Lecture 7 – Challenges in Computation

Alexander Lam

Matrices in Python

- A matrix in Python is represented as a list of lists.
- Each inner list represents a row of the matrix.

```
matrix = [
    [1,2,3],
    [4,5,6],
    [7,8,9]
]
```

Matrix Multiplication

- **Input:** An m x n matrix A and an n x p matrix B.
- **Output:** A m x p matrix C = AB.

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix}$$

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{pmatrix}$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

E.g. $c_{11} = a_{11}b_{1j} + a_{12}b_{21} + \cdots + a_{1n}b_{n1} = \sum_{k=1}^{n} a_{1k}b_{k1}$

Matrix Multiplication

- **Input:** An m x n matrix A and an n x p matrix B.
- **Output:** A m x p matrix C = AB.

$$A = \begin{pmatrix} 4 & 3 \\ 5 & 2 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & -2 \\ 3 & 4 \end{pmatrix}$$

$$A = \begin{pmatrix} 3 & 4 \\ 1 & 0 \\ -5 & 2 \end{pmatrix} \qquad B = \begin{pmatrix} 3 & 0 & 5 \\ 1 & 1 & 2 \end{pmatrix}$$

Matrix Multiplication

- **Input:** An m x n matrix A and an n x p matrix B.
- **Output:** A m x p matrix C = AB.

- Matrix multiplication is easy for computers to do
  - $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj}$

```
For i from 1 to n:
    For j from 1 to p:
        sum = 0
        For k from 1 to m:
            sum = sum + a_ik * b_kj
        Set c_ij = sum
```

Other easy problems

- Sorting a list
- Searching for a specific object in a list
- Finding the shortest path from point A to point B

Knapsack Problem

- You have a knapsack that can carry up to W weight of items.

- There are many available items, each item has a weight w_i and a value v_i




...

- You want to fit as much value into your knapsack as possible.

Knapsack Problem

- **Input:** Maximum weight capacity W, items with weights w_i and values v_i
- **Output:** A subset of items such that $\sum_i w_i \leq W$ and $\sum_i v_i$ is maximized.

Item	Weight	Value
Water Bottle	3	7
Headphones	4	8
Food	5	9
First-Aid Kit	6	10
Laptop	7	13
Tent	8	14

Knapsack Capacity is 15

Knapsack Problem

- **Input:** Maximum weight capacity W, items with weights w_i and values v_i
- **Output:** A subset of items such that $\sum_i w_i \leq W$ and $\sum_i v_i$ is maximized.

Item	Weight	Value	Value/Weight
Water Bottle	3	7	2.33
Headphones	4	8	2.00
Food	5	9	1.80
First-Aid Kit	6	10	1.67
Laptop	7	13	1.86
Tent	8	14	1.75

Knapsack Problem

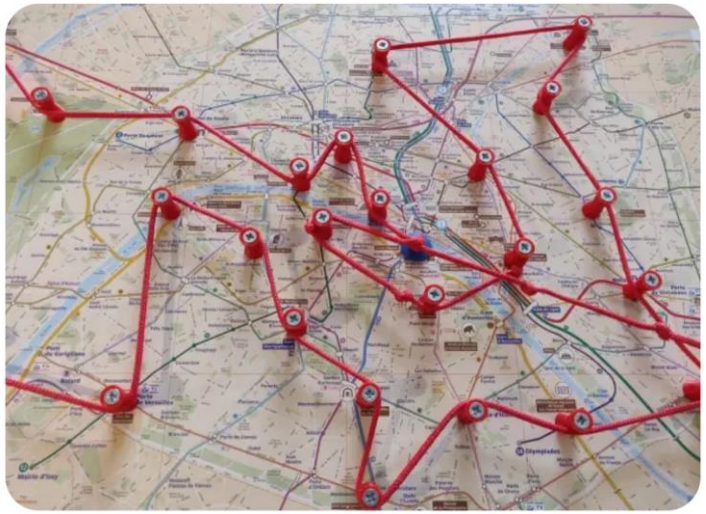
- **Input:** Maximum weight capacity W, items with weights w_i and values v_i
- **Output:** A subset of items such that $\sum_i w_i \leq W$ and $\sum_i v_i$ is maximized.

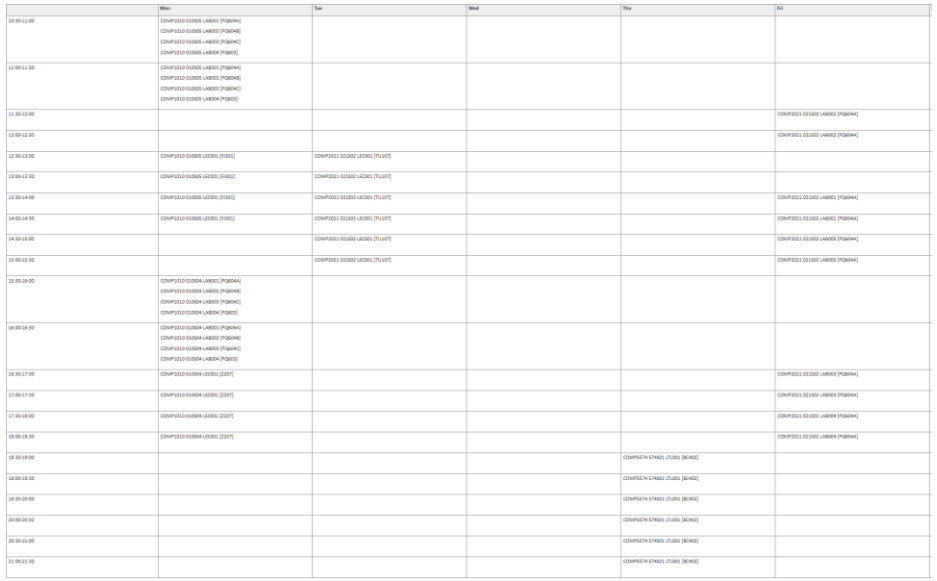
- We can't really do better than guessing and checking every combination of items

- The Knapsack Problem is hard for computers to solve

Other examples of ‘Hard’ problems

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9





Domino Matching

- Dominos
  - There are two parts of a domino: upper half and lower half.
  - Each part contains some pattern.
  - There are an infinite supply of each domino type.

- Problem
  - Given a set of domino types, is it possible to arrange them so that the combined pattern on upper half is the same as the combined pattern on the lower half?
- Example
  - Four types

b	abc	a	ca
ca	c	ab	a

  - Another four types

010	111	001	11
0	000	0101	10110

Domino Matching

- Example

b	abc	a	ca
ca	c	ab	a

- Here is a solution:
  - With 1 dominos of first type, 1 domino of second type, 2 dominos of third type, and 1 domino of fourth type.
  - Order: 3,1,4,3,2.

a	b	ca	a	abc
ab	ca	a	ab	c

abcaaabc
abcaaabc

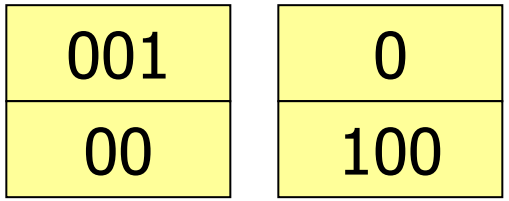
Domino Matching

- Example

010	111	001	11
0	000	0101	10110

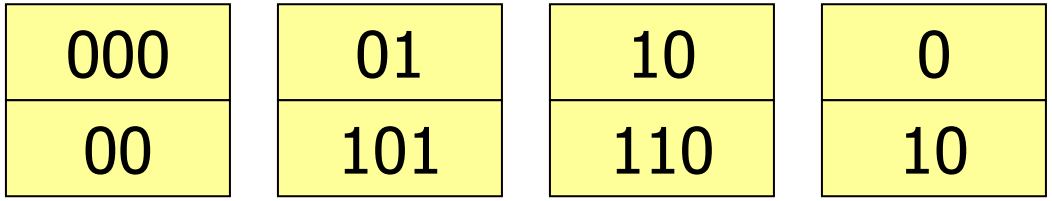
Domino Matching

- Example

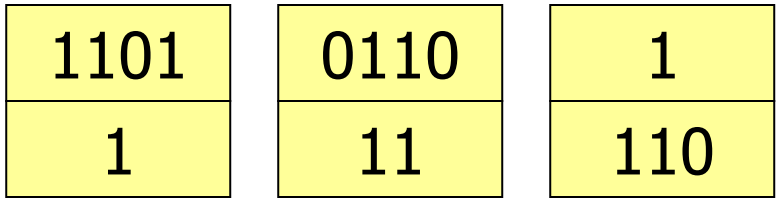


Domino Matching

- Example



Domino Matching

- Is it possible to write a program?
  - Input: a set of domino types.
  - Output: yes or no for existence of solution.
- Is there a solution for 

?
- Yes, but we need a total of 252 dominoes.

Domino Matching

- Is it possible to write a program?
  - Input: a set of domino types.
  - Output: yes or no for existence of solution.

- No, this program cannot exist.
- The problem is **undecidable**

- See Post Correspondence Problem for more details

Halting Problem

- **Input:** A program.
- **Output:** Whether or not that program eventually halts.


```
def MyProgram(Program):
    if HaltOrNot(Program) == Yes{
        loop forever
    else
        do nothing
```

Halting Problem

- **Input:** A program.
- **Output:** Whether or not that program eventually halts.


```
def MyProgram(Program):
    if HaltOrNot(Program) == Yes{
        loop forever
    else
        do nothing
```

```
MyProgram(MyProgram):
    if HaltOrNot(MyProgram) == Yes{
        loop forever
    else
        do nothing
```

What happens if HaltOrNot(MyProgram)==Yes?

What happens if HaltOrNot(MyProgram)==No?

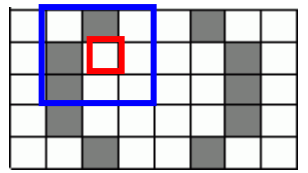
Halting Problem

- **Input:** A program.
- **Output:** Whether or not that program eventually halts.

- This problem is **also** undecidable!

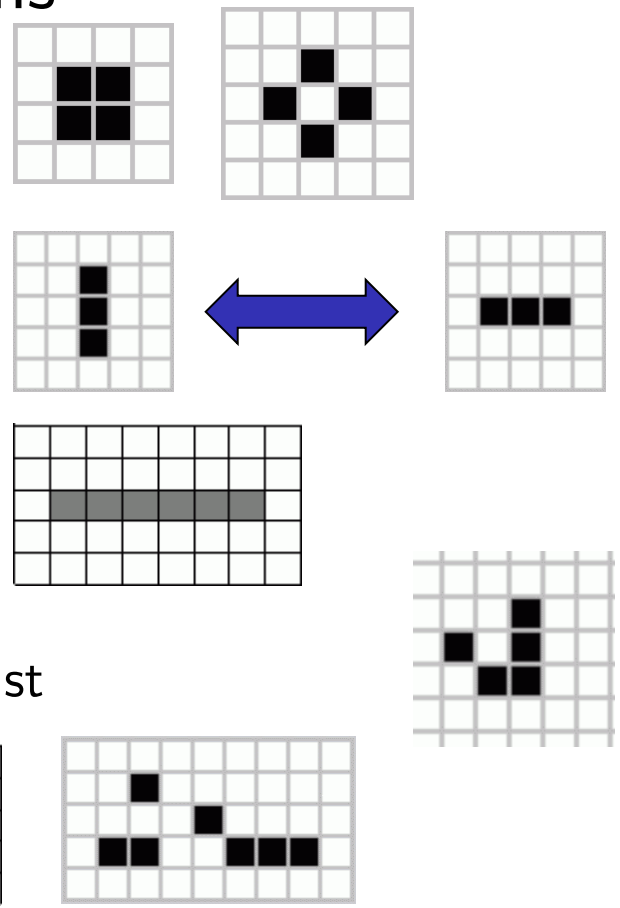
Game of Life

- Game of life is a classical problem in computing.
  - There are infinite square cells in rectangular shape.
    - Each cell has 8 neighbors.
  - Each cell either has an organism (e.g. amoeba) or not.
    - A cell with an organism is called a live cell.
  - On each generation (e.g. each clock tick):
    - The organism in a cell may die or survive.
    - An empty cell may give birth to an organism and become live.
  - Rules for next generation:
    - The organism in a live cell with 2 or 3 live neighbors survives.
    - The organism in a live cell with less than 2 or more than 3 live neighbors dies (lack of food or over-crowded).
    - An empty cell with exactly three live neighbors will give birth to an organism and become live.



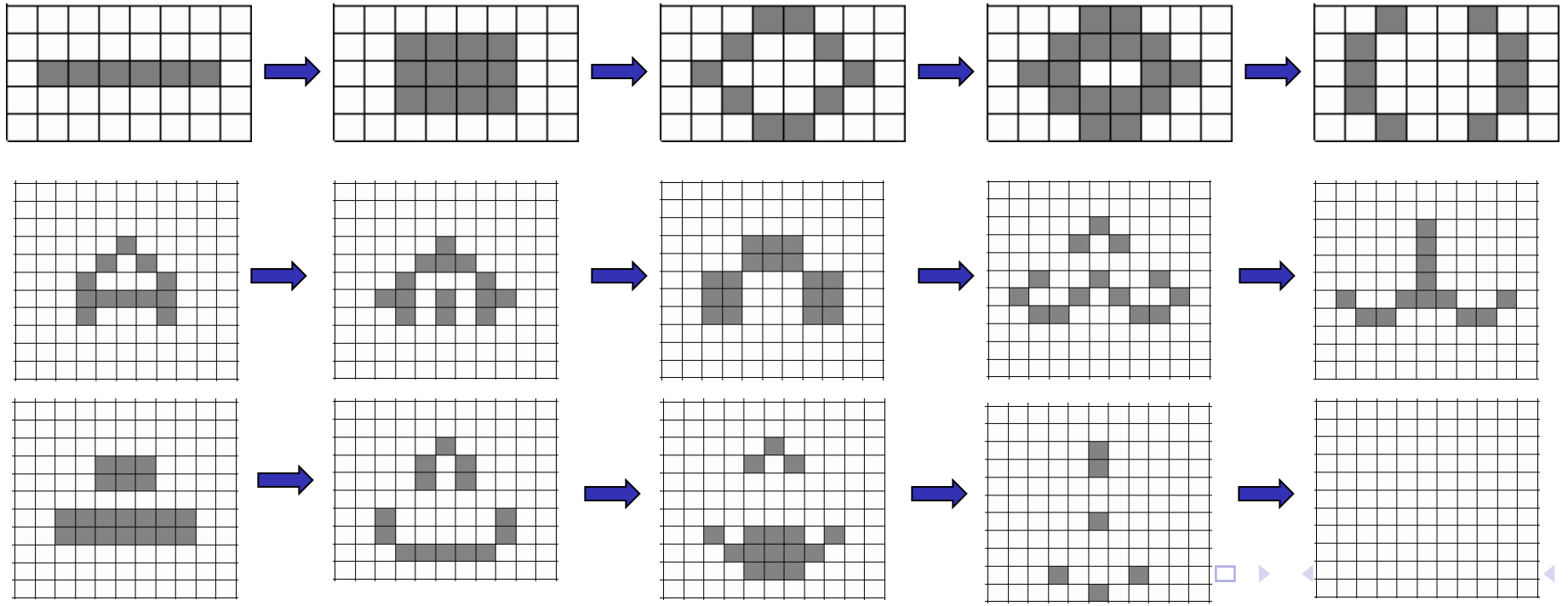
Game of Life

- Examples and evolutions
  - Stable patterns
    - Will not change
  - Pattern with cycle
    - Cycle = 2 to ...
  - A dying pattern
    - Die after 12 rounds
  - A moving pattern
    - Move towards south-east
  - Others



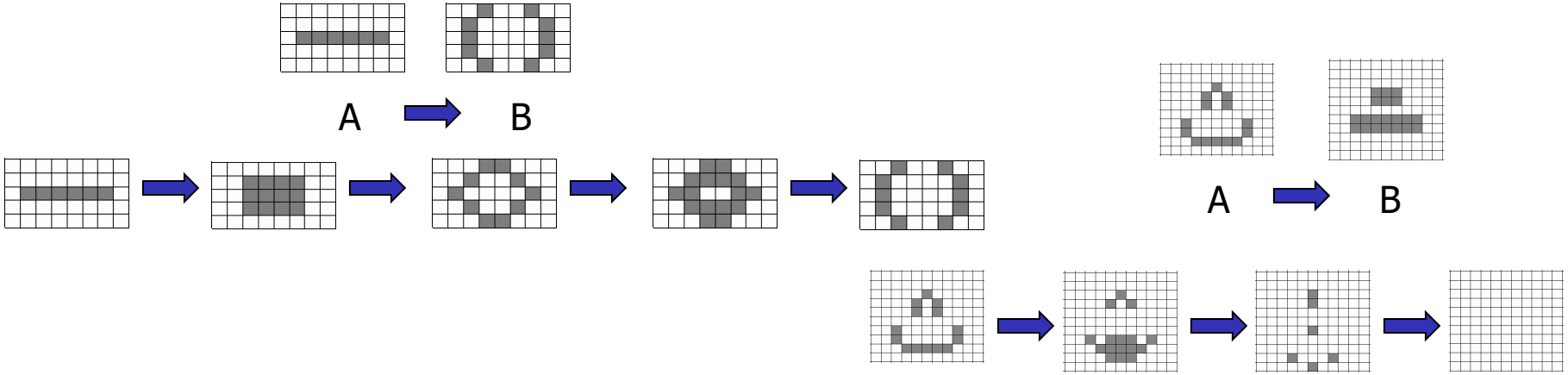
Game of Life

- Given a certain pattern, game of life will evolve to show different patterns as time goes on.
- Examples:



Game of Life

- Here is the problem:
  - Given a pattern A and another pattern B, is it possible for A to evolve into B after some steps?



- Is it possible to write a program?
  - Input: given input patterns A and B.
  - Output: output "yes" if A can evolve into B and output "no" if it is not possible for A to evolve into B.

Computation

- Computers can tirelessly execute programs.
- But bad programs will take too long to run!
- Is it always possible to find a fast solution?
  - Sometimes yes, but unfortunately, not always.
  - Some problems always take long time to run!
- Some problems even cannot be solved!
  - Examples: domino matching, game of life evolution
  - No need to attempt solving them
- For problems with solutions, one should look at the efficiency
  - Easy problems: should find efficient solutions
  - Hard problems: should find some solution


That's all Folks!