DD1320 Tillämpad datalogi

A-labben
A1 Paper

1 Task description

1.1 Summary

Björn likes the square root of two, $\sqrt{2} = 1.41421356\ldots$ very much. He likes it so much that he has decided to write down the first 10000 digits of it on a single paper. He started doing this on an A4 paper, but ran out of space after writing down only 1250 digits. Being pretty good at math, he quickly figured out that he needs an A1 paper to fit all the digits. Björn doesn't have an A1 paper, but he has smaller papers which he can tape together.

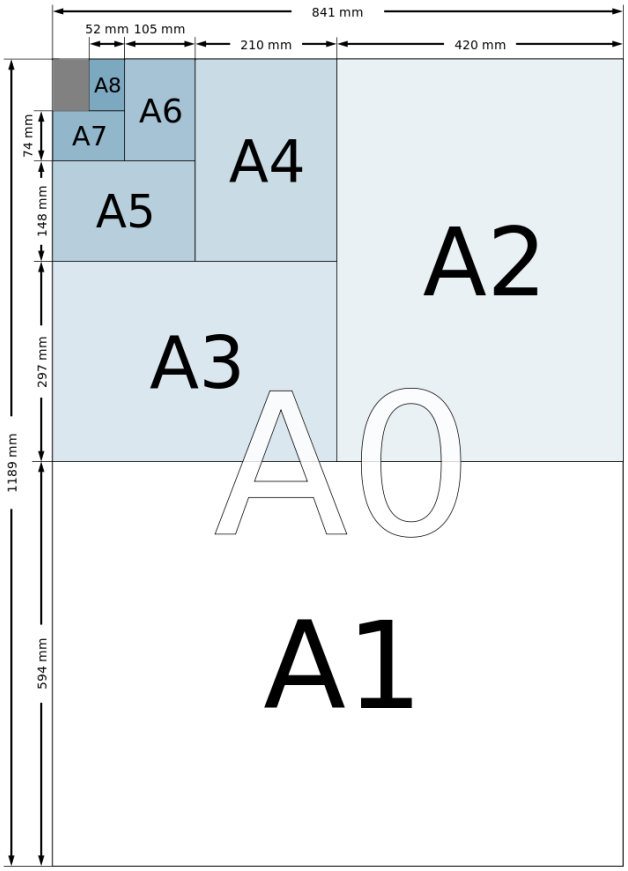

Figure 1: Paper Size Comparison Chart

Taping two A2 papers together along their long side turns them into an A1 paper, two A3 papers give an A2 paper, and so on. Given the number of papers of different sizes that Björn has, can you figure out how much tape he needs to make an A1 paper? Assume that the length of tape needed to join together two sheets of papers is equal to their long side. An A2 paper is $2^{-5/4}$ meters by $2^{-3/4}$ meters and each consecutive paper size (A_3, A_4,) have the same shape but half the area of the previous one. The first line of input contains a single integer $2 \leq n \leq 30$, the A-size of the smallest papers Björn has. The second line contains $n-1$ integers giving the number of sheets he has of each paper size starting with A2 and ending with A n. Björn doesn't have more than 10^9 sheets of any paper size.

1.2 Output

If Björn has enough paper to make an A1 paper, output a single floating point number, the smallest total length of tape needed in meters. Otherwise output "impossible". The output number should have an absolute error of at most 10^{-5}.

2 Kattis

`https://kth.kattis.com/courses/DD1320/tildav23/assignments/jc5hof/submissions/11101672`

3 Data structures

Since the solution for this problem is mostly focused on recursion there aren't any complex data structures, like trees or graphs, used in this code. Multiple single-value variables are used to hold data throughout the program. The most complex data structure used in this code is a list used to save user input and represents the quantities of available paper at each level.

4 Algorithm

We want to find the minimal length of tape required to tape together smaller sizes of A-papers into an A1-sized paper. This naturally hints that the main idea behind the solution is recursion. First we need to figure out if we have enough paper to make an A1 paper. Then under the assumption that we have enough paper we need to figure out how much tape do we need. So by following this we can divide the algorithm into several parts.

Input and the data needed

Since the count of papers always begins from A2 and then progresses to smaller sizes the only information we need besides the number of different size papers we are working with is the dimensions of the A2 sheet of paper.

We are relay only interested in the number of different A-size papers available. This is why we can ignore the first line, which represents the smallest paper size Björn has and only focus on the second line that includes the count of papers for each size starting from A2.

Pre-check

Now that know what we are working with we need to define a process that would check if we have enough paper to make an A1 sheet. Our target is to have two A2 sheets of paper. We can do this by iteratively adding the area of available papers, until we reach or exceed the target area of two A2 sheets of paper meaning we have enough resources to make an A1 sheet. If we don't have enough paper area, it is "impossible" to make an A1 sheet and we can give up.

Recursive process

If there's enough paper area to make two A2 sheets, the algorithm then enters the main recursive process, to calculate the minimal length of tape required.

For each size of paper, we first calculate the length of the tape required to tape two pieces of that size of paper together. This is done by dividing the length or width of an A2 paper by powers of 2, depending on the current level of recursion, which represents the size of paper.

Next, we need to check if there's enough paper of the current size. If there is, we returns the tape length required. If there isn't, we calculate the amount of the next smaller size of paper needed and go into the next recursion level with the updated tape length and paper requirement.

Output

For the result of the recursive process we have two cases:

If the recursive process results in a value, we found the minimal total length of tape required.

Otherwise we can assume there's not enough paper and it is "impossible" to create a A1 sheet.

4.1 Short hand

Set constants for A2 LENGTH and A2 WIDTH.

MAIN FUNCTION:

1. Ignore the first line of user input.
2. Take a second line of input, representing the amounts of available paper.
3. Convert this input into a list of integers.
4. Check if there is enough paper to create an A2 sheet:
   - If there is, perform a recursive calculation to compute the length of tape required.
   - If the recursive function returns a value, print this value otherwise (returns None), print "impossible".
   - If there isn't enough paper to start with, print "impossible".

RECURSIVE FUNCTION:

1. Based on the current index and the need for paper, calculate the required length of tape at this level.
2. If there is enough available paper for this level, return the total tape length.
3. If there isn't enough paper, calculate the paper needed for the next level and recursively call the function for the next level.

CHECKING FUNCTION:

1. Begin with an index of 0, a target area of 2.0, and an available area of 0.0.
2. While there is still paper and the available area has not reached the target area:
   - Calculate the new available area based on the current index and increment the index.
   - Check if the available area is now greater than or equal to the target area. If it is, return that there is enough paper.
   - Continue this until there is enough paper or all paper has been considered.

5 Time complexity

The time complexity depends on the time complexity of the recursion function and the the time complexity of the checking function.

Checking function loops through each element in the data list once, so it has a time complexity of $O(n)$, where n is the size of the list.

Recursion function can loop through each level of paper size recursively in the worst-case scenario. Hence, we can also assume it has a time complexity of $O(n)$, where n is the number of levels of paper meaning the size of the list.

Thus the overall time complexity of your code is $O(n)$ where n is the number of different paper sizes that Björn has. The time complexity is linear.

6 Appendix 1: Source code

```
import math

# We will use the following constants in our program
A2_LENGTH = 0.5946035575013605
A2_WIDTH = 0.42044820762685725

def main():

    # First I need to deal with the input. I will read the first line and ignore it.
    # The only think we are interested in is the second line
    # which contains the the amount of paper available.

    _ = input()
    resources = input()

    # Here I will convert the string of numbers into a list of integers.
    resources_list = list(map(int, resources.split()))

    # Now I will call the function that will do the actual work.
    if check_if_enough_paper(resources_list):

        # if there is enough paper, I will call the recursion function.
        tape_length = recursion(0, 2, 0, resources_list)

        # if the recursion fuction return a value, I will print it.
        if tape_length is not None:
            print(tape_length)

        # if the recursion function return None, I will print impossible.
        else:
            print("impossible")

    # if there is not enough paper, I will print impossible.
    else:
        print("impossible")


def recursion(index, need_to_use, tape_length, data):

    # Here I check if we have some paper to work with.
    if index >= len(data):
        return None

    # calculate the length of tape required at a particular level in the recursive function
    # if the index is even then the length of tape required is A2_LENGTH / (2 ** (indx / 2.0))
    # if the index is odd then the length of tape required is A2_WIDTH / (2 ** ((indx - 1) / 2.0))
    if index % 2 == 0:
        length = A2_LENGTH / (2 ** (index / 2.0))
    else:
        length = A2_WIDTH / (2 ** ((index - 1) / 2.0))

    #avaliable_paper is the amount of paper available at a particular level in the recursive function.
    available_paper = data[index]

    #tape_length is the total length of tape required to make joints at a particular level in the recur
    #need_to_use is the amount of paper required to make joints at a particular level in the recursive
    tape_length +=(need_to_use / 2.0) * length

    # if the available paper is less than the paper required at a particular level in the recursive fun
```

```
    # then we need to go to the next level
    if available_paper >= need_to_use:
        return tape_length
    else:
        next_level_need_to_use = (need_to_use - available_paper) * 2 # calculate next level need to use
        return recursion(index + 1, next_level_need_to_use, tape_length, data) # return recursion call

# This function checks if there is enough paper to make an A2 sheet.
def check_if_enough_paper(data):

    index = 0
    target_area = 2.0
    available_area = 0.0

    # enough is a boolean variable that will be true if there is enough paper.
    enough = available_area > target_area or math.isclose(available_area, target_area)

    # while there is still paper and we don't have enough...
    while not enough and index < len(data):

        # calculate available area
        available_area = available_area + data[index] * (2 ** -index)

        # check if enough
        enough = available_area >= target_area or math.isclose(available_area, target_area)

        # increment index
        index += 1

    return enough # return if enough


if __name__ == "__main__":
    main()
```

7 Appendix 2: Test data

7.1 Normal Case

Here we just test the Normal Case of having enough sheets of paper.

So the input would be:

4
1 2 0

The expected output is: 1.0150517651282178

7.2 Edge Case

Here we just test the Edge Case of having two A2 sheets which is the minimum required to make an A1 sheet.

So the input would be:

4
2 0 1

The expected output is the length of A2 sheet: 0.5946035575013605

7.3 Invalid Case

Here we just test cases where the output should be "impossible"

So the input would be:

5
0 0 1 0

The expected output is: *impossible*

7.4 Floating point

Given the nature of your program where calculations are made using floating point numbers. The problem is defined so that we cant have partial parts of a sheet. We need to check that the program does not give solutions for inputs like:

5
1.99 0 1 0

The expected output is: *error*

7.5 Alternating paper availability

Test the program's ability to correctly skip over sizes of paper that are not available.

So the input would be:

6
1 0 2 0 8

The expected output is: 2.624707087757796