Mini-projet : décongestion d'un réseau

Auteurs: Loïc Cozdenmat et Noam Bendiaf

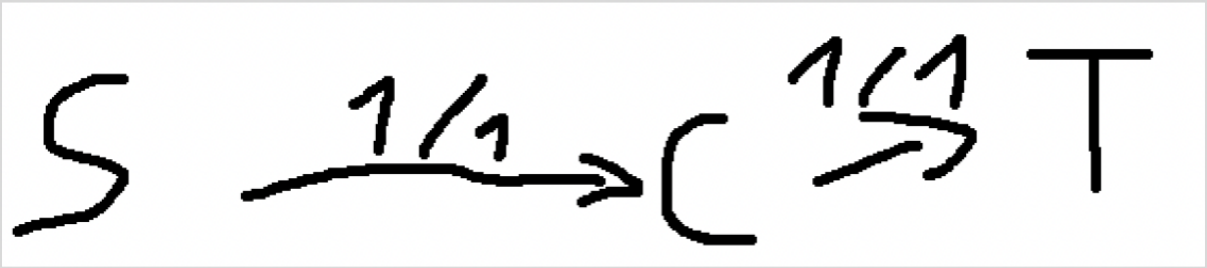
Manque les deux derniers algorithmes

Question théoriques :

1)

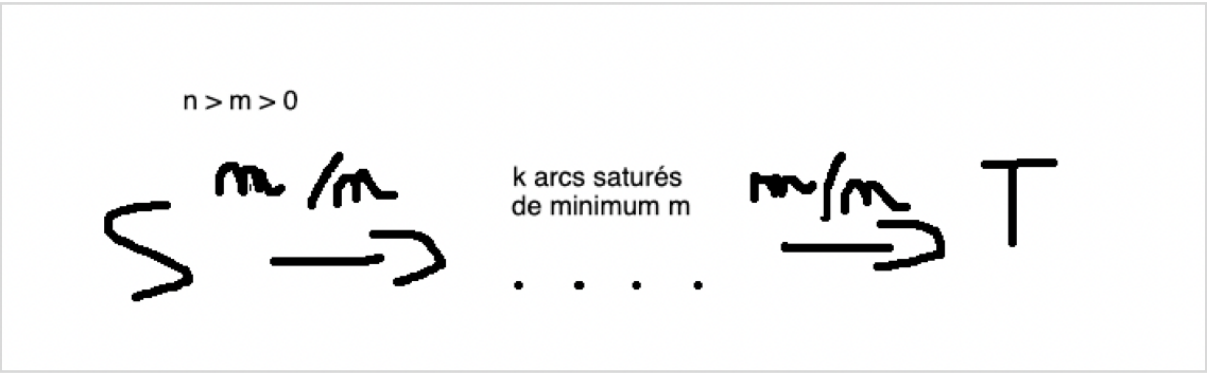
On peut imaginer un réseau où aucune augmentation de capacité d'un seul arc est dite « utile » :

Ici il faudrait augmenter S → B ET C → T de 1



2)

Pour un graphe G tel que S et T sont respectivement une source et un puits relié par des arcs dans un chemin P avec k arcs saturé par un flot maximum : Il faudrait augmenter k capacité d'arcs pour pouvoir augmenter le flot du graphe G



3)

Si le flot est maximal, augmenter la capacité d'un arc non saturé n'augmentera pas le flot tout simplement car si celui ci n'est pas saturé c'est que k autres arcs saturés empêche le

> flot de passer

Presentation des Algorithmes:

J'ai voulu au départ utiliser l'algorithme de Dinitz pour avoir un complexité O(|V|^2 |A|) qui est meilleur dans quasiment tout les cas. J'ai eu quelque problème donc j'ai opté finalement pour celui de Ford Fulkerson.

Pour ensemble_minimum_augmentations_utiles_calibres j'ai utilisé le principe de la programmation génétique qui permet de trouver le meilleur chemins pour une complexité stable et satisfaisante

Voici les algorithmes demandés:

```
FLOT_MAXIMUM(G):
'''
Basée sur l'algoritme de Ford Fulkerson
Complexité O(|A| |V|^2). On utilisera des fonctions annexes permetant d'avoir
un seul puits/source "artificiel"
Paramètres: Graphe G non vide
Résultat : couple (unité de flot, tableau associant les arcs à leurs flots)
'''
  source_artificiel, puits_artificiel <- source/puits unique temporaire attaché
à G
  flot <- tableau associatif (G.arcs(), 0)
  Graphe_f <- copie de G
  chemin <- CHEMIN_AUGMENTANT(G, source_artificel, puits_articiel)
  tant que Chemin existe:
      AUGMENTER_FLOT(flot, chemin)
      MAJ_RESIDUELl(G, Graphe_f, source_artificiel, puits_artificiel)
      chemin <- CHEMIN_AUGMENTANT(Graphe_f, source_artificel, puits_articiel)
// Il faut rétablir les sources et puits originaux
  RETABLIR_GRAPHE(G)
  retourne (somme(flot[(u,v)] si l'arc est orienté vers un puit), flot)

Reseau_residuel(reseau, flot):
'''
Renvoie le residuel du reseau selon le flot maximum: chaque arc (u,v,c)
```

```
reprenste le reste du flot du reseau en parametre.
Paramètres : un graphe reseau et son flot maximum
Résultat: un reseau residuel qui pour chaque u,v represente le  "reste" du flot
'''
  residuel <- Nouveau graphe
  arcs <- reseau.arcs()
  arcs <- arcs U arcs inversé
  MAJ_RESIDUEL(reseau, residuel, arcs, flot)
  retourne residuel

Parcours_largeur_level(residuel, source):
'''
  Parcours le reseau residuel en largeur et par niveau (c'est à dire uniquement
si l'arc s'éloigne du sommet actuel). On utilise cette fonction pour avoir S
dans coupe_minimum.
Compléxité : O(n^2)
Paramètres : réseau residuel et source de départ du graphe
Résultat: les sommets parcouru
'''
  sommets_parcouru <- ensemble vide
  file <- enfiler(source)
  niveau_parcouru <- ensemble contenant source
  tant que file n'est pas vide:
      actuel <- file.defiler()
      si actuel n'est pas dans sommets_parcouru:
          niveau_parcouru <- niveau_parcouru U sommets
      pour sucesseur dans G.voisins(actuel):
          si sucesseur n'est pas dans niveau_parcouru:
              sommets_parcouru.ajout(voisin)
              file.enfiler(voisin)
      pour (u,v) dans residuel.arcs():
          si v est égal à actuel:
              niveau_parcouru.ajout(u)
  retourne sommets_parcouru

Coupe_minimum(residuel, source)
'''
  Donne la plus petite coupe S,T
Paramètres: réseau residuel et une source de départ du graphe
```

```
Complexité
Résultat: couple (S,T) deux groupes de sommets contenant la source et la puits
respectivement
'''
  source_artificiel <- source unique temporaire attaché à residuel
  S <- PARCOURS_LARGEUR_LEVEL(residuel, source)
  T <- ensemble de sommets de residuel/S
  RETABLIR_GRAPHE(residuel)
  retourne (S,T)

Augmentations_unique_utiles(graphe, flot_maximum)
'''
  Renvoie les arcs dont la capacité doit être augmenté pour augmenté le flot
maximal
Complexité(N * M) avec N taille de S et M taille de T
Paramètres: un graphe et un de ses flots_maximum
Résultat: ensemble des arcs à améliorer
'''
  a_augmenter <- ensemble vide
  residuel <- RESEAU_RESIDUEL(graphe, flot_maximum)
  S,T <- COUPE_MINIMUM(residuel, source)
  pour s dans S:
      pour t dans T:
          si graphe.contient_arc(s,t):
              a_augmenter.ajout((s,t))
  retourne a_augmenter

Augmentation_uniques_utiles_calibrees(graphe, flot_max):
'''
  Renvoie les arcs dont augmenter la capacité est utile et de combien
O(N^2)
Paramètres: un graphe et un de ses flots_maximum
Résultat: dictionnaire (arcs -> capacité à augmenter)
'''
  ancienne_valeur_flot, _ <- FLOT_MAXIMUM(graphe)
  nouveau_graphe <- copie de graphe
  source_artificiel, puits_artificiel <- source/puits unique temporaire attaché
à nouveau_graphe
  a_augmenter <- AUGMENTATOIN_UNIQUE_UTILES(nouveau_graphe, flot_max)
```

```
  resultat <- tableau associatif vide
  pour S,T dans a_augmenter:
      flot_actuel <- copie de flot_max
      Graphe_f <- copie de graphe
      residuel <- RESEAU_RESIDUEL(nouveau_graphe, flot_actuel)     calibre <- 0
      chemin_depuis_source, chemin_a_puits <-
CHEMIN_AUGMENTANT(new_graphe,fake_source, S),
CHEMIN_AUGMENTANT(new_graphe,fake_source, S)
      tant que chemin_depuis_source n'est pas vide:
          pour (u,v,c) dans chemin_depuis_source.arcs():
              si residuel.contient_arc(u,v):
                  calibre augemnte de c
      tant que chemin_a_puits n'est pas vide:
          pour (u,v,c) dans chemin_a_puits.arcs():
              si residuel.contient_arc(u,v):
                  calibre augemnte de c
      nouveau_graphe.ajouter_arc(S,T, ancien poids + calibre)
      nouvelle_valeur_flot, _ <- FLOT_MAXIMUM(graphe)
      // On simule un réseau dont l'arc à été augmenté de calibre, si le gain
est supérieur à 0, on ajoute la nouvelle capacité dans le dictionnaire
      gain = nouvelle_valeur_flot - ancienne_valeur_flot
      si gain est supérieur à 0:
          resultat[(S,T)] <- gain
      sinon:
          si le flot sur l'arc n'est pas saturé:
              resultat[(S,T)] <- new_graphe.poids_arc(S,T) - flot[(S,T)]
          sinon :
              // On augmente de 1 pour débloquer
              resultat[(S,T)] <- 1
      nouveau_graphe <- copie de graphe
  retourne resultat


Ensemble_minimum_augmentations_utiles_calibrees(graphe, flot_max, valeur):
'''
  Renvoie le chemin coutant le moins cher pour augmenter le flot + valeur.
  On prend un peuple de 128 chemins dont on gardera les 15 meilleurs, 60
mutations de ces derniers, et 40 chemins aléatoires 1000 fois. (les nombres ont
été choisis déliberement après de multiples tests, il se peut que le chemin
```

```
soit différents pour une valeur élévé mais le cout max reste le même
Complexité : O(N^2 * (1000 * (60 + 40 + 15))
Paramètres : graphe et son flot maximum, la valeur à augmenter de flot_maximum
Résultat: le chemin le moins couteux de la dernière génération
'''
  flot_value, _ <- FLOT_MAXIMUM(graphe)
  but <- flot_value + value
  chemins <- tableau de taille 128 (GENERATION_CHEMIN(graphe, but)
  chemins.trier(cout)
  repeter 1000 fois:
      pour i entre 10 et 70:
          chemins[i] <- MUTATIONS(graphe, chemins[i modulo 10], but)
      pour entre 70 et 128:
          chemins[i] <- GENERATION_CHEMIN(graphe,but)
      chemins.trier(cout)
  retourne chemin[0]
```

Repartitions des taches :

Loïc Cozdenmat : congestion.py, rapport

Noam Bendiaf : Graphe.py

#L3/AG