DS 2

Option informatique, deuxième année

Julien Reichert

Partie 1 : Graphes (cours)

Exercice 1 : Avec les types ci-dessous définissant de trois façons différentes un graphe, écrire une fonction convertissant une représentation de type `graphe3` d'un graphe en la représentation de type `graphe2` du même graphe (en créant des noms de sommets au choix mais cohérents) et une fonction convertissant une représentation de type `graphe2` d'un graphe en la représentation de type `graphe1` du même graphe (en utilisant les mêmes noms de sommets).

```
type graphe1 = { mutable sommets : string list;
mutable arcs : (string * string) list; };;
type graphe2 = (string * string list) list;;
type graphe3 = bool array array;;
```

Exercice 2 : Avec le type `graphe2` ci-avant et le type `graphep` ci-après, écrire une fonction convertissant une représentation de type `graphe2` d'un graphe orienté en la représentation de type `graphep` du graphe pondéré de mêmes sommets et mêmes arcs, avec un poids nul sur chaque arc. Écrire ensuite la fonction réciproque, qui se contente d'effacer la mention des poids des arcs.

```
type graphep = (string * (string * int) list) list;;
```

Exercice 3 : Déterminer le plus court chemin du sommet `A` au sommet `G` dans le graphe pondéré (non orienté) dessiné en figure 1. L'algorithme utilisé sera au choix parmi Dijkstra, Floyd-Warshall et Bellman-Ford, et il faudra détailler les étapes de calcul.

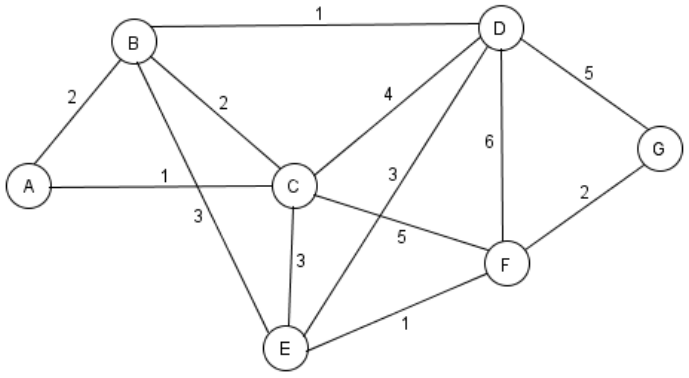


Figure 1 – Graphe pondéré

Partie 2 : Graphes (annales)

Graphe du Web

Le World Wide Web, ou Web, est un ensemble de pages Web (identifiées de manière unique par leurs adresses Web, ou URL pour Uniform Resource Locators) reliées les unes aux autres par des hyperliens. Le Web est souvent modélisé comme un graphe orienté dont les sommets sont les pages Web et les arcs les hyperliens entre pages. Le Web étant potentiellement infini, on s'intéresse à des sous-graphes du Web obtenus en naviguant sur le Web, c'est-à-dire en le parcourant page par page, en suivant les hyperliens d'une manière bien déterminée. Ce parcours du Web pour en collecter des sous-graphes est réalisé de manière automatique par des logiciels autonomes appelés Web crawlers ou crawlers en anglais, ou collecteurs en français.

Fonctions utilitaires

Nous allons tout d'abord coder certaines fonctions de manipulation de structures de données de base, qui seront utiles dans le reste de l'exercice.

Question 2.1 : Coder une fonction `aplatir : ('a * 'a list) list -> 'a list`, telle que, si `liste` est une liste de couples $[(x_1,l_{x_1});\ldots;(x_n,l_{x_n})]$, où chaque x_i est un élément de type `'a`, et l_{x_i} une liste d'éléments de type `'a` et de la forme $[y_{i1};\ldots;y_{ik_i}]$, `aplatir liste` est une liste d'éléments de type `'a` :

$$[x_1;y_{11};\ldots;y_{1k_1};\ldots;x_n;y_{n1};\ldots;y_{nk_n}].$$

Question 2.2 : Coder une fonction `tri_fusion : ('a * 'b) list -> ('a * 'b) list` triant une liste de couples `(x, y)` par ordre **décroissant** de la valeur de la seconde composante `y` de chaque couple. On devra utiliser l'algorithme de tri par partition-fusion (aussi appelé « tri fusion »). Quelle est la complexité de cet algorithme ?

On va utiliser dans la suite de l'exercice un type de données `dictionnaire` qui permet de stocker des couples formés d'une chaîne de caractères (une clef) et d'un entier (une valeur). On dit que le dictionnaire associe la valeur à la clef. À chaque clef présente dans le dictionnaire est associée une seule valeur. Les fonctions suivantes sont supposées être prédéfinies :

- `dictionnaire_vide : unit -> dictionnaire`. L'appel `dictionnaire_vide ()` crée un nouveau dictionnaire vide.
- `ajoute : string -> int -> dictionnaire -> dictionnaire`. L'appel `ajoute clef valeur dict` renvoie un nouveau dictionnaire identique au dictionnaire `dict`, sauf qu'un couple `(clef, valeur)` y a été ajouté. Cette fonction s'exécute en temps $O(\log n)$ où n est le nombre d'entrées du dictionnaire.
- `contient : string -> dictionnaire -> bool`. L'appel `contient clef dict` renvoie un booléen indiquant s'il y a un couple dont la clef est `clef` dans le dictionnaire `dict`. Cette fonction s'exécute en temps $O(\log n)$ où n est le nombre d'entrées du dictionnaire.
- `valeur : string -> dictionnaire -> int`. L'appel `valeur clef dict` renvoie la valeur associée à la clef `clef` dans le dictionnaire `dict`. Cette fonction s'exécute en temps $O(\log n)$ où n est le nombre d'entrées du dictionnaire. Cette fonction ne peut être appelée que si la clef `clef` est présente dans le dictionnaire.

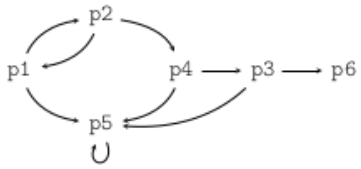
On suppose que le type de données `dictionnaire` est prédéfini ; on ne demande pas de l'implémenter.

Question 2.3 : Coder `unique : string list -> string list * dictionnaire`, qui est telle que `unique liste` renvoie un couple `(liste', dict)` où `liste'` est la liste des chaînes de caractères de `liste` distinctes (dans l'ordre de leur première occurrence dans `liste` et où `dict` associe à chaque chaîne de caractères dans `liste'` sa position dans `liste'` (en numérotant à partir de `0`). Ainsi l'appel `unique ["x";"zz";"x";"x";"zz";"yt"]` renvoie un couple formé de la liste `["x";"zz";"yt"]` et d'un dictionnaire associant à `"x"` la valeur `0`, à `"zz"` la valeur `1` et à `"yt"` la valeur `2`.

Question 2.4 : Quelle est la complexité de la fonction `unique` en terme de la longueur `n` de la liste `liste` en argument et du nombre `m` d'éléments distincts dans la liste `liste` ? Justifier la réponse.

Crawler simple

Nous allons maintenant implémenter un crawler simple en Caml. On suppose fournie une fonction `recupere_liens : string -> string list` prenant en argument l'URL d'une page Web `p` et renvoyant la liste des URL des pages `q` pour lesquelles il existe un hyperlien de `p` à `q`, dans l'ordre lexicographique. Pour illustrer le comportement de cette fonction, nous considérons un exemple de mini-graphe du Web à six pages et neuf hyperliens comme suit :



Dans cette représentation, `p1`, `p2`, etc., sont les URL de pages Web (simplifiées pour l'exemple), et les arcs représentent les hyperliens. Dans ce mini-graphe, un appel à `recupere_liens "p1"` retourne `["p2"; "p5"]`.

Un crawler est un programme qui, à partir d'une URL, parcourt le graphe du Web en visitant progressivement les pages dont les liens sont présents dans chaque page rencontrée, en suivant une stratégie de parcours de graphe (par exemple, largeur d'abord, ou profondeur d'abord). À chaque nouvelle page, si celle-ci n'a pas déjà été visitée, tous ses hyperliens sont récupérés et ajoutés à une liste de liens à traiter. Le processus s'arrête quand une condition est atteinte (par exemple, un nombre fixé de pages ont été visitées). Le résultat renvoyé par le crawler, que l'on définira plus précisément plus loin, est appelé un crawl.

Question 2.5 : Coder `crawler_bfs : int -> string -> (string * string list) list` qui prend en entrée un nombre `n` de pages et une URL `u` et renvoie en sortie une liste de longueur au plus `n` de couples `(v, l)` où `v` est l'url d'une page visitée (les pages apparaissant dans l'ordre où elles ont été visitées) et `l` la liste des liens récupérés sur la page `v`. On demande que `crawler_bfs` parcoure le graphe du Web en suivant une stratégie en largeur d'abord (breadth-first search), c'est-à-dire en visitant en priorité les pages rencontrées le plus tôt dans l'exploration. Le crawler doit visiter `n` pages distinctes, et donc appeler `n` fois la fonction `recupere_liens` (sauf s'il n'y a plus de pages à visiter). On utilisera une variable de type `dictionnaire` pour se souvenir des pages déjà visitées.

Par exemple, sur le mini-graphe, `crawler_bfs 4 "p1"` pourra renvoyer le résultat :

```
["p1", ["p2"; "p5"]; "p2", ["p1"; "p4"]; "p5", ["p5"]; "p4", ["p3"; "p5"]]
```

Question 2.6 : Même question mais avec une fonction `crawler_dfs` qui doit suivre une stratégie en profondeur d'abord (depth-first search), c'est-à-dire en visitant en priorité les pages rencontrées le plus récemment dans l'exploration.

Par exemple, sur le mini-graphe, `crawler_dfs 4 "p1"` pourra renvoyer le résultat :

```
["p1", ["p2"; "p5"];  "p2", ["p1"; "p4"]; "p4", ["p3"; "p5"]; "p3", ["p5"; "p6"]]
```

Question 2.7 : Coder une fonction Caml `construit_graphe : (string * string list) list -> string list * int array array` telle que si `crawl` est le résultat renvoyé par un crawler, alors `construit_graphe crawl` est un couple `(l, g)` où `l` est une liste de toutes les URL de pages constenues dans la liste `crawl` et `g` est la matrice d'adjacence du sous-graphe partiel du Web restreint aux pages de la liste `l` : g_{ij} est le nombre de liens découverts dans le crawl de la page d'indice `i` dans `l` vers la page d'indice `j` dans `l`. On fera commencer les indices à `0`. Pour coder la fonction `construit_graphe`, on pourra utiliser les fonctions `aplatir` et `unique`.

Par exemple, sur le mini-graphe, si `crawl` est une variable contenant le résultat de l'appel `crawler_bfs 4 "p1"`, alors `construit_graphe crawl` doit renvoyer :

```
["p1"; "p2"; "p5"; "p4"; "p3"],
[|[|0; 1; 1; 0; 0|]; [|1; 0; 0; 1; 0|]; [|0; 0; 1; 0; 0|]; [|0; 0; 1; 0; 1|]; [|0; 0; 0; 0; 0|]|]
```

En particulier, `p3` apparaît même s'il n'a pas été visité dans le crawl, mais `p6` n'apparaît pas car il n'a pas été découvert dans le crawl. En outre, l'hyperlien de `p3` à `p5` n'apparaît pas car `p3` n'a pas été visité.

Partie 3 : Logique (annales)

Imaginez-vous ethnologue. Vous étudiez une peuplade primitive qui présente un comportement manichéen extrême : lorsque plusieurs personnes participent à une même conversation sur un sujet donné, elles vont toutes avoir le même comportement manichéen tant que la conversation reste sur le même sujet, c'est-à-dire que toutes les affirmations seront soit des vérités, soit des mensonges. Par contre, si le sujet de la conversation change, la nature des affirmations, soit mensonge, soit vérité, peut changer, mais toutes les affirmations seront de la même nature tant que le sujet ne changera pas à nouveau. Pour être autorisé à séjourner dans cette peuplade, vous devez respecter cette règle. Vous participez à une conversation avec trois de leurs membres que nous appellerons X, Y et Z. Ceux-ci vous indiquent comment rejoindre leur village. Si vous n'arrivez pas à le rejoindre, vous ne serez pas autorisé à y séjourner. Le premier sujet abordé est la région dans laquelle se trouve le village :

— X indique : « Le village se trouve dans la vallée » ;
— Z réplique : « Non, il ne s'y trouve pas » ;
— X reprend : « Ou alors dans les collines ».

Nous noterons V et C les variables propositionnelles associées à la région dans laquelle se trouve le village. Nous noterons X_1 et Z_1 les formules propositionnelles correspondant aux affirmations de X et de Z sur le premier sujet. Puis, le second sujet est abordé : le chemin qui permet de rejoindre le village dans la région concernée.

— X dit : « Le chemin de gauche conduit au village » ;
— Z répond : « Tu as raison » ;
— X complète : « Le chemin de droite y conduit aussi » ;
— Y affirme : « Si le chemin du milieu y conduit, alors celui de droite n'y conduit pas » ;
— Z indique : « Celui du milieu n'y conduit pas ».

Nous noterons G, M, D les variables propositionnelles correspondant respectivement au fait que le chemin de gauche, du milieu et de droite, conduit au village. Nous noterons X_2, Y_2 et Z_2 les formules propositionnelles correspondant aux affirmations de X, de Y et de Z sur le second sujet.

Question 3.1 : Représenter le comportement manichéen des interlocuteurs dans le premier sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles X_1 et Z_1.

Question 3.2 : Représenter les informations données par les participants sous la forme de deux formules du calcul des propositions X_1 et Z_1 dépendant des variables V et C.

Question 3.3 : En utilisant la résolution avec les propriétés des opérateurs booléens et les formules de De Morgan en calcul des propositions, déterminer dans quelle région vous devez vous rendre pour rejoindre le village.

Question 3.4 : Représenter le comportement manichéen des interlocuteurs dans le second sujet abordé sous la forme d'une formule du calcul des propositions dépendant des formules propositionnelles X_2, Y_2 et Z_2.

Question 3.5 : Représenter les informations données par les participants sous la forme de trois formules du calcul des propositions X_2, Y_2 et Z_2 dépendant des variables G, M et D.

Question 3.6 : En utilisant la résolution avec les tables de vérité en calcul des propositions, déterminer quel chemin vous devez suivre pour rejoindre le village.

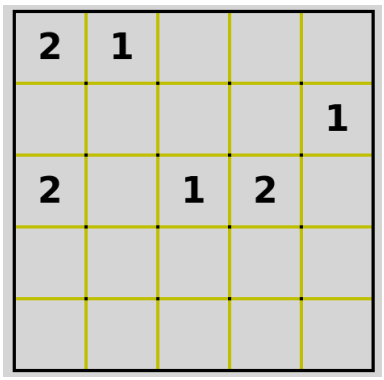
Question 3.7 : En admettant que les trois participants aient menti, pouviez-vous prendre d'autres chemins ? Si oui, le ou lesquels ?

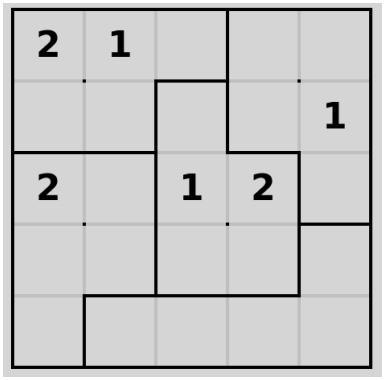
Partie 4 : Logique (la traditionnelle énigme)

Résoudre l'instance du jeu « Palissade »ci-après.

Le principe du jeu est de séparer une grille rectangulaire en secteurs connexes de même surface. La surface est mentionnée au côté de la grille, et des indices dans certaines cases de la grille imposent le nombre de délimiteurs (palissades) qui devront être au contact de la case en question, parmi les quatre côtés possibles.

Un exemple et sa solution permettent de mieux fixer les idées, avec une surface de cinq unités d'aire pour chaque secteur.





Et voici la grille de l'exercice, avec une surface de dix unités d'aire pour chaque secteur.