Programming Design In-class Practices

Variables and Arrays

Ling-Chieh Kung

Department of Information Management
National Taiwan University

Problem 1: 0.1 + 0.2 = ?

- What is the sum of 0.1 + 0.2?
- Write a program to:
  - Declare two **`float`** a and b.
  - Let the user enter 0.1 and 0.2 for a and b.
  - Print out $a + b$.
  - Check if $a + b$ equals 0.3. If so, print out "**`a + b is 0.3.`**"; otherwise, print out "**`a + b is not 0.3.`**"

- Include the library **`<iomanip>`**.
  - Print out $a + b$ by writing **`cout << setprecision(10) << a + b;`**.
  - What do you see?

Problem 2: the knapsack problem

- You have n items.
  - The weight and value of item i is w_i kg and v_i dollars, respectively.
  - None of these items is divisible.
- You have a knapsack (i.e., a backpack) that can carry at most B kg.
- Which items should you choose to maximize the total value without breaking the knapsack?
- This is the well-known **knapsack** problem.

$$
\begin{aligned}
\max \quad & \sum_{i=1}^{n} v_i x_i \\
\text{s.t.} \quad & \sum_{i=1}^{n} w_i x_i \leq B \\
& x_i \in \{0, 1\} \quad \forall i = 1, ..., n.
\end{aligned}
$$

- Suppose you have 4 items of weights 2, 3, 4, 3 and values 2, 4, 5, 3, respectively, and a backpack of capacity 9,
  - The best thing to do is to choose items 1, 2, and 3.

Problem 2: the knapsack problem

- In general, solving a knapsack problem with many items is hard.
  - In Computer Science, we say that the knapsack problem is NP-hard, which means most researchers in the world believe that there is no “efficient” method for finding an optimal solution.
- In this problem, we will only ask you to evaluate one candidate solution.
  - Is it feasible?
  - If so, what is its total weight and value?

Problem 2: the knapsack problem

- Input:
  - The first line: An integer $n \in \{1, \ldots, 100\}$, a white space, and an integer $B \in \{1, \ldots, 10000\}$.
  - The second line: n positive integers w_1, w_2, ..., and w_n. Each two integers are separated by a white space. $w_i \in \{1, \ldots, 100\}$.
  - The third line: n positive integers v_1, v_2, ..., and v_n. Each two integers are separated by a white space. $v_i \in \{1, \ldots, 100\}$.
  - The fourth line: n binary values $x_i \in \{0, 1\}$, $i = 1, \ldots, n$. $x_i = 1$ if item i is selected or 0 otherwise.
- Output:
  - If the solution is infeasible, print out "0".
  - Otherwise, print out the total weight and value of this solution, separated by a white space.

Problem 2: the knapsack problem

- Sample input/output:

Input:
`4 9`
`2 3 4 3`
`2 4 5 3`
`1 0 1 1`
 	
Output:
`9 10`
	Input:
	`4 9`
	`2 3 4 3`
	`2 4 5 3`
	`1 1 1 1`
	
	Output:
	`0`

Problem 3: moving average

- Sometimes we need to predict future demands.
  - To prepare inventory.
  - To hire workers.
  - To plan for capacity.
  - To negotiate prices.
- We do **demand forecasting** typically according to past sales/demand.
- While there are many ways, an easiest way is **the moving average method**.
  - It is one kind of **time series** methods.
  - It uses nothing but the **past sales/demand** information.

Problem 3: moving average

- Moving average:
  - We say that the next period demand will be the **average of the demands in the past n periods**.
  - The term "moving" comes from the fact that we keep **moving the reference window** when we move one period further.
- As an example, suppose that $n = 3$:

Demand	14	23	26	17	17	12	24	19	10	18	N/A
Forecast	N/A	N/A	N/A	21	22	20	15.3	17.7	18.3	17.7	15.7

- Let's implement the moving average method.

Problem 3: moving average

- Input:
  - The first line: An integer $n \in \{2, 3, 4, 5, 6\}$.
  - The second line: An integer m that is within 10 to 1000.
  - The third line: m integers $x_1, x_2, \ldots, x_m$ that are with 0 and 10000. Two consecutive numbers are separated by a white space.
- Output:
  - The forecasts $y_i = \left\lfloor \frac{x_{i-1}+x_{i-2}+\cdots+x_{i-n}}{n} \right\rfloor$ for all $i = n+1, n+2, \ldots, m+1$.
  - Separate two consecutive values by a white space.

Problem 3: moving average

- Sample input/output:

Input:
```
3
10
14 23 26 17 17 12 24 19 10 18
```

Output:
```
21 22 20 15 17 18 17 15
```

Problem 4: choosing the window size n

- The window size n is typically set to be within 2 and 6.
- How to choose n?
  - In practice, sometimes n is chosen with domain knowledge.
  - For most products, however, this is impossible.
- One thing we may do is to let the computer **learn** from **historical observations**.
  - Let's consider past demands and ask "what if we have chosen n to be that value."

Problem 4: choosing the window size n

- For moving average:
  - We may simply try each of $n = 2, 3, \ldots, N$, where N is given.
  - All the past sales/demand for the testing set.
- For each value of n, we need to calculate its "**performance**."
  - A simple measurement is the **mean absolute error** (MAE):

$$\text{MAE} = \sum_{t=1}^{T} |s_t - f_t| \Big/ T \,,$$

  - s_t and f_t are the sales and forecast in period t, respectively.
  - $f_t = \frac{x_{t-1} + x_{t-2} + \cdots + x_{t-n}}{n}$ is the original forecast value (not truncated to an integer).
  - We will choose the window size **whose MAE is the smallest**.

Problem 4: choosing the window size n

- Input:
  - The first line: An integer $N \in \{6, 7, \ldots, 10\}$.
  - The second line: An integer m that is within 10 to 1000.
  - The third line: m integers $x_1, x_2, \ldots, x_m$ that are with 0 and 10000. Two consecutive numbers are separated by a white space.
- Output:
  - The $n \in \{2, 3, \ldots, N\}$ that minimizes MAE, a white space, and then the minimized MAE truncated to the second digit after the decimal point.

Problem 4: choosing the window size *n*

- Sample input/output:

```
Input:
6
13
12 23 21 15 24 8 9 12 17 32 28 33 24

Output:
2 6.86
```