The CENTRE for EDUCATION in MATHEMATICS and COMPUTING

cemc.uwaterloo.ca

2022 CCC Senior Problem Commentary

Creation of this commentary is a new initiative for the Canadian Computing Competition. Our goal is to give a brief outline of what is required to solve each problem and what challenges may be involved. The notes below can be used to guide anyone interested in trying to solve these problems, but are not intended to describe all the details of a correct solution. We encourage everyone to try and implement these details on their own using their programming language of choice.

S1 Good Fours and Good Fives

The first problem is designed to be accessible with some insight required to obtain full marks.

The first subtask can be hard-coded. That is, the following table can be calculated by hand and a solution can mimic looking up the correct output in the table.

N	1	2	3	4	5	6	7	8	9	10
Output	0	0	0	1	1	0	0	1	1	1

For the second and third subtasks, we can notice that we need $N = 4a + 5b$ for non-negative integers a and b where $a \leq \frac{n}{4}$ and $b \leq \frac{n}{5}$. This means we can try all possible values for a and b using two nested loops.

For a full solution, we can loop through only all possible values of a to determine if there is a corresponding value of b. For each of these values i, this is equivalent to checking if $N - 4i$ is divisible by 5. Alternatively, we take a similar approach but loop through only all possible values of b.

There is also a clever extremely quick solution that involves extending the table listed above for the first subtask to $N = 20$ and considering the quotient and remainder when N is divided by 20.

S2 Good Groups

The second problem in this year's competition requires input to be stored together in a data structure. Because the assignment of students to groups is not given until the end of the input, the constraints cannot be fully considered as they are encountered, and so must be stored in memory. It is helpful to also store the student groups in memory.

Once both types of input are in memory, a solution can be obtained by considering each constraint individually and maintaining a count of how many are violated.

Likely pitfalls include incorrectly assuming something about the order in which student names appear, and erroneously thinking that a group can only violate one constraint.

Care must be taken with the Boolean logic required to test if a given constraint is violated. This will require determining if a given student is in a given group. Doing this inefficiently by looping through all the groups or looping through all the students should have allowed a submission to earn 14 of the available 15 marks. To earn the final mark, this look-up must be made more efficient. This can be accomplished using a data structure built-in to the language designed for this very purpose (e.g. a dictionary in Python, a `HashMap` in Java, or an `unordered_map` in C++).

S3 Good Samples

For the first subtask, it suffices to run a brute-force check of all possible outputs to determine if there is a piece that will satisfy the clarinet players.

For the second subtask, note that no sequence of length 3 is good. Also, all sequences of length 1 are good. Thus, we should try to find a piece with $K - N$ good samples of length 2. We can build it as we go. From the left, if we still need good samples, we can swap to the other pitch, otherwise, we can keep the same pitch. Note that all values between N and $2N - 1$ inclusive are possible and no others are.

For the third subtask, we can build on our solution to the second subtask. As before, suppose we are building a piece, our current sequence is S, and our required count is $K - N$. Depending on how many more good samples we need, we can decide what to append to our sequence S. For instance, if we no longer need any good samples, we can simply add a value equal to the current last value of the sequence. If we need 3 more, we can add the note 4 notes back. That way, there are 3 additional good samples. We can also pick a new number if we want even more good samples. It suffices to greedily choose the number that gives us the most samples up to the remaining amount. When implementing this idea, it's a good idea to keep track of the longest suffix of your sequence S that is good.

The last subtask involves the same idea but when adding a new number to the end, we have to instead do a proper check to see if the new number is too large.

S4 Good Triplets

For the first subtask, it suffices to follow the definition of a good triplet. Firstly, there are $O(N^3)$ ways to select (a, b, c). Secondly, consider the condition that the origin is strictly inside the triangle. This is equivalent to saying that P_a, P_b, and P_c divide the circle into 3 arcs, and each arc is strictly shorter than $\frac{C}{2}$. This leads to the following approach. Sort the positions so that $P_a \leq P_b \leq P_c$, then check that

$$P_b - P_a < \frac{C}{2}, P_c - P_b < \frac{C}{2}, \text{ and } (P_a + C) - P_c < \frac{C}{2}.$$

For example, the first of these checks can be implemented in code as `P[b]-P[a] < (C+1)/2` in C++ or Java, or `P[b]-P[a] < (C+1)//2` in Python 3.

For the second subtask, the goal is to find a solution that works well when C is small. Let L_x be the number of points drawn at location x. If three locations i, j, k satisfy the second condition, we want to add $L_i \times L_j \times L_k$ to the answer. This approach is $O(N + C^3)$ and is too slow. However, we can improve from three nested loops to two nested loops. If i and j are chosen already, either k does not exist, or k is in an interval. It is possible to replace the third loop with a prefix sum array. This algorithm's time complexity is $O(N + C^2)$.

It is possible to optimize this approach to $O(N + C)$ and earn 15 marks. The idea is to eliminate both the j and k loop. Start at $i = 0$ and compute $L_j \times L_k$ in $O(C)$ time. The next step is to transition from $i = 0$ to $i = 1$, and to maintain $L_j \times L_k$ in $O(1)$ time. This requires strong familiarity with prefix sum arrays. Care with overflow and off-by-one errors is required.

S5 Good Influencers

In the first subtask, the students and their friendships form a line.

One approach to this subtask involves dynamic programming. Let $DP[i]$ be the minimum cost required for the first i students to end up intending to write the CCC (ignoring any later students). Note that $DP[0] = 0$, and our answer will be $DP[N]$.

La version française figure à la suite de la version anglaise.

For each value of i from 0 to $N-1$, we'll consider transitions onwards from that state, with the subarray of students from $i+1$ to j (for each possible j such that $i < j \leq N$) ending up intending to write the CCC. If at least one of those students has a P value of `Y`, then it's possible to perform this transition by choosing a `Y` student and "expanding their influence" to all others, updating $DP[j]$ accordingly.

For each i, we can consider all possible values of j while computing their transitions' minimum costs in a total of $O(N)$ time, resulting in an overall time complexity of $O(N^2)$.

To solve the remaining subtasks, we'll use a different dynamic programming formulation, this time on the tree structure formed by the students and their friendships. We'll consider the tree to be rooted at an arbitrary node (student), such as node 1.

Let $DP[i][j]$ (defined for $1 \leq i \leq N$ and $0 \leq j \leq 2$) be the minimum cost required for all students in i's subtree to end up intending to write the CCC, such that:

- if $j = 0$, i will be influenced by its parent
- if $j = 1$, i will have no particular interaction with its parent
- if $j = 2$, i will influence its parent

Our answer will then be $DP[1][1]$.

As is typical for DP on trees, we'll recurse through the tree, and for each node i, we'll compute $DP[i][0..2]$ based on the DP values of i's children. Each value $DP[i][j]$ must be computed carefully, with different logic depending on the values of $P[i]$ and j.

An implementation of this algorithm may take $O(N^2)$ time (with some DP transitions taking $O(N)$ time each to compute), but optimizations can be applied to reduce it to an overall time complexity of $O(N)$ and have it earn full marks.

La version française figure à la suite de la version anglaise.

Le CENTRE d'ÉDUCATION en MATHÉMATICS et en INFORMATIQUE

cemc.uwaterloo.ca

Commentaires sur le CCI de niveau sénior de 2022

La création de ce commentaire est une nouvelle initiative pour le Concours canadien d'informatique. Son objectif est de donner un bref aperçu de ce qu'il faut faire pour résoudre chaque problème et des défis à relever. Les notes ci-dessous peuvent être utilisées pour guider toute personne intéressée à essayer de résoudre ces problèmes. Or, elles n'ont pas pour but de décrire tous les détails d'une solution correcte. Nous encourageons toute personne intéressée à essayer d'élaborer une solution correcte en utilisant le langage de programmation de son choix.

S1 Des quatre et des cinq

Le premier problème est conçu pour être accessible à tous, mais demande une certaine perspicacité pour obtenir le nombre maximum de points.

La première sous-tâche peut être codée en dur. C'est-à-dire que le tableau suivant peut être calculé à la main et une solution peut consister à rechercher la sortie correcte dans le tableau.

N	1	2	3	4	5	6	7	8	9	10
Sortie	0	0	0	1	1	0	0	1	1	1

Pour les deuxième et troisième sous-tâches, on remarque que $N = 4a + 5b$, a et b étant des entiers non négatifs tels que $a \leq \frac{n}{4}$ et $b \leq \frac{n}{5}$. Cela signifie que l'on peut essayer toutes les valeurs possibles de a et b en utilisant deux boucles imbriquées.

Pour une solution complète, on peut parcourir en boucle toutes les valeurs possibles de a pour déterminer s'il existe une valeur correspondante de b. Pour chacune de ces valeurs i, cela revient à vérifier si $N - 4i$ est divisible par 5. On peut également adopter une approche similaire en ne parcourant en boucle que toutes les valeurs possibles de b.

Il existe également une solution astucieuse et extrêmement rapide qui consiste à prolonger le tableau de la première sous-tâche jusqu'à $N = 20$ et de considérer le quotient et le reste lorsque N est divisé par 20.

S2 Travail de groupe

Le deuxième problème du concours de cette année exige qu'une collection d'entrées soit stockée ensemble dans une structure de données. Comme les groupes d'élèves ne sont donnés qu'à la fin de l'entrée, les contraintes ne peuvent pas être entièrement évaluées au fur et à mesure qu'elles sont rencontrées, et doivent donc être stockées en mémoire. Il est utile de stocker également les groupes d'élèves en mémoire.

Une fois que les deux types d'entrée sont en mémoire, une solution peut être obtenue en considérant chaque contrainte individuellement et en gardant le compte du nombre de contraintes violées.

Quelques pièges possibles sont le fait de faire une supposition erronée quant à l'ordre dans lequel paraissent les noms des élèves ou de supposer à tort qu'un groupe ne peut violer qu'une seule contrainte.

Il faudra veiller à bien développer la logique booléenne nécessaire pour tester si une contrainte donnée est violée. Pour le faire, il faudra déterminer si un élève donné se trouve dans un groupe donné. Une manière

inefficace de le faire serait de parcourir en boucle tous les groupes ou tous les élèves. Une soumission employant cette technique ne se verrait attribuer que 14 des 15 points disponibles. Pour obtenir le dernier point, cette recherche doit être plus efficace. Pour ce faire, on peut utiliser une structure de données intégrée au langage et conçue à cet effet (par exemple, un dictionnaire en Python, une `HashMap` en Java ou un `unordered_map` en C++).

S3 Des échantillons qui sont bons

Pour la première sous-tâche, il suffit d'effectuer une vérification par force brute de toutes les sorties possibles pour déterminer s'il existe un morceau qui satisfera aux exigences des clarinettistes.

Pour la deuxième sous-tâche, remarquons qu'aucune séquence de longueur 3 n'est bonne. Or, toutes les séquences de longueur 1 sont bonnes. Ainsi, on devrait essayer de trouver un morceau contenant $K - N$ échantillons de longueur 2 qui sont bons. On peut le construire au fur et à mesure. En partant de la gauche, si on a toujours besoin de bons échantillons, on peut passer à l'autre hauteur de notes, sinon, on peut garder la même hauteur. Remarquons que seules les valeurs de N à $2N - 1$ sont possibles.

Pour la troisième sous-tâche, on peut s'appuyer sur notre solution à la deuxième sous-tâche. Comme précédemment, supposons que l'on est en train de construire un morceau, que notre séquence actuelle est S, et qu'on cherche à avoir $K - N$ échantillons qui sont bons. Selon le nombre de bons échantillons supplémentaires dont on a besoin, on peut décider de ce que l'on doit ajouter à notre séquence S. Par exemple, si on n'a plus besoin de bons échantillons, on peut tout simplement ajouter une valeur égale à la dernière valeur actuelle de la séquence. Si on a besoin de 3 bons échantillons supplémentaires, on peut ajouter la note 4 notes plus tôt dans la séquence à cette fin. On peut également choisir un nouveau nombre si on veut avoir un plus grand nombre de bons échantillons. Il suffit de choisir le nombre qui nous donne le plus d'échantillons jusqu'à la quantité restante. En mettant en œuvre cette idée, il serait souhaitable de garder une trace du plus long suffixe de votre séquence S qui est bon.

On peut élaborer la dernière sous-tâche en partant de la même idée. Or, en ajoutant un nouveau nombre à la fin, il faut faire une vérification pour s'assurer que le nouveau nombre ne soit pas trop grand.

S4 De bons triplets

Pour la première sous-tâche, il suffit de suivre la définition d'un bon triplet. Premièrement, il y a $O(N^3)$ façons de choisir (a, b, c). Deuxièmement, considérons la condition que l'origine soit située strictement à l'intérieur du triangle. Cela revient à dire que P_a, P_b et P_c divisent le cercle en 3 arcs, et que chaque arc est strictement plus court que $\frac{C}{2}$. Cela nous mène à l'approche suivante. On trie les emplacements de manière que $P_a \leq P_b \leq P_c$ et on vérifie ensuite que

$$P_b - P_a < \frac{C}{2}, P_c - P_b < \frac{C}{2} \text{ et } (P_a + C) - P_c < \frac{C}{2}.$$

Par exemple, la première de ces vérifications peut être codée sous la forme `P[b]-P[a] < (C+1)/2` dans C++ ou Java, ou `P[b]-P[a] < (C+1)//2` dans Python 3.

Pour la deuxième sous-tâche, l'objectif est de trouver une solution qui marche bien lorsque C est petit. Soit L_x le nombre de points que l'on dessine à l'emplacement x. Si trois emplacements i, j, k satisfont à la deuxième condition, alors on veut ajouter $L_i \times L_j \times L_k$ à la réponse. Cette approche est $O(N + C^3)$ et est donc trop lente. Cependant, on peut passer de trois boucles imbriquées à deux boucles imbriquées. Si i et j sont déjà choisis, soit k n'existe pas, soit k est dans un intervalle. Il est possible de remplacer la troisième boucle par un tableau de sommes de préfixes. Cet algorithme a une complexité en temps de $O(N + C^2)$.

Il est possible d'optimiser cette approche à $O(N + C)$ et d'obtenir les 15 points possibles. L'idée serait

d'éliminer les boucles j et k. On commencerait à $i = 0$ et on calculerait $L_j \times L_k$ en temps $O(C)$. L'étape suivante consiste à passer de $i = 0$ à $i = 1$ et à maintenir le calcul $L_j \times L_k$ en temps $O(1)$. Cela nécessite une grande familiarité avec les tableaux de sommes de préfixes. Il est nécessaire de faire attention aux erreurs de dépassement d'entier et aux erreurs de type « off-by-one ».

S5 De bons influenceurs

Dans la première sous-tâche, les élèves et leurs amitiés forment une ligne.

Une approche possible serait de faire appel à la programmation dynamique. Soit $PD[i]$ le coût minimum nécessaire pour que les i premiers élèves finissent par avoir l'intention de rédiger le CCI (sans tenir compte des élèves après eux). Remarquons que $PD[0] = 0$. On aura donc $PD[N]$ comme réponse.

À partir de ce point, on considère les transitions possibles pour chaque valeur de i de 0 à $N - 1$, où le sous-ensemble d'élèves de $i + 1$ à j (pour chaque j possible tel que $i < j \leq N$) finissent par avoir l'intention de rédiger le CCI. Si au moins un de ces élèves a une valeur P de `Y`, alors il est possible d'effectuer cette transition en choisissant un élève `Y` et en « étendant son influence » à tous les autres, tout en mettant à jour $PD[j]$ en conséquence.

Pour chaque i, on peut considérer toutes les valeurs possibles de j tout en calculant les coûts minimaux de leurs transitions en temps total $O(N)$, ce qui donne en tout une complexité en temps de $O(N^2)$.

Pour résoudre les sous-tâches restantes, nous utiliserons une autre formulation de programmation dynamique, cette fois sur l'arborescence formée par les élèves et leurs amitiés. Considérons que l'arbre a pour racine un nœud quelconque (élève), tel que le nœud 1.

Soit $PD[i][j]$ (i et j vérifiant respectivement $1 \leq i \leq N$ et $0 \leq j \leq 2$) le coût minimum nécessaire pour que tous les élèves dans le sous-arbre de i finissent par avoir l'intention de rédiger le CCI de manière que :

- si $j = 0$, i sera influencé par son parent
- si $j = 1$, i n'aura aucune interaction particulière avec son parent
- si $j = 2$, i influencera son parent

Notre réponse sera alors $PD[1][1]$.

Ensuite, on fait un parcours récursif dans l'arbre et, pour chaque noeud i, on calcule $PD[i][0..2]$ en nous basant sur les valeurs PD des enfants de i. Chaque valeur $PD[i][j]$ doit être calculée soigneusement, avec une logique différente selon les valeurs de $P[i]$ et j.

La mise en œuvre de cet algorithme peut prendre $O(N^2)$ temps de calcul (avec certaines transitions PD étant chacune calculable en $O(N)$). Or, il est possible de l'optimiser pour qu'il s'exécute en temps $O(N)$, ce qui mériterait un nombre maximum de points.