DMAI

Lecture 2

Sebastian Wandelt

Beihang University

Outline

- Recap: Graphs
- Weighted graph algorithms:
  – Minimum spanning trees
- Greedy algorithms
  – Knapsack

Weighted Graphs
Finding shortest paths

Dijkstra’s algorithm

$d[s] = 0$
for each $v \in V - \{s\}$
  do $d[v] = \infty$
$S = \varnothing$
$Q = V$ ⊳ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
  $u = $ Extract-Min(Q)
  $S = S \cup \{u\}$
  for each $v \in Adj[u]$
   if $d[v] > d[u] + w(u, v)$
    then
    $d[v] = d[u] + w(u, v)$

Time complexity?

- What is the time complexity of Dijkstra, given a graph with |N| nodes and |E| links?

Time complexity?

- $O(|N|+|E|)$
- Every link in the graph is only followed one time

Summary: Greedy-property

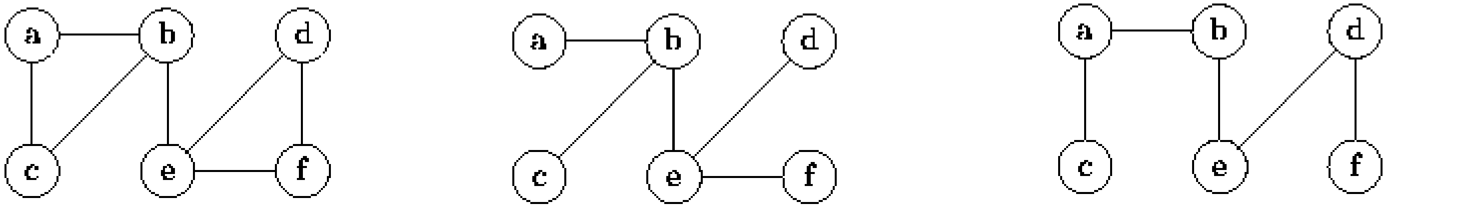
- Given a weighted directed graph, we can find the shortest distance between two vertices by:
  - starting with a trivial path containing the initial vertex
  - growing this path by always going to the next vertex which has the shortest current distance to the initial vertex

- Such a strategy is called **greedy** in Computer Science
  - Greedily-chosen, locally best solutions might lead to globally best solutions
  - One can prove that greedy is an optimal strategy for shortest path computation with non-negative link weights
- We will look at another greedy algorithm next!

Weighted Graphs
Finding minimum spanning trees

Spanning Tree

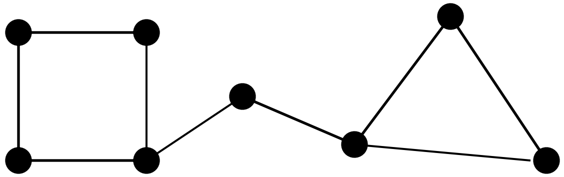
- A **spanning tree** for a graph is a subgraph that includes every vertex of the original, and is a tree.



(a) Graph G (b) One spanning tree (c) Another spanning tree

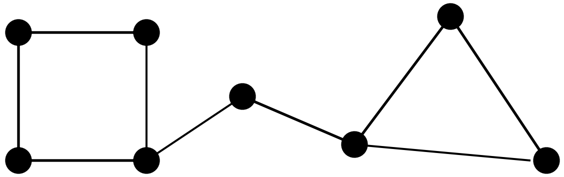
Finding a Spanning Tree

Find a spanning tree for the graph below.

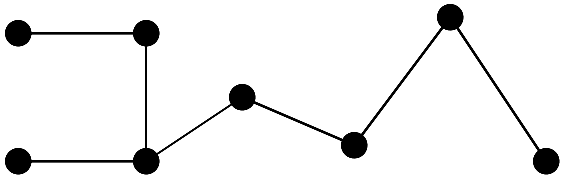


Finding a Spanning Tree

Find a spanning tree for the graph below.



We could break the two cycles by removing a single edge from each. One of several possible ways to do this is shown below.



Minimum Spanning Tree

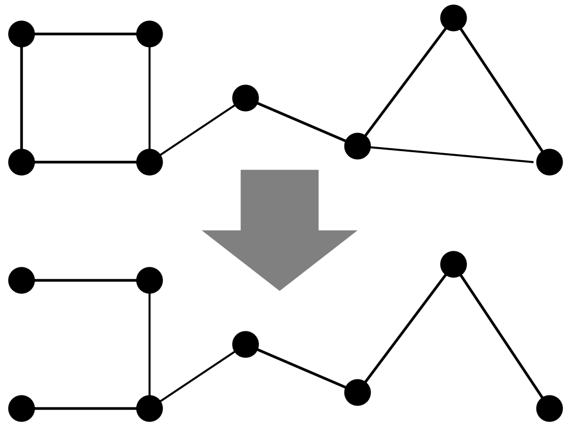
- A spanning tree that has minimum total weight is called a **minimum spanning tree** for the graph.
  - Technically it is a minimum-weight spanning tree.
- If all edges have the same weight … what can we do to obtain a minimum spanning tree?

Minimum Spanning Tree

- A spanning tree that has minimum total weight is called a **minimum spanning tree** for the graph.
  - Technically it is a minimum-weight spanning tree.
- If all edges have the same weight, breadth-first search or depth-first search will yield minimum spanning trees.

Minimum Spanning Tree

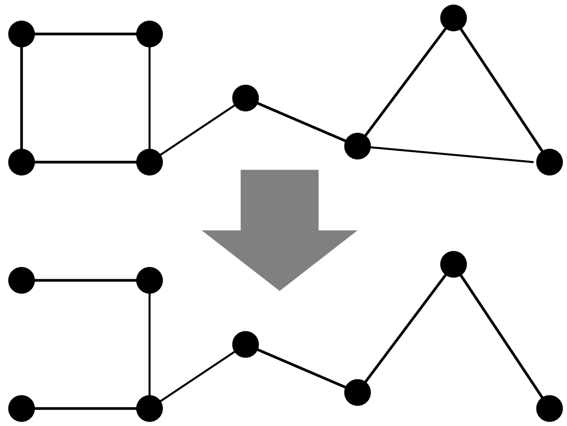
- A spanning tree that has minimum total weight is called a **minimum spanning tree** for the graph.
  - Technically it is a minimum-weight spanning tree.
- If all edges have the same weight, breadth-first search or depth-first search will yield minimum spanning trees.



Created by BFS or DFS?

Minimum Spanning Tree

- A spanning tree that has minimum total weight is called a **minimum spanning tree** for the graph.
  - Technically it is a minimum-weight spanning tree.
- If all edges have the same weight, breadth-first search or depth-first search will yield minimum spanning trees.
  - For the rest of this discussion, we assume the edges have weights associated with them.



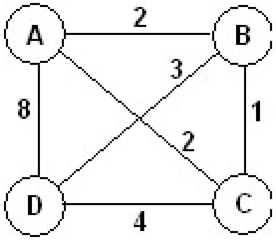
Created by BFS or DFS?

Minimum Spanning Tree

- Minimum-cost spanning trees have many applications.
  - Building cable networks that join *n* locations with minimum cost.
  - Building a road network that joins *n* cities with minimum cost.
  - Obtaining an independent set of circuit equations for an electrical network.
  - In pattern recognition minimal spanning trees can be used to find noisy pixels.

Minimum Spanning Tree

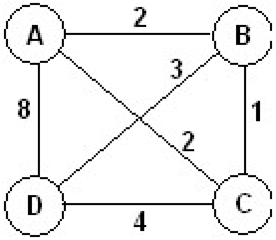
- Consider this graph.



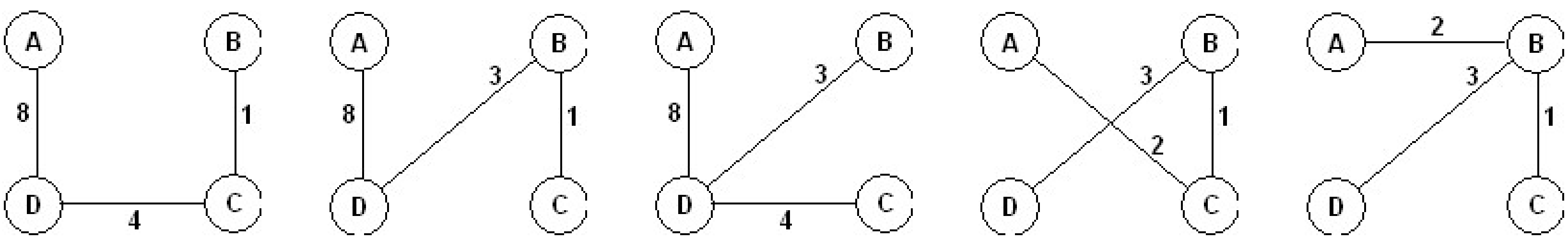
- How many spanning trees can you find?
- Which ones are minimum spanning trees?

Minimum Spanning Tree

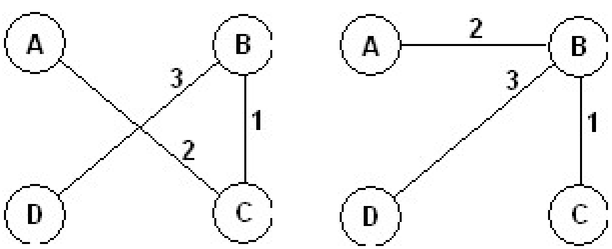
- Consider this graph.



- Some spanning trees are:



- There are two minimum-cost spanning trees, each with a cost of 6:



Minimum Spanning Tree

- Brute Force option:
  1. For all possible spanning trees
     i. Calculate the sum of the edge weights
     ii. Keep track of the tree with the minimum weight.
- Step i) requires N-1 time, since each tree will have exactly N-1 edges.
- If there are M spanning trees, then the total cost will O(MN).
- Consider a complete graph, with N(N-1) edges. How big can M be?

Brute Force MST

- For a complete graph, it has been shown that there exist N^{N-2} possible spanning trees!
- Alternatively, given N items, you can build N^{N-2} distinct trees to connect these items.

Minimum Spanning Tree

- There are many approaches to computing a minimum spanning tree. We could try to detect cycles and remove edges, but the two algorithms we will study build them from the bottom-up in a *greedy* fashion.
- **Kruskal's Algorithm – *starts with a forest of single node trees*** and then adds the edge with the minimum weight to connect two components.
- **(Prim's Algorithm – *starts with a single vertex*** and then adds the minimum edge to extend the spanning tree.)
  - Not covered here, but quite easy to understand

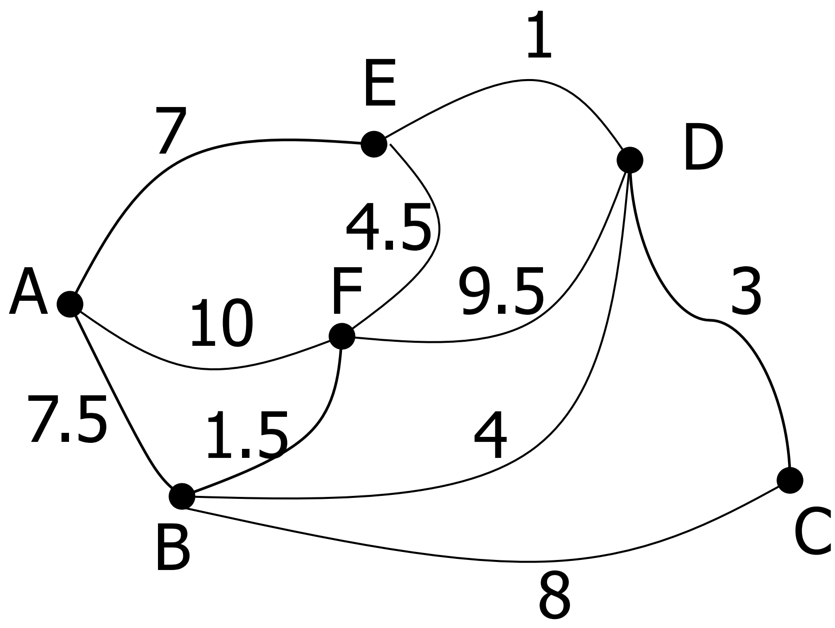
Kruskal’s Algorithm

- Greedy algorithm to choose the edges as follows.

Step 1	**First edge: choose any edge with the minimum weight.**
Step 2	**Next edge: choose any edge with minimum weight from *those not yet selected*. (The subgraph can be disconnected at this stage.)**
Step 3	**Continue to choose edges of minimum weight from those not yet selected, except *do not select any edge that creates a cycle* in the subgraph.**
Step 4	**Repeat step 3 until the subgraph connects all vertices of the original graph.**

Kruskal’s Algorithm

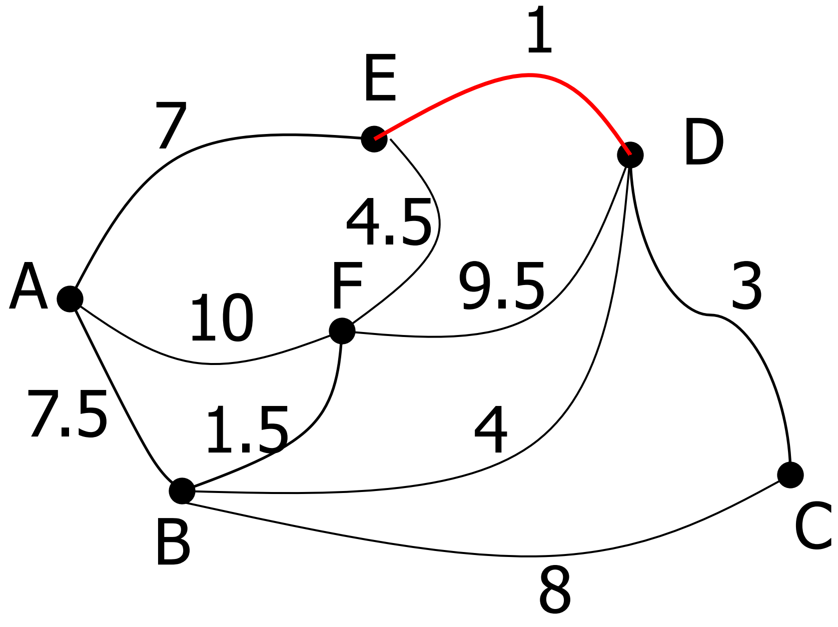
Use Kruskal’s algorithm to find a minimum spanning tree for the graph.



Kruskal's Algorithm

Solution

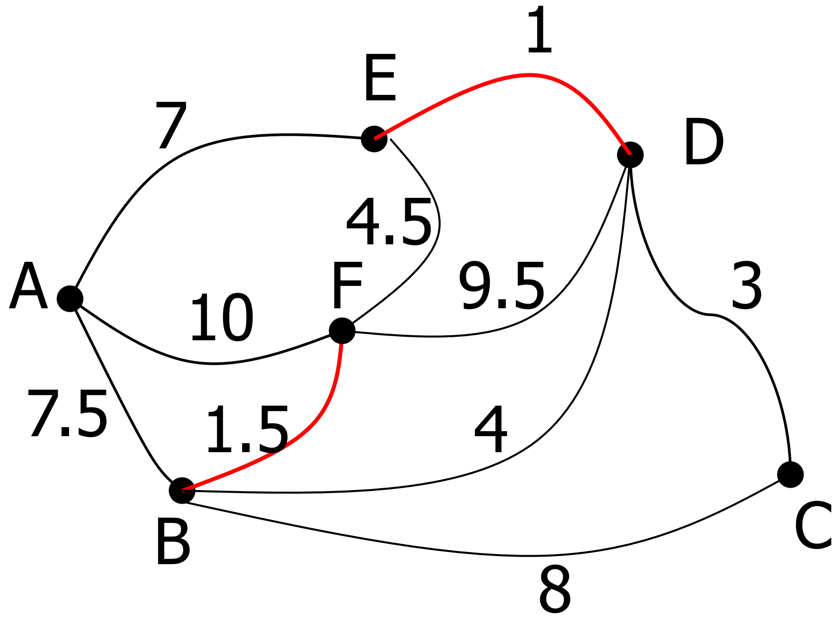
First, choose ED (the smallest weight).



Kruskal's Algorithm

Solution

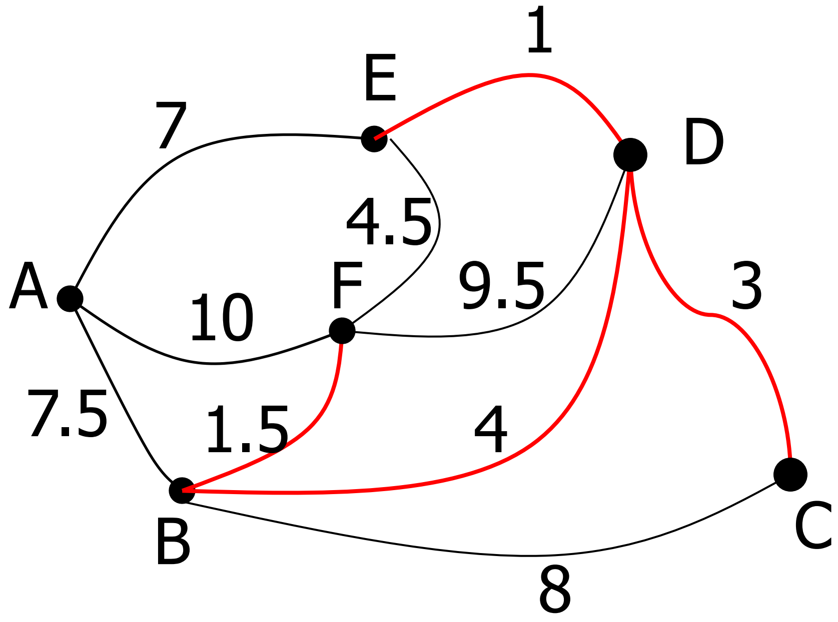
Now choose BF (the smallest remaining weight).



Kruskal's Algorithm

Solution

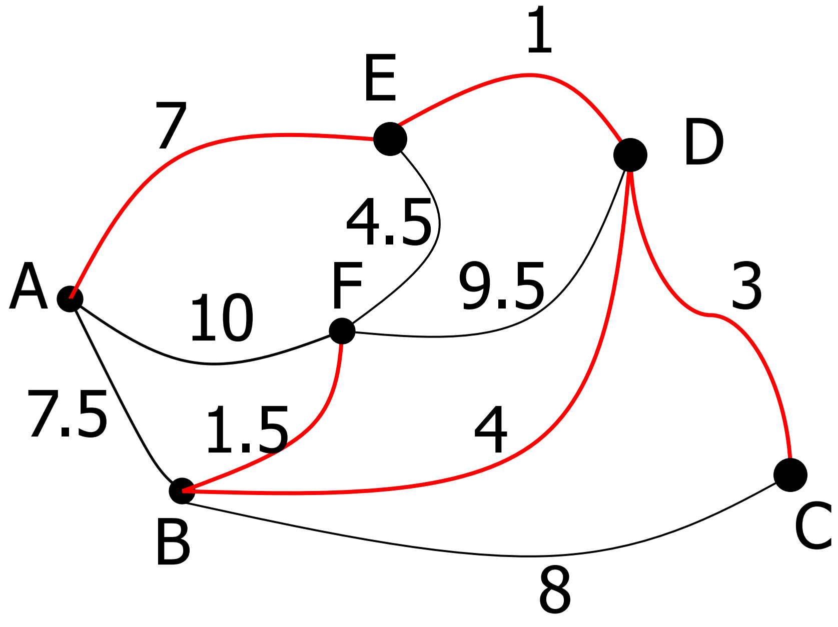
Now CD and then BD.



Kruskal's Algorithm

Solution

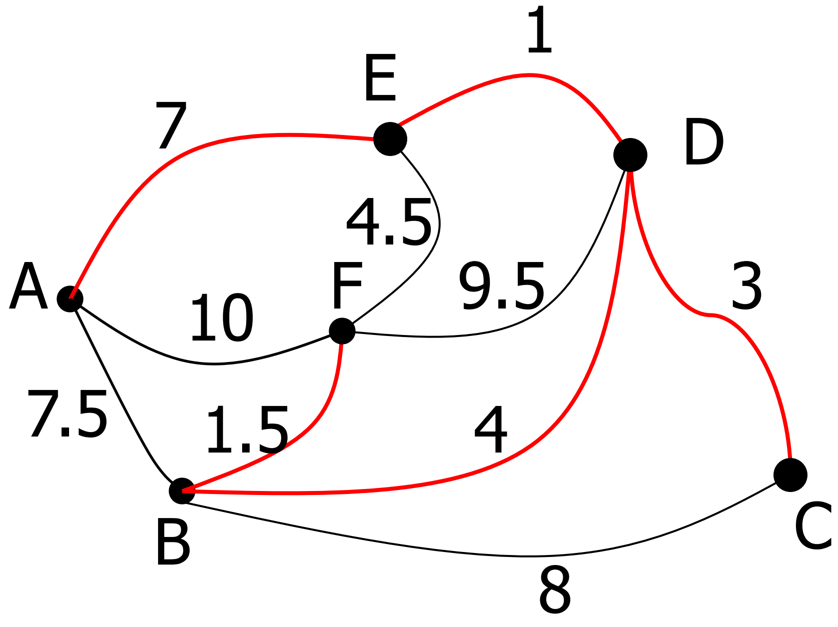
Note EF is the smallest remaining, but that would create a cycle. Choose AE and we are done.



Kruskal's Algorithm

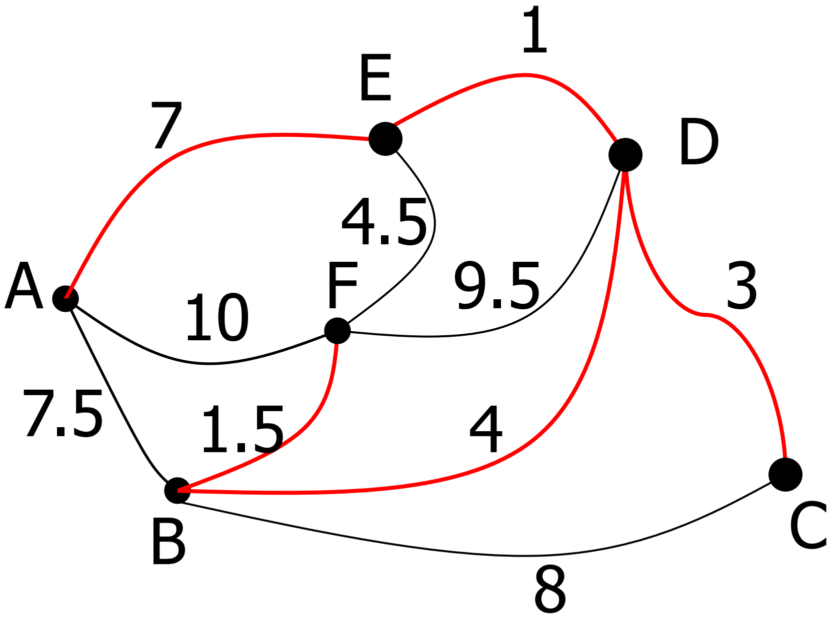
Solution

The total weight of the tree is 16.5.



Kruskal's Algorithm

- Some questions:
  1. How do we know we are finished?
  2. How do we check for cycles?



Kruskal’s Algorithm

```
Build a priority queue (min-based) with all of the edges of G.
T = ϕ ;
while (queue is not empty)
   get minimum edge e from priorityQueue
   if(e does not create a cycle with edges in T)
       add e to T

return T
```

Open problem

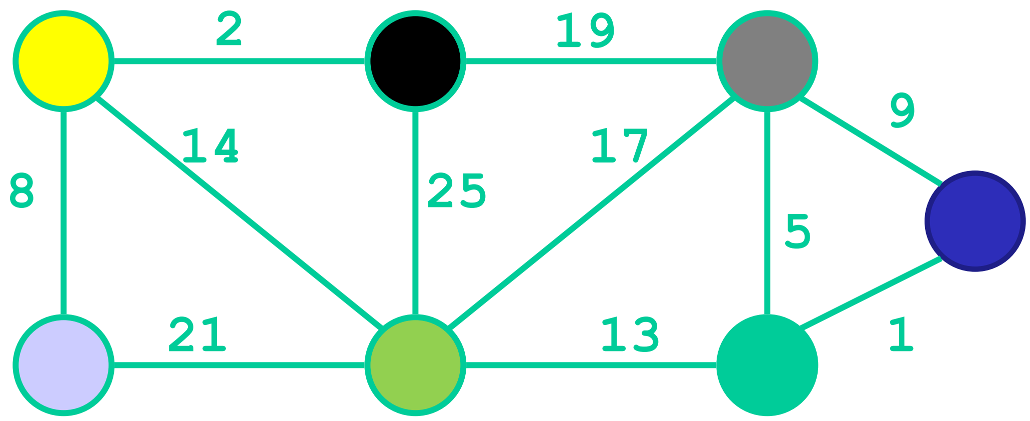
- How to efficiently check for existence of cycles?

Kruskal's Algorithm (avoiding cycles)

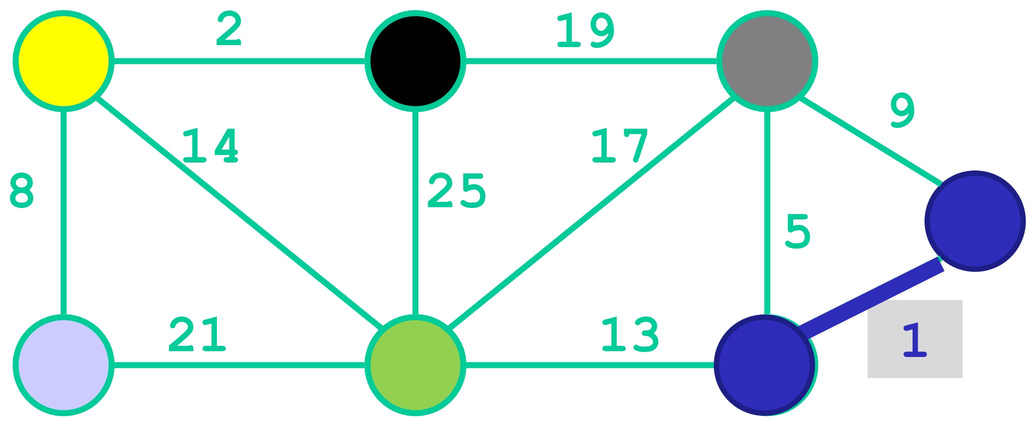
- An implementation based on sets

```
def Kruskal()
    T = ∅
    for each v ∈ V
        MakeSet(v)
    sort E by increasing edge weight w
    for each (u,v) ∈ E (in sorted order)
        if FindSet(u) ≠ FindSet(v)
            T = T U {{u,v}}
            Union(FindSet(u), FindSet(v))
```

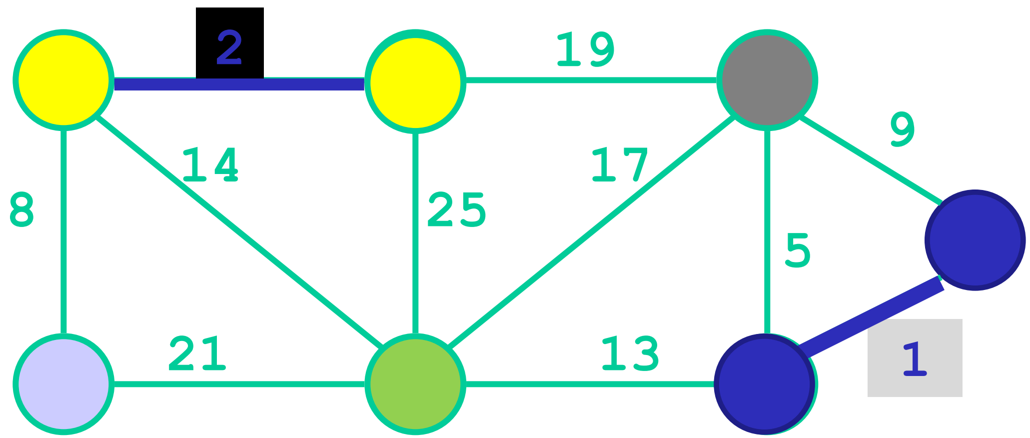
Kruskal’s Algorithm



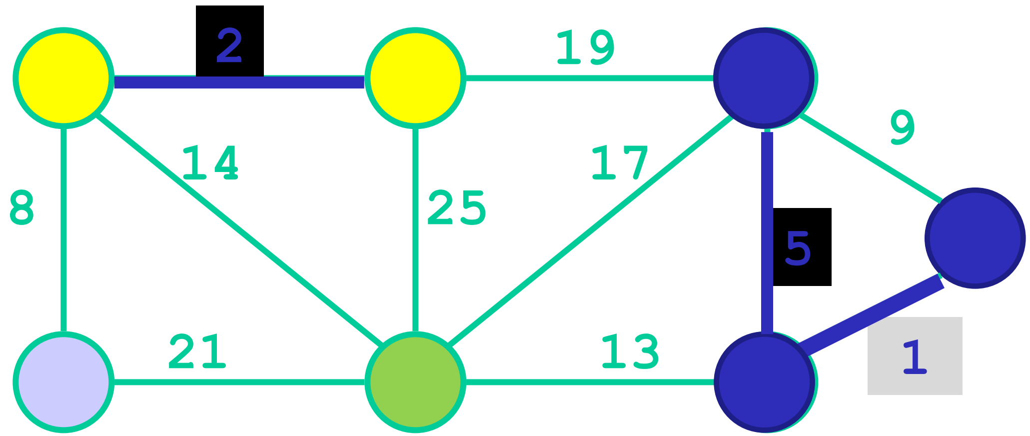
Kruskal's Algorithm



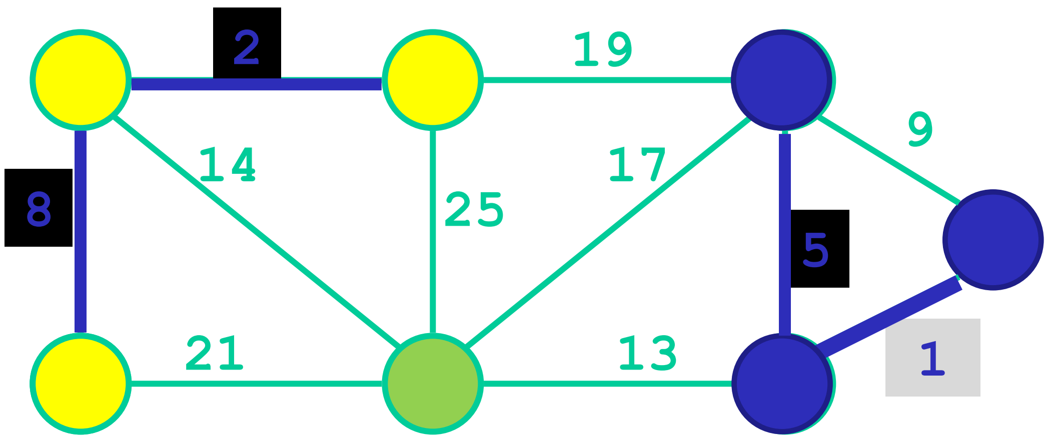
Kruskal’s Algorithm



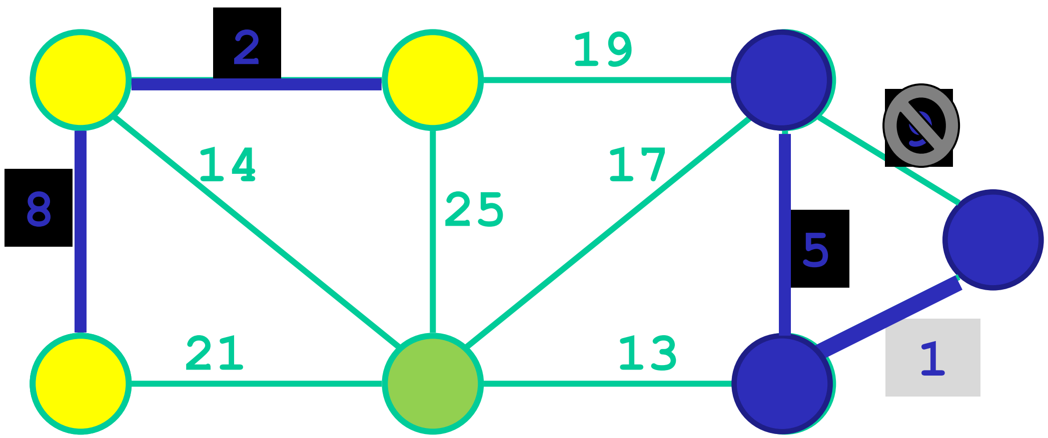
Kruskal’s Algorithm



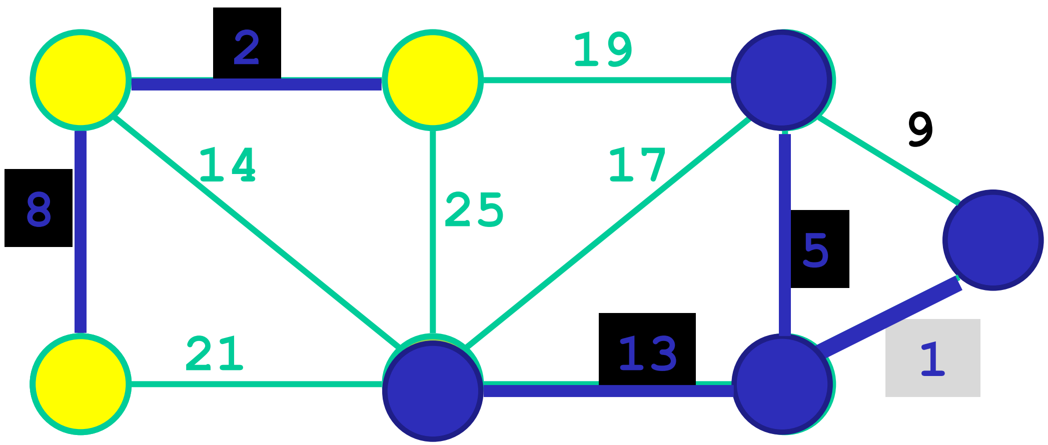
Kruskal’s Algorithm



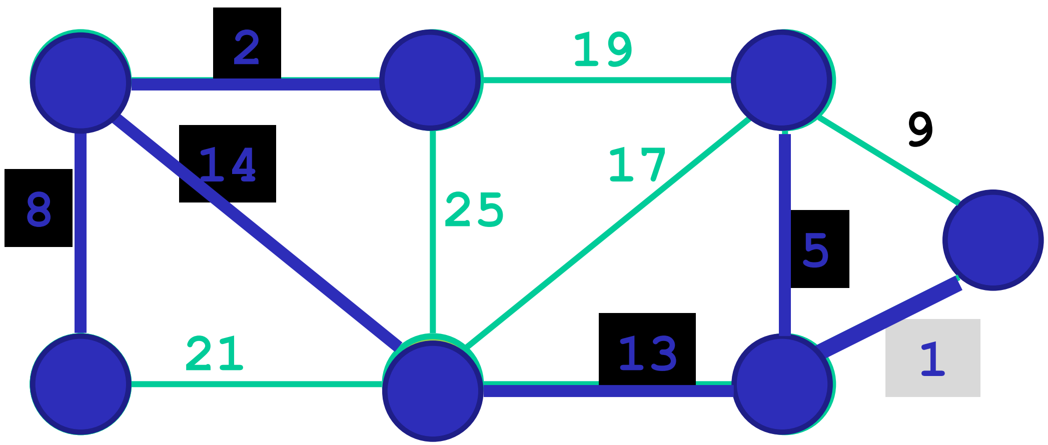
Kruskal’s Algorithm



Kruskal's Algorithm



Kruskal’s Algorithm



Time complexity (without proof)

- O(E log E) or O(E log V)
- Are these two really different?

Time complexity (without proof)

- O(E log E) or O(E log V)
- Are these two really different?
  - No:
    - E is at most V*V
    - log E <=log V*V=2*log V=O(log V)
- Such a low time complexity is quite surprising
  - Achieved by greedy property

Greedy algorithms

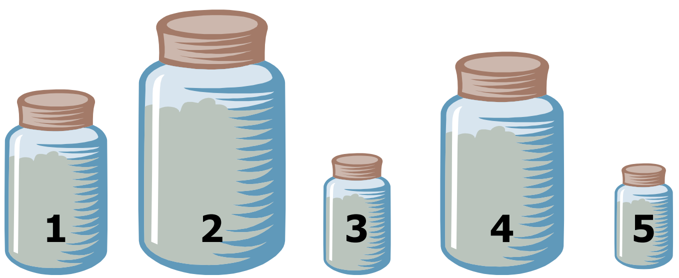
Greedy algorithms

- Greedy algorithms make best choices locally
  - Shortest path
  - Minimum spanning tree
- Does this always work?

Knapsack problem (fractional)

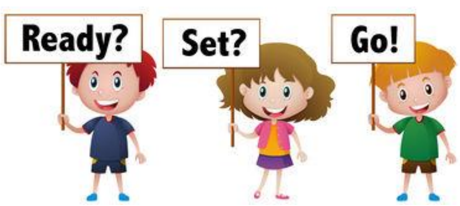
- Given: A set S of n items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.

Items:



Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit:	\$12	\$32	\$40	\$30	\$50
Value: ($ per ml)	3	4	20	5	50

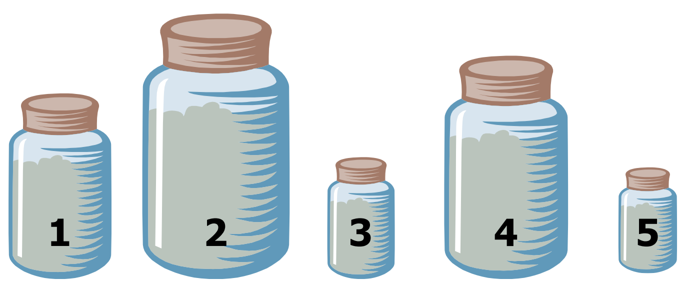
10 ml



Example (fractional)

- Given: A set S of n items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.

Items:



Items:	1	2	3	4	5
Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit:	\$12	\$32	\$40	\$30	\$50
Value: (\$ per ml)	3	4	20	5	50

“knapsack”

10 ml

Solution:
- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

The 0-1 Knapsack problem

- We cannot take away fractions of items!
  - Does a greedy strategy still work?
  - Can you find a counter example?

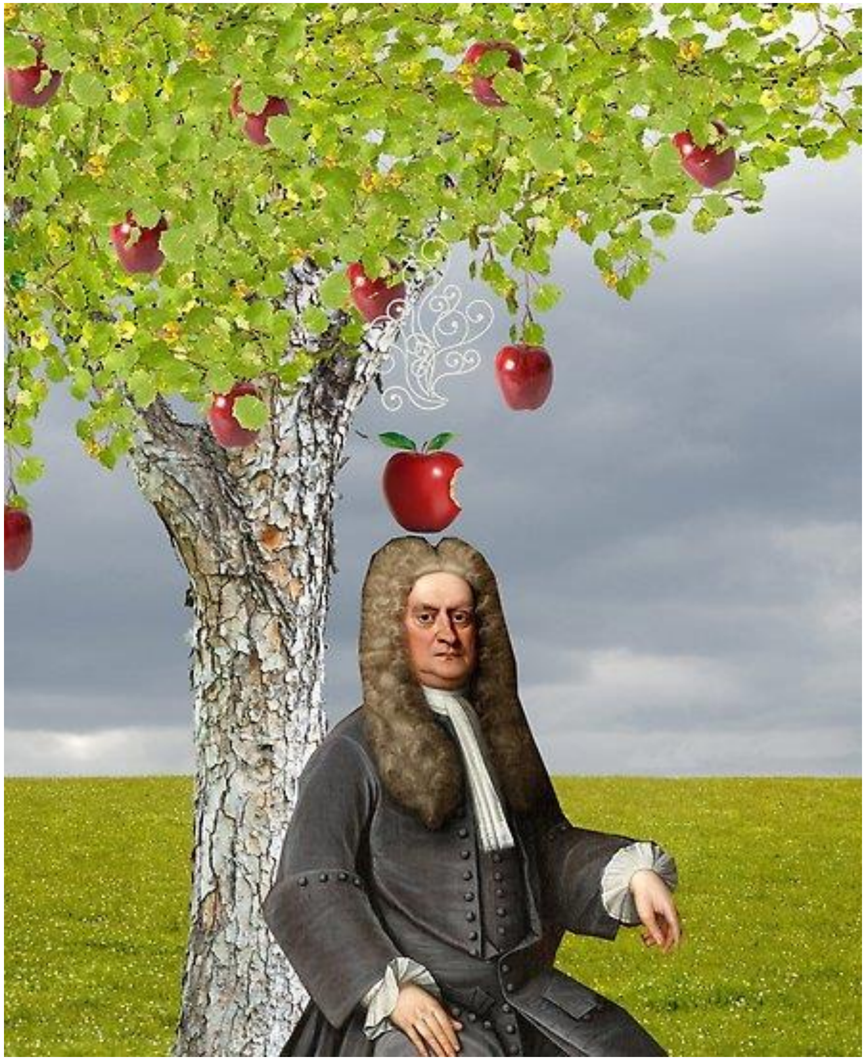
Counter example

- 0-1 Knapsack with capacity 4
- Item 1: w=3, b=2, v=0.66
- Item 2: w=2, b=1.1, v=0.55
- Item 3: w=2, b=1.1, v=0.55

Data Management and Artificial Intelligence

A motivating example

Engineering involves experiments



- Apple?
- Tree?
- Any ideas? ☺

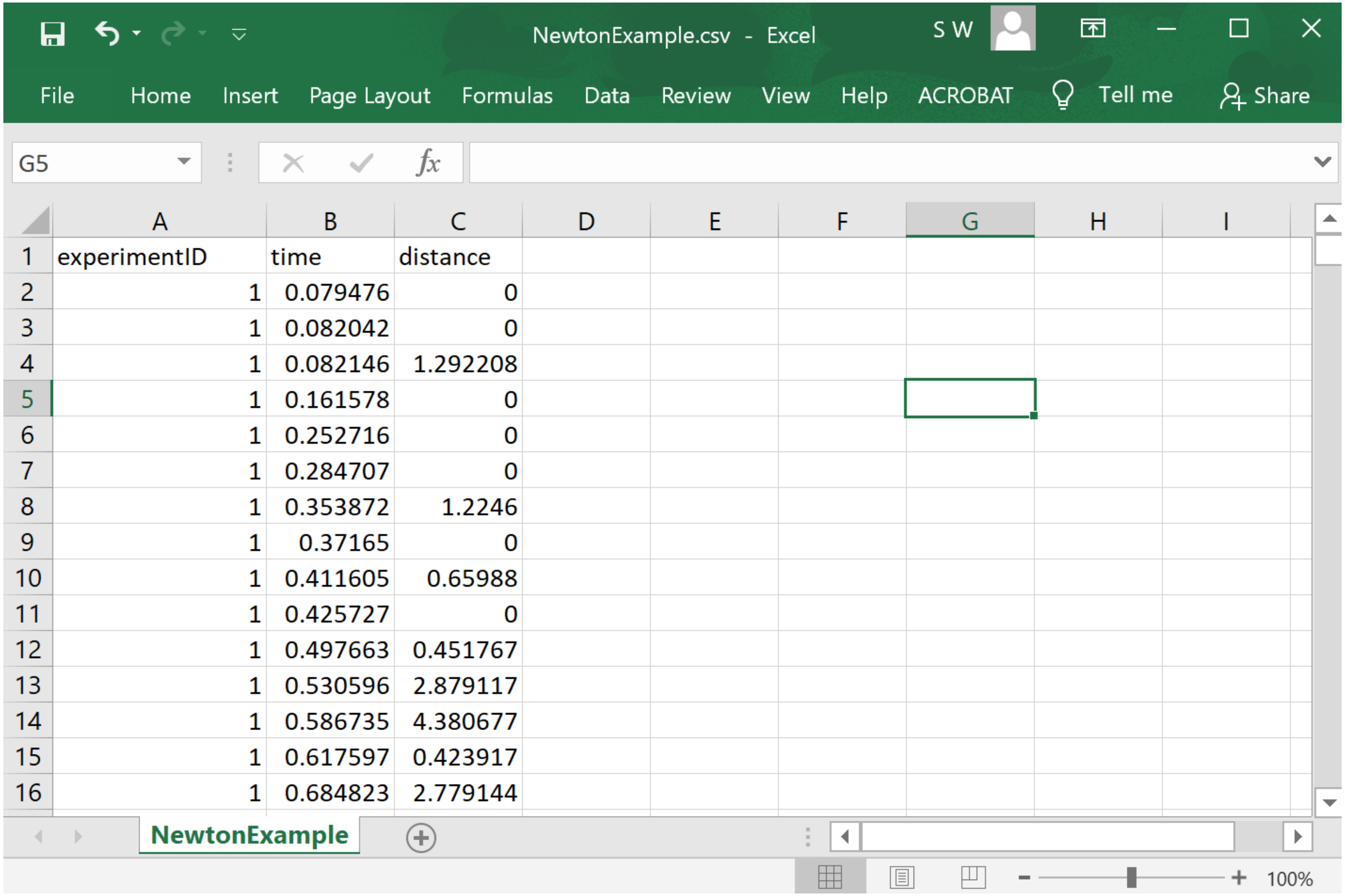
Small experiment

- Let us try to find the (simplest) equation for free falling bodies:

$$d = \frac{1}{2}gt^2$$

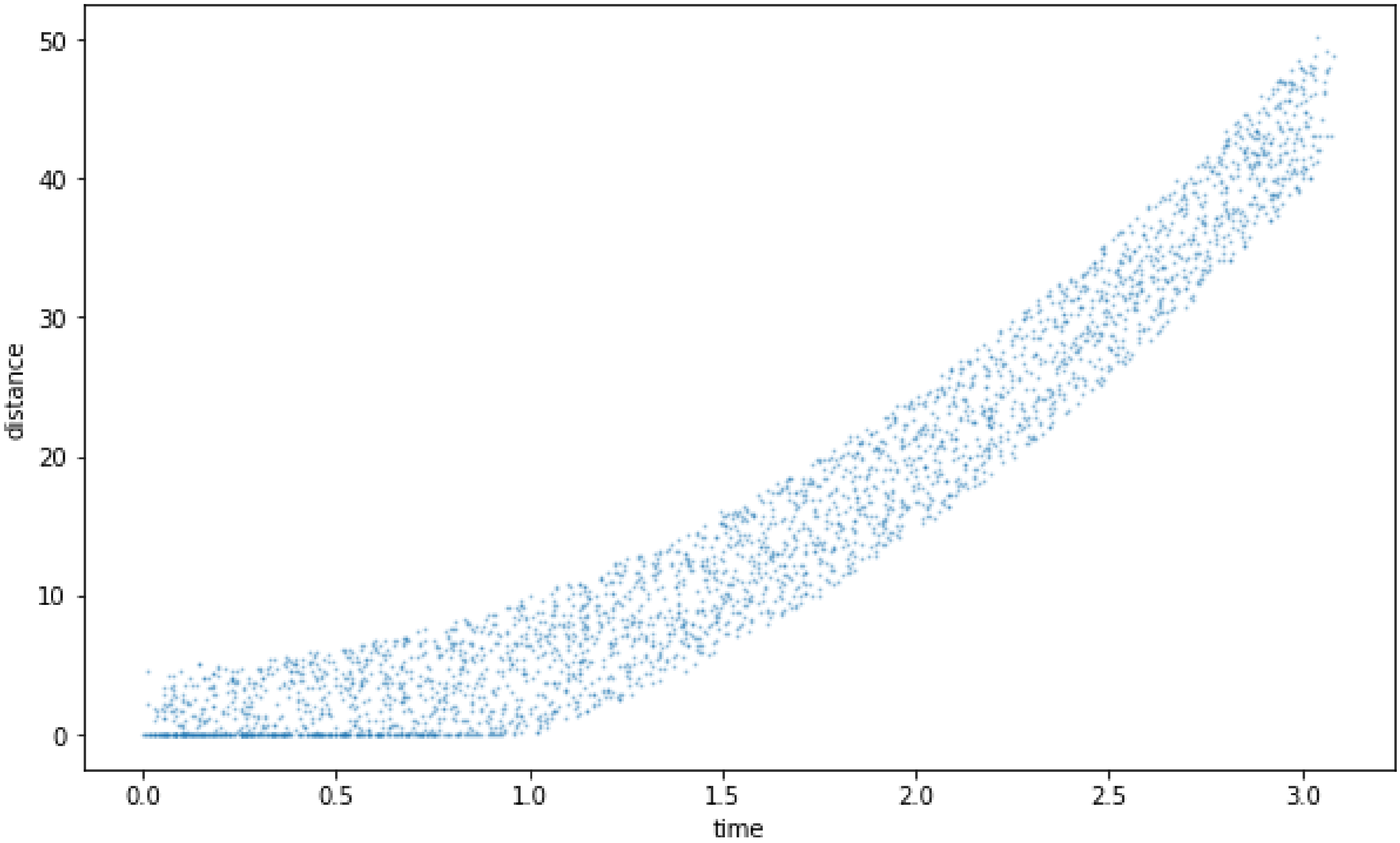
- Idea:
  - Do an experiment with an apple falling from a tree
  - Record the time and distance traveled by the apple
  - Repeat the experiments 50 times
  - Derive a good formula for describing the relationship between time and distance

What does the data look like?



	A	B	C
1	experimentID	time	distance
2	1	0.079476	0
3	1	0.082042	0
4	1	0.082146	1.292208
5	1	0.161578	0
6	1	0.252716	0
7	1	0.284707	0
8	1	0.353872	1.2246
9	1	0.37165	0
10	1	0.411605	0.65988
11	1	0.425727	0
12	1	0.497663	0.451767
13	1	0.530596	2.879117
14	1	0.586735	4.380677
15	1	0.617597	0.423917
16	1	0.684823	2.779144

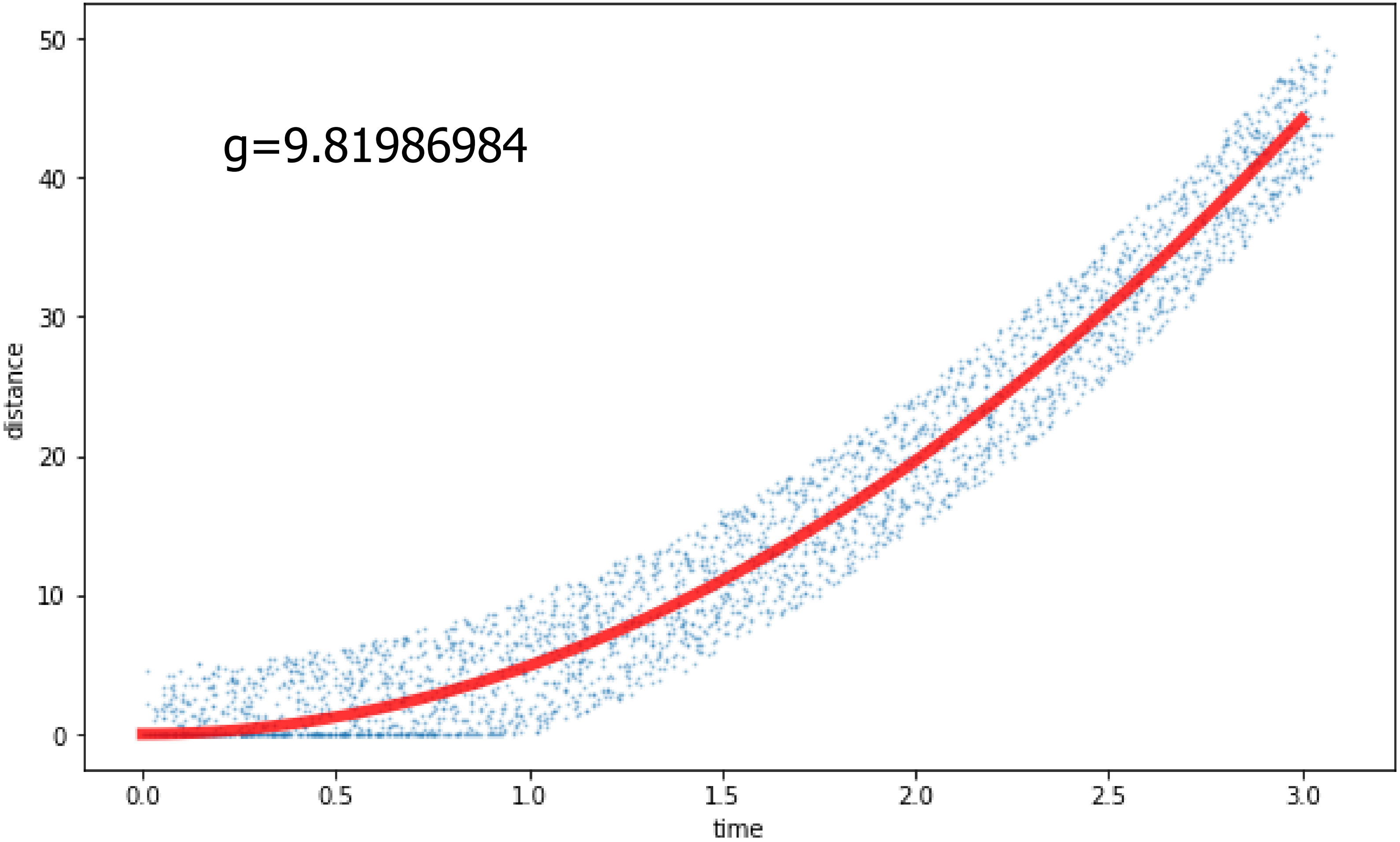
What does the data look like?



What does the code look like?

```
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt

def f(x, c):
    return c*x**2

# Load data
df=pd.read_csv("NewtonExample.csv")

# Plot data
fig,ax=plt.subplots(1,1,figsize=(10,6))
plt.scatter(df["time"],df["distance"],s=1,alpha=0.3)
plt.xlabel("time")
plt.ylabel("distance")

# Compute fit
copt,pcov=scipy.optimize.curve_fit(f, df["time"], df["distance"])
print(copt*2)

# Visualize fit
x=np.linspace(0,3,100)
plt.plot(x,f(x,copt),"r-",linewidth=5,alpha=0.8)
```

What does the data look like (with fit)?



What does the code look like?

```python
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt

def f(x, c):
    return c*x**2

# Load data
df=pd.read_csv("NewtonExample.csv")

# Plot data
fig,ax=plt.subplots(1,1,figsize=(10,6))
plt.scatter(df["time"],df["distance"],s=1,alpha=0.3)
plt.xlabel("time")
plt.ylabel("distance")

# Compute fit
copt,pcov=scipy.optimize.curve_fit(f, df["time"], df["distance"])
print(copt*2)

# Visualize fit
x=np.linspace(0,3,100)
plt.plot(x,f(x,copt),"r-",linewidth=5,alpha=0.8)
```

Data

+

Visualization

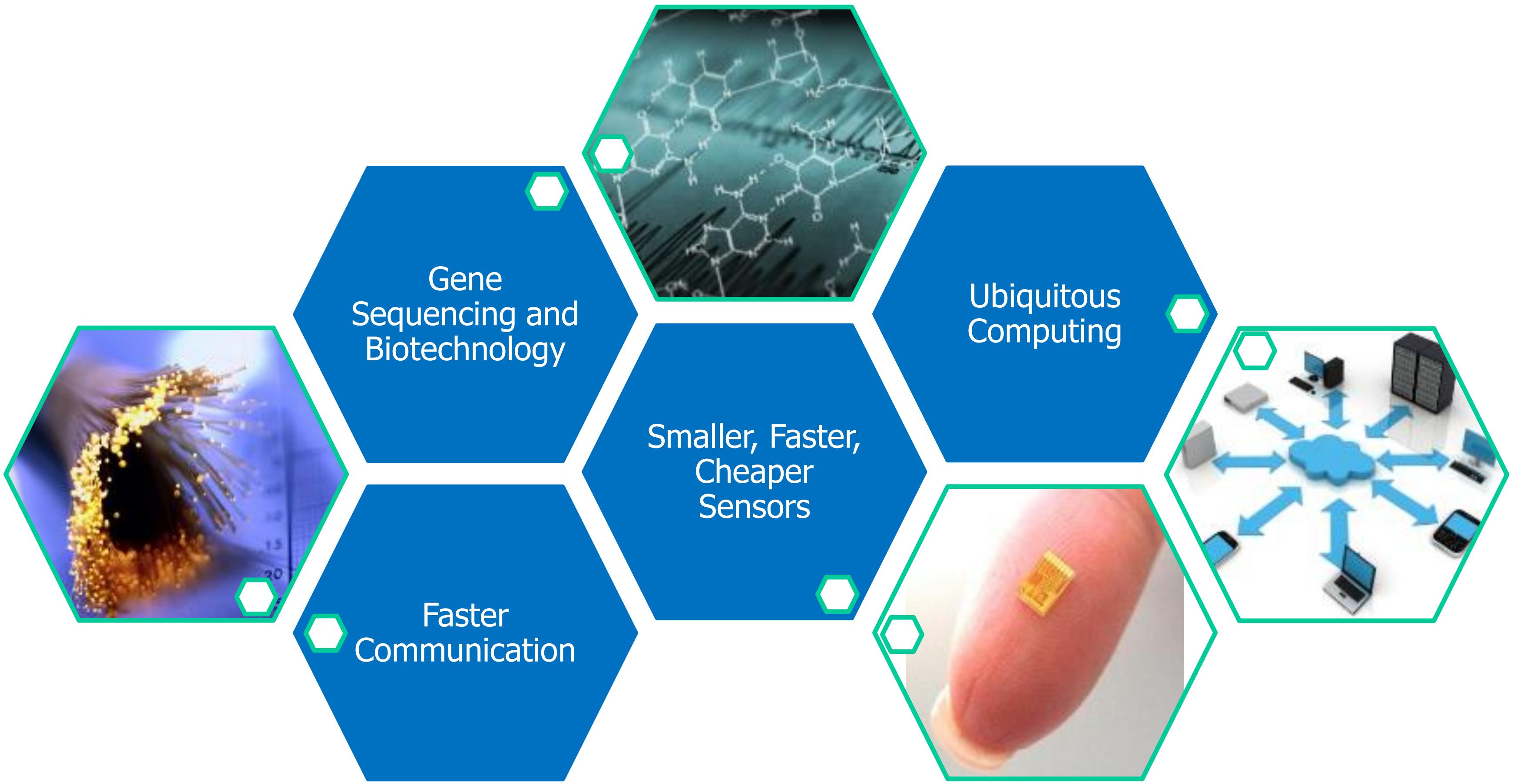
+

Intelligence

This is data management and AI at one of its simplest forms!

What is data management?

Some breakthroughs in the 21th century

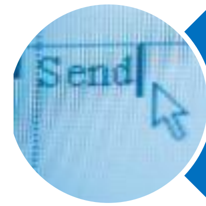


We live in a world of data

- 2.9 million emails are sent every second
- 20 hours of video are uploaded to YouTube every minute
- 24 PBs of data are processed by Google every day
- 50 million tweets are generated per day
- 700 billion total minutes are spent on Facebook each month
- 72.9 items are ordered on Amazon every second
- Nearly 500 Exabytes per day are generated by the Large Hadron Collider experiments (even not all recorded!)

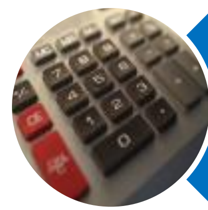
Activities related to data management

Store



Share

Query



Mine

Encrypt

.... and more!

We want to do these *seamlessly* and *fast*...

Why should you study data management?

- Data is *everywhere* and is *critical* to our lives
- Data need to be recorded, maintained, accessed and manipulated *correctly*, *securely*, *efficiently* and *effectively*
- Database management systems (DBMSs) are indispensable software for achieving such goals
- The principles and practices of DBMSs are now an integral part of computer science curricula
  - They encompass OS, languages, theory, AI, multimedia, and logic, among others

As such, the study of database systems can prove to be richly rewarding in more ways than one!

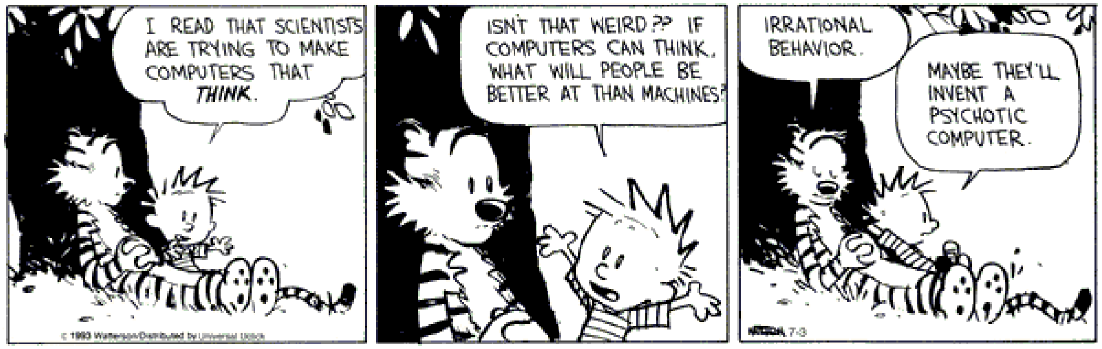
What is artificial intelligence?

What is artificial intelligence (AI)?

- Popular conception driven by science fiction
  - Robots good at everything except emotions, empathy, appreciation of art, culture, …
    - … until later in the movie.
- Current AI is also bad at lots of simpler stuff!
- There is a lot of AI work on thinking about what other agents are thinking

Real Artificial Intelligence

- General-purpose AI like the robots of science fiction is incredibly hard
  - Human brain appears to have lots of special and general functions, integrated in some amazing way that we really do not understand (yet)
- Special-purpose AI is more doable (nontrivial)
  - E.g., chess/poker/Go playing programs, logistics planning, automated translation, speech and image recognition, web search, data mining, medical diagnosis, keeping a car on the road, …



Simple engineering example

- Assume we want to build a vehicle which can drive on the moon (or any other rough surface)
- How would such a car look like?
  - It took humans years to design such a Rover.
- How can a computer design such a car (automatically)?
- Nice webpage:
  - http://boxcar2d.com/index.html

Let's play a little bit and see whether we are lucky!

BoxCar 2D

Home | Designer | Best Cars | Forum | News | FAQ | The Algorithm | Versions | Contact

Computation Intelligence Car Evolution Using Box2D Physics (v3.2)

58 fps average
Physics step: 0 ms (Infinity fps
22 MB used

Hide

Input Seed / Choose Terrain

Generation: 0 Max Score: 100

100

50

Copy All Copy Selected

Car	Score	Time
0	100	0:08

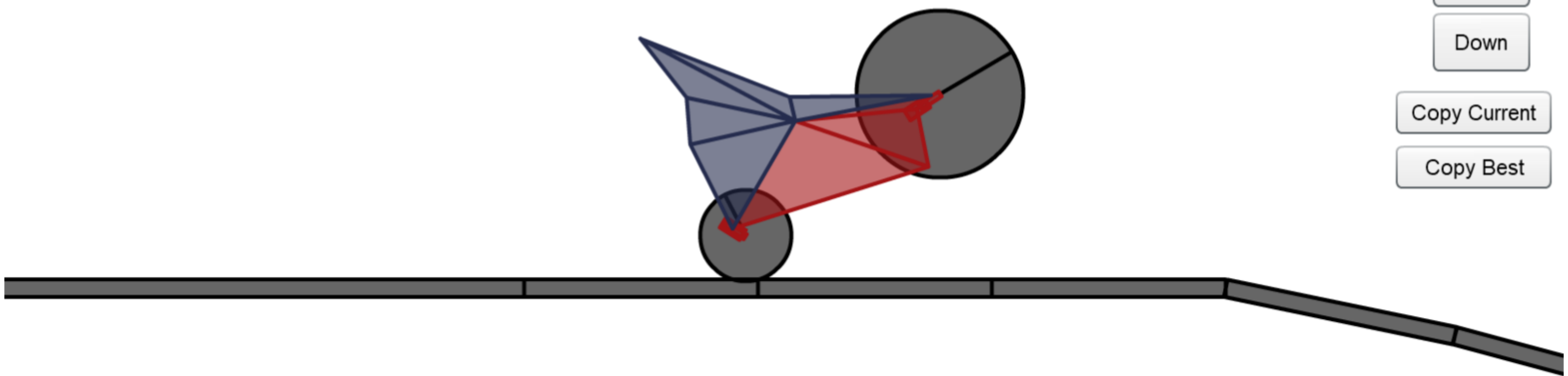
Up

Next

Down

Copy Current

Copy Best



A rather good design:

eNqzXzkTBGY5MD1NcOw097F/s/z93E26fPa3LoAELOzv9cdofD
3U78DBAAb215h+5OmcOWN/dW6HtbR+k/3e4qD1PmfOOrBJ
SwR9nDnL/o6Dw3fJmTPt76zhMFR7WWF/u0811P7/f/uj5pGqY
P2dB9mBFjqwvbIXk7i97z8Q2H+cbio3X+qe/QGWNkugW4DK
WO0/up/4FKNqYv8pDQxAdjswW2+9zHM30P779HDpZ2AxZgf
mzQfn7Sl7Y3+NUR3iQgY2ByHpJR8P/91o/6h9sgtEjN2Bof5BrO
Ht/fZPTF40Rc6cCTaPgc9ow/3Hk+2flams6DlzBuQWB8aVR8L+
aIfb7775VygtLZ1BVccPjhM33GNok/8AZvsvVmDQ1rwEpt9plqG
oA+G/jplgGuhGAKc8iiU=

File processing

Motivation

- Most computers are used for data processing, as a big growth area in the “information age”
- Data processing from a computer science perspective:
  - Storage of data
  - Organization of data
  - Access to data
  - Processing of data

Why should one use files?

- Until now, the Python programs you have been writing use pretty simple input and output
  - User types input at the keyboard
  - Results (output) are displayed in the console
- This is fine for short and simple input…

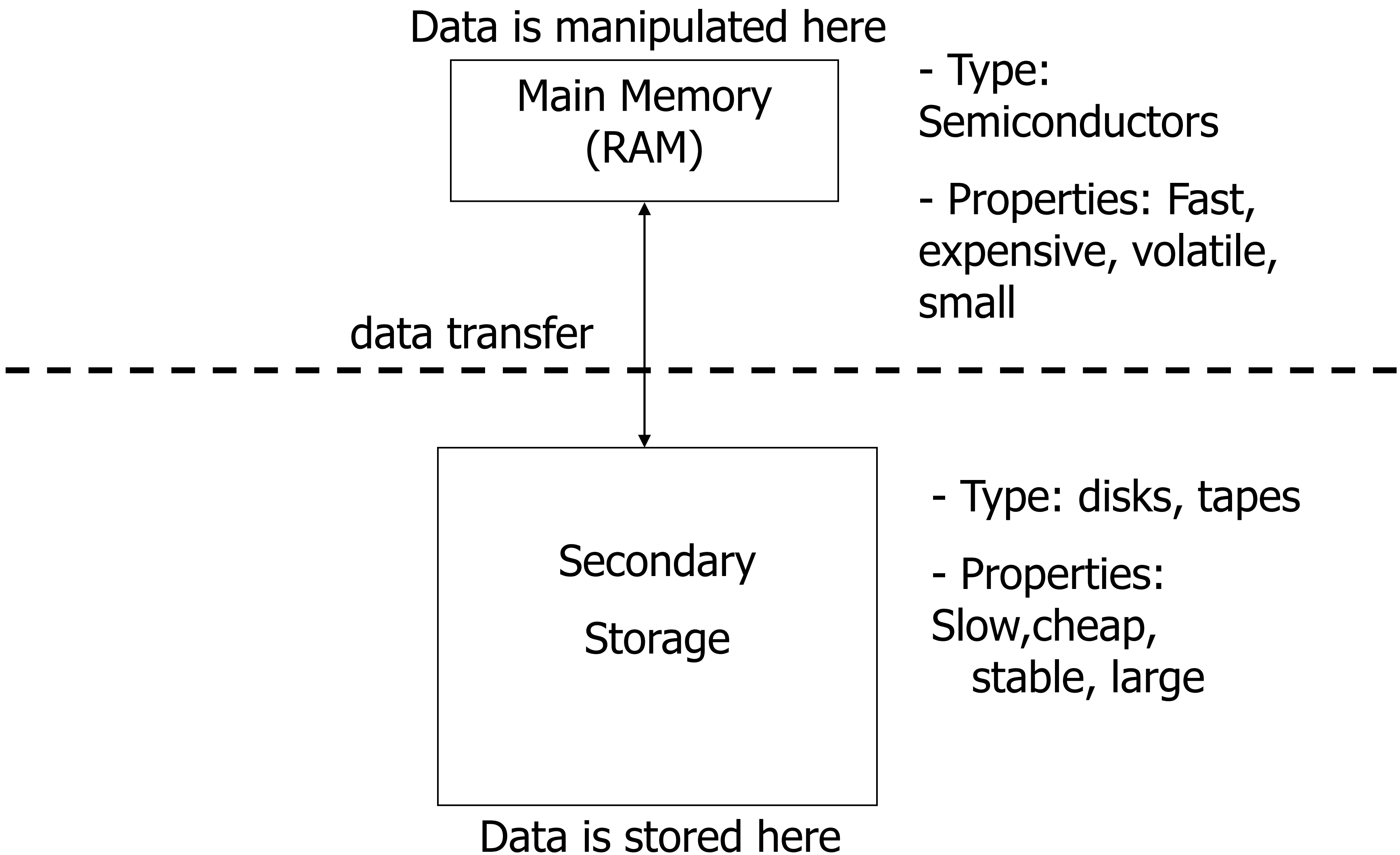
Why should one use files?

- Until now, the Python programs you have been writing use pretty simple input and output
  - User types input at the keyboard
  - Results (output) are displayed in the console
- This is fine for short and simple input…
  - But what if we want to compute the largest prime number and are interrupted for some reason? ☺
  - Start all over again?

Data structures vs file structures

- Both involve:
  - Representation of Data + Operations for accessing data
- Difference:
  - Data structures: deal with data in the main memory
  - File structures: deal with the data in the secondary storage

Review: Computer architecture



- Type: Semiconductors

- Properties: Fast, expensive, volatile, small

- Type: disks, tapes

- Properties: Slow,cheap, stable, large

How fast is main memory?

- Typical time for getting info from:

  Main memory: ~10 nanosec = 10×10^{-9} sec

  Hard disks: ~10 milisec = 10×10^{-3} sec

- An analogy keeping same time proportion as above:

  seconds versus **week**

File processing

- In order to do interesting things with files, we need to be able to perform certain operations:
  - Associate an external file with a program object
    - Opening the file using its path
  - Manipulate the file object
    - Reading from or writing to the file object
  - Close the file
    - Making sure the object and file match at the end

Opening and closing a file with Python

Syntax for `open()` function

```
myFile = open(file_name [, access_mode])
```

Syntax for `open()` function

```
myFile = open(file_name [, access_mode])
```

`file_name`

- This argument is a string the contains the name of the file you want to access
  - `"input.txt"`
  - `"numbers.dat"`
  - `"roster.txt‟`
  - `‟D:\\myfiles\\data.txt‟ (on Windows)`

Syntax for `open()` function

```
myFile = open(file_name [, access_mode])
```

`access_mode` (optional argument)

- This argument is a string that determines which of the modes the file is to be opened in
  - **`"r"`** (open for reading)
  - **`"w"`** (open for writing)
  - **`"a"`** (open for appending)

Examples of using `open()`

- In general, we will use commands like:

```
myFile  = open("scores.txt")
dataIn  = open("stats.dat",  "r")
dataOut = open("stats2.dat", "w")
```

an example input file

```
scores.txt
Lisa 100
Bart 60
Homer 20
```

Syntax for `close()` member function

```
myfile.close()
```

- File variables are actually objects!
- They can be used like lists, sets, etc.
  - The have member variables
  - The have member functions
  - …

Full example

```
myFile=open("scores.txt")
myFile.close()
```

What do we get so far? Not much ...

Test: Using `open()`

- Which of these are valid uses of **`open()`**?

```
1. myFile  = open(12, "r")
2. fileObj = open("HELLO.txt")
3. writeTo = open(fileName, "w")
4. "file"  = open("test.dat", "R")
5. theFile = open("file.dat", "a")
```

Test: Using `open()`

- Which of these are valid uses of `open()`?

```
✗ 1. myFile  = open(12, "r")
✓ 2. fileObj = open("HELLO.txt")
✓ 3. writeTo = open(fileName, "w")
✗ 4. "file"  = open("test.dat", "R")
✓ 5. theFile = open("file.dat", "a")
```

Reading a file with Python

Three Ways to Read a File

- There are three different ways to read in a file:

1. Read the whole file in as one big long string

   `myFile.read()`

2. Read the file one line at a time

   `myFile.readline()`

3. Read the file as a list of strings (each is one line)

   `myFile.readlines()`

Entire Contents into One String

```
myFile=open("scores.txt")
wholeThing=myFile.read()
print(wholeThing)
myFile.close()
```

```
Lisa 100
Bart 60
Homer 20
```

What is this really?

Our input file:

scores.txt

```
Lisa 100
Bart 60
Homer 20
```

Entire Contents into One String

```
myFile=open("scores.txt")
wholeThing=myFile.read()
print(repr(wholeThing))
```

```
'Lisa 100\nBart 60\nHomer 20'
```

it's literally one giant string!

our input

scores.txt

```
Lisa 100
Bart 60
Homer 20
```

One Line at a Time

```
>>> myFile=open("scores.txt")
>>> lineOne=myFile.readline()
>>> lineOne
'Lisa 100\n'
>>> lineTwo=myFile.readline()
'Bart 60\n'
```

our input file

```
scores.txt
Lisa 100
Bart 60
Homer 20
```

As a List of Strings

```
>>> info = open("hours.txt")
>>> listOfLines = info.readlines()
>>> listOfLines
['Lisa 100\n',
 'Bart 100\n',
 'Homer 20\n']
```

our input file

```
scores.txt
Lisa 100
Bart 60
Homer 20
```

Writing to a file with Python

Opening a File for Writing

- Use **`open()`** just like we do for reading
  - Provide the filename <u>and the access mode</u>

```
fileObj = open("output.txt", "w")
```

  - Opens the file for writing
  - Wipes the contents!

```
fileObj = open("myNotes.txt", "a")
```

  - Opens the file for appending
  - Writes new data to the end of the file

Writing to a File

- Once a file has been opened, we can write to it
  - What do you think the function to write is called?

```
myFile.write( "hello world!" )
```

- We can also use a string variable in **`write()`**

```
myFile.write( writeString )
```

Details About `write()`

- **`write()`** only writes exactly what it's given!
  - This means whitespace (like **`"\n"`**) is up to you
  - Unlike **`print()`**, which adds a newline for you

```
myFile = open("greeting.dat", "w")
myFile.write("Hello\nWorld\n")
myFile.close()
```

Word of Caution

- Write can only take <u>one string</u> at a time!

- These won't work:

```
fileObj.write("hello", "my", "name")
fileObj.write(17)
```

Why don't these work?
the first is multiple strings
the second is an int, not a string

- But this will:

```
fileObj.write("hello" + " my " + "name")
fileObj.write(str(17))
```

Why does this work?
concatenation creates <u>one</u> string
casting turns the int into a string

Complete example

Task

- Load the data from scores.txt and convert it to a dictionary with keys being names and values being scores!

our input file

```
scores.txt
Lisa 100
Bart 60
Homer 20
```

Solution?

- Load the data from scores.txt and convert it to a dictionary with keys being names and values being scores!

```
myFile=open("scores.txt")
lines=myFile.readlines()
d={}
for l in lines:
    elements=l[:-1].split(" ")
    d[elements[0]]=elements[1]
print(d)
```

Any problems?

Printout: {'Lisa': '100', 'Bart': '60', 'Homer': '2'}

Solution!

- Load the data from scores.txt and convert it to a dictionary with keys being names and values being scores!

```
myFile=open("scores.txt")
lines=myFile.readlines()
d={}
for l in lines:
    if l[-1]=="\n":
        elements=l[:-1].split(" ")
    else:
        elements=l.split(" ")
    d[elements[0]]=elements[1]
print(d)
```

Printout: {'Lisa': '100', 'Bart': '60', 'Homer': '20'}

Thank you very much!