Lecture 11
Applications

COS10003 Computer Logic and Essentials (Hawthorn)

SWINBURNE UNIVERSITY OF TECHNOLOGY

Semester 1 2021

Today

1. Turing machines
2. Cellular automata
3. Knapsack problem

Why discrete maths is important

Some key CS problems

Turing Machines

The Turing Machine was proposed by Alan Turing in 1936 in a paper called "On Computable Numbers, with an Application to the Entscheidungsproblem".

Is there a mechanical procedure for determining whether a given mathematical statement is True or False?

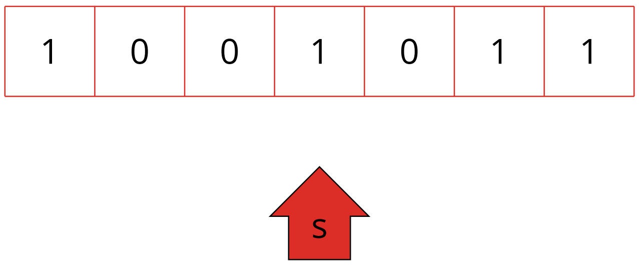
Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

Definition

The "hardware" of a Turing Machine consists of:

- A "tape" divided into squares where each square may contain any one of a finite number of symbols from the machine's "alphabet".
- A "read-write head" (RWH) that is positioned on the tape at some square and can read the symbol in that square, overwrite that symbol with a symbol, and then move one square left or one square right.
- A finite set of "states" (like gears in a car), one of which is designated the starting state, which we denote by s.

The tape and head



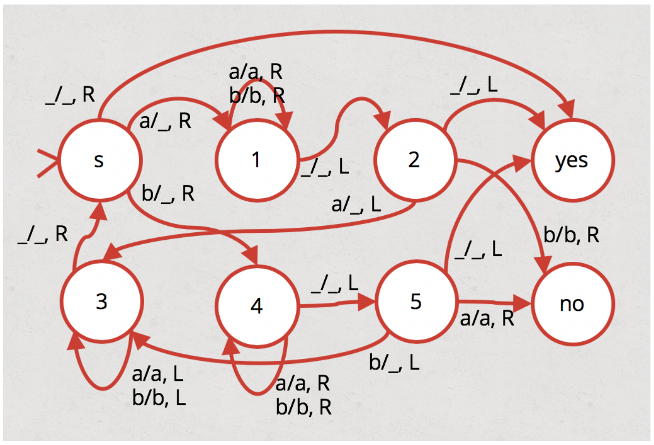
Definition

The actions of the machine are determined by a finite set of “quintuples” of the following form:
(p, x: y, dir, q)
where p and q are states of the machine, x and y are symbols in the machine’s alphabet, and dir is a direction the RWH might move (either L for left or R for right).

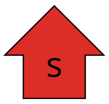
Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

Another example

Can we determine whether certain characters make a palindrome?



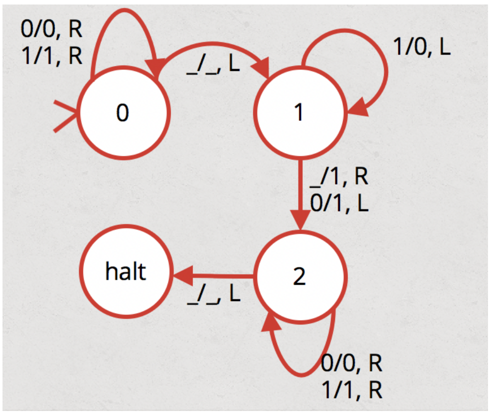
_	a	b	a	b	a	_



Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

For you to try

What does this machine do?



_	1	0	0	1	1	_

Universal Turing Machine

Given the process of a Turing Machine,

- could this process be described as an algorithm?
- could this process be done by a computer program?
- could this process be done by a Turing Machine, with enough states and tape-symbols?

Universal Turing Machine

Turing himself described how to construct a Universal Turing Machine, U. The input to U is a description $< M >$ of (the quintuples of) a Turing Machine M together with a description of the input tape $< w >$ for M. Then, U simulates the action of M on its own tape. This Universal Turing Machine is a model of modern, stored program computers; they accept programs as input and simulate the action of each program on its own data.

Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

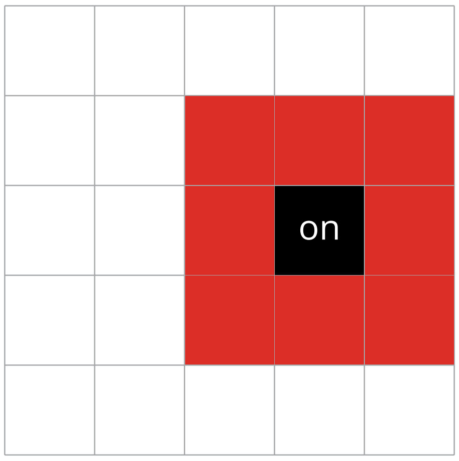
Church-Turing thesis

> The Church-Turing Thesis says, in effect, that any task that can be done by a computer (program) can be done by a Turing Machine. Therefore, any task that cannot be done by a Turing Machine cannot be done by a computer. Turing Machines not only provide a context for ascertaining algorithmic complexity, they determine what is possible to be computed and what numbers are "computable".
>
> Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

What is a cellular automaton?

- A cellular automaton is a model that helps understand real-world systems.
- It consists of a grid of cells, and each cell can be in a number of states.
- Each cell has neighbours, and at each step or generation, rules are used to determine whether the cell changes state.

In pictures



Game of Life

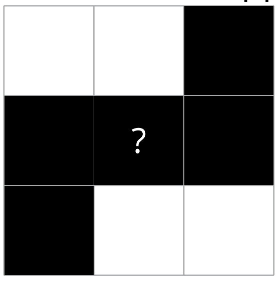
States are alive or dead, and each cell has 8 neighbours.

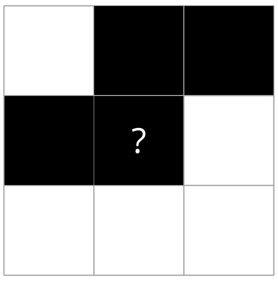
1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

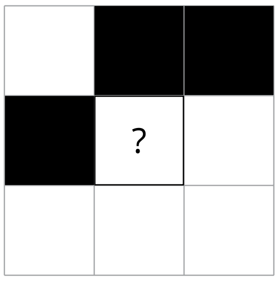
https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

For you to do

What will happen to the middle cell on the next step?



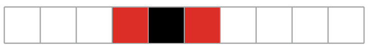




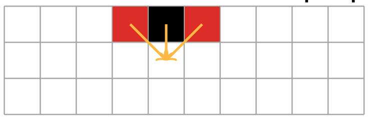
1D automata

- The grid is a line of cells

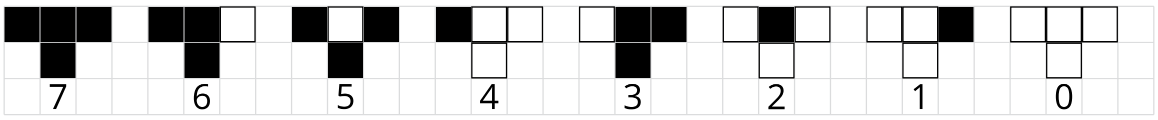
- Each cell depends on the cell left, above and right



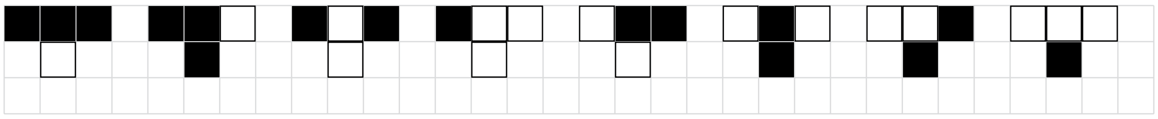
- Generations or steps proceed downwards



Different rules



Another example



Classes

- Class 1: Cellular automata which rapidly converge to a uniform state. Examples are rules 0, 32, 160 and 232.
- Class 2: Cellular automata which rapidly converge to a repetitive or stable state. Examples are rules 4, 108, 218 and 250.
- Class 3: Cellular automata which appear to remain in a random state. Examples are rules 22, 30, 126, 150, 182.
- Class 4: Cellular automata which form areas of repetitive or stable states, but also form structures that interact with each other in complicated ways.

https://en.wikipedia.org/wiki/Elementary_cellular_automaton

Class 4

Rule 110 is chaotic, and is also capable of universal computation.

Lecture 11
Applications

COS10003 Computer Logic and Essentials (Hawthorn)

SWINBURNE UNIVERSITY OF TECHNOLOGY

Semester 1 2021

Knapsack problem

Suppose you have (or a cat-burglar finds) a set of n objects $U = \{O_1, O_2, ..., O_n\}$ where each object O_j has a positive "weight" W_j and a positive "value" V_j. Suppose also that there is a positive value B equal to the total weight you (or the burglar) are willing to carry in your knapsack.

Find a subset X of U with a maximum total value subject to the constraint that the total weight of X is less than B.

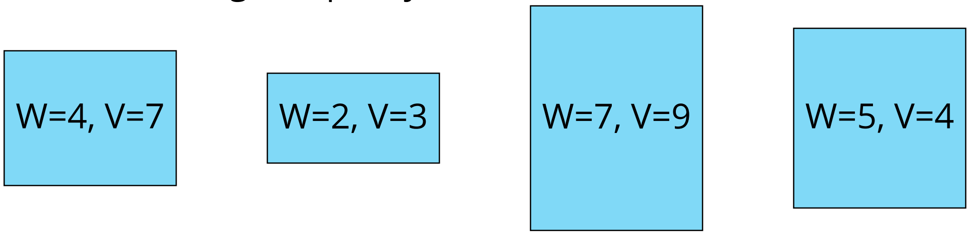
Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

Knapsack: brute force

We are considering the 0-1 knapsack problem, where only 0 or 1 of each item is in the knapsack. Other variations are fractional (items can be divided up) and multiple (many identical items exist).
A brute force solution is to generate all combinations and check for the best solution. For n items, this means 2^n combinations (each item is either in or out, and each is independent of each other).

Knapsack problem: greedy algorithm

Optimising this problem is NP-hard (no known polynomial time algorithm). A greedy approach is to take the heaviest item, then the next heaviest item, and so on. For weight capacity = 12:

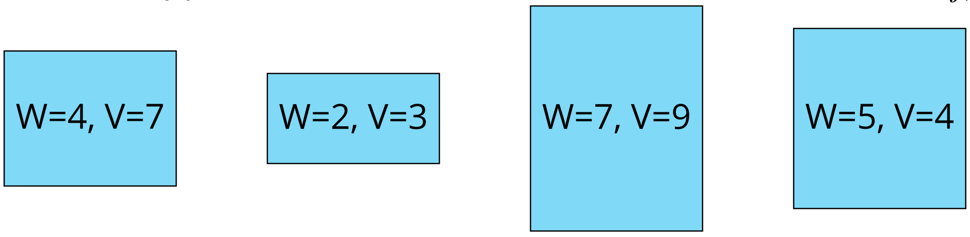


Weight=12, Value=13

Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

A greedy algorithm

Optimising this problem is NP-hard (no known polynomial time algorithm).
Another approach is to take the most ”value-dense” item (V_j/W_j).



Weight = 11, Value = 7 + 3 + 4 = 14

Jenkyns and Stephenson, Fundamentals of Discrete Math for Computer Science: A Problem-Solving Primer

Dynamic programming approach

Dynamic programming allows us to divide the problem into smaller problems, and then store incremental solutions or answers as we progress.
The algorithm for the knapsack problem proceeds to check one item at a time to see whether it is possible and beneficial to add it.

Algorithm

```
W = knapsack capacity
For w = 0 to W:
    Initialise V[0,w] = 0
For each item i:
    wi = weight of item i, vi = value of item i
    For each weight w from 0 to W:
        if w - wi >= 0:
            V[i,w] = max(V[i-1,w], vi + V[i-1,w-wi])
        else:
            V[i,w] = V[i-1,w]
        end if
    end for
end for
```

DP for knapsack

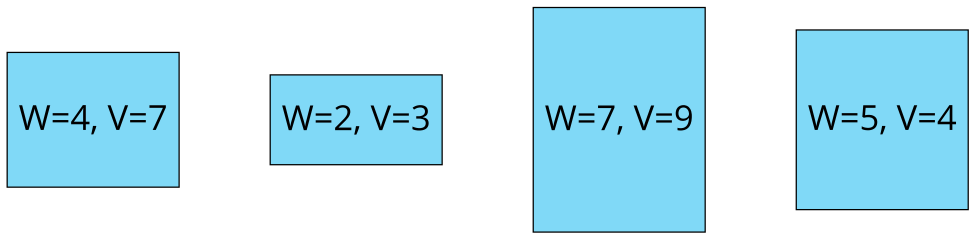
	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
item 1													
item 2													
item 3													
item 4													

Final table

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
item 1	0	0	0	0	7	7	7	7	7	7	7	7	7
item 2	0	0	3	3	7	7	10	10	10	10	10	10	10
item 3	0	0	3	3	7	7	10	10	10	12	12	16	16
item 4	0	0	3	3	7	7	10	10	10	12	12	16	16

Count backwards from the right-hand bottom corner to find the combination of items.

Final result



Weight = 11, Value = 7 + 9 = 16

Complexity of the knapsack problem

- Dominant factor is the nested loop: one with the number of items n and the other with each weight up to W
- Complexity is $O(nW)$
- Time needed can be improved by taking GCD of capacity and weights.

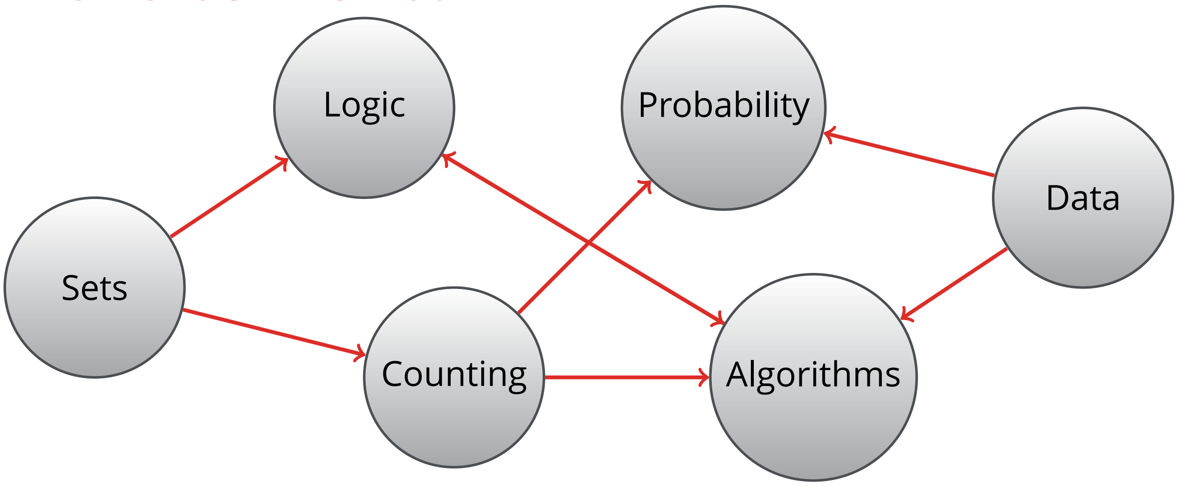
Applications of the knapsack problem

- Manufacturing: working out what to cut from which sheet
- Logistics: packing pallets, loading trucks
- Gaming: best items to put in inventory

Reflecting

- How are Turing machines defined?
- Which real-world problems is the knapsack problem useful for?
- How can complex systems help us understand the world around us?

Where to next?



In which we review the unit and talk about the exam.

Lecture 11
Applications

COS10003 Computer Logic and Essentials (Hawthorn)

SWINBURNE UNIVERSITY OF TECHNOLOGY

Semester 1 2021

Questions I still have

Topics I need to review