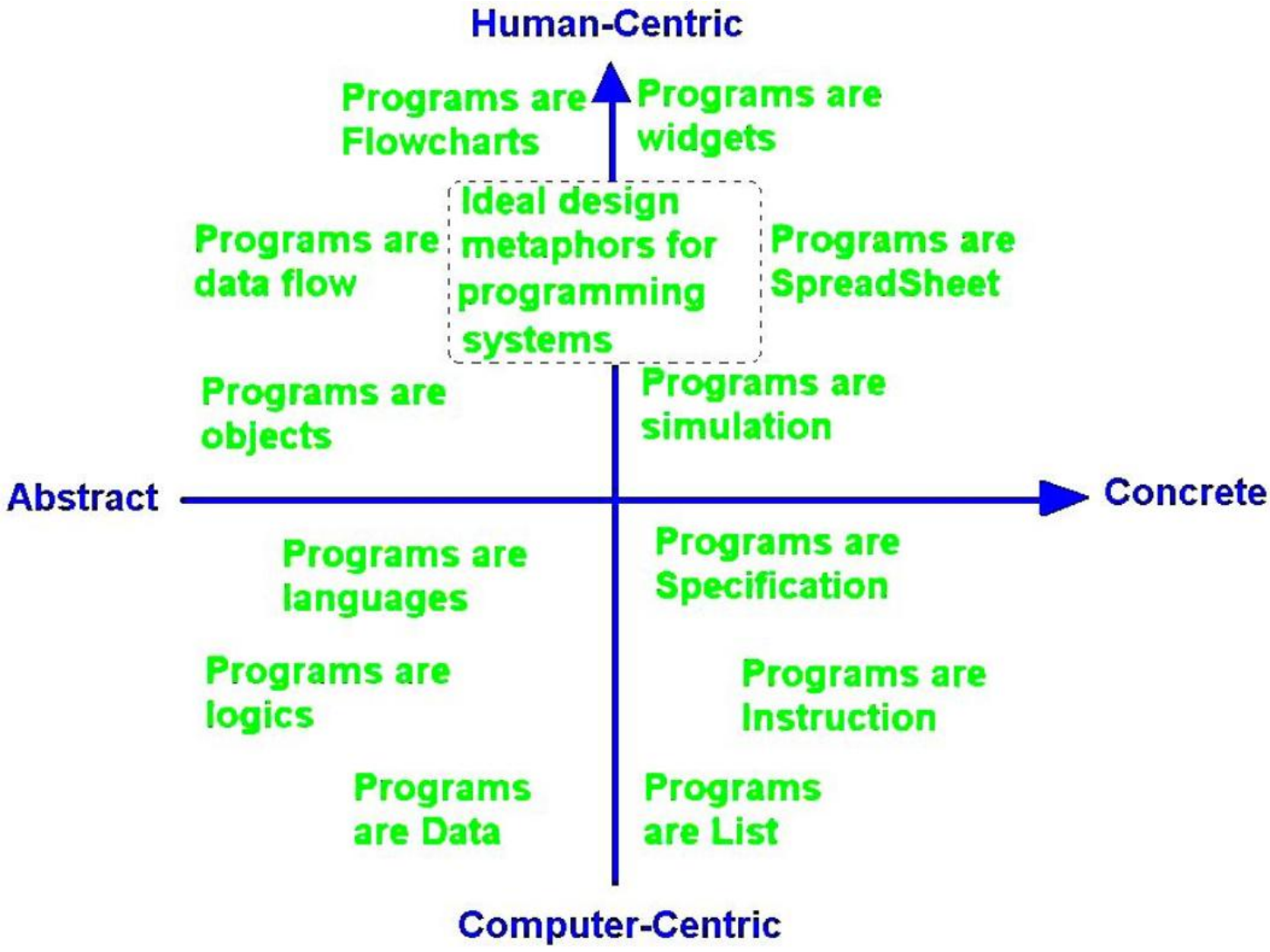
Algorithms

Using algorithms

D.S. Hwang

Department of Computer Science,
Dankook University


Human-Centric
Programs are Flowcharts
Programs are widgets
Ideal design metaphors for programming systems
Programs are data flow
Programs are SpreadSheet
Programs are objects
Programs are simulation
Abstract
Concrete
Programs are languages
Programs are Specification
Programs are logics
Programs are Instruction
Programs are Data
Programs are List
Computer-Centric

1. The web server sends the query to the index servers. The content inside the index servers is similar to the index in the back of a book--it tells which pages contain the words that match any particular query term.

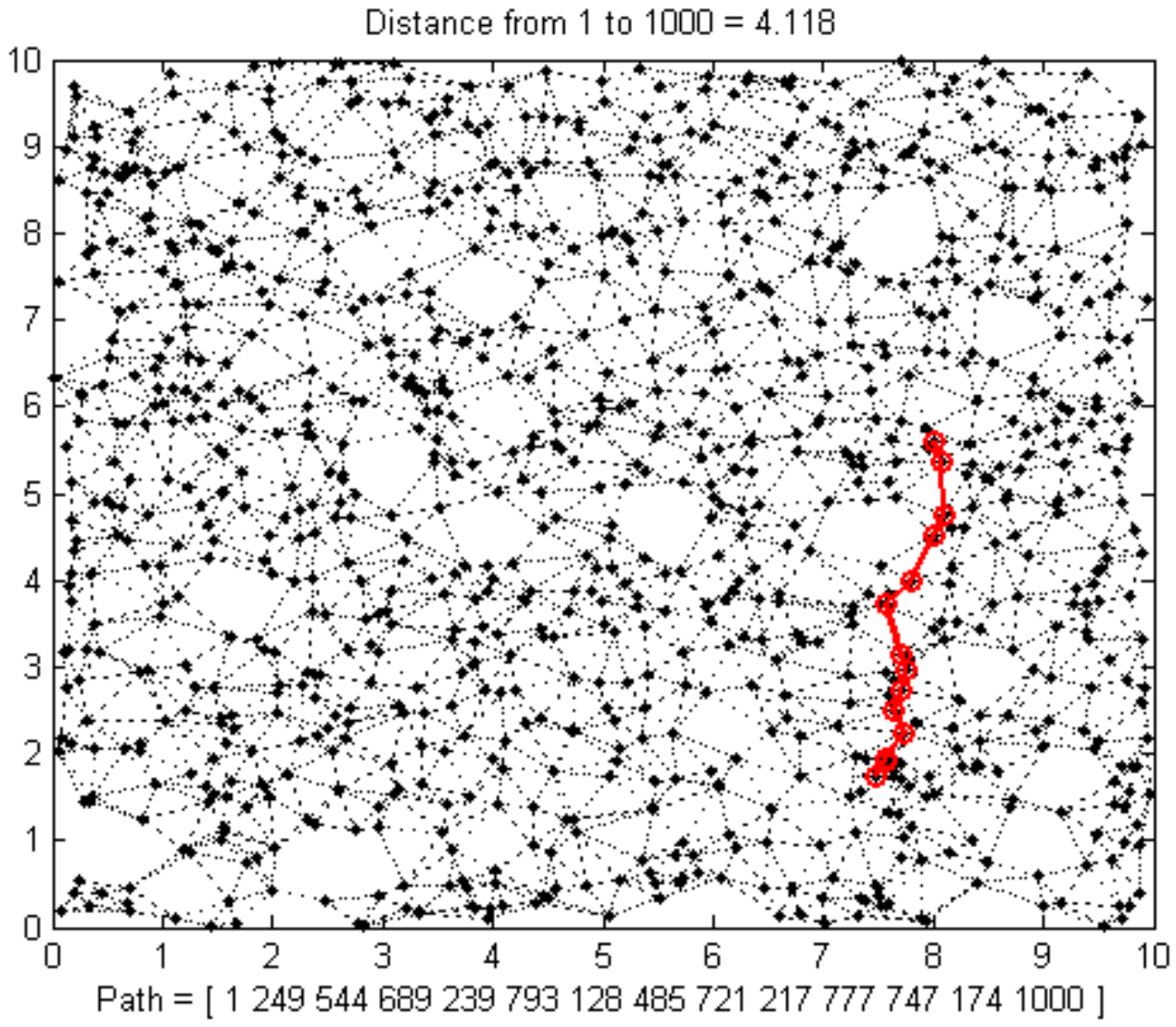
2. The query travels to the doc servers, which actually retrieve the stored documents. Snippets are generated to describe each search result.

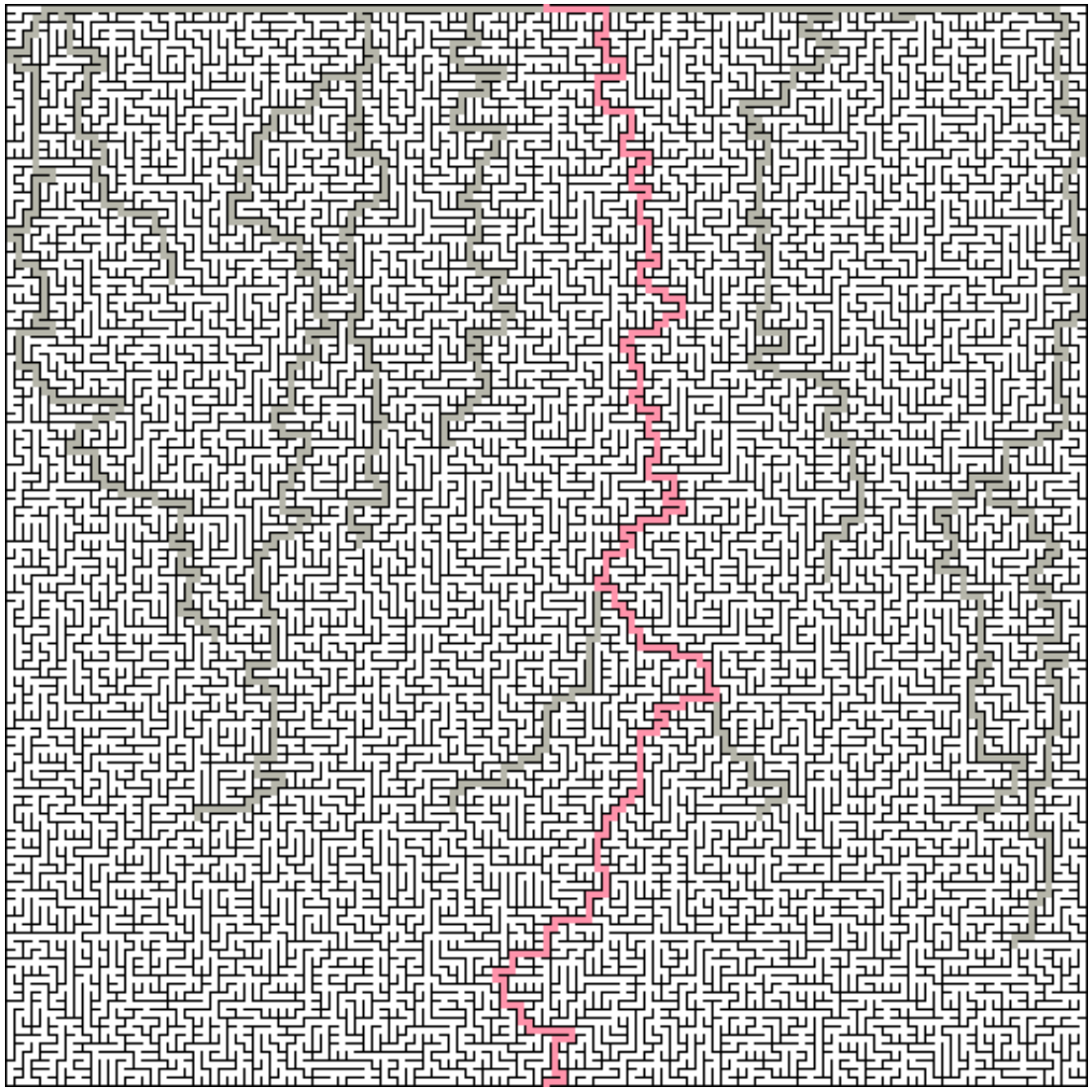
3. The search results are returned to the user in a fraction of a second.

Outline

Overview

- Algorithm
- How algorithms work?
- What is the systematic way of solving a computational problem?
- Algorithm-writing technique
- Kinds of algorithms-searching and timing


Distance from 1 to 1000 = 4.118
0
1
2
3
4
5
6
7
8
9
10
Path = [1 249 544 689 239 793 128 485 721 217 777 747 174 1000]



Algorithm

- a set of steps that solves a problem
- How to describe?
  - programming language
  - a human language
  - mathematics

Algorithm-writing technique

Top-down design

- describe your solution in your language
- mark the phrases that correspond directly to Python statements
- rewritten in more detail in your language, until everything in your description can be written in Python.

Algorithm-writing technique

Top-down design

- describe your solution in your language
- mark the phrases that correspond directly to Python statements
- rewritten in more detail in your language, until everything in your description can be written in Python.

Searching

Data on the no of humpback whales sighted off the coast of British Columbia over the past ten years

```
809 834 477 478 307 122 96 102 324 476
```

Which year had the lowest number of sightings during those years?

```
1 >>> counts = [809, 834, 477, 478, 307, 122, 96, 102,
2 324, 476]
3 >>> min(counts)
4 96
```

Searching

What year is the lowest?

```
>>> counts = [809, 834, 477, 478, 307, 122, 96, 102,
324, 476]
>>> low = min(counts)
>>> min_index = counts.index(low)
>>> print(min_index)
6
>>> counts.index(min(counts))
6
```

Searching

What if we want to find the indices of the two smallest values?

- *Find, remove, find*. Find the index of the minimum, remove that element from the list, and find the index of the new minimum element in the list.
- *Sort, identify minimums, get indices*. Sort the list, get the two small- est numbers, and then find their indices in the original list.
- *Walk through the list*. Examine each value in the list in order, keep track of the two smallest values found so far, and update these values when a new smaller value is found.

Searching

Find, remove, find.

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of the two
3         smallest values in list L.'''
4      find the index of the minimum element in L
5      remove that element from the list
6      find the index of the new minimum element
7         in the list
8      return the two indices
```

Searching

Find, remove, find.

```
1 def find_two_smallest(L):
2     '''Return a tuple of the indices of the two
3        smallest values in list L.'''
4     get the minimum element in L
5     find the index of that minimum element
6     remove that element from the list
7     find the index of the new minimum element
8        in the list
9     return the two indices
```

Searching

Find, remove, find.

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of the two
3         smallest values in list L.'''
4      smallest = min(L)
5      min1 = L.index(smallest)
6      L.remove(smallest) # delete the smallest one
7      next_smallest = min(L)
8      min2 = L.index(next_smallest)
9
10     return the two indices
```

Searching

Find, remove, find.

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of the two
3         smallest values in list L.'''
4      smallest = min(L)
5      min1 = L.index(smallest)
6      L.remove(smallest) # delete the smallest one
7      next_smallest = min(L)
8      min2 = L.index(next_smallest)
9
10     put smallest back into L
11     if min1 comes before min2, add 1 to min2
12
13     return the two indices
```

Searching

Find, remove, find.

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of the two
3         smallest values in list L.'''
4      smallest = min(L)
5      min1 = L.index(smallest)
6      L.remove(smallest) # delete the smallest one
7      next_smallest = min(L)
8      min2 = L.index(next_smallest)
9
10     L.insert(min1, smallest)
11     if min1 <= min2:
12         min2 += 1
13
14     return (min1, min2)
```

Searching

Sort, Identify Minimums, Get Indices

```
1 def find_two_smallest(L):
2     '''Return a tuple of the indices of
3       the two smallest values in list L.'''
4     sort a copy of L
5     get the two smallest numbers
6     find their indices in the original list L
7     return the two indices
```

Searching

Sort, Identify Minimums, Get Indices

- use `L.sort()`
- work on a copy of `L`

```
1 def find_two_smallest(L):
2     '''Return a tuple of the indices of
3       the two smallest values in list L.'''
4     temp_list = L[:]
5     temp_list.sort()
6     smallest = temp_list[0]
7     next_smallest = temp_list[1]
8     find their indices in the original list L
9     return the two indices
```

Searching

Sort, Identify Minimums, Get Indices

- use `L.sort()`
- work on a copy of `L`

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3        the two smallest values in list L.'''
4      temp_list = L[:]
5      temp_list.sort()
6      smallest = temp_list[0]
7      next_smallest = temp_list[1]
8      min1 = L.index(smallest)
9      min2 = L.index(next_smallest)
10     return (min1, min2)
```

Searching

Walk Through the List

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3         the two smallest values in list L.'''
4      examine each value in the list in order
5      keep track of the indices of
6        the two smallest values found so far
7      update these values when a new smaller value
8        is found
9      return the two indices
```

Searching

Walk Through the List

- move the second line before the first one because it describes the whole process

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3         the two smallest values in list L.'''
4      keep track of the indices of
5        the two smallest values found so far
6      examine each value in the list in order
7        to update these values when a new smaller value is found
8      return the two indices
```

Searching

Walk Through the List

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3         the two smallest values in list L.'''
4      set min1 and min2 to the indices of the smallest
5         and next-smallest values at the beginning of L
6      examine each value in the list in order
7         update these values
8           when a new smaller value is found
9      return the two indices
```

Searching

Walk Through the List

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3         the two smallest values in list L.'''
4      #set min1 and min2 to the indices of the smallest
5      #   and next-smallest values at the beginning of L
6      if L[0] < L[1]:
7          smallest, next_smallest = 0, 1
8      else:
9          smallest, next_smallest = 1, 0
10     examine each value in the list in order
11         update these values
12             when a new smaller value is found
13     return the two indices
```

Searching

Walk Through the List

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3         the two smallest values in list L.'''
4      #set min1 and min2 to the indices of the smallest
5      #   and next-smallest values at the beginning of L
6      if L[0] < L[1]:
7          smallest, next_smallest = 0, 1
8      else:
9          smallest, next_smallest = 1, 0
10     #examine each value in the list in order
11     for i in range(2, len(values)):
12         update min1 and or min2
13             when a new smaller value is found
14     return the two indices
```

Searching

Walk Through the List

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3         the two smallest values in list L.'''
4      #set min1 and min2 to the indices of the smallest
5      #   and next-smallest values at the beginning of L
6      if L[0] < L[1]:
7          smallest, next_smallest = 0, 1
8      else:
9          smallest, next_smallest = 1, 0
10     #examine each value in the list in order
11     for i in range(2, len(values)):
12         L[i] is larger than
13             both min1 and min2, smaller than both, or in between.
14         if L[i] is larger than
15             both min1 and min2, skip it
16         if L[i] is smaller than
17             min1 and min2, update them both
18         if L[i] is in between,
19             update min2
20     return (min1, min2)
```

Searching

Walk Through the List

```
1  def find_two_smallest(L):
2      '''Return a tuple of the indices of
3         the two smallest values in list L.'''
4      #set min1 and min2 to the indices of the smallest
5      #   and next-smallest values at the beginning of L
6      if L[0] < L[1]:
7          min1, min2 = 0, 1
8      else:
9          min1, min2 = 1, 0
10     #examine each value in the list in order
11     for i in range(2, len(values)):
12         if L[i] < L[min1]:
13             min2 = min1; min1 = i
14         elif L[i] < L[min2]:
15             min2 = i
16     return (min1, min2)
```

Timing I

Profiling a program measures

- how long it takes to run(time)
- how much memory it uses(space)

Compare those algorithms

Algorithm	Running Time(ms)
Find, remove, find	1.117
Sort, identify, index	2.128
Walk through the list	1.472

Timing II

Estimate the running time

```
1 import time
2
3 t0 = time.time()
4     stmt 1
5         ...
6     stmt k
7 t1 = time.time() - t0
```

Summary

- Top-down design is the most effective way to design algorithms.
- The performance of a program is characterized by time and memory use.
- Almost all problems have more than one correct solution.

Problem 1 I

Homework

A DNA sequence is a string made up of the letters A, T, G, and C. To find the complement of a DNA sequence, As are replaced by Ts, Ts by As, Gs by Cs, and Cs by Gs. For example, the complement of AATTGCCGT is TTAACGGCA.

1. Write a pseudo code in English of the algorithm that takes a DNA sequence and return its complement.
2. Test your algorithm.
   - `ttcccatcaa gccctagggc tcctcgtggc tgctgggagt tgtagtctga acgcttctat`
   - `cttggcgaga agcgcctacg ctccccctac cgagtcccgc ggtaattctt aaagcacctg`
   - `caccgccccc ccgccgcctg cagagggcgc agcaggtctt gcacctcttc tgcatctcat`
   - `tctccaggct tcagacctgt ctccctcatt caaaaaatat ttattatcga gctcttactt`

Problem 1 II

Homework

3. Write a function named `complement` that takes a DNA sequence and returns the complement of it. Here, we can get some examples from the `https://www.ncbi.nlm.nih.gov`. For example, `p53.rtf` is given.

Problem 2

Homework

Develop a function that finds the minimum or maximum value in a list, depending on the caller's request.

1. Write a loop (including initialization) to find both the minimum value in a list and that value's index in one pass through the list.
2. Write a function named `min_index` that takes a list and returns a tuple containing the minimum value in the list and that value's index in the list.
3. Write a function named `max_index` that takes a list and returns a tuple containing the maximum value in the list and that value's index in the list.