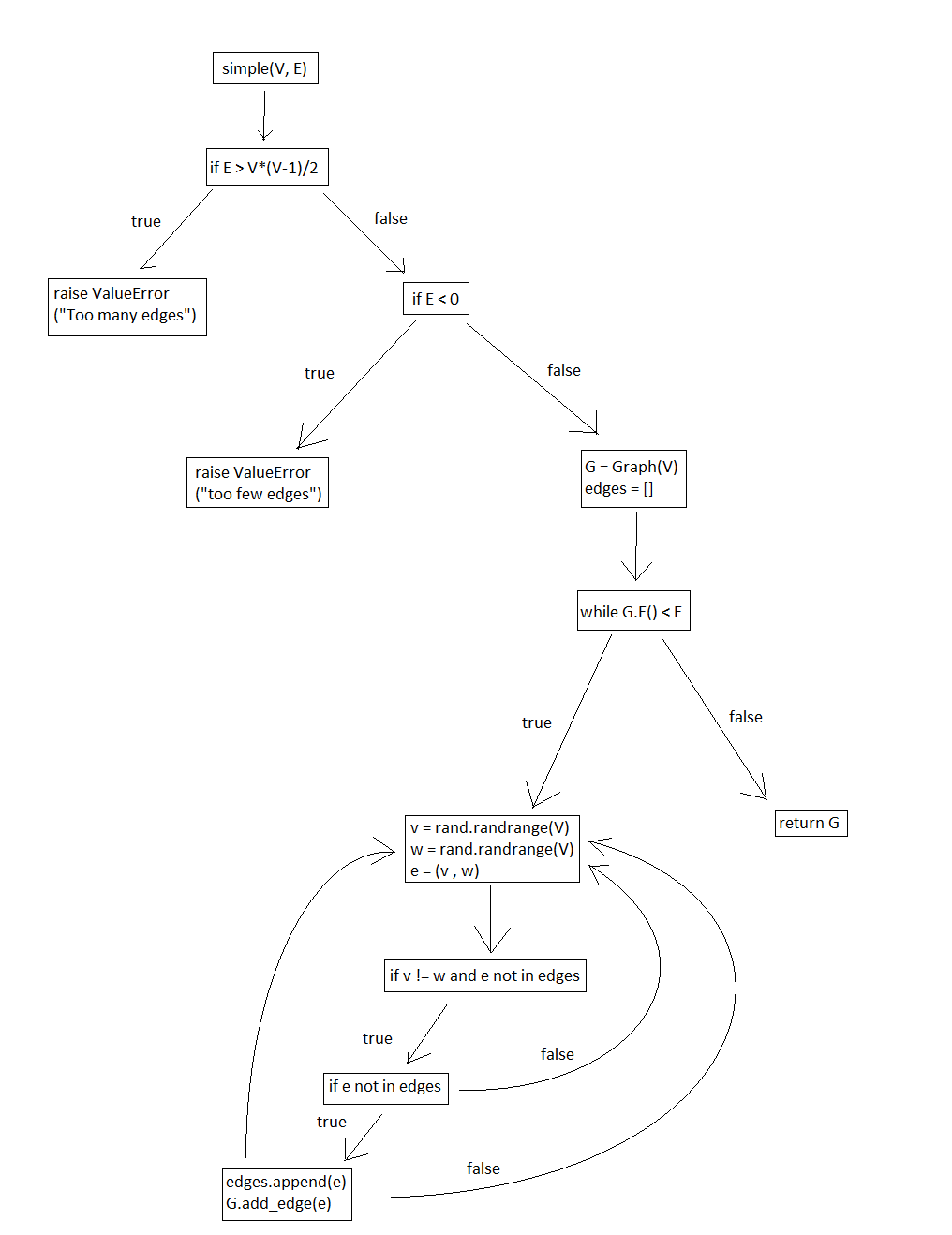
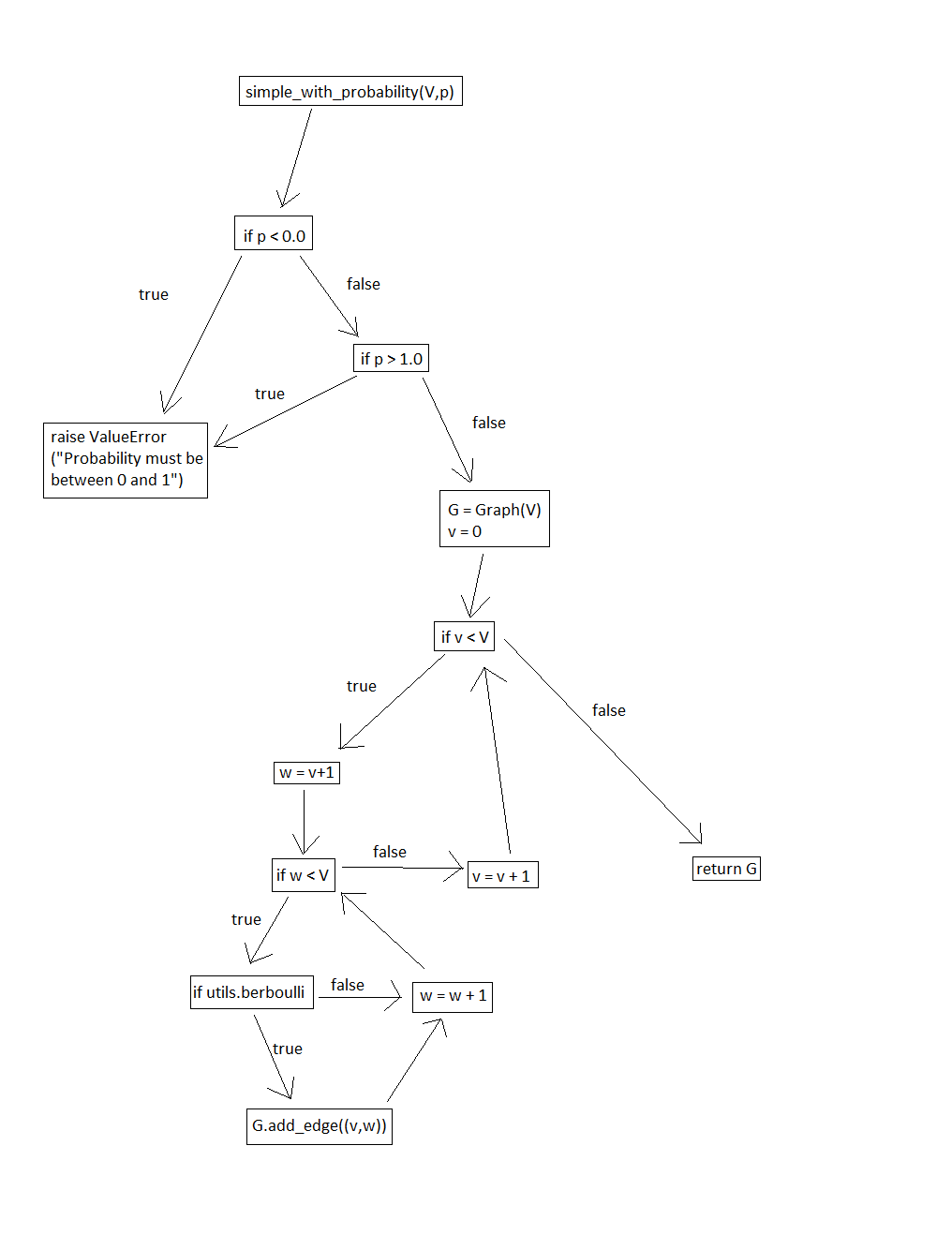
Question 4:

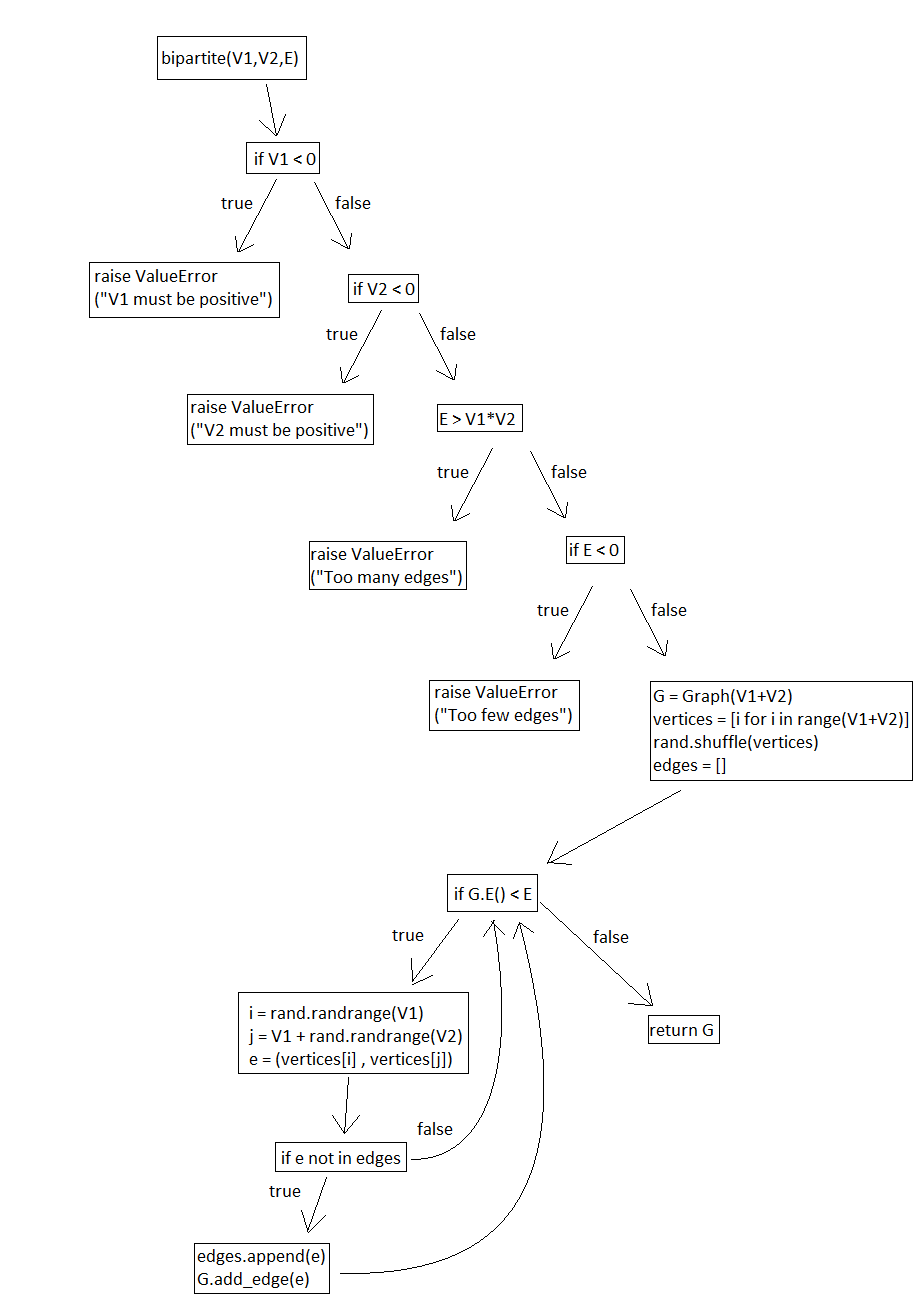
CFG simple :



CFG simple\_with\_probability :



CFG bipartite:



CFG bipartite\_with\_probability:

CFG eulerianCycle:

CFG regular:

Question 5:

Il y avait plusieurs choses à changer dans les fonctions à tester, notamment l’ajout de conditions pour vérifier que le nombre de nœuds et le nombre d’arrêtes mis en arguments ne dépassent pas les bornes permises. Pour les fonctions bipartite et bipartite\_with\_probability, il fallait ajouter des conditions permettant de vérifier que V1 et V2 soient positifs. Pour le cycle eulérien, il fallait borner le nombre d’arrête au maximum possible dans un graphe soit nbNoeuds \* (nbNoeuds-1) / 2. Pour regular, il fallait s’assure que le nombre de nœuds soit supérieur ou égal à 3 et que le nombre d’arrêtes par nœuds soit positif et strictement inférieur au nombre de nœuds.

Il y avait entre autres une erreur dans la conception des fonctions eulerianCycle and regular. En effet, il était possible que parmi les arrêtes générées aléatoirement il y en ait des similaires, ce que le graphe rejette; le copies d’arrêtes se feront rejeter par ce dernier, ce qui au final donnerait des graphes erronés en fonction de la méthode utilisée. Pour résoudre ce problème, il fallait ajouter du code permettant de vérifier que l’ensemble des arrêtes générées aléatoirement ne possède pas de copies. Ainsi, les fonctions eulerianCycle et et regular seront aptes à générer des graphes aléatoires valides.