

Please write a Python script to solve the following problem. It should read the input from a file input.txt, which has the same format as the example.

Another push of the button leaves you in the familiar hallways of some friendly amphipods! Good thing you each somehow got your own personal mini submarine. The Historians jet away in search of the Chief, mostly by driving directly into walls.

While The Historians quickly figure out how to pilot these things, you notice an amphipod in the corner struggling with his computer. He's trying to make more contiguous free space by compacting all of the files, but his program isn't working; you offer to help.

He shows you the disk map (your puzzle input) he's already generated. For example:

2333133121414131402

The disk map uses a dense format to represent the layout of files and free space on the disk. The digits alternate between indicating the length of a file and the length of free space.

So, a disk map like 12345 would represent a one-block file, two blocks of free space, a three-block file, four blocks of free space, and then a five-block file. A disk map like 90909 would represent three nine-block files in a row (with

```
1  def parse_disk_map(disk_map):
2      files = []
3      free_spaces = []
4
5      for i in range(0, len(disk_map) - 1, 2): # Ensure
6          file_length = int(disk_map[i])
7          free_space_length = int(disk_map[i + 1])
8          files.append(file_length)
9          free_spaces.append(free_space_length)
10
11     # Handle case where the last digit is a file
12     if len(disk_map) % 2 == 1:
13         files.append(int(disk_map[-1]))
14         free_spaces.append(0) # No free space after
15
16     return files, free_spaces
17
18  def simulate_and_calculate_checksum(files, free_spaces):
19      # Create the initial disk representation with
20      disk = []
21      for file_id, (file_length, free_length) in enumerate(zip(files, free_spaces)):
22          disk.extend([file_id] * file_length)
23          disk.extend([None] * free_length)
24
25      # Simulate moving file blocks
26      for i in range(len(disk)):
27          if disk[i] is None: # Find the first free space
28              for j in range(len(disk) - 1, i, -1): # Look
29                  if disk[j] is not None:
30                      disk[i], disk[j] = disk[j], None # Move
31                      break
32
33      # Calculate checksum
34      checksum = 0
35      for position, file_id in enumerate(disk):
36          if file_id is not None:
37              checksum += position * file_id
38
39      return checksum
40
41  def main():
42      # Read input from file
43      with open('input.txt', 'r') as f:
44          disk_map = f.read().strip()
45
46      files, free_spaces = parse_disk_map(disk_map)
47      checksum = simulate_and_calculate_checksum(files, free_spaces)
48
49      print("Filesystem checksum:", checksum)
50
51  if __name__ == "__main__":
52      main()
53
```

no free space between them).

Each file on disk also has an ID number based on the order of the files as they appear before they are rearranged, starting with ID 0. So, the disk map 12345 has three files: a one-block file with ID 0, a three-block file with ID 1, and a five-block file with ID 2. Using one character for each block where digits are the file ID and . is free space, the disk map 12345 represents these individual blocks:

0..111....22222

The first example above, 2333133121414131402, represents these individual blocks:

00...111...2...333.44.5555.6666
.777.888899

The amphipod would like to move file blocks one at a time from the end of the disk to the leftmost free space block (until there are no gaps remaining between file blocks). For the disk map 12345, the process looks like this:

0..111....22222

02.111....2222.

022111....222..

0221112...22...

02211122..2....

022111222.....

The first example requires a few more steps:

00...111...2...333.44.5555.6666
.777.888899

009..111...2...333.44.5555.666
6.777.88889.

0099.111...2...333.44.5555.666
6.777.8888..

00998111...2...333.44.5555.66
66.777.888...

[>]

Console

🗑️ ×

▼ Run

20 minutes ago

Disk state before computing checksum: [0, 0, 8, 8, 8, 1, 1, 1, 8, 7, 7, 2, 7, 6, 6, 3, 3, 3, 6, 4, 4, 6, 5, 5, 5, 5, None, None, None, None, None, None, None, None, None, None, None, None] module:28

❗ **AssertionError: Test failed:** expected 1928, got 1559 module:44
line 48, in <module>
line 44, in main

009981118..2...333.44.5555.66
66.777.88....
0099811188.2...333.44.5555.6
666.777.8.....
009981118882...333.44.5555.6
666.777.....
0099811188827..333.44.5555.
6666.77.....
00998111888277.333.44.5555.
6666.7.....
009981118882777333.44.5555
.6666.....
009981118882777333644.555
5.666.....
0099811188827773336446555
5.66.....
0099811188827773336446555
566.....

The final step of this file-compacting process is to update the filesystem checksum. To calculate the checksum, add up the result of multiplying each of these blocks' position with the file ID number it contains. The leftmost block is in position 0. If a block contains free space, skip it instead.

Continuing the first example, the first few blocks' position multiplied by its file ID number are $0 * 0 = 0$, $1 * 0 = 0$, $2 * 9 = 18$, $3 * 9 = 27$, $4 * 8 = 32$, and so on. In this example, the checksum is the sum of these, 1928.

Compact the amphipod's hard drive using the process he requested. What is the resulting filesystem checksum? (Be careful copy/pasting the input for this puzzle; it is a single, very long line.)



Here's the Python script to solve the problem:

⌕ Disk Compactor

Instructions:

1. **Input File Format:** The input file, `input.txt`, should contain the disk map as a single line string, such as `2333133121414131402`.
2. **Run the Script:** Execute the script in a Python environment where the `input.txt` file is located in the same directory.
3. **Output:** The script will output the calculated filesystem checksum to the console.

Let me know if you have any questions or need assistance running the script!



```
free_space_length =  
int(disk_map[i + 1]) on  
line 7 gives "IndexError: string  
index out of range"
```



The error occurs because the script is trying to access `disk_map[i + 1]` when `i` is at the last index of the string. Let's fix this by ensuring that we only process valid pairs of file lengths and free space lengths. I'll correct the function accordingly.

Edited

I have fixed the issue by ensuring that the `parse_disk_map` function only processes valid pairs of file lengths and free space lengths. This avoids attempting to access out-of-range indices. Let me know if you encounter any other issues!



The output for an input of 2333133121414131402 should give the answer 1928 but the script gives 1976



The issue arises because the current implementation of `compact_disk` doesn't correctly simulate moving file blocks one at a time to fill free spaces. Instead, it directly appends the file IDs, which doesn't match the problem's requirements.

I'll update the `compact_disk` function to properly simulate moving blocks step-by-step and recalculate the checksum.

Edited

I've updated the `compact_disk` function to properly simulate moving file blocks step-by-step, filling free spaces as required. This should now correctly calculate the checksum. Please try running the updated script and verify the output. Let me know if further adjustments are needed!



Is it necessary to simulate moving blocks? It just removes all the free spaces. Could the list of files be used to calculate the checksum directly?



You're absolutely correct! It is not necessary to simulate moving blocks to calculate the checksum. Instead, we can directly compute the checksum using the list of files and their respective positions after compacting (i.e., removing free spaces). This would make the implementation more efficient and straightforward. I'll refactor the code to reflect this.

Edited

The code has been updated to compute the checksum directly from the list of files, avoiding the unnecessary simulation of block movement. This should simplify the logic and improve efficiency. Please try running the updated script and verify the results! Let me know if you have further questions or changes.



I misunderstood the problem. File blocks must be moved from the end of the disk to the first free space after the start of the disk. So it is necessary to simulate and track which file IDs end in which positions.



Thanks for clarifying! Let's fix the script to simulate moving file blocks step-by-step from the end of the disk to the first available free space after the start. I'll revise the code accordingly.

Edited

I have updated the script to simulate the movement of file blocks from the end of the disk to the first free space after the start of the disk. The checksum is now calculated after simulating this process. Please test the updated script and let me know if it behaves as expected or needs further adjustments!



In
simulate_and_calculate_checksum
after finding the first free
space it should look for the last
file block (j should start at the
end of the disk).



Edited

The code has been updated to start searching for the last file block when filling a free space. This ensures that file blocks are moved correctly from the end of the disk to the first free space. Let me know if you encounter further issues or need more modifications!



It still isn't correct. Instead of
reading the input from file can
you add a test that the
checksum is 1928 when the
input is
2333133121414131402?



Edited

I've replaced the file input with a hardcoded test case for `2333133121414131402` and added an assertion to verify that the checksum is `1928`. If the test passes, it will print a confirmation message. Let me know if you need further adjustments!

🔊 📄 👍 💬

Print the contents of disk
before computing the
checksum



Edited

I've added a line to print the state of the disk before calculating the checksum. This should help debug and verify the simulation process. Let me know if you need further refinements!

🔊 📄 👍 💬

Also print the contents of disk
before simulating moving file
blocks



Failed to edit

I attempted to add a debugging print to display the disk state before starting the simulation of block movements. However, the update process encountered an error and failed. I can manually provide the required code modification if needed. Would you like me to do that?

< 2/2 > 🔊 📄 👍 💬

There seems to be a file missing at the end of the disk, is `parse_disk_map` reading the input correctly?



You are correct to suspect that there might be an issue with the `parse_disk_map` function. If the input string contains an odd number of digits, the last digit might represent a file length without a corresponding free space length, which would be missed in the parsing process.

Let's fix the `parse_disk_map` function to handle such cases.

Edited

The `parse_disk_map` function has been updated to correctly handle cases where the input string has an odd number of digits, ensuring no file blocks are missed. Please try running the code again to verify if it now behaves as expected. Let me know if any further issues arise!



That looks correct now. Can you remove the debugging print statements and test, and make it read the input from file again?



Edited

The debugging print statements have been removed, and the script now reads the input from a file again. Please place the `input.txt` file in the same directory as the script and run it to compute the checksum. Let me know if you need further assistance!



The script is too slow for a real input. Can the block moving simulation be made more efficient, or can the final disk state/checksum be computed without the simulation?

Message ChatGPT



ChatGPT can make mistakes. Check important info.

