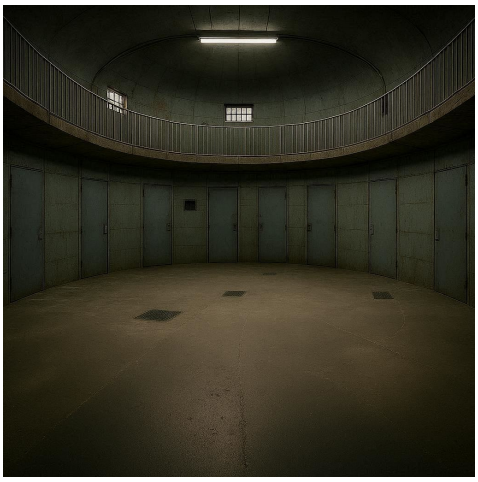
La prison circulaire de taille inconnue



Énoncé

Vous avez été enfermés dans une prison formant un anneau parfait de n cellules (vous ne connaissez pas n), toutes identiques, isolées, et chacune contenant exactement un prisonnier. Dans chaque cellule se trouvent un interrupteur et une ampoule, mais le câblage a été manifestement conçu par un ingénieur fou. Si, à midi pile, l'interrupteur d'une cellule est en position « on », alors l'ampoule de la cellule voisine (dans le sens horaire) émet un bref flash. Sinon, la lampe reste éteinte.
Afin d'empêcher toute forme de communication, chaque nuit à minuit, le gardien diffuse du gaz anesthésiant dans tout l'anneau, remet tous les interrupteurs sur « off », et réarrange les prisonniers comme bon lui semble. Mais il garde une règle immuable : une personne par cellule.
Un jour, le gardien entre dans votre cellule et vous lance un défi. Pour gagner votre liberté — et celle de tous les autres — une seule règle : à n'importe quel moment, n'importe quel prisonnier peut crier « Nous sommes n prisonniers ! » S'il dit vrai : tout le monde est libre. S'il se trompe : tout le monde est exécuté.
Le gardien vous autorise à envoyer un unique message écrit à tous les autres prisonniers, dans lequel vous pouvez expliquer les règles et proposer un plan. Eux n'ont pas le droit de répondre. Naturellement, le gardien lira soigneusement votre note... et ensuite, il mélangera les prisonniers autant que nécessaire pour tenter de faire échouer votre stratégie.

Questions :

1. 🌶[2.2] On admet que les prisonniers peuvent lancer des pièces, ce qui leur permet de prendre des décisions aléatoires que le gardien ne peut pas prévoir, même s'il connaît leur stratégie. Décrivez une stratégie permettant aux prisonniers de déterminer n (presque sûrement).
2. 🌶[2.9] Donnez une solution qui n'utilise aucune forme d'aléatoire et qui garantit d'aboutir en un nombre déterminé de jours.
3. 🌶[3.0] On suppose que la prison n'est plus circulaire, mais forme un graphe orienté fortement connexe, chaque cellule contenant au moins un interrupteur et une lumière, et chaque lumière étant contrôlée par exactement un interrupteur situé dans une autre cellule. Décrivez une stratégie permettant aux prisonniers de déterminer n.

Solution

1. Il est crucial que vous soyez désigné(e) par le gardien, car cela rompt la symétrie entre vous et les autres, vous permettant ainsi d'agir comme le/la capitaine du groupe.

   Lemme (Recherche d'une borne supérieure). *Nous effectuons une suite de tours (tour 1, tour 2, etc.), chaque tour étant composé d'une phase de* croissance *et d'une phase de* décroissance*. Au sein d'un tour, les prisonniers sont soit* actifs *(ils actionneront leur interrupteur ce jour-là), soit* inactifs *(ils ne l'actionneront pas).*
   La phase de croissance du tour k dure k jours :

— *Au début de cette phase, le capitaine commence actif, et tous les autres commencent inactifs.*
— *Pendant cette phase, tout prisonnier qui voit un flash de lumière devient actif, et le reste pour toute la durée de la phase.*

La phase de décroissance du tour k dure 2^k jours :
— *Au début de cette phase, les prisonniers conservent leur statut (actif ou inactif) obtenu à la fin de la phase de croissance.*
— *Si un prisonnier ne voit aucun flash de lumière, il devient immédiatement inactif et le reste pour toute la phase.*

À la fin d'un tour k, soit tous les prisonniers sont actifs, soit tous sont inactifs. Si tous sont actifs, alors $n \leq 2^k$.

Démonstration. Considérons d'abord le nombre de prisonniers actifs à la fin de la phase de croissance. Chaque prisonnier actif ne peut activer qu'un seul autre prisonnier par jour ; ainsi, le nombre de prisonniers actifs peut au plus doubler chaque jour. Comme la phase de croissance dure k jours, on en déduit qu'il y a au plus 2^k prisonniers actifs à son terme.

Examinons maintenant la phase de décroissance.

Cas 1 : tous les prisonniers sont actifs à la fin de la phase de croissance (dans ce cas, on obtient immédiatement l'inégalité $n \leq 2^k$). Alors chacun actionne son interrupteur chaque jour ; personne ne devient inactif, et le tour se termine avec tout le monde encore actif.

Cas 2 : au moins un prisonnier est inactif à la fin de la phase de croissance. Alors il existe une frontière entre un prisonnier actif et un prisonnier inactif. Chaque jour, cette frontière fait décroître le nombre de prisonniers actifs d'au moins un. Puisqu'il y avait initialement au plus 2^k prisonniers actifs, après 2^k jours tous les prisonniers seront devenus inactifs.

Ainsi, dans tous les cas, à la fin du tour k, l'ensemble des prisonniers est soit entièrement actif, soit entièrement inactif. □

Remarque. *Ce processus termine nécessairement : lors du k-ième tour, il existe au moins k prisonniers actifs à la fin de la phase de croissance, et donc, au plus, le processus s'achève au tour n.*

Le mécanisme fondamental des solutions repose sur ce que nous appellerons une *annonce*.

Définition (Annonce). *Pour un prédicat P, une annonce pour P est une procédure qui rend publique et connue de tous l'information suivante : existe-t-il au moins un prisonnier satisfaisant P ?*

La période d'annonce dure B jours, où B est une borne supérieure sur le nombre de prisonniers.
— *Tout prisonnier qui satisfait P reste toujours actif (il actionne toujours son interrupteur).*
— *Les autres prisonniers deviennent actifs dès qu'ils voient un flash, et restent actifs par la suite.*

À la fin de la période d'annonce, si au moins un prisonnier satisfait P, alors tous les prisonniers sont actifs ; sinon, tous sont inactifs. Ainsi, le fait qu'un prisonnier satisfasse ou non P devient une information publique et connue de tous.

Stratégie s'appuyant sur l'aléatoire. Le but est d'assigner à chaque prisonnier un numéro de 1 à n. À chaque instant, les numéros 1 à d sont déjà attribués, et d est connu de tous. Le capitaine commence avec le numéro 1, et tous les autres prisonniers sont initialement non numérotés (c'est-à-dire $d = 1$). On répète la séquence suivante :

(a) **Phase d'annonce (B jours) :** Effectuer une annonce pour « est non numéroté ».
   — Si le résultat est négatif, alors tous les prisonniers sont numérotés et chacun connaît n ; la procédure s'arrête.
   — Sinon, poursuivre.

(b) **Jour de sélection des candidats (1 jour) :** Ce jour est crucial.
   — Chaque prisonnier numéroté lance simultanément une pièce.
   — S'il obtient pile, il actionne son interrupteur aujourd'hui ; sinon, il ne le fait pas.
   — Les prisonniers non numérotés ne touchent pas à leur interrupteur.
   — Tout prisonnier qui voit un flash aujourd'hui est appelé *candidat*.

(c) **Phase de comptage des candidats ($d \cdot B$ jours) :** Pour chaque i de 1 à d, le prisonnier portant le numéro i effectue une annonce pour « a tiré pile ». Après cette phase, tous les prisonniers savent exactement combien de piles ont été obtenues, c'est-à-dire combien de candidats il y a.

(d) **Annonce finale (B jours) :** Une annonce est faite pour « est candidat non numéroté ».
- Si cette annonce est positive et qu'un seul « pile » a été tirée, alors il existe un unique candidat non numéroté.
- Ce candidat s'attribue le numéro $d + 1$, et tous les prisonniers incrémentent d.

Répété suffisamment de fois, ce procédé termine presque sûrement. Par exemple, pour numéroter le dernier prisonnier, tous les prisonniers numérotés doivent tirer face, sauf celui adjacent au prisonnier non numéroté. Cela est peu probable mais finit par se produire. La même logique s'applique à toutes les étapes précédentes pour numéroter un prisonnier.

Le gardien peut réarranger les prisonniers comme il le souhaite, mais comme chaque prisonnier numéroté est capable de nommer un candidat, ses actions ne peuvent pas empêcher la procédure de réussir. Sans recours à l'aléatoire, le gardien peut anticiper quel prisonnier tentera de nommer un nouveau candidat et agir pour l'en empêcher.

On peut utiliser le code Python suivant pour simuler la stratégie.

python ⬇️ Télécharger le code

```python
import random

def UpperBound(p, l):
    p.counter += 1
    if p.counter <= p.r:
        p.is_active = l or p.is_active
        return f"[Upper Bound] ({p.counter}/{p.r})"
    else:
        p.is_active = l and p.is_active
        B = 2**p.r
        if p.counter == B + p.r:
            if p.is_active:
                p.phase = 'Next'
                p.B = B
                return f"[Upper Bound] ({B}/{B}) B = {p.B}"
            p.counter = 0
            p.is_active = p.is_captain
            p.r += 1
            return f"[Upper Bound] ({B}/{B}) repeat with r = {p.r}"
        return f"[Upper Bound] ({p.counter-p.r}/{B})"

def PrepareAnnouncement(p, predicate, name):
    p.phase = name
    p.is_active = predicate
    p.counter = 0

def Announcement(p, l, name, name_next):
    p.counter += 1
    p.is_active = l or p.is_active
    if p.counter == p.B:
        p.phase = name_next
        return f"[{name}] ({p.counter}/{p.B}) Announcement = {p.is_active}",
            p.is_active
    return f"[{name}] ({p.counter}/{p.B})", None

class Prisoner:
    def __init__(self, is_captain=False):
        self.is_captain = is_captain
        self.is_active = is_captain
        self.phase = 'Upper Bound'
```

```
        self.r, self.B = 1, None
        self.counter = 0
        self.number = 1 if is_captain else None
        self.is_candidate = False
        self.n_candidates = 0
        self.i, self.d = 1, 1

    def log(self, msg, day):
        if self.is_captain:
            print(f'[{day}]'+msg)

    def turn_on(self):
        return self.is_active

    def see_light(self, l, day):

        # Execute current phase using today's light input

        current_phase = self.phase
        if current_phase == 'Upper Bound':
            msg = UpperBound(self, l)
            self.log(msg, day)

        if current_phase == 'Unnumbered Announcement':
            msg, answer = Announcement(self, l, 'Unnumbered Announcement', '
                Prepare Candidate Selection Day')
            if answer is not None:
                if not answer:
                    self.log(f"[End] everyone is numbered, and everyone
                        knows n={self.d}", day)
                    return True
            self.log(msg, day)

        if current_phase == 'Candidate Selection Day':
            self.log(f"[{current_phase}]", day)
            self.is_candidate = l
            self.phase = 'Prepare Candidate Announcement'

        if current_phase == 'Candidate Announcement':
            msg, answer = Announcement(self, l, 'Candidate Announcement', '
                Prepare Candidate Announcement')
            if answer is not None:
                self.i += 1
                if answer:
                    self.n_candidates += 1
                self.log(msg + f', n_candidates = {self.n_candidates}', day)
                if self.i > self.d:
                    self.phase = 'Prepare Unnumbered Candidate Announcement'
            else:
                self.log(msg, day)

        if current_phase == 'Unnumbered Candidate Announcement':
            msg, answer = Announcement(self, l, 'Unnumbered Candidate
                Announcement', 'Next')
            if answer is not None:
                if answer:
                    if self.n_candidates == 1:
                        self.d += 1
                        if self.is_candidate:
                            self.number = self.d
                    self.log(msg + f', d = {self.d}', day)
                else:
```

```
                    self.log(msg + ', retry', day)
                self.i = 1
                self.n_candidates = 0
            else:
                self.log(msg, day)

        # Preparation for tomorrow

        if self.phase == 'Prepare Candidate Selection Day':
            self.is_active = False
            self.flip = False
            if self.number:
                self.flip = random.random() < .5
                self.is_active = self.flip
            self.phase = 'Candidate Selection Day'
        if self.phase == 'Prepare Unnumbered Candidate Announcement':
            PrepareAnnouncement(self, self.is_candidate & (self.number is
                None), 'Unnumbered Candidate Announcement')
        if self.phase == 'Prepare Candidate Announcement':
            PrepareAnnouncement(self, (self.number == self.i) & self.flip, '
                Candidate Announcement')
        if self.phase == 'Next':
            PrepareAnnouncement(self, self.number is None, 'Unnumbered
                Announcement')

def simulate_prisoners(n):
    prisoners = [Prisoner(i==0) for i in range(n)]

    day = 0
    while True:
        day += 1
        prisoners_shuffled = random.sample(prisoners, len(prisoners))

        lights = []
        for i in range(n):
            lights.append(prisoners_shuffled[i].turn_on())
        for i in range(n):
            stop = prisoners_shuffled[(i+1) % n].see_light(lights[i], day)
        if stop:
            break

if __name__ == "__main__":
    simulate_prisoners(5)
```

2. On commence par établir une borne supérieure B sur n (comme pour la première question). Ensuite on travaille par partitions itératives des prisonniers. À tout instant, on dispose d'une partition connue $\mathcal{S} = (S_1, \ldots, S_k)$ telle que chaque prisonnier connaît son indice j (prisonnier $\in S_j$) et tous connaissent k. Initialement $S_1 = \{\text{capitaine}\}$ et $S_2 = \{\text{autres}\}$.

   La procédure de raffinement consiste à « couper » les S_j :

   (a) Pour chaque sous-ensemble $I \subset \{1, \ldots, k\}$ non trivial ($I \neq \varnothing$ et $I \neq \{1, \ldots, k\}$), tous les prisonniers de $\bigcup_{i \in I} S_i$ allument leur lumière pendant 1 jour. Notons T l'ensemble des prisonniers ayant vu la lumière.

   (b) Pour chaque $i = 1, \ldots, k$, on effectue deux annonces de B jours chacune :
   — annonce pour la propriété $S_i \cap T \neq \varnothing$;
   — annonce pour la propriété $S_i \setminus T \neq \varnothing$.

   (c) Si S_i est coupé ($S_i \cap T \neq \varnothing$ et $S_i \setminus T \neq \varnothing$), on remplace S_i par $S_i \cap T$ et on ajoute $S_i \setminus T$ comme nouvelle classe. On interrompt et recommence la procédure avec la nouvelle partition.

   (d) Si aucune classe n'est coupée, l'état atteint permet de déduire les tailles des classes.

   Pour montrer ce dernier point, fixons l'un des sous-ensembles I. Comme k n'a pas augmenté lors de la dernière tentative, nous savons que T n'a pas pu découper l'un des S_i cette fois-ci.

Autrement dit, pour tout i, soit tous les prisonniers de S_i ont vu une lumière, soit aucun n'en a vu.
Notons I' l'ensemble des indices j tels que les prisonniers de S_j ont vu une lumière. Comme le nombre de prisonniers voyant une lumière doit être égal au nombre de prisonniers ayant actionné l'interrupteur, nous obtenons l'équation :

$$|T| = \sum_{i \in I} |S_i| \;=\; \sum_{j \in I'} |S_j|.$$

Remarquons que I' ne peut pas être un sous-ensemble de I. En effet, la structure circulaire de la prison implique que, sauf lorsque I est l'ensemble total ou l'ensemble vide, il doit exister au moins un prisonnier hors de I qui voit une lumière, ce qui exclut de tels cas dans la procédure.
En posant $x_i = |S_i|$, nous obtenons ainsi un système d'équations portant sur les variables $x_1, \ldots, x_k$. Fait important : les prisonniers connaissent également ces équations. Grâce aux annonces effectuées lors des tentatives de découpage par T, ils savent si $S_j \cap T$ ou $S_j \setminus T$ est vide, ce qui leur permet de déterminer I' (ils connaissent aussi I, puisqu'ils s'accordent tous sur l'ordre d'itération).
Enfin, ils savent que S_1 contient uniquement le capitaine, et donc

$$x_1 = 1.$$

Lemme. *Sous ces contraintes, il existe exactement une unique solution. Une fois celle-ci déterminée, les prisonniers connaissent alors la taille exacte de chaque sous-ensemble. Ils peuvent donc gagner en déduisant simplement que*

$$n = \sum_i x_i.$$

Démonstration. Soient $x_1, \ldots, x_k$ et $y_1, \ldots, y_k$ deux solutions satisfaisant les contraintes. Posons

$$r \;=\; \min_i \frac{y_i}{x_i}, \qquad z_i \;=\; y_i - r x_i.$$

Observons les trois faits suivants :
— Chaque $z_i \geq 0$, puisque $r \leq \frac{y_i}{x_i}$.
— Il existe au moins un indice i tel que $z_i = 0$, car $r = \frac{y_i}{x_i}$ pour au moins un i.
— Les z_i constituent également une solution, puisqu'ils sont une combinaison linéaire de solutions.
Supposons par contradiction que certains z_i soient non nuls. Soit

$$Z = \{\, i \mid z_i = 0 \,\},$$

qui n'est pas, par hypothèse, l'ensemble complet des indices. Considérons maintenant l'équation associée au sous-ensemble Z. Il existe un sous-ensemble $Z' \not\subseteq Z$ tel que

$$\sum_{i \in Z} z_i \;=\; \sum_{j \in Z'} z_j.$$

Le membre de gauche vaut zéro par définition de Z. Le membre de droite comporte au moins un terme $z_j > 0$ puisque $Z' \setminus Z$ est non vide et que tous les z_j sont positifs ou nuls. On obtient donc une contradiction. Ainsi, tous les z_i sont nuls, ce qui implique

$$y_i = r x_i \qquad \text{pour tout } i.$$

Comme $x_1 = y_1 = 1$, on en déduit $r = 1$, puis $y_i = x_i$ pour tout i. Les deux solutions sont donc identiques. □

Complexité temporelle. Le processus doit finir par s'arrêter, car dans le pire des cas on a $k = n$ et tous les sous-ensembles sont de taille 1. On peut exécuter la procédure au plus $n - 2$ fois avant que tous les sous-ensembles ne soient réduits à des singletons. Il faudra parcourir au plus $2^n - 2$ sous-ensembles, chacun nécessitant $2B + 1 \leq 2^{n+1} + 1$ jours. On obtient alors $(n-2)\,(2^n - 2)\,(2^{n+1} + 1)$ jours.
On peut modifier la classe `Prisoner` du code Python précédent comme suit pour implémenter la stratégie.

python ⬇️ Télécharger le code

```python
import itertools
import numpy as np

def proper_subsets(k):
    S = list(range(1, k+1))
    subsets = []
    for r in range(1, k):
        for combo in itertools.combinations(S, r):
            subsets.append(list(combo))
    return subsets

class Prisoner:
    def __init__(self, is_captain=False):
        self.is_captain = is_captain
        self.is_active = is_captain
        self.phase = 'Upper Bound'
        self.r, self.B = 1, None
        self.counter = 0
        self.k = 2
        self.proper_subsets = proper_subsets(self.k)
        self.proper_subsets_index = 0
        self.Iprime = []
        self.j = 1 if is_captain else 2
        self.i = 1
        self.eqs = []

    def log(self, msg, day):
        if self.is_captain:
            print(f'[{day}]'+msg)

    def turn_on(self):
        return self.is_active

    def see_light(self, l, day):

        # Execute current phase using today's light input

        current_phase = self.phase
        if current_phase == 'Upper Bound':
            msg = UpperBound(self, l)
            self.log(msg, day)

        if current_phase == 'Union Flash':
            self.log(f"[{current_phase}], I={self.proper_subsets[self.
                proper_subsets_index]}, i={self.i}", day)
            self.is_in_T = l
            self.phase = 'Prepare First Announcement'

        if current_phase == 'First Announcement':
            msg, answer = Announcement(self, l, 'First Announcement', '
                Prepare Second Announcement')
            if answer is not None:
                self.first_announcement = answer
            self.log(msg, day)

        if current_phase == 'Second Announcement':
            msg, answer = Announcement(self, l, 'Second Announcement', 'Next
                ')
            if answer is not None:
                if self.first_announcement and answer:
```

```
                    self.k += 1
                    if (self.j == self.i) & (not self.is_in_T):
                        self.j = self.k
                    self.proper_subsets = proper_subsets(self.k)
                    self.proper_subsets_index = 0
                    self.Iprime = []
                    self.eqs = []
                    self.i = 1
                else:
                    self.phase = 'Prepare First Announcement'
                    if self.first_announcement:
                        self.Iprime.append(self.i)
                    self.i += 1
                    if self.i > self.k:
                        self.phase = 'Next'
                        self.i = 1
                        v = np.zeros(self.k)
                        v[np.array(self.proper_subsets[self.
                            proper_subsets_index]) - 1] += 1
                        v[np.array(self.Iprime) - 1] -= 1
                        self.eqs.append(v)
                        self.proper_subsets_index += 1
                        self.Iprime = []
                        if self.proper_subsets_index >= len(self.
                            proper_subsets):
                            _, S, Vt = np.linalg.svd(self.eqs)
                            null_mask = (S < 1e-10)
                            null_space = Vt[null_mask].T
                            assert null_space.shape[1] == 1
                            x = null_space[:,0]
                            self.log(f"[End] everyone knows n={round(sum(x)/
                                x[0])}", day)
                            return True
                self.log(msg+f', k={self.k}', day)
            else:
                self.log(msg, day)

        # Preparation for tomorrow

        if self.phase == 'Next':
            self.is_active = self.j in self.proper_subsets[self.
                proper_subsets_index]
            self.phase = 'Union Flash'
        if self.phase == 'Prepare First Announcement':
            PrepareAnnouncement(self, (self.j == self.i) & self.is_in_T, '
                First Announcement')
        if self.phase == 'Prepare Second Announcement':
            PrepareAnnouncement(self, (self.j == self.i) & (not self.is_in_T
                ), 'Second Announcement')
```

3. On peut établir une borne supérieure B sur n de la même manière que précédemment.

Lemme. *On peut contrôler la procédure de façon que, le jour t, chaque prisonnier ayant « déjà vu » la lumière effectue au plus $t-1$ transmissions. On note S_t le nombre de prisonniers (distincts) ayant vu la lumière à la fin du jour t (le capitaine étant considéré comme un prisonnier qui a « vu » la lumière dès le premier jour, i.e. $S_1 = 1$). Le nombre de prisonniers pouvant avoir vu la lumière après t jours de propagation est borné par $t!$.*

Démonstration. On a

$$\text{nombre de nouveaux prisonniers atteignables le jour } t \leq S_{t-1} \cdot (t-1).$$

Ainsi,

$$S_t \leq S_{t-1} + S_{t-1} \cdot (t-1) = S_{t-1} \cdot t \leq \cdots \leq t!. \qquad \square$$

Les prisonniers déterminent une borne supérieure M sur le nombre maximal d'interrupteurs et L sur le nombre maximal de lumières qu'une cellule peut contenir. Pour cela, ils effectuent, pour tout $x = 1, 2, \ldots$, des annonces successives :

"As-tu plus de x interrupteurs ?" et "As-tu plus de x ampoules ?"

Initialement, $k = 2$, $S_1 = \{\text{capitaine}\}$ et $S_2 = \{\text{autres}\}$.

(a) **Allumage massif.** Tous les prisonniers appartenant à $\bigcup_{i \in I} S_i$ allument *tous* leurs interrupteurs pendant une journée. On note $T_{m,\ell}$ l'ensemble des prisonniers ayant allumé exactement m interrupteurs et vu exactement ℓ lumières.

(b) **Annonce du nombre de lumières vues et du nombre d'interrupteurs utilisés.** Pour chaque $i = 1, \ldots, k$, $m = 1, \ldots, M$ et $\ell = 0, \ldots, L$, on effectue l'annonce :

"As-tu vu exactement ℓ lumières et allumé exactement m interrupteurs en étant dans S_i ?"

(c) **Découpage.** Si S_i est coupé pour un couple (m, ℓ) (i.e. $S_i \cap T_{m,\ell} \neq \varnothing$ et $S_i \setminus T_{m,\ell} \neq \varnothing$), on remplace S_i par $S_i \cap T_{m,\ell}$ et on ajoute $S_i \setminus T_{m,\ell}$ comme nouvelle classe. On interrompt alors et on recommence la procédure avec la nouvelle partition.

(d) **Équation obtenue.** Si aucune classe n'est coupée, notons $I'_{m,\ell}$ l'ensemble des indices j tels que les prisonniers de S_j ont allumé exactement m interrupteurs et vu exactement ℓ lumières. Les prisonniers savent si $S_j \cap T_{m,\ell}$ ou $S_j \setminus T_{m,\ell}$ est vide, et connaissent donc $I'_{m,\ell}$. Pour chaque j, on note m_j, ℓ_j tels que $S_j \cap T_{m_j,\ell_j} = S_j$. On obtient alors, avec $x_i = |S_i|$,

$$\sum_{i \in I} x_i \cdot m_i \;=\; \sum_{m,\ell} \sum_{j \in I'_{m,\ell}} x_j \cdot \ell.$$

Comme le graphe est fortement connexe, la solution $(x_i)_i$ est unique (car en ayant allumé tous les interrupteurs de $\bigcup_{i \in I} S_i$, au moins une lumière s'allume dans $\bigcup_{i \notin I} S_i$), et

$$n = \sum_i x_i$$

est alors connu de tous les prisonniers.

Notes et références

Cette énigme est connue dans la littérature mathématique sous le nom de « The Cyclic Prisoners » (Les prisonniers cycliques). Elle a été publiée pour la première fois (avec sa solution complète) par Peter Winkler [1]. Winkler retrace également la provenance de cette énigme : Nathan Bowler de l'Universität Hamburg est crédité comme l'auteur original de cette énigme des prisonniers. Cette énigme a aussi largement circulé sur Internet. Avant et après sa publication officielle, on la retrouve sur des forums de logique, des blogs de mathématiques récréatives et surtout dans des discussions sur StackExchange [2], où elle a gagné en popularité. L'énigme de la prison circulaire s'inscrit dans le cadre des réseaux distribués anonymes, un domaine où des agents identiques tentent de se coordonner malgré une symétrie parfaite et un adversaire puissant. Ce type de problème a été largement étudié dans l'algorithmique distribuée, notamment dans les travaux de Angluin, Fich ou Peleg.

Sources

[1] Peter Winkler. The cyclic prisoners. *The Mathematics of Various Entertaining Subjects : Research in Games, Graphs, Counting, and Complexity, Volume 2*, 2017.

[2] The circular prison of unknown size. https://puzzling.stackexchange.com/questions/16168/the-circular-prison-of-unknown-size, 2015. Puzzling StackExchange.