LANGAGES DE PROGRAMMATION
IFT-3000
Hiver 2024

Travail pratique 2 - TRAVAIL INDIVIDUEL OU EN ÉQUIPE
Pondération de la note finale : 15%

À remettre, par le biais du site Web du cours, et avant 23h59, le vendredi 26 avril 2024

1 Énoncé

Ce travail pratique consiste à implanter différentes fonctions pour le calcul de graphes de dépendances entre cours ; il constitue une suite logique du travail pratique 1. Plus précisément, dans ce travail pratique, on s'intéresse aux 3 scénarios suivants :

1. Graphe de dépendances d'un cours donné indépendemment de tout programme.
2. Graphe de dépendances d'un cours donné restreint aux cours d'un programme donné.
3. Graphe de dépendances de tous les cours d'un programme donné.

Pour un cours donné, le graphe de dépendances comprend :
- toutes les chaines de préalables de ce cours ;
- toutes les chaines de cours qui dépendent de ce cours, c'est-à-dire que dans leurs chaines de préalables, on retrouve ce cours.

Pour un programme donné, on va retrouver toutes les dépendances entre les cours qui le composent.

Remarques Dans ce travail pratique :
- on ne tient pas compte des structures «ET» et «OU» qui regroupent les cours dans les préalables ; ainsi, si on considère les préalables du cours IFT-3001, soit :

  (IFT-2008 OU GLO-2100) ET (MQT-1102 OU STT-1000 OU STT-2000 OU STT-2920) ET (MAT-1310 OU MAT-1919)

  on ne retiendra que le fait que les cours dont dépend IFT-3001 sont : IFT-2008 ; GLO-2100 ; MQT-1102 ; STT-1000 ; STT-2000 ; STT-2920 ; MAT-1310 et MAT-1919 ; on néglige ainsi que certains cours étaient regoupés dans un «OU» et d'autres regroupés dans un «ET». C'est une perte d'information non négligeable mais ça facilite grandement les traitements ; on pourrait penser à une extension de ce travail pratique qui consisterait à garder, tout au long des différents traitements, ces structures de conjonctions et disjonctions liant les cours.
- on suppose qu'on n'a pas de cours qui admettent dans leurs chaines de préalables leur numéro de cours (autrement dit, on n'admet pas de définition cyclique de cours) ;
- on suppose que des cours équivalents ont les mêmes préalables.

Par ailleurs, notons qu'on prendra tout de même soin de regouper les cours équivalents ; par conséquent, pour IFT-3001, on aurait la liste des cours préalables suivants : [IFT-2008 ; GLO-2100] ; MQT-1102 ; STT-1000 ; STT-2000 ; STT-2920 ; MAT-1310 et MAT-1919 ; pour ce cours, seuls IFT-2008 et GLO-2100 ont été regroupés puisqu'équivalents.

Pour réaliser ce travail, vous disposez d'un fichier compressé «tp2.zip» (et non d'un dépôt Github) qui comprend pratiquement les mêmes éléments que ceux du Tp1 ; voici ce qui le distingue du Tp1 :

1. Le fichier README.html résume les étapes à suivre pour réaliser le Tp ;
2. Le dossier «lib» comprend les fichiers du Tp : seul le fichier (**tp2.ml**) est à remmettre, une fois complété ;

3. Le dossier «gcp/lib» comprend désormais, en plus, le Tp1 de l'hiver 2024 ;
4. Le fichier «docs/index.html» permet d'accéder à la documentation générée automatiquement ;
5. Un fichier «c.sh» comprend des commandes «Unix/Linux» permettant de copier le fichier produit par «js_of_ocaml» dans un répertoire accédé par l'application Web (voir section 3).

Au sujet du bassin de cours et des programmes d'études L'information précisée dans l'énoncé du Tp1 est toujours valable.

2 Fonctions à compléter

La signature du TP2 comprend les éléments suivants (en bleu, les 5 fonctions à compléter) :

```
type id_cours = string
 and element_graphe = Noeud of id_cours * num_cours list | Arete of id_cours * id_cours

val graphe_pre_cours: cours list -> num_cours list * num_cours -> element_graphe list
val graphe_post_cours: cours list -> num_cours list * num_cours -> element_graphe list
val graphe_cours: cours list -> num_cours list * num_cours -> element_graphe list

val ret_lnc_pre: cours list -> num_cours list * num_cours -> num_cours list
val ret_lnc_post: cours list -> num_cours list * num_cours -> num_cours list

val graphe_pre_post_cours: (cours list -> num_cours list * num_cours -> num_cours list) ->
                           (num_cours * num_cours -> element_graphe) ->
                           cours list -> num_cours list * num_cours ->
                           element_graphe list

val graphe_pgm: cours list -> programme -> element_graphe list
val ordonner_id_cours_par_session: element_graphe list -> string list list
```

Quelques remarques au sujet de ces éléments (en plus de l'information précisée dans «tp2.mli») :

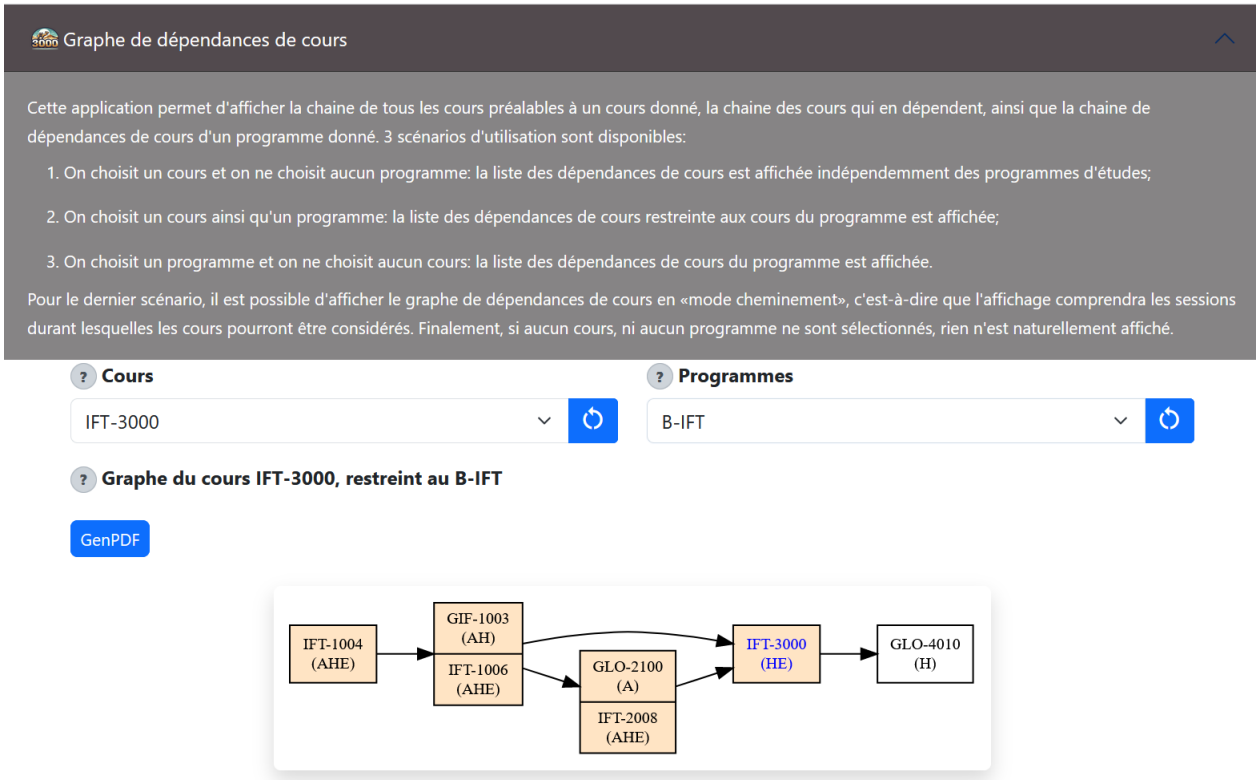
- les principales fonctions manipulent des graphes orientés liant des identifiants de cours entre eux ; un noeud de cet arbre correspond à une paire composée d'un identifiant du noeud et d'une liste de numéros de cours [1] (équivalents et ordonnés selon la fonction du Tp1 «regroupe_cours_equiv») ; l'identifiant correspond à celui du premier cours de la liste ; plus précisément, il s'agit du numéro du cours sans le tiret qui sépare le sigle du numéro de cours (voir fonctions «id» et «idl» dans «gcp») ;
- en théorie, seules les fonctions «graphe_cours», «graphe_pgm» et «ordonner_id_cours_par_session» devraient être exportées (sont utiles pour un client du module Tp2) ; cependant, comme les autres fonctions font partie de celles que vous devez compléter, elles figurent naturellement dans cette signature afin de pouvoir les tester ;
  - plus précisément, une des principales fonctions que vous devez compléter est la fonction auxiliaire «graphe_pre _post_cours» qui est appelée par les fonctions fournies «graphe_pre_cours» et «graphe _post_cours» ; cette fonction permet de calculer la liste des cours qui se situent à la gauche d'un cours donné dans le graphe (il s'agit donc des cours qu'on retrouvent dans ses chaines de préalables), ou ceux qui se trouvent à sa droite (soit les cours qui en dépendent, identifiés par cours «post» pour «postrequis» par analogie à «prerequis») ; la fonction fournit la partie du graphe «pre» ou la partie «post» dépendemment de son 1^{er} argument («ret_lnc» : voir code source) qui est une fonction qui retourne, pour un numéro de cours donné, soit la chaine des cours préalables, soit la chaine des cours qui en dépendent [2] ; nous vous fournissons 2 fonctions «graphe_pre_cours» et «graphe_post_cours» car elles font justement appel à «graphe_pre_post_cours» pour respectivement retourner la partie «pre» et la partie «post» de l'arbre d'un cours donné (vous avez ainsi 2 exemples concrets de l'utilisation de la fonction «graphe_pre_post_cours») ; d'ailleurs, dans le corrigé (voir testeur), c'est via ces 2 fonctions que la fonction «graphe_pre_post_cours» est testée ;
- les fonctions à compléter «graphe_pgm» et «ordonner_id_cours_par_session» sont bien expliquées dans «tp2.mli» ; à noter que dans le corrigé, la fonction «graphe_pgm» fait appel à «graphe_post_cours» puisqu'il s'agit, pour la construction d'un graphe de dépendances des cours d'un programme, de considérer

1. Rappel : peut-être une liste singleton pour les cours qui n'admettent de cours équivalents.
2. Le 2^e argument de la fonction «graphe_pre _post_cours» permettant d'orienter correctement les arêtes.

ses cours de première session (sans préalables[3]), et de déterminer la liste des cours qui en dépendent (l'opérateur «++» pouvant «absorber» les mêmes éléments de graphe produits à plus d'une reprise) ;
— en tout, 4 fonctions sont fournies dans le fichier «tp2.ml» : «graphe_pre_cours», «graphe_post_ cours» et «graphe_cours», mais aussi une petite fonction utile, «pas_de_prealables», utilisée dans le corrigé, et que vous pouvez évidemment utiliser.

3 Application Web

Une application Web est fournie afin de visualiser les résultats des fonctions du Tp2 que vous devez compléter :



Pour votre information, cette application utilise la librarie Javascript de l'outil Graphviz et comprend :
— un fichier «index.html» qui comprend l'interface de l'application Web (différents composants[4] Web) ;
— un fichier «script.js» qui comprend certaines fonctions Javascript pour l'interaction avec l'utilisateur ;
— un fichier «tp2js.bc.js» produit par «js_of_ocaml» et qui correspond aux fonctions du travail pratique (ainsi que du paquetage Gcp) traduites en Javascript ; ce fichier est naturellement référencé par «index.html» ; «script.js» fait naturellement appel à certaines fonctions qui s'y trouvent.

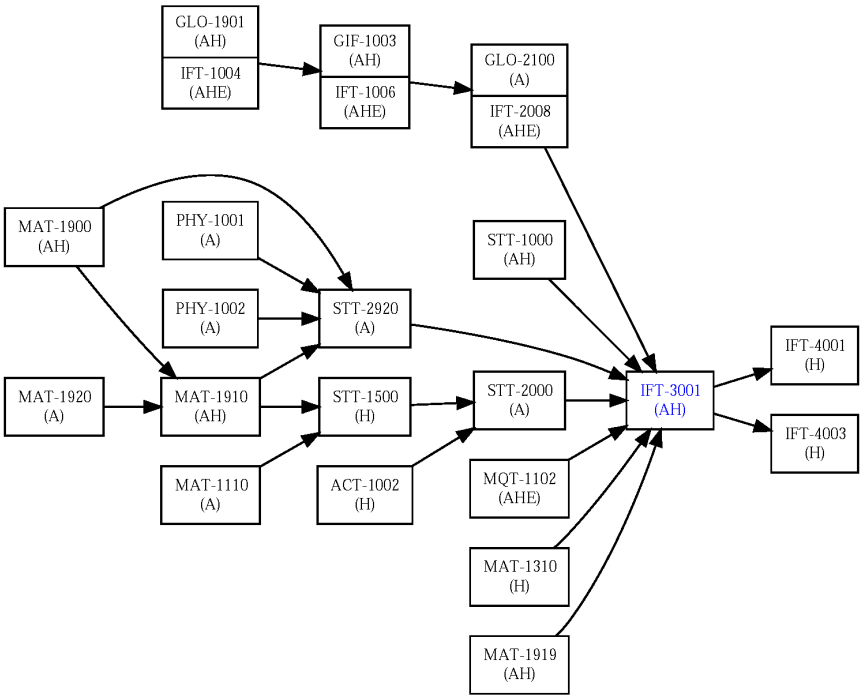
Vous n'avez pas besoin de vous soucier de ces fichiers et codes ; à chaque fois que vous aller faire un «build» de votre travail pratique, «js_of_ocaml» va automatiquement générer le fichier «tp2js.bc.js». Si vous voulez tester les fonctions de votre Tp via l'application Web, vous devez procéder comme suit :
— copier le fichier «tp2js.bc.js» produit par «js_of_ocaml» du répertoire «_build/default/js» vers le répertoire «js/resources/js/gcp» ; le fichier, à la racine, «c.sh»[5] comprend les commandes à exécuter ;
— lancer un serveur Web local (par exemple, via le greffon VSCode «Live Server») et accéder à l'URL suivante via un navigateur : `http://localhost:5500/js`.

3. Dans ce Tp, voir fonction fournie «pas_de_prealables», un cours sans préalables admet soit «Aucun» comme préalable, soit «CRE n» (pour un n entier).
4. Utilisation de Bootstrap.
5. Dans la machine virtuelle, il faut entrer la commande suivante, à partir d'un terminal, afin de pouvoir lancer les commandes incluses dans ce fichier : «chmod +x c.sh». Une fois ce changement effectué, un simple «./c.sh» dans le terminal exécutera les commandes pour vous.

4 Exemples

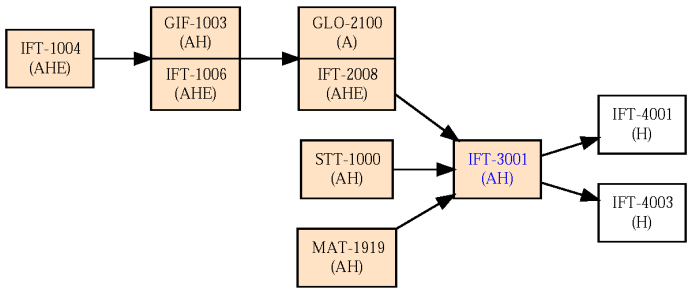
Les fichiers «tp2.mli» et «testeur.ml» comprennent un ensemble d'exemples d'utilisation des fonctions du Tp à compléter. Dans cette section, je fournis quelques exemples visualisés [6] grâce à l'application Web fournie.

Graphe de dépendances du cours IFT-3001, indépendemment des programmes



Le cours pour lequel le graphe a été construit est marqué en bleu. Aucun cours n'est précisé (en couleur) comme étant obligatoire puisqu'aucun programme n'est sélectionné.

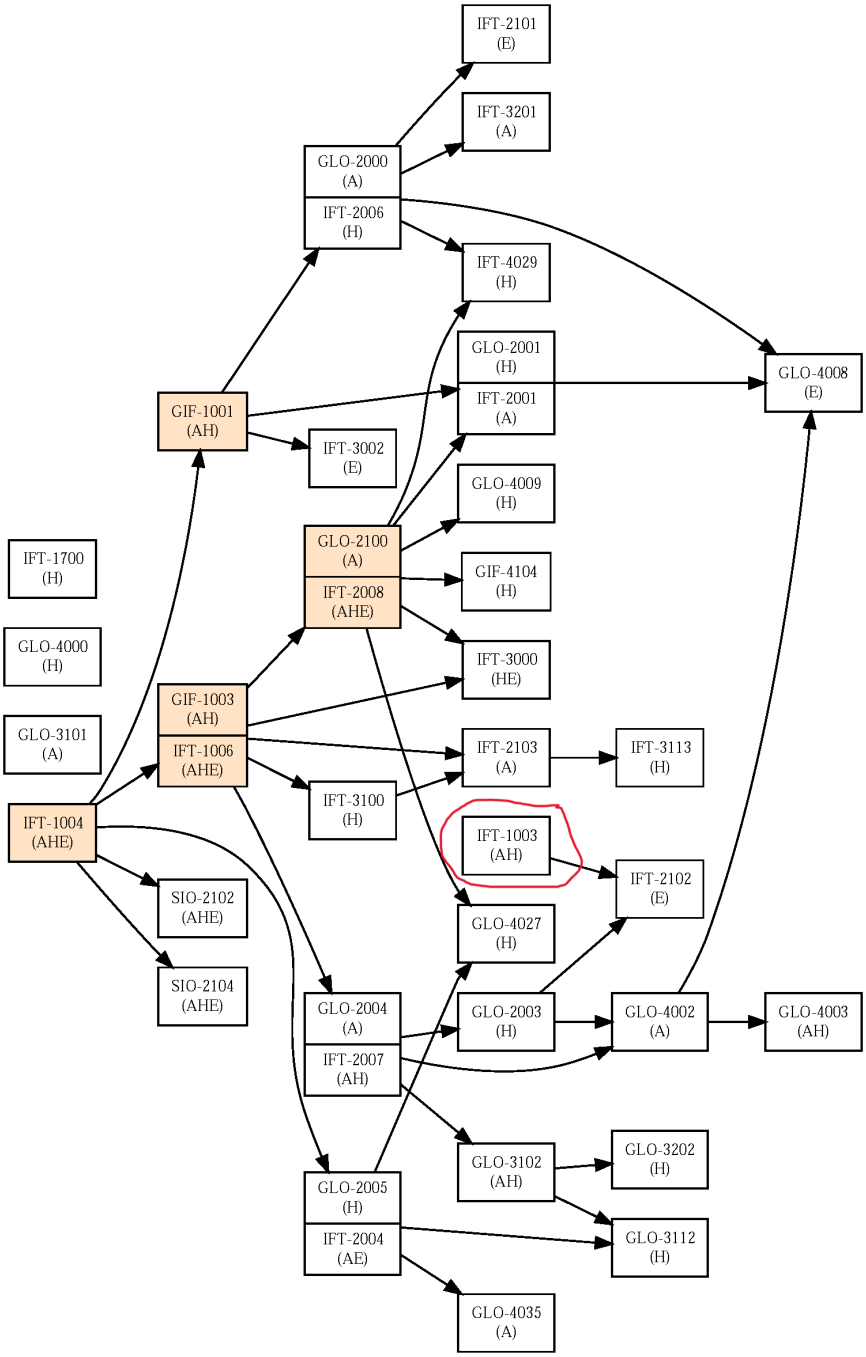
Graphe de dépendances du cours IFT-3001 restreint aux cours du B-IFT



Le cours pour lequel le graphe a été construit est marqué en bleu ; les cours obligatoires dans le programme sont aussi marqués en couleur.

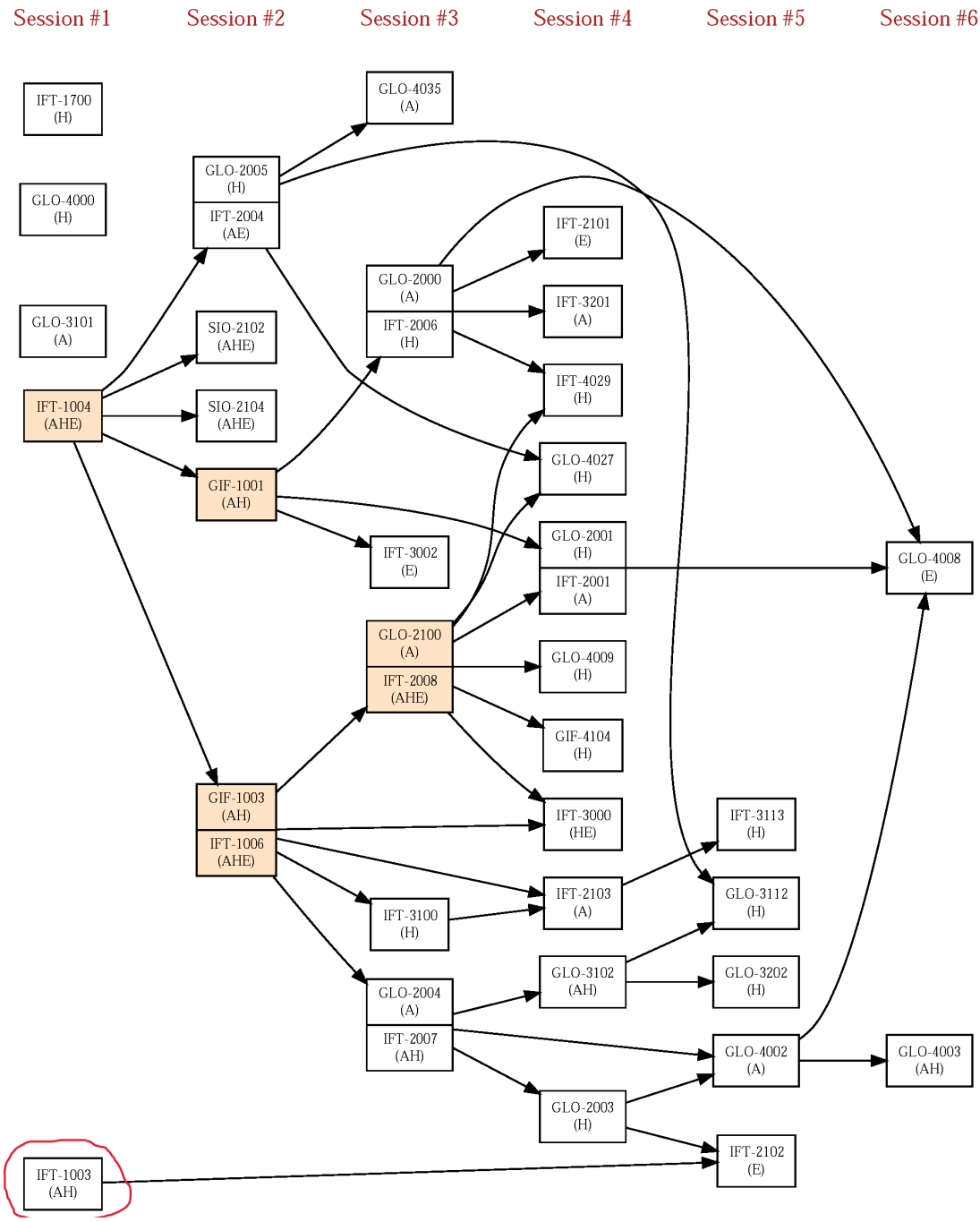
6. En fait, l'application Web offrant la possibilité de générer en PDF les graphes, il est assez simple de les récupérer.

Graphe de dépendances des cours du C-IFT



On remarque que le cours IFT-1003 est seulement prérequis au cours IFT-2102, et n'admet aucun préalable ; contrairement aux cours IFT-1700, GLO-4000, GLO-3101 et IFT-1004, on ne perçoit pas clairement que le cours IFT-1003, n'admettant pas de préalables, devrait aussi se situer au niveau des cours de «première session».

Graphe de dépendances des cours du C-IFT en «mode cheminement»



Grâce au résultat fourni par la fonction ««ordonner_id_cours_par_session», on ajoute des colonnes précisant les sessions durant lesquelles les cours pourraient être considérés, en fonction des préalables ; en particulier, on perçoit mieux que le cours IFT-1003 devrait clairement se situer en première session.

5 Démarche à suivre

À l'instar du TP1, il s'agit essentiellement de compléter une première fonction, puis de la tester grâce au testeur, ou via vos propres tests à l'aide d'un interpréteur ; puis de passer à la prochaine fonction jusqu'à terminer votre Tp.

Au niveau de la correction, nous allons nous assurer que :

1. le contenu du fichier «tp2.ml» que vous allez remettre respecte ce qui est spécifié dans «tp2.mli» ;
2. votre travail passe les différents tests (nous utiliserons d'autres cas de tests) ;
3. vos différentes fonctions sont correctement implantées (revue de code).

Notez qu'on utilisera la version du fichier «tp2.mli» remise avec l'énoncé (de même pour «gcp.mli» et «gcp.ml»).

Remarque : Vous pouvez utiliser toute la matière vue dans le cours (n'importe quel paradigme) pour réaliser ce travail pratique.

6 À remettre

Il faut remettre un fichier .zip contenant uniquement le fichier "tp2.ml" complété. Le code doit être clair et bien structuré[7].

7 Remarques importantes

IA générative : Tel que décrit dans le plan de cours, vous n'êtes pas autorisés à utiliser l'IA générative pour la réalisation de votre travail pratique. Conformément au Règlement disciplinaire à l'intention des étudiants et étudiantes de l'Université Laval, le fait d'obtenir une aide non autorisée pour réaliser une évaluation (travail pratique, en ce qui nous concerne) est considéré comme une infraction relative aux études. Dans le cadre de ce cours, l'utilisation de l'IA générative est considérée comme une aide non autorisée. Cette infraction pourrait mener à l'application des sanctions prévues au Règlement disciplinaire.

Notez que nous nous gardons le droit de prendre au hasard des travaux et de poser des questions pour nous assurer que le contenu du travail en question est bien le fruit du travail de l'étudiant concerné.

Plagiat : Tel que décrit dans le plan de cours, le plagiat est interdit. Une politique stricte de tolérance zéro est appliquée en tout temps et sous toutes circonstances. Tous les cas seront référés à la direction de la Faculté.

Travail remis en retard : Tout travail remis en retard sera refusé et obtiendra la note 0 ; désolé, aucune tolérance car le corrigé sera rendu public, aussitôt la date limite de remise passée. La remise doit se faire par la boite de dépôt du TP2 dans la section «Évaluation et résultats».

Barème : Au niveau du code à implanter, le barème est précisé dans le fichier "tp2.ml". Notez cependant que :

— (-10pts), si vous ne remettez pas un unique fichier nommé "tp2.ml".
— (-10pts), si votre code ne compile pas, ou n'est pas conforme à sa spécification.
— tout code en commentaire ne sera pas corrigé.

Bon travail.

7. Suggestion : http://ocaml.org/learn/tutorials/guidelines.html