CSCI 2100B
Data Structures

Homework Problems Version 1.17

---

1 Algorithm Analysis

Exercise 1.1 *Evaluate the following series.*

1. $\sum_{i=0}^{\infty} \frac{1}{4^i}$.
2. $\sum_{i=0}^{\infty} \frac{i}{4^i}$.
3. $\sum_{i=0}^{\infty} \frac{i^2}{4^i}$.
4. $\sum_{i=1}^{n} i$
5. $\sum_{i=1}^{n} a^i$
6. $\sum_{i=1}^{n} ia^i$
7. $\sum_{i=1}^{k} 2^{k-i}i^2$
8. $\sum_{i=0}^{n} i^2$.
9. $\sum_{i=0}^{n} i^3$
10. *(**)* $x_n = \frac{2}{n}\sum_{i=0}^{n-1} x_i + an + b$ *where* $a > 0$ *and* $b > 0$.
11. *Show that for any real constants* a *and* $b, b > 0$*, that* $(n+a)^b = \Theta(n^b)$.
12. *Is* $2^{n+1} = O(2^n)$*?*
13. *Is* $2^{2n} = O(2^n)$*?*

Exercise 1.2 *About the GCD function.*

1. *Is the GCD function distributive? Associative? Commutative?*
2. *Show that* $GCD(ma, mb) = mGCD(a, b)$.
3. *If* $GCD(a, b) = p$ *and* $GCD(c, d) = q$*, is* $GCD(ac, bd) = pq$ *true for all the* a, b, c, d*? Either prove it or give a counterexample.*
4. *Show that* $GCD(2^p - 1, 2^q - 1) = 2^{GCD(p,q)} - 1$.

Exercise 1.3 *Evaluate the following recurrence relations.*

1. $T(n) = aT(n-1) + bn, T(1) = 1$
2. $T(n) = T(n/2) + bn\log n, T(1) = 1$

3. $T(n) = aT(n-1) + bn^c, T(1) = 1$

4. $T(n) = aT(n/2) + bn^c, T(1) = 1$

5. *Solve* $x_n = x_{n-1} - \frac{1}{4}x_{n-2}$, *with* $x_0 = 1, x_1 = 1/2$.

6. *(**) Solve* $x_n = x_{n-1} - \frac{1}{4}x_{n-2} + 2^{-n}$

7. *(**) Solve* $T(n) = 2T(\sqrt{n}) + \log_2 n$ *Hint: make a change of variables* $m = \log_2 n$. *Only worry about numbers* $2^i, i = 2, 4, 8, 16,$ *etc. with* $T(2) = 1$.

8. *Solve* $T(1) = 1$, *and for all* $n \geq 2, T(n) = 3T(n-1) + 2$.

9. *Solve* $T(1) = 1$, *and for all* $n \geq 2$ *a power of 2,* $T(n) = 2T(n/2) + 6n - 1$.

Exercise 1.4 *Prove the following statements.*

1. *Prove by induction on* $n \geq 1$ *that* $\sum_{i=1}^{n} 1/2^i = 1 - 1/2^n$.

2. *Prove by induction on* $n \geq 0$ *that* $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$.

3. *Prove* $\sum_{i=1}^{n}(2i-1) = n^2$.

4. *Prove that* $(n+1)^2 = O(n^2)$.

5. *Prove* $2\lg(n!) > n \lg n$ *by using Induction, where* n *is a positive integer greater than 2.*

6. *The number generated by the formula* $n^2 + n + 17$ *is prime for* $n \geq 0$, *where* n *is an integer. Either prove it or disprove it by counterexample.*

7. *Proof that the set of all real numbers between 0 and 1 is uncountable.*

8. *Prove the solution to* $T_n = 2T_{n-1} + 1, T_0 = 0$ *is* $T_n = 2^n - 1$.

Exercise 1.5 *Find the smallest integer value of* n *for which the column entry becomes larger than the row entries in a table for the following growth orders,* $10, 5.6n, 0.25n^2, 0.1n^3, 0.001\ 2^n, n\log n,$ *and* $100\ \log n$. *Use base 2 for the* log *operation. Convert the entries into the big-O notation and then rank them in a non-decreasing order. Produce a table as follows.*

	10	$5.6n$	$0.25n^2$	$0.1n^3$	$0.001\ 2^n$	$n\log n$	$100\ \log n$
10							
$5.6n$							
$0.25n^2$							
$0.1n^3$							
$0.001\ 2^n$							
$n\log n$							
$100\ \log n$							

Exercise 1.6 *Analyze the following* `for-next` *loop statements and give (1) the precise* $f(n)$ *with* n *being the input and (2) the big-O notation for each expression, i.e.,* $g(n)$.

1.
```
for i = 1 to n;
  for j = 1 to n;
    x := x + 1;
```

2.
```
for i = 1 to n;
  for j = i to n;
    x := x + 1;
```

3.
```
for i = 1 to n;
  for j = i to n;
   for k = 1 to j;
    x := x + 1;
```

4.
```
for i = 1 to n;
  for j = i to n;
   for k = 1 to i*n;
    x := x + 1;
```

5.
```
for i = 1 to n;
  for j = i to n;
   for k = 1 to 1000;
    x := x + 1;
```

6.
```
for i = 1 to n-1;
  for j = i+1 to n;
   for k = 1 to j;
    x := x + 1;
```

7.
```
for i = 1 to n;
  for j = 1 to i;
   for k = j to i+j;
    for l = 1 to i+j-k;
     x := x + 1;
```

Exercise 1.7 *Aside from measuring performance from a point of computational efficiency, other common measures of performance include fault-tolerance, reliability, security, cost, cost/performance ratio, accuracy, and robustness to user error. Perform the following tasks:*

1. *Define each term in the context for performance analysis. (Hint: use the Internet to help you find the definition. Remember to reference the source.)*

2. *Discuss how each of these problems can be attacked at several design levels.*

Exercise 1.8 *Consider the following algorithm to evaluate* $f(x) = \sum_{i=0}^{n} a_i x^i$ *(Horner's rule):*

poly = 0;
for(i=n; i>=0; i - -)
  $poly = x * poly + a_i$

1. *Show how the steps are performed by this algorithm for* $x = 3, f(x) = 4x^4 + 8x^3 + x + 2$.
2. *Explain why this algorithm works.*
3. *What is the running time of this algorithm with the following assumptions? What is its big-O notation?*

Statement	**Time Unit**
assignment	*1*
$+$	*1.25*
$*$	*1.75*
for-next loop set-up	*2.3*
each loop	*1.5*

Exercise 1.9 *Given* $1 \leq n \leq 100$ *as the input parameter, two algorithms are available to calculate a function with the following time and space complexity as shown in Table 1.*

Table 1: Time and Space Complexity of Algorithms

Algorithm	**Time Complexity**	**Space Complexity**
A	$t(n) = \begin{cases} n^2 & \text{if } 1 \leq n < 10 \\ n & \text{if } 10 \leq n < 50 \\ n^3 & \text{if } 50 \leq n \leq 100 \end{cases}$	$s(n) = \begin{cases} n & \text{if } 1 \leq n < 20 \\ 1.5n & \text{if } 20 \leq n \leq 100 \end{cases}$
B	$t(n) = \begin{cases} n & \text{if } 1 \leq n < 30 \\ n^2 & \text{if } 30 \leq n < 70 \\ n^3 & \text{if } 70 \leq n \leq 100 \end{cases}$	$s(n) = \begin{cases} 5n & \text{if } 1 \leq n < 50 \\ 0.5n & \text{if } 50 \leq n \leq 100 \end{cases}$

1. *Plot the time and space complexity function for algorithm* A *and* B *for* $1 \leq n \leq 100$.
2. *Calculate the time and space complexity for* $n = 10, 20, 30, 50, 70,$ *and* 100 *for each algorithm.*
3. *If we define the total cost,* $C(n)$*, of the algorithm as:*

$$C(n) = t(n) + 5 * s(n).$$

   Now calculate the average cost for each of the two algorithms. Which one is the better algorithm? By how much?
4. *Come up with a strategy that you would use to minimize the time and space complexity individually? (Hint: you can have a hybrid function where you can define which algorithm to use for a given* n*.)*
5. *Now calculate the total cost for the hybrid algorithm (assuming no overhead) that you have formulated in the previous question. How much better is the hybrid algorithm than algorithm* A *and* B *by itself.*

Exercise 1.10 *One day in January 2006, Prof. King asks his tutors to do summation of integers in different input formats, but they are so poor and don't know how to do it. You, as the best friends of them, are experts on C, and want to help these poor guys.*

Input and Output *There are several test cases, each test case has different kind of input formats. It starts with a format ID, and the ID can be A, B, C, or D.*

For format $ID = A$, there is a single line consisting of four non-negative integers separated by white spaces, output the summation of them in a line.

For format $ID = B$, there is a single line consisting of two positive integers, $1 \leq C \leq 100$ and $1 \leq D \leq 100$ separated by white spaces. In the next C lines, each line has $C + D$ integers separated by white spaces. For each line, output the summation of the $C + D$ integers in a line.

For format $ID = C$, there are more than one line. Each line starts with a positive integer $1 \leq E \leq 100$, and then followed by E integers separated by white spaces. For each line (except the line for $E = 101$), output the summation of the E integers in a line. This test case will end when $E = 101$.

For format $ID = D$, there are more than one line. Each line has at least one integer and at most 100 integers that separated by white spaces. For each line (except for the line beginning with 0), if number of integers is even, output the summation of the integers in a line; if number of integers is odd, output the number of integers. This test case will end when a line begins with 0.

In this program, you don't need to worry about the overflow problem.

Sample Input

```
A
0 1 0 1
B
2 2
1 1 1 1
0 2 2 2
C
2 1 2
5 1 0 0 0 2
101
D
1 1 2
2 0 2 0
0
A
2 3 4 5
```

Sample Output

```
2
```

```
4
6
3
3
3
4
14
```

Exercise 1.11 *Write a program to calculate $F_n(x, y)$ according to the following definition:*

$$F_n(x,y) = \begin{cases} x+y & \text{if } n = 0 \\ x & \text{if } y = 0 \\ F_{n-1}(F_n(x, y-1), F_n(x, y-1) + y) & \text{otherwise.} \end{cases} \quad (1)$$

Input

The input consists of the number of test cases, m, in the first line and followed by m lines of three non-negative integers $n(0 \leq n \leq 5), x(0 \leq x \leq 5)$, and $y(0 \leq y \leq 5)$ separated by a space as the inputs to $F_n(x, y)$.

For example,

```
3
0 3 1
3 2 0
1 3 1
```

Output

The output should be m lines of Fn(x, y).

```
4
2
7
```

Exercise 1.12 *One day, Prof. King asks his tutors (chlaw, Gordon and Yousef) to do summation of integers in different input formats, but they are so poor and don't know how to do it. You, as the best friends of them, are experts on C, and want to help these two poor guys.*

Input and Output *There are several test cases, each test case has different kind of input formats. It starts with a format ID, and the ID can be 1, 2, 3, or 4.*

For format $ID = 1$, there is a single line consisting of two positive integers separated by white spaces, output the summation of them in a line.

For format $ID = 2$, there is a single line consisting of two positive integers, $1 \leq C \leq 100$ and $1 \leq D \leq 100$ separated by white spaces. In the next C lines, each line has D integers separated by white spaces. For each line, output the summation of the D integers in a line.

For format $ID = 3$, there are more than one line. Each line starts with a positive integer $1 \leq E \leq 100$, and then followed by E integers separated by white spaces. For each line (except the line for $E = 0$), output the summation of the E integers in a line. This test case will end when $E = 0$.

For format $ID = 4$, there are more than one line. Each line has at least one integer and at most 100 integers that separated by white spaces. For each line, output the summation of the integers in a line (except for the line beginning with 0). This test case will end when a line begins with 0.

In this program, you don't need to worry about the overflow problem.

Sample Input

```
1
0 1
2
2 2
1 1
0 2
3
2 1 2
5 1 0 0 0 2
0
4
1 1 2
2 0 2 0
0
1
2 3
```

Sample Output

```
1
2
2
```

3
3
4
4
5

Exercise 1.13 *A k-th order linear recurrence with constant coefficients defines a series as*

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + c_{k+1},$$

where $c_1, \cdots, c_{k+1}$ are real numbers. Write a program that calculate such recurrence equation. Write a program that with input $k, a_1, \cdots, a_k, c_1, \cdots, c_{k+1}$, and N produces the output $a_1, \cdots, a_N$.

Input *The input consists of the number of test cases, m, in the first line and followed by m lines inputs. Each line consists of $N, k, a_1, \cdots, a_k, c_1, \cdots, c_{k+1}$. For example,*

```
2
2 2 10 10 5 5 5
1 3 1 2 3 4 5 6 7
```

Output *The output should be in m groups and a total of $m \times (N + 1)$ lines of numbers.*

Exercise 1.14 *Given a pair of integers. Calculate the summation and subtraction of these two integers.*

Input *The input consists of the number of test cases, m, in the first line and followed by m groups of 4 lines as inputs. The group consists of a symbol, either "+" or "−", two lines of integers followed by a carriage return. The integer can have 100 digits and can also be negative. An example is as follows,*

```
2
+
123456
111111

-
0
-10
```

Output *The output should be m lines of numbers. Each line should be the summation or the difference of the two integers.*

```
234567
10
```

Exercise 1.15 *Given a pair of non-negative integers between 0 and 65535. Find the number of bits that are different in their respective binary representation. For example, 3 in decimal is equivalent to 0000000000000011 in binary and 1 in decimal is equivalent to 0000000000000001 in binary so that the number of bits that are different in these two binary patterns is 1.*

Input *The input consists of the number of test cases, m, in the first line and followed by m lines of two positive integers as inputs. For example,*

```
3
1 3
100 100
65535 0
```

Output *The output should be m lines of numbers.*

```
1
0
16
```

Exercise 1.16 *The Ackermann function is the simplest example of a well-defined Total Function which is Computable but not Primitive Recursive, providing a counterexample to the belief in the early 1900s that every Computable Function was also Primitive Recursive. It grows faster than an exponential function, or even a multiple exponential function. The Ackermann function $A(x, y)$ is defined by*

$$A(x,y) = \begin{cases} y+1 & \textit{If } x = 0 \\ A(x-1,1) & \textit{If } y = 0 \\ A(x-1, A(x, y-1)) & \textit{Otherwise.} \end{cases}$$

Write a program to calculate the Ackermann function. How large of x, y can you calculate on your computer?

Input *The input consists of the number of test cases, m, in the first line and followed by m lines of two positive integers x and y separated by a space as the inputs to $A(x, y)$. For example,*

```
3
3 0
0 3
3 5
```

Output *The output should be m lines of $A(x, y)$.*

```
5
4
253
```

Exercise 1.17 *A prime number is a Positive Integer $p > 1$ which has no positive integer Divisors other than 1 and p itself. (More concisely, a prime number p is a positive integer having exactly one positive divisor other than 1.) For example, the only divisors of 13 are 1 and 13, making 13 a prime number, while the number 24 has divisors 1, 2, 3, 4, 6, 8, 12, and 24 (corresponding to the factorization $24 = 2^3 \times 3$), making 24 not a prime number. Positive Integers other than 1 which are not prime are called "Composite".*

Although the number 1 used to be considered a prime, it requires special treatment in so many definitions and applications involving primes greater than or equal to 2 that it is usually placed into a class of its own. Since 2 is the only Even prime, it is also somewhat special, so the set of all primes excluding 2 is called the "Odd Primes." The first few primes are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, ... Moreover, let the function $\pi(n)$ returns the n-th prime number, e.g., $\pi(1) = 2, \pi(3) = 5$, etc.

The Fundamental Theorem of Arithmetic states that any Positive Integer can be represented in exactly one way as a Product of primes. Euclid's Second Theorem demonstrated that there are an infinite number of primes. Here are some prime-related problems that have not been solved:

1. *It is not known if there are an infinite number of primes of the form $n^2 + 1$.*
2. *Whether there are an Infinite number of Twin Primes, e.g., $\pi(n) - \pi(n+1) = 2$ so $\pi(2)$ and $\pi(3)$ are twin primes.*
3. *Whether a prime can always be found between n^2 and $(n+1)^2$.*

Write a program to calculate $\pi(n)$ for $n < 50$.

Input *The input consists of the number of test cases, m, in the first line and followed by m lines of numbers as input to $\pi(n)$. For example,*

```
4
1
6
2
10
```

Output *The output should be m lines of $\pi(n)$.*

```
2
13
3
29
```

Exercise 1.18 *Given a series of integers, calculate the summation of the integers which are in the odd positions and even positions respectively.*

Input *The input consists of the number of test cases, m, in the first line and followed by m test cases.*

In each test case, the first line is n, it means the number of the integers, then it is followed by n integers. n is at most 20.

An example is as follows,

```
2
5
-24
233
2147480012
-3
2147480012
2
0
-10
```

Output *The output should be m lines of numbers. Each line should be the summations of the integers which are in the odd positions and even positions in each case. Each two integers in each line should be separated by a space.*

```
4294960000 230
0 -10
```

Exercise 1.19 *Given a string, among all the letters appeared in the string, we define n_{max} as the largest number of occurence and n_{min} as the smallest number of occurence. If $n_{max} - n_{min}$ is a prime, we say that we have found a "Lucky String". For example, in the string of "givemeanexample", 'e' occurs the most times, which is 4 and 'g' occurs the least times, which is 1. Then $n_{max} - n_{min}$ equals to 3, which is a prime. Thus, this string is a "Lucky String".*

Input and Output *The input consists of the number of strings, n, in the first line and followed by n strings in separate lines. Each string will only consists of lowercase letters and not exceed 256 in length.*

The output should have n lines. Each line should be "YES" or "NO" to indicate whether the corresponding string is a "Lucky String".

Sample Input

```
3
itisasimpleproblem
butpleasebecareful
wishugoodluck
```

Sample Output

```
YES
YES
NO
```

Exercise 1.20 *Given some lines of strings, you are asked to delete all the digits in the strings and print other characters in their original order.*

Input *There are multiple test cases. The first line of input is an integer* T *(*$1 \leq T \leq 10$*) indicating the number of test cases. Each case contains a line indicating a string. The length of the string is between* 1 *and* 30*, inclusive. Besides, the string consists of only digits and English letters.*

Output *For each string, delete all the digits and print the remaining characters in a line.*

Sample Input

```
2
CSCI2100BDataStructure
1plus1equals2
```

Sample Output

```
CSCIBDataStructure
plusequals
```

2 Lists, Queues, and Stacks

Exercise 2.1 *Write the routines to implement queues using*

1. *linked lists with both links to descendants*
2. *linked lists with one link to descendant and the other to a sibling*
3. *arrays*

Exercise 2.2 *Unfortunately, one of the CSC2100B tutor's calculator broke down last week. Now he is left with his computer, which has no calculator application (he use MS-DOS 6.2, the most powerful dos version!), while paper and pencil is too tiresome for a CSE student. But fortunately, he can give a programming assignment for his student to write him a calculator application! He laughs in his office: "HA Ha ha ha...."*.

Input *The input consists of these rules.*

1. *Each line will contain* **exact one valid** *expression.*
2. *The expression will contain non-negative number only (include integer and floating point number), so there will be no + and − sign for the number.*
3. *The expression will contain operators "$+, -, *, /, (,)$", in which "$+, -$" have the lowest priority, "$*, /$" have higher priority, and "$(,)$" have the highest priority.*
4. *Here is the recursive definition of expression:*

   *Exp → (Exp) | Exp + Exp | Exp - Exp | Exp * Exp | Exp / Exp*

   Exp → non-negative number
5. *Don't need to do rounding or truncation during the calculation*
6. *You don't need to worry about the overflow or division by zero problem, and also floating point error.*

Output *For each line of expression, output the result with 4 decimal place in one line.*

Sample Input

```
1 + 1
1 + (1*(0-1)) / 1
1/3 + 0.5 + (2e-5)
```

Sample Output

```
2.0000
0.0000
0.8334
```

Exercise 2.3 *In the beginning of each semester, CSE department has many add/drop records, and the records are managed by manpower only. But this semester is really troublesome, as there are too many add/drop in the class CSC2100B, so department thinks that it is better to do it by a program. You, as the chief programmer in the CSE department, are responsible to do this job.*

Input *Each line will contain a command. There will be only 4 kinds of command, they are:*

ADD S C - add student S into course C

DROP S C - drop student S from course C

PRINTS S - print out all the course that student S takes in the recent status

PRINTC C - print out all the student that take course C in the recent status

where $1 \leq S \leq 1,000,000$ *and* $1 \leq C \leq 1,000,000$. *On average, each student will take 4 courses. Most of the commands are "ADD" and "DROP", only a few of them are "PRINTS" and "PRINTC". Please use an effective data structures and efficient operations to implement this program! (Hint: linked-list is helpful)*

Output *For the command "PRINTS" and "PRINTC", print out the numbers in ascending order with space separation in a line. If there is nothing in the list, just print out "NIL" in a line.*

Sample Input

```
ADD 3 1
ADD 1 1
ADD 2 1
ADD 2 1
PRINTC 1
PRINTC 2
ADD 1 2
ADD 2 2
PRINTS 1
DROP 2 1
DROP 4 4
PRINTS 4
PRINTS 2
PRINTC 2
```

Sample Output

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

Exercise 2.4 *Given a pair of integers. Calculate the integer multiplication and division of these two integers using the linked list data structure. Use the Division Theorem which states that the remainder should be positive and less than the absolute value of the divisor.*

Input *The input consists of the number of test cases, m, in the first line and followed by m groups of 4 lines as inputs. The group consists of a symbol, either "∗" or "/", two lines of integers followed by a carriage return. The integer can have 100 digits and can also be negative. An example is as follows,*

```
4
*
123456789123456789
111111111111

/
523
-10

/
-523
-10

/
10
0
```

Output *The output should be m groups of numbers. Each group should be the multiplication or the division of the two integers. For the integer division, you should include the quotient and the remainder.*

```
13717421013703703578986282579

-52 3

53 7

Error: division by zero.
```

Exercise 2.5 *Given two sorted lists, L_1 and L_2, write a routine to compute $L_1 \cap L_2$ and $L_1 \cup L_2$ using only the queue data structure based on linked lists. You should use the queue operations such as* `enqueue` *and* `dequeue` *for the insertion and deletion operations. Indicate clearly in your source code where you are using the queue data structure based on linked lists.*

Input *The input consists of the number of test cases, m, in the first line and followed by m groups of an operator and two lists of sorted integers in a strictly increasing order. The operator is either "+" for union or "-" for the intersection of the two lists. The lists will be followed by a blank line separating the lists. The list may contain up to 1,000 elements.*

```
2
+

-1
3
5

-2
4
6

-

-10
0
1
2
3

0
2
4
6
```

Output *The output should be m groups integers in a non-decreasing order followed by a blank line. There will not be an empty list after the intersect operation.*

```
-2
-1
3
4
5
6

0
2
```

Exercise 2.6 ***Knight Moves*** *(c.f.* `http://acm.uva.es/p/v4/439.html`*)*

A friend of you is doing research on the Traveling Knight Problem (TKP) where you are to find the shortest closed tour of knight moves that visits each square of a given set of n squares on a chessboard exactly once. He thinks that the most difficult part of the problem is determining the smallest number of knight moves between two given squares and that, once you have accomplished this, finding the tour would be easy.

Of course you know that it is vice versa. So you offer him to write a program that solves the "difficult" part.

Your job is to write a program that takes two squares a and b as input and then determines the number of knight moves on a shortest route from a to b.

Input Specification *The input file will contain one or more test cases. Each test case consists of one line containing two squares separated by one space. A square is a string consisting of a letter (a-h) representing the column and a digit (1-8) representing the row on the chessboard.*

Output Specification *For each test case, print one line saying "To get from xx to yy takes n knight moves.".*

Sample Input

```
e2 e4
a1 b2
b2 c3
a1 h8
a1 h7
h8 a1
b1 c3
f6 f6
```

Sample Output

```
To get from e2 to e4 takes 2 knight moves.
To get from a1 to b2 takes 4 knight moves.
To get from b2 to c3 takes 2 knight moves.
To get from a1 to h8 takes 6 knight moves.
To get from a1 to h7 takes 5 knight moves.
To get from h8 to a1 takes 6 knight moves.
To get from b1 to c3 takes 1 knight moves.
To get from f6 to f6 takes 0 knight moves.
```

Exercise 2.7 *You are given a text file containing the typical alphanumeric symbols and the following set of special symbols: (,), [,], {, }, /, and *. Your task is to write a program that will read in the file (less than 1k in length) and check for the following balancing symbols:* `begin end`*, (), [], {}, /* */ and print out the proper error message for each type of error. Each of the token should be separated by a white space and embedding symbols do not count as balancing symbols. For example,* '`begin[`' *and* '`begin [`' *are different. The first is not a pair of balancing symbols, but a single token. The second consists of the balancing symbols of* '`begin`' *and* '`[`'*.*

You should use the stack operations such as `push` *and* `pop`*. Indicate clearly in your source code where you are using the stack data structure based on array.*

For each line of the file, you should have an output message. If it is an error message, you are only required to report the first error.

Input and Output *The input is a filename which contains multiple lines of expressions that need to be evaluated. Each line contains one expression. The file contains alphanumeric symbols, e.g., "space", "newline", "tab", a-z,A-Z, and 0-9, and also special symbols: (), [], {}, and /* */. For example, the input can be "q4.in". Where "q4.in" is a file which may contain the following possible text.*

Sample Input

```
beGin a b c e ( esdf ) [ a sd ( dsf ) [ we Begin ] ]
beGin a b c e ( esdf  [ ) a sd ( dsf ) [ we Begin ] ]
beGin a b c e ( esdf ) [ a sd ( dsf ) [ we Begin ] ] )
beGin a b c e ( ( ( esdf ) [ a sd ( dsf ) [ we Begin ] ] )
```

Sample Output

```
No error.
Unmatched Error: (.
Error: stack empty.
Error: Too many leftover symbol(s).
```

Exercise 2.8 ***The Blocks Problem***

(c.f. `http://icpcres.ecs.baylor.edu/onlinejudge/` *(Problem Set Vol. I 101))*

Many areas of Computer Science use simple, abstract domains for both analytical and empirical studies. For example, an early AI study of planning and robotics (STRIPS) used a block world in which a robot arm performed tasks involving the manipulation of blocks. In this problem you will model a simple block world under certain rules and constraints. Rather than determine how to achieve a specified state, you will "program" a robotic arm to respond to a limited set of commands.

The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are n blocks on the table (numbered from 0 *to* $n-1$*) with block* b_i *adjacent to block* b_{i+1} *for all* $0 \leq i < n-1$ *as shown in the diagram below:*

0	1	2	3	4	...	$n-1$

Initial Blocks World

The valid commands for the robot arm that manipulates blocks are:

1. *move a onto b*

   where a and b are block numbers, puts block a onto block b after returning any blocks that are stacked on top of blocks a and b to their initial positions.

2. *move a over b*

   where a and b are block numbers, puts block a onto the top of the stack containing block b, after returning any blocks that are stacked on top of block a to their initial positions.

3. *pile a onto b*

   where a and b are block numbers, moves the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto block b. All blocks on top of block b are moved to their initial positions prior to the pile taking place. The blocks stacked above block a retain their order when moved.

4. *pile a over b*

   where a and b are block numbers, puts the pile of blocks consisting of block a, and any blocks that are stacked above block a, onto the top of the stack containing block b. The blocks stacked above block a retain their original order when moved.

5. *quit terminates manipulations in the block world.*

   Any command in which $a = b$ *or in which a and b are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.*

Input *The input begins with an integer n on a line by itself representing the number of blocks in the block world. You may assume that* $0 < n < 25$*. The number of blocks is followed by a sequence of block commands, one command per line. Your program should process all commands until the quit command is encountered. You may assume that all commands will be of the form specified above. There will be no syntactically incorrect commands.*

Output *The output should consist of the final state of the blocks world. Each original block position numbered i (0 <= i < n) where n is the number of blocks) should appear followed immediately by a colon. If there is at least a block on it, the colon must be followed by one space, followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space. Don't put any trailing spaces on a line. There should be one line of output for each block position (i.e., n lines of output where n is the integer on the first line of input).*

Sample Input

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```

Sample Output

```
0: 0
1: 1 9 2 4
2:
3: 3
4:
5: 5 8 7 6
6:
7:
8:
9:
```

Exercise 2.9 ***Web Navigation***
(c.f. `http://acm.pku.edu.cn/JudgeOnline/problem?id=1028`*)*

Standard web browsers contain features to move backward and forward among the pages recently visited. One way to implement these features is to use two stacks to keep track of the pages that can be reached by moving backward and forward. In this problem, you are asked to implement this. The following commands need to be supported:

1. ***BACK****: Push the current page on the top of the forward stack. Pop the page from the top of the backward stack, making it the new current page. If the backward stack is empty, the command is ignored.*
2. ***FORWARD****: Push the current page on the top of the backward stack. Pop the page from the top of the forward stack, making it the new current page. If the forward stack is empty, the command is ignored.*
3. ***VISIT****: Push the current page on the top of the backward stack, and make the URL specified the new current page. The forward stack is emptied.*
4. ***QUIT****: Quit the browser.*

Assume that the browser initially loads the web page at the URL http://www.acm.org/

Input *Input is a sequence of commands. The command keywords BACK, FORWARD, VISIT, and QUIT are all in uppercase. URLs have no whitespace and have at most* 70 *characters. You may assume that no problem instance requires more than* 100 *elements in each stack at any time. The end of input is indicated by the QUIT command.*

Output *For each command other than QUIT, print the URL of the current page after the command is executed if the command is not ignored. Otherwise, print "Ignored". The output for each command should be printed on its own line. No output is produced for the QUIT command.*

Sample Input

```
VISIT http://acm.ashland.edu/
VISIT http://acm.baylor.edu/acmicpc/
BACK
BACK
BACK
FORWARD
VISIT http://www.ibm.com/
BACK
BACK
FORWARD
FORWARD
FORWARD
QUIT
```

Sample Output

```
http://acm.ashland.edu/
http://acm.baylor.edu/acmicpc/
http://acm.ashland.edu/
http://www.acm.org/
Ignored
http://acm.ashland.edu/
http://www.ibm.com/
http://acm.ashland.edu/
http://www.acm.org/
http://acm.ashland.edu/
http://www.ibm.com/
Ignored
```

Exercise 2.10 ***Printer Queue***

(c.f. `http://acm.pku.edu.cn/JudgeOnline/problem?id=3125`*)*

The only printer in the computer science students' union is experiencing an extremely heavy workload. Sometimes there are a hundred jobs in the printer queue and you may have to wait for hours to get a single page of output.

Because some jobs are more important than others, the Hacker General has invented and implemented a simple priority system for the print job queue. Now, each job is assigned a priority between 1 *and* 9 *(with* 9 *being the highest priority, and* 1 *being the lowest), and the printer operates as follows:*

1. *The first job J in queue is taken from the queue.*
2. *If there is some job in the queue with a higher priority than job J, then move J to the end of the queue without printing it.*
3. *Otherwise, print job J (and do not put it back in the queue).*

In this way, all those important muffin recipes that the Hacker General is printing get printed very quickly. Of course, those annoying term papers that others are printing may have to wait for quite some time to get printed, but that's life.

Your problem with the new policy is that it has become quite tricky to determine when your print job will actually be completed. You decide to write a program to figure this out. The program will be given the current queue (as a list of priorities) as well as the position of your job in the queue, and must then calculate how long it will take until your job is printed, assuming that no additional jobs will be added to the queue. To simplify matters, we assume that printing a job always takes exactly one minute, and that adding and removing jobs from the queue is instantaneous.

Input *One line with a positive integer: the number of test cases (at most* 100*). Then for each test case:*

1. *One line with two integers n and m, where n is the number of jobs in the queue ($1 \leq n \leq 100$) and m is the position of your job ($0 \leq m \leq n-1$). The first position in the queue is number* 0*, the second is number* 1*, and so on.*
2. *One line with n integers in the range* 1 *to* 9*, giving the priorities of the jobs in the queue. The first integer gives the priority of the first job, the second integer the priority of the second job, and so on.*

Output *For each test case, print one line with a single integer; the number of minutes until your job is completely printed, assuming that no additional print jobs will arrive.*

Sample Input

```
3
1 0
5
4 2
1 2 3 4
6 0
1 1 9 1 1 1
```

Sample Output

```
1
2
5
```

Exercise 2.11 ***Joseph***

(c.f. `http://acm.pku.edu.cn/JudgeOnline/problem?id=1012`*)*

The Joseph's problem is notoriously known. For those who are not familiar with the original problem: from among n *people, numbered* $1, 2, ..., n$*, standing in circle every* m^{th} *is going to be executed and only the life of the last remaining person will be saved. Joseph was smart enough to choose the position of the last remaining person, thus saving his life to give us the message about the incident. For example when* $n = 6$ *and* $m = 5$ *then the people will be executed in the order* $5, 4, 6, 2, 3$ *and* 1 *will be saved.*

Suppose that there are k *good guys and* k *bad guys. In the circle the first* k *are good guys and the last* k *bad guys. You have to determine such minimal* m *that all the bad guys will be executed before the first good guy.*

Input *The input consists of separate lines containing* k*. The last line in the input contains* 0*. You can suppose that* $0 < k < 14$*.*

Output *The file will consist of separate lines containing* m *corresponding to* k *in the input.*

Sample Input

```
3
4
0
```

Sample Output

```
5
30
```

Exercise 2.12 ***Cube Stacking*** *I*

(c.f. `http://acm.pku.edu.cn/JudgeOnline/problem?id=1988`*)*

Farmer John and Betsy are playing a game with N ($1 \leq N \leq 300$) identical cubes labeled 1 through N. They start with N stacks, each containing a single cube. Farmer John asks Betsy to perform P ($1 \leq P \leq 1,000$) operation. There are two types of operations: moves and counts.

1. *In a move operation, Farmer John asks Bessie to move the stack containing cube X on top of the stack containing cube Y.*
2. *In a count operation, Farmer John asks Bessie to count the number of cubes on the stack with cube X that are under the cube X and report that value.*

Write a program that can verify the results of the game.

Input

1. *Line 1: A single integer, P*
2. *Lines $2...P+1$: Each of these lines describes a legal operation. Line 2 describes the first operation, etc. Each line begins with a 'M' for a move operation or a 'C' for a count operation. For move operations, the line also contains two integers: X and Y. For count operations, the line also contains a single integer: X. Note that the value for N does not appear in the input file. No move operation will request a move a stack onto itself.*

Output *Print the output from each of the count operations in the same order as the input file.*

Sample Input

```
6
M 1 6
C 1
M 2 4
M 2 6
C 3
C 4
```

Sample Output

```
1
0
2
```

Exercise 2.13 ***Cube Stacking*** *II*

(c.f. `http://acm.pku.edu.cn/JudgeOnline/problem?id=1988`*)*

Farmer John and Betsy are playing a game with N *(*$1 \leq N \leq 30,000$*) identical cubes labeled* 1 *through* N*. They start with* N *stacks, each containing a single cube. Farmer John asks Betsy to perform* P *(*$1 \leq P \leq 100,000$*) operation. There are two types of operations: moves and counts.*

1. *In a move operation, Farmer John asks Bessie to move the stack containing cube* X *on top of the stack containing cube* Y*.*
2. *In a count operation, Farmer John asks Bessie to count the number of cubes on the stack with cube* X *that are under the cube* X *and report that value.*

Write a program that can verify the results of the game.

Input 1. *Line* 1*: A single integer,* P

2. *Lines* $2...P+1$*: Each of these lines describes a legal operation. Line* 2 *describes the first operation, etc. Each line begins with a 'M' for a move operation or a 'C' for a count operation. For move operations, the line also contains two integers:* X *and* Y*. For count operations, the line also contains a single integer:* X*. Note that the value for* N *does not appear in the input file. No move operation will request a move a stack onto itself.*

Output *Print the output from each of the count operations in the same order as the input file.*

Sample Input

```
6
M 1 6
C 1
M 2 4
M 2 6
C 3
C 4
```

Sample Output

```
1
0
2
```

Exercise 2.14 ***Team Queue***

(c.f. `http://acm.pku.edu.cn/JudgeOnline/problem?id=2259`*)*

Queues and Priority Queues are data structures which are known to most computer scientists. The Team Queue, however, is not so well known, though it occurs often in everyday life. At lunch time the queue in front of the Mensa is a team queue, for example.

In a team queue each element belongs to a team. If an element enters the queue, it first searches the queue from head to tail to check if some of its teammates (elements of the same team) are already in the queue. If yes, it enters the queue right behind them. If not, it enters the queue at the tail and becomes the new last element (bad luck). Dequeuing is done like in normal queues: elements are processed from head to tail in the order they appear in the team queue.

Your task is to write a program that simulates such a team queue.

Input *The input will contain one or more test cases. Each test case begins with the number of teams* t *(*$1 \leq t \leq 1000$*). Then* t *team descriptions follow, each one consisting of the number of elements belonging to the team and the elements themselves. Elements are integers in the range* $0 - 999999$*. A team may consist of up to* 1000 *elements. Finally, a list of commands follows. There are three different kinds of commands:*

1. ***ENQUEUE*** $\boldsymbol{x}$ *- enter element* x *into the team queue*
2. ***DEQUEUE*** *- process the first element and remove it from the queue*
3. ***STOP*** *- end of test case*

The input will be terminated by a value of 0 *for* t*. Warning: A test case may contain up to* 200000 *(two hundred thousand) commands, so the implementation of the team queue should be efficient: both enqueing and dequeuing of an element should only take constant time.*

Output *For each test case, first print a line saying "Scenario #k", where* k *is the number of the test case. Then, for each DEQUEUE command, print the element which is dequeued on a single line. Print a blank line after each test case, even after the last one.*

Sample Input

```
2
3 101 102 103
3 201 202 203
ENQUEUE 101
ENQUEUE 201
ENQUEUE 102
ENQUEUE 202
ENQUEUE 103
ENQUEUE 203
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
```

```
DEQUEUE
STOP
2
5 259001 259002 259003 259004 259005
6 260001 260002 260003 260004 260005 260006
ENQUEUE 259001
ENQUEUE 260001
ENQUEUE 259002
ENQUEUE 259003
ENQUEUE 259004
ENQUEUE 259005
DEQUEUE
DEQUEUE
ENQUEUE 260002
ENQUEUE 260003
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
STOP
0
```

Sample Output

```
Scenario #1
101
102
103
201
202
203

Scenario #2
259001
259002
259003
259004
259005
260001
```

Exercise 2.15 ***Evaluation of Postorder Binary Expression Trees***

*Given a valid and well-formed postorder binary operator expression, the binary operators are only + , − , ∗ . If we encounter a number, we push the number into the stack, if we encounter the binary operator, we pop the top two numbers of the stack, then compute the result by these two numbers and the binary operator, then push the result into the stack, if we encounter the P letter, print the top number of the stack. In the case that there is nothing on the stack, the P operator will print 'NULL'. The type of number is integer. Assume that there is no error input to handle, for example, P , 4 * P , * P are all error inputs, we don't need to handle these situations. More precisely, without the letter P a well-form and valid postorder binary operator expression S can be written as* $S \rightarrow SS+|SS-|SS*|A$. *A is an integer. All numbers and calculations should be between -16,384 to +16,383.*

For example, for 3 4 P - P; we encounter 3, put 3 into the stack; then we encounter 4, put 4 into the stack; then we encounter P, print the top number of the stack (that is 4); then we encounter $*$*, we pop the top two number of the stack (that is 4, 3), we compute the result by these two numbers and the binary operator, that is* $3-4=-1$*, then we push -1 into the stack; then we encounter P again, we print the top number of the stack (that is -1), so the output is 4 -1.*

Input *The input consists of the number of test cases, m in the first line and followed by m test cases, each test case is a valid and well-formed postorder binary operator expression. Assume that there are at most 20 integers in each postorder binary operator expression. And there is at least a letter P in each postorder binary operator expression.*

```
2
P 3 4 P * P P
10 9 P * P 5 6 P - + P
```

Output *The output should be m lines of numbers. Each line should be the output printed in each test case.*

```
NULL 4 12 12
9 90 6 89
```

Exercise 2.16 ***Train Re-arrangement***

You are a railroad operator and you are asked to see whether you can re-arrange the carts in some order by using an auxiliary track, which can be regarded as a stack. The operations on the carts include the following:

- *'Straight Through', which means that you let the cart pass the main track directly without using the stack;*
- *'Push', which means that you put the cart into the stack;*
- *'Pop', which means that you pull the toppest cart out from the stack.*

Figure 1 shows an example of three carts, which are represented by rectangles of different colors. In this example, the initial order is Green, Yellow, and Red, the demanded final order is Yellow, Green, and Red. It shows that you can achieve the demanded order by operation sequence 'Push, Straight Through, Pop, Straight Through'. Please note that after you push the Green cart, you can push the Yellow cart and then pop it immediately. However, you will use one extra operation than straight through the Yellow cart. The manage hopes you to save the operations as many as possible.

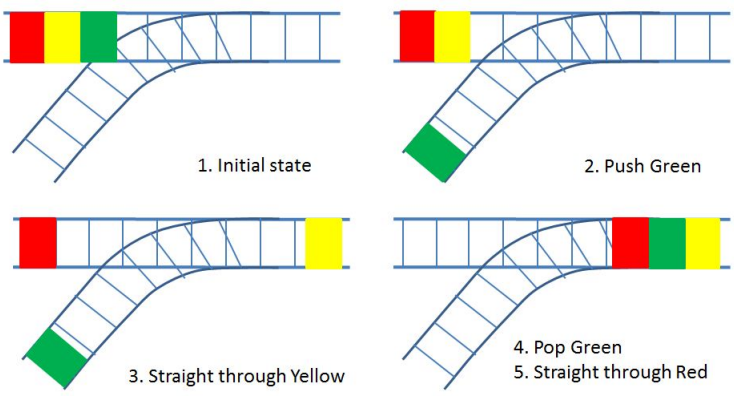


Figure 1: One example of re-arrangement

Input *The input consists of the number of test cases, m in the first line and followed by m test cases. In each test case, the first integer is the total number of carts, n ($1 \leq n \leq 100,000$), followed by the order your manager wants you to achieve after your re-arrangement. Assume that in the initial state, all the carts are ordered from 1 to n, with Cart 1 in the first place.*

```
3
3 2 1 3
7 3 6 7 5 4 2 1
5 5 1 2 3 4
```

Output *The output should be m lines of operations. Each line should be the* ***shortest*** *sequence of 'S', 'I', and 'O' if you can achieve the demanded order by your re-arrangement.*

We use 'S' to denote 'Straight Through', 'I' to denote 'Push', and 'O' to denote 'Pop'. There is no space between operations. If you cannot achieve the demanded order, please output 'Impossible'.

```
ISOS
IISIISSOOOO
Impossible
```

Exercise 2.17 ***Train Re-arrangement II***

You are a railroad operator and you are asked to see whether you can re-arrange the carts in some order by using an auxiliary track, which can be regarded as a queue. The operations on the carts include the following:

- *'Straight Through', which means that you let the cart pass the main track directly without using the queue;*
- *'Enqueue', which means that you put the cart into the queue;*
- *'Dequeue', which means that you pull the top cart out from the queue.*

The manage hopes you to save the operations as many as possible.

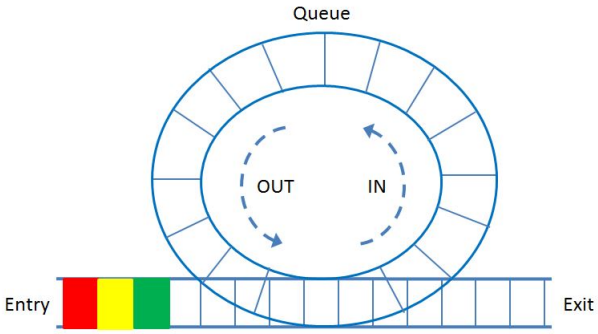


Figure 2: The structure of railroad

Input *The input consists of the number of test cases, m in the first line and followed by m test cases. In each test case, the first integer is the total number of carts, n ($1 \leq n \leq 100,000$), followed by the order your manager wants you to achieve after your re-arrangement. Assume that in the initial state, all the carts are ordered from 1 to n, with Cart 1 in the first place.*

```
3
3 2 1 3
7 3 6 7 5 4 2 1
5 5 1 2 3 4
```

Output *The output should be m lines of operations. Each line should be the* ***shortest*** *sequence of 'S', 'I', and 'O' if you can achieve the demanded order by your re-arrangement. We use 'S' to denote 'Straight Through', 'I' to denote 'Enqueue', and 'O' to denote 'Dequeue'. There is no space between operations. If you cannot achieve the demanded order, please output 'Impossible'.*

```
ISOS
Impossible
IIIISOOOO
```

Exercise 2.18 ***Fruit Merge***

The harvest season has come! Since Maggie works in an orchard, she has picked all the fruits and separated them into n piles according to the kind, e.g. one pile for apples, one pile for oranges, etc. After that, she wants to merge all these piles into one pile. Every time, she can only merge two piles, and the energy cost of merging them is the sum of weight of the two piles. It is easily to see that after $n-1$ merges, there is only one pile left, and the total energy cost is sum of energy cost in all these $n-1$ merges. Of course, Maggie wants to save her energy as much as possible.

Now, assume that the weight of each fruit is 1 and you have known the total number of piles and number of fruits in each pile. Your target is to output the minimum total energy cost of Maggie.

Hint *It has been proved that you can achieve the minimum total energy cost by merging the two piles with the smallest number of fruits every time.*

Input *There are multiple test cases. In each test case, the first line is an integer n ($1 \leq n \leq 1,000$) representing the total number of piles. The next line includes n positive integers with each representing the number of fruits in each pile. Please note that we have sorted all these numbers in non-decreasing order for your convenience. The end of input is specified by a line in which $n = 0$.*

```
3
1 2 9
6
1 1 3 4 4 6
0
```

Output *For each test case, you should use one line to output your result. The result includes only one integer, which is the minimum total energy cost of Maggie after merging all the fruit piles into one pile. We guarantee that this value will be always less than 2^{31}.*

```
15
45
```

3 Trees

Exercise 3.1 *Translate each of the following infix expressions into prefix expressions.*

1. $(6 * 4)/(7 - (3 + 2))$
2. $(6 * 4)/((7 - 3) + 2))$
3. $(2 + 7) * (8/6 - (3 + 1))$
4. $(2 + 7) * (8/((6 - 3) + 1))$

Exercise 3.2 *Translate each of the following prefix expressions into infix expressions and then evaluate it.*

1. $* - 9 + 2\ 3 - 7\ 1$
2. $* + -9\ 2\ 3 - 7\ 1$
3. $/ + *3\ 8\ 4 + 6 - 3\ 2$
4. $/ + 4 * 3\ 8 + -3\ 2\ 6$

Exercise 3.3 *Translate each of the following postfix expressions into infix expressions and then evaluate it.*

1. $4\ 5 * 6 - 7/8+$
2. $4\ 5\ 6 * 7\ 8 + /-$
3. $6\ 5\ 4\ 3\ 2\ 1 - +/ + *$
4. $6\ 5 * 4\ 3\ 2 + 1 - /+$

Exercise 3.4 *Would it make sense to call a stack*

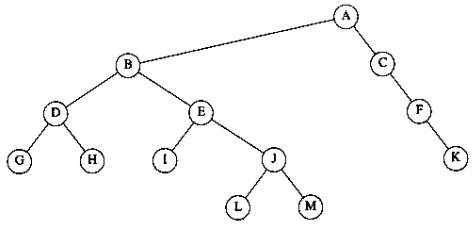
1. *a "LILO" structure? Justify your answer.*
2. *a "FILO" structure? Justify your answer.*

Exercise 3.5 *What are the advantages and disadvantages of the linked implementation of a stack relative to the continuous implementation?*

Exercise 3.6 *Determine whether each of the following is true about postfix expressions:*

1. $x\ y\ +\ z\ +\ =\ x\ y\ z\ +\ +$
2. $x\ y\ +\ z\ -\ =\ x\ y\ z\ -\ +$
3. $x\ y\ -\ z\ +\ =\ x\ y\ z\ +\ -$
4. $x\ y\ -\ z\ -\ =\ x\ y\ z\ -\ -$

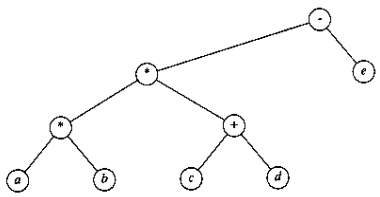
Exercise 3.7



1. *Which node is the root?*
2. *Which nodes are leaves?*
3. *Name the parent node for each node.*
4. *List the children for each node.*
5. *List the siblings for each node.*
6. *Compute the depth for each node.*
7. *Compute the height for each node.*
8. *What is the depth of the tree?*

Exercise 3.8 *Show that the maximum number of nodes in a binary tree of height h is $2^{h+1}-1$.*

Exercise 3.9 *Give the prefix, infix, and postfix expressions corresponding to the tree below.*



Exercise 3.10

1. *Show the result of inserting 3, 1, 4, 6, 9, 2, 8, 5, 7, 0 into an empty binary search tree.*
2. *Show the result of deleting the root twice. Demonstrate intermediate steps and the rotation(s) needed.*

Exercise 3.11

1. *Show the result of inserting 3, 1, 4, 6, 9, 2, 8, 5, 7, 0 into an empty AVL tree.*
2. *Show the result of deleting the root twice. Demonstrate intermediate steps and the rotation(s) needed.*

Exercise 3.12 *In each of the following, insert the keys, in the order shown, to build them into an AVL tree (assuming lexical ordering like a dictionary). Demonstrate intermediate steps and the rotation(s) needed.*

1. *A, B, C, D, E, F, G, H, I, J, K, L*
2. *S, T, R, U, C, TT, U, R, E*
3. *A, V, L, T, R, E, I, S, O, K*
4. *C, U, H, K, I, S, G, R, E, A, T*

Exercise 3.13 *Delete each of the keys inserted in the previous exercise above from the AVL tree, in LIFO order (last key inserted is first deleted). Demonstrate intermediate steps and the rotation(s) needed.*

Exercise 3.14 *Delete each of the keys inserted in the previous exercise above from the AVL tree, in FIFO order (first key inserted is first deleted). Demonstrate intermediate steps and the rotation(s) needed.*

Exercise 3.15 *Prove that the number of (single or double) rotations done in deleting a key from an AVL tree cannot exceed half the height of the tree.*

Exercise 3.16 *Prove that an AVL tree with n nodes has height* $O(\log n)$. *(Hint: Prove that in an AVL tree of height h, there are at least* F_h *nodes, where* F_h *is the h-th Fibonacci number.)*

Exercise 3.17 *Show that for an AVL tree with n nodes, it takes* $O(\log n)$ *times and perform* $O(1)$ *rotations for an insertion operation.*

Exercise 3.18

1. *Show the result of inserting 3, 1, 4, 6, 9, 2, 8, 5, 7, 0 into an empty 2-3 tree.*
2. *Show the step result of deleting 0, 9, 1, and 5 from the 2-3 tree just created.*

Exercise 3.19 *Use the basic AVL tree to implement a simple address book containing unique last name and first name only. Implement operations such as* `create`, `insert`, `delete`, `find`, *and* `list`.

Exercise 3.20 *Write a program that counts the leaves in a binary tree.*

Exercise 3.21 *Write a program that counts the number of nodes in a binary tree that have one external and one internal child.*

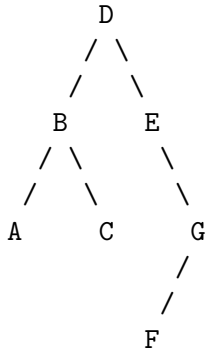
Exercise 3.22 ***Black Box*** *(c.f.* `http://acm.uva.es/p/v5/501.html`*).*

Exercise 3.23 ***S Tree*** *(c.f.* `http://acm.uva.es/p/v7/712.html`*).*

Exercise 3.24 ***Tree Recovery*** *(c.f.* `http://acm.uva.es/p/v5/536.html`*)*

Little Valentine liked playing with binary trees very much. Her favorite game was constructing randomly looking binary trees with capital letters in the nodes.

This is an example of one of her creations:



To record her trees for future generations, she wrote down two strings for each tree: a preorder traversal (root, left subtree, right subtree) and an inorder traversal (left subtree, root, right subtree).

For the tree drawn above the preorder traversal is DBACEGF and the inorder traversal is ABCDEFG.

She thought that such a pair of strings would give enough information to reconstruct the tree later (but she never tried it).

Now, years later, looking again at the strings, she realized that reconstructing the trees was indeed possible, but only because she never had used the same letter twice in the same tree.

However, doing the reconstruction by hand, soon turned out to be tedious.

So now she asks you to write a program that does the job for her!

Input Specification

The input file will contain one or more test cases. Each test case consists of one line containing two strings preord and inord, representing the preorder traversal and inorder traversal of a binary tree. Both strings consist of unique capital letters. (Thus they are not longer than 26 characters.)

Input is terminated by end of file.

Output Specification

For each test case, recover Valentine's binary tree and print one line containing the tree's postorder traversal (left subtree, right subtree, root).

Sample Input

```
DBACEGF ABCDEFG
BCAD CBAD
```

Sample Output

```
ACBFGED
CDAB
```

Exercise 3.25 ***Tree Summing*** *(c.f.* `http://acm.uva.es/problemset/v1/112.html`*)*

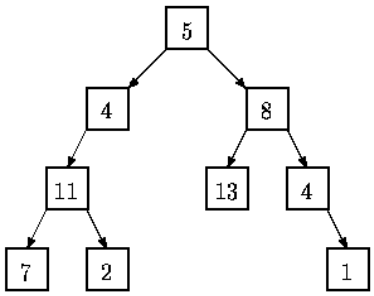
Background

LISP was one of the earliest high-level programming languages and, with FORTRAN, is one of the oldest languages currently being used. Lists, which are the fundamental data structures in LISP, can easily be adapted to represent other important data structures such as trees.

This problem deals with determining whether binary trees represented as LISP S-expressions possess a certain property.

The Problem

Given a binary tree of integers, you are to write a program that determines whether there exists a root-to-leaf path whose nodes sum to a specified integer. For example, in the tree shown below there are exactly four root-to-leaf paths. The sums of the paths are 27, 22, 26, and 18.



Binary trees are represented in the input file as LISP S-expressions having the following form.

```
empty tree  ::=   ()

tree  ::=   empty tree  |  (integer tree tree)
```

The tree diagrammed above is represented by the expression `(5 (4 (11 (7 () ()) (2 () ()) ) ()) (8 (13 () ()) (4 () (1 () ()) ) ) )`

Note that with this formulation all leaves of a tree are of the form `(integer () () )`

Since an empty tree has no root-to-leaf paths, any query as to whether a path exists whose sum is a specified integer in an empty tree must be answered negatively.

The Input

The input consists of a sequence of test cases in the form of integer/tree pairs. Each test case consists of an integer followed by one or more spaces followed by a binary tree formatted as an S-expression as described above. All binary tree S-expressions will be valid, but expressions may be spread over several lines and may contain spaces. There will be one or more test cases in an input file, and input is terminated by end-of-file.

The Output

There should be one line of output for each test case (integer/tree pair) in the input file. For each pair I, T (I represents the integer, T represents the tree) the output is the string yes if there is a root-to-leaf path in T whose sum is I and no if there is no path in T whose sum is I.

Sample Input

```
22 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
20 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
10 (3
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )
5 ()
```

Sample Output

```
yes
no
yes
no
```

Exercise 3.26 ***Tree*** *(c.f.* `http://acm.uva.es/p/v5/548.html`*)*

Background

You are to determine the value of the leaf node in a given binary tree that is the terminal node of a path of least value from the root of the binary tree to any leaf. The value of a path is the sum of values of nodes along that path.

Input

The input file will contain a description of the binary tree given as the inorder and postorder traversal sequences of that tree. Your program will read two line (until end of file) from the input file. The first line will contain the sequence of values associated with an inorder traversal of the tree and the second line will contain the sequence of values associated with a postorder traversal of the tree. All values will be different, greater than zero and less than 10000. You may assume that no binary tree will have more than 10000 nodes or less than 1 node.

Output

For each tree description you should output the value of the leaf node of a path of least value. In the case of multiple paths of least value you should pick the one with the least value on the terminal node.

Sample Input

```
3 2 1 4 5 7 6
3 1 2 5 6 7 4
7 8 11 3 5 16 12 18
8 3 11 7 16 18 12 5
255
255
```

Sample Output

```
1
3
255
```

Exercise 3.27 ***Trees on the level*** *(c.f.* `http://acm.uva.es/p/v1/122.html`*)*

Background

Trees are fundamental in many branches of computer science. Current state-of-the art parallel computers such as Thinking Machines' CM-5 are based on fat trees. Quad- and octal-trees are fundamental to many algorithms in computer graphics.

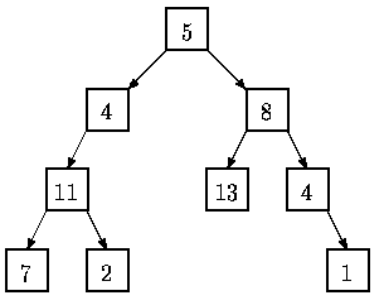
This problem involves building and traversing binary trees.

The Problem

Given a sequence of binary trees, you are to write a program that prints a level-order traversal of each tree. In this problem each node of a binary tree contains a positive integer and all binary trees have fewer than 256 nodes.

In a level-order traversal of a tree, the data in all nodes at a given level are printed in left-to-right order and all nodes at level k are printed before all nodes at level $k+1$.

For example, a level order traversal of the tree



is: 5, 4, 8, 11, 13, 4, 7, 2, 1.

In this problem a binary tree is specified by a sequence of pairs (n, s) where n is the value at the node whose path from the root is given by the string s. A path is given be a sequence of L's and R's where L indicates a left branch and R indicates a right branch. In the tree diagrammed above, the node containing 13 is specified by (13,RL), and the node containing 2 is specified by (2,LLR). The root node is specified by (5,) where the empty string indicates the path from the root to itself. A binary tree is considered to be completely specified if every node on all root-to-node paths in the tree is given a value exactly once.

The Input

The input is a sequence of binary trees specified as described above. Each tree in a sequence consists of several pairs (n, s) as described above separated by whitespace. The last entry in each tree is (). No whitespace appears between left and right parentheses.

All nodes contain a positive integer. Every tree in the input will consist of at least one node and no more than 256 nodes. Input is terminated by end-of-file.

The Output

For each completely specified binary tree in the input file, the level order traversal of that tree should be printed. If a tree is not completely specified, i.e., some node in the tree is NOT given a value or a node is given a value more than once, then the string "not complete" should be printed.

Sample Input

```
(11,LL) (7,LLL) (8,R)
(5,) (4,L) (13,RL) (2,LLR) (1,RRR) (4,RR) ()
(3,L) (4,R) ()
```

Sample Output

```
5 4 8 11 13 4 7 2 1
not complete
```

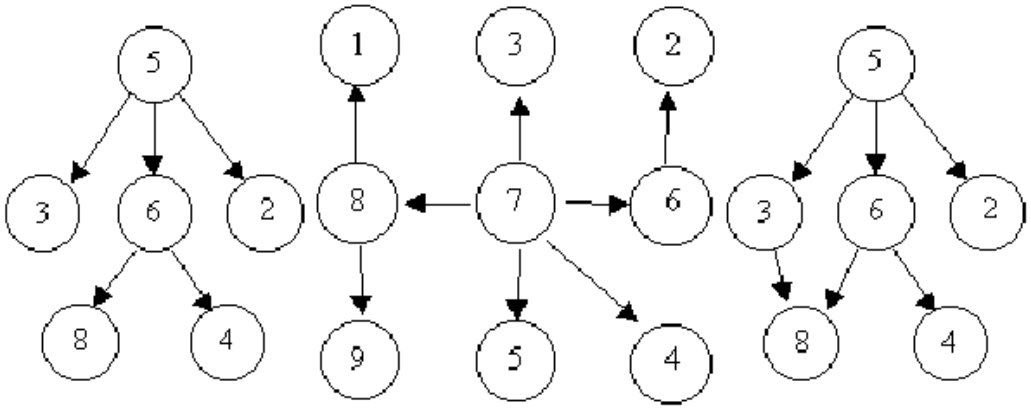
Exercise 3.28 ***Is It A Tree?*** *(c.f.* `http://acm.uva.es/p/v6/615.html`*)*

Background

A tree is a well-known data structure that is either empty (null, void, nothing) or is a set of one or more nodes connected by directed edges between nodes satisfying the following properties.

- *There is exactly one node, called the root, to which no directed edges point.*
- *Every node except the root has exactly one edge pointing to it.*
- *There is a unique sequence of directed edges from the root to each node.*

For example, consider the illustrations below, in which nodes are represented by circles and edges are represented by lines with arrowheads. The first two of these are trees, but the last is not.



In this problem you will be given several descriptions of collections of nodes connected by directed edges. For each of these you are to determine if the collection satisfies the definition of a tree or not.

Input

The input will consist of a sequence of descriptions (test cases) followed by a pair of negative integers. Each test case will consist of a sequence of edge descriptions followed by a pair of zeroes Each edge description will consist of a pair of integers; the first integer identifies the node from which the edge begins, and the second integer identifies the node to which the edge is directed. Node numbers will always be greater than zero.

Output

For each test case display the line "Case k is a tree." or the line "Case k is not a tree.", where k corresponds to the test case number (they are sequentially numbered starting with 1).

Sample Input

```
6 8  5 3  5 2  6 4
5 6  0 0

8 1  7 3  6 2  8 9  7 5
7 4  7 8  7 6  0 0

3 8  6 8  6 4
5 3  5 6  5 2  0 0
-1 -1
```

Sample Output

```
Case 1 is a tree.
Case 2 is a tree.
Case 3 is not a tree.
```

Exercise 3.29 ***Quadtrees*** *(c.f.* `http://acm.uva.es/p/v2/297.html`*)*

Background

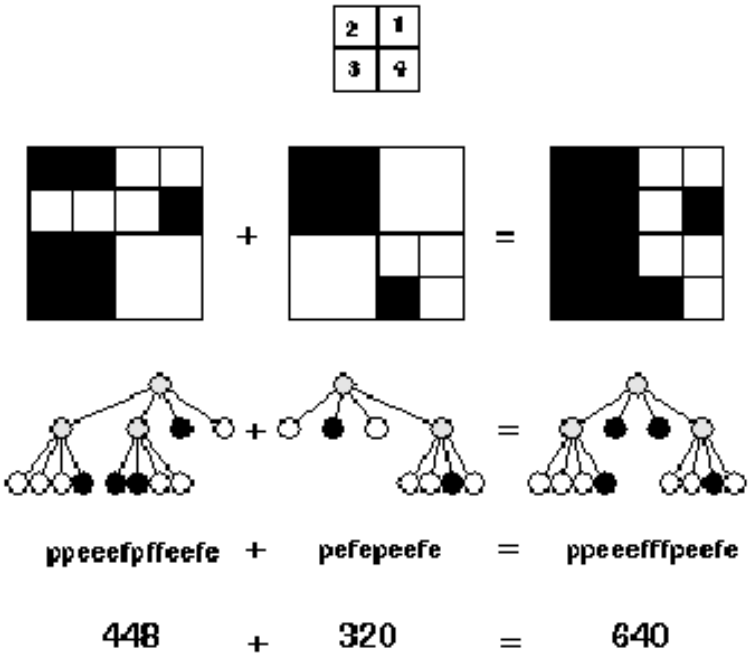
A quadtree is a representation format used to encode images. The fundamental idea behind the quadtree is that any image can be split into four quadrants. Each quadrant may again be split in four sub quadrants, etc. In the quadtree, the image is represented by a parent node, while the four quadrants are represented by four child nodes, in a predetermined order.

Of course, if the whole image is a single color, it can be represented by a quadtree consisting of a single node. In general, a quadrant needs only to be subdivided if it consists of pixels of different colors. As a result, the quadtree need not be of uniform depth.

A modern computer artist works with black-and-white images of 32×32 *units, for a total of 1024 pixels per image. One of the operations he performs is adding two images together, to form a new image. In the resulting image a pixel is black if it was black in at least one of the component images, otherwise it is white.*

This particular artist believes in what he calls the preferred fullness: for an image to be interesting (i.e. to sell for big bucks) the most important property is the number of filled (black) pixels in the image. So, before adding two images together, he would like to know how many pixels will be black in the resulting image. Your job is to write a program that, given the quadtree representation of two images, calculates the number of pixels that are black in the image, which is the result of adding the two images together.

In the figure, the first example is shown (from top to bottom) as image, quadtree, pre-order string (defined below) and number of pixels. The quadrant numbering is shown at the top of the figure.



Input Specification

The first line of input specifies the number of test cases (N) your program has to process.

The input for each test case is two strings, each string on its own line. The string is the pre-order representation of a quadtree, in which the letter '`p`*' indicates a parent node, the letter '*`f`*' (full) a black quadrant and the letter '*`e`*' (empty) a white quadrant. It is guaranteed that each string represents a valid quadtree, while the depth of the tree is not more than 5 (because each pixel has only one color).*

Output Specification

For each test case, print on one line the text 'There are `X` *black pixels.', where* `X` *is the number of black pixels in the resulting image.*

Example Input

```
3
ppeeefpffeefe
pefepeefe
peeef
peefe
peeef
peepefefe
```

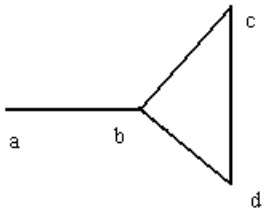
Example Output

```
There are 640 black pixels.
There are 512 black pixels.
There are 384 black pixels.
```

Exercise 3.30 ***The Forrest for the Trees*** *(c.f.* `http://acm.uva.es/p/v5/599.html`*)*

Background *A graph G is a set of point $V(G)$, together with a set of edges $E(G)$, where each element of $E(G)$ is an unordered pair of distinct points of $V(G)$.*

Example 1*: Let G be a graph where $V(G) = \{a, b, c, d\}$ and $E(G) = \{(a, b), (b, c), (c, d), (d, b)\}$. The figure gives a depiction of G.*

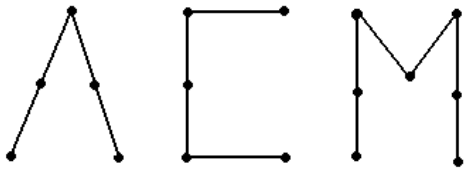


Notice that G contains the "cycle" , $\{(b, c), (c, d), (d, b)\}$. A graph devoid of cycles is called a tree. A path in a graph G is an alternating sequence of points and edges, (beginning and ending with a point) such that all the points of the path are distinct. In the graph of example 1, is a path.

Fact*: Every two points of a tree are joined by a unique path.*

A graph is called connected if every pair of points are joined by a path. The graph of example 1 is connected. If a graph is not connected then it is made up of "subgraphs" which are. Each one of these subgraphs is called a connected component of the graph G.

A graph for which each connected component is a tree is called a forest, see figure below.



One extreme case worth mentioning is the case when one of the component trees has one point but no edges joined to it. This tree likes like an isolated dot. We will call this an acorn. We are ready to define the problem.

Problem*: Given a forest you are to write a program that counts the number of trees and acorns.*

Input

The first line of the input file contains the number of test cases your program has to process. Each test case is a forest description consisting of two parts:

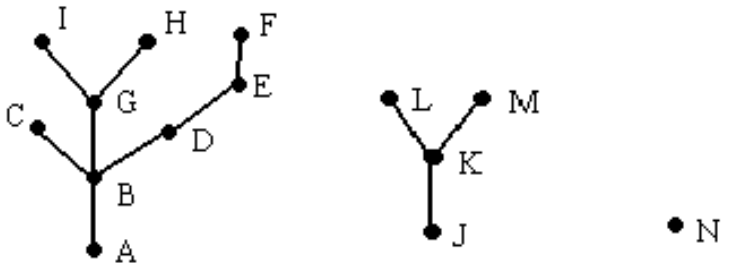
1. *A list of edges of the tree (one per line, given as an unordered pair of capital letters delimited by a row of asterisks).*
2. *A list of points of the tree (these will be given on one line with a maximum of 26 corresponding to the capital letters, $A - Z$).*

Output

For each test case your program should print the number of trees and the number of acorns, in a sentence, for example:

"`There are x tree(s) and y acorn(s).`*", where x and y are the numbers of trees and acorns, respectively.*

Example 2: *Let G be a graph whose edges and points are given by the first test case in the sample input. A depiction of this graph is given in figure following.*



Notes*: A forest may have no trees and all acorns, all trees and no acorns, or anything inbetween, so keep your eyes open and don't miss the forest for the trees!*

Sample Input

```
2
(A,B)
(B,C)
(B,D)
(D,E)
(E,F)
(B,G)
(G,H)
(G,I)
(J,K)
(K,L)
(K,M)
****
A,B,C,D,E,F,G,H,I,J,K,L,M,N
(A,B)
(A,C)
(C,F)
**
A,B,C,D,F
```

Sample Output

```
There are 2 tree(s) and 1 acorn(s).
There are 1 tree(s) and 1 acorn(s).
```

Exercise 3.31 ***A Variation on Tree Summing*** *(c.f.* `http://acm.uva.es/problemset/v1/112.html`*)*

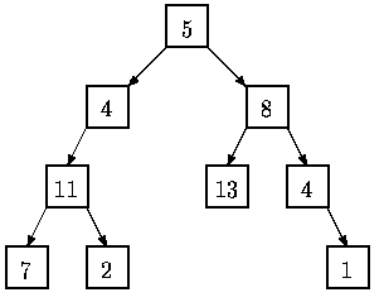
Background

LISP was one of the earliest high-level programming languages and, with FORTRAN, is one of the oldest languages currently being used. Lists, which are the fundamental data structures in LISP, can easily be adapted to represent other important data structures such as trees.

This problem deals with determining whether binary trees represented as LISP S-expressions possess a certain property.

The Problem

Given a binary tree of integers, you are to write a program that determines how many root-to-leaf path whose nodes sum to be less than a specified integer, equal to a specified integer, and greater than a specified integer. For example, in the tree shown below there are exactly four root-to-leaf paths. The sums of the paths are 27, 22, 26, and 18. Hence, if the specified integer is 22, the output should be 1 1 2.



Binary trees are represented in the input file as LISP S-expressions having the following form.

```
empty tree  ::=   ()

tree  ::=   empty tree  |  (integer tree tree)
```

The tree diagrammed above is represented by the expression `(5 (4 (11 (7 () ()) (2 () ()) ) ()) (8 (13 () ()) (4 () (1 () ()) ) ) )`

Note that with this formulation all leaves of a tree are of the form `(integer () () )`

Since an empty tree has no root-to-leaf paths, any query as to whether a path exists whose sum is a specified integer in an empty tree must be answered negatively.

The Input

The input consists of a sequence of test cases in the form of integer/tree pairs. Each test case consists of an integer followed by one or more spaces followed by a binary tree formatted as an S-expression as described above. All binary tree S-expressions will be valid, but expressions may be spread over several lines and may contain spaces. There will be one or more test cases in an input file, and input is terminated by end-of-file.

The Output

There should be one line of output for each test case (integer/tree pair) in the input file. For each pair I, T (I represents the integer, T represents the tree) the output is the string of three integers. The first integer should be the number of root-to-leaf paths that the sum is less than the specified integer. The second integer should be the number of root-to-leaf paths that the sum is equal to the specified integer. The third integer should be the number of root-to-leaf paths that the sum is greater than the specified integer.

Sample Input

```
22 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
20 (5(4(11(7()())(2()()))()) (8(13()())(4()(1()()))))
10 (3
     (2 (4 () () )
        (8 () () ) )
     (1 (6 () () )
        (4 () () ) ) )
5 ()
```

Sample Output

```
1 1 2
1 0 3
2 1 1
0 0 0
```

Exercise 3.32 ***Binary Tree Traversal***

In this class, we introduced three tree traversal strategies: pre-order traversal, in-order traversal and post-order traversal. Now, given the pre-order and in-order traversal sequences of a binary tree, you are asked to output the post traversal order of this tree. Note that the structure of a binary tree can be uniquely decided by its pre-order and in-order.

Input *The input consists of a sequence of descriptions (test cases) followed by a line with a single integer -1. Each test case consists of three lines: the first line contains an integer N, which represents the number of nodes of a binary tree; the second line consists of a sequence of integers separated by spaces indicating the pre-order of the binary tree (suppose nodes of the tree are represented by integers); correspondingly, the third line consists of a sequence of integers separated by spaces indicating the in-order of the binary tree. Integers in the traversal sequence will always be greater than zero.*

Output *For each test case output the post-order of the tree. Adjacent integers in the sequence should be separated by one space.*

Sample Input

```
3
4 1 5
1 4 5
4
2 4 6 3
4 6 2 3
9
8 1 4 12 7 3 2 9 6
12 4 7 1 3 8 9 6 2
-1
```

Sample Output

```
1 5 4
6 4 3 2
12 7 4 3 1 6 9 2 8
```

Exercise 3.33 ***Disk Tree*** *(c.f.* `http://acm.pku.edu.cn/JudgeOnline/problem?id=1760`*)*

Hacker Bill has accidentally lost all the information from his workstation's hard drive and he has no backup copies of its contents. He does not regret for the loss of the files themselves, but for the very nice and convenient directory structure that he had created and cherished during years of work. Fortunately, Bill has several copies of directory listings from his hard drive. Using those listings he was able to recover full paths (like "

```
WINNT\SYSTEM32\CERTSRV\CERTCO~1\X86
```

") for some directories. He put all of them in a file by writing each path he has found on a separate line. Your task is to write a program that will help Bill to restore his state of the art directory structure by providing nicely formatted directory tree.

Input *The first line of the input file contains single integer number N ($1 \leq N \leq 500$) that denotes a total number of distinct directory paths. Then N lines with directory paths follow. Each directory path occupies a single line and does not contain any spaces, including leading or trailing ones. No path exceeds 80 characters. Each path is listed once and consists of a number of directory names separated by a back slash ("). Each directory name consists of 1 to 8 uppercase letters, numbers, or the special characters from the following list: exclamation mark, number sign, dollar sign, percent sign, ampersand, apostrophe, opening and closing parenthesis, hyphen sign, commercial at, circumflex accent, underscore, grave accent, opening and closing curly bracket, and tilde ("*

```
!#$%&'()-@^_`{}~
```

").

Output *Write to the output file the formatted directory tree. Each directory name shall be listed on its own line preceded by a number of spaces that indicate its depth in the directory hierarchy. The subdirectories shall be listed in lexicographic order immediately after their parent directories preceded by one more space than their parent directory. Top level directories shall have no spaces printed before their names and shall be listed in lexicographic order. See sample below for clarification of the output format.*

Sample Input

```
7
WINNT\SYSTEM32\CONFIG
GAMES
WINNT\DRIVERS
HOME
WIN\SOFT
GAMES\DRIVERS
WINNT\SYSTEM32\CERTSRV\CERTCO~1\X86
```

Sample Output

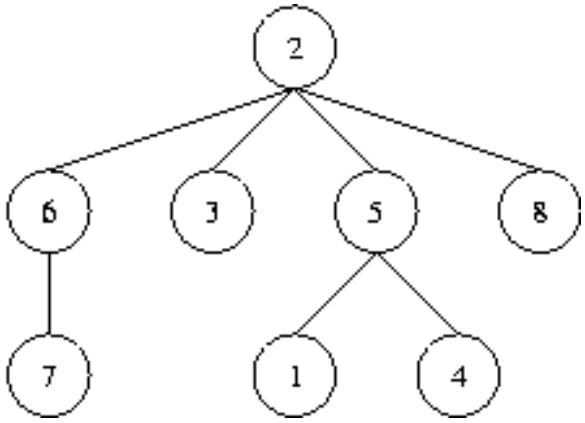
```
GAMES
 DRIVERS
HOME
WIN
 SOFT
WINNT
 DRIVERS
 SYSTEM32
  CERTSRV
   CERTCO~1
    X86
  CONFIG
```

Exercise 3.34 ***Code the Tree*** *(c.f.* `http://acm.pku.edu.cn/JudgeOnline/problem?id=2567`*)*

A tree (i.e. a connected graph without cycles) with vertices numbered by the integers 1, 2, ..., *n is given. The "Prufer" code of such a tree is built as follows: the leaf (a vertex that is incident to only one edge) with the minimal number is taken. This leaf, together with its incident edge is removed from the graph, while the number of the vertex that was adjacent to the leaf is written down. In the obtained graph, this procedure is repeated, until there is only one vertex left (which, by the way, always has number* n*). The written down sequence of* $n-1$ *numbers is called the Prufer code of the tree. Your task is, given a tree, to compute its Prufer code. The tree is denoted by a word of the language specified by the following grammar:*

```
T ::= "(" N S ")"
S ::= " " T S
    | empty
N ::= number
```

That is, trees have parentheses around them, and a number denoting the identifier of the root vertex, followed by arbitrarily many (maybe none) subtrees separated by a single space character. As an example, take a look at the tree in the figure below which is denoted in the first line of the sample input. Note that, according to the definition given above, the root of a tree may be a leaf as well. It is only for the ease of denotation that we designate some vertex to be the root. Usually, what we are dealing here with is called an "unrooted tree".



Input *The input contains several test cases. Each test case specifies a tree as described above on one line of the input file. Input is terminated by EOF. You may assume that* $1 \leq n \leq 50$.

Output *For each test case generate a single line containing the Prufer code of the specified tree. Separate numbers by a single space. Do not print any spaces at the end of the line.*

Sample Input

```
(2 (6 (7)) (3) (5 (1) (4)) (8))
(1 (2 (3)))
(6 (1 (4)) (2 (3) (5)))
```

Sample Output

```
5 2 5 2 6 2 8
2 3
2 1 6 2 6
```

Exercise 3.35 ***The Weight of Trees***

A tree is a connected graph consisting of N vertices and $N-1$ edges. Trees also have the property of there being exactly a single unique path between any pair of vertices.

You will be given a tree in which every edge is assigned a weight - a non negative integer. The weight of a path is the product of the weights of all edges on the path. The weight of the tree is the sum of the weights of all paths in the tree. Paths going in opposite directions (A to B and B to A) are counted only once.

Write a program that, given a tree, calculates its weight modulo 1000000007.

Input *The first line contains the integer $N(2 \leq N \leq 1000)$, the number of vertices in the tree. The vertices are numbered 1 to N. Each of the following $N-1$ contains three integers A, B and $W(1 \leq A, B \leq N, 0 \leq W \leq 1000)$ describing one edge. The edge connects vertices A and B, and its weight is W.*

Output *Output the weight of the tree, modulo 1000000007.*

Sample Input

```
5
1 2 2
2 3 3
4 3 2
5 3 2
```

Sample Output

```
55
```

4 Hashing Function

Exercise 4.1 *Given input* $\{4371, 1323, 6173, 4199, 4344, 9679, 1989\}$ *and a hash function* $h(x) = x \ mod \ 10$*, show the resulting*

1. *open hash table.*
2. *closed hash table using linear probing.*
3. *closed hash table using quadratic probing.*
4. *closed hash table with second hash function* $h_2(x) = 7 - (x \ mod \ 7)$*.*
5. *What are the advantages and disadvantages of the various collision strategies in (1) - (4)?*

Exercise 4.2 *Show the result of rehashing the hash tables in the previous question.*

Exercise 4.3 *A spelling checker reads an input file and prints out all words not in some online dictionary. Suppose the dictionary contains 30,000 words and the file is one megabyte, so that the algorithm can make only one pass through the input file. A simple strategy is to read the dictionary into a hash table and look for each input word as it is read. Assuming that an average word is seven characters and that it is possible to store words of length* L *in* $L+1$ *bytes (so space waste is not much of a consideration), and assuming a closed table, how much space does this require?*

Exercise 4.4 *If memory is limited and the entire dictionary cannot be stored in a hash table, we can still get an efficient algorithm that almost always works. We declare an array* H_Table *of bits (initialized to zeros) from 0 to* $Table\_Size - 1$*. As we read in a word, we set* $H\_Table[Hash(Word)] = 1$*. Suppose we choose* $Table\_Size = 3000,007$*.*

1. *True or False: if a word hashes to a location with value 0, the word is not in the dictionary.*
2. *True or False: if a word hashes to a location with value 1, the word is in the dictionary.*
3. *How much memory does this require?*
4. *What is the probability of an error in this algorithm?*
5. *A typical document might have about three actual misspellings per page of 500 words. Is this algorithm usable?*

Exercise 4.5 *Show the result of inserting the keys 10111101, 00000010, 10011011, 10111110, 01111111, 01010001, 10010110, 00001011, 11001111, 10011110, 11011011, 00101011, 01100001, 11110000, 01101111 into an initially empty extendible hashing data structure with* $m = 4$*.*

Exercise 4.6 *If the extendible hashing table is small enough to fit in main memory, how does its performance compare with open and closed hashing?*

Exercise 4.7 *Write a program to compute the χ^2 statistic for the hash values of N keys with table size M. This number is defined by the equation*

$$\chi^2 = \frac{M}{N} \sum_{0 \le i < M} \left(f_i - \frac{N}{M} \right)^2 ,$$

where f_i is the number of keys with hash value i. If the hash values are random, this statistic, for $N > cM$, should be $M \pm \sqrt{M}$ with probability $1 - 1/c$.

Exercise 4.8 *Use the χ^2 statistic program to evaluate the hash function* `618033 * x % 10000` *for keys that are random positive integers less than 10^6.*

Exercise 4.9 *Consider the idea of implementing modular hashing for integer keys with the code* `(a*x) % M`*, where* `a` *is an arbitrary fixed prime. Does this change mix up the bits sufficiently well tha you can use nonprime* `M`*?*

Exercise 4.10 *Prove that* `(((a x) mod M) + b) mod M = (a x + b) mod M`*, assuming that* `a`*,* `b`*,* `x`*, and* `M` *are all nonnegative integers.*

Exercise 4.11 *Use the χ^2 statistic program to evaluate the hash function* `97 * x % M` *for all table sizes between 100 and 200, using 10^2 random positive integers less than 10^3 as keys.*

Exercise 4.12 *Use the χ^2 statistic program to evaluate the hash function* `97 * x % M` *for all table sizes between 100 and 200, using 10^3 random positive integers less than 10^6 as keys.*

Exercise 4.13 *Use the χ^2 statistic program to evaluate the hash function* `100 * x % M` *for all table sizes between 100 and 200, using 10^3 random positive integers less than 10^6 as keys.*

Exercise 4.14 *How long could it take in the worst case to insert N keys into an initially empty table, using separate chaining with (i) unordered lists and (ii) ordered lists?*

Exercise 4.15 *How long could it take in the worst case to insert N keys into an initially empty table, using linear probing?*

Exercise 4.16 *Tutors do not know how to implement hashing. It is a good chance for you to help your tutors. Please implement a hash table and a hash function to store non-negative integers. If you code it well, the tutors will give you many candies. Of course, the tutors will use a mass of commands to test your code. So you should read in the commands and do the corresponding operations.*

Input and Output *Each line contains exactly one command. There are totally 3 types of commands:*

Insert num

Insert ***num*** *into the hash table.* ***num*** *is a non-negative integer smaller than* 2^{31}*.* ***Insert*** *and* ***num*** *are separated by a space. If* ***num*** *has already existed in the hash table, just ignore the command.*

Delete num

Delete ***Delete num*** *from the hash table. No output is required.* ***Delete*** *and* ***num*** *are separated by a space. If there is no such element in the hash table, just ignore the command.*

Find num

Search for ***Delete num*** *in the hash table.* ***Find*** *and* ***num*** *are separated by a space. Output* ***Found num*** *in a line if it is found, otherwise, output* ***num not found****.*

Sample Input

```
Find 54
Delete 35
Insert 8
Insert 8
Insert 17
Delete 8
Find 17
Find 8
```

Sample Output

```
54 not found
Found 17
8 not found
```

Exercise 4.17 *Alice found some magic shapes in the Wonderland. There is a unique number on each shape, i.e., two shapes cannot have the same number. Alice figured out that two shapes can form a key to unlock a door as long as the number on that door is the sum of two numbers on the two shapes. However, Alice is not good at math and she asks you for help. You need to tell her whether a door can be opened or not given all the shapes.*

Input *The first line contains the integer N ($2 \leq N \leq 100000$), the number of shapes, and M ($1 \leq M \leq 20$), the number of doors. Each of the following N lines contains the unique number on one shape. Then, each of the following M lines contains the number on one door. The numbers are in the range of int (from -2^{31} to $2^{31}-1$).*

Output *For each door, you should output 'Yes' if you can find two shapes to unlock the door, and 'No' otherwise.*

Sample Input

```
6 3
-5
5
2
-1
10000
-999
0
1000
-1000
```

Sample Output

```
Yes
No
Yes
```

Explanation $0 = -5 + 5$ *and* $-1000 = -1 - 999$*, so the doors can be opened. However, you cannot find two numbers whose sum is 1000.*

5 Heaps

Exercise 5.1

1. *Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2, one at a time, into an initially empty binary heap.*
2. *Show the result of using the linear-time algorithm to build a binary heap using the same input.*

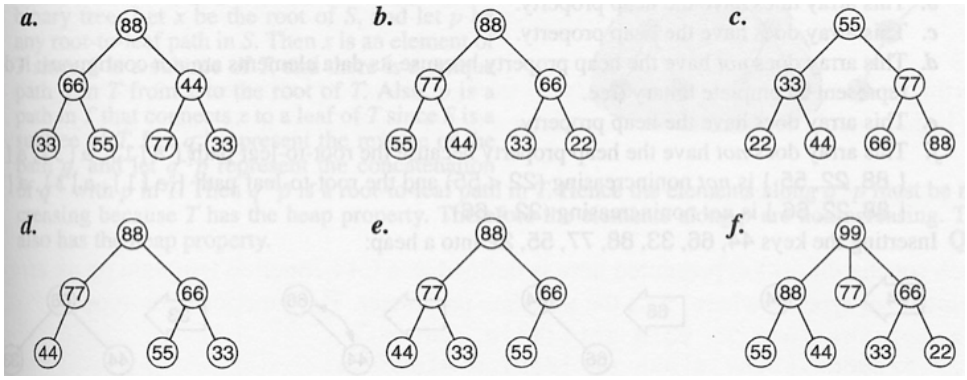
Exercise 5.2 *Show the result of performing three Delete_Min operations in the heap of the previous exercise.*

Exercise 5.3 *What are the two main applications of heaps?*

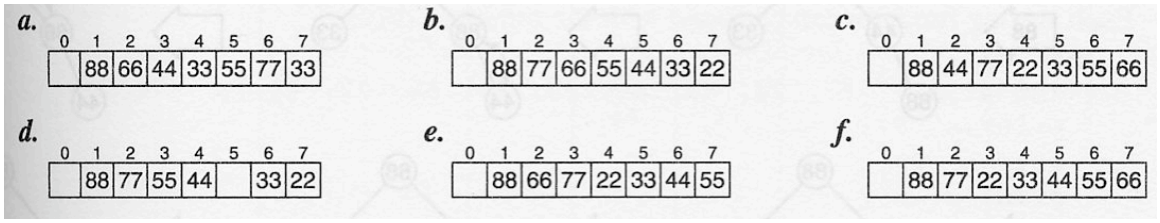
Exercise 5.4 *What is the difference between a queue and a priority queue?*

Exercise 5.5 *Prove that every subtree of a heap is a heap.*

Exercise 5.6 *Determine which of the following binary trees is a heap.*



Exercise 5.7 *Determine which of the following arrays have the heap property.*



Exercise 5.8 *Can one use a priority queue ADT to implement a stack ADT? Justify your answer.*

Exercise 5.9 *Can one use a priority queue ADT to implement a queue ADT? Justify your answer.*

Exercise 5.10 *Tutors do not know how to implement a heap. It is a good chance for you to help your tutors. Please implement a binary heap to store character strings. ASCII code order is used in maintaining the heap property. If you code it well, the tutors will give you candies. Of course, the tutors will use a mass of commands to test your code. So you should read in the commands and do the corresponding operations.*

Input and Output *Each line contains exactly one command. There are totally 3 types of commands:*

Insert word

Insert the character string ***word*** *into the heap. Each word consists of alphabets only. The length of* ***word*** *is from one to twenty.* ***Insert*** *and* ***word*** *are separated by a space.*

DeleteMin

Delete the alphabetically minimum word from the heap. No output is required. If there is no element in the heap, just ignore the command.

Sort

Output the sequence of character strings in a line. The strings should be sorted in alphabetically ascending order. Every two adjacent strings are separated by a space. If the heap is empty originally, just output an blank line. After output, please empty the heap.

Sample Input

```
Insert car
Sort
Insert bg
Insert chzzz
Insert bh
Sort
Sort
DeleteMin
Insert abc
Insert dez
DeleteMin
Sort
```

Sample Output

```
car
bg bh chzzz

dez
```

Exercise 5.11 ***Simulation of Binary Minheap***

Given a sequence of positive integers, insert these integers one by one into a binary minheap. All the integers are between -16,384 to +16,383.

Given a sequence of positive integers, insert these integers one by one into a binary minheap. All the integers are between 0 to +16,383.

Input *The input consists of the number of test cases, m in the first line and followed by m test cases, Each test case is a sequence of positive integers. Each test case would end with '-1'. There are at most 20 integers in each test case.*

```
3
1 2 3 -1
3 2 1 -1
2 89 100 9 8 1 -1
```

Output *The output should be m lines of numbers. Each line is just the breadth-first traversal (top-down and left-to-right) of the minheap in each test case.*

```
1 2 3
1 3 2
1 8 2 89 9 100
```

6 Sorting

Exercise 6.1 *About inversions.*

1. *List out the inversion pairs for the input 142, 543, 123, 65, 453, 879, 572, 434, 111, 242, 811, 102. How many are there?*
2. *What is the maximum number of inversions possible for a given set of n unique elements?*
3. *Given two inputs, input A has a higher inversions than B, does it mean that it will take more time to sort input A than B? Justify your answer.*

Exercise 6.2 *Illustrate results after each pass. Sort the sequence 3, 1, 4, 1, 5, 9, 2, 6, 5 using*

1. *bubble sort*
2. *insertion sort*
3. *selection sort*

Exercise 6.3 *Illustrate results after each pass. Using Shellsort to sort the input 9, 8, 7, 6, 5, 4, 3, 2, 1 with the increments* $\{7, 3, 1\}$.

Exercise 6.4 *Determine the running time of mergesort for*

1. *sorted input*
2. *reverse-ordered input*
3. *random input*

Exercise 6.5 *Show how heapsort processes the input 142, 543, 123, 65, 453, 879, 572, 434, 111, 242, 811, 102.*

Exercise 6.6 *Illustrate results after each pass. Sort 3, 1, 4, 1, 5, 9, 3, 6, 5, 3, 5 using*

1. *quicksort with middle element partitioning and a cutoff of 3.*
2. *quicksort with median-of-three partitioning and a cutoff of 3.*
3. *quicksort with the largest of the first two nondistinct keys and a cutoff of 3.*
4. *quicksort with the largest of the first two distinct keys and a cutoff of 3.*
5. *quicksort with the average of all keys in the set and a cutoff of 3.*

Exercise 6.7 *Using the quicksort implementation in the reference book, determine the running time of quicksort for*

1. *sorted input*

2. *reverse-ordered input*

3. *random input*

Exercise 6.8 *Construct a permutation of 20 elements that is as bad as possible for quicksort using median-of-three partitioning and a cutoff of 3.*

Exercise 6.9 *A sorting algorithm is* stable *if elements with equal keys are left in the same order as they occur in the input. Which of the sorting algorithms in this chapter are stable and which are not? Why?*

Exercise 6.10 ***Calculation of Inversions***

Statement of the Problem *Let $a_1, a_2, \ldots, a_n$ be a permutation of the set $\{1, 2, \ldots, n\}$. If $i < j$ and $a_i > a_j$, the pair (a_i, a_j) is called an "inversion" of the permutation; for example, the permutation $3, 1, 4, 2$ has three inversions: (3, 1), (3, 2), and (4, 2). Each inversion is a pair of elements that is "out of sort," so the only permutation with no inversions is the sorted permutation $1, 2, \ldots, n$.*

Input Format *The input file may contain several instances of the problem. The first line tells the number of problem instances. Each problem instance consists of the number of elements in the first line, a set of elements to be sorted, each separated by a space, and a new line with "0" as the separator. The elements are all unique positive integers less than 32768. There will be no more than 20 elements in an instance of problem.*

Output Format *For each instance of the problem, your program should print the case number, number of inversions, and followed by the inversion pairs, sorted by the first element and then the second element in an ascending order, each in a new line.*

Sample Input

```
3
4
3 1 4 2
0
5
99 10 1 4 33
0
6
10 20 30 40 50 60
0
```

Sample Output

```
1
3
3 1
3 2
4 2
2
6
10 1
10 4
99 1
99 4
99 10
99 33
3
0
```

Exercise 6.11 ***Implementation of Quicksort***

Statement of the Problem *You are asked to implement the Quicksort algorithm. You may use anyone of the method you find best to select the pivot element. Indicate clearly at the top of your program file which pivot selection method you are implementing. Problem 5 should provide you with a good simulation before you go ahead to implement the Quicksort algorithm.*

Input Format *The input file may contain several instances of the problem. The first line tells the number of problem instances. Each problem instance consists of the number of elements in the first line, a set of elements to be sorted, each separated by a space, and a new line with "0" as the separator. The elements are all positive integers less than 32768.*

Output Format *For each instance of the problem, your program should print the case number in a line by itself and the sorted numbers in a new line separated by a white space between numbers. The output should be in a non-descending order.*

Sample Input

```
3
4
3 1 4 2
0
5
99 10 1 4 33
0
6
10 20 30 40 50 60
0
```

Sample Output

```
1
1 2 3 4
2
1 4 10 33 99
3
10 20 30 40 50 60
```

7 Graphs

Exercise 7.1 *Find a topological ordering for the graph below. Illustrate intermediate steps with a table and figures. (Use the table similar to Figure 9.6 and illustrate after each step with an intermediate figure of the graph).*

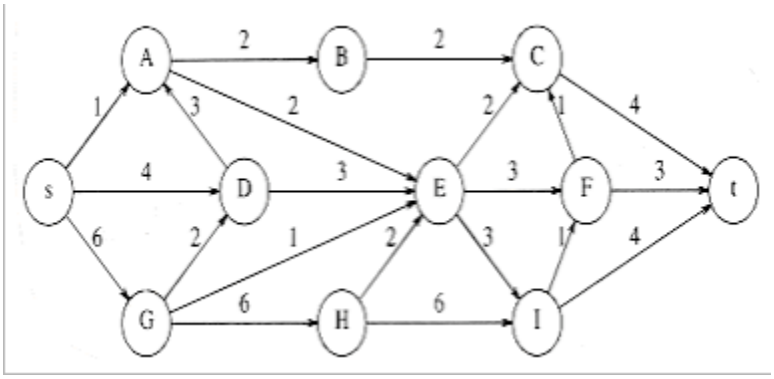


Figure 3: Find the topological ordering.

Exercise 7.2 *If a stack is used instead of a queue for the topological sort algorithm in Section 9.2, does a different ordering result? Why might one data structure give a "better" answer?*

Exercise 7.3

1. *Find the shortest path from A to all other vertices for the graph below. Illustrate intermediate steps with a table and figures.*
2. *Find the shortest unweighed path from B to all other vertices for the graph below. Illustrate intermediate steps with a table and figures.*

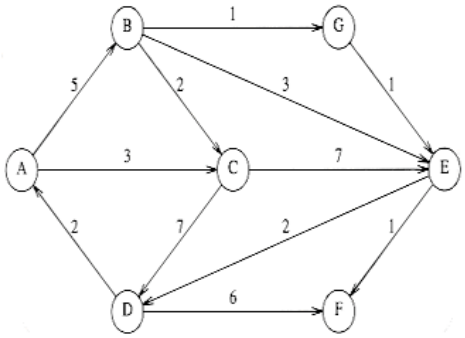


Figure 4: Find the shortest path and unweighed path.

Exercise 7.4 *Find the maximum flow in the network of Fig. 3 above. Illustrate intermediate steps with a table and figures.*

Exercise 7.5

1. *Find a minimum spanning tree for the graph below using both Prim's and Kruskal's algorithm. Illustrate intermediate steps with a table and figures.*
2. *Is this minimum spanning tree unique? Why?*

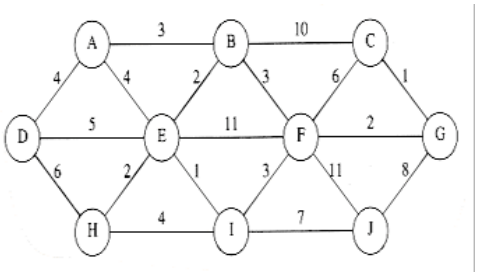


Figure 5: Find the minimum spanning tree.

Exercise 7.6 *Give an algorithm to find a* maximum *spanning tree. Is this harder than finding a minimum spanning tree?*

Exercise 7.7 *Draw a graph with 5 nodes that is*

1. *strongly connected*
2. *weakly connected*
3. *completely connected*
4. *undirected and fully connected*
5. *directed acyclic graph with weight*

Exercise 7.8 ***From the On-Line Judge***

1. ***Minimum Spanning Tree (MST)***
   (a) *http://acm.uva.es/p/v103/10397.html*
   (b) *http://acm.uva.es/p/v103/10307.html*
   (c) *http://acm.uva.es/p/v103/10369.html*
2. ***Shortest Path***
   (a) *http://acm.uva.es/p/v5/567.html*
   (b) *http://acm.uva.es/p/v3/383.html*

Exercise 7.9 File Names

Input File: *q8.in*

Output File: *q8.out*

Statement of the Problem

You are asked to write a program to find the shortest path from a node to all the nodes in a directed graph.

Input Format

The input file may contain several instances of the problem. The first positive integer in the file tells the number of problem instances. Each problem consists of a number of lines followed by the source node id. Each problem instance is terminated by a single '-1' all by itself in a new line. Each line in the problem instance consists of three elements: the begin node id, the end node id, and the cost. The node id and cost are all positive integers. The node id is unique. There will be no multiple paths from one node to another node directly. There will be no more than 20 totally unique nodes in each problem instance. You may assume that the input file is correct.

Output Format

For each instance of the problem, your program should print the shortest distance from the source node to all other nodes in an increasing order. We define the node to itself has a cost of 0. Each output of an instance is terminated with a blank line.

Sample Input

```
2
1 2 10
2 3 20
3 4 20
1 4 5
3
-1
4 3 20
1 2 10
2 1 10
2 3 20
3 4 20
3 2 20
1 4 5
4 1 5
1
-1
```

Sample Output

```
1 Cannot be reached
2 Cannot be reached
3 0
4 20

1 0
2 10
3 25
4 5
```

Exercise 7.10 ***Tree*** *(c.f.* `http://acm.uva.es/p/v8/820.html`*)*

Background

On the Internet, machines (nodes) are richly interconnected, and many paths may exist between a given pair of nodes. The total message-carrying capacity (bandwidth) between two given nodes is the maximal amount of data per unit time that can be transmitted from one node to the other. Using a technique called packet switching, this data can be transmitted along several paths at the same time.

For example, the following figure shows a network with four nodes (shown as circles), with a total of five connections among them. Every connection is labeled with a bandwidth that represents its data-carrying capacity per unit time.

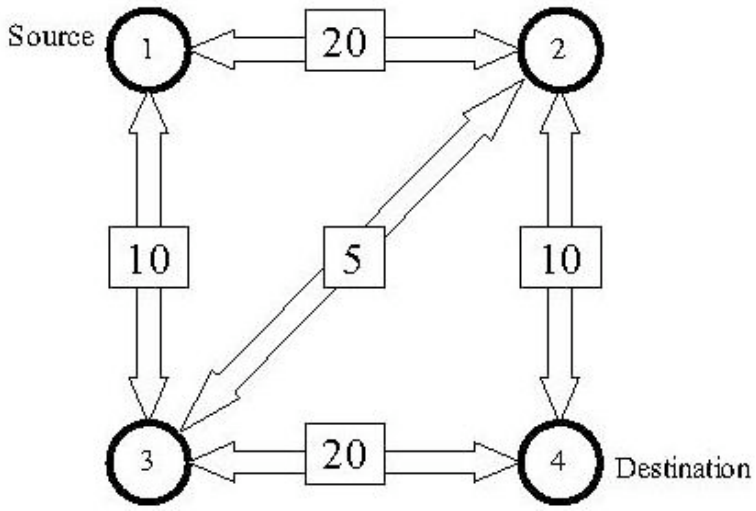


Figure 6: Internet Bandwidth.

In our example, the bandwidth between node 1 and node 4 is 25, which might be thought of as the sum of the bandwidths 10 along the path 1-2-4, 10 along the path 1-3-4, and 5 along the path 1-2-3-4. No other combination of paths between nodes 1 and 4 provides a larger bandwidth.

You must write a program that computes the bandwidth between two given nodes in a network, given the individual bandwidths of all the connections in the network. In this problem, assume that the bandwidth of a connection is always the same in both directions (which is not necessarily true in the real world).

Input

The input file contains descriptions of several networks. Every description starts with a line containing a single integer $n(2 \leq n \leq 100)$*, which is the number of nodes in the network. The nodes are numbered from 1 to n. The next line contains three numbers s, t, and c. The numbers s and t are the source and destination nodes, and the number c is the total number of connections in the network. Following this are c lines describing the*

connections. Each of these lines contains three integers: the first two are the numbers of the connected nodes, and the third number is the bandwidth of the connection. The bandwidth is a non-negative number not greater than 1000.

There might be more than one connection between a pair of nodes, but a node cannot be connected to itself. All connections are bi-directional, i.e., data can be transmitted in both directions along a connection, but the sum of the amount of data transmitted in both directions must be less than the bandwidth.

A line containing the number 0 follows the last network description, and terminates the input.

Output

For each network description, first print the number of the network. Then print the total bandwidth between the source node s and the destination node t, following the format of the sample output. Print a blank line after each test case.

Sample Input

```
4
1 4 5
1 2 20
1 3 10
2 3 5
2 4 10
3 4 20
0
```

Sample Output

```
Network 1
The bandwidth is 25.
```