Épreuve orale de « *Mathématiques et algorithmique* » de la Banque PT – Rapport 2026

Les futurs candidats trouveront dans ce rapport des remarques et des conseils qui pourraient leur être utiles pour leur futur passage. Quelques exercices type posés cette année sont annexés en fin de document.

Nous suggérons aux futurs candidats de consulter le **site de la Banque PT**, où ils trouveront le mémento *Python* fourni lors de l'oral, les rapports des années antérieures et des exercices qui ont été posés lors de sessions antérieures, à titre d'exemples.

1 Objectifs de l'épreuve

Le but de l'épreuve *Mathématiques et algorithmique* est avant tout de contrôler l'assimilation des connaissances des programmes de première et deuxième années, en mathématiques (algèbre, analyse, géométrie et probabilités) comme en informatique (semestres 1 et 2, ainsi que le paragraphe 3.2 du semestre 3), et ce pour toute la filière, sans oublier les connaissances de base du programme du lycée (seconde, première, terminale). L'esprit de l'épreuve induit donc un fort niveau d'exigence sur les savoirs et savoir-faire fondamentaux, en mathématiques comme en informatique.

Cette épreuve permet aussi d'examiner des compétences générales attendues d'un ingénieur :

- l'aptitude du candidat à *lire* attentivement un sujet ;
- son aisance à *exposer* clairement ses idées avec un vocabulaire précis, des enchainements logiques cohérents, un niveau de détails suffisant et à répondre précisément aux questions posées ;
- sa capacité d'initiative et son autonomie et, en même temps, son aptitude à écouter l'interrogateur·rice, à prendre en compte ses indications, à lui demander des précisions si besoin ;
- son aptitude à mettre en œuvre ses connaissances et son savoir-faire en mathématiques et informatique pour *résoudre un problème* par la réflexion et non par la mémorisation de solutions toutes faites ;
- sa faculté à *interpréter et critiquer* les résultats obtenus et à changer de méthode en cas de besoin.

L'évaluation prend en compte cet ensemble de connaissances et compétences.

L'épreuve n'est pas un simple écrit transposé à l'oral. Si les questions du sujet suivent une progression construite, elles servent surtout de point de départ à un dialogue, qui permet au jury d'évaluer au plus juste les capacités des candidates et candidats. Il ne faut donc pas s'étonner, ni s'inquiéter, si le jury propose d'expliquer oralement une seconde méthode ou une variante du problème : cela fait partie du principe même de l'épreuve. D'ailleurs, il est tout à fait possible d'obtenir la note maximale sans avoir traité l'intégralité du sujet.

L'épreuve n'est pas non plus une « *colle* » : il s'agit bien d'une évaluation et non d'une formation ; il ne faut donc pas s'attendre à ce que l'on donne la solution à la fin de l'épreuve. D'autre part, il est

important que les candidates et candidats exposent avec précision l'intégralité des méthodes et calculs permettant la résolution du problème posé comme si le jury n'était pas spécialiste. Certains sont, à tort, tentés de laisser des arguments implicites : même si le jury devine où la candidate ou le candidat veut en venir, l'évaluation ne peut prendre en compte que les éléments effectivement exprimés et juge la qualité d'exposition.

Les jurys accueillent les candidates et candidats avec bienveillance. Les remarques du jury apportent des précisions ou des indications, l'objectif étant toujours de valoriser le travail réalisé, les connaissances acquises, les compétences maîtrisées. Nous invitons donc les candidates et candidats à accueillir nos questions et commentaires avec sérénité et à les prendre en compte pour la suite de l'épreuve.

2 Modalités de l'épreuve

Cette session s'est déroulée sur 11 jours dans les locaux de *l'École Nationale Supérieure d'Arts et Métiers*, Paris (13^{e} arrondissement). Les candidates et candidats sont convoqués 15 minutes avant l'heure de passage. La durée de l'oral de *Mathématiques et algorithmique* est de 1 heure et commence par la vérification de l'identité et de la convocation puis par la signature du cahier de jury.

L'épreuve comporte ensuite deux parties d'environ 30 minutes, dont l'ordre est défini par l'interrogateur, l'une portant sur un exercice de mathématiques et l'autre sur un exercice d'algorithmique.

- L'exercice de mathématiques porte sur le programme de mathématiques des deux années de la filière PTSI/PT (algèbre, analyse, géométrie et probabilités) et se déroule au tableau.
- L'exercice d'algorithmique se déroule sur ordinateur et porte sur les semestres 1 et 2, sur le paragraphe 3.2 du semestre 3 du programme d'informatique, ainsi que sur la maîtrise des outils numériques et mathématiques de base, mentionnés en annexe des programmes de physique-chimie de la filière PTSI/PT.

  Pour cet exercice, les candidates et candidats disposent d'un ordinateur (Windows, clavier français Azerty) dans lequel sont installés `Python 3`, ses bibliothèques standard, ainsi que `numpy`, `scipy` et `matplotlib`, aides incluses, d'un mémento plastifié en couleurs au format A3, et de feuilles de brouillon.

  ⇨ **À partir de la session 2027,** l'environnement de développement sera `Pyzo` (avec également un environnement `Idle` mais il n'y aura plus `Idlex`).

Au cours de chaque exercice, on alterne des phases de réflexion et d'écriture (code ou tableau) lors desquelles la candidate ou le candidat est autonome et des phases d'interaction avec le jury, par le biais éventuel d'une feuille de brouillon pour l'exercice sur ordinateur si cela facilite les échanges.

Les salles d'interrogation accueillent en général deux jurys simultanément et chaque jury gère en parallèle deux candidates ou candidats (une personne entrant toutes les 30 minutes). Le temps passé auprès d'une candidate ou d'un candidat correspond au temps d'autonomie de l'autre. Il est donc important de respecter les temps de parole et de réflexion accordés par le jury, ainsi que d'adopter une attitude compatible avec le travail des autres (volume sonore, attitude générale). Précisons également qu'il n'est pas raisonnable, sauf exception, de souhaiter porter des bouchons d'oreille pendant l'interrogation.

Le respect de la durée d'interrogation peut être frustrant mais est essentiel pour l'équilibre de l'évaluation (les notes des deux exercices sont indépendantes) ainsi que pour le bon déroulement de la journée (certains enchainent plusieurs épreuves) et pour garantir l'égalité de traitement de tous. En cas de bref retard d'un candidat, le jury essaye d'être bienveillant mais n'est pas obligé de compenser le temps perdu ; en cas de retard prononcé justifié, le cas est référé à la direction du concours par le jury de coordination.

Les candidates et candidats disposant d'un aménagement d'épreuve suivent un déroulé d'épreuve similaire, avec par exemple 5 minutes de plus par exercice (tiers-temps préparation) ou 10 minutes de plus par exercice (tiers-temps préparation et interrogation). Certains aménagements spécifiques (ou cas de force majeure pour tous) peuvent conduire à l'isolement de la candidate ou du candidat avec le jury de coordination, dans des conditions d'épreuve semblables.

3 Analyse statistique de la session 2026

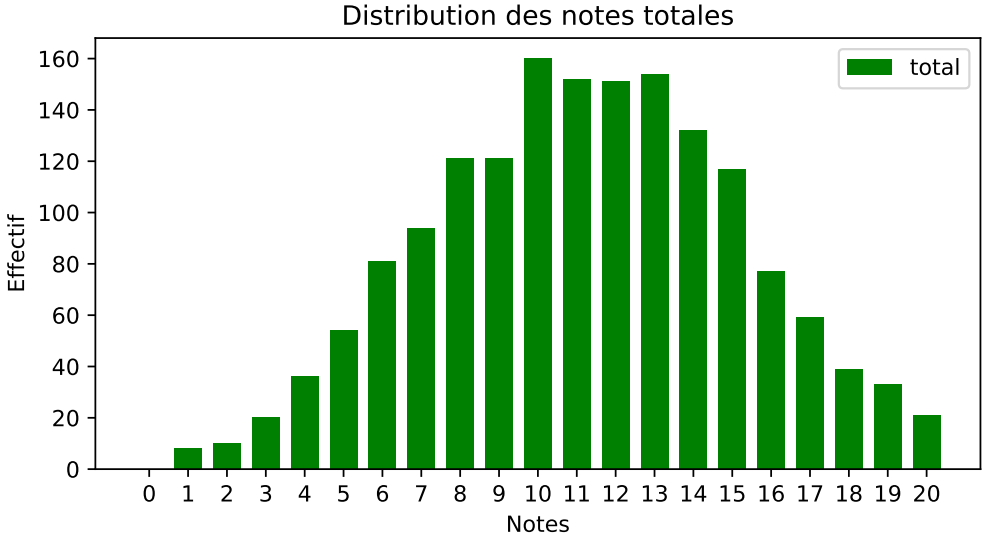
La session 2026 comportait 10 jurys en plus du jury de coordination. Le jury était constitué de 6 interrogatrices et 14 interrogateurs, enseignant·e·s en classe préparatoire (15), à l'université (4) ou en grande école (1), en région parisienne (9) ou en province (11). Précisons qu'aucun membre du jury n'est en poste ou colleur en filière PTSI/PT.

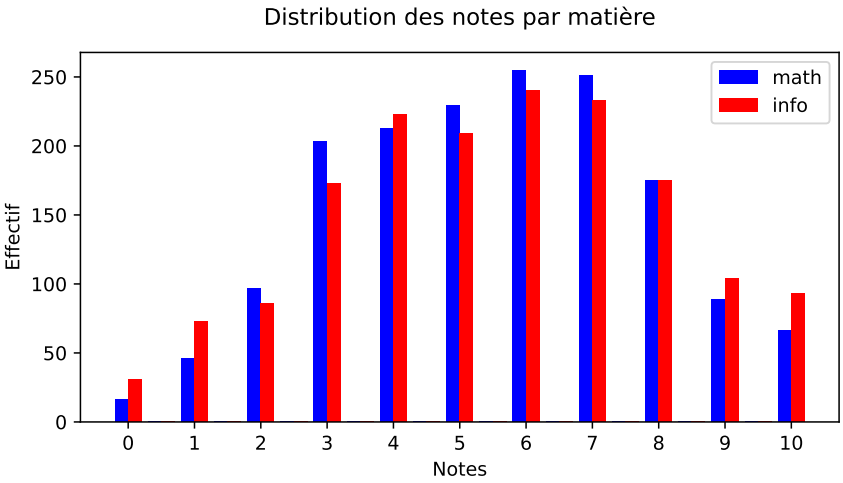
En 2026, 1642 candidats ont passé cet oral. Le jury s'est efforcé de poser des exercices balayant l'ensemble du programme, tant en mathématiques qu'en algorithmique. Ainsi, 202 exercices différents d'analyse et de probabilités ainsi que 164 exercices différents de géométrie et d'algèbre ont été posés. De plus, 309 exercices différents d'algorithmique ont été posés.

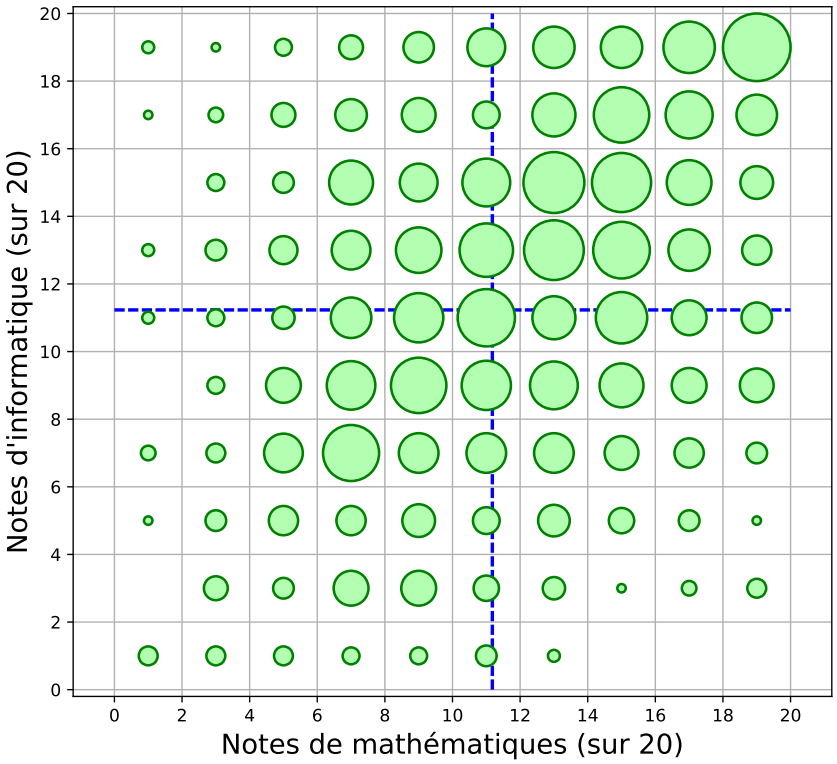
La moyenne de l'épreuve est de 11,21/20 avec un écart-type de 3,93. Les écarts entre interrogateurs sont minimes et cohérents avec un tirage d'échantillons aléatoires proportionnés aux nombres d'interrogations réalisés.

La plage de notation s'étend de 1 à 20. L'histogramme des notes est équilibré et reflète le fait que les jurys ont valorisé les compétences des candidates et candidats avec finesse. La distribution symétrique des notes entre les exercices de mathématiques et d'informatique atteste que les deux exercices sont évalués indépendamment et qu'une difficulté avec l'un peut, au moins partiellement, être compensée avec l'autre.

Oral 2026	**Note (sur 20)**	Math. (sur 20)	Algo. (sur 20)
Moyenne	**11,21**	11,18	11,23
Écart-type	**3,93**	4,56	4,90
Minimum	1	0	0
Maximum	20	20	20







4 Conseils aux candidates et candidats

Comme lors des sessions précédentes, la plupart des candidats se sont montrés bien préparés à cette épreuve.

Nous avons cependant observé, chez un certain nombre d'entre eux, une tendance à survoler l'énoncé plutôt qu'à le lire attentivement, et à se précipiter vers une réponse (calcul ou code) sans s'être assurés d'avoir bien compris l'ensemble de la question. Cette attitude traduit **une difficulté grandissante dans la compréhension du langage**, en particulier à l'écrit, ainsi qu'**un manque de précision dans l'expression et dans le raisonnement** – deux points de vigilance importants pour de futurs ingénieurs, et sur lesquels un travail régulier permet des progrès. De manière générale, mieux vaut éviter aussi bien la précipitation ou la sur-excitation que la passivité, le mutisme ou l'obstination dans une voie infructueuse.

Nous invitons donc les candidats à être **précis** dans leurs propos ; en particulier, lorsqu'ils énoncent

une définition, une propriété ou un théorème au programme de mathématiques, il est important de rappeler l'ensemble des hypothèses, sans en oublier. Le jury attend des candidats qu'ils connaissent les résultats au programme. Les justifications et commentaires gagnent à être donnés au moment même où l'on est interrogé ; le temps étant limité, il n'est pas nécessaire de développer de longues phrases, sauf demande explicite du jury pour clarifier un point.

On attend des candidates et candidats qu'ils sachent identifier et décrire précisément le type de problème à résoudre. Plutôt que d'appliquer des « recettes », il est valorisant de discuter des outils les mieux adaptés au problème posé. Être capable de justifier le choix d'une **méthode**, et de reconnaître les cas où elle ne s'applique pas, est une compétence particulièrement appréciée du jury.

Les candidats doivent également s'attendre à être interrogés sur la nature des objets qu'ils manipulent ; par exemple, savoir distinguer sans les confondre un nombre, une fonction, un vecteur, une équation cartésienne ou une représentation paramétrique. Au détour d'une question sur un équivalent ou sur les éléments propres d'une matrice, il est donc utile d'avoir en tête quelques définitions clés. D'une manière générale, la maîtrise des techniques de calcul gagne à s'accompagner d'une bonne connaissance des définitions des **concepts** sous-jacents.

Le jury donne volontiers des indications lorsqu'un candidat reste bloqué trop longtemps, ou lorsque celui-ci sollicite de l'aide par des questions dont il reconnaît implicitement ignorer la réponse (par exemple : « *Est-ce que je peux utiliser tel théorème ?* », ou « *Pourquoi la figure ne s'affiche-t-elle pas ?* »). Il est toujours préférable, lorsqu'on sollicite de l'aide, d'expliquer les pistes déjà envisagées et les raisons pour lesquelles elles ne semblent pas aboutir, plutôt que de se contenter d'un « *Je ne vois pas* » ou d'un « *Ça ne marche pas* ». Nous encourageons les candidats à bien écouter les consignes du jury : il vaut mieux attendre qu'il ou elle ait terminé de parler avant de répondre, quitte à reformuler brièvement l'indication ou la question pour s'assurer de l'avoir bien comprise ; de même, une consigne du type « *je vous laisse continuer* » signifie que la phase d'échange est terminée et qu'il convient de poursuivre sa réflexion en autonomie.

En mathématiques, nous avons repéré des **fragilités sur les calculs**, y compris les plus simples. Nous encourageons donc les candidats à consolider leur aisance avec les nombres complexes et leurs différentes représentations, avec les développements ou simplifications d'expressions algébriques, ainsi qu'avec les calculs de déterminants, de dérivées, d'intégrales, etc.

Pour l'informatique, nous conseillons vivement de se placer, tout au long des deux années de préparation, dans les conditions réelles de passage de l'oral, en installant par exemple l'environnement virtuel Python 3 dédié (voir **Formations Python 3 Arts et Métiers**). Nous rappelons qu'à partir de la prochaine session, l'environnement de travail par défaut sera `Pyzo`.

Nous invitons les candidats à respecter avant toute chose le principe de base suivant : « ne pas appeler plusieurs fois la même fonction avec les mêmes arguments » ; son non-respect traduit une compréhension encore incomplète de l'algorithmique et de la programmation. De même, sans entrer dans le détail de la complexité, il est important de rester attentif à l'ordre de grandeur des tâches demandées à l'outil informatique : on a vite fait de lancer 10^9 calculs en appliquant un algorithme en $O(N^3)$ avec $N = 1000$, et il est utile de savoir repérer (et éviter) ce type de situation.

Le code doit toujours être testé de façon appropriée, soit dans l'éditeur, soit dans la console, comme cela est précisé dans l'en-tête de chaque énoncé. Lors du dialogue avec le jury, nous encourageons les candidats à illustrer leurs résultats par des évaluations en console, plutôt qu'en commentant – ou pire en effaçant – la suite du code en cours de développement pour y ajouter une ligne `print(...)` en réponse à une question orale. Une bonne utilisation de la console témoigne d'une réelle aisance avec l'outil informatique, ainsi que de capacités à déboguer un code et à dialoguer avec un futur collègue ingénieur.

Il est utile de maîtriser les fonctionnalités de base des listes, chaînes et itérables (`sum`, `max`, `min`, `sort`, `sorted`, `x in L`, `nom[a:b]`, `nom[-1]`, `append`, `extend`, `index`, `list`, `split`, `replace`, `strip`), ce qui évite

de consacrer l'essentiel du temps à recoder maladroitement l'une d'entre elles. Il convient toutefois de bien veiller à les utiliser à bon escient : `append`, par exemple, est souvent sur-employé.

Comme en mathématiques, la qualité de l'expression est essentielle en informatique. Il est important de savoir identifier le type des objets manipulés et de cultiver une bonne hygiène de code. Nous conseillons de préférer une boucle `for` à un `while` lorsque le nombre d'itérations est connu à l'avance. Les interruptions de boucle par `return` ou par `break` sont autorisées, à condition de veiller à l'indentation et de pouvoir en justifier l'usage sur le plan algorithmique. L'utilisation d'une boucle `while` nécessite de s'assurer de sa terminaison, par l'évolution d'au moins une variable de boucle, en ajoutant éventuellement un contrôle fixant le nombre maximum d'itérations.

Sébastien Pellerin et François Vigneron, Coordonnateurs de l'épreuve orale de « *Mathématiques et algorithmique* » de la Banque PT, le 20 juillet 2026.

Annexe - exemples d'exercices d'informatique

Pour finir ce rapport nous publions quelques exercices sortis de la banque cette année et qui sont représentatifs de ce qui peut être posé aux candidats.

Exercice d'informatique 1

1. Écrire une fonction `miroir` qui prend en entrée une liste et qui renvoie une liste où l'ordre des éléments est inversé. Si la liste est `[0,1,2,1]`, la fonction renvoie `[1,2,1,0]`.
2. On appelle *monotonie* une liste (éventuellement vide) triée par ordre croissant ou strictement décroissant : `[]`, `[2]`, `[1,1,2,3,3]`, `[3,2,0]` sont des monotonies, mais pas `[3,3,1]` ni `[3,4,2]`.

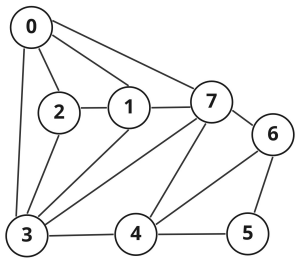
   Écrire une fonction `monotone` d'argument une monotonie `L` et un entier `x`, et qui renvoie `True` si `L+[x]` est une monotonie et `False` sinon.
3. Écrire une fonction `decoupe` qui prend en entrée une liste `L` et qui renvoie une liste des sous-listes de `L` qui partitionnent `L` en monotonies non vides.

   Par exemple, `decoupe([1,0,2,3,3,4,1,2,1])` renvoie `[[1,0],[2,3,3,4],[1,2],[1]]`.
4. Écrire une fonction `rearrange` qui prend en entrée une liste de monotonies et qui renvoie, en réarrangeant les monotonies strictement décroissantes, une liste de monotonies croissantes.

   Par exemple, `rearrange([[0,2],[1,0],[3,2,1],[2]])` renvoie `[[0,2],[0,1],[1,2,3],[2]]`.
5. Écrire une fonction `fusion` qui prend en entrée deux listes croissantes et qui renvoie la liste (croissante) résultant de la fusion des deux listes. Discuter de la complexité de cette fonction.
6. Écrire une fonction `tri` qui prend en argument une liste d'entiers et qui renvoie la liste triée dans l'ordre croissant en fusionnant les unes avec les autres les monotonies calculées, celles qui sont décroissantes ayant au préalable été inversées.

Exercice d'informatique 2

On considère le graphe non orienté suivant :



1. Représenter le graphe de l'énoncé à l'aide d'un dictionnaire `G` dont les clés sont les noms (entiers de 0 à 7) des sommets et les valeurs associées sont les listes des voisins.
2. Écrire une fonction `voisins(gr, i, j)` où `gr` est un dictionnaire représentant un graphe comme ci-dessus, `i` et `j` sont des numéros de sommets différents, et qui renvoie un booléen indiquant si ces deux sommets sont voisins ou non.
3. Écrire une fonction `aretes(gr)` où `gr` est un dictionnaire représentant un graphe comme ci-dessus et qui renvoie le nombre d'arêtes de ce graphe.

   Proposer un jeu de trois tests pour illustrer cette fonction en expliquant vos choix.

Le but de ce qui suit est de colorer ce graphe (c'est-à-dire d'attribuer une couleur pour chaque sommet) de sorte que deux sommets adjacents n'aient pas la même couleur et en cherchant à minimiser le nombre de couleurs. Les couleurs seront représentées par des numéros : 0, 1, etc.

4. La fonction `colorersommet(gr,i,couleurs)` ci-dessous prend en arguments un dictionnaire `gr` représentant un graphe (comme ci-dessus), un entier `i` correspondant à un sommet et une liste `couleurs` (dont la longueur est le nombre de sommets) et qui indique le numéro de la couleur attribuée au sommet ou -1 si le sommet n'est pas coloré.

   Cette fonction ne renvoie rien mais modifie la liste `couleurs` pour attribuer le plus petit numéro possible au sommet `i`.

   Compléter les lignes 5, 6, 8 et 9 en remplaçant les tirets :

```
1  def colorersommet(gr, i, couleurs):
2      n = len(couleurs)
3      col = [] # liste des couleurs des voisins de i deja colores
4      for j in gr[i]:
5          if ------
6              ------
7      new = 0
8      while ------:
9          ------
10     couleurs[i] = new
```

5. Écrire une fonction `colorer(gr)` d'argument un dictionnaire `gr` représentant un graphe et qui renvoie la liste des couleurs attribuées aux sommets.
6. Écrire une fonction analogue `colorerbis` prenant un argument supplémentaire : une liste `ordre` fixant l'ordre de coloration des sommets. Cet ordre semble-t-il avoir une influence sur le nombre de couleurs utilisées ? Pour le cas de figure traité ici, quel est le nombre minimum de couleurs utilisées ?

Exercice d'informatique 3

On suppose que n personnes veulent prendre le train un jour donné. La personne `i` veut prendre le train numéro `voyageur[i]`. Le train `j` a une capacité `capacites[j]`. Les trains partent dans l'ordre de leur numéro.

1. Écrire une fonction `voeux` qui prend en entrée une liste `voyageurs` correspondant aux voeux des voyageurs et un nombre de trains `nt`, qui renvoie une liste `trains` de taille `nt` telle que `trains[i]` est la liste dans l'ordre croissant des voyageurs qui désirent prendre le train numéro `i`.

   Par exemple si `voyageurs=[1,2,1,0,4,1,0]` et `nt=5` alors `trains=[[3,6],[0,2,5],[1],[],[4]]`.
2. Écrire une fonction `possible1`, qui prend en entrée une liste `voyageurs` et une liste `capacites` correspondant aux capacités des différents trains, qui renvoie un booléen indiquant s'il est possible de satisfaire tous les voyageurs.
3. On suppose maintenant que si le train `voyageur[i]` est trop plein pour que la personne `i` puisse le prendre, elle est prête à prendre le train `voyageur[i] + 1` s'il existe.

   Écrire une fonction `possible2`, qui prend en entrée une liste `voyageurs` et une liste `capacites` correspondant aux capacités des différents trains, qui renvoie un booléen indiquant s'il est possible de satisfaire tous les voyageurs avec cette nouvelle possibilité.
4. On suppose qu'il est possible de faire voyager tout le monde dans ces nouvelles conditions.

   Écrire alors une fonction `affecte`, qui prend en entrée une liste `voyageurs` et une liste `capacites`, qui renvoie une liste `affectation` telle que `affectation[j]` est la liste des voyageurs qui vont voyager dans le train numéro `j`.

Exercice d'informatique 4

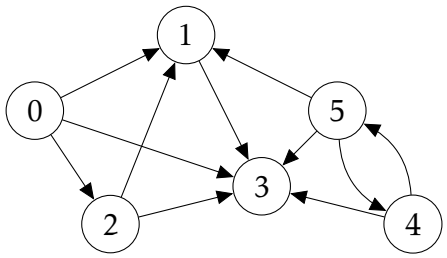
On considère un groupe de n personnes (numérotées de 0 à $n-1$ pour les distinguer) et leur graphe orienté suivant :

$$i \to j \text{ si et seulement si } i \text{ connaît } j.$$

Une *star* est quelqu'un qui ne connaît personne mais que tous les autres connaissent. Pour démasquer une star, s'il en existe une, vous avez juste le droit de poser des questions, à n'importe quel individu i du groupe, du type « est-ce que vous connaissez j ? » (noté « $i \to j$? »), on suppose que les individus disent la vérité.

On veut un algorithme qui trouve une star, s'il en existe une, ou sinon qui garantit qu'il n'y a pas de star dans le groupe (si possible en posant le moins de questions possibles).

1. Le graphe `G0` ci-dessous contient-il une star ?



2. Combien y a-t-il de stars au maximum dans un groupe ?
3. Construire une liste `L` de listes d'adjacence de `G0` puis une fonction `versmatrice(L)` qui à partir de la liste `L` de listes d'adjacence renvoie la matrice `M` d'adjacence du graphe. Tester la fonction sur le graphe de l'énoncé.
4. Comment tester en Python « $i \to j$ » ?

   Écrire une fonction `comptepredecesseur` qui prend en entrée la matrice d'adjacence `M` et un sommet `i` d'un graphe, et qui renvoie le nombre de personnes connaissant `i`.

   En notant n le nombre de personnes (le nombre de sommets du graphe), quelle est la complexité en le nombre de questions de votre fonction `comptepredecesseur` (en fonction de n) ?
5. Écrire une fonction `trouvestar(M)` qui prend en entrée la matrice d'adjacence `M` et qui renvoie la star si elle existe et `None` sinon. Que renvoie `trouvestar(G0)` ?

   Quelle est la complexité en le nombre de questions de votre fonction `trouvestar(M)` (en fonction de n) ?
6. Tester le code suivant sur le graphe de la question 1. Expliquer son comportement.

```
n = len(L)
i, j = 0, 1

while j < n:
    if M[i][j] == 1:
        i = j
    j += 1
```

7. Écrire une fonction `trouvestarlin(M)` qui renvoie la star si elle existe et `None` sinon et qui est en complexité linéaire ($O(n)$) en le nombre de questions posées. La tester sur le graphe de l'énoncé.