Concours commun Mines-Ponts, France

Session 2025, filière MPI

Informatique 2

Durée : 4 heures

Ce que traite le sujet

- Section 1.1 : implémentation du jeu en langage C
- Section 1.2 : étude de tournois en langage SQL
- Section 2 : stratégies de jeu
- Section 3 : arbres couvrants disjoints

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/mines/mpi/2025/info2

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

A2025 – INFO II MPI

ÉCOLE NATIONALE DES PONTS et CHAUSSÉES,
ISAE-SUPAERO, ENSTA PARIS,
TÉLÉCOM PARIS, MINES PARIS,
MINES SAINT-ÉTIENNE, MINES NANCY,
IMT ATLANTIQUE, ENSAE PARIS,
CHIMIE PARISTECH - PSL.

Concours Mines-Télécom,
Concours Centrale-Supélec (Cycle International).

CONCOURS 2025

DEUXIÈME ÉPREUVE D'INFORMATIQUE

Durée de l'épreuve : 4 heures

L'usage de la calculatrice ou de tout dispositif électronique est interdit.

Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :

INFORMATIQUE II - MPI

Cette épreuve concerne uniquement les candidats de la filière MPI.

L'énoncé de cette épreuve comporte 9 pages de texte.

Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

.

Les sujets sont la propriété du GIP CCMP. Ils sont publiés sous les termes de la licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Pas de Modification 3.0 France.

Tout autre usage est soumis à une autorisation préalable du Concours commun Mines-Ponts.

Le jeu de Shannon

Préliminaires

L'épreuve est composée d'un problème unique, comportant 33 questions.

Dans ce problème, nous nous intéressons au *jeu de Shannon*, qui est un jeu de stratégie dans lequel l'un des joueurs doit parvenir à établir une certaine connexion en posant des pièces de jeu, tandis que son opposant doit l'en empêcher en essayant de le bloquer. Le problème est divisé en trois sections. Dans la première section (page 1), nous introduisons les règles du jeu et une étude de tournois. Dans la deuxième section (page 5), nous analysons les stratégies de jeu conduisant l'un des joueurs à la victoire. Dans la troisième section (page 7), nous étudions la construction de deux arbres couvrants disjoints dans un graphe, ce qui est un prérequis de l'une des stratégies d'un joueur, vue dans la deuxième section.

Dans tout l'énoncé, un même identificateur écrit dans deux polices de caractères différentes désigne la même entité, mais du point de vue mathématique pour la police en italique (par exemple n) et du point de vue informatique pour celle en romain avec espacement fixe (par exemple `n`).

Travail attendu

Pour répondre à une question, il est permis de réutiliser le résultat d'une question précédente, même sans avoir réussi à établir ce résultat.

Selon les consignes, il faudra coder des fonctions à l'aide du langage de programmation C exclusivement, en reprenant le prototype de fonction fourni par le sujet, ou en pseudo-code (c.-à-d. dans une syntaxe souple mais conforme aux possibilités offertes par le langage C). Inclure les entêtes tels que `<assert.h>`, `<stdbool.h>`, etc., n'est pas demandé.

Quand l'énoncé demande de coder une fonction, sauf indication explicite, il n'est pas nécessaire de justifier que celle-ci est correcte ou de tester que des préconditions sont satisfaites.

Le barème tient compte de la clarté et de la concision des programmes : nous recommandons de choisir des noms de variables intelligibles ou encore de structurer de longs codes par des blocs ou par des fonctions auxiliaires dont on décrira le rôle.

1 Le jeu de Shannon

1.1 Implémentation du jeu en langage C

Un graphe non orienté et connexe $G = (V, E)$ est fixé, ainsi que deux sommets particuliers : le sommet s et le sommet t. La lettre n désigne le nombre de sommets.

Définition : Le *jeu de Shannon* sur le graphe G est un jeu qui se joue entre deux joueurs, *Positus* et *Minus*, qui agissent à tour de rôle. Chaque coup consiste à marquer une arête qui n'a pas été marquée précédemment. Dans ce jeu, *Positus* remporte la victoire s'il parvient à marquer un ensemble d'arêtes dans lequel les sommets s et t sont reliés par un chemin (appelé aussi une *chaîne* d'arêtes). *Minus* gagne le jeu si toutes les arêtes ont été marquées sans que *Positus* ne soit parvenu à relier les sommets s et t.

En pratique, *Positus* « verrouille » des arêtes tandis que *Minus* « supprime » des arêtes encore jouables.

Indication C : Dans l'ensemble du sujet, les sommets du graphe $G = (V, E)$ sont numérotés entre 0 et $n-1$, $S = \{0, 1, 2, \ldots, n-1\}$. Par convention, les sommets s et t correspondent aux sommets de numéros 0 et 1. Les arêtes des graphes sont de la forme $\{u, v\}$ et donc E est un sous-ensemble de $P_2(S) = \{X \subset S \mid |X| = 2\}$. Nous notons $M \subseteq E$ l'ensemble des arêtes marquées. Nous introduisons les déclarations suivantes.

```
1.  #define s 0
2.  #define t 1
3.
4.  struct link {
5.      int node;
6.      struct link * next;
7.  };
8.  typedef struct link link_t;
9.
10. struct game {
11.     int n;              // Nombre de sommets
12.     int nm;             // Nombre d'aretes non marquees
13.     link_t ** adj_arr; // Voisins non marques
14.     int * cmps;         // Composantes connexes creees par Positus
15. };
16. typedef struct game game_t;
17.
18. game_t gm;
```

Le type `link_t` permet de représenter des maillons de listes simplement chaînées. Le graphe $(V, E \setminus M)$ est représenté par un tableau de listes d'adjacences. Le type `game_t` permet de représenter un jeu de Shannon en cours de déroulement. Dans les enregistrements de ce type,

— le champ `n` désigne le nombre de sommets,

— le champ `nm` désigne le nombre d'arêtes non marquées,

— le champ `adj_arr` renvoie à un tableau de pointeurs où, pour tout entier u compris entre 0 et $n-1$, le pointeur `adj_array[u]` désigne le premier maillon d'une liste simplement chaînée contenant tous les voisins du sommet u dans le graphe $(V, E \setminus M)$,

— le champ `cmps` renvoie à un tableau permettant de représenter les sommets reliés entre eux par des arêtes marquées par *Positus* selon une convention détaillée aux questions 6 et 7.

Une variable globale `gm` est utilisée dans tout le sujet pour désigner le jeu en cours. On notera que les arêtes marquées au cours du jeu sont perdues et ne figurent plus dans la variable `gm`.

☐ 1 – Écrire une fonction C `void gm_init(int n)` qui initialise la variable globale `gm` afin qu'elle contienne un jeu sur le graphe à n sommets et zéro arêtes. Il est attendu que le tableau `gm.cmps` contienne initialement les entiers 0, 1, …, $n-1$, dans cet ordre.

☐ 2 – Écrire une fonction C `void gm_addedge(int u, int v)` dont l'effet est d'ajouter une arête entre les sommets u et v au graphe stocké dans la variable globale `gm`. On veillera à représenter une arête par deux arcs orientés (u, v) et (v, u).

☐ 3 – Écrire une fonction C `bool gm_remove(int u, int v)` dont l'effet est de retirer une arête entre les sommets u et v au graphe `gm` et dont la valeur de retour est `true` si la suppression s'est bien déroulée, et `false` sinon.

Nous rappelons que la structure « unir & trouver » (ou, en anglais, *union find*) permet de représenter les classes d'équivalence d'une relation d'équivalence pour un ensemble E. On choisit d'implémenter cette structure à l'aide d'un tableau associatif A, de taille n pour un ensemble E de cardinal n dont les éléments sont numérotés de 0 à $n-1$. Le tableau rend compte d'une forêt où chaque arbre est orienté des feuilles vers la racine et représente une classe d'équivalence. Pour un élément de numéro u, $A[u]$ vaut u s'il s'agit de la racine, que l'on appellera **identifiant** de la classe, ou $A[u]$ vaut le numéro de son parent dans l'arbre dans tout autre cas.

☐ 4 – Écrire une fonction C `int uf_find(int n, int * A, int u)` dont la valeur de retour est l'identifiant de la classe à laquelle appartient l'élément u dans la relation représentée par A. Le tableau A ne doit pas être modifié.

☐ 5 – Écrire une fonction C `bool uf_unite(int n, int * A, int u, int v)` dont l'effet est de réunir les classes des éléments u et v dans la relation d'équivalence représentée par A. Il n'y a pas d'effet si les éléments u et v appartiennent déjà à la même classe. La valeur de retour est `true` si A a été modifié et `false` sinon.

Nous appelons $M^+ \subseteq E$ l'ensemble des arêtes marquées par le joueur *Positus*. Nous notons $\leftrightsquigarrow_P$ la relation d'équivalence dans laquelle deux sommets u et v sont en relation si et seulement s'il existe une chaîne entre u et v formée d'arêtes de M^+. Les classes d'équivalence sont ainsi les composantes connexes du graphe (V, M^+). Nous utilisons le champ `cmps` de la variable `gm` afin de stocker cette relation d'équivalence selon une structure « unir & trouver ».

☐ 6 – Écrire une fonction C `int gm_find(int u)` dont la valeur de retour est l'identifiant de la composante connexe à laquelle appartient le sommet u dans le graphe (V, M^+).

☐ 7 – Écrire une fonction C `void gm_unite(int u, int v)` dont l'effet est de réunir les composantes connexes des sommets u et v.

☐ 8 – Écrire une fonction C `bool has_P_won(void)` dont la valeur de retour indique si le jeu est dans un état gagnant pour le joueur *Positus*.

☐ 9 – Écrire une fonction C `bool has_P_lost(void)` dont la valeur de retour indique si le jeu est dans un état perdant pour le joueur *Positus*.

☐ 10 – Écrire une fonction C `void play_P(int u, int v)` dont l'effet est de transformer la variable `gm` en jouant l'arête entre les sommets u et v par le joueur *Positus*. On se défendra, à l'aide d'une assertion, du cas où le coup joué par *Positus* est illicite.

☐ 11 – Écrire une fonction C `void play_M(int u, int v)` dont l'effet est de transformer la variable `gm` en jouant l'arête entre les sommets u et v par le joueur *Minus*. On se défendra, à l'aide d'une assertion, du cas où le coup joué par *Minus* est illicite.

1.2 Étude de tournois en langage SQL

Un *tournoi* de jeux de Shannon est une compétition dans laquelle s'affrontent exactement deux joueurs (A et B) au cours de plusieurs matchs. Le vainqueur d'un tournoi est le joueur qui a gagné le plus grand nombre de matchs. Le rôle de chaque joueur (*Positus* ou *Minus*) peut changer entre chaque match.

On dispose d'une base de donnée SQL composée de trois tables (les clés primaires sont soulignées)

— Joueurs(JoueurID, NomJ)

— Tournois(TournoiID, NomT, JoueurAID, JoueurBID)

— Matchs(MatchID, TournoiID, RoleJoueurA, VainqueurID)

dont voici certains extraits :

JoueurID	NomJ
2	Alice
8	Bob
7	Charlie

TournoiID	NomT	JoueurAID	JoueurBID
82	TournoiDu12Dec	2	8
59	TournoiDu17Fev	7	8
61	TournoiDu29Mai	8	2

MatchID	TournoiID	RoleJoueurA	VainqueurID
119	82	Positus	2
934	82	Minus	8
925	82	Minus	2
384	61	Positus	2

Par exemple, le tournoi du 12 décembre a opposé Alice et Bob, il s'est joué en 3 matchs. Alice, qui a gagné deux des trois matchs, a été victorieuse.

☐ 12 – Écrire une requête SQL renvoyant le nombre de matchs joués dans chaque tournoi.

☐ 13 – Écrire une requête SQL renvoyant le nombre de tournois gagnés par chaque joueur.

☐ 14 – Écrire une requête SQL renvoyant le nombre de fois où chaque joueur a joué le rôle de *Positus*.

2 Stratégies de jeu

2.1 Stratégies gagnantes

☐ 15 – Démontrer, à l'aide d'un variant, que le jeu se termine toujours et justifier qu'il ne peut pas se terminer par un match nul.

☐ 16 – Rappeler la définition du terme *stratégie gagnante*.

☐ 17 – Nous supposons que *Positus* dispose d'une stratégie gagnante s'il joue en second. Démontrer qu'il dispose aussi d'une stratégie gagnante s'il joue en premier.

☐ 18 – Démontrer qu'il existe trois types d'instances du jeu de Shannon :

- un type d'instance dans laquelle *Positus* dispose d'une stratégie gagnante qu'il soit premier ou second joueur,
- un type dans laquelle *Minus* dispose d'une stratégie gagnante qu'il soit premier ou second joueur,
- et enfin un type dans laquelle le joueur qui commence la partie dispose d'une stratégie gagnante.

On donnera un exemple de graphe G illustrant chacun des trois types.

2.2 Une stratégie pour *Positus*

Définition : Selon la terminologie usuelle, nous appelons *arbre couvrant* tout sous-ensemble de $n-1$ arêtes $T \subseteq E$ ne contenant pas de cycle. Il est rappelé les faits suivants :

- le graphe (V, T) est connexe ;
- tout ajout d'une arête $e \in E \setminus T$, à un arbre couvrant T, crée un unique cycle noté $C(T, e)$;
- si T est un arbre couvrant, alors, pour tout couple de sommets u et v de V, il existe un unique chemin simple reliant u et v avec des arêtes de T.

☐ 19 – Soient deux arbres couvrants $T \subseteq E$ et $T' \subseteq E$ et une arête α appartenant à T. Montrer qu'il existe une arête β appartenant à T' telle que l'ensemble $T'' = T \cup \{\beta\} \setminus \{\alpha\}$ est encore un arbre couvrant.

Indication C : Nous représentons tout arbre couvrant par un tableau `T` de taille n tel que `T[0]` vaut 0 et, pour tout autre sommet u, `T[u]` vaut le parent du sommet u dans l'arbre enraciné en 0.

Pour représenter un arc, nous déclarons :

```
19.  struct pair {
20.      int first;
21.      int second;
22.  };
23.  typedef struct pair pair_t;
```

☐ 20 – Écrire une fonction C `pair_t edge_find(int n, int * T, int * Tprime, pair_t alpha)` ainsi spécifiée :
Précondition : Les variables `T` et `Tprime` contiennent des arbres couvrants T et T' sur les mêmes n sommets. La variable α est un arc de l'arbre T orienté des feuilles vers la racine.
Valeur de retour : Une arête β, appartenant à l'arbre T', telle que $T'' = T \cup \{\beta\} \setminus \{\alpha\}$ est encore un arbre couvrant.

☐ 21 – Écrire une fonction C `void edge_swap(int n, int * T, pair_t alpha, pair_t beta)` ainsi spécifiée :
Précondition : La variable `T` correspond à un arbre couvrant T à n sommets. La variable α est un arc de l'arbre T orienté des feuilles vers la racine. La variable β est une arête qui n'appartient pas à l'arbre T telle que $T'' = T \cup \{\beta\} \setminus \{\alpha\}$ est un arbre couvrant.
Effet : Le tableau `T` est transformé afin de contenir l'arbre T''.

Dans ce qui suit, nous supposons que *Minus* joue en premier.

Définition : Pour tout ensemble de sommets $U \subseteq V$ qui contient les sommets s et t, nous notons G_U le *graphe induit par* U, c'est-à-dire le sous-graphe (U, E_U) où,

$$E_U = \{\{u, v\} \in E \mid u, v \in U\}.$$

Pour tout entier naturel k, nous notons $M_k^+ \subseteq E$ l'ensemble des k arêtes marquées par *Positus*, après que k coups ont été joués par *Positus*. De même, nous notons $M_k^- \subseteq E$ l'ensemble des k arêtes marquées par *Minus* après que k coups ont étés joués par *Minus*. Enfin, nous appelons $(✠_k)$ l'assertion suivante :

> Après que $2k$ coups ont été joués par *Minus* et *Positus*, il existe un ensemble de sommets U contenant les sommets s et t ainsi que deux arbres couvrant le graphe induit G_U, que l'on note $T_k \subseteq E_U$ et $T'_k \subseteq E_U$, tels que l'on a $T_k \cap T'_k = M_k^+$ et $(T_k \cup T'_k) \cap M_k^- = \varnothing$. $(✠_k)$

☐ 22 – Soit k un entier naturel. Dans cette question, nous supposons que $2k$ coups ont d'ores et déjà étés joués et que l'assertion $(✠_k)$ est vérifiée ; pour son $(k+1)^{\text{e}}$ coup, le joueur *Minus* choisit de marquer une arête appartenant à l'arbre T_k. Montrer que le joueur *Positus* peut alors trouver une arête à marquer telle que l'assertion $(✠_{k+1})$ est vérifiée.

☐ 23 – Soit k un entier naturel. Dans cette question, nous supposons que $2k$ coups ont étés joués et que l'assertion $(✠_k)$ est vérifiée ; le joueur *Minus* marque une arête n'appartenant ni à l'arbre T_k, ni à l'arbre T'_k. Montrer que le joueur *Positus* peut marquer une arête telle que l'assertion $(✠_{k+1})$ est vérifiée.

☐ 24 – Démonter que, si au cours de la partie, il existe un entier naturel k_0 tel que l'assertion $(✠_{k_0})$ est satisfaite, alors *Positus* possède une stratégie gagnante.

☐ 25 – Dans cette question, nous supposons que la paire $\{s,t\}$ n'appartient pas à E et nous notons $\hat{G}$ le graphe obtenu à partir de G en ajoutant une arête supplémentaire entre les sommets s et t. Nous supposons que l'assertion $(✠_0)$ est vraie pour le nouveau graphe $\hat{G}$. Démontrer que le joueur *Positus* dispose d'une stratégie gagnante lorsqu'il joue en premier.

3 Arbres couvrants disjoints

Dans toute cette section, nous supposons que la paire $\{s,t\}$ n'appartient pas à l'ensemble des arêtes E. Nous notons $\hat{G}$ le graphe obtenu à partir de G en ajoutant une arête supplémentaire entre les sommets s et t. Nous appelons $\hat{E}$ l'ensemble d'arêtes $E \cup \{s,t\}$.

3.1 Une condition suffisante pour $(✠_0)$

Soient $T \subseteq \hat{E}$ et $T' \subseteq \hat{E}$ deux arbres couvrants de $\hat{G}$ et α_0 une arrête n'appartenant pas à $T \cup T'$. Notons $L_0 = \{\alpha_0\}$. Notons $L_1 \subseteq \hat{E}$ l'unique cycle $C(T, \alpha_0)$ présent dans l'ensemble $T \cup \{\alpha_0\}$.

Définition : Pour tout couple d'arbres couvrants, $T \subseteq \hat{E}$ et $T' \subseteq \hat{E}$, notons $\delta(T,T')$ le cardinal de l'intersection $T \cap T'$. Deux arbres couvrants T_0 et T_0' sont *éloignés* si le couple (T_0, T_0') atteint le minimum global de la fonction δ parmi tous les couples d'arbres couvrants de $\hat{G}$.

Définition : Une arête $\alpha \in \hat{E}$ est *principale* s'il existe un couple d'arbres couvrants T et T' éloignés, tel que l'arête α n'appartient pas à $T \cup T'$.

☐ 26 – En supposant qu'il existe une arête $\alpha_1 \in L_1$ appartenant à $T \cap T'$, construire deux arbres couvrants $\tilde{T}$ et $\tilde{T}'$ du graphe $\hat{G}$ tels que

$$\delta(\tilde{T}, \tilde{T}') < \delta(T, T').$$

Si L_1 ne contient aucune arête de $T \cap T'$, nous appelons $L_2 \subseteq \hat{E}$ la réunion des cycles $\bigcup_{\alpha \in L_1} C(T', \alpha)$ qui se créent lorsque l'on ajoute à T' l'une des arêtes de L_1.

☐ 27 – En supposant qu'il existe une arête $\alpha_2 \in L_2$ appartenant à $T \cap T'$, construire deux arbres couvrants $\tilde{T}$ et $\tilde{T}'$ du graphe $\hat{G}$ tels que

$$\delta(\tilde{T}, \tilde{T}') < \delta(T, T').$$

Plus généralement, nous construisons par récurrence la suite croissante $(L_k)_{k \in \mathbb{N}} \subseteq \hat{E}$ d'ensembles d'arêtes où

— pour tout entier $k \geqslant 1$, si l'ensemble L_{2k} ne contient aucune arête de $T \cap T'$, alors on pose

$$L_{2k+1} = \bigcup_{\alpha \in L_{2k}} C(T, \alpha),$$

— pour tout entier $k \geqslant 0$, si l'ensemble L_{2k+1} ne contient aucune arête de $T \cap T'$, alors on pose

$$L_{2k+2} = \bigcup_{\alpha \in L_{2k+1}} C(T', \alpha).$$

☐ 28 – Démontrer que l'un ou l'autre des cas suivants se produit :

— la suite $(L_k)_{k\in\mathbb{N}} \subseteq E^{\mathbb{N}}$ n'est définie que pour un nombre fini de termes et dans ce cas montrer qu'il existe deux arbres couvrants $\tilde{T}$ et $\tilde{T}'$ du graphe $\hat{G}$ tels que

$$\delta(\tilde{T}, \tilde{T}') < \delta(T, T') \ ;$$

— ou bien la suite $(L_k)_{k\in\mathbb{N}} \subseteq E$ est définie pour tout rang k et donc qu'il existe un rang à partir duquel la suite est égale à une constante $\Lambda(\alpha_0) \subseteq \hat{E}$.

☐ 29 – Démontrer que, si l'arête $\{s, t\}$ est une arête principale du graphe $\hat{G}$, alors l'assertion ($\maltese_0$) est vérifiée.

3.2 Recherche des arêtes principales

☐ 30 – Nommer un algorithme qui construit un arbre couvrant pour le graphe G et qui soit cohérent avec les choix opérés par le type `game_t` de représenter un graphe par des listes d'adjacence.

Soit T et T' un couple d'arbres couvrants éloignés. On pose

$$P = \bigcup_{\alpha_0 \in E \setminus (T \cup T')} \Lambda(\alpha_0).$$

On appelle U l'ensemble des arêtes incidentes à P. On appelle n_U le cardinal de U, m_P le cardinal de P, et $c = |E \setminus (T \cup T')|$ le nombre d'arêtes hors de T et T'. On note κ le nombre de composantes connexes de (U, P).

Soit $\tilde{T}$ et $\tilde{T}'$ un autre couple d'arbres couvrants éloignés.

Notons $\tilde{c}_1$ le cardinal de $(E \setminus (\tilde{T} \cup \tilde{T}')) \cap P$.

☐ 31 – Montrer que

$$c \geqslant \tilde{c}_1 \geqslant m_P - 2(n_U - \kappa).$$

En déduire que le cas 2 de la question 28 se produit pour toute arête α_0 uniquement quand T et T' sont éloignés.

☐ 32 – D'après la question 28, pour toute arête α_0 n'appartenant pas à $T \cup T'$, l'ensemble $\Lambda(\alpha_0) \subseteq \hat{E}$ est bien défini. Démontrer que P est l'ensemble des arêtes principales.

☐ 33 – Expliquer comment on peut construire l'ensemble des arêtes principales et en déduire une stratégie gagnante pour un joueur. Une réponse en pseudo-code est permise, mais seule la logique de programmation sera évaluée.

Fin de l'épreuve

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MPI chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Langage C : types, tableaux, pointeurs, allocation
- Spécifier, tester, prouver : terminaison et correction

Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière : **écrire sur WhatsApp**

Les autres années et épreuves : prepahelp.com/annales/mines/mpi