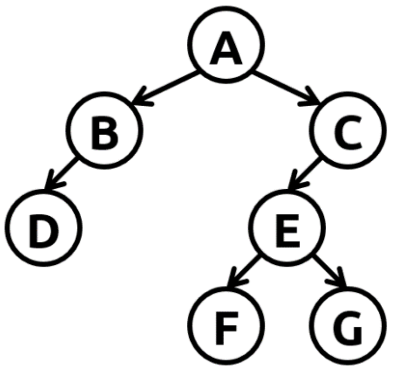
CS 602
Algorithm Design and Implementation

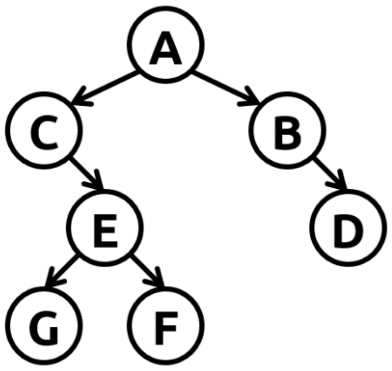
Assignment 2

[Question 1] Height of Symmetric Binary Trees

A symmetric binary tree is a tree which is the same as its mirrored binary tree at its root. The mirrored binary tree can be defined as: (1) the mirrored binary tree of the empty tree is the empty tree; (2) the left subtree of the mirrored binary tree is the mirrored binary tree of the original right subtree (subtree rooted at the right child of the root), and the right subtree of the mirrored binary tree is the mirrored binary tree of the original left subtree (subtree rooted at the left child of the root). In addition to symmetric binary trees, we can also define a recursive symmetric binary tree as: (1) an empty tree is a recursive symmetric tree; or (2) a symmetric binary tree with both left subtree and right subtree being recursive symmetric trees.



A binary tree T



The mirrored binary tree of T

Given a binary tree, compute both the height of tallest symmetric binary subtrees and the height of tallest recursive symmetric binary subtrees.

Inputs begin with the number of lines below. Each line describes a binary tree by a list of tree nodes with values separated by ‘,’, where values are between 1 and 10000. Two consecutive ‘,’ represent a null node in between. To simplify the tree structure, the length of node is always $2^{h+1} - 1$ for a tree of height h. All null nodes are indicated, even if they are descendants of another null node. For each line of input, output two numbers h_1 and h_2 where h_1 is the height of tallest symmetric binary subtrees and h_2 is the height of tallest recursive symmetric binary subtrees.

There will be 8 testcases with 1 mark each. The remaining 2 marks are awarded with the correct justification of the time complexity of your algorithm.

Sample Input

[Please refer to the file A2Q1.in.]

Sample Output

```
1 1
3 3
2 1
```

[Question 2] Metal Melter

In the city of Temapura, a metal recycling company is specialized in melting metal blocks of the same type but different sizes together, so metals can be recycled and resold. Every step, a machine, called metal melter, takes two blocks from a pool and puts the melted block back to the pool. The melting processes cost energy, which is equal to the sum of the weight of the two blocks in the unit of kilojoule. Given a pool of metal blocks, the machine operator can choose any sequence to melt the blocks and put the melted block back to the pool, until there is only one block left in the pool. For example, if a pool of blocks of weight 1kg, 2kg and 4kg is given, the operator can choose to melt 1kg and 2kg first, or to melt 1kg and 4kg first, or to melt 2kg and 4kg first. If he melts 1kg and 2kg first, 3 kilojoules are spent to produce a 3kg block, and then melts 3kg and 4kg together, which costs 7 kilojoules. A total of 3 + 7 = 10 kilojoules is spent on this whole process. If he starts with melting 2kg and 4kg first, the machine spends 6 kilojoules in the first step, and 7 kilojoules in the second step, which gives 13 kilojoules in total. The company would like to minimize the energy spent in the melting process, the first option is thus chosen. However, they do not know how to decide on the melting sequence and need your help.

Implement an algorithm with appropriate data structure for the function `metal_melter`.

Test inputs begin with the number of lines. Each line contains a list of integer weights of metal blocks. The length of each list is between 2 and 5000, and the total weight is at most 5×10^5. For each list, output one single integer indicating the minimum possible amount of energy to spend in kilojoules. You may import Python libraries which are commonly used, but you are not supposed to ask what the right library is, as this question tests you on choosing the appropriate data structure. Instead, you may submit the skeleton code with "import xxx" to test if library xxx is included in the online judge (If it is included, you'll get WA for the skeleton code).

Sample Input

```
2
2 4 1
6 5 4 7
```

Sample Output

```
10
44
```

[Question 3] Tabu Bigrams

A bigram is a sequence of two adjacent digits from an integer. For example, 1415926 contains 14, 41, 15, 59, 92, 26 as its bigrams. Given a list of tabu bigrams, an integer is said tabu-free if none of its bigrams are in the list. How many such tabu-free integers of *n* ($1 \leq n \leq 100$) digits in based *m* ($1 \leq m \leq 36$) are there?

Implement an algorithm as the function `count_tabu_free`.

Test inputs begin with the number of lines. Each line contains a pair integers *m* and *n*, followed by the list of tabu bigrams. For *m* > 10, digits beyond 9 are represented by lower-case letters from a to z. Thus, it is possible that some tabu bigrams contains letters. However, letters with their Unicode value greater than *m*+86 do not appear in the bigrams. Likewise, when $m \leq 10$, the bigrams will only contain digits between 0 and *m*-1 inclusive. In short, the tabu bigrams are all valid with base *m*. In addition, please take note that the first digit of any *n*-digit integer cannot be 0.

Sample Input

```
6
4 1 11 12 23
4 2 11 12 23
8 6 30 04 25 13
12 4 0a 1a 2a 3a 4a 5a 6a 7a 8a 9a bb
15 8 ab ac ad ae ba bc bd be ca cb cd ce
10 10 00 11 22 33 44 55 66 77 88 99
```

Sample Output

```
3
9
167899
14796
1667482542
3486784401
```

Running python skeleton with sample input:

1. Open “Anaconda Prompt”
2. Go to the directory where you put the file **A2Q1.py** and **A2Q1.in**, using command `cd`
3. Run command **python A2Q1.py < A2Q1.in**
4. You may want to create a test input called **my_own_test.in** to design a test case for your own program, the command would then be **python A2Q1.py < my_own_test.in**
5. Same applies to Question 2, so you may run **python A2Q2.py < A2Q2.in**