Projet d’Algorithme de graphe

NGUYEN Binh Minh - L3 MIAGE en alternance

Sujet: Régler le problème de **Directed Feedback Vertex Set**, solution *heuristique*

Langage utilisé: Python

Implémentation du graphe

J'ai choisi d'utiliser une approche **orientée objet**, ce qui signifie que j'ai créé une classe “Graph” pour stocker des informations ainsi que des méthodes liées au graphe. Pour la **structure du graphe**, j’ai utilisé la structure qu’on a vu en TP1 en utilisant la **dictionnaire**:

```
G = {
   "a": ["b"],
   "b": ["a", "c", "d"],
   "c": ["b", "d"],
   "d": ["b", "c", "e"],
   "e": ["d"]
}
```

Voici la structure de ma classe “Graph”:

- Attributes:

    - nbVertexes: int: stocker nombre de sommets du graphe
    - nbEdges: int: stocker nombre de sommets du graphe
    - graph: dictionnary: stocker la structure du graphe. Les sommets sont stockés du type int
    - Time: int: faire partie de l'algorithme de Tarjan pour trouver les CFCs du graphe
    - listSCC: list: liste des CFCs du graphe (chaque CFC est sous forme une liste aussi)
- Méthodes:
    - getVertexes(): int - obtenir le nombre de sommets du graphe
    - getEdges(): int - obtenir le nombre d'arcs du graphe
    - getNbNeighbors(v: int): (int, int) - obtenir le nombre de voisins entrant et sortant du sommet "v"
    - removeVertex(v: int): Graph - obtenir un nouveau graphe avec sommet "v" effacé
    - getHighestDegreeVertexe(): int - obtenir le sommet ayant la plus grande degrée du graphe
    - SCCUtil(…):void - faire partie de l'algorithme de Tarjan pour trouver les CFCs du graphe
    - SCC(dictionary: dictionary):void - faire partie de l'algorithme de Tarjan pour trouver les CFCs du graphe. Je stocke le resultat dans l'attribute "listSCC".

Le code d'algorithme Tarjan que je trouve sur l'Internet marche seulement avec le graphe ayant la structure "ordonné". Cependant, dans quelques cas, après qu'on efface les sommets du graphe par exemple, la structure du graphe n'est plus "ordonné". Donc, j'ai traduit la structure du graphe sous forme "ordonné" avant être appliqué par l'algorithme Tarjan, et ajouté le paramètre "dictionary" qui va stocker les traductions. Grâce à ça, on peut re-traduire les sommets à leurs états initiaux après l'application de l'algorithme.

```
#structure "ordonné"
{
```

```
  1: [2, 3]
  2: [1, 3]
  3: [2]
}
#structure pas "ordonné"
{
  1: [2, 3]
  3: [1, 3]
  4: [2]
}
#dictionary
{
  1: 2 #newName: oldName
  2: 1 #newName: oldName
  3: 4 #newName: oldName
  4: 3 #newName: oldName
}
```

- getListSCC(): list(Graph) - le code obtenir la liste des CFCs mais chaque CFC maintenant est sous forme d'un graphe.

J'ai écrit en plus cette méthode parce que le code d'algorithme Tarjan que je trouve sur l'Internet retourne la liste de CFCs avec chaque CFC sous forme d'une liste.

- runSCC(): void - créer le dictionnaire "dictionnary" de traduction et lancer méthode SCC(dictionnary)

Les algorithmes utilisés

Algorithme principal

Mon **algorithme principal** base sur ce procédure:

```
Data: A Digraph G = (X, U)
Result: A FVS S
begin
    S ⟵ ∅
    LL_graph_reductions(G, S)
    L ⟵ get_SCC(G)
    while |L| ≠ 0 do
        remove g from L
        v ⟵ MFVSmean_selection(g)
        remove v from g
        S ⟵ S + {v}
        LL_reductions(g, S)
        L ⟵ get_SCC(g) + L
    end
    S ⟵ remove_redundant_nodes(G, S)
    return S
end
```

Algorithmes pour sélectionner un sommet à supprimer

J'ai utilisé 2 algorithmes différents pour sélectionner un sommet à supprimer

- **MFVSmean_selection**: choisir le sommet avec le plus petit "mean return time" en appliquant la Chaine de Markov:

```
Algorithm 3: MFVSmean_selection
Data: A Digraph G = (X, U)
Result: A vertex v
begin
    P ⟵ CreateTransitionMatrix(G)
    π' ⟵ ComputeStationaryDistributionVector(P)
    P ⟵ CreateTransitionMatrix(G^-1)
    π'' ⟵ ComputeStationaryDistributionVector(P)
    π ⟵ π' + π''
    determine v ∈ V with π_v = ||π||_∞
    return v
end
```

- Choisir le sommet avec le plus grand nombre de voisins (j'ai créé la méthode **getHighestDegreeVertexe**() dans la classe "Graph" pour ça)

Algorithme Tarjan

J’utilise **algorithme Tarjan** pour trouver les **CFCs** du graphe. Cependant, le code d’**algorithme Tarjan** que je trouve sur l’Internet écrit sous forme de récursion, tandis que Python a une limite de récursion. J’ai dû utiliser la fonction `sys.setrecursionlimit` de Python pour étendre cette limite (Je pense que à cause de ça, il y a de problème “d’utilisation de threads disponibles sur la machine hôte”)

Algorithme secondaire

Pour chaque CFC ayant taille plus que 1, j’enlève simplement ses sommets jusqu’à ce que ce CFC devient de taille 1 (comme ça le graphe initial va devenir sûrement acylique). J’ajoute ces sommets enlevés dans la solution

Solution

Ma solution consiste en **2 grandes étapes**:

Preprocessing

- **Supprimer des sommets et/ou arcs sans perdre d'information**: Supprimer les sommets qui ont degré sortant/degré entrant de 0
    - J'ai profité des boucles dans le processus de lecture du graphe à l’entrée standard pour effectuer le **Preprocessing**. La fonction **handleStdin** donc non seulement me retourne le graphe sous forme d'un objet de classe Graph, mais également me retourne la liste réduite des sommets (reductedList)
    - “reductedList” sert à avoir une solution minimale pour les graphes qui sont trop grands (pour ces grands graphes, après avoir les réduit, je juste simplement retourne tous leurs sommets qui restend comme solution). J’avais voulu aussi l'utiliser pour réduire le nombre de sommets à traiter, mais non seulement cela n'a pas réduit le temps d’exécution mais l’a également augmenté.

Solution initiale

J’ai utilisé les solutions différentes pour les graphes de tailles diffférentes. Je trouve les “zones de division” en recensant jusqu’à quelle taille du graphe chaque algorithme peut traiter, et quel algorithme peut traiter le mieux une certaine taille de graphe

Pour les graphes ayant plus que 10000 sommets ou plus que 170000 arcs

Pour ces graphes, je retourne directement “reductedList” (obtenu dans le processus de lecture du graphe à l’entrée standard) comme solution

Pour les graphes ayant v sommets et m arcs: 8000<v<10000 et 30000<m<170000

Pour ces graphes, j’utilise l’**algorithme secondaire**

Pour les autres graphes

Pour ces graphes, j’utilise l’**algorithme principal avec MFVSmean_selection** comme méthode de sélection du sommet pour la **solution principale.** J’utilise l**’algorithme principal avec getHighestDegreeVertexe()** comme méthode de sélection du sommet comme un “plan d'urgence”, au cas où le premier l’algorithme ne pourrait pas retourner une solution avant 10 minutes, on va retourner immédiatement la solution du “plan d'urgence” (qui n’est pas bien sûr si optimal que la **solution principale**)

Pour les graphes particuliers

Pour certains graphes, je me suis rendu compte qu'il peut être plus optimisé en appliquant les algorithmes différents que ceux appliqués dans leurs “zones de division”. Donc, pour ces graphes particuliers, je leur ai appliqué des algorithmes différents que ce qu'ils auraient dû être, pour avoir des solutions plus optimales

Gestion du temps limité

Je n’ai pas réussi à utiliser le `class Killer` pour gérer le signal `SIGTERM` . J’ai donc calculé en permanence le temps d'exécution du programme en utilisant `time.time()`. Quand ce temps est sur le point d'atteindre 10 minutes, je vais retourner immédiatement la solution du “plan d'urgence”

Références

https://math.nist.gov/mcsd/Seminars/2014/2014-06-10-Shook-presentation.pdf

- Algorithme principal

- MFVSmean_selection

https://kups.ub.uni-koeln.de/2547/1/Dissertation.pdf

- Algorithme principal
- MFVSmean_selection

https://montjoile.medium.com/l0-norm-l1-norm-l2-norm-l-infinity-norm-7a7d18a4f40c

- MFVSmean_selection

https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/

- Algorithme Tarjan

https://ninavergara2.medium.com/calculating-stationary-distribution-in-python-3001d789cd4b

- MFVSmean_selection

https://machinelearningmastery.com/vector-norms-machine-learning/

- MFVSmean_selection

https://medium.com/analytics-vidhya/increase-maximum-recursion-depth-in-python-using-context-manager-1c67eaf4e71b

- Résoudre le problème de récursion en Python