CS481/CS583: Bioinformatics Algorithms

Can Alkan
EA509
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs481/

CS481/CS583

- Class hours:
  - Tue 13:30-15:20 - Fri 09:30-10:20
- Classroom: EB104
- Office hour: by appointment
  - Zoom or Meet only
  - My public calendar: http://cs.bilkent.edu.tr/~calkan/calendar.html
- TA: Ricardo Román Brenes: ricardo@bilkent.edu.tr
- Grading:
  - 1 midterm: 25% + 1 final exam: 35%
  - Homeworks (programming): 30% (n=5)
  - Quizzes: 10% (n=5, announced)
- Due to the YÖK (Higher Education Council) regulations, we are taking attendance and will report it to the Department at the end of the semester.
  - But attendance has no *direct* effect on grades

CS481/CS583: Resources

- All slides are available on the web page
  - http://cs.bilkent.edu.tr/~calkan/teaching/cs481/
  - Google Slides links -- read only
- Recommended Material
  - Genome-Scale Algorithm Design, Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, Alexandru I. Tomescu, 2015, Cambridge University Press
  - Other recommended books are listed on the web page
  - Problem sets on Rosalind: http://rosalind.info/problems/locations/
  - Bioinformatics Algorithms online book & videos:
    - https://www.bioinformaticsalgorithms.org/

CS481/CS583

- Recommended Textbooks
  - Genome Scale Algorithm Design, Veli Makinen, et al., Cambridge University Press, 2015
  - An Introduction to Bioinformatics Algorithms (Computational Molecular Biology), Neil Jones and Pavel Pevzner, MIT Press, 2004
  - https://www.bioinformaticsalgorithms.org/
- Additional:
  - Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Dan Gusfield, Cambridge University Press
  - Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison, Cambridge University Press
  - Bioinformatics: The Machine Learning Approach, Second Edition, Pierre Baldi, Soren Brunak, MIT Press
  - ROSALIND problem sets: http://rosalind.info/problems/locations/

CS481/CS583

- This course is about **algorithms** in the field of bioinformatics:
  - What are the problems?
  - What algorithms are developed for what problem?
  - Algorithm design techniques
- This course is **not** about how to analyze biological data using available tools:
  - Recommended course: MBG 326: Introduction to Bioinformatics

CS481/CS583 and other courses

- Includes elements from:
  - CS201/202: data structures -- implicit prerequisite
  - CS473: algorithms, dynamic programming, greedy algorithms, branch-and-bound, etc.
  - CS476: complexity, context-free grammars, DFA/NFA
  - CS464: hidden Markov models (not covered in CS481, but related topic)

CS481/CS583: Assumptions

- You are assumed to know/understand
  - Computer science basics (CS101/102 or CS111/112)
    - CS201/202 would be better
    - CS473 would be even better
  - Data structures (trees, linked lists, queues, etc.)
  - Elementary algorithms (sorting, hashing, etc.)
  - Programming: C, C++ (preferred); Python, Java, Rust
    - Note: we will give bonus points for the “fastest” code in some homeworks
- You don’t *have to* be a “biology expert” and **we will not teach any biology** in this course: MBG 110 would be sufficient

Bioinformatics Algorithms

- Development of methods based on computer science for problems in biology and medicine
  - Sequence analysis (combinatorial and statistical/probabilistic methods)
  - Graph theory
  - Data mining
  - Database
  - Statistics
  - Image processing
  - Visualization
  - …..

CS 481/CS583

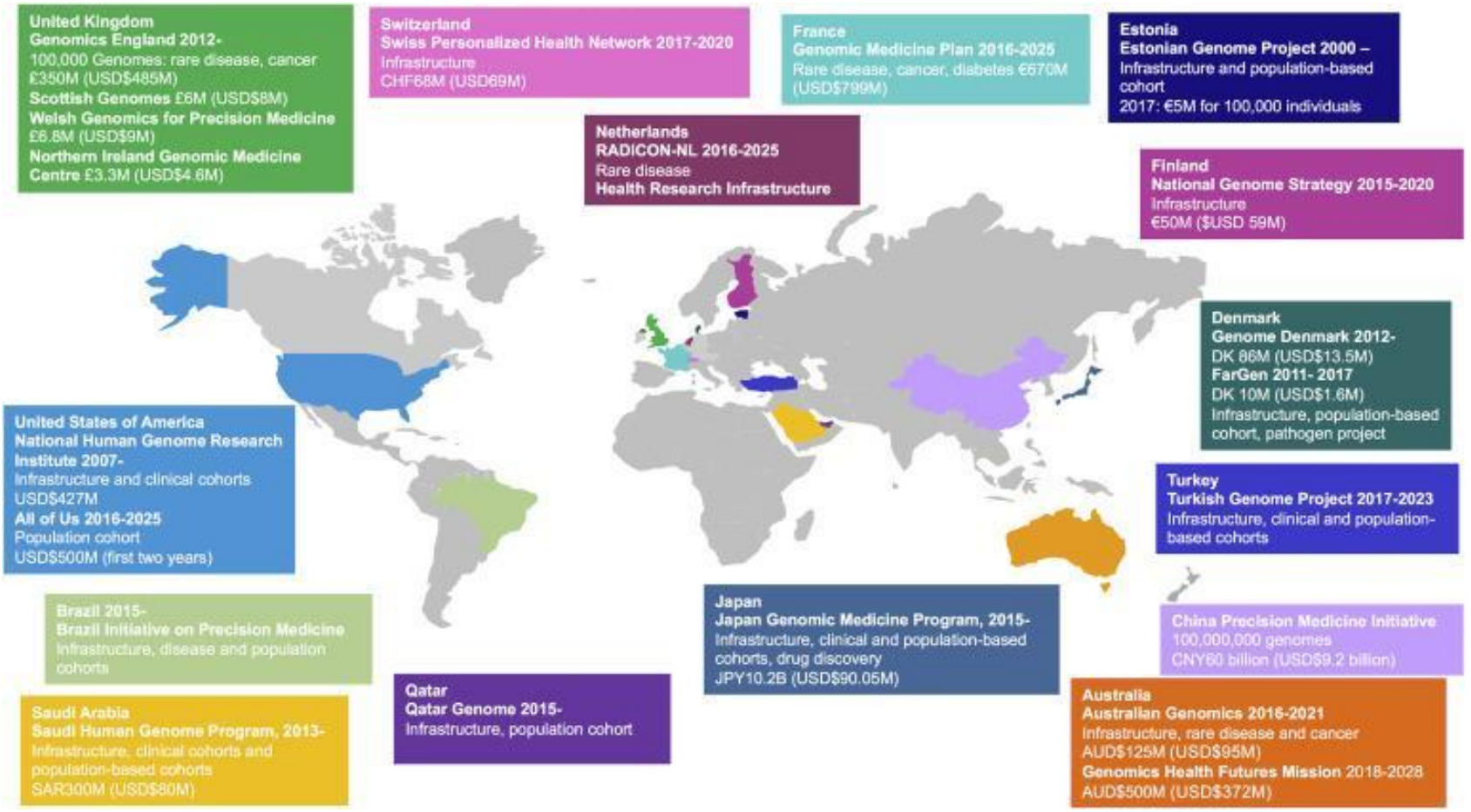
Bioinformatics: Applications

- Human disease
  - Personalized Medicine
- Genomics: Genome analysis, gene discovery, regulatory elements, etc.
- Population genomics
- Evolutionary biology
- Proteomics: analysis of proteins, protein pathways, interactions
- Transcriptomics: analysis of the transcriptome (RNA sequences)
- …

Why would you learn these algorithms?

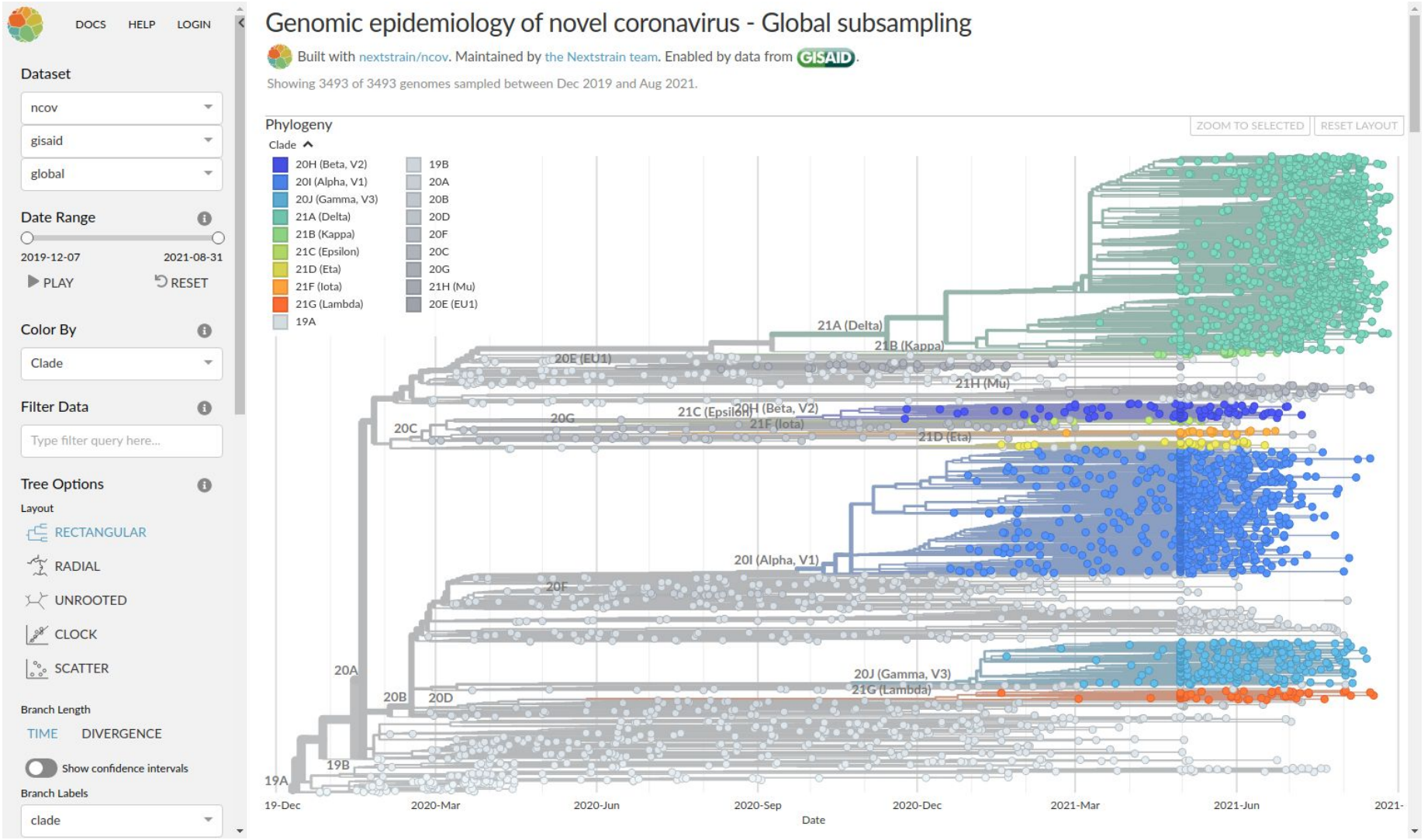
- Most developed for research within other fields that include string processing, clustering, text-pattern search, etc.
- Bioinformatics (non-academic) jobs on the rise:
  - Genomics England, Genome Asia, etc.: 100,000 genome projects
  - DNAnexus, SevenBridges, LifeBit: genome analysis on the cloud.

Genomics and healthcare



Stark et al., AJHG 2019

Tracking pathogens



http://www.nextstrain.org

(VERY) BRIEF INTRODUCTION TO COMPLEXITY

Tractable vs intractable

- Tractable problems: there exists a solution with O(f(n)) run time, where f(n) is *polynomial*
- *P* is the set of problems that are **known** to be *solvable* in polynomial time
- *NP* is the set of problems that are *verifiable* in polynomial time (or, solvable by a non-deterministic Turing Machine in polynomial time)
  - *NP: "non-deterministically polynomial"*

$$P \subset NP$$

NP-hard

- *NP-hard:* non-deterministically polynomial -hard
  - Set of problems that are "*at least as hard as the hardest problems in NP*"
  - There are no known polynomial time optimal solutions
  - There *may* be polynomial-time *approximate* solutions

NP-Complete

- A *decision problem* C is in NPC if :
  - C is in NP
  - Every problem in NP is **reducible** to C in polynomial time

That means: if you could solve any NPC problem in polynomial time, then you can solve all of them in polynomial time

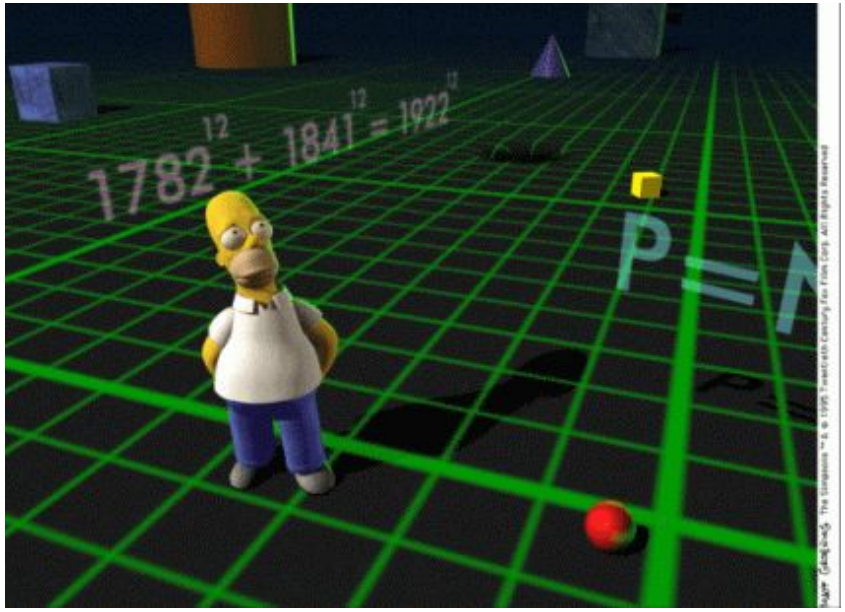
Decision problems: outputs “yes” or “no”

NP-intermediate

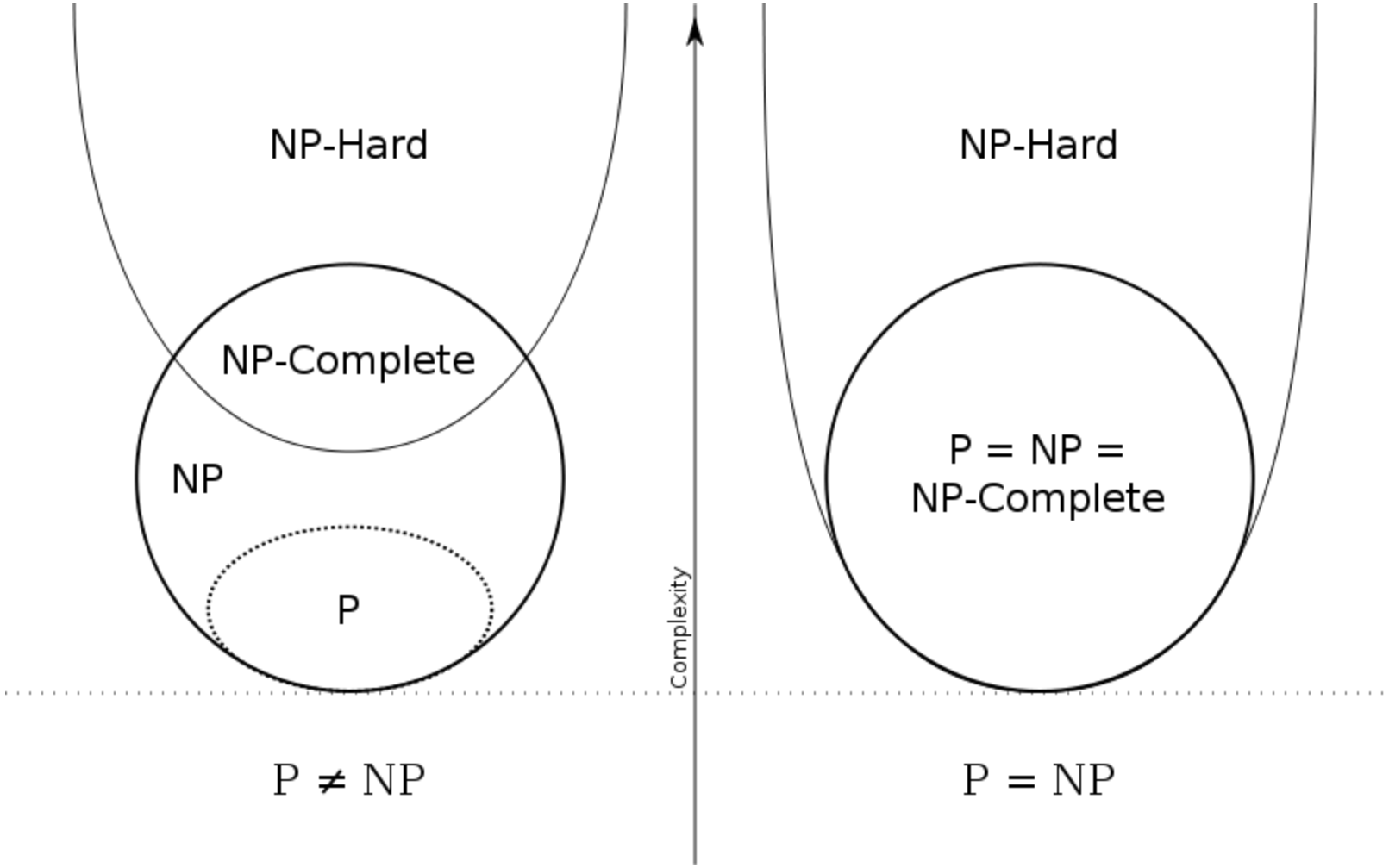
- Problems that are in NP; but not in either NPC or NP-hard (as far as we know)

P vs. NP

- We do not know whether P=NP or P≠NP
  - Principal unsolved problem in computer science
  - Most likely P≠NP



P vs. NP vs. NPC vs. NP-hard



Examples

- P:
  - Sorting numbers, searching numbers, pairwise sequence alignment, etc.
- NP-complete:
  - Subset-sum, traveling salesman, etc.
- NP-intermediate:
  - Factorization, graph isomorphism, etc.

Historical reference

- The notion of NP-Completeness: Stephen Cook and Leonid Levin independently in 1971
  - First NP-Complete problem to be identified: Boolean satisfiability problem (SAT)
    - Cook-Levin theorem
- More NPC problems: Richard Karp, 1972
  - “21 NPC Problems”
- Now there are thousands….

ALGORITHM DESIGN TECHNIQUES

Sample problem: Change

- Input: An amount of money M, in cents
- Output: Smallest number of coins that adds up to M
  - Quarters (25c): q
  - Dimes (10c): d
  - Nickels (5c): n
  - Pennies (1c): p
  - Or, in general, c_1, c_2, …, c_d (*d* possible denominations)

Algorithm design techniques

- **Exhaustive search / brute force**
  - Examine every possible alternative to find a solution

```
BRUTEFORCECHANGE(M, c, d)
1  smallestNumberOfCoins ← ∞
2  for each (i_1, ..., i_d) from (0, ..., 0) to (M/c_1, ..., M/c_d)
3      valueOfCoins ← Σ_{k=1}^{d} i_k c_k
4      if valueOfCoins = M
5          numberOfCoins ← Σ_{k=1}^{d} i_k
6          if numberOfCoins < smallestNumberOfCoins
7              smallestNumberOfCoins ← numberOfCoins
8              bestChange ← (i_1, i_2, ..., i_d)
9  return (bestChange)
```

Algorithm design techniques

- **Greedy algorithms:**
  - Choose the "most attractive" alternative at each iteration

BETTERCHANGE$(M, \mathbf{c}, d)$

```
1  r ← M
2  for k ← 1 to d
3        i_k ← r/c_k
4        r ← r − c_k · i_k
5  return (i_1, i_2, ..., i_d)
```

USCHANGE(M)

```
1  r ← M
2  q ← r/25
3  r ← r − 25 · q
4  d ← r/10
5  r ← r − 10 · d
6  n ← r/5
7  r ← r − 5 · n
8  p ← r
9  return (q, d, n, p)
```

Algorithm design techniques

- **Dynamic Programming:**
  - Break problems into subproblems; solve subproblems; merge solutions of subproblems to solve the real problem
  - Keep track of computations to avoid recomputing values that you already solved
    - *Dynamic programming table*
  - Essentially, recursive algorithms that remember previous solutions

DP example: Rocks game

- Two players
- Two piles of rocks with p_1 rocks in pile 1, and p_2 rocks in pile 2
- In turn, each player picks:
  - One rock from either pile 1 or pile 2; OR
  - One rock from pile 1 and one rock from pile2
- The player that picks the last rock wins

DP algorithm for Player 1

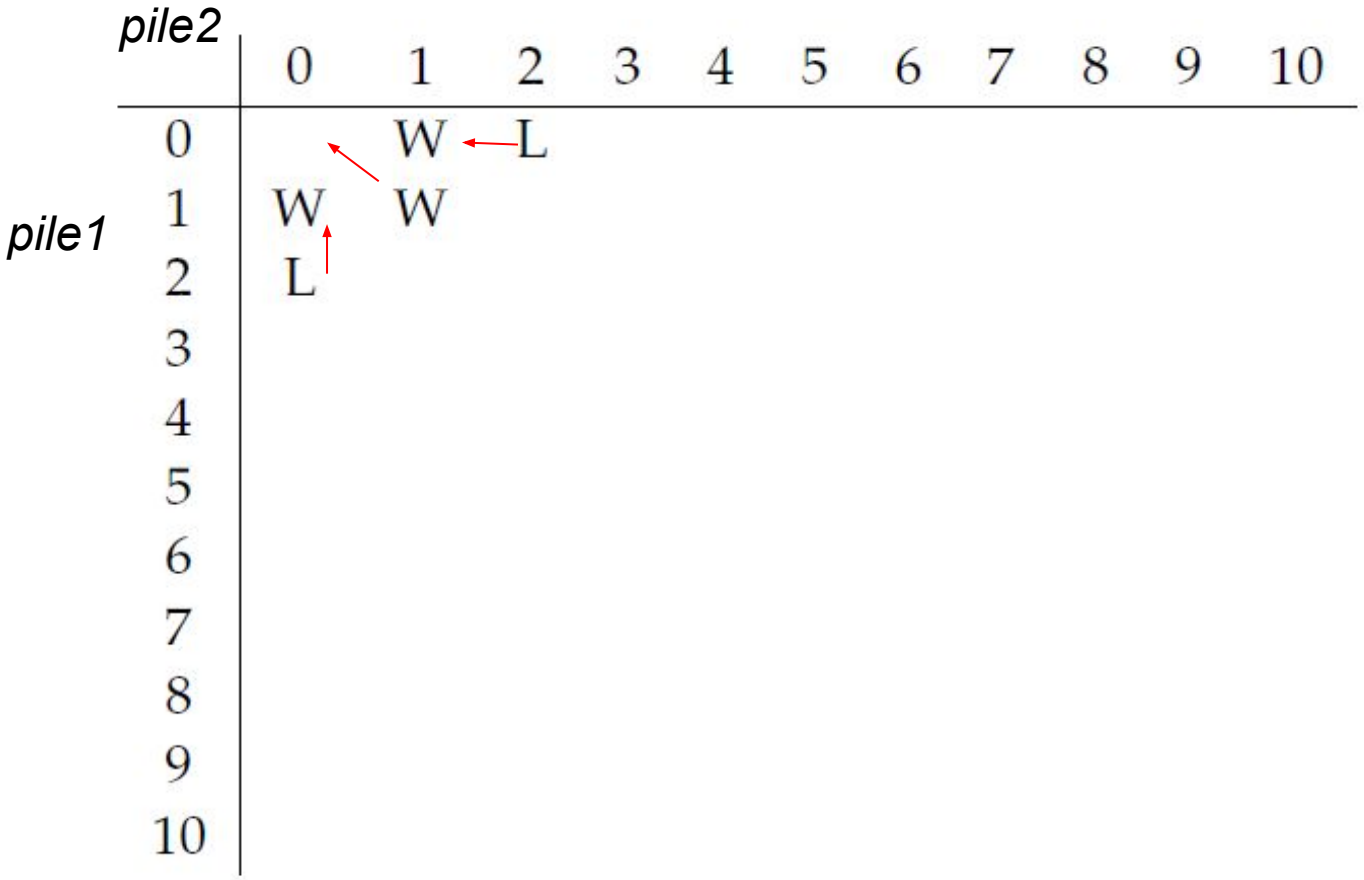
- Problem: $p_1 = p_2 = 10$
- Solve more general problem of $p_1 = n$ and $p_2 = m$
- It's hard to directly calculate for n=5 and m=6; we need to solve smaller problems

DP algorithm for Player 1

pile1 \ pile2	0	1	2	3	4	5	6	7	8	9	10
0		W									
1	W	W									
2											
3											
4											
5											
6											
7											
8											
9											
10											

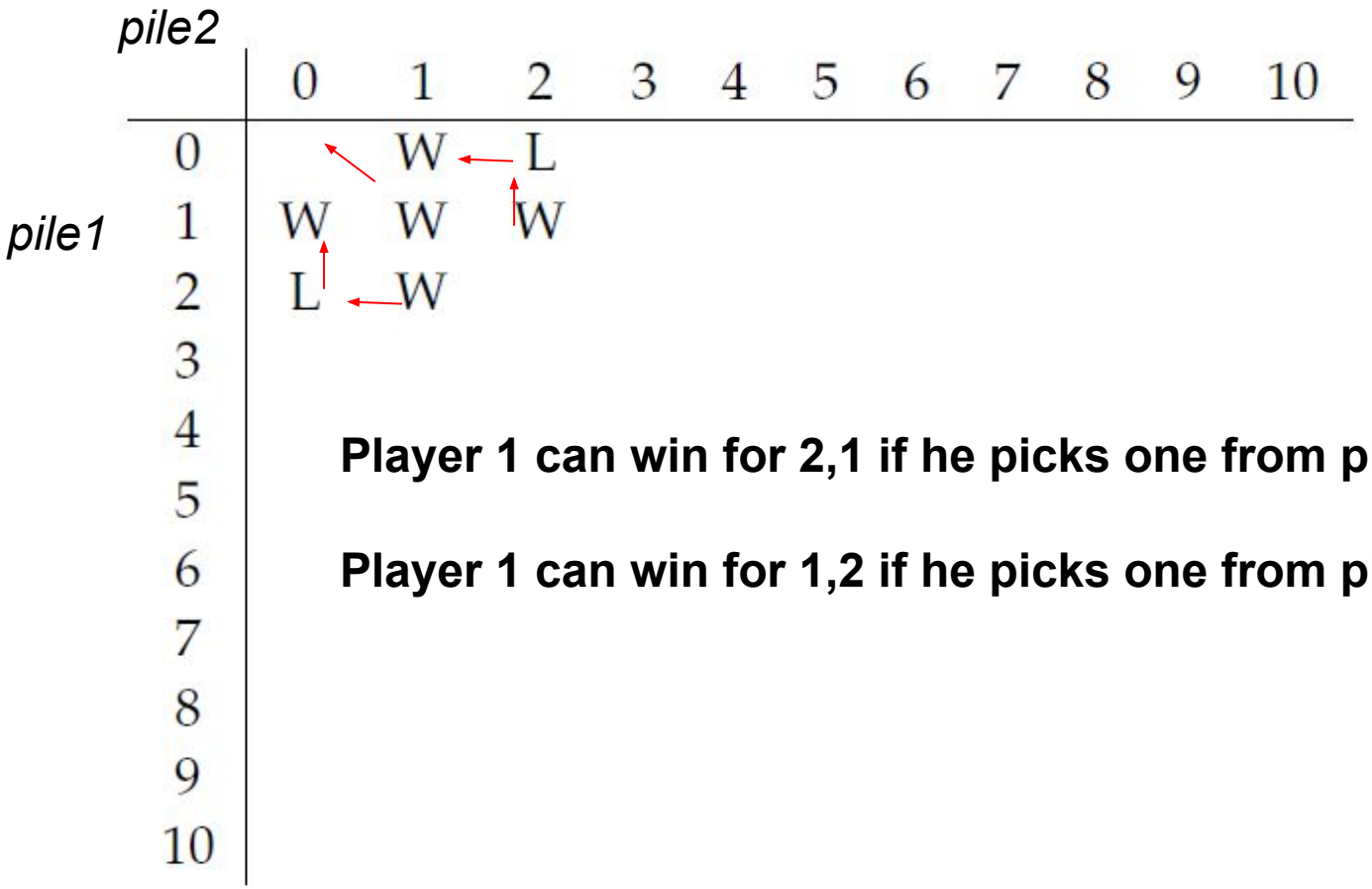
Initialize; obvious win for Player 1 for 1,0; 0,1 and 1,1

DP algorithm for Player 1



Player 1 cannot win for 2,0 and 0,2

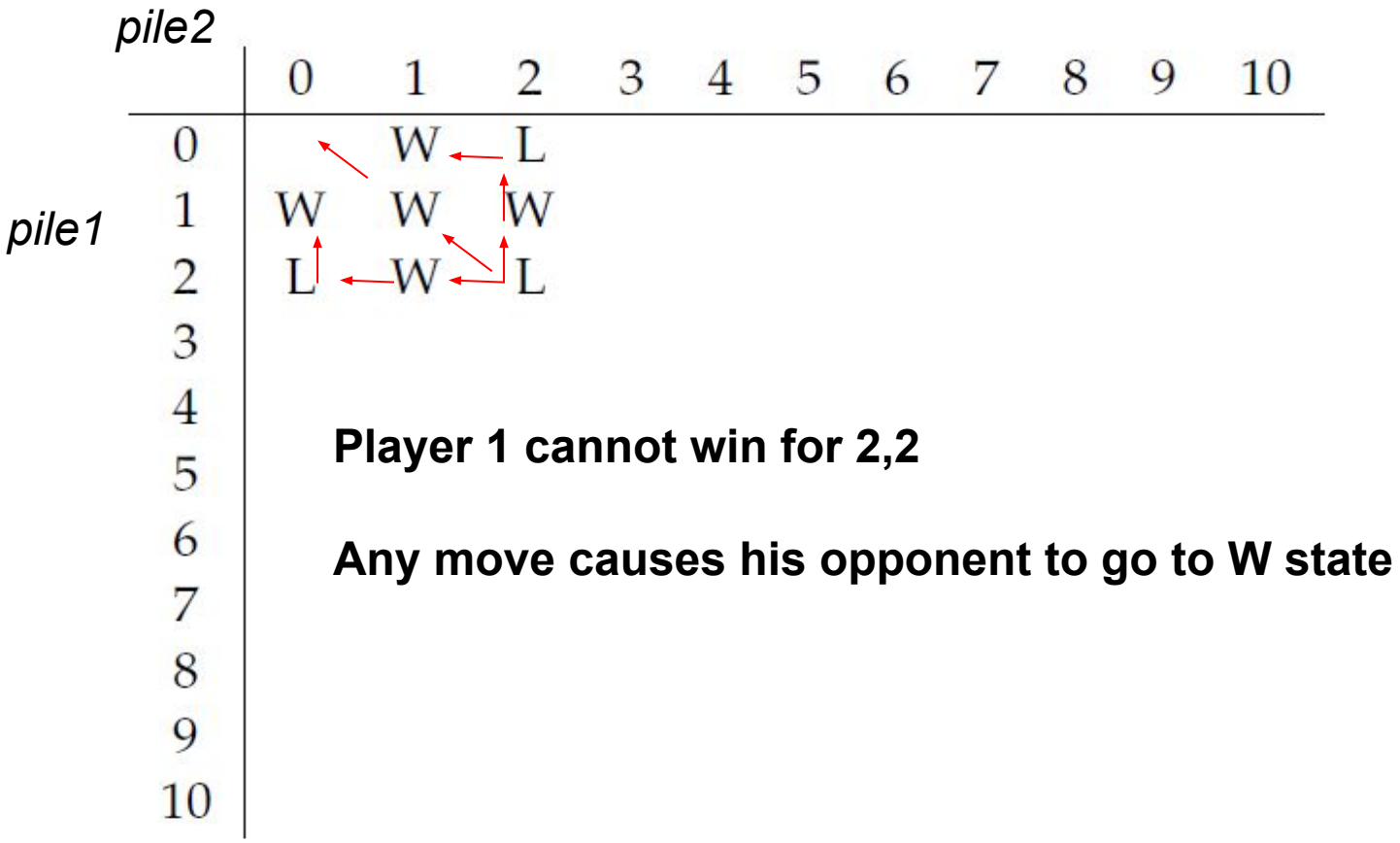
DP algorithm for Player 1



Player 1 can win for 2,1 if he picks one from pile2

Player 1 can win for 1,2 if he picks one from pile1

DP algorithm for Player 1



Player 1 cannot win for 2,2

Any move causes his opponent to go to W state

DP "moves"

When you are at position *(i,j)*

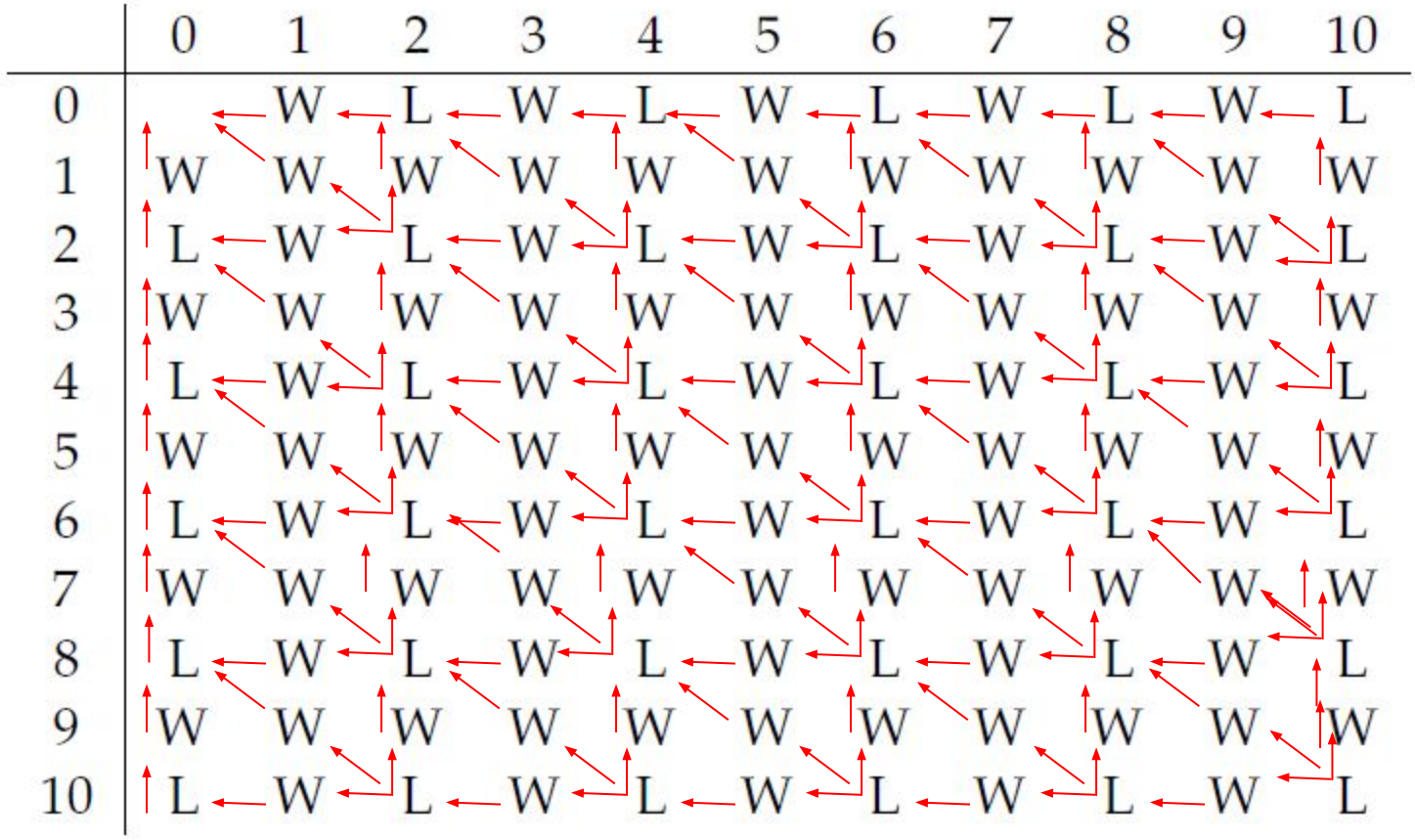
Go to:

Pick from pile 1: ***(i-1, j)***

Pick from pile 2: ***(i, j-1)***

Pick from both piles 1 and 2: ***(i-1, j-1)***

DP final table



	0	1	2	3	4	5	6	7	8	9	10
0		W	L	W	L	W	L	W	L	W	L
1	W	W	W	W	W	W	W	W	W	W	W
2	L	W	L	W	L	W	L	W	L	W	L
3	W	W	W	W	W	W	W	W	W	W	W
4	L	W	L	W	L	W	L	W	L	W	L
5	W	W	W	W	W	W	W	W	W	W	W
6	L	W	L	W	L	W	L	W	L	W	L
7	W	W	W	W	W	W	W	W	W	W	W
8	L	W	L	W	L	W	L	W	L	W	L
9	W	W	W	W	W	W	W	W	W	W	W
10	L	W	L	W	L	W	L	W	L	W	L

Also keep track of the choices you need to make to achieve W and L states: *traceback table*

Player 1 LOSES the game

Note on dynamic programming

Dynamic programming methods solve smaller subproblems first to solve the bigger problem.

INDUCTION

Therefore, when computing the solution:

DO NOT EVER START FROM THE SINK!

SOURCE -> SINK: computing the answer

SINK -> SOURCE: backtracking after the answer is calculated

Algorithm design techniques: CS473

- **Branch and bound:**
  - Omit a large number of alternatives when performing brute force
- **Divide and conquer:**
  - Split, solve, merge
    - Mergesort
- **Machine learning (CS 464):**
  - Analyze previously available solutions, calculate statistics, apply most likely solution
- **Randomized algorithms:**
  - Pick a solution randomly, test if it works. If not, pick another random solution

CS481/CS583: Attendance

- Classical sign-based attendance sheet is not sanitary during the pandemic
- One possible “solution”
  - Checkboxes on Moodle -- check your name if you attended for each day
    - Trust needed - but attendance does not affect grades directly anyway
    - Class photos may be added

Regulations due to the pandemic

1. As per university regulations, masks are compulsory
2. Keep social distancing whenever possible
3. Do NOT approach the dais before/after class to ask questions
    a. Make use of office hours & appt. instead
    b. All out-of-classroom meetings through Zoom only
4. Failure to wear masks or violating regulations any other way will result in being removed from the class and additional disciplinary action

Regulations due to the pandemic

1. Eating/drinking are not allowed during class
2. We can have 3-4 minute water break mid-lecture -- students can exit the room to have water
3. If you need to drink/eat at other times, you may leave the class without asking for permissions -- do not bother your classmates when doing so

Major Objective: keeping everyone healthy during the semester

Taking the attendance

- Taking the attendance is difficult during the pandemic
- No signup sheet
- (current) approximate solution: self reporting
- Fill in the Google Sheet below after each lecture day:
  - https://tinyurl.com/cs481attend
    - Link available on Moodle
  - Comment-only editing allowed
  - We will lock edits the next day & back up