The following algorithm relies on the fact that spaces are a good hint for word separation. The example input’s block is:

\_ë\_odr¥ would you ¤\_Mifä\_ñ9 %Ø\_ IT\_ I$\_I$to solveÿÿÿÿÿÿtask?ÿBrute force? \_$\_I$\_I$\_I$\_JÛ Well$I Ûm¶Û...

\_[ÞWhatx\_^\_ü\_#9\_×odo û3 youðhad<one`hour (\_I$\_I$\_ this I$\_I$ I$\_I$\_I$\_I$\_IRandom search?[m¶Ûm Ûowoul...

What would you do if you had one hour to solve this task? Brute force? Random Search? Well I woul...

These are the two lower blocks (the only relevant ones). They’re place in the matrix at offset 6, char index ~96700.

The algorithm assumes:

* If there’s a space in another string at the following index, that’s where the next word begins.
* If we found an invalid char, we look for any valid char in any of the following strings at the same index. We’ll use the first one right away. It also must follow the “Space separating” rule – to prevent False results.
* Any further errors will be filtered using the message heuristic function. The probability for a False is statistically unlikely at this point, given all the minimal message conditions provided in the assignment.

process\_strings\_collection(strings):

results = []

for string in strings:

a = process\_string(string, strings)

if a is not empty:

results.add(a)

return results

process\_string(work\_string, all\_strings):

if work\_string[0] is not a letter:

return *empty*

message = “”

index = 0

while index < work\_string.length:

# if we see a space, it has a priority – skip to that string as current.

for string in all\_strings:

If string[index] is a space:

work\_string = string

break loop

# current string is not workable anymore. Try to switch to another one.

if work\_string[index] is not space, symbol or letter:

stop = True

for string in all\_strings:

if string[index] is a letter:

work\_string = string

stop = False

if stop:

break loop

# Char is valid for copy, get it and continue.

message += work\_string[index]

index = index+1

return message if heuristic(message) else *empty*

Recursive:

decode\_recursive(strings: List[str], message: str) -> Optional[str]:

message\_valid = is\_message\_valid(message)

for string in strings:

# Invalid char – end of message?

if string[0] not in VALID\_CHARS:

continue

result = decode\_recursive([s[1:] for s in strings], message + string[0])

if result:

return result

return message if message\_valid else None