Algorithm for file updates in Python

Project description

Responsible for periodically updating the list of IP addresses authorized to access the restricted subnet, based on involvement with patients' personal records. Access is granted only to IPs associated with specific roles. A removal list is also maintained to revoke access when necessary.

In this project, a Python algorithm was created to check whether the permission list contains any IP addresses found in the removal list. If so, those IPs are removed from the permissions file, which is then updated to maintain security.

Open the file that contains the allow list

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# First line of `with` statement

with open(import_file, "r") as file:
```

Initially, the *.txt* file containing all IP addresses is assigned to a variable named *import_file*. There is also a second variable, *remove_list*, which stores the IP addresses that need to be removed. Next, the *with* statement is used in conjunction with the *open()* function to open the file containing the allow list. This approach ensures that the file is properly opened in read mode and automatically closed afterward, even if an error occurs during processing, promoting safe and efficient resource management.

Read the file contents

```
with open(import_file, "r") as file:

  # Use `.read()` to read the imported file and store it in a variable named `ip_addresses`

    ip_addresses = file.read()
```

Continuing, the second parameter was set to *"r"* to indicate that the file should be opened in read mode. Then, the *.read()* method was used with the *file* object, and the result was stored in a variable named *ip_addresses*. The *.read()* method reads the entire content of the file as a single string, which allows further processing—such as splitting or filtering—to be performed based on specific criteria, like identifying and removing unwanted IP addresses.

Convert the string into a list

```
# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()

# Display `ip_addresses`

print(ip_addresses)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.170', '192.168
.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219', '192.168.52.37', '192.1
68.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

At this stage, the *.split()* method was used to convert the content of the *ip_addresses* variable from a single string into a list. The *.split()* method, when called without any arguments, separates the string by whitespace characters such as spaces, tabs, or newline characters. In this algorithm, it allows each IP address—originally stored on separate lines in the file—to become an individual element in a list, making it easier to compare and manipulate specific entries, such as identifying those that appear in the *remove_list*.

Iterate through the remove list

```
# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

```
ip_address
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
192.168.52.90
192.168.158.170
192.168.90.124
192.168.186.176
192.168.133.188
192.168.203.198
192.168.201.40
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.58.57
192.168.69.116
```

Next, a *for* loop was created to iterate through each IP address within the *ip_addresses* list, assigning each IP to a temporary variable named *element*. The *for* loop allows sequential access to each item in the list, enabling individual processing or evaluation. In this case, the loop was used to print each IP address to the screen, which can be useful for verifying the data loaded from the file and ensuring that the list was properly parsed before proceeding with removal operations.

Remove IP addresses that are on the remove list

```
for element in ip_addresses:

  # Build conditional statement
  # If current element is in `remove_list`,

    if element in remove_list:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)

# Display `ip_addresses`

print(ip_addresses)
```

```
['ip_address', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.176', '192.168
.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.69.116']
```

Then, an *if* conditional was used within the loop to check whether each *element* from the *ip_addresses* list was also present in the *remove_list*. If the condition was met, the *.remove()* method was called to delete that specific *element* from the original *ip_addresses* list. The *.remove()* method searches for the first occurrence of the specified value and removes it from the list. This is essential in the algorithm to ensure that unauthorized IP addresses are effectively excluded from the updated allow list, enhancing access control and system security.

Update the file with the revised list of IP addresses]

```
# Convert `ip_addresses` back to a string so that it can be written into the text file
ip_addresses = " ".join(ip_addresses)
# Build `with` statement to rewrite the original file
with open(import_file, "w") as file:
  # Rewrite the file, replacing its contents with `ip_addresses`
  file.write(ip_addresses)
```

Finally, the file was updated with the revised list of authorized IP addresses by converting the list back into a string using the *"\n".join()* method. This method combines all elements of the *ip_addresses* list into a single string, separating each IP address with a newline character to maintain the original line-by-line format. A new *with* statement was then used with the second parameter set to *"w"*, indicating write mode, which allows the file to be overwritten. Inside this block, the *.write()* method was used to write the updated string back to the file. The *.write()* method replaces the file's contents with the new data, ensuring the allow list remains current and reflects only valid, authorized IP addresses.

Summary

In this project, a Python algorithm was developed to manage access control based on IP addresses in a healthcare environment. The process begins by assigning a *.txt* file containing all authorized IP addresses to the *import_file* variable and a second variable, *remove_list*, with the IPs to be revoked. Using the *with* statement and the *open()* function in read mode (*"r"*), the file is safely opened, and its content is read into the *ip_addresses* variable using the *.read()* method. This string is then converted into a list using *.split()*, allowing for easier manipulation.

A *for* loop iterates through each *element* (IP address) in the list, and an *if* condition checks whether any of them appear in the *remove_list*. If so, the *.remove()* method is used to eliminate those IPs from the original list. After filtering, the updated list is converted back into a string using *"\n".join()*. Finally, another *with* statement opens the file in write mode (*"w"*) and uses *.write()* to overwrite it with the updated list of authorized IP addresses, ensuring secure and up-to-date access control.