

株式会社フィックスポイント社プログラミング試験問題 設計プログラム説明書

2021年3月21日

宇都宮大学大学院
地域創生科学研究科工農総合科学専攻1年
藤田 樹

1. プログラムの実行方法

プログラムは以下の手順で実行する.

- 1) pythonをインストールしたOSへプログラムをダウンロードする.
- 2) 読み込む監視ログファイルを“logfile.txt”という名前でプログラムと同じフォルダへ保存する.
- 3) コマンド上にて「python (実行プログラムのファイル名)」と入力する.

2. プログラムの仕様

2.1. 設問1のプログラム“problem1.py”の仕様

2.1.1. プログラムの概要

このプログラムは, 監視ログファイルのログをもとに故障状態のサーバアドレスとその故障期間をコマンドプロンプトに一覧として出力するプログラムである.

出力形式は「<サーバアドレス>, <故障期間>」とした.

2.1.2. データ構造

① 定数宣言

なし

② グローバル変数

以下の3つの変数にてサーバの情報を管理する. サーバの管理は確認順に要素番号をつけていくことで識別を行う.

servrList	: string 型リスト. サーバアドレスの一覧
serverConnectLastTime	: string 型リスト. 応答を確認した最新の時間
failureStatus	: int 型リスト. サーバの状態 (0:故障中, 1:稼働中)

2.1.3. アルゴリズム

図1に“problem1.py”のPAD図を記載する.

まず, ログファイルを一行読み込み, カンマで分割することで変数リストlineに<確認日時>, <サーバアドレス>, <応答時間>の情報をリストとして格納する.

この時, サーバアドレスが未確認の際はサーバの情報をグローバル変数へ登録する.

次に, サーバアドレスをもとに該当するサーバを格納している要素番号を探し出す. ここでサーバの状態が稼働中であれば, 該当サーバの確認時間を更新する.

その後、応答時間を確認する。応答がなければ該当サーバの状態を故障中に更新する。応答があれば、該当サーバの状態を確認する。この時に故障中であれば、serverConnectLastTimeの時間と読み込んだ<応答時間>から故障期間を計算し、「該当サーバアドレスと故障期間」を出力する。出力後、該当サーバの状態を稼働中に戻す。これをログファイルすべて読み込むまで繰り返す。

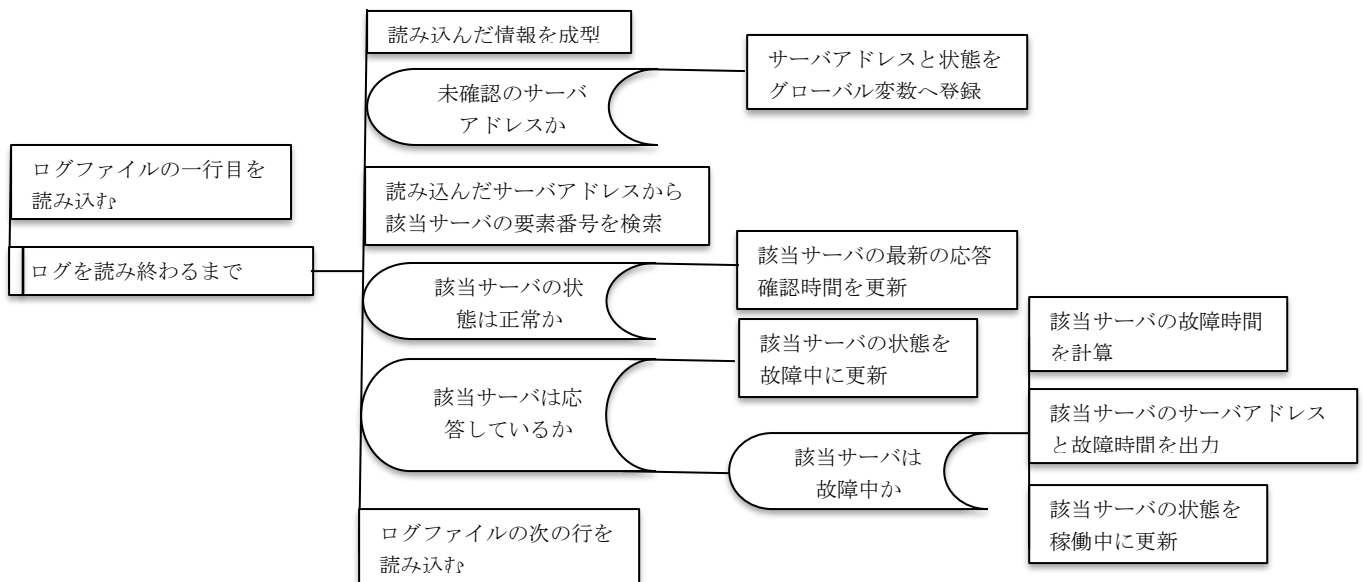


図1 “problem1.py” のPAD図

2.2. 設問2のプログラム“problem2.py”の仕様

2.2.1. プログラムの概要

このプログラムは、設問1のプログラムをもとに作成した、監視ログファイルのログをもとにN回連続で応答がなかったサーバを故障状態と見なし、そのアドレスとその故障期間をコマンドプロンプトに一覧として出力するプログラムである。

出力形式は「<サーバアドレス>,<故障期間>」とした。

2.2.2. データ構造

① 定数宣言

N : int 型定数, サーバのタイムアウト許容回数.

② グローバル関数

設問1同様に、以下の3つの変数にてサーバの情報を管理する。サーバの管理は確認順に要素番号をつけていくことで識別を行う。

servrList : string 型リスト. サーバアドレスの一覧
serverConnectLastTime : string 型リスト. 応答を確認した最新の時間
failureStatus : int 型リスト. タイムアウト許容回数(初期値 N)

2.2.3. アルゴリズム

図2に “problem2.py” のPAD図を記載する。

まず，ログファイルを一行読み込み，カンマで分割することで変数リストlineに<確認日時>,<サーバアドレス>,<応答時間>の情報をリストとして格納する。

この時，サーバアドレスが未確認の際はサーバの情報をグローバル変数へ登録する。

次に，サーバアドレスをもとに該当するサーバを格納している要素番号を探し出す。ここでサーバの状態が稼働中であれば，該当サーバの確認時間を更新する。

その後，応答時間を確認する。応答がなければ該当サーバのタイムアウト許容回数を1減少させる。応答があれば，該当サーバの状態を確認する。この時に故障中であれば，serverConnectLastTimeの時間と読み込んだ<応答時間>から故障期間を計算し，「該当サーバアドレスと故障期間」を出力する。出力後，該当サーバのタイムアウト許容回数をNにリセットする。

これをログファイルすべて読み込むまで繰り返す。

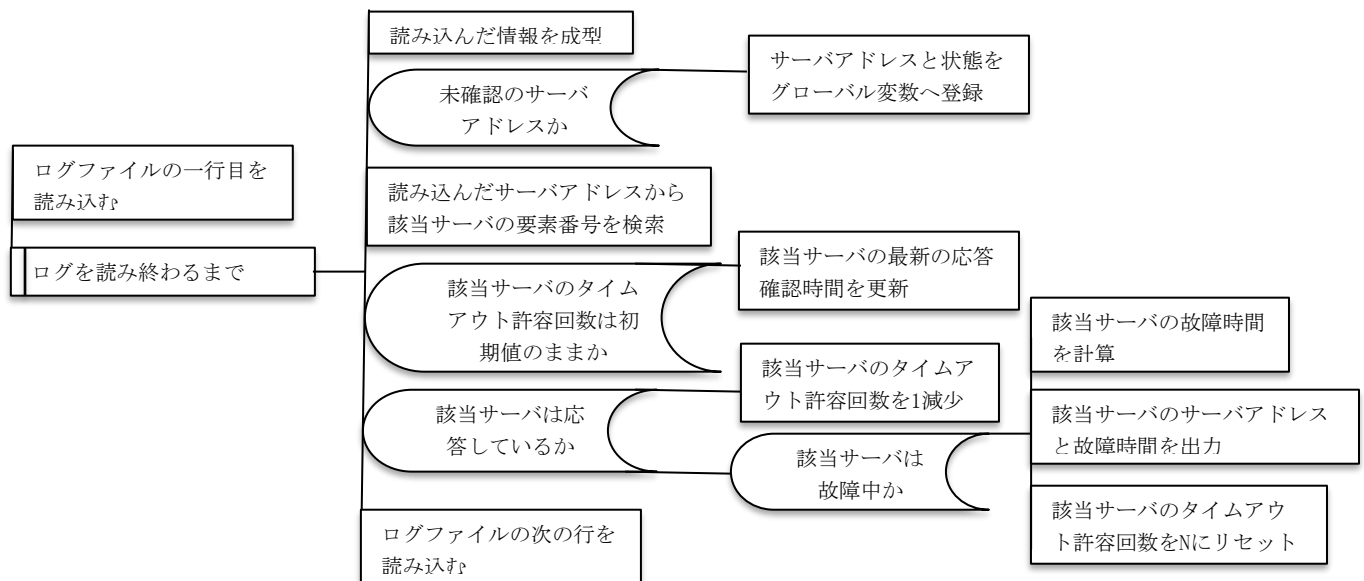


図2 “problem2.py” のPAD図

2.3. 設問3のプログラム “problem3.py” の仕様

2.3.1. プログラムの概要

このプログラムは，設問2のプログラムをもとに作成した，サーバの故障状態と過負荷状態をコマンドプロンプトに一覧として出力するプログラムである。

出力形式以下のとおりである。

- 故障状態 : 「failure server : <サーバアドレス>,<故障期間>」.
- 過負荷状態 : 「overload condition : <サーバアドレス>,<過負荷状態確認開始時間>~<過負荷状態確認終了時間>」

2.3.2. データ構造

① 定数宣言

N : int 型定数, サーバのタイムアウト許容回数.
m : int 型定数, サーバの応答時間の記録個数.
t : int 型定数, サーバが過負荷状態を判断する閾値.

② グローバル変数

前の変数に加えて, 4 つの変数を加えた計 7 つの変数にてサーバの情報を管理する. サーバの管理は確認順に要素番号をつけていくことで識別を行う.

servrList	: string 型リスト. サーバアドレスの一覧
serverConnectLastTime	: string 型リスト. 応答を確認した最新の確認時間
failureStatus	: int 型リスト. タイムアウト許容回数(初期値 N)
responseBuffer	: int 型リスト. サーバの最新の m 回の応答確認時間 (初期値-1, 格納時「確認日時+応答時間」)
oldestBuffer	: int 型リスト. responseBuffer 内の 最古のバッファ内の要素番号
overloadStatus	: int 型リスト. サーバが過負荷状態か (0:正常, 1:過負荷)
overloadTime	: string 型リスト. サーバが過負荷状態となり始めた最新の時間

2.3.3. アルゴリズム

図3に “problem3.py” のPAD図を記載する.

プログラムの内容は” problem2.py” へ追加処理を加えたものへなるため追加部分を記載する. 追加処理として, サーバアドレスとその状態の登録の際に追加した4つのグローバル変数への登録も行う.

サーバの故障状態を確認する一連の処理が終了した後に過負荷状態化を確認する処理を行う. まず, 応答結果を確認し, 正常であれば該当サーバの応答確認時間の記録を行う. 記録にはリングバッファを用い, バッファ内のもっとも古い確認応答時間を更新する. 次に, 該当サーバの応答確認時間がm個あることを確認したうえで平均応答時間を求める. 平均応答時間が過負荷状態の閾値tを超えていた場合, 該当サーバが過負荷状態であるかを確認する. もしそうでなかった場合は過負荷状態が開始したとみなし, その時の確認日時を該当サーバのoverloadTimeへの代入, overloadStatusを過負荷状態への更新を行う.

もし, 平均応答時間がtを下回っていた場合, 該当サーバが過負荷状態であったかを確認する. 過負荷状態であった場合は「該当サーバのサーバアドレス, overloadTimeに格納されている最新の過負荷状態開始時間, 現在の<確認日時>」を出力したうえで, overloadStatusを正常へ更新する.

これをログファイルすべて読み込むまで繰り返したあと, overloadStatusが過負荷状態のままであるサーバがあるかを調べる. 存在した場合は, 「該当サーバのサーバアドレス, overloadTimeに格納されている最新の過負荷状態開始時間, serverConnectLastTimeの最後に応答を確認できた時間」の出力を行う.

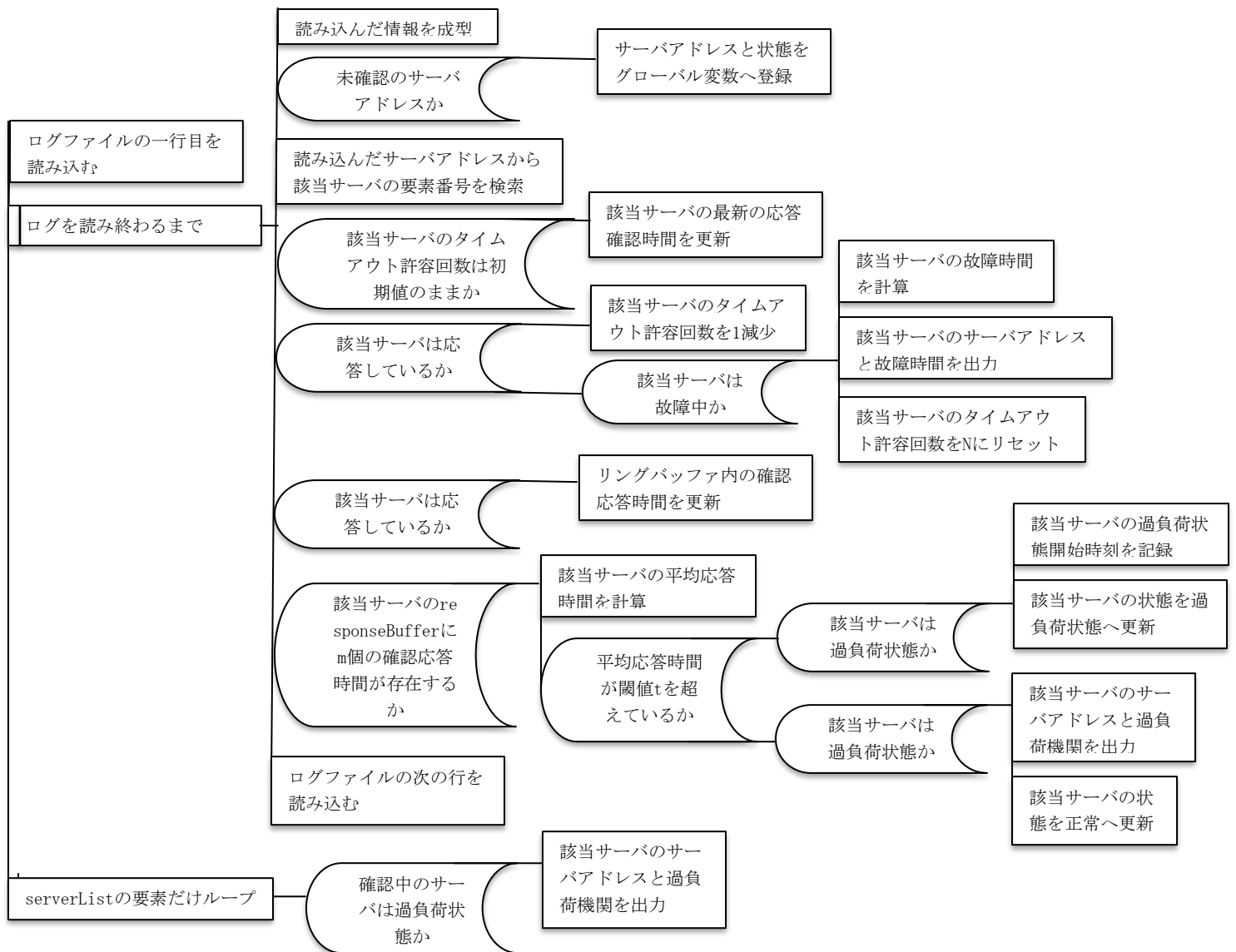


図3 “problem3.py” のPAD図

2.4. 設問4のプログラム“problem4.py”の仕様

2.4.1. プログラムの概要

このプログラムは、設問2のプログラムをもとに作成した、サーバの故障状態とサブネットごとのネットワークの故障期間をコマンドプロンプトに一覧として出力するプログラムである。

出力形式以下のとおりである。

- サーバの故障状態 : 「failure server : <サーバアドレス>, <故障期間>」.
- サブネットの故障状態 : 「failure subnet : <サブネットアドレス>, <故障期間>」

2.4.2. データ構造

① 定数宣言

N : int 型定数, サーバのタイムアウト許容回数.

② グローバル変数

サーバの情報を管理する 3 つの変数に加え, サブネットの情報を管理する 3 つの変数を追加した. サーバとサブネットの管理は確認順にそれぞれで要素番号をつけていくことで識別を行う.

serverList : string 型リスト. サーバアドレスの一覧
serverConnectLastTime : string 型リスト. 応答を確認した最新の時間
failureStatus : int 型リスト. タイムアウト許容回数(初期値 N)

2.4.3. アルゴリズム

図4に “problem4.py” のPAD図を記載する.

まず, ログファイルを一行読み込み, カンマで分割することで変数リストlineに<確認日時>, <サーバアドレス>, <応答時間>の情報をリストとして格納する. また, <サーバアドレス>をもとに該当サーバの存在するサブネットスイッチのネットワークアドレス(以下, サブネットアドレス)も計算する. サーバアドレスとサブネットアドレスが判明したら, 該当サーバおよび該当サブネットが未確認の際はそれぞれの情報を該当するグローバル変数へ登録する.

次に, サーバアドレスとサブネットアドレスをもとに該当するサーバおよびサブネットを格納している要素番号をそれぞれ探し出す. ここで該当サーバおよび該当サブネットの状態が稼働中であれば確認時間の更新を行う.

その後, 応答時間を確認する. 応答がなければ該当サーバのタイムアウト許容回数を1減少させる. 応答があれば, 該当サーバの状態を確認する. この時に故障中であれば, serverConnectLastTimeの時間と読み込んだ<応答時間>から故障期間を計算し, 「該当サーバアドレスと故障期間」を出力する. 出力後, 該当サーバのタイムアウト許容回数をNにリセットする. 同様の処理を該当サブネットに対しても行い, 必要に応じて, 該当サブネットのタイムアウト許容回数更新とサブネットの故障状態の出力を行う.

これをログファイルすべて読み込むまで繰り返す.

また, この処理を行う中で以下の関数を定義した.

① 関数定義

名称 : subnetNetworkCalc
機能 : 引数 address に対応するネットワークアドレスを計算.
引数 address : string 型の CIDR 表記で示されたサーバアドレス.
返り値 : 引数 address に対応するネットワークアドレス

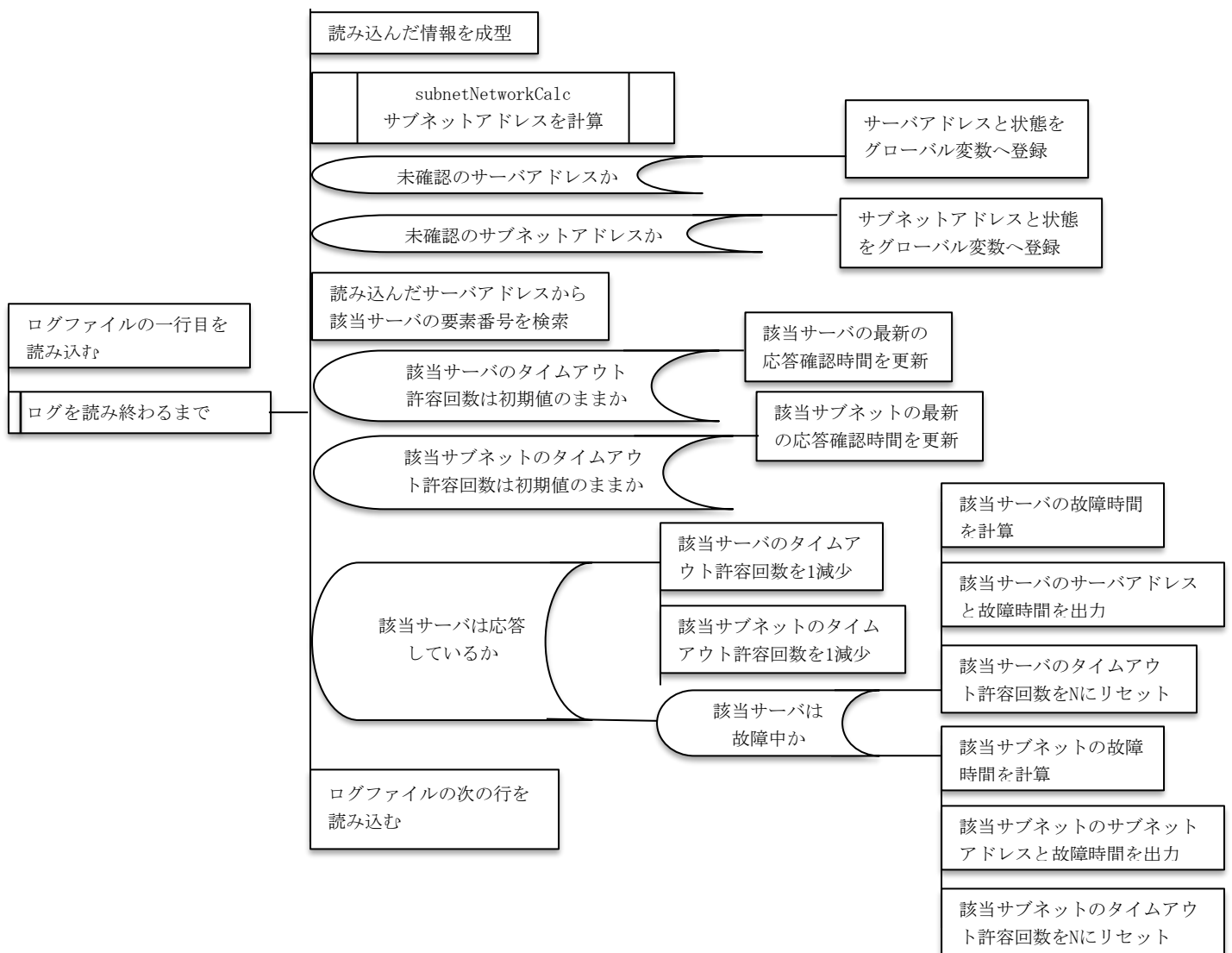


図4 “problem4.py” のPAD図

3. 実行環境

作成したプログラムは以下の環境にて実行した。

OS : windows11 home
使用言語 : python 3.10.3

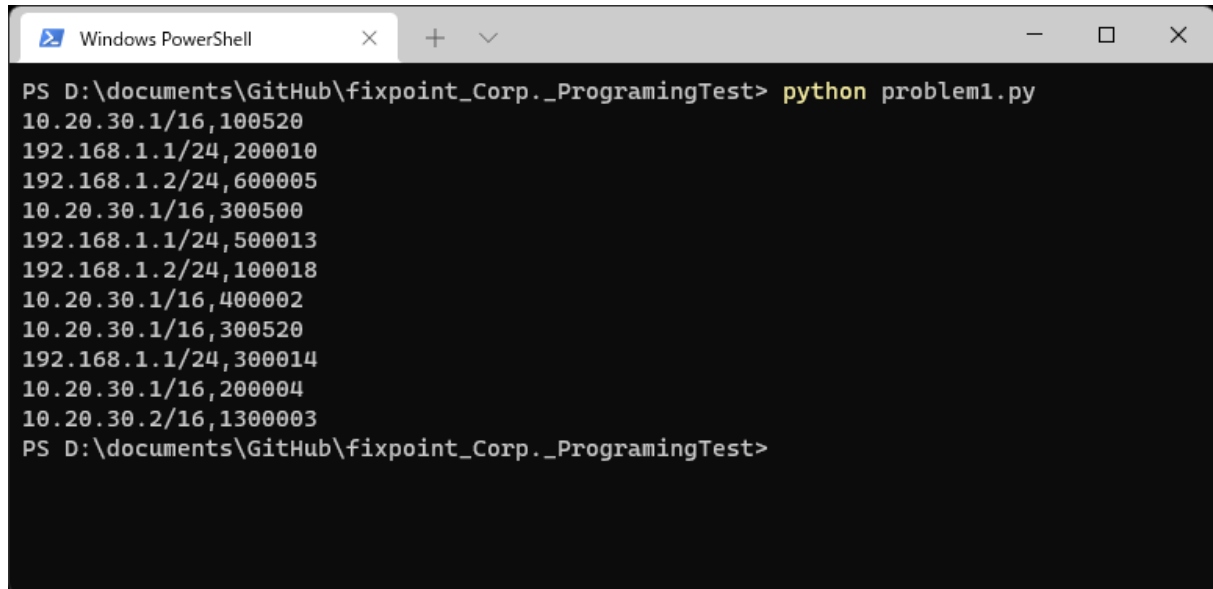
4. 実行結果

以下に設問ごとに作成したプログラムとその実行結果を記載する。

(設問 1)

使用プログラム：“problem1.py”

入力データ：同フォルダ内の“logfile.txt”



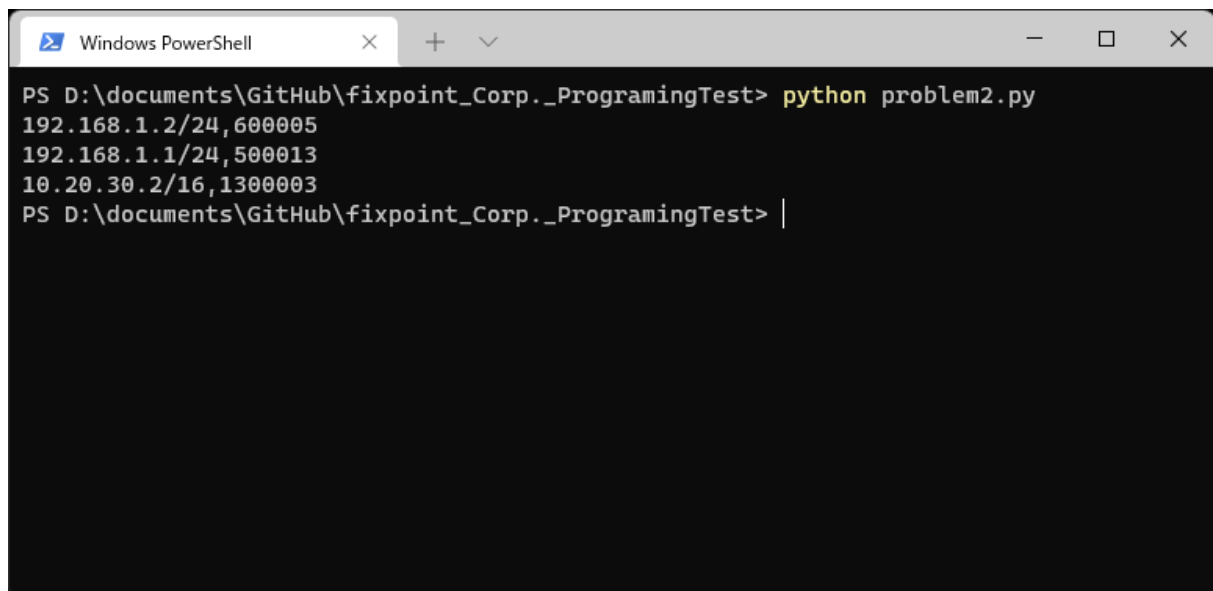
```
Windows PowerShell
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest> python problem1.py
10.20.30.1/16,100520
192.168.1.1/24,200010
192.168.1.2/24,600005
10.20.30.1/16,300500
192.168.1.1/24,500013
192.168.1.2/24,100018
10.20.30.1/16,400002
10.20.30.1/16,300520
192.168.1.1/24,300014
10.20.30.1/16,200004
10.20.30.2/16,1300003
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest>
```

(設問 2)

使用プログラム：“problem2.py”

入力データ：同フォルダ内の“logfile.txt”

パラメータ：N=5



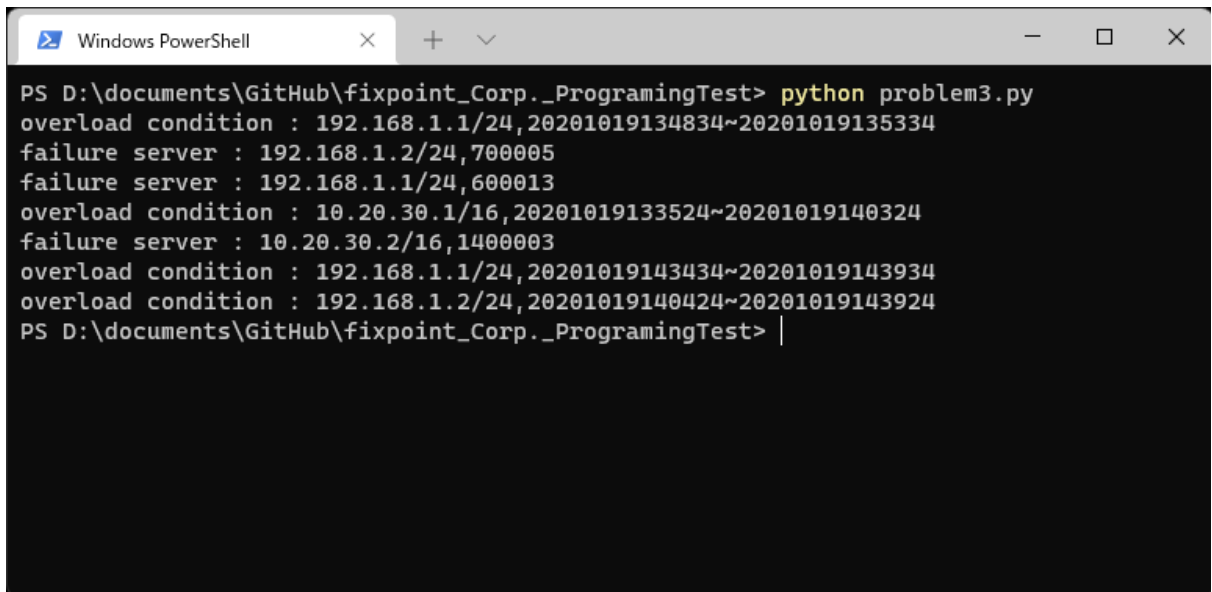
```
Windows PowerShell
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest> python problem2.py
192.168.1.2/24,600005
192.168.1.1/24,500013
10.20.30.2/16,1300003
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest> |
```


(設問 3)

使用プログラム：“problem2.py”

入力データ：同フォルダ内の“logfile.txt”

パラメータ：N=5, m=5, t=100



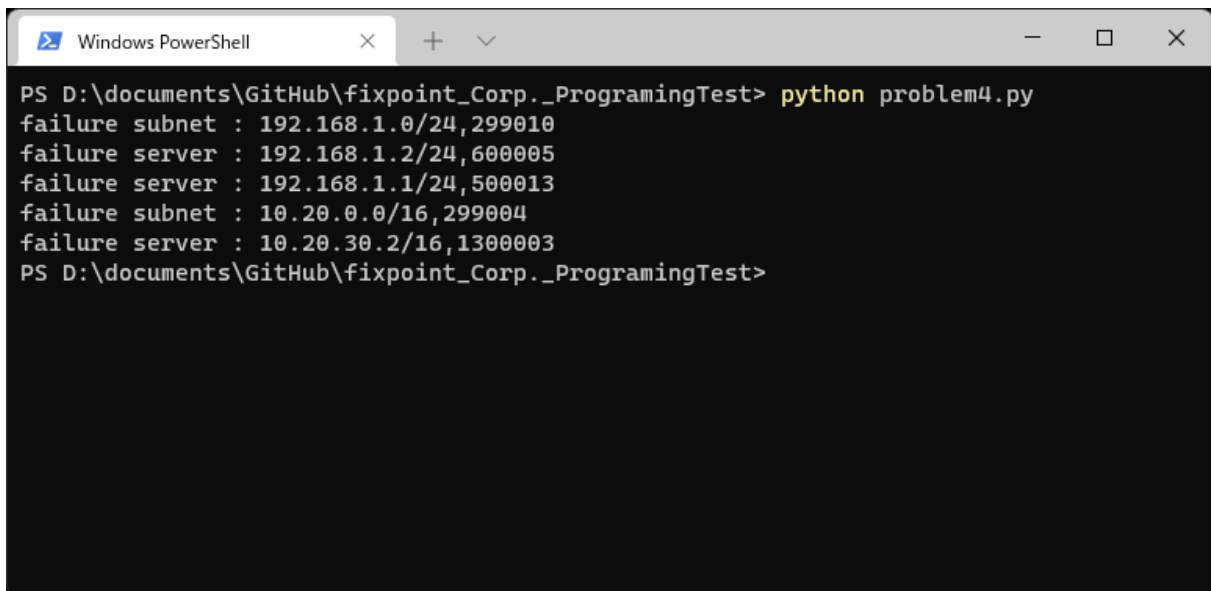
```
Windows PowerShell
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest> python problem3.py
overload condition : 192.168.1.1/24,20201019134834~20201019135334
failure server : 192.168.1.2/24,700005
failure server : 192.168.1.1/24,600013
overload condition : 10.20.30.1/16,20201019133524~20201019140324
failure server : 10.20.30.2/16,1400003
overload condition : 192.168.1.1/24,20201019143434~20201019143934
overload condition : 192.168.1.2/24,20201019140424~20201019143924
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest> |
```

(設問 4)

使用プログラム：“problem2.py”

入力データ：同フォルダ内の“logfile.txt”

パラメータ：N=5



```
Windows PowerShell
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest> python problem4.py
failure subnet : 192.168.1.0/24,299010
failure server : 192.168.1.2/24,600005
failure server : 192.168.1.1/24,500013
failure subnet : 10.20.0.0/16,299004
failure server : 10.20.30.2/16,1300003
PS D:\documents\GitHub\fixpoint_Corp._ProgramingTest>
```