

International Computer Science Competition

Feuille de problèmes du Qualification Round



Problèmes

Les problèmes de l'ICSC sont composés de problèmes conceptuels et de problèmes de programmation. Le type de problème est indiqué par les symboles suivants :

💡 Problème conceptuel : Nécessite un raisonnement logique ou mathématique, résoluble à la main.

⚡ Problème de programmation : Nécessite l'implémentation d'une solution en code.

Tu peux obtenir jusqu'à 25 points au total. La difficulté varie d'une étoile (le plus facile) à trois étoiles (le plus difficile).

Nous avons compilé du matériel de formation pour t'aider avec les concepts pertinents pour les problèmes posés. Tu peux y accéder ici : <https://icscompetition.org/training>.

Ta Solution

Tu dois fournir une description écrite de ta solution pour **tous les problèmes** dans le document de solution que tu soumetts. Assure-toi d'expliquer ta solution, y compris tout code que tu as écrit. Tu peux soumettre des solutions manuscrites ou tapées.

Pour les problèmes de programmation, tu peux aussi, en option, télécharger tes fichiers de code pour te qualifier pour une mention honorifique spéciale sur ton certificat. Tu peux soumettre tes solutions de codage en Python (3.9), Java (OpenJDK 11), C (C99) ou C++ (GCC 15.1).

Informations de Soumission

Lorsque tu soumetts ta solution, un compte participant sera créé automatiquement pour t'aider à gérer ta soumission. Tu peux accéder à ton compte et à ta soumission en utilisant les identifiants que tu as définis pendant le processus de soumission : <https://icscompetition.org/login>.

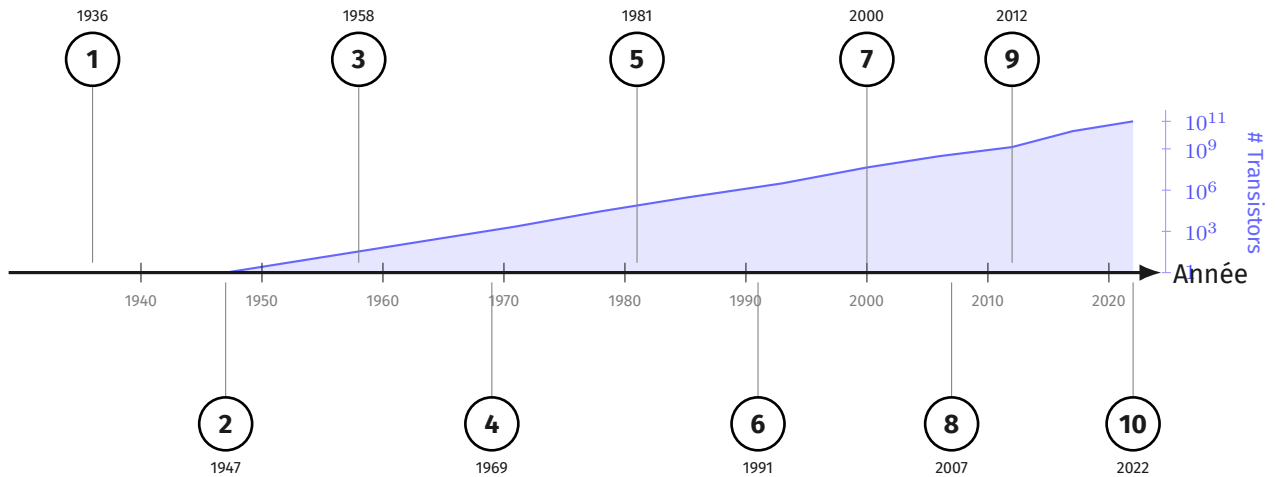
Veuille vérifier que ta solution et tes fichiers de code ont été téléchargés correctement. Tout problème d'exécution de ton code (facultatif) sera signalé dans ton compte participant. Tu dois soumettre ta solution en ligne avant le *Sunday, 5 July 2026, 23:59 UTC+0* à <https://icscompetition.org/submission>.

Pour te qualifier pour le Pre-Final Round, tu dois obtenir au moins 12/14/17 points. Si tu as des questions ou des commentaires, n'hésite pas à nous contacter par e-mail à : info@icscompetition.org.

Nous espérons que tu apprécieras de t'attaquer aux problèmes. Bonne chance !

Problem A**[5 points]**

La chronologie ci-dessous marque dix événements marquants de l'histoire de l'informatique et de la technologie, placés par rapport à la croissance du nombre de transistors sur un microprocesseur commercial.¹



Identifie et nomme chaque événement numéroté sur la chronologie ci-dessus (il peut y avoir plusieurs bonnes réponses) :

1. (1936):

.....

6. (1991):

.....

2. (1947):

.....

7. (2000):

.....

3. (1958):

.....

8. (2007):

.....

4. (1969):

.....

9. (2012):

.....

5. (1981):

.....

10. (2022):

.....

¹Un transistor est un interrupteur ou un amplificateur qui régule le flux de courant dans un circuit, ce qui en fait l'élément de base de tous les ordinateurs et de l'électronique modernes.

Problem B**[5 points]**

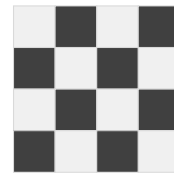
Tu peux télécharger le squelette de code pour ce problème [ici](#).

Le pixel art est une forme d'art numérique où les images sont créées en remplissant des cellules individuelles dans une grille. Ta tâche est d'écrire une fonction qui génère des motifs géométriques spécifiques sur une grille $N \times N$, où chaque cellule est soit 1 (remplie) soit 0 (vide).

Tu dois implémenter les deux formes suivantes :

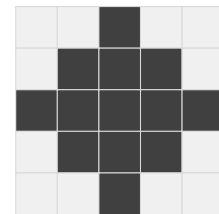
1. Checkerboard ("checkerboard") : Les cellules alternent entre remplies et vides, comme sur un échiquier. La cellule en haut à gauche est toujours 0. Exemple pour $N = 4$:

```
0 1 0 1
1 0 1 0
0 1 0 1
1 0 1 0
```



2. Diamond ("diamond") : Une forme de diamant (losange) centrée dans la grille. N est toujours impair pour cette forme. Exemple pour $N = 5$:

```
0 0 1 0 0
0 1 1 1 0
1 1 1 1 1
0 1 1 1 0
0 0 1 0 0
```



Pour implémenter ces formes, écris la fonction suivante :

```
list<list<int>> generate_shape(int n, string shape)
```

- n : La taille de la grille (grille $N \times N$, $5 \leq N \leq 51$, toujours impair pour la forme diamond).
- $shape$: Soit "checkerboard", soit "diamond".
- La fonction doit retourner une liste/un tableau 2D d'entiers (0 ou 1).

Format de sortie : Affiche la grille sous forme de N lignes, chaque ligne contenant N valeurs séparées par des espaces (0 ou 1).

Problem C**[5 points]**

Tu as peut-être remarqué dans le Problème A que le nombre de transistors sur un micro-processeur commercial ne croît pas de manière linéaire. Dès 1975, Gordon Moore a affirmé que le nombre de transistors doublait environ tous les **2 ans** ; une tendance qui se maintient depuis plus de cinq décennies.

Cela donne la formule :

$$T(t) = T_0 \times 2^{t/2}$$

où T_0 est le nombre de transistors lors d'une année de référence, et t est le nombre ang de années écoulées depuis cette référence.

L'Intel 4004 a été lancé en 1971 en tant que premier microprocesseur commercial monocristal au monde, avec un nombre de transistors de 2,300.

(a) Imagine que tu sois en 1971. En utilisant la formule ci-dessus, prédis le nombre de transistors d'un microprocesseur en 2026.

(b) Compare ta prédiction à un processeur réel de 2026, comme l'Apple M5. Dans quelle mesure la loi de Moore tient-elle bon sur 55 ans ?

(c) Étant donné le nombre réel de transistors en 2026 que tu as trouvé en (b), quelle période de doublement (en années) correspondrait le mieux à la croissance de l'Intel 4004 jusqu'à aujourd'hui ?

Problem D

[5 points]

Tu peux télécharger le squelette de code pour ce problème [ici](#).

Tu travailles sur le serveur d'un jeu de combat 2D en ligne. Deux joueurs sur des PC séparés envoient leurs commandes via Internet à un serveur central à chaque frame (1/60ème de seconde).

C'est la Grand Final. Le match se termine par la victoire du Joueur 1. Le Joueur 2 est furieux et soutient que son coup de grâce est tombé exactement sur la même frame que celui du Joueur 1, et que cela aurait dû être un double KO. Selon les règles du jeu, tous les événements sur une même frame doivent être appliqués avant de vérifier s'il y a un KO. Pour régler le litige, tu rejoues l'intégralité de la partie via la fonction `processGame` pour recalculer les HP finaux, et tu soupçonnes la présence d'un bug. Voici son pseudocode :

processGame

Input: events: liste de (player, frame, attack_value) dans un ordre arbitraire
(à cause de la gigue réseau)

Output: player1_hp, player2_hp

```
1: tri stable des events par numéro de frame
2: for all event in events do
3:   if player1_hp ≤ 0 OU player2_hp ≤ 0 then
4:     break
5:   end if
6:   opponent_hp ← opponent_hp - event.attack_value
7: end for
8: return player1_hp, player2_hp
```

De plus, les logs d'événements du combat montrent :

- (1, 512, 20)
- (2, 512, 20)

avec `player1_hp = 10` et `player2_hp = 15` au moment où la relecture atteint la frame 512.

(a) Quel est le problème avec le pseudocode ci-dessus ? Comment cela cause-t-il des erreurs ?

(b) Implémente la fonction corrigée `processGame` en C, C++, Python ou Java. Elle reçoit le log complet des événements pour un match, commence avec `H` HP pour les deux joueurs, rejoue chaque événement dans l'ordre des frames, et renvoie les HP finaux de chaque joueur.

```
list<int> processGame(list<tuple<int,int,int>> events, int H)
```

- `events` : Une liste de tuples (`player`, `frame`, `attack_value`) où `player` est 1 ou 2, `frame` est un entier non négatif, et `attack_value` est un entier positif.
- `H` : les HP de départ pour les deux joueurs (un entier positif).
- Renvoie une liste de deux entiers : [`hp1`, `hp2`], chacun étant limité à un minimum de 0.

Problem E

[5 points]

En 1998, PageRank a propulsé Google devant tous les autres moteurs de recherche. L'idée : une page est importante si des pages importantes y mènent. Formellement, le PageRank de la page p_i est :

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

où $M(p_i)$ est l'ensemble des pages qui mènent à p_i , $L(p_j)$ est le nombre de liens sortants de p_j (son degré sortant), N est le nombre total de pages, et $d = 0.85$ est le **facteur d'amortissement**.

Aujourd'hui, la "slop" générée par l'IA inonde le web. Quand ces pages pourraient-elles prendre le dessus sur l'algorithme de recherche ?²

(a) Explique comment PageRank fonctionne : l'interprétation du surfeur aléatoire, le rôle de d , pourquoi l'équation est récursive et comment elle est résolue en pratique.

(b) Considère N pages divisées en h pages humaines et $a = N - h$ pages d'IA adverses. Les pages humaines renvoient à k pages choisies uniformément au hasard parmi toutes les N pages. Les pages d'IA renvoient **uniquement** à d'autres pages d'IA. Soit $\alpha = a/N$.

Dérive une expression sous forme fermée pour $S_A = \sum_{p \in A} PR(p)$, le PageRank total de toutes les pages d'IA A , en fonction de α et d .

(c) Trouve la fraction de basculement α^* à laquelle $S_A = S_H$ (c'est-à-dire que les pages d'IA détiennent collectivement autant de PageRank que les pages humaines). Ensuite, discute de la fraction de basculement à $d = 0.85$ et explique les problèmes de PageRank que notre scénario révèle.

²Google et d'autres moteurs modernes utilisent aujourd'hui des centaines de signaux pour classer les pages. Notre scénario fictif se concentre sur un moteur de recherche qui utilise PageRank exclusivement et suppose donc un scénario simplifié.