CMPU4060-E
Object Oriented Software Development 1

Dr Andrew Hines

Room KE-G 026A
School of Computing
Dublin Institute of Technology

Algorithm Definitions

Stone, 1972

A set of rules that precisely defines a sequence of operations

Algorithm Definitions

Boolos and Jeffrey (1974, 1999)

No human being can write fast enough, or long enough, or small enough[a] to list all members of an enumerably infinite set by writing out their names, one after another, in some notation. But humans can do something equally useful, in the case of certain enumerably infinite sets: They can give explicit instructions for determining the nth member of the set, for arbitrary finite n. Such instructions are to be given quite explicitly, in a form in which they could be followed by a computing machine, or by a human who is capable of carrying out only very elementary operations on symbols.

---

[a] "smaller and smaller without limit ...you'd be trying to write on molecules, on atoms, on electrons"

https://www.youtube.com/watch?t=61&v=CvSOaYi89B4

Algorithms

- Precise step-by-step instructions
- A formal (computer) algorithm is detailed
- Remove inferred or implied steps or inputs
- No ambiguities (that we take for granted) in person to person interactions

http://people.cs.pitt.edu/ jmisurda/teaching/cs4/2064/cs0004-2064-algorithm.htm
http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/

Group Task

1. Create an algorithm to explains an everyday task
2. Specify Inputs (recipe ingredients)
3. Specify method – process by which you accomplish task
4. Specify outputs
5. List any assuptions

Example: Calling a friend on the telephone

Input

The telephone number of your friend to ask what time they are meeting you tomorrow

Output

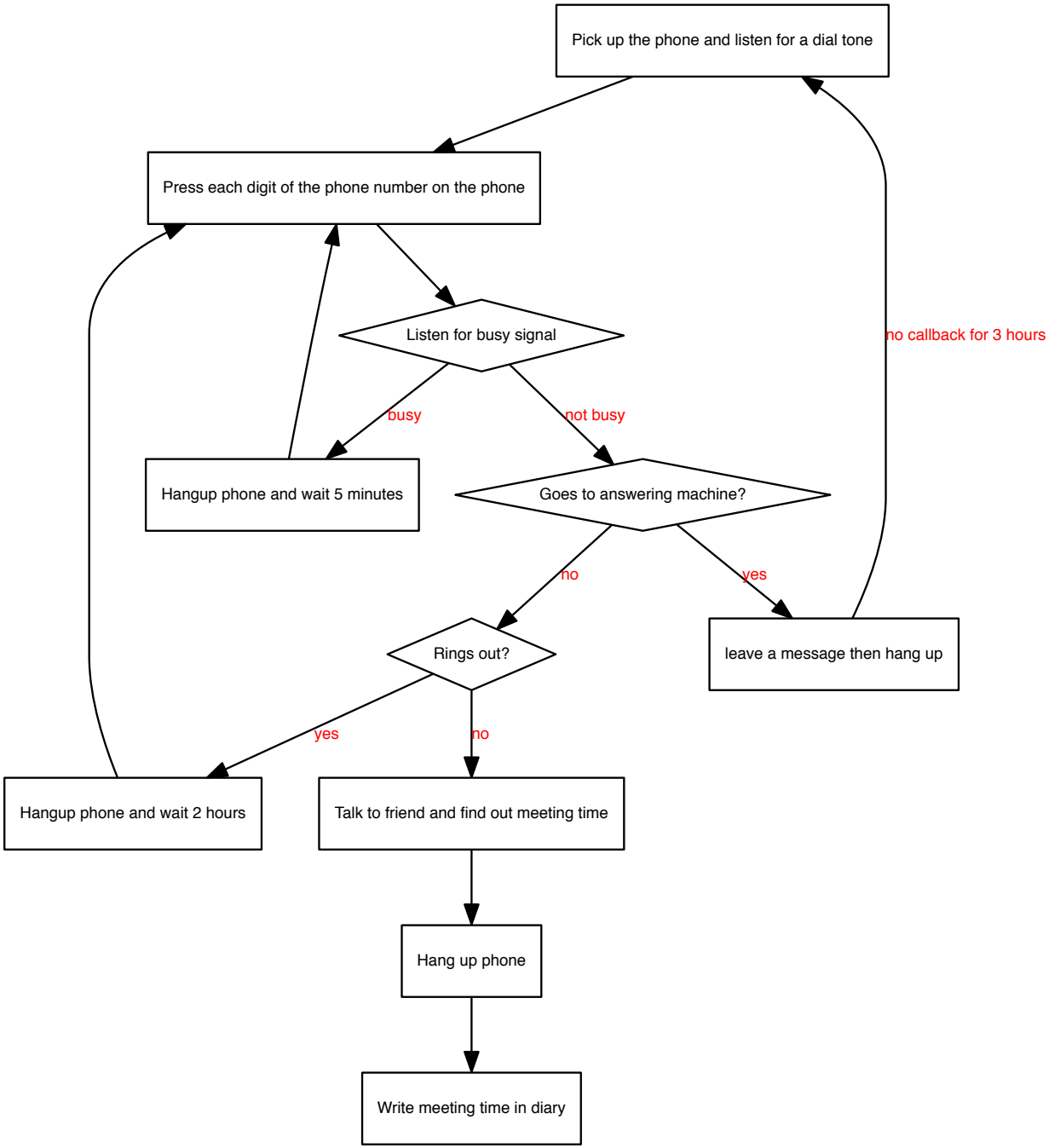
Meeting time

Steps:

1. Pick up the phone and listen for a dial tone
2. Press each digit of the phone number on the phone
3. If busy, hang up phone, wait 5 minutes, jump to step 2
4. If no one answers, leave a message then hang up
5. If no answering machine, hang up and wait 2 hours, then jump to step 2
6. Talk to friend and find out meeting time
7. Hang up phone
8. Write meeting time in diary

Assumptions:

1. Step 1 assumes that you live alone and no one else could be on the phone
2. The algorithm assumes the existence of a working phone and active service
3. The algorithm assumes you are not deaf or mute
4. The algorithm assumes a normal corded phone


Pick up the phone and listen for a dial tone
Press each digit of the phone number on the phone
Listen for busy signal
busy
not busy
Hangup phone and wait 5 minutes
Goes to answering machine?
no
yes
Rings out?
leave a message then hang up
no callback for 3 hours
yes
no
Hangup phone and wait 2 hours
Talk to friend and find out meeting time
Hang up phone
Write meeting time in diary

Group Algorithms

- Make tea (multiple cups? preferences?)
- Brushing teeth (how to make sure they were all equally cleaned?
- getting dressed (different weather)
- Reading a book
- Pumping your bike tyres

20 mins – Goal: 1 page (legible!) algorithm

Group Work Reflection

Group Algorithms Review – Picture Sheets

Individual Questions

- Were the algorithms precise?
- Were they different?
- What might you do differently next time you try to write a program?

Lists

A list is a stored sequence of items

- We saw lists of numbers last week
  - e.g. `oddNumbers = [1, 3, 5, 7]`
- They can also contain other types (e.g. strings) of data or even mixtures of items
  - `names = ['Frank', 'Bob', 'Alice']`
  - `miscStuff = [15, 'Bob', -3.14, True]`
- Like loop ranges they are zero indexed

0	15
1	'Bob'
2	-3.14
3	True

Lists

We can carry out a many kinds of operations on lists
`miscStuff = [15, 'Bob', -3.14, True]`

0	15
1	'Bob'
2	-3.14
3	True

- Access values in lists via their index, e.g. `myName = miscStuff[1]`
- Find the number of items in a list `len(miscStuff)` will return 4

Read more about lists in chapter 14 of Johnson textbook (p.75)

Strings

- Basic Python data type (like integer, float, boolean)
- Strings can be thought of as a collection (or list of characters)
- Strings can be contained between either single (') or double (") quotation marks (but they need to match)
- We have used lots of strings, as variables, inputs and outputs
  - `print("Hello World!")`
  - `input("What is your name?")`
  - `airportCode = 'DUB'`

```
# piglatin.py
def firstvowelindex(word):
    i = 0
    while word[i] not in "aeiouy":
        i += 1
    return i

def piglatin(word):
    i = firstvowelindex(word)
    if i == 0:
        return word + "yay"
    return word[i:] + word[:i] + "ay"

def main():
    w = input("Enter a word: ")
    print("In Pig Latin, that is", piglatin(w))

main()
```

Indexing (and Negative Indexing)

```
cityName = 'Dublin'
```

`cityName` →

0	1	2	3	4	5
D	u	b	l	i	n

`cityName` →

-6	-5	-4	-3	-2	-1
D	u	b	l	i	n

`s[i]` Character at index i

What is `cityName[2]`? Or `cityName[-2]`?

Slicing

Slicing

Accessing a group of consecutive characters

`s[i:j]` Slice from `i` to `j-1`

Examples

`cityName[2:5]` is `'bli'` `cityName[-4:-2]` is `'bl'`

`s[:j]` Slice from beginning to `j-1`
`s[i:]` Slice from `i` to end
`s[:]` Full slice, which creates a **copy** of `s`
`s[i:j:k]` Slice using a step (as per `range`)

Concatenation

Gluing strings together

`s = s + t` `s` becomes `s` with `t` appended
`s +=t` same as above
`s = s * n` The string `s` repeated `n` times

Example

```
cityName = 'Dublin'
cityName += ' Six' cityName now contains 'Dublin Six'
```

Substrings

in/not in

Useful to check if a string contains another string or a character, e.g. is there an 'i' in 'Dublin'?

`x in s`	True if x is a substring of s; otherwise false
`x not in s`	Opposite of above

Read more about strings in chapter 11 of Johnson textbook (p.61). There are a lot more you can do with strings (covered in chapters 12 and 16).

Dictionaries

A dictionary is a mapping that stores keys with associated values.

```
dict = <key>:<value>, ..., <key>:<value>
```

For every key in a dictionary, there is exactly one value. Sets of "`key:value`" pairs. To look up data in the dictionary:

- provide the key
- dictionary will return the value.

Example: Dictionary of peoples ages

```
ages = {"Alice": 21, "Bob": 25, "Charlie": 34}
```

Dictionaries

- Add values using the key as the index
- Retrieve from the dictionary quickly using the key – no need to look at every entry.

`d[key] = value` Set value associated with key in d to value.
`d[key]` Get value associated with key in d.
(`KeyError` raised if key not in d.)

Using a list as a dictionary

```
def getAgeOfPerson(searchName, peopleList, ageList):
    for personIndex, personName in enumerate(peopleList):
        if searchName == personName:
            return ageList[personIndex]
    return None #No person found

def main():

    peopleList=["Alice","Bob","Charlie"]
    ageList=[21,25,34]

    for searchName in peopleList:
        age=getAgeOfPerson(searchName, peopleList, ageList)
        print(searchName, "is", age)

    print(getAgeOfPerson("Frank", peopleList, ageList))

main()
```

Using a dictionary as a dictionary

```
ageDict={"Alice": 21, "Bob": 25, "Charlie": 34}

peopleList=list(ageDict)

for searchName in peopleList:
        age=ageDict[searchName]
        print(searchName, "is", age)
```

Using a dictionary as a dictionary

```
import time

def getAgeOfPerson(searchName, peopleList, ageList):
    for personIndex, personName in enumerate(peopleList):
        if searchName == personName:
            return ageList[personIndex]
    return None #No person found

def main():
    peopleList=["Alice","Bob","Charlie"]
    ageList=[21,25,34]
    start = time.clock()
    for searchName in peopleList:
        age=getAgeOfPerson(searchName, peopleList, ageList)
        print(searchName, "is", age)
    stop = time.clock()
    elapsed = stop - start
    print("Time to execute",elapsed)

    ageDict = {"Alice": 21, "Bob": 25, "Charlie": 34}
    start = time.clock()
    for searchName in peopleList:
            age=ageDict[searchName]
            print(searchName, "is", age)
    stop = time.clock()
    elapsed = stop - start
    print("Time to execute",elapsed)

main()
```

Output

```
Alice is 21
Bob is 25
Charlie is 34
Time to execute 6.0000000000004494e-05
Alice is 21
Bob is 25
Charlie is 34
Time to execute 2.2999999999995246e-05
```

Class Test Next Week: 15 October

- Test next week will be in lab on Thursday 15 October, 2–4 pm.
- Assessing theory covered in first 4 weeks - short written answers
- Lab exercises creating small programs in Python
- You will submit you code via webcourses
- There will be marks for correctly following the instructions
  - submit through webcourses before timelimit.
  - Name your file <studentnumber>.py

Arrive on time (or early) and get logged into PC, Webcouses, and pycharm open