Algorithms 2: Machines

Algorithms 2: Machines

- The problem
- Programs
- Flowcharts
- Pseudocode

The problem

- Calculation by hand was boring and error-prone
  - Logarithms in particular were hard to work with
- Machines had been invented to do some calculations
  - Addition, subtraction, multiplication, division
- The machine could do the calculation, but everything around the calculation was done by hand
  - Which operation to do (addition, subtraction, etc.)
  - What values to operate on
  - Any decisions that had to be made (e.g., don’t divide by zero)
  - Any repetition of operations that needed to be done

The problem

- One can imagine the poor mathematician, sitting at a desk trying to find an answer
- First they set a number on some dials
- Then they set another number on some other dials
- Then they hit the “add” button
- Then they turn a mechanical crank so the answer shows up on another set of dials
- Then they write the answer down so they can use it later
- It must have been monotonous
  - But at least the answer was right
  - As long as they typed in the right numbers and followed the algorithm

The problem

- Eventually Charles Babbage invented a machine called the Analytical Engine
  - Around 1830-1870
- The big thing about this was that you could feed it a list of instructions which it could carry out on its own, rather than having to manually set up the numbers and then press the operation button and turn the crank yourself
- It could support more complicated instructions like

1. Here are two numbers, a and b
2. Add them
3. Do the following 5 times
4. Take the result of step 2 and add it to itself
5. Tell me the answer

The problem

- This is a recipe for calculating 6 * (a + b)
- The big change here is that, rather than having a mathematician follow the algorithm and use the tool for some calculations
- The Analytical Engine allowed for the machine to both follow the algorithm and do the calculations
- In a very real sense, one was specifying the algorithm to the machine
- Being able to do this was such a good idea that, had anybody understood how good an idea it was, they would have instantly wanted to use one
- Unfortunately, only Lady Ada Lovelace did
  - It would be a while before Turing put a rigorous mathematical foundation underneath everything
  - And even now we are not sure exactly how important an idea this is

The problem

- She decided to make the Analytical Engine calculate Bernoulli numbers
  - The Bernoulli numbers are a sequence of numbers with some interesting properties
- People can calculate Bernoulli numbers by hand
  - But it is tedious and time-consuming
- When she started, she had a well-understood algorithm
- The problem was there was a mismatch between the algorithm as it stood, which humans could do, and the operations the Analytical Engine could do
- Basically, the Analytical Engine needed more instructions, and more explicit instructions, than people did
- The algorithm had to be spelled out in a lot more detail than it had been before

The problem

- The machine was, simply put, really dumb compared to people
- So how could she express this algorithm so that the Analytical Engine could carry it out?
- And how could she tell other people what she had done?

Programs

- This lead to the idea of programs and programming
- A program is simply an algorithm that has been spelled out in enough detail that a machine can carry it out
- Programming is the process of creating a program
- Programming is hard
  - You will spend years learning how to do it
- It requires specialized languages that the computer can understand
  - Programming languages
- We are not concerned here with the properties of programming languages
  - That is another class

Programs

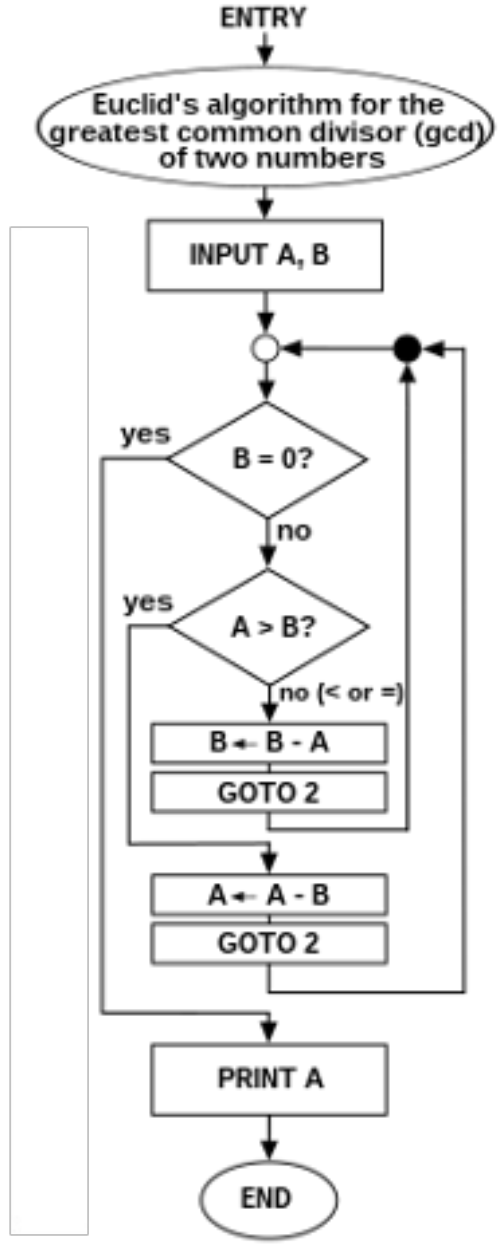
- What does concern us is that programming languages are not a good way for most people to communicate
- If you hand most people a program they will not understand what it does
  - The bigger the program, the less likely it is that anyone will understand it
- They could, given enough time and patience, carry out the program
  - But they still would probably not understand what the algorithm is or what it is for
- There are some deep questions here about what it means to ”explain” and “understand”
  - These are important questions in Artificial Intelligence
- But the main point is that programs are not good ways to describe algorithms to people
- We need a different way

Flowcharts

- A flowchart is a way to express an algorithm
  - It is particularly well-suited to expressing algorithms and computer programs in a way that human beings can read and understand
- Flowcharts are graphical
  - There are boxes and lines and symbols and text
- They have well-understood conventions
  - Certain symbols mean certain things
- The best introduction is to look at one

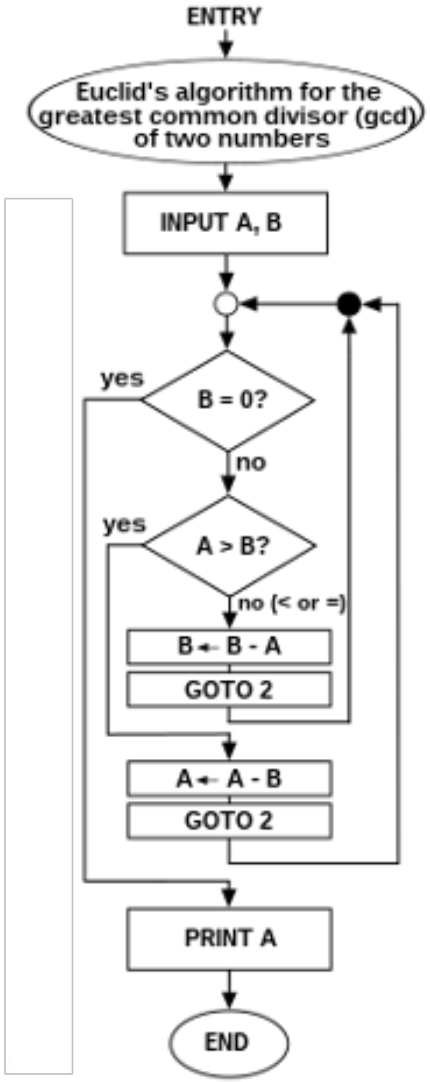
Flowcharts

Here is a flowchart for Euclid’s algorithm (from Wikipedia)



Flowcharts

- The parts are as follows
- ENTRY is where the algorithm starts
- The arrow shows where to go next
- The oval is a comment
  - Not part of the algorithm, it explains what is happening
- The box is an operation
  - Operations are simple things a computer can do
  - For example, read input, arithmetic operations, assign a value to a variable
- The diamond is a decision
  - Make the comparison in the diamond
  - Follow the yes or no arrow as appropriate
- The little balls are just there to show that lines connect and go to the same place

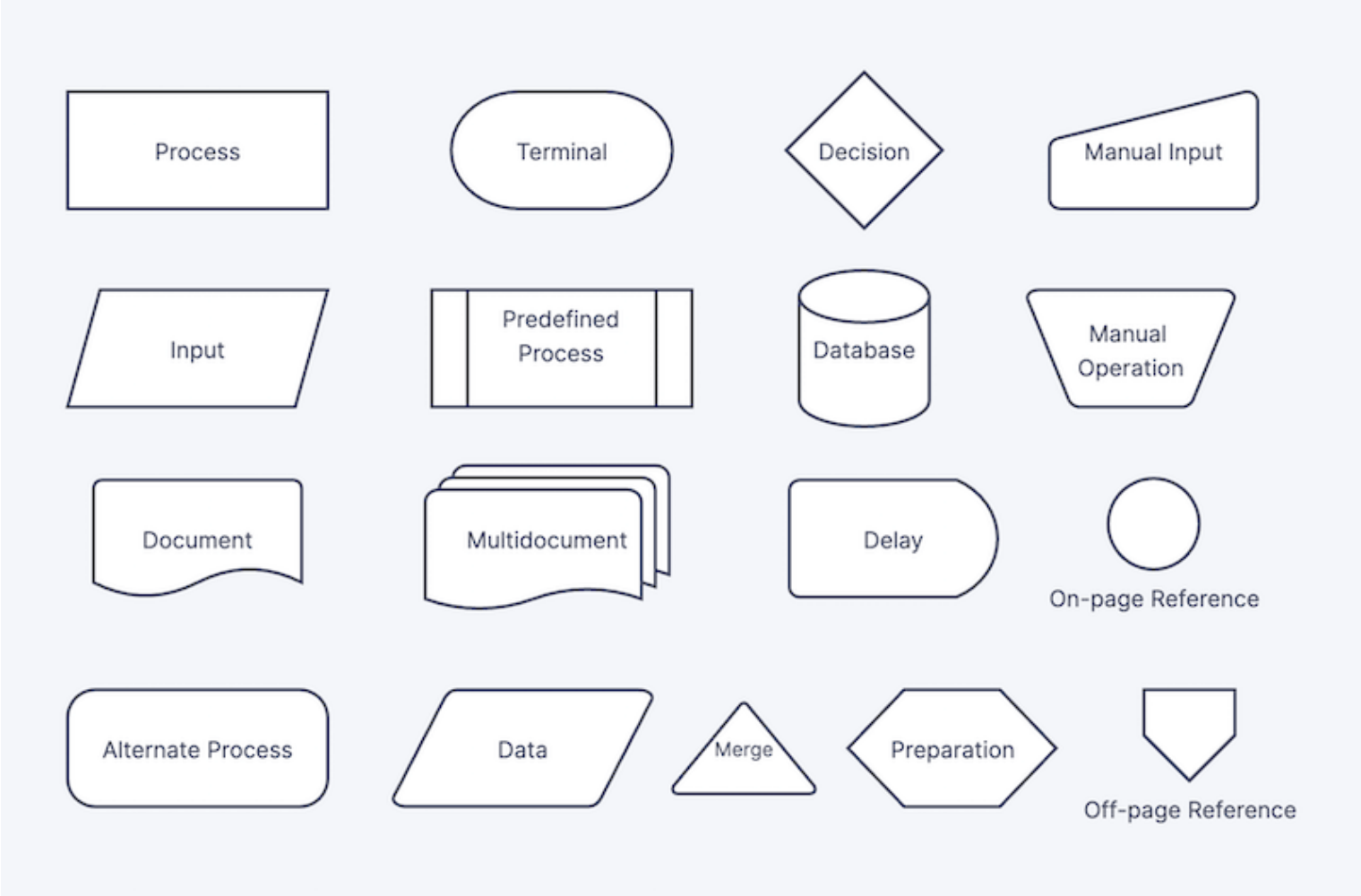


Flowcharts

- There are many different conventions for flowcharts
  - They are generally pretty similar
- For example:
- The text inside the shapes describes what the shapes do or represent
  - A question for a decision diamond
  - A name for a database icon
- The text associated with a line describes something about the line
  - What had to be true for that line to be followed
  - What data is being transmitted when the line is followed
- A couple of pictures will illustrate some different conventions

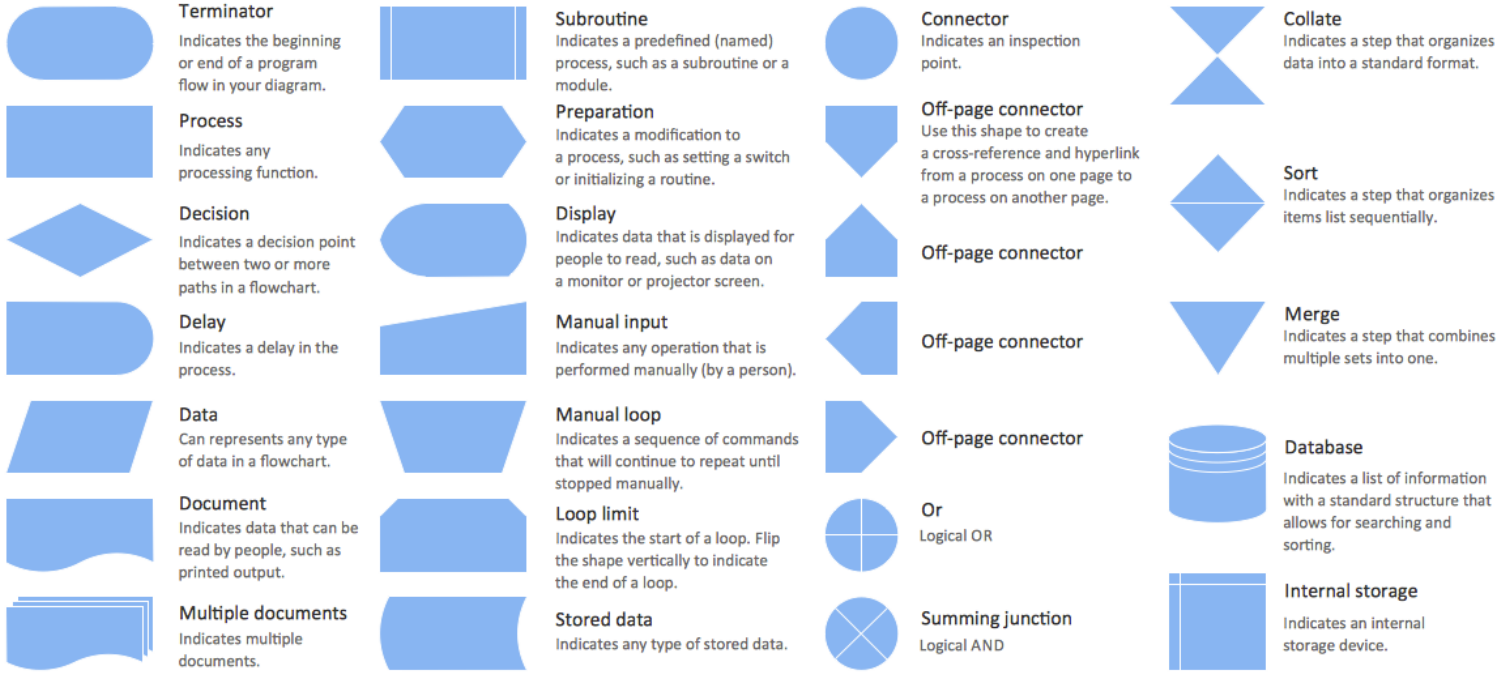
Here is one from zenflowchart.com

Flowcharts



And one from conceptdraw.com

Flowcharts



Flowcharts

- Despite differences in details, flowchart formats tend to be pretty similar
- For example:
- The text inside the shapes describes what the shapes do or represent
  - A question for a decision diamond
  - A name for a database icon
- The text associated with a line describes something about the line
  - What had to be true for that line to be followed
  - What data is being transmitted when the line is followed
- There is a great deal of flexibility in how an algorithm can be expressed as a flowchart
  - The point is to communicate, not to get the notation exactly right

Flowcharts

- When done well flowcharts can serve as an alternate means of communicating an algorithm
  - They are more detailed and precise than natural language
  - They are easier to understand than a programming language
  - They can be written at any level of abstraction so you can add or remove details according to the target audience
- They can be tricky
  - It can be difficult to read a flowchart that is bigger than one page
  - If you simplify things to fit on one page you can lose the precision that was part of the point in the first place

Flowcharts

- When flowcharts first came into practice it was thought that writing a program would involve the following steps

1. Think of an algorithm
2. Express it in a flowchart
3. If the flowchart is detailed enough, produce code directly from it
4. More likely, refine the flowchart by adding boxes and more detail until it is detailed enough to produce code from

- The flowchart could also be used to communicate with other people about the algorithm/program

Flowcharts

- One could show it to one's boss to get approval
- Other programmers could look at it and offer suggestions or help debug
- It could serve as documentation
  - Easier to read and understand than code
- If there were several pages of flowchart it could be divided up so that several programmers could work on the program at the same time
- It did not work out this way
- It turned out, once people learned how to write code better, and better programming languages came along, it was just easier and more accurate to read the code itself

Flowcharts

- There is a notation – the Unified Modeling Language (UML) – that takes the idea of flowcharts and runs with it
- UML is mostly confined to high-precision, high-stakes projects where every detail must be specified explicitly and clearly
- Your best bet if you ever have to create a flowchart is to find a program that lets you easily draw flowcharts
  - Many do
  - I like yEd
  - Your mileage may differ
- For small flowcharts drawing the chart by hand is usually adequate

Flowcharts

- When reading a flowchart it can be helpful to use a finger and follow along
- When creating a flowchart the same idea applies
  - Envision the algorithm as a series of steps that flow one into the other
- The key question is “what happens next?”
- There are three categories of what can happen next

1. A simple statement
   - Print something, do a calculation, get input
2. A decision
   - Compare two values, ask the user whether to continue
3. Repeat a previous box
   - Go back to some previous spot in the flowchart and do something again

Flowcharts

- When expressing an algorithm as a flowchart:

1. Try to understand the algorithm yourself before starting to make a flowchart
2. Start at the beginning and ask “what happens next?”
3. If that is something new
   - Pick the appropriate icon for that box and write it down
   - Fill in the box with a description of what is supposed to happen
   - Draw a line from the current box to the new one
4. Otherwise, draw a line from the current box to the already-existing box
   - If the current box is a decision, be sure to do this for both options
5. Go back to step 2, using the current box rather than the beginning, until you have covered the entire algorithm

Pseudocode

- Pseudocode is closer to code than flowcharts
- It is written in a more linear fashion, as mostly text
- The end result is something that looks a lot more like a program than a flowchart, but will not actually run on the computer
- The goal in pseudocode is to use various notations that are available to people, but are not in the computer language being used
- Pseudocode can vary a great deal in how much like code it is
  - It is not nearly as standardized as flowcharts are

Pseudocode

- Here is pseudocode that is very informal

This is the pseudocode for a Game of Monopoly, including one person's move as a procedure:

```
Main Procedure Monopoly_Game
    Hand out each player's initial money.
    Decide which player goes first.
    Repeat
        Call Procedure Monopoly_Move for next player.
        Decide if this player must drop out.
    Until all players except one have dropped out.
    Declare the surviving player to be the winner.

Procedure Monopoly_Move
    Begin one's move.
    Throw the dice.
    Move the number of spaces on the board shown on the dice.
    If the token landed on "Go to Jail,"
        then go there immediately.
    Else if the token landed on "Chance" or "Community Chest,"
        then draw a card and follow its instructions.
    Else
        follow the usual rules for the square (buying property,
        paying rent, collecting $200 for passing "Go", etc.).
    End one's move.
```

Pseudocode

- This pseudocode is much more formal
- Note that, despite the mathematical specificity, it still will not run as a program

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;
for $k = 1...K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
  end
  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
end
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Figure 1: graphs

Pseudocode

Ideally, one could translate (more or less) directly from pseudocode into code (and back again)

Example usage

i	x_i	y_i
1	in function main	`int main() {`
2	let n be integer	`int n;`
3	read n	`cin >> n;`
4	let A be vector of integers	`vector<int> A;`
5	set size of A = n	`A.resize(n);`
6	read n elements into A	`for(int i = 0; i < A.size(); i++) cin >> A[i];`
7	for all elements in A	`for(int i = 0; i < A.size(); i++) {`
8	set min_i to i	`int min_i = i;`
9	for j = i + 1 to size of A exclusive	`for(int j = i+1; j < A.size(); j++) {`
10	set min_i to j if A[min_i] > A[j]	`if(A[min_i] > A[j]) { min_i = j; }`
11	swap A[i], A[min_i]	`swap(A[i], A[min_i]);`
12	print all elements of A	`for(int i=0; i<A.size(); i++) cout<<A[i]<<" ";`
		`}`

Public test case 1 (out of 5): `5 3 2 4 1 5` → `1 2 3 4 5`
Hidden test case 1 (out of 8): `8 9 2 4 5 6 2 7 1` → `1 2 2 4 5 6 7 9`

Figure 1: Given L pseudocode lines $x_{1:L}$ (with indentation levels $\ell_{1:L}$) and public test cases, our task is to synthesize a program with code lines $y_{1:L}$. The program is evaluated against both public and hidden test cases.

Pseudocode

- When translating from code to pseudocode a few guidelines are useful
- Try to leave out language-specific wording
  - Note how the previous example changes loops to “do this n times” or “do this for all x”
- Type declarations can be relaxed or eliminated
- Typography can be relaxed
  - Programming languages do not support subscripts, superscripts, integration signs, or the like
  - But they are fine in pseudocode
- Understand your target audience
  - Do not put a bunch of math into pseudocode meant for a general audience
  - Just describe what the equation is supposed to calculate