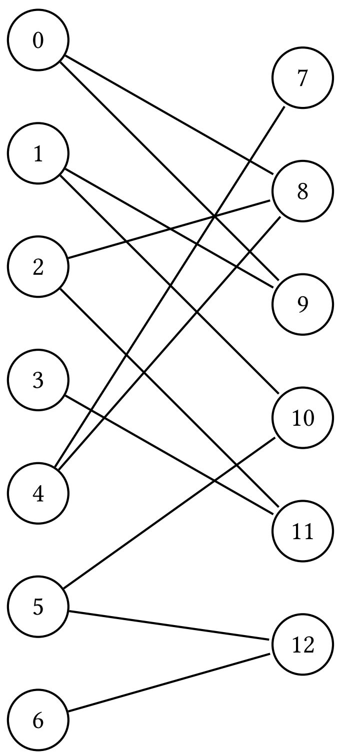
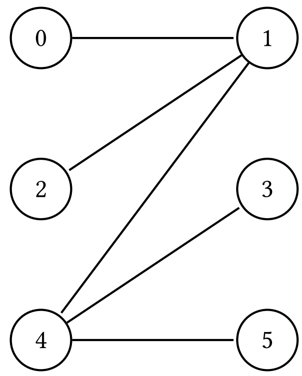
Couplages dans un graphe biparti

A Implémentation

On fournit avec ce TP un fichier `couplages.ml` à trou. Dans ce fichier se trouvent les deux graphes suivants (pour les tests).



(a) Graphe biparti `gb` exemple du cours



(b) Graphe biparti `gb2` tel que le meilleur couplage vérifie |C| = 2

Le graphe `gb_non_connexe` est constitué de ces deux graphes, mis l'un à côté de l'autre (deux composantes connexes).

On travaillera dans ce TP avec les types suivants :

OCaml
```ocaml
type sommet = int
type graphe = sommet list array

type arete = {x : sommet; y : sommet}

type couplage = arete list
type graphe_biparti = { g : graphe; partition : bool array }
```

- Comme d'habitude, on représente un graphe par listes d'adjacence.
- Une arête est donnée par ses deux extrémités (notées `x` et `y`).
- Un couplage est donné sous la forme d'une liste d'arêtes.
- Enfin, un graphe biparti est la donnée d'un graphe (qui doit être biparti) et d'une bipartition de ce graphe, sous la forme d'un tableau de booléens. Si $S = X \sqcup Y$, alors les sommets $s \in S$ appartenant à l'ensemble X vérifient `partition.(s) = true` et ceux appartenant à Y vérifient `partition.(s) = false`.

N'hésitez pas à regarder les deux exemples `gb` et `gb2` fournis.

A.1 Fonctions intermédiaires utiles

Commençons par quelques fonctions intermédiaires sur les couplages.

1. Écrire une fonction `est_dans_couplage : arete -> arete list -> bool` telle que `est_dans_couplage ar c` renvoie `true` si l'arête `ar` est présente dans le couplage `c` et `false` sinon.

   Attention : une arête entre deux sommets u et v peut être représentée de deux manières suivantes, dans le sens $u \to v$ ou $v \to u$. Dans les deux cas, on veut renvoyer `true`.

2. Écrire une fonction `difference_symetrique : arete list -> arete list -> arete list` telle que `difference_symetrique c1 c2` renvoie l'ensemble d'arêtes `c1`Δ`c2`, sous la forme d'une liste d'arêtes (sans doublons).

   Indication : si besoin, vous pouvez commencer par une fonction intermédiaire `prive_de` *qui renvoie* `c\c'`.

3. Écrire une fonction `est_couvert : sommet -> arete list -> bool` telle que `est_couvert s c` renvoie `true` si le sommet `s` est couvert par le couplage `c` et `false` sinon, c'est à dire si le sommet `s` donné est une extrémité d'une des arêtes du couplage.

A.2 Graphe de couplage et recherche de chemin augmentant

On rappelle l'algorithme générique pour trouver un couplage de cardinalité maximale dans un graphe :

Algorithme 4 : Couplage maximum dans un graphe

Entrées : Un graphe $G = (S, A)$ non orienté.
Sorties : Un couplage $C \subseteq A$ de cardinal maximal.

$C \leftarrow \emptyset$
tant que *il existe un chemin c augmentant pour C* **faire**
  $C \leftarrow C \Delta c$
renvoyer C

Pour implémenter cet algorithme dans le cas des graphes bipartis, nous avons besoin de construire les graphes du couplage G_C successifs.

4. Écrire une fonction `graphe_de_couplage : graphe_biparti -> arete list -> graphe` telle que `graphe_de_couplage gb c` renvoie le graphe G_C du couplage `c` dans le graphe biparti `gb`.
   Le sommet s ajouté sera d'indice n et le sommet t ajouté sera d'indice $n+1$, avec $n = |S|$.

   Remarque : le graphe G_C renvoyé est orienté, et on ne demande pas d'en renvoyer une bipartition (ça n'aurait pas de sens), seulement le graphe.

   Indication : Si besoin, relisez bien attentivement la définition de graphe de couplage du cours, et essayez de la suivre pas à pas (ajouter les arêtes de s vers ..., puis de ... vers t, etc). Cette définition est à connaître !

Dans le graphe du couplage G_C, on souhaite trouver un chemin de s à t pour en déduire un chemin augmentant de C dans le graphe d'origine. Pour cela, on effectue un simple parcours de graphe (qui retient le tableau des prédécesseurs).

5. Écrire une fonction `arbre_parcours : graphe -> sommet -> sommet array` telle que `arbre_parcours g s` renvoie l'arbre issu d'un parcours de graphe (de votre choix) depuis `s`, sous la forme d'un tableau `pred` des prédécesseurs de chaque sommet dans le parcours. La racine aura elle-même pour prédécesseur, et les sommets inaccessibles auront la valeur `-1` pour prédécesseur.

6. Écrire une fonction `chemin : graphe -> sommet -> sommet -> arete list` telle que `chemin g s t` renvoie un chemin reliant les sommets `s` à `t` dans le graphe `g`, s'il en existe, sous la forme d'une liste des arêtes à emprunter successivement depuis `s` pour atteindre `t` (**dans le bon sens, x vers y**). S'il n'en existe pas, on renverra la liste vide `[]`.

A.3 Recherche de couplage maximum

On a désormais tous les outils dont nous avons besoin ! Il ne reste plus qu'à implémenter l'algorithme de recherche de couplage de cardinalité maximale, rappelé plus haut !

Toute la difficulté réside dans le fait de tester et trouver un chemin augmentant c pour C, ce qui nécessite, à chaque tour de boucle, de construire le graphe G_C du couplage actuel. A vous de jouer !

7. Écrire une fonction `couplage_maximum_biparti : graphe_biparti -> arete list` telle que `couplage_maximum_biparti gb` renvoie un couplage de cardinalité maximale du graphe biparti `gb` donné en entrée, sous la forme d'une liste d'arêtes (sans doublons).
8. Testez votre fonction sur les graphes `gb`, `gb2` et `gb_non_connexe` fournis [1]. Pour le premier, vous devriez obtenir un couplage (valide) à 6 arêtes. C'est le mieux qu'on puisse faire car $|Y| = 6$. Pour le graphe `gb2`, vous ne pourrez obtenir qu'un couplage à 2 arêtes, il n'est pas possible de former trois paires dans cette configuration.

1. J'espère toutefois que vous avez testé vos autres fonctions au fur et à mesure !