1.まえがき

試験において作成したプログラムの内容、実行方法、実行結果などをここに示す。

2.作成したプログラムの共通事項

　作成したプログラムの名前はFixpointTestN.pyである(Nは設問の番号に対応)。同フォルダ内にある監視ログファイルlogN.txt(Nは設問の番号に対応)を読み込み処理の結果をコマンドライン上に出力する。どのプログラムにもitertoolsとdatetimeライブラリをインポートしている。プログラムの実行はコマンドプロンプトで行い、テストの結果はresult.txtに記録した。PythonのバージョンはPython 3.8.3である。

なお、監視ログファイルは確認日時によってソートされているものとしている。

3.設問1のプログラムについて

　監視ログファイルを文字列として読み込み、行ごとに確認日時, サーバアドレス,応答結果で分割しそれをタプルに格納。各タプルを順番にリストに格納している。このリストをサーバアドレスでソートを行い、アドレスごとにグループ化することでサーバごとに故障しているかの確認と故障時間の出力の処理を行う。

サーバが故障しているかを判断するために変数TO\_stateを用意した。TO\_stateはサーバからの応答がタイムアウトしているときにTrue、応答がある場合にFalseをとる。順番に応答結果を確認して次のパターンでそれぞれ処理を行う。

・タイムアウトしていない状態で応答が無い場合(TO\_state=Falseで応答結果が"-"の場合)

　サーバが故障したとみなして確認日時を変数TO\_clockに格納する。なお、確認日時はstr型で格納されているが、TO\_clockに格納する際に日付型に変換している。

・タイムアウトしている状態で応答があった場合(TO\_state=Trueで応答結果が"-"以外の場合)

　サーバが復旧したとみなして確認日時を変数Recov\_clockに格納する。なお、確認日時はstr型で格納されているが、Recov\_clockに格納する際に日付型に変換している。Recov\_clockからTO\_clockを引くことで故障期間を計算し、結果を「address:サーバアドレス failure period: 故障時間」と出力する。その後TO\_clockはリセットする。

サーバの全ての応答を確認した後にTO\_state=Trueだった場合、サーバは現在故障中であるとみなして「address:サーバアドレス In failure」と出力する。

4.設問2のプログラムについて

　基本的な構造は設問1のプログラムと同様であるが、プログラムのパラメータとしてNと連続で応答が無かった回数を示すTO\_counterを追加。NとTO\_counterを利用した条件分岐を入れることでN回以上連続でのタイムアウトを故障とみなしている。各条件分岐と処理を以下に示す。

・タイムアウトしていない状態で応答が無い場合(TO\_state=Falseで応答結果が"-"の場合)

設問1と同様の処理をするとともにTO\_counterに1を代入する。

・タイムアウトしている状態で応答が無い場合(TO\_state=Trueで応答結果が"-"の場合)

　TO\_counterに1を足す

・タイムアウトしている状態で応答があった場合(TO\_state=Trueで応答結果が"-"以外の場合)

ここでTO\_counterがN以上か(N回以上連続で応答が無いか)で条件分岐をし、TO\_counterがN以上の場合は設問1と同様の処理を行い故障時間を出力。TO\_counterがN未満の場合は故障では無いとして出力は行わない。どちらの場合も処理を行った後にTO\_clockをリセットする。サーバの全ての応答を確認した後にTO\_state=TrueかつTO\_counterがN以上だった場合、サーバは現在故障中であるとみなして「address:サーバアドレス In failure」と出力する。

5.設問3のプログラムについて

プログラムのパラメータにmとtを追加した。また、設問2で実装したサーバの故障判断、故障時間の出力を関数TO\_check(group,key,N)に実装した。また、サーバの直近m回の平均応答時間を返す関数delay\_ave(c,m)とサーバが過負荷状態か判断して過負荷状態となっている期間を出力する関数delay\_check(group,m,t,key)を実装した。

・関数delay\_ave(c,m)

入力された応答結果のリストcの最新の応答時間m個の平均をとる関数。cの要素数がm未満の場合はcの全ての応答時間の平均をとる。なお、タイムアウトの応答"-"が来た場合はその結果を平均応答時間の計算には入れないものとしている。

関数delay\_check(group,m,t,key)

入力のリストgroup、パラメータm、t、サーバアドレスkeyを入力に取っている。基本的に設問1で実装した故障の判断と同様の処理を行っている。リストcheckに応答結果を順番に追加して関数delay\_ave(c,m)に渡すことで平均応答時間を計算し変数ave\_resに格納。

サーバが過負荷状態かを示す変数OL\_stateと、ave\_resがt以上かそうでないかで過負荷状態になったか、過負荷状態から復旧したかを判断している。

過負荷状態から復旧した場合は「address:サーバアドレス failure period: 過負荷状態の時間」を出力し、サーバの全ての応答を確認した後にOL\_state=Trueだった場合、サーバは現在過負荷状態であるとみなして「address:サーバアドレス overload」と出力する。

サーバの故障のチェックを行った後に過負荷のチェックを行い、結果も故障時間の出力の後に過負荷時間の出力がされる。

6.設問4のプログラムについて

新たにipaddressライブラリをインポートして、入力のリストに(確認日時, サーバアドレス,応答結果,ネットワークアドレス)のタプルを順番にリストに格納するようにした。

設問3と同様に設問2で実装したサーバの故障判断、故障時間の出力を関数TO\_check(group,key,N)に実装した。

サブネットの故障判断、故障時間の出力は関数TO\_check\_sub(group,key,N)で実装し、入力のリストをネットワークアドレスでグループ化してTO\_check\_subに渡すことで各ネットワークで故障の判断などを行っている。

関数TO\_check\_sub(group,key,N)

最初に入力リストgroupをサーバアドレスごとにグループ化。ネットワーク内にあるサーバの数を変数host\_countに格納、サーバそれぞれに番号を割り振り、辞書host\_numにサーバアドレスと番号の対応を格納する。次にgroupを確認日時でソートし直して処理を行う。処理は基本的に設問2の処理である関数TO\_check(group,key,N)と同様だが、TO\_stateとTO\_counterはネットワーク内のサーバそれぞれに用意して、リストに格納している。サブネットが故障しているかはTO\_stateが全てTrue、つまり全てのサーバが故障状態であるかで判断し、それぞれのサーバのTO\_stateは以下のように処理している。

・全てのTO\_state=Trueでは無いときに応答結果が"-"の場合

サーバのTO\_counterに1を足してTO\_counterがN以上の場合サーバのTO\_stateをTrueにする。このとき全てのサーバのTO\_stateがTrueになった場合サブネットが故障したとして確認日時を変数TO\_clockに格納する。

・全てのTO\_state=Trueでは無いときに応答結果が"-"以外の場合

サーバのTO\_stateをFalseにしてサーバのTO\_counterをリセットする。

・全てのTO\_state=Trueのときに応答結果が"-"以外の場合

サブネットが復旧したとして確認日時を変数Recov\_clockに格納する。Recov\_clockからTO\_clockを引くことで故障期間を計算し、結果を「subnet address:ネットワークアドレス failure period: 故障時間」と出力する。その後TO\_clockはリセットする。

サーバの全ての応答を確認した後に全てのTO\_state=Trueだった場合、サブネットは現在故障中であるとみなして「subnet address:ネットワークアドレス In failure」と出力する。