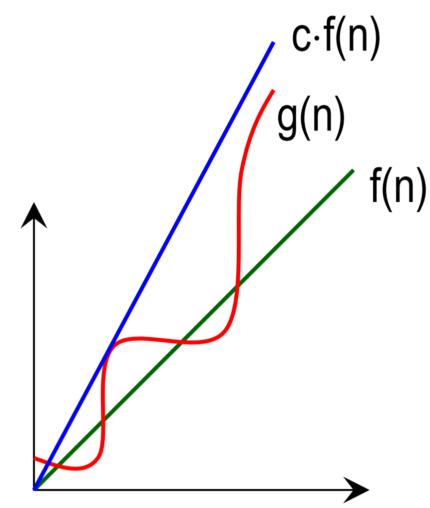
- Do we care if the running time is $2.53n^2 + 3.42n$ or $2.55n^2 + 3.39n$?
- We will mostly represent algorithms by pseudocode.
  Implementation details can change running time by some constant
- We are interested in more conceptual differences between algorithms, and will be happy with a rough classification

Then $2.53n^2 + 3.42n$ is more or less similar to n^2

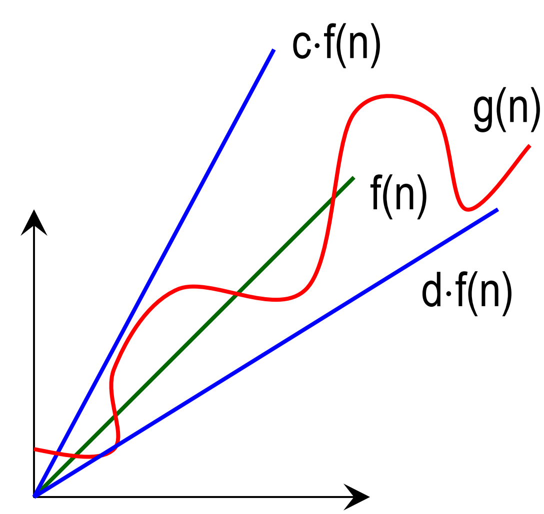
Asymptotic Notation

- For two functions f,g: $\mathbb{N} \to \mathbb{R}$

- g is in O(f) if there is c such that starting from some k: $g(n) \le c \cdot f(n)$

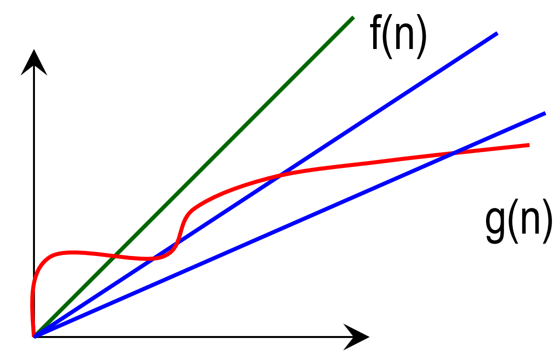


- g is in Θ(f) if there are c,d > 0 such that starting from some k: $d \cdot f(n) \le g(n) \le c \cdot f(n)$



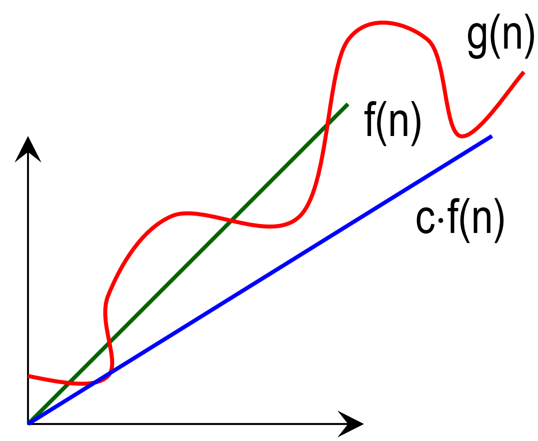
Asymptotic Notation

- g is in $o(f)$ if for any c starting from some $k(c)$: $g(n) < c \cdot f(n)$



- g is in $\Omega(f)$ if there is c such that starting from some k:

  $g(n) \geq c \cdot f(n)$



Homework

Consider more general Stable Matching Problem:

- different number of men and women
- incomplete preference lists

Change the G.-S. algorithm to deal with these complications

What can be achieved using a G.-S.-like algorithm?