2016 APL Problem Solving Competition – Phase II Problem Descriptions

Overview

This year's Phase II problems are divided into three sets, representing three categories – Bioinformatics, Finance and General Computing. Each set has three problems; a problem can be broken down into multiple tasks.

Three grand prizes will be awarded, one in each of the categories.
To be considered for a grand prize, you must solve all of the problems in a category.

You may submit an entry that completes more than one category. However, you can only win once – for example, if your entries in all three categories are judged to be the best in each category, the judging committee will select the entry they deem the best and declare you the winner in that category while other competitors will be selected as the winners of the other categories.

Good Luck and Happy Problem Solving!

Note:
Some of the examples are displayed using the user command setting `]Boxing on` to more clearly depict the structure of the displayed data.

```
      ('Dyalog' 'APL')(4 4⍴⍳16) 5
  Dyalog  APL     1  2  3  4  5
                  5  6  7  8
                  9 10 11 12
                 13 14 15 16

      ]Boxing on
Was OFF

      ('Dyalog' 'APL')(4 4⍴⍳16) 5
┌─────────────┬───────────┬─┐
│             │ 1  2  3  4│5│
│┌──────┬───┐ │ 5  6  7  8│ │
││Dyalog│APL│ │ 9 10 11 12│ │
│└──────┴───┘ │13 14 15 16│ │
└─────────────┴───────────┴─┘
```

Description of the Contest2016 Template Files

Two template files are available for download from the contest website. Which file you use depends on how you choose to implement your problem solutions.

If you use Dyalog APL, you should use the template workspace `Contest2016.DWS`

The Contest2016 workspace contains:

- Three namespaces, one for each of the problem categories. Each namespace contains:
    - stubs for all of the functions described in the problem descriptions. The function stubs are implemented as traditional APL functions (tradfns) but you can change the implementation to dynamic functions (dfns) if you want to do so. Either form is acceptable.
    - any sample data elements mentioned in the problem descriptions.

    Any sub-functions that you develop as a part of your solution should be co-located in the category namespace.

  The namespaces are:

    - `#.bio` – for the bioinformatics problem set
    - `#.fin` – for the finance problem set
    - `#.gen` – for the general computing problem set

- **`#.SubmitMe`** – a function used to package your solution for submission.

Make sure you save your work using the `)SAVE` system command!

Once you have developed and are ready to submit your solutions, run the **`#.SubmitMe`** function, enter the requested information and click the **Save** button. **`#.SubmitMe`** will create a file called **Contest2016.dyalog** which will contain any code or data you placed in the **`#.bio`**, **`#.fin`**, and **`#.gen`** namespaces. You will then need to upload the **Contest2016.dyalog** file using the contest website.

If you use some other APL system, you can use the template script file `Contest2016.dyalog`

This file contains the correct structure for submission. You can populate it with your code, but do not change the namespace structure. Once you have developed your solution, edit the variable definitions as indicated at the top of the file and upload the file using the contest website. If you use some other APL system to develop your application, **it will still need to execute under Dyalog APL**; it is recommended that your solution use APL features that are common between your APL system and Dyalog.

Set 1: Bioinformatics Problems

The Bioinformatics problems presented here are based on problems presented on the website http://rosalind.info. The descriptions have been adapted to be more suitable for APL syntax and this competition.

Please note that your solutions should perform on arguments as described in the "**Given:**" section of the problem descriptions on rosalind.info. For example, in "Translating RNA into Protein" it states:

> "**Given:** An RNA string s corresponding to a strand of mRNA (of length at most 10 kbp)."

This means your solution should work with arguments up to 10,000 base pairs.

Bioinformatics Problem 1 **(1 task)**

Task 1 – Comparing Spectra with the Spectral Convolution

The complete description of this task can be found at http://rosalind.info/problems/conv/.

Suppose you have two mass spectra, and you want to check if they both were obtained from the same protein; you will need some notion of spectra similarity.

Write an APL function named **maxMult** which:

- takes vectors of positive real numbers representing multisets as its left and right arguments
- returns a 2-element vector of
  - [1] the largest multiplicity of the inputs
  - [2] the absolute value of the number maximizing the shared masses of the inputs

Example:

```
    s1←186.07931 287.12699 548.20532 580.18077 681.22845 706.27446 782.27613
968.35544 968.35544
    s2←101.04768 158.06914 202.09536 318.09979 419.14747 463.17369

    s1 maxMult s2
3 85.0163
```

Bioinformatics Problem 2 (1 task)

Task 1 – Interleaving Two Motifs

The complete description of this problem can be found at http://rosalind.info/problems/scsp/.

In this problem, we are given two different motifs and we wish to interleave them together in such a way as to produce a shortest possible genetic string in which both motifs occur as subsequences.

Write an APL function named **`interleave`** which:

- takes character vectors representing DNA strings as its left and right arguments
- returns a character vector representing a shortest common supersequence of the arguments

Example:

```
      'ATCTGAT' interleave 'TGCATA'
ATGCATGAT
```

Example Explained:

The result represents the shortest sequence in which the two arguments are presented completely and in order.

```
Left argument:     AT C TGAT
Result:            ATGCATGAT
Right argument:     TGCAT A
```

Bioinformatics Problem 3 **(1 task)**

***Task 1* – Sorting by Reversals**

The complete description of this problem can be found at http://rosalind.info/problems/rear/.

Perhaps the most common type of genome rearrangement is an inversion, which flips an entire interval of DNA found on the same chromosome. In this problem we would like to determine the minimum number of inversions that have occurred on the evolutionary path between two chromosomes.

Write an APL function named **`reversals`** which:

- takes an integer vector left right argument representing π as described on Rosalind.info
- takes an integer vector left argument representing σ as described on Rosalind.info
- returns a 2-element vector with elements of

  [1] an integer representing the reversal distance between π and σ

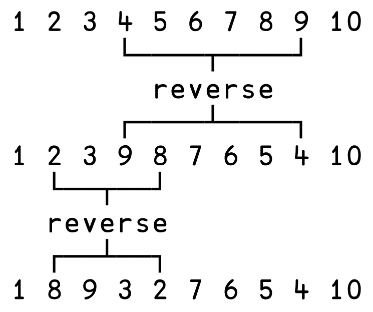
  [2] a vector of 2-element integer vectors representing the collection of reversals sorting π into σ

Example:

```
      1 2 3 4 5 6 7 8 9 10 reversals 1 8 9 3 2 7 6 5 4 10
┌─┬─────────┐
│2│┌───┬───┐│
│ ││4 9│2 5││
│ │└───┴───┘│
└─┴─────────┘
```

Example Explained:

To perform the conversion, two reversals are necessary – the first from index positions 4-9, and the second from index positions 2-5:



Set 2: Finance Problems

Finance Problem 1 – (1 task)

Task 1 – Calculating Forward Rates from the Yield Curve

According to Investopedia – a yield curve is a line that plots the interest rates, at a set point in time, of bonds having equal credit quality, but differing maturity dates. The most frequently reported yield curve compares the three-month, two-year, five-year and 30-year U.S. Treasury debt. This yield curve is used as a benchmark for other debt in the market, such as mortgage rates or bank lending rates. The curve is also used to predict changes in economic output and growth.

The spot rate is the return on a bond of a specified maturity (contract length) purchased immediately.
The forward rate is the predicted yield of a bond purchased at a specified time in the future.

Forward rates can be estimated from spot rates on the yield curve in the following manner[1]:

$$F_{1,2} = \frac{R_2T_2 - R_1T_1}{T_2 - T_1}$$

where R_1 and R_2 are consecutive spot rates on the yield curve and T_1 and T_2 are their corresponding maturities.

For example, a 1-year bond has a spot rate of 2.96% and a 2-year bond has a spot rate of 3.29%, then the market's expectation for the spot rate for a 1-year bond purchased 1 year from now, the forward rate, can be calculated as:

$$F_{1,2} = \frac{(3.29 \times 2) - 2.96}{2 - 1} = 3.62$$

Write an APL function named `forward` which

- takes a numeric vector left argument representing a list of maturities expressed in years; fractional years are allowed (e.g. .25 would correspond to a 3-month maturity)
- takes a numeric vector right argument representing a list of spot rates
- returns a numeric vector of the computed forward rates

Example:

We first create a vector of maturities expressed in years:

```
 maturities←.25 .5 1 2 3 5 7 10 20
```

Next we create a vector of Treasury spot rates (the yield curve) obtained from the U.S. Treasury web site corresponding to those maturities:

```
 rates←2.51 2.79 2.96 3.29 3.43 3.71 3.92 4.14 4.64
```

```
      maturities forward rates
2.51 3.07 3.13 3.62 3.71 4.13 4.445 4.6533 5.14
```

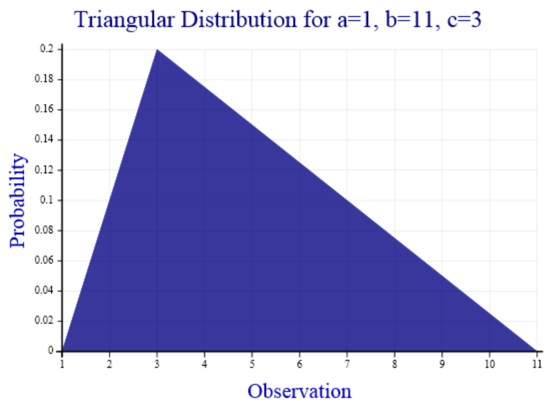
Note: the first forward rate is the same as the first spot rate; the second forward rate uses the first and second spot rates and the first and second maturities; the third forward rate uses the second and third values, etc.

[1] (Source: Hull, John C. *Options, Futures and Other Derivatives, 5th Edition.* P. 93 Prentice-Hall, 2003. page 99)

Finance Problem 2 – (2 tasks)

Triangular Distribution

Wikipedia defines triangular distribution as follows:
In probability theory and statistics, the triangular distribution is a continuous probability distribution with lower limit **a**, upper limit **b** and mode **c**, where **a < b** and **a ≤ c ≤ b**.
A probability density function (PDF), or density of a continuous random variable, is a function that describes the relative likelihood for this random variable to take on a given value.



Produced using Dyalog APL Chart Wizard

Task 1 – What's the density?

Write an APL function named **`pdf`** which implements the probability density function a triangular distribution and which:

- takes a single numeric right argument
- takes a 3-element numeric vector left argument describing the triangular distribution (representing **a**, **b**, and **c** respectively)
- returns the value of the probability density function for the triangular distribution described by the right argument . The value should be between 0 and 1.

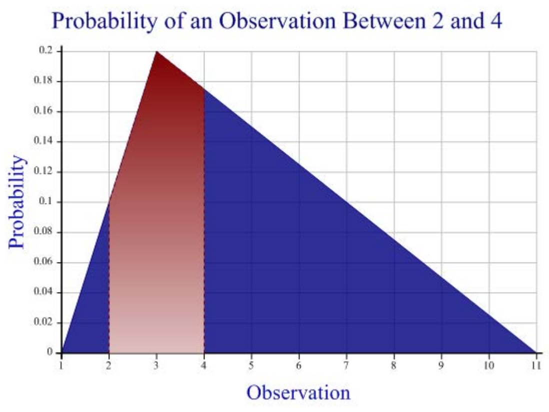
Example:

```
      1 11 3 pdf 3  ⍝ what's the probability density at 3?
0.2

      'X' 'P(X)'⍪(⍳11),⍪1 11 3∘pdf¨⍳11    ⍝ probability density table for integer
values
 X  P(X)
 1 0
 2 0.1
 3 0.2
 4 0.175
 5 0.15
 6 0.125
 7 0.1
 8 0.075
 9 0.05
10 0.025
11 0
```

Task 2 – ***What's the probability?***

Suppose we want to find the probability of an observation within a range. For example, what's the probability of an observation between 2 and 4 occurring?



Produced using Dyalog APL Chart Wizard

Write an APL function named **pdf2** which:

- takes either a single numeric right argument or a 2-element numeric vector representing an observation or a range of observations.
- takes a 3-element numeric vector left argument representing **a**, **b**, and **c** respectively
- returns the probability (expressed as a number between 0 and 1) that a value in the right argument will occur in the distribution described by the left argument

Example:

```
      1 11 3 pdf2 3    ⍝ what's the probability density at 3?
0.2

      1 11 3 pdf2 2 4  ⍝ what's the probability a value between 2 and 4 will occur?
0.3375
```

Finance Problem 3 – (1 task)

Take a Trip to Monte Carlo

Monte Carlo methods in finance are often used to evaluate investments in projects at a business unit or corporate level, or to evaluate financial derivatives. They can be used to model project schedules, where simulations aggregate estimates for worst-case, best-case, and most likely durations for each task to determine outcomes for the overall project. Monte Carlo methods are also used in option pricing and default risk analysis.[2]

This problem uses Monte Carlo methods to evaluate the proposed release of a new product.

The profitability of a product can be oversimplified to:

$$\mathbf{profit} = \left(\mathbf{unitsSold} \times (\mathbf{unitPrice} - \mathbf{unitCost})\right) - \mathbf{fixedCosts}$$

In our model:

- The number of units sold and the price per unit can vary based on the economic climate.
- Unit cost can vary based on suppliers, labour costs, etc.
- Fixed costs are known and don't have any variability.

Our sales forecasters have estimated the following scenarios over the expected lifetime of the product:

Economic Climate	**Likelihood**	**unitsSold**	**unitPrice**
Recession	10%	50,000	$13.00
Slow	30%	75,000	$10.00
Moderate	40%	100,000	$8.50
Hot	20%	125,000	$8.00

Our production planning department estimates that the **unitCost** will vary from $5.00 to $8.00 with a most likely value of $7.00. This sounds like an excellent opportunity to use triangular distribution.
fixedCosts are known to be $125,000.

[2] Wikipedia

Task 1 – ***Go or No Go?***

Your job is to build a model to determine the likelihood that going forward with the production of this product will be profitable. You should run the simulation for a minimum of 10,000 trials.

Your task is to write an APL function named **`profit`** which:

- returns a 2-column numeric matrix of:
  - `[;1]` profit (or loss) in $10,000 increments
  - `[;2]` the probability expressed as a percentage to 2 decimal places

Example (N.B. the numbers below are made up to show the formatting and are not indicative of actual values):

```
      profit
¯10000  6.12
     0 17.34
 10000 46.91
 20000 20.11
 30000  9.52
```

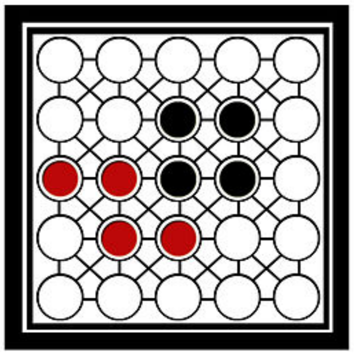
These results would indicate a 6.12% chance of losing $0-$10,000, a 17.34% chance of making $0-$10,000, a 46.91% chance of making $10,000-$20,000, etc.

Note:
You can structure the arguments to the model in any way you choose – for example, global variables, variables defined inline within `profit`, etc.

Set 3: General Computing Problems

General Computing Problem 1 – (1 task)

Teeko



According to Wikipedia, Teeko is an abstract strategy game invented by John Scarne in 1937 and rereleased in refined form in 1952 and again in the 1960s. The Teeko board consists of twenty-five spaces arranged in a five-by-five grid. There are eight markers in a Teeko game, four black and four red. One player, "Black" plays the black markers, and the other, "Red", plays the red. Black moves first and places one marker on any space on the board. Red then places a marker on any unoccupied space; black does the same; and so on until all eight markers are on the board. The object of the game is for either player to win by having all four of his markers in a straight line (vertical, horizontal, or diagonal) or on a square of four adjacent spaces. Adjacency is horizontal, vertical, or diagonal, but does not wrap around the edges of the board. If neither player has won after the "drop" (when all eight pieces are on the board), then they move their pieces one at a time, with Black playing first. A piece may be moved only to an adjacent space.

In APL, a Teeko board can easily be represented as a 5 × 5 matrix of different values for empty, black, and red spaces. For instance the board in the image above could be represented by:

```
      ⊢board←5 5⍴'∘∘∘∘∘∘∘BB∘RRBB∘∘RR∘∘∘∘∘∘∘'
∘∘∘∘∘
∘∘BB∘
RRBB∘
∘RR∘∘
∘∘∘∘∘
```

Task 1 – Do We Have a Winner?

Write an APL function named `winner` which:

- takes a right argument of a 5×5 character matrix representing a Teeko board
- if there is a winner, returns the appropriate character for the winner, otherwise returns the character representing an unoccupied space.

Examples:

Using `board` as defined above:

```
      winner board
B
      ⊢board2←5 5⍴'∘∘∘∘∘∘∘B∘BRRBB∘∘RR∘∘∘∘∘∘∘'
∘∘∘∘∘
∘∘B∘B
RRBB∘
∘RR∘∘
∘∘∘∘∘

      winner board2
∘
```

General Computing Problem 2 – (2 tasks)

Chi-Square Test of Independence

A Chi-Square test for independence is applied when you have two categorical variables from a single population. It is used to determine whether there is a significant association between the two variables.

The Chi-Square test statistic is calculated using the following formula:

$$X = \sum_i \sum_j \frac{\left(OBS_{ij} - EXP_{ij}\right)^2}{EXP_{ij}}$$

The observed values are the number of observations in each category.
The expected values (EXP) are calculated as follows:

$$EXP_{ij} = \frac{\sum_i n_{ij} \sum_j n_{ij}}{\sum_i \sum_j n_{ij}}$$

Where n_{ij} = The number of observations in category i and in category j

For example, in a class of 38 students there are the following counts:

Observed Values	Left	Middle	Right	Total
Female	3	2	4	9
Male	8	9	12	29
Total	11	11	16	38

If the variable GENDER is independent of the variable POLITICS, then the expected counts would be:

Expected Values	Left	Middle	Right	Total
Female	2.6053	2.6053	3.7895	9
Male	8.3947	8.3947	12.211	29
Total	11	11	16	38

Observe that the row totals and column totals remain the same (the displayed numbers are rounded).

The expected values are calculated from the observed values.
For example, the expected number of Left Females is 2.6053:

$$\frac{\boldsymbol{NUMBER\ OF\ LEFTs \times NUMBER\ OF\ FEMALEs}}{\boldsymbol{NUMBER\ IN\ SURVEY}} = \frac{\mathbf{11 \times 9}}{\mathbf{38}} = \mathbf{2.6053}$$

***Task 1* – Meeting Expectations**

Write an APL function named **`expected`** which:

- takes a right argument of a numeric matrix of counts
- returns a numeric matrix of expected values

Example:

```
      ⊢counts←2 3⍴3 2 4 8 9 12
3 2  4
8 9 12
      expected counts
2.605263158 2.605263158  3.789473684
8.394736842 8.394736842 12.21052632
```

***Task 2* – Chi-Square Test**

Write an APL function named **`chiSquareTest`** which:

- takes a right argument of a numeric matrix of counts
- returns the Chi-square test statistic

Example:

```
      ⊢counts←2 3⍴3 2 4 8 9 12
3 2  4
8 9 12
      chiSquareTest counts
0.2779519331
```

General Computing Problem 3 – What's in a Name? (1 task)

For this problem, imagine that you work in the pharmaceutical industry. Your company has a database which consists of a list of normalized drug names. Your company also collects information from physicians regarding the drugs they are prescribing. Ideally, all of the names in the list from physicians would be found in the database of normalized names. However, data entry errors can occur or the physician's office may enter additional, unnecessary, information – a dosage amount in addition to the drug name for instance. Therefore there are a number of entries that do not match exactly. The mismatched names need to be analyzed and matched, if possible, with the database names. Identifying and correcting the mismatches by hand is a costly process. As such, you've been tasked with automating as much of the process as possible.

The two lists are found in text files included in the zip file downloaded from the competition web site. The database data is found in **/data/drugs.txt** and the input data is found in **/data/inputs.txt** in the folder to which you unzipped the competition files.

```
database←{(⎕NUNTIE ⍵)⊢('.'⎕R'\u0'⎕OPT'Mode' 'L')⍵}'unzipped_path/data/drugs.txt' ⎕NTIE 0
inputs←{(⎕NUNTIE ⍵)⊢('.'⎕R'\u0'⎕OPT'Mode' 'L')⍵}'unzipped_path/data/inputs.txt' ⎕NTIE 0
```

You can copy and paste the above statements into your Dyalog APL session and execute them after replacing "unzipped_path" with the actual path name where you unzipped the competition files.
The statements will (for their respective text files):

- tie (open) the text file (using `⎕NTIE`)
- read the contents of the file, splitting each line into a separate element (`⎕OPT 'Mode' 'L'`)
- normalize the elements by making them uppercase (`'.'⎕R'\u0'`)
- untie (close) the text file (using `⎕NUNTIE`)
- return the normalized elements (`⊢`)

Pretty cool, huh?

If you use another APL to solve this problem, you can use whatever tools are available in that environment to accomplish the same.

The sample data files contain 2729 drug names and 4400 inputs.

Task 1 – Match Maker

Write an APL function named `matchDrugs` which:

- takes a right argument representing the list of physician inputs
- takes a left argument representing the list of database drug names
- returns an integer vector where each element is index in the list of drug names representing the best match for the corresponding element in the list of inputs.

Example:

```
      d←'ASPIRIN' 'METAMUCIL' 'PENICILLIN'
      i←'PENICILIN' 'ASPIRIN 250MG' 'METAMUSEL' 'ASPRIN'
      d matchDrugs i
3 1 2 1
```