

# Data Structures and Algorithms with Python

**Heikki Peura**

`h.peura@imperial.ac.uk`

**Lecture 5**

# We've covered a lot

## Toolkit:

- ▶ Computational and algorithmic thinking
- ▶ Complexity analysis
- ▶ Data structures and OOP
- ▶ Python for problem solving

## Applications:

- ▶ Square roots, palindromes, etc
- ▶ Searching and sorting
- ▶ Monster fights
- ▶ Shortest paths in social networks (Kevin Bacon): BFS; Dijkstra's algorithm (extra)
- ▶ Greedy algorithms and heuristics, knapsack problem (today)

# Heron's recipe

## How to find the square root of $x$ :

- ▶ Start with a guess  $g$
- ▶ If  $g * g$  is close to  $x$ , stop and return  $g$  as the answer
- ▶ Otherwise make new guess as the average of  $g$  and  $x/g$
- ▶ Repeat process using new guess

```
1 eps = 0.01
2 x = 50
3 guess = x/2
4 while abs(guess**2-x) >= eps:
5     guess = (guess + x/guess)/2
6 print("The square root of ", x, " is approximately:", guess)
```

# Analysing algorithm complexity

**Principle 0:** measure amount of work the computer does: count basic operations as function of input size

**Principle 1:** focus on worst-case analysis

**Principle 2:** ignore constant factors and lower-order terms

**Principle 3:** only care about large inputs

**Formal way to describe this approach:**

- ▶ Big-Oh notation: upper bound on worst-case running time

# Binary search

**Algorithm** for finding  $x$  in sorted list  $L$ :

- ▶ Pick an index  $i$  roughly dividing  $L$  in half
- ▶ If  $L[i] == x$ , return True (if nothing left to search return False)
- ▶ If not:
  - ▶ If  $L[i] > x$ , recursively search left half of  $L$
  - ▶ Otherwise recursively search right half

Find number 24 in a list  $L = [9, 24, 32, 56, 57, 59, 61, 99]$

First iteration

9	24	32	<b>56</b>	57	59	61	99
---	----	----	-----------	----	----	----	----

9	24	32	56	57	59	61	99
---	----	----	----	----	----	----	----

$L[i] = 56 > 24 \rightarrow$  discard right half and recursively call binary search on left half

Second iteration

9	<b>24</b>	32	56	57	59	61	99
---	-----------	----	----	----	----	----	----

$L[i] = 24 \rightarrow$  return True

# Complexity classes

**Fast algorithm**: worst-case running time grows slowly with input size

- ▶  $O(1)$ : constant running time — basic operations
- ▶  $O(\log n)$ : logarithmic running time — binary search
- ▶  $O(n)$ : linear running time — linear search
- ▶  $O(n \log n)$ : log-linear running time — merge sort
- ▶  $O(n^c)$ : polynomial running time — selection sort
- ▶  $O(c^n)$ : exponential running time — ??

# Everything is an object

An **object** has:

- ▶ A type: `int`, `str`, `list` (`L=[0,1]` is an **instance** of a `list`)
- ▶ An internal representation of data
- ▶ A set of functions that operate on that data

```
1 L = [1,1999,0,-2,9]
2 L.append(8)
3 L.insert(2,1000)
4 t = L.pop()
5 L.remove(1)
6 help(L)
```

## The point:

- ▶ Interface: the user knows what she can do with a list
- ▶ Abstraction barrier: the user does not need to know the details of what goes on under the hood (similarly to functions)
- ▶ Invaluable in managing complexity of programs

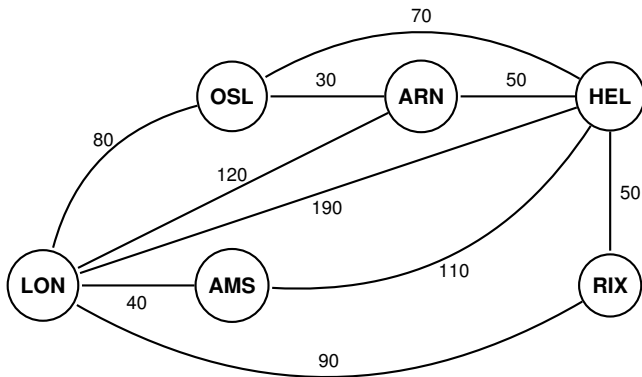
# How to organize data?

## What is the right data structure for my problem?

- ▶ What operations do I need to perform on data?
- ▶ Is a built-in data structure enough?
- ▶ Is algorithm efficiency important?



# Graphs



A set of **nodes** (or vertices) connected by **edges**

Edges may or may not be directed -> **directed** or **undirected** graph  
(Twitter vs Facebook?)

Edges may have **weights** (eg prices): weighted graph

# Breadth-first search (BFS)

## Find shortest-distance paths between people in social networks?

- ▶ Path: a way to get from node  $v$  to node  $w$  via graph edges
- ▶ Distance: how many steps you need to take

### Idea: explore graph in layers

- ▶ Choose starting node
- ▶ Explore all nodes connected to the starting node (1st degree of separation)
- ▶ Explore all nodes connected to these nodes (2nd degree of separation)
- ▶ Repeat...

# Breadth-first search (BFS)

Algorithm: BFS(unweighted graph  $G$ , starting node  $s$ )

- ▶ **Initialize**: mark all nodes unexplored except  $s$  explored
- ▶ Use  $Q$  = queue data structure, add  $s$  to  $Q$
- ▶ **Main loop**: While  $Q$  is not empty:
  - ▶ Remove the first node of  $Q$  and call it  $v$
  - ▶ For each edge  $(v, w)$ : if  $w$  unexplored:
    - ▶ Mark  $w$  explored
    - ▶ Add  $w$  to  $Q$

What is a **queue**?

- ▶ First in, first out (just like in a café)
- ▶ Remove from front, insert to back in constant time  $O(1)$
- ▶ Here we keep a queue of nodes to process next

# Knapsack problem

**Problem:** fill a bag with the most valuable items available.

**Input:**

- ▶ Set of  $n$  items, with values  $v_i$  and sizes  $w_i$  (integer)
- ▶ Capacity  $W$

**Output:** subset  $S$  of items that maximizes the sum of values subject to a capacity constraint:

- ▶  $\max \sum_{i \in S} v_i$
- ▶ subject to  $\sum_{i \in S} w_i \leq W$

# Knapsack problem applications



Many problems with **budget constraints** are versions of knapsack

- ▶ Selecting portfolios (eg projects to invest in)

# Two-item example

## Items to pack

- ▶ **Shirt**: value 5, weight 5
- ▶ **Bottle**: value 10, weight 5

## Subsets:

- ▶  $\{\}, \{\text{Shirt}\}, \{\text{Bottle}\}, \{\text{Shirt}, \text{Bottle}\}$

## Knapsack size limits feasible solutions

- ▶  $W < 5$ : can pick neither
- ▶  $5 \leq W < 10$ : neither or just one
- ▶  $W \geq 10$ : all subsets feasible

# Go through all possibilities?

## Input:

- ▶ Set of  $n$  items, with values  $v_i$  and sizes  $w_i$  (integer)
- ▶ Capacity  $W$

**Output:** subset  $S$  of items that maximizes the sum of values subject to capacity constraint:

- ▶  $\max \sum_{i \in S} v_i$
- ▶ subject to  $\sum_{i \in S} w_i \leq W$

## Exhaustive (brute-force) search?

- ▶ Go through all subsets of  $\{1, 2, 3, \dots, n\}$
- ▶ Suppose we have 50 items:  $O(2^n)$  subsets...

# Greedy approaches for knapsack problem

## Input:

- ▶ Set of  $n$  items, with values  $v_i$  and sizes  $w_i$
- ▶ Capacity  $W$

**Output:** subset  $S$  of items that maximizes the sum of values subject to capacity constraint:

- ▶  $\max \sum_{i \in S} v_i$
- ▶ subject to  $\sum_{i \in S} w_i \leq W$

**Greedy approaches?** — Pick myopically without worrying about future choices

- ▶ **Highest-value** item first?
- ▶ **Lowest-size** item first?
- ▶ Some easy way of **combining value and size**?



# Greedy algorithms for knapsack

## Greedy approaches?

- ▶ **Highest-value** item first?
- ▶ **Lowest-size** item first?

**Example:** capacity  $W = 20$ , three item with values  $v = [10, 10, 11]$ , weights  $w = [10, 10, 20]$ .

- ▶ Picking highest value first is **bad**...

**Example:** capacity  $W = 20$ , three item with values  $v = [10, 10, 20]$ , weights  $w = [10, 10, 11]$ .

- ▶ Picking lowest weight first is **bad**...

Some easy way of **combining value and size**?

- ▶ Eg **sort items by unit weight**  $v_i/w_i$  and pick them in this order

# Greedy algorithm for knapsack

## Greedy algorithm:

- ▶ Sort items in decreasing  $v_i/w_i$
- ▶ Pick items until capacity full

## Running time?

- ▶ Sorting?
- ▶ Loop?
- ▶ Total  $O(n \log n)$

# Is the algorithm correct?

It would be if we assumed that we **can divide items into fractions**

- ▶ But we cannot...

**Example:** capacity  $W=510$ , three items with values  $v = [10, 10, 500]$ , weights  $w = [10, 10, 501]$ .

- ▶ Greedy picking is **bad**...

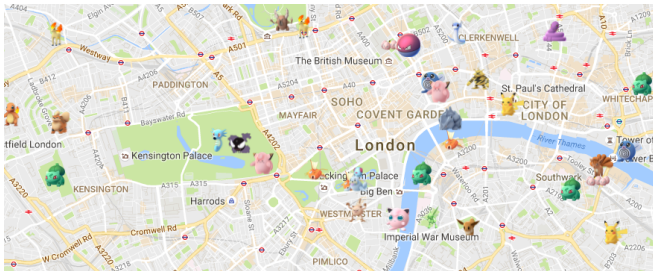
We would need a different approach to find correct solution — dynamic programming

- ▶ Exploit problem structure to loop through all items and possible capacities
- ▶ Correct solution,  $O(nW)$  time (“pseudo-polynomial”)

# Many important problems are intractable

**Tractable problem** = Solvable in polynomial time  $O(n^k)$  for some  $k$

Finding the **shortest route to a Pikachu nest** (Dijkstra) vs. finding the **shortest tour of all Pokemon nests**



Example of intractability: **traveling salesman problem** (TSP)

- ▶ **Input:** undirected graph with non-negative edge costs
- ▶ **Goal:** find minimum cost tour visiting every node
- ▶ **Conjecture:** no polynomial-time algorithm [**sidebar: P vs NP**]
- ▶ Many other important problems too...

# My problem is intractable!

## What can you do?

1. There may be **tractable special cases** (small knapsack DP)
2. Get an **approximate solution using heuristics** — fast but not “correct” (next slide)
3. Solve in exponential time but try to **improve on brute force** (large knapsack DP)

# Greedy knapsack heuristic

Greedy knapsack was **incorrect** but very fast:  $O(n \log n)$

## Greedy algorithm:

- ▶ Sort items in decreasing bang-for-buck  $v_i/w_i$
- ▶ For each item, pick the item until reach total  $W$

## How bad is it?

- ▶ **Example:** capacity  $W=510$ , three items with values  $v = [10, 10, 500]$ , weights  $w = [10, 10, 501]$ .
- ▶ “Worst-case scenario”: leave out (a single) extremely valuable object that would fit into knapsack
- ▶ **But often items are small**  $\rightarrow$  the greedy choice cannot leave out many of them  $\rightarrow$  the heuristic will be much better

# MY HOBBY:

## EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT	
~ APPETIZERS ~	
MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80
~ SANDWICHES ~	
BARBECUE	6.55



Pic: xkcd

Paris  
Where to  
eat for  
under €10  
travel →

Morgan  
Robinson  
TV's top  
mimic  
guide →

America  
and the gun  
by Gary  
Younge

Weekend →  
Katie Couric  
and the new  
TV season  
page 32 →

New exclusive collection  
of 20 simple recipes

Free  
magazine

# Easy Ottolenghi

Plus how  
to survive  
a weekend  
family →

**the guardian**

£1.00 for  
mainstream  
page 32 →

## One in seven takeaways fail hygiene tests

Guardian analysis also reveals one in 13  
restaurants fall short of basic standards

Kate Coppen and Rebecca Dargatzis

More than one in seven takeaway restaurants in the UK fail to meet basic hygiene standards, according to a new analysis by the Guardian. The study, which looked at data from the Food Hygiene Inspection (FHI) database, found that 14.5% of takeaway restaurants had failed their last inspection. This is a significant increase on the 10.5% failure rate recorded in 2014. The analysis also found that one in 13 restaurants, or 7.7%, had failed to meet the minimum standards for food safety and hygiene. This is a worrying trend, given that the food industry is a major source of foodborne illness in the UK. The study was carried out by the Guardian's food and drink team, in partnership with the Food Hygiene Inspection database. The database contains information on all food businesses in the UK that are subject to food hygiene inspections. The data was analysed using a custom-built software program that identified patterns in the data. The results show that the most common reasons for failure were poor hygiene practices, such as not washing hands properly, and not using clean equipment. The study also found that restaurants that had failed their last inspection were more likely to fail again in the next inspection. This suggests that there is a need for more rigorous enforcement of food hygiene standards in the takeaway sector.

The study also found that restaurants that had failed their last inspection were more likely to fail again in the next inspection. This suggests that there is a need for more rigorous enforcement of food hygiene standards in the takeaway sector. The study was carried out by the Guardian's food and drink team, in partnership with the Food Hygiene Inspection database. The database contains information on all food businesses in the UK that are subject to food hygiene inspections. The data was analysed using a custom-built software program that identified patterns in the data. The results show that the most common reasons for failure were poor hygiene practices, such as not washing hands properly, and not using clean equipment. The study also found that restaurants that had failed their last inspection were more likely to fail again in the next inspection. This suggests that there is a need for more rigorous enforcement of food hygiene standards in the takeaway sector.

Is this the most powerful woman in fashion?



Warning that damage to Labour could be 'terminal'

Labour's leading candidate, Ed Miliband, has warned that the party's current position is "terminal" and that it must make a radical break with the past if it is to have any chance of winning the next general election. Miliband, who is the current leader of the Labour Party, made the statement during a speech at a conference in London. He said that the party had lost touch with the people and that it was time to "reconnect" with them. He called for a "new direction" for the party, one that would focus on the needs of the working class and the poor. Miliband's speech was seen as a warning shot to the party's leadership, who have been accused of being out of touch with the electorate. The speech also came at a time when Labour's support in the polls is at its lowest in decades. This suggests that the party is in a difficult position and that it may need to make significant changes if it is to survive.

**Be coldhearted.**  
Remember Greenpeace in your Will and  
help keep the Arctic as it should be. Cold.  
Find out how to make space for a polar bear  
or two in your Will.  
[www.greenpeace.org.uk/polarbears](http://www.greenpeace.org.uk/polarbears)  
**GREENPEACE**



[https://www.theguardian.com/world/2016/sep/23/  
revealed-uk-takeaways-fail-food-hygiene-tests-restaurants-takeaways](https://www.theguardian.com/world/2016/sep/23/revealed-uk-takeaways-fail-food-hygiene-tests-restaurants-takeaways)



# Workshop

**After the break...**

**Knapsack problem and fantasy football**

**Extra: Python for your safety — South Ken food hygiene**

**Algorithms, Python, and your future**