Rapport problèmes de la 2ème édition du MTYM
Moroccan Tournament of Young Mathematicians

Spider-Math
Membres de l'équipe:
-Zineb El Ofir
-Abderrahhmane Ait El Kaddi
-Rayane Hakam
Encadrants:
-Hassan Hilal
-Othmane Sabor

December 17, 2024

Contents

1 Remerciement

Je souhaite exprimer ma profonde gratitude à l'équipe organisatrice du MTYM et à tous ceux qui ont rendu cette expérience unique possible. Grâce à votre dévouement et à votre professionnalisme, ce tournoi a été bien plus qu'une simple compétition : il a été une aventure inoubliable et enrichissante.

Merci pour cette expérience exceptionnelle: Tout au long de ces journées, nous, participants, avons eu l'opportunité de nous immerger dans un environnement stimulant, entourés de personnes partageant la même passion pour les mathématiques. Chaque problème proposé, chaque échange et chaque débat ont été une source d'inspiration et de motivation. Cette expérience a non seulement renforcé mes connaissances, mais elle m'a également permis de développer de précieuses compétences en résolution de problèmes et en travail d'équipe.

Un merci spécial à l'Université Al Akhawayn et aux Partenaires: Je tiens également à remercier l'Université Al Akhawayn et tous les partenaires pour leur soutien exceptionnel. Grâce à votre générosité et votre accueil, nous avons pu profiter d'un cadre inspirant et favorable à l'échange et à l'apprentissage. Votre contribution a permis à chacun de vivre pleinement cette aventure mathématique, dans un environnement serein et convivial.

Reconnaissance envers les Conférenciers et Intervenants: Les conférences et ateliers ont été des moments forts de cette expérience, offrant des perspectives nouvelles et ouvrant des horizons insoupçonnés. Un immense merci aux conférenciers et intervenants pour le temps et le savoir qu'ils ont partagé avec nous, pour leurs conseils et leur passion contagieuse qui nous ont inspirés à aller encore plus loin dans notre parcours scientifique.

Gratitude envers l'Équipe Organisatrice et les Bénévoles: À tous les membres de l'organisation et aux bénévoles : merci pour votre travail exceptionnel et votre engagement sans faille. Grâce à vous, nous avons pu vivre un tournoi fluide et enrichissant, où chaque détail a été soigneusement pensé pour notre bien-être et notre réussite. Ce tournoi restera gravé dans ma mémoire comme une étape marquante de mon parcours académique et personnel. J'espère que cette expérience me permettra de progresser davantage, de persévérer dans ma passion pour les mathématiques et de continuer à explorer de nouveaux défis.

Encore une fois, merci à tous pour cette aventure inoubliable et pour avoir fait du MTYM un événement unique et inspirant.

2 les difficultés rencontrées

En général, nous n'avons pas rencontré beaucoup de problèmes, mais trois principaux défis se sont posés.
Premièrement, l'utilisation de LaTeX pour rédiger les solutions peut être compliquée. Les commandes sont parfois difficiles à retenir, et une petite erreur peut bloquer tout le document. Rédiger des équations ou structurer les solutions demande une certaine maîtrise du logiciel, ce qui peut prendre du temps. De plus, l'ajout de graphiques ou de schémas nécessite souvent des extensions, ce qui complique encore les choses, surtout pour les débutants.
Deuxièmement, trouver un équilibre entre les études et la résolution des problèmes est un vrai défi, car la majorité de l'équipe est en terminale. Cela rend la gestion du temps entre les exigences scolaires et le travail pour le MTYM particulièrement difficile.
Troisièmement, la distance qui nous sépare d'un des membres de l'équipe complique la coordination. Nous devons organiser des réunions hebdomadaires pour répartir les tâches et maintenir une bonne communication. À cela s'ajoutent les défis liés à chaque problème mathématique, qui demandent du temps et une réflexion approfondie. Malgré tout, nous travaillons pour surmonter ces obstacles ensemble.
Amassons les jetons:
1-Difficultés concernant la généralisation: Passer de deux cases et dix jetons à des configurations avec K cases et N jetons a été difficile. Ce qui était efficace pour les cas simples ne marchait pas pour des données complexes. 2-Des conjectures fausses : Nous avons fait des hypothèses qui nous semblaient justes au début, mais après avoir trouvé des contradictions, ça s'est avéré que c'était faux. Cela nous a forcés à tout recommencer, ce qui a été frustrant, mais nécessaire pour progresser. 3-Des méthodes erronées au début : Nous avions utilisé des méthodes non efficaces. Après avoir essayé plusieurs fois , nous avons réussi à trouver des méthodes qui s'alignaient avec les données du problème. 4-Beaucoup d'essais et de persévérance : Nous avons essayé, réessayé,et réessayé à plusieurs reprises,sans baisser les bras,avant d'arriver à des solutions assez convaincantes. 5-Un défi stimulant : Malgré toutes ces difficultés, le problème était vraiment intéressant. Il nous a mené à chercher, apprendre , réfléchir, et à collaborer pour surmonter toutes sortes d'obstacles.

Ces défis ont non seulement testé nos compétences , mais ils nous ont aussi appris à persévérer face à l'incertitude. Toujours valoriser le processus de recherche et accepter le fait de ne pas être parfait, commettre des fautes , puis les corriger après une longue recherche et réflexion.
Quoi pour qui ?:
Lors de la résolution du problème 2, nous avons fait face à plusieurs problèmes dont: La découverte de plusieurs nouveaux concepts qui se présentent dans la sous modularité , les algorithmes non usuels... ce qui nous a pris du temps pour chercher et comprendre, puis résoudre le problème. Nous avons aussi utilisé le langage de programmation Python pour nous simplifier les calculs et pour se focaliser sur les méthodes. Lors de ce problème, nous avons beaucoup appris au niveau des concepts mathématiques et aussi au niveau de la recherche scientifique ,ce qui a développé nos compétences mathématiques ,ainsi que nos compétences générales en recherche et en travail d'équipe, même avec les différents obstacles qu'on a rencontrés.
Painville:
Le problème 3 a été le problème où on a trouvé le plus de difficultés, la première chose était que le problème était difficile à comprendre. Le temps a été un vrai inconvénient pour gérer entre la compréhension du problème , la compréhension des questions, la compréhension des théorèmes, puis la résolution des questions. Le problème était très long ce qui était un majeur problème puisqu'on avait pas assez de temps. On a aussi du faire beaucoup de recherche pour comprendre les théories qu'on utilisera lors de la résolution comme la théorie des graphes. Au niveau du latex, représenter les graphes était vraiment non usuel ce qui nous a rajouté un autre problème juste pour la rédaction sur latex. Ce problème était très exceptionnel pour plusieurs raisons dont le fait qu'il soit le dernier problème, après le tas d'efforts fournis dans le problème 1 et le problème 2, nous avons découvert un autre niveau et un long processus qu'on devait faire pendant un temps très limité. Ce problème nous a appris la gestion du temps, le travail d'équipe et biensur l'esprit de recherche, ça a aussi renforcé et développé nos compétences mathématiques.

3 Amassons les jetons

3.1 Résolution :

3.1.1 Échauffement

1/

1. (a) Abla joue $(5,5)$ et Brahim répond $(3,7)$. Le résultat de la configuration est donné par la formule :

$$R((a_1,a_2);(b_1,b_2)) = a_1 \cdot (b_1+b_2) + 2 \cdot a_2 \cdot b_2.$$

Substituons les valeurs :

$$\begin{aligned} R((5,5);(3,7)) &= 5 \cdot (3+7) + 2 \cdot 5 \cdot 7, \\ &= 5 \cdot 10 + 2 \cdot 5 \cdot 7, \\ &= 50 + 70, \\ &= 120. \end{aligned}$$

Ainsi, $R((5,5);(3,7)) = 120$. Cependant, $(3,7)$ ne permet pas à Brahim d'atteindre le plus petit résultat possible. Pour minimiser R, Brahim doit jouer $(10,0)$, ce qui donne :

$$\begin{aligned} R((5,5);(10,0)) &= 5 \cdot (10+0) + 2 \cdot 5 \cdot 0, \\ &= 5 \cdot 10, \\ &= 50. \end{aligned}$$

Le plus petit résultat possible est donc $\boxed{50}$ lorsque Brahim joue $(10,0)$.

1. (b) Lorsque Abla joue $(4,6)$, le résultat s'écrit :

$$R((4,6);(b_1,b_2)) = 4 \cdot (b_1+b_2) + 2 \cdot 6 \cdot b_2.$$

Si Brahim veut minimiser R, il joue $(10,0)$:

$$\begin{aligned} R((4,6);(10,0)) &= 4 \cdot (10+0) + 2 \cdot 6 \cdot 0, \\ &= 4 \cdot 10, \\ &= 40. \end{aligned}$$

Le plus petit résultat est donc $\boxed{40}$.

1-c/On a Abla qui remarque que lorsqu'elle joue $(5,5)$, la valeur du plus petit résultat est différente de celle obtenue en jouant $(4,6)$. On sait que brahim veut minimiser le resultat de la configuration donc :

$$Min(R((a_1,a_2);(b_1,b_2))) = Min(a_1 \cdot (b_1+b_2) + 2 \cdot a_2 \cdot b_2)$$
$$Min(R((a_1,a_2);(b_1,b_2))) = Min(a_1 \cdot 10 + 2 \cdot a_2 \cdot b_2)$$

et pour minimiser son score Brahim doit necessairement jouer tous ses jetons dans b_1 Donc :

$$Min(R((a_1,a_2);(b_1,b_2))) = 10 \cdot a_1$$

Par suite abla pour maximiser son score elle doit jouer tous ses jetons dans a_1 pour $a_1 = 10$

$$Max(Min(R((a_1,a_2);(b_1,b_2)))) = 100$$

par suite Abla joue (10;0) et Brahim (10;0)

2/

2. Calculons $V(10,0)$ et $V(5,5)$.

(a) Lorsque $(b_1,b_2)=(10,0)$:

$$\begin{aligned} V((10,0)) &= \max_{a_1+a_2=10} R((a_1,a_2);(10,0)),\\ &= \max_{a_1+a_2=10} \big(a_1\cdot 10+2\cdot a_2\cdot 0\big),\\ &= \max_{a_1+a_2=10} 10\cdot a_1,\\ &= 10\cdot 10,\\ &= 100. \end{aligned}$$

Donc, $V((10,0))=\boxed{100}$.

(b) Lorsque $(b_1,b_2)=(5,5)$:

$$\begin{aligned} V((5,5)) &= \max_{a_1+a_2=10} R((a_1,a_2);(5,5)),\\ &= \max_{a_1+a_2=10} \big(a_1\cdot 10+2\cdot a_2\cdot 5\big),\\ &= \max_{a_1+a_2=10} \big(10\cdot a_1+10\cdot a_2\big),\\ &= 10\cdot(a_1+a_2),\\ &= 10\cdot 10,\\ &= 100. \end{aligned}$$

3/

3-a/ On a Abla joue $(5,5)$ et Brahim répond $(3,7)$ donc le score de la configuration est :

$$S((a_1,a_2);(b_1,b_2))=\frac{R((a_1,a_2);(b_1,b_2))}{V(b_1,b_2)}.$$

$$S((5,5);(3,7))=\frac{R((5,5);(3,7)}{V(3,7)}$$

$$S((5,5);(3,7))=\frac{5\cdot 10+2\cdot 5\cdot 7}{Max(a_1\cdot 10+2\cdot a_2\cdot 7)}$$

$$\boxed{S((5,5);(3,7))=\frac{120}{140}\simeq 0.85714}$$

$(3,7)$ ne permet pas à Brahim d'atteindre le score de configuration le plus petit possible , Par exemple $(10,0)$ nous donne un score :

$$S((5,5);(10,0))=1/2=0.5$$

On a montrer par **contre-exemple** que (3;7) ne permet pas à Brahim d'atteindre le score de configuration le plus petit possible

Donc cherchons la configuration qui donne un score minimal :

La démo-1 nous donne la formule de V pour b_2 (car b_1 n'a pas d'effet direct sur la relation, il n'est le coefficient d'aucun a) dans des intervalles précis. Lors de cette démo, on a deux intervalles possibles :

$$b_2\in[0,5] \quad \text{ou} \quad b_2\in]0,10]$$

Demo-1 On a

$$V((b_1, b_2)) = Max(a_1 \cdot (b_1 + b_2) + 2 \cdot a_2 \cdot b_2)$$

$$V((b_1, b_2)) = Max(a_1 \cdot 10 + a_2 \cdot 2 \cdot b_2)$$

Par suite on a 2 cas pour ce maximum Comparons donc 10 et $2 \cdot b_2$:

- Si $b_2 \leq 5$

$$\boxed{\mathrm{V((b1,\ b2))} = 10\mathrm{a}_1 = 100}$$

- Si $b_2 > 5$

$$\boxed{\mathrm{V((b1,\ b2))} = 20\mathrm{b}_2}$$

Trouvons Donc Le score minimale que peut atteindre Brahim :

$$\rightarrow Min(S) = Min(\frac{R((a_1, a_2); (b_1, b_2))}{V((b1, b2))})$$

$$Min(S) = Min(\frac{50 + 10 \cdot b_2}{V((b1, b2))})$$

- Si $b_2 \leq 5$:

$$Min(S) = Min(\frac{50 + 10 \cdot b_2}{100})$$

$$Min(S) = Min(\frac{50}{100} + \frac{1 \cdot b_2}{10})$$

$$\boxed{Min(S) = 50/100 = 0.5}$$

Pour Brahim avec la configuration $(10, 0)$.

- Si $b_2 > 5$:

$$Min(S) = Min(\frac{50 + 10 \cdot b_2}{20 \cdot b_2})$$

$$Min(S) = Min(\frac{50}{20 \cdot b_2} + \frac{1}{2})$$

$$\boxed{Min(S) = 3/4 = 0.75}$$

Pour Brahim avec la configuration $(0, 10)$.
Donc pour que Brahim atteint le minimale possible apres que abla joue (5,5) il doit jouer (10,0) , Pour un score de 0.5 (car $0{,}75 > 0.5$)

3-b/ On a Abla joue maintenant $(4, 6)$ Donc le score s ecrit sous forme de :

$$S((4, 6); (b_1, b_2)) = \frac{R((4, 6); (b_1, b_2))}{V(b_1, b_2)}.$$

$$S((4, 6); (b_1, b_2)) = \frac{40 + 12 \cdot b_2}{V(b_1, b_2)}.$$

D'apres la **Demo-1** On a :

- Si $b_2 \leq 5$:

$$Min(S) = Min(\frac{40 + 12 \cdot b_2}{100})$$
$$Min(S) = Min(\frac{40}{100} + \frac{12 \cdot b_2}{100})$$
$$\boxed{Min(S) = 40/100 = 0.4}$$

Pour Brahim avec la configuration $(10, 0)$.

- Si $b_2 > 5$:

$$Min(S) = Min(\frac{40 + 12 \cdot b_2}{20 \cdot b_2})$$
$$Min(S) = Min(\frac{40}{20 \cdot b_2} + \frac{12}{20})$$
$$\boxed{Min(S) = 1/5 + 12/20 = 0.8}$$

Pour Brahim avec la configuration $(0, 10)$.
Donc pour que Brahim atteint le minimale possible apres que abla joue (5,5) il doit jouer (10,0) , Pour un score de 0.4 (car 0,4 $<$ 0.8)

-

3-c/ Cherchons a ce que doit jouer Abla pour que le plus petit résultat de la configuration finale soit le plus grand possible et quelle réponse Brahim doit-il donner face à cela : On sait que le score s'ecrit sous forme de :

$$S((a_1, a_2); (b_1, b_2)) = \frac{R((a_1, a_2); (b_1, b_2))}{V(b_1, b_2)}.$$

$$S((a_1, a_2); (b_1, b_2)) = \frac{a_1 \cdot (b_1 + b_2) + 2 \cdot a_2 \cdot b_2}{V(b_1, b_2)}.$$

D'apres la **Demo-1** On aura deux cas :

- Si $b_2 \leq 5$ ($b_1 = 10 et b_2 = 0$):

$$Min(S) = Min(\frac{a_1 \cdot (b_1 + b_2) + 2 \cdot a_2 \cdot b_2}{100})$$
$$Min(S) = Min(\frac{10 \cdot a_1}{10} + \frac{2 \cdot b_2}{100})$$
$$Min(S) = Min(\frac{10 \cdot a_1}{10})$$
$$\boxed{Min_1(S) = 0.1 \cdot a_1}$$

- Si $b_2 > 5$ ($b_1 = 0$ et $b_2 = 10$):

$$Min(S) = Min(\frac{a_1 \cdot (b_1 + b_2) + 2 \cdot a_2 \cdot b_2}{20 \cdot b_2})$$
$$Min(S) = Min(\frac{10 \cdot a_1}{20 \cdot b_2} + \frac{2 \cdot a_2 \cdot b_2}{20 \cdot b_2})$$
$$Min(S) = Min(\frac{10 \cdot a_1}{200} + \frac{2 \cdot a_2}{20})$$
$$Min_2(S) = 0.05 \cdot a_1 + 0.1 \cdot a_2$$

$$\boxed{Min_2(S) = -0.05 \cdot a_1 + 1}$$

D'apres les deux Min(S) on remarque que si Abla joue $a_1 = n$ elle augmente son score dans Min_1 de 0.1*n et le diminue dans Min_2 de 1-0.05*n Donc pour pouvoire augmenter le Min au max dans les deux cas elle doit equilibrer les deux Min d'ou on resout $S(Min_1) = S(Min_2)$)car Min_1 est une fonction croissante et Min_2 decroissante donc la valeur maximale de ce Min est l'intersection des deux fonctions) :

$$Min_2(S) = Min_1(S)$$

$$-0.05 \cdot a_1 + 1 = 0.1 \cdot a_1$$

$$-0.15 \cdot a_1 + 1 = 0$$

$$a_1 = 20/3$$

$$\boxed{a_1 \simeq 6.6666667}$$

et on a :

$$a_1 + a_2 = 10$$

$$\boxed{a_2 = 10/3 \simeq 3.33333}$$

Pour trouver nos entiers arrondissons a_1 et a_2 ce qui donne ($a_1 = 7$ et $a_2 = 3$) ou ($a_1 = 6$ et $a_2 = 4$)
D'ou pour que abla atteint le maximum du minimum elle doit jouer (7,3) ou (6,4) Et pour Brahim on sait que :

$$\begin{cases} \text{Si } b_2 \leq 5 \ (b_1 = 10 \text{ et } b_2 = 0) : \\ \quad Min_1(S) = 0.1 \cdot a_1 \\ \\ \text{Si } b_2 > 5 \ (b_1 = 0 \text{ et } b_2 = 10) : \\ \quad Min_2(S) = -0.05 \cdot a_1 + 1 \end{cases}$$

1. Apres que abla joue $(7, 3)$:

$$Min_1(S) = 0.1 \cdot a_1$$

$$\boxed{Min_1(S) = 0.7}$$

et :

$$Min_2(S) = -0.05 \cdot a_1 + 1$$

$$\boxed{Min_2(S) = 0.65}$$

Et puisque

$$Min_2 < Min_1$$

Donc

$$\boxed{\textbf{Min(S)=0.65}}$$

2. Apres que abla joue $(6, 4)$:

$$Min_1(S) = 0.1 \cdot a_1$$

$$\boxed{Min_1(S) = 0.6}$$

et :

$$Min_2(S) = -0.05 \cdot a_1 + 1$$

$$\boxed{Min_2(S) = 0.7}$$

Et puisque

$$Min_2 > Min_1$$

Donc

$$\boxed{\textbf{Min(S)=0.6}}$$

parsuite d'aprés 1 et 2 on a $Max(Min(S)) = 0.65$ pour Abla qui va jouer (7,3) D'ou brahim va repondre a abla par (0,10) ce qui nous donne un score de 0.65

3.1.2 Generalisation-Demo

"Après les questions d'échauffement, nous avons pu explorer la méthodologie ainsi que certaines conjectures qui pourraient avoir un grand impact sur la généralisation. La première conjecture stipule que Brahim doit nécessairement jouer tous ses jetons sur une seule case. La deuxième conjecture, quant à elle, affirme que, pour qu'Ablâ atteigne son maximum, il est nécessaire de résoudre un système où l'on égalise toutes les fonctions minimales afin de trouver des valeurs a_i qui nous donneront le maximum, quel que soit le mouvement final de Brahim. Nous avons utilisé ces deux conjectures pour aboutir à une solution finale sous forme de formule. Dans cette partie, nous allons démontrer ces deux conjectures ainsi que la formule générale."

3.1.2.0 conj-1

Montrons que Brahim doit nécessairement jouer tous ses jetons sur une seule case pour atteindre le minimum :

On suppose que brahim joue ses jetons de tel facons qui il ne les met pas toute sur une case
Donc

$$S((b_1,...,b_k)) = \frac{a_1 \cdot N + \cdots + K \cdot a_K \cdot b_K}{V((b_1,...,b_k))}$$

et donc

$$V((b_1,...,b_k)) = N{\cdot}Max(V((b_1,b_2,\ldots,b_K)) = N{\cdot}\max\left(a_1 \cdot (b_1+b_2+\cdots+b_K) + 2a_2 \cdot (b_2+\cdots+b_K) + \cdots + K \cdot a_K \cdot b_K\right.$$

Et pour trouver ce max on doit necessairement multiplier a_i par le plus grand facteur donc on peut ecrire ce V en d autre facon :

$$V((b_1,...,b_k)) = N \cdot Max(i \cdot \sum_{j=i}^{K} b_j)$$

Donc on aura plusieurs cas de V :

$$\begin{cases} V_1((b_1,\ldots,b_k)) = N^2 \\ V_2((b_1,\ldots,b_k)) = 2N(b_2+b_3...+b_k) \\ V_3((b_1,\ldots,b_k)) = 3N(b_3+b_4...+b_k) \\ \vdots \\ V_k((b_1,\ldots,b_k)) = kNb_k \end{cases}$$

Donc le score minimum pour brahim sera :

$$\begin{cases} MinS_1((b_1,\ldots,b_k)) = \frac{a_1 \cdot N}{N^2} + \frac{2a_2 \cdot (b_2+\cdots+b_K)}{N^2} + \ldots + \frac{ka_kb_k}{N^2} \\ MinS_2((b_1,\ldots,b_k)) = \frac{a_1 \cdot N}{2N(b_2+b_3...+b_k)} + \frac{2a_2 \cdot (b_2+\cdots+b_K)}{2N(b_2+b_3...+b_k)} + ... \\ \vdots \\ MinS_k((b_1,..,b_k)) = \frac{a_1 \cdot N}{kNn_k} + \frac{2a_2 \cdot (b_2+\cdots+b_K)}{kNb_k} + \cdots + \frac{k \cdot a_k \cdot b_k}{kNb_k} \end{cases}$$

Mais Si brahim joue tous ses jetons sur une case on a (notons le score S') :

$$\begin{cases} \text{Si } b_1 = N: \quad MinS'_1((b_1,\ldots,b_k)) = \frac{a_1 \cdot N}{N^2} \\ \text{Si } b_2 = N: \quad MinS'_2((b_1,\ldots,b_k)) = \frac{a_1 \cdot N}{2N^2} + \frac{2 \cdot a_2 \cdot N}{2N^2} \\ \text{Si } b_3 = N: \quad MinS'_3((b_1,..,b_k)) = \frac{a_1 \cdot N}{3N^2} + \frac{2 \cdot a_2 \cdot N}{3N^2} + \frac{3 \cdot a_3 \cdot N}{3N^2} \\ \vdots \\ \text{Si } b_k = N: \quad MinS'_k((b_1,..,b_k)) = \frac{a_1 \cdot N}{kN^2} + \frac{2 \cdot a_2 \cdot N}{kN^2} + \cdots + \frac{k \cdot a_k \cdot N}{kN^2} \end{cases}$$

Comparons entre MinS et MinS' :

$$\begin{cases} MinS_1((b_1,\ldots,b_k)) \text{et} MinS'_1((b_1,\ldots,b_k)) \\ MinS_2((b_1,\ldots,b_k)) \text{et} MinS'_2((b_1,\ldots,b_k)) \\ \vdots \\ MinS_k((b_1,..,b_k)) \text{et} MinS'_k((b_1,..,b_k)) \end{cases}$$

On remarque que :

$$\begin{cases} \frac{a_1 \cdot N}{N^2} + \frac{2a_2 \cdot (b_2+\cdots+b_K)}{N^2} + \ldots + \frac{ka_kb_k}{N^2} > \frac{a_1 \cdot N}{N^2} \\ \frac{a_1 \cdot N}{2N(b_2+b_3\ldots+b_k)} + \frac{2a_2 \cdot (b_2+\cdots+b_K)}{2N(b_2+b_3\ldots+b_k)} + \ldots > \frac{a_1 \cdot N}{3N^2} + \frac{2 \cdot a_2 \cdot N}{3N^2} + \frac{3 \cdot a_3 \cdot N}{3N^2} \\ \vdots \\ \frac{a_1 \cdot N}{kNb_k} + \frac{2a_2 \cdot (b_2+\cdots+b_K)}{kNb_k} + \cdots + \frac{k \cdot a_k \cdot N}{kNb_k} > \frac{a_1 \cdot N}{kN^2} + \frac{2 \cdot a_2 \cdot N}{kN^2} + \cdots + \frac{k \cdot a_k \cdot N}{kN^2} \end{cases}$$

D'ou

$$\begin{cases} MinS_1((b_1,\ldots,b_k)) > MinS'_1((b_1,\ldots,b_k)) \\ MinS_2((b_1,\ldots,b_k)) > MinS'_2((b_1,\ldots,b_k)) \\ \vdots \\ MinS_k((b_1,..,b_k)) > MinS'_k((b_1,..,b_k)) \end{cases}$$

Donc Brahim doit necessairement jouer tous ses jetons sur une case pour atteindre le minimum possible

---

3.1.2.0 conj-2

Montrons que pour atteindre le maximum du minimu abla doit equilibreer et egaliser les fonction S_{min}
Supposons, par l'absurde, que Abla ne joue pas l'égalité. Soit l'ensemble où Abla joue pour l'égalité :

$$A' = (a'_1, a'_2, \ldots, a'_k)$$

et l'autre ensemble, supposé meilleur que celui ci-dessus :

$$A = (a_1, a_2, \ldots, a_k)$$

Puisque l'ensemble A' égalise les scores et que l'autre ensemble A est supposé meilleur, la différence entre les scores de A et A' doit être strictement positive. En effet, si cette différence n'était pas positive, Brahim trouverait un score inférieur au nôtre. Les inégalités deviennent donc :

$$\begin{cases} a_1 - a'_1 > 0 \\ a_1 + 2a_2 - a'_1 - 2a'_2 > 0 \\ a_1 + 2a_2 + 3a_3 - a'_1 - 2a'_2 - 3a'_3 > 0 \\ \vdots \\ a_1 + 2a_2 + 3a_3 + \cdots + ka_k - a'_1 - 2a'_2 - 3a'_3 - \cdots - ka'_k > 0 \end{cases}$$

Posons $\alpha_i = a_i - a'_i$, les inégalités deviennent :

$$\begin{cases} \alpha_1 > 0 \\ \alpha_1 + 2\alpha_2 > 0 \\ \alpha_1 + 2\alpha_2 + 3\alpha_3 > 0 \\ \vdots \\ \alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots + k\alpha_k > 0 \end{cases}$$

Nous avons également :

$$\alpha_1 + \alpha_2 + \alpha_3 + \cdots + \alpha_k = 0$$

car $a_1 + a_2 + a_3 + \cdots + a_k = a'_1 + a'_2 + a'_3 + \cdots + a'_k$, et donc :

$$a_1 - a'_1 + a_2 - a'_2 + a_3 - a'_3 + \cdots + a_k - a'_k = 0.$$

Cette égalité est très intéressante, car si l'on trouve des coefficients convenables pour chaque inégalité et qu'on fait leur somme de manière à obtenir 0, nous arriverons à une absurdité. Mais existe-t-il ces coefficients ?

Posons le coefficient de la i-ème inégalité par C_i, c'est-à-dire :

$$C_i \cdot (\alpha_1 + 2\alpha_2 + \cdots + i\alpha_i) > 0.$$

Nous cherchons à trouver les propriétés de ces coefficients. Cela donne les inégalités suivantes :

$$\begin{cases} C_1 \cdot (\alpha_1) > 0, \\ C_2 \cdot (\alpha_1 + 2\alpha_2) > 0, \\ C_3 \cdot (\alpha_1 + 2\alpha_2 + 3\alpha_3) > 0, \\ \vdots \\ C_k \cdot (\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots + k\alpha_k) > 0. \end{cases}$$

En faisant la somme de toutes ces inégalités, l'équation devient :

$$(C_1 + C_2 + C_3 + \cdots + C_k)\alpha_1 + 2(C_2 + C_3 + \cdots + C_k)\alpha_2 + 3(C_3 + C_4 + \cdots + C_k)\alpha_3 + \cdots + kC_k\alpha_k = 0.$$

Nous cherchons à trouver les coefficients tels que :

$$\left[\begin{array}{c} C_1 + C_2 + C_3 + \cdots + C_k = 2(C_2 + C_3 + \cdots + C_k), \\ 2(C_2 + C_3 + \cdots + C_k) = 3(C_3 + C_4 + \cdots + C_k), \\ 3(C_3 + C_4 + \cdots + C_k) = 4(C_4 + C_5 + \cdots + C_k), \\ \vdots \\ (k-1)(C_{k-1} + C_k) = kC_k \end{array} \right]$$

Si l'on prend deux égalités successives :

$$m(C_m + C_{m+1} + \cdots + C_k) = (m+1)(C_{m+1} + C_{m+2} + \cdots + C_k),$$

$$(m+1)(C_{m+1} + C_{m+2} + \cdots + C_k) = (m+2)(C_{m+2} + C_{m+3} + \cdots + C_k),$$

cela revient à :

$$mC_m + m(C_{m+1} + C_{m+2} + \cdots + C_k) = (m+1)(C_{m+1} + C_{m+2} + \cdots + C_k),$$

$$(m+1)C_{m+1} + (m+1)(C_{m+2} + \cdots + C_k) = (m+2)(C_{m+2} + \cdots + C_k).$$

Cela donne :

$$mC_m = C_{m+1} + C_{m+2} + \cdots + C_k.$$

En faisant la différence, on obtient :

$$mC_m - (m+1)C_{m+1} = C_{m+1},$$

ce qui est équivalent à :

$$C_{m+1} = \frac{m}{m+2} C_m.$$

On remarque que si C_1 est positif, alors tous les coefficients sont positifs. Ainsi, puisque l'on est assuré que les coefficients existent, il existe un seul terme des inégalités qui est négatif ou nul, ce qui est absurde.

Par conséquent, Abla doit absolument jouer en respectant l'égalité pour gagner.

3.1.2.0 gen-formule

On sait que :

$$S((b_1,...,b_k)) = \frac{\sum_{i=1}^{K} i \cdot a_i \cdot (b_i + b_{i+1} + \ldots + b_K)}{V((b_1,...,b_k))}$$

$$S((b_1,...,b_k)) = \frac{a_1 \cdot (b_1 + b_2 + \cdots + b_K) + 2a_2 \cdot (b_2 + \cdots + b_K) + \cdots + K \cdot a_K \cdot b_K}{V((b_1,...,b_k))}$$

$$S((b_1,...,b_k)) = \frac{a_1 \cdot N + \cdots + K \cdot a_K \cdot b_K}{V((b_1,...,b_k))}$$

On sait que d'aprés **conj-1**:

$$\begin{cases} \text{Si } b_1 = N : & V_1((b_1,\ldots,b_k)) = N^2 \\ \text{Si } b_2 = N : & V_2((b_1,\ldots,b_k)) = 2N^2 \\ \text{Si } b_3 = N : & V_3((b_1,\ldots,b_k)) = 3N^2 \\ \vdots & \\ \text{Si } b_k = N : & V_k((b_1,\ldots,b_k)) = kN^2 \end{cases}$$

Donc :

$$\begin{cases} \text{Si } b_1 = N : & MinS((a_1,\ldots,a_k)(b_1,\ldots,b_k)) = \frac{a_1\cdot N+\cdots+a_K\cdot b_K}{N^2} & \Leftrightarrow & MinS_1((b_1,\ldots,b_k)) = \frac{a_1\cdot N}{N^2} \\ \text{Si } b_2 = N : & MinS((a_1,\ldots,a_k)(b_1,\ldots,b_k)) = \frac{a_1\cdot N+\cdots+a_K\cdot b_K}{2N^2} & \Leftrightarrow & MinS_2((b_1,\ldots,b_k)) = \frac{a_1\cdot N}{2N^2} + \frac{2\cdot a_2\cdot N}{2N^2} \\ \text{Si } b_3 = N : & MinS((a_1,..,a_k)(b_1,..,b_k)) = \frac{a_1\cdot N+\cdots+a_K\cdot b_K}{3N^2} & \Leftrightarrow & MinS_3((b_1,..,b_k)) = \frac{a_1\cdot N}{3N^2} + \frac{2\cdot a_2\cdot N}{3N^2} + \frac{3\cdot a_3\cdot N}{3N^2} \\ \vdots & & & \\ \text{Si } b_k = N : & MinS((a_1,..,a_k)(b_1,..,b_k)) = \frac{a_1\cdot N+\cdots+a_K\cdot b_K}{kN^2} & \Leftrightarrow & MinS_k((b_1,..,b_k)) = \frac{a_1\cdot N}{kN^2} + \frac{2\cdot a_2\cdot N}{kN^2} + \cdots + \frac{k\cdot a_k\cdot N}{kN^2} \end{cases}$$

D'prés **conj-2** on a prouver que abla doit equilibrer toutes les fonction Smin donc on a le systeme suivant :

$$\begin{bmatrix} S_1((b_1,\ldots,b_k)) = S_2((b_1,\ldots,b_k)) \\ S_2((b_1,\ldots,b_k)) = S_3((b_1,\ldots,b_k)) \\ \vdots \\ S_{k-1}((b_1,\ldots,b_k)) = S_k((b_1,\ldots,b_k)) \\ a_1 + a_2 + a_3 + \cdots + a_k = N \end{bmatrix}$$

Donc :

$$\begin{bmatrix} \frac{a_1\cdot N}{N^2} = \frac{a_1\cdot N}{2N^2} + \frac{2\cdot a_2\cdot N}{2N^2} \\ \frac{a_1\cdot N}{2N^2} + \frac{2\cdot a_2\cdot N}{2N^2} = \frac{a_1\cdot N}{3N^2} + \frac{2\cdot a_2\cdot N}{3N^2} + \frac{3\cdot a_3\cdot N}{3N^2} \\ \vdots \\ \frac{a_1\cdot N}{(k-1)N^2} + \frac{2\cdot a_2\cdot N}{(k-1)N^2} + \cdots + \frac{(k-1)\cdot a_{k-1}\cdot N}{(k-1)N^2} = \frac{a_1\cdot N}{kN^2} + \frac{2\cdot a_2\cdot N}{kN^2} + \cdots + \frac{k\cdot a_k\cdot N}{kN^2} \\ a_1 + a_2 + a_3 + \cdots + a_k = N \end{bmatrix}$$

Donc on simplifie par N :

$$\left[\begin{array}{c} \frac{a_1}{N} = \frac{a_1}{2N} + \frac{a_2}{N} \\ \frac{a_1}{2N} + \frac{a_2}{N} = \frac{a_1}{3N} + \frac{2\cdot a_2}{3N} + \frac{a_3}{N} \\ \vdots \\ \frac{a_1}{(k-1)N} + \frac{2\cdot a_2}{(k-1)N} + \cdots + \frac{a_{k-1}}{N} = \frac{a_1}{kN} + \frac{2\cdot a_2}{kN} + \cdots + \frac{a_k}{N} \\ a_1 + a_2 + a_3 + \cdots + a_k = N \end{array}\right]$$

Puis on multiplie le tous par N :

$$\left[\begin{array}{c} a_1 = \frac{a_1}{2} + a_2 \\ \frac{a_1}{2} + a_2 = \frac{a_1}{3} + \frac{2\cdot a_2}{3} + a_3 \\ \vdots \\ \frac{a_1}{(k-1)} + \frac{2\cdot a_2}{(k-1)} + \cdots + a_{k-1} = \frac{a_1}{k} + \frac{2\cdot a_2}{k} + \cdots + a_k \\ a_1 + a_2 + a_3 + \cdots + a_k = N \end{array}\right]$$

On passe tous les membres a un coté et on simplifie:

$$(1)\left[\begin{array}{c} a_1 = 2a_2 \\ 2a_2 = 3a_3 \\ \vdots \\ (k-1)a_{k-1} = ka_k \\ a_1 + a_2 + a_3 + \cdots + a_k = N \end{array}\right]$$

Donc maintenant on a la relation suivante :

$$\begin{aligned} & a_1 + a_2 + a_3 + ... + a_k = N \\ \Leftrightarrow\ & a_1 + \frac{a_1}{2} + \frac{a_1}{3} + ... + \frac{a_1}{k} = N \\ \Leftrightarrow\ & \sum_{i=1}^{k} \frac{a_1}{j} = N \\ \Leftrightarrow\ & a_1 \cdot \sum_{j=1}^{k} \frac{1}{j} = N \\ \Leftrightarrow\ & a_1 = \frac{N}{\sum_{j=1}^{k} \frac{1}{j}} \end{aligned}$$

Et puisqu on sait d'aprés (1) que $a_1 = k \cdot a_k$ donc :

$$k \cdot a_k = \frac{N}{\sum_{j=1}^{k} \frac{1}{j}}$$

$$\Leftrightarrow \boxed{a_k = \frac{N}{k \cdot \sum_{j=1}^{k} \frac{1}{j}}}$$

3.1.2.0 Nadi.py

Cette relation nous donne les a_k necessaire pour chaque case pour que abla atteint le maximum du minimum quelque soit la configuration que brahim va jouer mais ele est plus exacte dans $\mathbb{R}$ que dans $\mathbb{N}$ a cause de l'arrondissment des resultat comme le cas dans la question 3-a ce qui necessite l'etude de tous les cas
Alors cherchons un algorithme qui va faire cela :

Algorithm 1 Algorithme pour trouver la stratégie optimale

K : Nombre de cases, N : Somme totale à répartir Meilleure configuration d'Abla, Valeur du meilleur score minimum, Meilleure configuration de Brahim **Étape 1 : Générer les configurations d'Abla** Calculer les fractions pondérées a_i pour $i \in \{1, \ldots, K\}$:

$$a_i = \frac{N}{\sum_{j=i}^{K} \frac{1}{j}}$$

Générer toutes les combinaisons d'arrondis (plancher ou plafond) des a_i telles que leur somme soit égale à N
Étape 2 : Générer les configurations spécifiques de Brahim Configurations de Brahim :

$$\{(N, 0, 0, \ldots, 0), (0, N, 0, \ldots, 0), \ldots, (0, 0, \ldots, N)\}$$

Étape 3 : Calculer le score R pour chaque paire (*Abla*, *Brahim*) configuration d'Abla configuration de Brahim Calculer $R(\text{Abla}, \text{Brahim})$:

$$R(\text{Abla}, \text{Brahim}) = \sum_{i=1}^{K} (i \cdot \text{Abla}[i] \cdot \sum_{j=i}^{K} \text{Brahim}[j])$$

Étape 4 : Calculer V pour chaque configuration de Brahim configuration de Brahim Calculer $V(\text{Brahim})$:

$$V(\text{Brahim}) = \max_{\text{Abla}} R(\text{Abla}, \text{Brahim})$$

Étape 5 : Optimiser la stratégie configuration d'Abla configuration de Brahim Calculer le score réduit $S(\text{Abla}, \text{Brahim})$:

$$S(\text{Abla}, \text{Brahim}) = \frac{R(\text{Abla}, \text{Brahim})}{V(\text{Brahim})}$$

Retenir la configuration d'Abla qui maximise le score minimum S sur toutes les configurations de Brahim
Retourner : Meilleure configuration d'Abla, Meilleur score minimum, Meilleure configuration de Brahim

maintenant traduisant le en code python :

```python
from itertools import product
from fractions import Fraction

K = int(input("Combien de cases on a k = "))
N = int(input("Entrez la valeur de N : "))

def generate_configurations(K, N):
    return [tuple(N if i == j else 0 for j in range(K)) for i in range(K)]


def R(abla_config, brahim_config):
    K = len(abla_config)
    result = 0
    for i in range(K):
        brahim_sum = sum(brahim_config[i:])
        result += (i + 1) * abla_config[i] * brahim_sum
```

```python
    return result

def V(brahim_config, K, N):
    max_value = 0
    abla_configs = generate_configurations(K, N)
    for abla_config in abla_configs:
        result = R(abla_config, brahim_config)
        max_value = max(max_value, result)
    return max_value

def calculer_ai_ensemble(k, N):
    somme_harmonique = sum(Fraction(N, j) for j in range(1, k + 1))
    a_values = [Fraction(N, i) / somme_harmonique for i in range(1, k + 1)]
    return a_values

def find_combinations(numbers, target_sum, multiplier):
    floor_ceiling = [(int((x * multiplier) // 1), int(-(-(x * multiplier) // 1))) for x in numbers]
    all_combinations = product(*(fc for fc in floor_ceiling))
    valid_combinations = [comb for comb in all_combinations if sum(comb) == target_sum]
    return valid_combinations

def find_optimal_strategy(K, N):
    a_values = calculer_ai_ensemble(K, N)
    custom_abla_configs = find_combinations(a_values, N, N)
    custom_abla_configs = [tuple(config) for config in custom_abla_configs]

    brahim_configs = generate_configurations(K, N)

    scores_min_abla = {}
    best_brahim_for_abla = {}

    for abla_config in custom_abla_configs:
        min_score = float('inf')
        best_brahim_config = None
        for brahim_config in brahim_configs:
            score_r = R(abla_config, brahim_config)
            v_value = V(brahim_config, K, N)
            if v_value > 0:
                s_value = score_r / v_value
                if s_value < min_score:
                    min_score = s_value
                    best_brahim_config = brahim_config
        scores_min_abla[abla_config] = min_score
        best_brahim_for_abla[abla_config] = best_brahim_config


    best_score_value = max(scores_min_abla.values())

    best_abla_configs = [
        abla_config for abla_config, score in scores_min_abla.items() if score == best_score_value
    ]
    best_brahim_configs = [
        best_brahim_for_abla[abla_config] for abla_config in best_abla_configs
    ]

    return best_abla_configs, best_score_value, best_brahim_configs, scores_min_abla

best_abla_configs, best_score_value, best_brahim_configs, scores_min_abla =
↪  find_optimal_strategy(K, N)

```

```python
print("\nMeilleures configurations d'Abla :")
for config in best_abla_configs:
    print(config)
print("\nValeur du meilleur score minimum :", best_score_value)
print("\nMeilleures configurations de Brahim contre ces configurations d'Abla :")
for config in best_brahim_configs:
    print(config)
print("\nScores minimums pour chaque configuration d'Abla :")
for config, score in scores_min_abla.items():
    print(f"Configuration {config} : Score minimum {score}")

```

REMARQUE : on a fait cela juste pour verifier que notre arrondissment et vraie et pour trouver tous les cas possibles des configurations on va le modifier a le trouver dans le presentation dans le jour decompetition mais il nous donne le score rechercher don si notre score avec le score de l'une des configuration et le meme donc notre configuration et vraie elle meme

3.1.3 Generalisation-Resolution

4/

- **K=3 et N=11** :

En utulisant la formule generale qu on a demontrer on a :

$$a_i = \frac{N}{i \cdot \sum_{j=1}^{k} \frac{1}{j}}$$

Donc $a_1 = 6$ et $a_2 = 3$ et $a_3 = 2$
D'ou abla doit jouer $\boxed{(6,3,2)}$ Pour brahim on sait que :

$$\begin{cases} \text{Si } b_1 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{N^2} \Leftrightarrow MinS_1((b_1,\dots,b_k)) = \frac{a_1}{N} = \frac{6}{11} \\ \text{Si } b_2 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{2N^2} \Leftrightarrow MinS_2((b_1,\dots,b_k)) = \frac{a_1}{2N} + \frac{a_2}{N} = \frac{6}{11} \\ \text{Si } b_3 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{3N^2} \Leftrightarrow MinS_3((b_1,\dots,b_k)) = \frac{a_1}{3N} + \frac{2 \cdot a_2}{3N} + \frac{a_3}{N} = \frac{7}{11} \end{cases}$$

Donc brahim doit jouer $b_3 = N = 11$
pour un score de $\boxed{S = 6/11}$
Ce cas ne necessite pas d'arrondissement car notre relation les donnent directement sous forme d'entier

---

- **K=5 et N=20** :

En utulisant la formule generale qu on a demontrer on a :

$$a_i = \frac{N}{i \cdot \sum_{j=1}^{k} \frac{1}{j}}$$

Donc $a_1 = 9$ et $a_2 = 4$ et $a_3 = 3$ et $a_4 = 2$ et $a_5 = 2$ des resultat apres arrondissment
D'ou abla doit jouer $\boxed{(9,4,3,2,2)}$ Pour brahim on sait que :

$$\begin{cases} \text{Si } b_1 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{N^2} \Leftrightarrow MinS_1((b_1,\dots,b_k)) = \frac{a_1}{N} = \frac{9}{20} \\ \text{Si } b_2 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{2N^2} \Leftrightarrow MinS_2((b_1,\dots,b_k)) = \frac{a_1}{2N} + \frac{a_2}{N} = \frac{17}{40} \\ \text{Si } b_3 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{3N^2} \Leftrightarrow MinS_3((b_1,\dots,b_k)) = \frac{a_1}{3N} + \frac{2 \cdot a_2}{3N} + \frac{a_3}{N} = \frac{13}{30} \\ \text{Si } b_4 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{4N^2} \Leftrightarrow MinS_4((b_1,\dots,b_k)) = \frac{a_1}{4N} + \dots = \frac{17}{40} \\ \text{Si } b_5 = N: \quad MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1 \cdot N + \dots + K \cdot a_K \cdot b_K}{5N^2} \Leftrightarrow MinS_5((b_1,\dots,b_k)) = \frac{a_1}{5N} + \dots = \frac{11}{25} \end{cases}$$

Donc brahim doit jouer $b_2 = N = 20$ ou $b_3 = N = 20 \Leftrightarrow \boxed{(0, 20, 0, 0, 0) ou (0, 0, 0, 20, 0)}$
pour un score de $\boxed{S = 17/40 = 0.425}$
verifions se score par notre algorithem d'arrondissement :

```
heisenberg@Heisenberg:~/Desktop$ python3 hada.py
Combien de cases on a k = 5
Entrez la valeur de N : 20

Configurations optimales pour Abla (maximisant le minimum de S) :
Configuration : (9, 4, 3, 2, 2)
Configuration : (9, 4, 3, 3, 1)

Valeur du maximum des minimums : 0.425

Scores minimums pour toutes les configurations d'Abla :
Configuration (8, 4, 3, 3, 2) : Score minimum 0.4
Configuration (8, 5, 2, 3, 2) : Score minimum 0.4
Configuration (8, 5, 3, 2, 2) : Score minimum 0.4
Configuration (8, 5, 3, 3, 1) : Score minimum 0.4
Configuration (9, 4, 2, 3, 2) : Score minimum 0.38333333333333336
Configuration (9, 4, 3, 2, 2) : Score minimum 0.425
Configuration (9, 4, 3, 3, 1) : Score minimum 0.425
Configuration (9, 5, 2, 2, 2) : Score minimum 0.4125
Configuration (9, 5, 2, 3, 1) : Score minimum 0.4166666666666667
Configuration (9, 5, 3, 2, 1) : Score minimum 0.41
heisenberg@Heisenberg:~/Desktop$
```

Figure 1: resultat du code python qui prouve que notre score finale est vraie

---

- **K=10 et N=100** :

En utulisant la formule generale qu on a demontrer on a :

$$a_i = \frac{N}{i \cdot \sum_{j=1}^{k} \frac{1}{j}}$$

Donc $a_1 = 34$ et $a_2 = 17$ et $a_3 = 11$ et $a_4 = 9$ et $a_5 = 7$ et $a_6 = 6$ et $a_7 = 5$ et $a_8 = 4$ et $a_9 = 4$ et $a_10 = 3$ des resultat apres arrondissment
D'ou abla doit jouer $\boxed{(34, 17, 11, 9, 7, 6, 5, 4, 4, 3)}$

Pour brahim on sait que :

$$\begin{cases} \text{Si } b_1 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{N} = 0.34 \\ \text{Si } b_2 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{2N} + \frac{a_2}{N} = 0.34 \\ \text{Si } b_3 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{3N} + \frac{2\cdot a_2}{3N} + \frac{a_3}{N} = \frac{101}{300} \\ \text{Si } b_4 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{4N} + \cdots = \frac{137}{400} \\ \text{Si } b_5 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{5N} + \cdots = \frac{43}{125} \\ \text{Si } b_6 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{6N} + \cdots = \frac{26}{75} \\ \text{Si } b_7 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{7N} + \cdots = \frac{243}{700} \\ \text{Si } b_8 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{8N} + \cdots = \frac{11}{32} \\ \text{Si } b_9 = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{9N} + \cdots = \frac{311}{900} \\ \text{Si } b_{10} = N : & MinS((a_1,\dots,a_k)(b_1,\dots,b_k)) = \frac{a_1}{10N} + \cdots = \frac{341}{1000} \end{cases}$$

Donc brahim doit jouer $b_1 = N = 100$ ou $b_2 = N = 100 \Leftrightarrow \boxed{(100,0,0,0,0,0,0,0,0,0)ou(0,100,0,0,0,0,0,0,0,0)}$ pour un score de $\boxed{S = 0.34}$
verifions se score par notre algorithem d'arrondissement :

```
heisenberg@Heisenberg:~/Desktop$ python3 hada.py
Combien de cases on a k = 10
Entrez la valeur de N : 100

Configurations optimales pour Abla (maximisant le minimum de S) :
Configuration : (34, 17, 12, 8, 7, 6, 5, 4, 4, 3)

Valeur du maximum des minimums : 0.34

Scores minimums pour toutes les configurations d'Abla :
Configuration (34, 17, 11, 8, 6, 6, 5, 5, 4, 4) : Score minimum 0.326
Configuration (34, 17, 11, 8, 7, 5, 5, 5, 4, 4) : Score minimum 0.33
Configuration (34, 17, 11, 8, 7, 6, 4, 5, 4, 4) : Score minimum 0.3314285714285714
Configuration (34, 17, 11, 8, 7, 6, 5, 4, 4, 4) : Score minimum 0.3325
Configuration (34, 17, 11, 8, 7, 6, 5, 5, 3, 4) : Score minimum 0.3325
Configuration (34, 17, 11, 8, 7, 6, 5, 5, 4, 3) : Score minimum 0.3325
Configuration (34, 17, 11, 9, 6, 5, 5, 5, 4, 4) : Score minimum 0.3283333333333333
Configuration (34, 17, 11, 9, 6, 6, 4, 5, 4, 4) : Score minimum 0.33
Configuration (34, 17, 11, 9, 6, 6, 5, 4, 4, 4) : Score minimum 0.334
Configuration (34, 17, 11, 9, 6, 6, 5, 5, 3, 4) : Score minimum 0.334
Configuration (34, 17, 11, 9, 6, 6, 5, 5, 4, 3) : Score minimum 0.334
Configuration (34, 17, 11, 9, 7, 5, 4, 5, 4, 4) : Score minimum 0.32857142857142857
Configuration (34, 17, 11, 9, 7, 5, 5, 4, 4, 4) : Score minimum 0.33625
```

Figure 2: resultat du code python qui prouve que notre score finale est vraie

-REMARQUE!: ces configurations ne sontpas les seuls a donner ce score im existe d'autres qu'on va ajouter avec un code python dans le jour de la competition .

5/
Dans cette partie on a N=1 donc notre formule generale devient :

$$a_i = \frac{1}{i \cdot \sum_{j=1}^{k} \frac{1}{j}}$$

-REMARQUE : Dans cette question Brahim peut jouer sur n importe quelle case tous ces jetons a conditions qu il les met tous sur une seul case pour atteindre le minimum . Donc tout au long de cette question brahim peut jouer $b_1 = 1$ ou $b_2 = 1$ $b_3 = 1$ ou $b_4 = 1$ .... $b_k = 1$

- Si **k=3**
  on a donc d apres la formule generale $a_1 = 6/11$ et $a_2 = 3/11$ et $a_3 = 2/11$
  D'ou abla doit jouer $\boxed{(\frac{6}{11}, \frac{3}{11}, \frac{2}{11})}$ Pour un score
  $$\boxed{S((a_1, ...a_3)(b_1, ..., b_3)) = \frac{a_1}{N} = \frac{6}{11}}$$

- Si **k=5**
  on a donc d apres la formule generale $a_1 = 60/137$ et $a_2 = 30/137$ et $a_3 = 20/137$ et $a_4 = 15/137$ et $a_5 = 12/137$
  D'ou abla doit jouer $\boxed{(\frac{60}{137}, \frac{30}{137}, \frac{20}{137}, \frac{15}{137}, \frac{12}{137})}$ Pour un score
  $$\boxed{S((a_1, ...a_5)(b_1, ..., b_5)) = \frac{a_1}{N} = \frac{60}{137}}$$

- Si **k=10** on a donc d apres la formule generale
  $a_1 = \frac{2520}{7381}$ ou $a_2 = \frac{1260}{7381}$ ou $a_3 = \frac{840}{7381}$ ou $a_4 = \frac{630}{7381}$ ou $a_5 = \frac{504}{7381}$ ou $a_6 = \frac{420}{7381}$ ou $a_7 = \frac{360}{7381}$ ou $a_8 = \frac{315}{7381}$ ou $a_9 = \frac{280}{7381}$ ou $a_{10} = \frac{252}{7381}$
  D'où Abla doit jouer :
  $$\boxed{\left(\frac{2520}{7381}, \frac{1260}{7381}, \frac{840}{7381}, \frac{630}{7381}, \frac{504}{7381}, \frac{420}{7381}, \frac{360}{7381}, \frac{315}{7381}, \frac{280}{7381}, \frac{252}{7381}\right)}$$
  pour un Score
  $$\boxed{S((a_1, ...a_10)(b_1, ..., b_10)) = \frac{a_1}{N} = \frac{2520}{7381}}$$

- Si **k=100** dans ce cas je vais pas calculer manuelement la configuration de abla pour cela je vais utiliser un algorithme sous forme du code python qui va utliser la relation de generalisation pour nous donner la configuration :

```python
from fractions import Fraction
k = int(input("Combien de case , k = "))
def calculer_ai_ensemble(k):
    somme_harmonique = sum(Fraction(1, j) for j in range(1, k + 1))
    a_values = [Fraction(1, i) / somme_harmonique for i in range(1, k + 1)]
    return a_values

resultat = calculer_ai_ensemble(k)
score_a1 = resultat[0]

resultat_str = [f"{val.numerator}/{val.denominator}" for val in resultat]
score_a1_str = f"{score_a1.numerator}/{score_a1.denominator}"
score_a1_approx = float(score_a1)

print(resultat_str)
print(f"Score = {score_a1_str} (approx: {score_a1_approx:.5f})")
print results
```

6/ Pour les autres pistes de recherche je propose l algorithme suivant :

algo-2

L algo-2 est un algorithme qui est capable de calculer toutes les possibilités des ensemble que abla et brahim peuvent jouer a condition que $N \in \mathrm{N}$ on lui donne un nombre de case comme variable et le nombre de jetons puis il calcule toutes les combinaison posssible de abla et brahim puis il est utulisent dans la relation initial . Il va fixé une configuration de abla puis il va calculer les Smin pour toutes les configurations que brahim peut jouer et looper cette operation pour toutes les configurations possible de abla en fin il va comparer toutes les Smin pour trouver un Max(Smin) et les configuration optimale . On a traduit cet algorithme au python :

```python
from itertools import product

def generate_configurations(K, N):

    configurations = [
        config for config in product(range(N + 1), repeat=K) if sum(config) == N
    ]
    return configurations

def R(abla_config, brahim_config):

    K = len(abla_config)  # Nombre de cases
    result = 0
    for i in range(K):
        brahim_sum = sum(brahim_config[i:])
        result += (i + 1) * abla_config[i] * brahim_sum
    return result

def V(brahim_config, K, N):

    max_value = 0
    abla_configs = generate_configurations(K, N)
    for abla_config in abla_configs:
        result = R(abla_config, brahim_config)
        max_value = max(max_value, result)
    return max_value

def find_optimal_strategy(K, N):

    abla_configs = generate_configurations(K, N)
    brahim_configs = generate_configurations(K, N)

    scores_min_abla = {}
    best_brahim_for_abla = {}

    for abla_config in abla_configs:
        min_score = float('inf')
        best_brahim_config = None
        for brahim_config in brahim_configs:
            score_r = R(abla_config, brahim_config)
            v_value = V(brahim_config, K, N)
            if v_value > 0:
                s_value = score_r / v_value
                if s_value < min_score:
                    min_score = s_value
                    best_brahim_config = brahim_config
        scores_min_abla[abla_config] = min_score
        best_brahim_for_abla[abla_config] = best_brahim_config

```

```
    best_abla_config = max(scores_min_abla, key=scores_min_abla.get)
    best_score_value = scores_min_abla[best_abla_config]
    best_brahim_config = best_brahim_for_abla[best_abla_config]

    return best_abla_config, best_score_value, best_brahim_config, scores_min_abla

# Demander les valeurs de K et N à l'utilisateur
try:
    K = int(input("Entrez le nombre de cases (K) : "))
    N = int(input("Entrez le nombre de jetons (N) : "))

    best_abla_config, best_score_value, best_brahim_config, scores_min_abla =
    ↪ find_optimal_strategy(K, N)

    print("\nMeilleure configuration d'Abla :", best_abla_config)
    print("Valeur du meilleur score minimum :", best_score_value)
    print("Meilleure configuration de Brahim contre cette configuration d'Abla :",
    ↪ best_brahim_config)
    print("\nScores minimums pour chaque configuration d'Abla :")
    for config, score in scores_min_abla.items():
        print(f"Configuration {config} : Score minimum {score}")

except ValueError:
    print("Veuillez entrer des entiers valides pour K et N.")


```

Ce code est fonctionnel le seul probleme est la performance il nessecite des hardware a performance elevé pour des grandes valeur de K et N , qui peuvent necesiter un ordinateur quantique pour cela on a cree le code **nadi.py**.
Et pour l' ensmble des reel R on peut utuliser un programme python plus efficace qui est basé sur nos demonstration précedente pour N=1 et K variable :

```
from fractions import Fraction
k = int(input("Combien de case , k = "))
def calculer_ai_ensemble(k):
    somme_harmonique = sum(Fraction(1, j) for j in range(1, k + 1))
    a_values = [Fraction(1, i) / somme_harmonique for i in range(1, k + 1)]
    return a_values

resultat = calculer_ai_ensemble(k)
score_a1 = resultat[0]

resultat_str = [f"{val.numerator}/{val.denominator}" for val in resultat]
score_a1_str = f"{score_a1.numerator}/{score_a1.denominator}"
score_a1_approx = float(score_a1)

print(resultat_str)
print(f"Score = {score_a1_str} (approx: {score_a1_approx:.5f})")
print results
```

Maintenant pour N jetons on a donc

```
from fractions import Fraction

k = int(input("Combien de cases, k = "))
N = int(input("Combien de jetons, N = "))

def calculer_ai_ensemble(k, N):
```

```python

    somme_harmonique = sum(Fraction(N, j) for j in range(1, k + 1))
    a_values = [Fraction(N, i) / somme_harmonique for i in range(1, k + 1)]
    return a_values

resultat = calculer_ai_ensemble(k, N)

score_a1 = resultat[0]

resultat_str = [f"{val.numerator}/{val.denominator}" for val in resultat]
score_a1_str = f"{score_a1.numerator}/{score_a1.denominator}"
score_a1_approx = float(score_a1)

print("Valeurs de a_i :", resultat_str)
print(f"Score a1 = {score_a1_str} (approximation: {score_a1_approx:.5f})")

```

Remarque : Ce programme n'est pas encore finalisé. Nous sommes en train de créer un outil python capable de résoudre tous les problèmes identifiés dans ce probléme. Il prendra en charge à la fois les nombres entiers et réels, et couvrira tous les concepts abordés précédemment.

3.2 Bibliographie

1. Serie-Harmonique
2. stackoverflow
3. Min-Max ALgo:
4. Deterministic games vs indermenstic games
5. theorie des jeux
6. python code for latex
7. Github
8. A lot of PDFs that I used but can't find now :)

4 Quoi pour qui ?

4.1 Résolution:

4.1.1 Formalisation

1/ Cas-1

$$\begin{cases} f_1(\{3\}) + f_2(\{1,2\}) = 7 \\ f_1(\{2\}) + f_2(\{1,3\}) = 8 \\ f_1(\{1\}) + f_2(\{2,3\}) = 4 \\ f_1(\{1,2\}) + f_2(\{3\}) = 7 \\ f_1(\{1,3\}) + f_2(\{2\}) = 6 \\ f_1(\{2,3\}) + f_2(\{1\}) = 5 \\ f_1(\{1,2,3\}) = 4, \\ f_2(\{1,2,3\}) = 5 \end{cases}$$

Ainsi, la répartition sera :

$$S_1 =\}, \quad S_2 = \{1,3\}$$

Cas-2

$$\begin{cases} f_1(\{1\}) + f_2(\{2,3\}) = 7 \\ f_1(\{2\}) + f_2(\{1,3\}) = 7 \\ f_1(\{1\}) + f_2(\{2,3\}) = 6 \\ f_1(\{1,2\}) + f_2(\{3\}) = 6 \\ f_1(\{1,3\}) + f_2(\{2\}) = 5 \\ f_1(\{2,3\}) + f_2(\{1\}) = 5 \\ f_1(\{1,2,3\}) = 4, \\ f_2(\{1,2,3\}) = 7 \end{cases}$$

On remarque qu'il existe trois répartitions optimales qui sont :

$$\begin{cases} S_1 = \{2\}, \quad S_2 = \{1,3\} \\ S_1 = \{1\}, \quad S_2 = \{2,3\} \\ S_1 = \emptyset, \quad S_2 = \{1,2,3\} \end{cases}$$

4.1.2 La d-concavité :

1/

1. Montrons que 1 **implique** 2.
prenons des ensembles aleatoire A et B tel que A,B $\subseteq U$ et B/A=$\{ x_1, x_2x_3....x_m\}=\chi_m$
on prends donc $\chi_i=\{x_1, x_2x_3....x_i\}$ pour i $\leq$ m
on pose donc $\begin{cases} \text{V= (A}\cap\text{B)}\cup\chi_i \\ \text{W= A}\cup\chi_i \end{cases}$
et on remplace dans la relation numero 1 avec x = x_{i+1} sachant que $\begin{cases} \text{x}_{i+1} \notin A \\ \text{x}_{i+1} \notin \text{A}\cap B \end{cases}$
ce qui nous donne :

$$f(W \cup \{x\}) - f(W) \leq f(V \cup \{x\}) - f(V)$$
$$f((A \cap B) \cup \chi_i \cup \{x_{i+1}\}) - f((A \cap B) \cup \chi_i) \geq f((A \cup \chi_i) \cup \{x_{i+1}\}) - f(A \cup \chi_i)$$

$$f((A\cap B)\cup\chi_{i+1})-f((A\cap B)\cup\chi_i)\geq f((A\cup\chi_{i+1}))-f(A\cup\chi_i)$$

en faisant la somme de i=0 a i=n-1 ; on obtient :

$$\sum_{k=0}^{m-1}(f((A\cap B)\cup\chi_{i+1})-f((A\cap B)\cup\chi_i))\geq\sum_{k=0}^{m-1}(f((A\cup\chi_{i+1}))-f(A\cup\chi_i))$$

par telescopage on trouve que :

$$f((A\cap B)\cup\chi_m)-f((A\cap B))\geq f((A\cup\chi_m))-f(A)$$

Or χ_m = B/A

Donc $\begin{cases}(B/A)\cup(A)=B\\(B/A)=A\cup B\end{cases}$

Alors :

$$f(A)-f(A\cap B)\geq f(A\cup B)-f(B)$$

D'ou on a :

$$\boxed{f(A)+f(B)\geq f(A\cup B)+f(A\cap B)}$$

2. Montrons que 2 **implique** 1. prenons $A\subseteq B$ et x $\in U/B$

on pose donc $\begin{cases}\text{V= A}\cup\{\text{x}\}\\\text{W= B}\end{cases}$

Donc :

$$f(V\cup W)+f(V\cap W)\leq f(V)+f(W).$$
$$f((A\cup\{x\})\cup B)+f((A\cup\{x\})\cap B)\leq f(A\cup\{x\})+f(B).$$

Dou on a :

$$\boxed{f(B\cup\{x\})-f(B)\leq f(A\cup\{x\})-f(A)}$$

Alors, d'après (1) et (2) (la double implication), on conclut que f sur un ensemble de base U est d-concave si et seulement si, pour tout $V,W\subseteq U$,

$$\boxed{f(V\cup W)+f(V\cap W)\leq f(V)+f(W)}$$

2/ Soit $x\in U/W$, alors :

$$\text{card}(V\cup\{x\})=n+1\quad\text{et}\quad\text{card}(W\cup\{x\})=m+1$$

L'inégalité :

$$n(n+1)<m(m+1)$$

équivaut à :

$$\frac{1}{n(n+1)}>\frac{1}{m(m+1)}$$

Ce qui donne :

$$\frac{1}{n}-\frac{1}{n+1}>\frac{1}{m}-\frac{1}{m+1}$$

Ce qui donne :

$$\boxed{f(V\cup\{x\})-f(V)>f(W\cup\{x\})-f(W)}$$

Donc ; f est d-concave.

3/Une fonction d-concave représente une situation où les gains ou les satisfactions ajoutés par la combinaison de ressources sont modérés et ne croissent pas de manière explosive. Autrement dit, si un enfant reçoit un cadeau alors qu'il possède déjà un ensemble de cadeaux, son bonheur ne sera pas le même que celui d'un autre enfant qui a moins de cadeaux ou qui n'en possède aucun.

4 Soit la fonction $f(S) = \sqrt{|S|}$, où $|S|$ est la cardinalité de l'ensemble $S \subseteq N$. Nous démontrons que cette fonction est sous-modulaire.

Pour qu'une fonction f définie sur les sous-ensembles d'un ensemble N soit sous-modulaire, il faut que pour tout $A \subseteq B \subseteq N$ et $x \notin B$, l'inégalité suivante soit vérifiée :

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B).$$

Calculons les valeurs marginales pour $f(S) = \sqrt{|S|}$:

1. Pour $A \subseteq N$, l'ajout de $x \notin A$ donne :

$$f(A \cup \{x\}) - f(A) = \sqrt{|A| + 1} - \sqrt{|A|}.$$

2. Pour $B \supseteq A$, l'ajout de $x \notin B$ donne :

$$f(B \cup \{x\}) - f(B) = \sqrt{|B| + 1} - \sqrt{|B|}.$$

Montrons que :

$$\sqrt{|A| + 1} - \sqrt{|A|} \geq \sqrt{|B| + 1} - \sqrt{|B|},$$

avec $|A| \leq |B|$.

La fonction $g(x) = \sqrt{x}$ est concave, car sa dérivée seconde est négative :

$$g''(x) = -\frac{1}{4x^{3/2}} < 0, \quad \text{pour } x > 0.$$

Pour une fonction concave, l'accroissement marginal $g(x+1) - g(x)$ décroît lorsque x augmente. Puisque $|A| \leq |B|$, on a :

$$\sqrt{|A| + 1} - \sqrt{|A|} \geq \sqrt{|B| + 1} - \sqrt{|B|}.$$

Ainsi, la fonction $f(S) = \sqrt{|S|}$ est sous-modulaire.

4.1.3 Un matroide :

5. (a) Si $V \in \mathcal{I}$ et $W \subseteq V$ alors $W \in \mathcal{I}$.
   Si $V \in \mathcal{I}$ et $W \subseteq V$, alors $\text{card}(W) < \text{card}(V)$
   Donc $\boxed{\text{W} \in \mathcal{I}}$
   (b)Si $V, W \in \mathcal{I}$ et $\text{card}(W) < \text{card}(V)$, alors il existe $x \in V$ tel que $W \cup \{x\} \in \mathcal{I}$.
   On a $\text{card}(W) < \text{card}(V)$, donc $\text{card}(W \cup \{x\}) \leq \text{card}(V) \leq k$, alors $W \cup \{x\} \in \mathcal{I}$.

6. Montrons que $(U, \mathcal{I})$ est une matroïde.

   (a) On a $\text{card}(V \cap E_i) \leq r_i$ et $W \subseteq V$, donc $\text{card}(W \cap E_i) \leq \text{card}(V \cap E_i) \leq r_i$. Par conséquent, $\boxed{\text{W} \in \mathcal{I}}$

   (b) Posons que $\forall x \in V \setminus W$, $\text{card}((W \cup \{x\}) \cap E_i) > r_i$. Prenons $V \setminus W = \{x_1, x_2, x_3, \ldots, x_p\}$. Alors, $\text{card}((W \cup x_1 \cup x_2 \cup x_3 \cup \cdots \cup x_p) \cap E_i) > r_i$. Cela implique que $\text{card}(V \cap E_i) > r_i$, mais $\text{card}(V \cap E_i) \leq r_i$ car $V \in \mathcal{I}$, ce qui est absurde.
   Donc, il existe un $x \in V$ tel que $\text{card}((W \cup \{x\}) \cap E_i) \leq r_i$. Par conséquent, $\boxed{\text{W} \cup \{x\} \in \mathcal{I}}$

4.1.4 Maximisez uniformément.

1/Supposons que $\text{card}(S^*) < k$. Donc, il existe $x \in U \setminus S^*$ tel que $S^* \cup \{x\} \subseteq U$ (car c'est une matroïde uniforme et $\text{card}(U) \geq k$).

Puisque f est une fonction strictement croissante, on a

$$f(S^* \cup \{x\}) > f(S^*).$$

Cela signifie que $S^* \cup \{x\}$ est une meilleure solution pour la maximisation, ce qui est une contradiction.

Ainsi, on doit avoir $\text{card}(S^*) = k$.
1/L'algorithme glouton ne donne pas toujours une solution optimale, car il s'intéresse à trouver un maximum local, ce qui peut l'empêcher de trouver le maximum global, atteint par la solution optimale.

Un exemple classique est le problème du rendu de monnaie. On veut former une somme S en utilisant le minimum possible de pièces. Considérons le système de pièces : $(100, 50, 20, 15, 2, 1)$ et $S = 321$.

L'algorithme glouton donnera :

$$S = 100 + 100 + 100 + 20 + 1,$$

ce qui nous donne 5 pièces. Cette solution correspond à la solution optimale.

Cependant, ce n'est pas toujours le cas. Prenons un autre exemple : $S = 63$. L'algorithme glouton donnera :

$$S = 50 + 2 + 2 + 2 + 2 + 2 + 2 + 1,$$

ce qui utilise 8 pièces. Or, une meilleure solution serait :

$$S = 20 + 20 + 20 + 2 + 1,$$

ce qui utilise 5 pièces.

Considérons maintenant un autre système de pièces : $(100, 50, 20, 10, 3)$. Pour $S = 181$, l'algorithme glouton ne trouve pas de solution, bien qu'il en existe une. L'algorithme glouton donnera :

$$S = 100 + 50 + 20 + 10,$$

mais après le 10, il n'y a pas de pièce pour 1.

Une solution correcte serait :

$$S = 100 + 50 + 10 + 3 + 3 + 3 + 3 + 3 + 3 + 3 = 181.$$

Ainsi, l'algorithme glouton ne trouve pas forcément la solution optimale

2/ soit $t \in \{1, 2, 3, \cdots k\}$

on prend $\begin{cases} \text{S}^*/S_t = \{x_1, x_2, \cdots x_m\} \\ \chi_i = \{x_1, x_2, \cdots x_i\ \} \end{cases}$
on sait que $f(S^*) \leq f(S^*) + f(S_t)$ (trivial car f est ≥ 0)
et par telescopage on arrive à :

$$f(S^*) \leq f(S_t) + \sum_{1}^{k} f(S_t \cup \chi_i) - f(S_t \cup \chi_{i-1})$$

et par D-concavité on peut l'ecrire sous la forme suivante :

$$f(S^*) \leq f(S_t) + \sum_{1}^{k} f(S_t \cup \{x_i\}) - f(S_t)$$

$$f(S^*) \leq f(S_t) + \sum_{1}^{k} \max_{i \in U} \left(f(S_t \cup \{i\}) - f(S_t)\right)$$

$$f(S^*) \leq f(S_t) + k * \max_{i \in U} \left(f(S_t \cup \{i\}) - f(S_t)\right)$$

$$\frac{1}{k} \cdot \left(f(S^*) - f(S_t)\right) \leq \max_{i \in U} \left(f(S_t \cup \{i\} - f(S_t))\right)$$

ce qui implique que

$$\boxed{\frac{1}{k} \left(f(S^*) - f(S_t)\right) \leq \max_{i \in U} \left(f(S_t \cup \{i\}) - f(S_t)\right)}$$

3/ On sait que $\max_{e \in U} \left[f(S_t \cup \{e\})\right] = f(S_{t+1})$. Ainsi, l'inégalité devient :

$$f(S^*) - f(S_t) \leq k \cdot \Big(f(S_{t+1}) - f(S_t)\Big).$$

Cela est équivalent à :

$$k \cdot \left(f(S^*) - f(S_{t+1})\right) \leq (k-1) \cdot \Big(f(S^*) - f(S_t)\Big).$$

Finalement, on obtient l'inégalité suivante :

$$f(S^*) - f(S_{t+1}) \leq \left(1 - \frac{1}{k}\right) \cdot \Big(f(S^*) - f(S_t)\Big).$$

Cela devient :

$$f(S_{t+1}) \geq f(S^*) - \left(1 - \frac{1}{k}\right) \cdot \Big(f(S^*) - f(S_t)\Big).$$

En itérant cette inégalité pour $t = 1, 2, \ldots, k-1$, on obtient :

$$f(S_k) \geq f(S^*) - \left(1 - \frac{1}{k}\right) \cdot \Big(f(S^*) - f(S_{k-1})\Big),$$

$$f(S_k) \geq f(S^*) - \left(1 - \frac{1}{k}\right)^2 \cdot \Big(f(S^*) - f(S_{k-2})\Big),$$

$$\vdots$$

$$f(S_k) \geq f(S^*) - \left(1 - \frac{1}{k}\right)^k \cdot f(S^*).$$

En utilisant la limite de la fonction $\left(1 - \frac{1}{x}\right)^x$ quand $x \to +\infty$, qui est égale à $\frac{1}{e}$, on obtient :

$$f(S_k) \geq f(S^*) - \frac{1}{e} \cdot f(S^*) = \left(1 - \frac{1}{e}\right) \cdot f(S^*).$$

Ainsi, on en déduit que :

$$\boxed{f(S_k) \geq \left(1 - \frac{1}{e}\right) \cdot f(S^*)}$$

et donc , l'algorithme nous donne au moins 1-1/e = 0.63212 ou $\boxed{63\%}$ de la solution optimale.

4.1.5 Toujours plus de maximisation

1/puisque la solution optimale est de cardianle maximale, et que la solution de l algorithme glouton est l union de chaque etape jusqua atteindre la contrainte k, et donc avoir k elemets , ce qui est le cardinale maximale , donc les deux ont le meme cardinale , ou $\boxed{|S*| = |Sg|}$
2/ on sait que pour tout $t \in \{1, \ldots, k\}$

$$\frac{1}{k} \cdot (f(S^*) - f(S_t)) \leq \max_{i \in U} (f(S_t \cup \{i\}) - f(S_t)))$$

$$\frac{1}{k} \cdot (f(S^*) - f(S_t)) \leq \max_{i \in U} (f(S_t \cup \{t\}) - f(S_t)))$$

$$\frac{1}{k} \cdot (f(S^*) - f(S_t)) \leq (f(S_{t+1}) - f(S_t)))$$

en faisant la somme de t = 0 à t = K - 1

$$\sum_{0}^{k-1} \frac{1}{k} \cdot (f(S^*) - f(S_t)) \leq \sum_{0}^{k-1} (f(S_{t+1}) - f(S_t)))$$

par telescopage on obtient

$$k \cdot \frac{1}{k} \cdot f(S^*) - \frac{1}{k} \cdot \sum_{0}^{k-1} f(S_t) \ \leq f(S_G) - f(\emptyset)$$

Donc on la note (1):

$$f(S^*) - f(S_G) \ \leq \frac{1}{k} \cdot \sum_{0}^{k-1} f(S_t)$$

Or on a $\forall t \in [|0; K-1|] : f(\mathrm{S_G}) \geq f(\mathrm{S_t})$ car f est une fonction croisssant et on a $|\mathrm{S}_g| > |\mathrm{S}_t|$

Donc on peut deduire que on la notant (2):

$$K \cdot f(S_G) \geq \sum_{0}^{k-1} f(S_t)$$

En faisant la somme de (1) et (2) :

$$(1) + (2) \Rightarrow f(S^*) - f(S_G) \leq \frac{1}{K} \cdot K \cdot f(S_G)$$

$$(1) + (2) \Rightarrow f(S^*) - f(S_G) \leq f(S_G)$$

D'ou :

$$\Rightarrow \boxed{f(S_G) \geq \frac{1}{2} \cdot f(S^*)}$$

Donc la performance de l'algorithme glouton, est égale à au moins la moitié de celle d'un algorithme optimal

4.1.6 Revenons à nos moutons

1/Notre problème peut être formulé comme un problème de maximisation **d'une fonction croissante, positive, et d-concave dans une matroïde de partition.**
La fonction est

$$f(S) = \sum_{i=1}^{m} f_i(S_i).$$

Avec $\begin{cases} \text{m le nombre des enfants} \\ S_i \text{ L'ensemble des cadeaux donnés à l'enfant i} \\ f_i \text{ la fonction de joie} \end{cases}$

On prend l'ensemble C des cadeaux de base U
Et la matroide $(C, \mathcal{I})$ et :

$$\boxed{\mathcal{I} = \{W \subseteq C \,:\, |W \cap E_i| \leq r_i \ \forall i \in [|1; m|]\}}$$

Avec : $\begin{cases} E_i \text{ L'ensemble finale des cadeaux donnés à l'enfant i} \\ r_i \text{ le nombre maximal donnés à un enfant i} \end{cases}$

Analyse d'injustice dans la répartition des cadeaux

Considérons les deux cas donnés dans l'énoncé pour les fonctions de satisfaction de Ismail et Omar :

- **Cas 1 :**

Ensemble des cadeaux S	$f_{\text{Ismail}}(S)$	$f_{\text{Omar}}(S)$
$\emptyset$	0	0
$\{1\}$	2	2
$\{2\}$	3	4
$\{3\}$	0	3
$\{1,2\}$	4	4
$\{1,3\}$	2	5
$\{2,3\}$	3	5
$\{1,2,3\}$	4	5

- **Cas 2 :**

Ensemble des cadeaux S	$f_{\text{Ismail}}(S)$	$f_{\text{Omar}}(S)$
$\emptyset$	0	0
$\{1\}$	1	2
$\{2\}$	2	4
$\{3\}$	1	3
$\{1,2\}$	3	5
$\{1,3\}$	2	5
$\{2,3\}$	3	6
$\{1,2,3\}$	4	7

Cas-1 Ainsi, la répartition sera :

$$S_1 = \{2\}, \quad S_2 = \{1,3\}$$

Cas-2

On remarque qu'il existe trois répartitions optimales qui sont :

$$\begin{cases} S_1 = \{2\}, \quad S_2 = \{1,3\} \\ S_1 = \{1\}, \quad S_2 = \{2,3\} \\ S_1 = \emptyset, \quad S_2 = \{1,2,3\} \end{cases}$$

Exemple d'injustice

Dans le **Cas 1**, supposons que les cadeaux sont distribués de manière à maximiser la satisfaction totale $f_{\text{Ismail}}(S_{\text{Ismail}})+f_{\text{Omar}}(S_{\text{Omar}})$ sans tenir compte de l'équité :

- Une solution optimale pour la satisfaction totale est :

$$S_{\text{Ismail}} = \{\emptyset, \quad S_{\text{Omar}} = \{1, 2, 3\},$$

donnant :

$$f_{\text{Ismail}}(S_{\text{Ismail}}) = 0, \quad f_{\text{Omar}}(S_{\text{Omar}}) = 7, \quad \text{Satisfaction totale} = 7.$$

Cependant, cette solution désavantage ismail sans cadeau par rapport à Ismail.

- Une autre solution serait :

$$S_{\text{Ismail}} = \{3\}, \quad S_{\text{Omar}} = \{1, 2\},$$

donnant :

$$f_{\text{Ismail}}(S_{\text{Ismail}}) = 0, \quad f_{\text{Omar}}(S_{\text{Omar}}) = 4, \quad \text{Satisfaction totale} = 4.$$

Cette solution maximise la satisfaction d'Omar, mais est extrêmement défavorable pour Ismail.

Algorithme glouton équitable

L'objectif de cet algorithme est d'attribuer des cadeaux à n enfants de manière équitable, en maximisant la satisfaction minimale à chaque étape.

Algorithm 2 Algorithme Glouton pour Maximiser la Satisfaction Minimale

```
procedure GloutonEquitable(n, m, v)
    Initialiser S_i ← ∅ pour tout i = 1, ..., n
    Initialiser U ← {1, 2, ..., m}                          ▷ Cadeaux non attribués
    while U ≠ ∅ do
        satisfaction_minimale ← −∞
        meilleur_cadeau, meilleur_enfant ← None, None
        for chaque c ∈ U do
            for chaque i ∈ {1, ..., n} do
                Calculer satisfaction_temp après attribution de c à S_i
                Calculer satisfaction_minimale_temp
                if satisfaction_minimale_temp > satisfaction_minimale then
                    satisfaction_minimale ← satisfaction_minimale_temp
                    meilleur_cadeau ← c
                    meilleur_enfant ← i
                end if
            end for
        end for
        Attribuer meilleur_cadeau à meilleur_enfant
        S_meilleur_enfant ← S_meilleur_enfant ∪ {meilleur_cadeau}
        Retirer meilleur_cadeau de U
    end while
    return S_1, S_2, ..., S_n
end procedure
```

comment fonctionne-t-il :

on a des entrés :
- n, le nombre d'enfants,
- m, le nombre de cadeaux, et
- $v_{i,j}$, une matrice $n \times m$ où $v_{i,j}$ représente la valeur (ou satisfaction) que l'enfant i attribue au cadeau j.

on va Attribuer chaque cadeau à un enfant, de sorte que la satisfaction minimale des enfants soit maximisée à chaque étape.
les etape de notre algo sont :

- Initialiser $S_i = \emptyset$ (l'ensemble des cadeaux reçus par chaque enfant i).
- Calculer l'ensemble des cadeaux non attribués U.
- **Pour chaque itération :**
  - Pour chaque cadeau $c \in U$, évaluer l'impact sur la satisfaction minimale si c est attribué à un enfant i.
  - Choisir le cadeau c et l'enfant i qui maximisent la satisfaction minimale parmi tous les enfants après attribution.
  - Ajouter c à S_i et retirer c de U.
- Répéter jusqu'à ce que tous les cadeaux soient attribués.

et finalement comme sortie on aura Une répartition $S_1, S_2, \ldots, S_n$ des cadeaux entre les n enfants.
code-python
traduisant maintenant notre algo en programme python

```python
def glouton_equitable(n, m, v):
    S = [[] for _ in range(n)]  # Cadeaux attribues à chaque enfant
    U = list(range(m))  # Cadeaux non attribués

    while U:
        satisfaction_minimale = -float('inf')
        meilleur_cadeau, meilleur_enfant = None, None

        # Chercheons le meilleur cadeau et enfant
        for c in U:
            for i in range(n):
                satisfaction_temp = [sum(v[i][gift] for gift in S[i]) for i in range(n)]
                satisfaction_temp[i] += v[i][c]
                min_satisfaction = min(satisfaction_temp)

                if min_satisfaction > satisfaction_minimale:
                    satisfaction_minimale = min_satisfaction
                    meilleur_cadeau = c
                    meilleur_enfant = i

        # On atribur le meilleur cadeau
        S[meilleur_enfant].append(meilleur_cadeau)
        U.remove(meilleur_cadeau)

    return S

# Exemple
def main():
    v = [
        [8, 6, 4, 7],  # Enfant 1
        [5, 9, 3, 6],  # Enfant 2
        [7, 3, 8, 5]   # Enfant 3
    ]
    n, m = 3, 4
    resultat = glouton_equitable(n, m, v)
    print("Répartition des cadeaux :", resultat)

if __name__ == "__main__":
    main()
```

Remarque on peut remplacer notre exemple par les valeur qu on veut

4.2 Bibliographie :

(a) Most-important-resource

5 Painville

5.1 Résolution:

5.1.1 Modèle mathématique

1/

Définition de la fonction de solde : Une fonction de solde $S : V \to \mathbb{Z}$ associe à chaque villageois $v \in V$ un entier $S(v)$, représentant le solde de pain de ce villageois.
Et :

- $S(v) > 0$: v a un surplus de pain.
- $S(v) = 0$: v est à l'équilibre.
- $S(v) < 0$: v est endetté.

Effet des opérations : Soit $\deg(v)$ le degré de v dans le graphe G (nombre total d'arêtes connectées à v, incluant les arêtes multiples). De plus, pour un voisin u_i, notons $\deg_{u_i v_i}$ le nombre d'arêtes reliant u_i et v_i et $S_0(v_i)$ le solde initial respectivement du villageois v_i et de chaque voisins u_i et $i \in \{1, 2, 3, 4, 5, 6, \cdots n \in \mathbb{N}\}$:

- *Mendicité (M_{vi}) :* Lorsque v_i demande une baguette de pain à chacun de ses amis :

$$\boxed{S(u_i) \mapsto S(u_i) - \deg_{u_i v_i} \quad \text{pour tous } u_i \text{ voisins de } v_i}$$

et :

$$\boxed{S(v_i) \mapsto S(v_i) + \deg(v_i)}$$

- *Charité (C_{vi}) :* Lorsque v donne une baguette de pain à chacun de ses amis :

$$\boxed{S(u_i) \mapsto S(u_i) + \deg_{u_i v_i} \quad \text{pour tous } u_i \text{ voisins de } v_i}$$

et :

$$\boxed{S(v_i) \mapsto S(v_i) - \deg(v_i)}$$

Problème à résoudre : Trouver une suite d'opérations $\{M_{vi}, C_{vi}\}$ pour chaque $v_i \in V$, de sorte que tous les soldes deviennent non négatifs, c'est-à-dire $S(v_i) \geq 0$ pour tout $v_i \in V$.

$$\boxed{Une fonction S \text{ est dite } \textit{résolvable} \text{ si cette suite d'opérations existe.}}$$

2/ Les exemples :
Exemple 1 (résolvable) :
Considérons un graphe G avec $V = \{v_1, v_2\}$ et une arête unique entre v_1 et v_2. Si $S(v_1) = 3$ et $S(v_2) = -1$, une solution consiste en une charité de v_1 (C_{v_1}), qui modifie les soldes comme suit :

$$S(v_1) = 3 - 1 = 2, \quad S(v_2) = -1 + 1 = 0.$$

Après cette opération, tous les soldes sont non négatifs.

Figure 3: Exemple 1 : Avant et après la charité.

Exemple 2 (résolvable avec plusieurs arêtes) :
Considérons un graphe G avec $V = \{v_1, v_2\}$ et 3 arêtes entre v_1 et v_2. Si $S(v_1) = 5$ et $S(v_2) = -4$, une solution consiste en une charité de v_1 (C_{v_1}), qui modifie les soldes comme suit :

$$S(v_1) = 5 - 3 = 2, \quad S(v_2) = -4 + 3 = -1.$$

Ensuite, une nouvelle charité de v_1 (C_{v_1}) résout la situation :

$$S(v_1) = 2 - 1 = 1, \quad S(v_2) = -1 + 1 = 0.$$

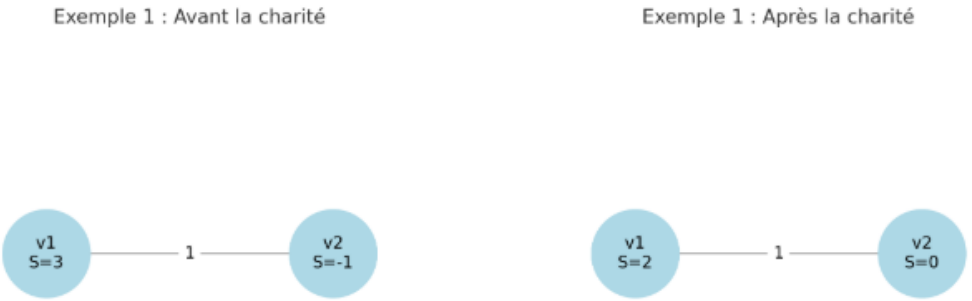


Figure 4: Exemple 1 : Avant et après la charité.

Exemple 3 (non résolvable) :
Considérons un graphe G avec $V = \{v_1, v_2\}$ et une arête unique entre v_1 et v_2. Si $S(v_1) = -2$ et $S(v_2) = -1$, il est impossible d'effectuer une suite d'opérations pour rendre les soldes non négatifs, car aucun villageois n'a de surplus de pain à redistribuer.

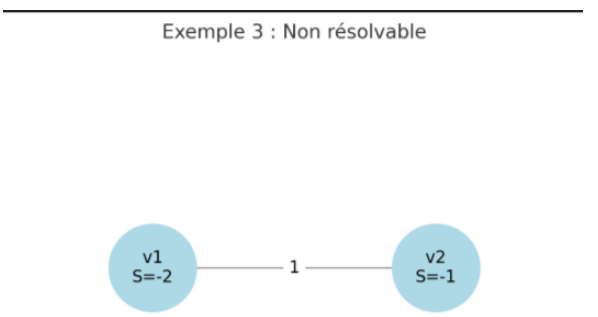


Figure 5: Exemple 1 : Avant et après la charité.

3/

Condition suffisante : Une fonction de solde S n'est pas résolvable si :

$$\boxed{\sum_{v \in V} S(v) < 0}$$

Cette condition exprime que le pain total disponible dans le village est insuffisant pour compenser les dettes. En effet, même avec une redistribution optimale, le pain manquant empêche d'atteindre un solde non négatif pour tous.

Condition nécessaire : La condition $\sum_{v \in V} S(v) < 0$ est suffisante mais **pas necessaire**. Un exemple où S est non résolvable est le suivant :

- Graphe G : $V = \{v_1, v_2, v_3\}$ avec des arêtes (v_1, v_2) et (v_2, v_3).
- Solde initial : $S(v_1) = 1, S(v_2) = -1, S(v_3) = 0$.

Bien que $\sum_{v \in V} S(v) \geq 0$, v_2 le graphe ne peut pas etre resolvable

Conclusion : La condition $\sum_{v \in V} S(v) < 0$ est suffisante pour la non-résolvabilité, mais d'autres conditions sont également non résolvable.

5.1.2 Opérations sur des fonctions de solde

1/ $F(S)$ est résolvable si l'on trouve un ensemble d'opérations de charité et de mendicité qui rend le solde final de chaque villageois non négatif :

$$\forall v \in V, \quad S(v) \geq 0.$$

Définissons l'opération de mendicité pour un villageois v_i comme suit :

$$M : \begin{cases} s_j \mapsto s_j + \deg(v_i) & \text{si } j = i, \\ s_j \mapsto s_j - e(v_i, j) & \text{si } j \neq i, \end{cases}$$

où : - $\deg(v_i) = \sum_{u \in V} e(v_i, u)$ est le degré total de v_i, soit le nombre total de pains échangés par v_i dans le graphe G,
- $e(v_i, j)$ est le nombre d'arêtes (ou la "force" de connexion) entre v_i et v_j.

Considérons maintenant la charité de tous les villageois sauf v_i :

$$C : \begin{cases} s_j \mapsto s_j + \sum_{u \in V} e(v_j, u) = s_j + \deg(v_j) & \text{si } j = i, \\ s_j \mapsto s_j - \sum_{u \in V} e(v_j, u) + \sum_{u \in V \setminus \{v_i\}} e(v_j, u) = s_j - e(v_j, v_i) & \text{si } j \neq i. \end{cases}$$

En comparant les deux définitions, remarquons une **similarité clé** : - La mendicité M pour un villageois v_i est **l'opération inverse** d'une charité globale sauf pour v_i. Cela signifie que toute opération de mendicité peut être formulée comme une combinaison d'opérations de charité, et vice versa.

Conclusion : Ainsi, toute fonction de solde peut être obtenue uniquement comme une suite d'opérations de charité : chaque opération de mendicité peut être remplacée par une charité équivalente. Autrement dit, pour toute opération F, il existe un ensemble réduit de charités qui produit le même résultat.

De plus, on peut classifier F comme une combinaison des opérations de charité et de mendicité à l'aide de la relation :

$$\alpha_i = c_i - m_i,$$

où c_i et m_i désignent respectivement le nombre de charités et de mendicités effectuées par v_i. Cela nous donne les cas suivants :

$$\begin{cases} \alpha_i > 0, & \text{on effectue } |\alpha_i| \text{ charités,} \\ \alpha_i < 0, & \text{on effectue } |\alpha_i| \text{ mendicités,} \\ \alpha_i = 0, & \text{on n'effectue aucune opération.} \end{cases}$$

2/ Nous avons établi que :

$$F : \mathrm{Sol}(G) \to \{\alpha_i \mid \alpha_i = C_i - M_i,\ i \in [|1, |G||]\},$$

où :

$$\begin{cases} C_i & \text{représente le nombre d'opérations de charité effectuées par le villageois } v_i, \\ M_i & \text{représente le nombre d'opérations de mendicité effectuées par le villageois } v_i. \end{cases}$$

Cette fonction est clairement une bijection, car pour un solde v_i d'un villageois, on peut l'associer à la mendience et à la charité des villageois adjacents :

$$S(v_i) \mapsto S(v_i) + (M_i - C_i) \cdot deg(v_i) - \sum_{v_j \in V} ((M_i - C_i) \cdot e(v_i; v_j))$$

Soit $e(v_i; v_j)$ défini par :

$$e(v_i; v_j) = \text{ le nombre de relations ou d'arrêts entre } v_i \text{ et } v_j,$$

Soit $e(v_i; v_j)$ le nombre de relations ou d'arrêts entre v_i et v_j, ou simplement le nombre d'arrêts de l'un ou l'autre entre ces deux éléments.

Remarquons que la fonction du solde peut permettre un nombre infini ou illimité de charité et de mendicité. Ainsi, nous définissons une fonction $F : \mathrm{sol}(G) \mapsto \mathbb{Z}^{|G|}$.

Dès l'exercice précédent, nous avons montré que la charité d'un villageois peut être exprimée par la mendicité des autres villageois, et vice-versa (même démonstration).

Par conséquent, nous pouvons réduire notre ensemble d'arrivées et de départs. Au lieu de réaliser les opérations sur l'ensemble $v_{|G|}$, nous pouvons effectuer ces opérations sur un villageois quelconque en le représentant par la charité et la mendicité avec tous les autres villageois. Ceci nous permet de réduire l'espace $\mathbb{Z}^{|G|}$ à

$$\mathbb{Z}^{\mathbf{|G|-1}}.$$

Pour la deuxième bijection, nous utilisons notre première bijection (sans la réduire). Cependant, nous pouvons réduire le nombre d'opérations de charité et de mendicité, puis la convertir uniquement en une fonction de charité.

Au lieu de faire la charité C_i fois et la mendicité M_i fois, nous pouvons simplifier en effectuant la charité $C_i - M_i$ fois. (Si $C_i - M_i < 0$, alors cela représente la mendicité, soit $M_i - C_i$ fois.)

Si **Cas 1** : on va rester avec des charités négatives (ou mendicité). Donc, si on applique la charité sur tous les s
on peut atteindre un nombre de charité où toutes les différences $M_i - C_i$ seront positives,
en laissant les villageois avec la plus petite différence : $\forall i \in [|1; |G||], M_j - C_j \leq M_i - C_i$.
Cas 2 : Dans ce cas, on reste avec des différences positives. Alors, on applique la mendicité sur tout le village en laissant le villageois avec la plus petite différence nulle.

3/ Pour chaque villageois v_i avec des fonctions socles σ_i, on a :

$$S_{\text{sol}(v_i)} = -\sigma_i \cdot \deg(v_i) + \sum_{v_j \in V} (\sigma_j \cdot e(v_i; v_j))$$

$$S_{\text{sol}(v_i)} = \sum_{v_j \in V} ((\sigma_j - \sigma_i) \cdot e(v_i; v_j))$$

5.1.3 Équations linéaires

1/ La matrice M avec les coefficients $M_{ii} = -\deg(v_i)$ et $M_{ij} = e(v_i, v_j)$ est l'opposée de la matrice laplacienne du graphe, c'est-à-dire $M = -M_{\text{laplacienne}}$. On peut également l'interpréter comme une représentation du graphe.
La matrice M est définie par :

$$M = A - D$$

$$A = \begin{bmatrix} 0 & e(v_1,v_2) & e(v_1,v_3) & \cdots & e(v_1,v_{|G|}) \\ e(v_2,v_1) & 0 & e(v_2,v_3) & \cdots & e(v_2,v_{|G|}) \\ e(v_3,v_1) & e(v_3,v_2) & 0 & \cdots & e(v_3,v_{|G|}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e(v_{|G|},v_1) & e(v_{|G|},v_2) & e(v_{|G|},v_3) & \cdots & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} \deg(v_1) & 0 & 0 & \cdots & 0 \\ 0 & \deg(v_2) & 0 & \cdots & 0 \\ 0 & 0 & \deg(v_3) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \deg(v_{|G|}) \end{bmatrix}$$

Donc notre matrice M ecrit sous la forme :

$$M = \begin{bmatrix} -\deg(v_1) & e(v_1,v_2) & e(v_1,v_3) & \cdots & e(v_1,v_{|G|}) \\ e(v_2,v_1) & -\deg(v_2) & e(v_2,v_3) & \cdots & e(v_2,v_{|G|}) \\ e(v_3,v_1) & e(v_3,v_2) & -\deg(v_3) & \cdots & e(v_3,v_{|G|}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e(v_{|G|},v_1) & e(v_{|G|},v_2) & e(v_{|G|},v_3) & \cdots & -\deg(v_{|G|}) \end{bmatrix}$$

2/Une fonction de *solde* sur un graphe $G = (V, E)$ sera une fonction où un villageois (nœud du graphe) va donner ou prendre des pairs de ses voisins en respectant sa relation d'amitié avec chaque autre.

1. Cette matrice à coefficients $M_{ii} = \deg(v_i)$, $M_{ij} = e(v_i, v_j)$ est notée la matrice **Laplacienne** de ce graphe G, où on peut l'interpréter comme une représentation du graphe.

$$M = \begin{bmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,|G|} \\ M_{2,1} & M_{2,2} & \cdots & M_{2,|G|} \\ \vdots & \vdots & \ddots & \vdots \\ M_{|G|,1} & \cdots & \cdots & M_{|G|,|G|} \end{bmatrix}$$

Le vecteur des soldes est noté $\vec{\sigma}$:

$$\vec{\sigma} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_{|G|} \end{bmatrix}$$

Le produit matriciel $M \cdot \vec{\sigma}$ donne :

$$M \cdot \vec{\sigma} = \begin{bmatrix} \sum_{i=1}^{|G|}(\sigma_i - \sigma_1)e(v_1, v_i) \\ \sum_{i=1}^{|G|}(\sigma_i - \sigma_2)e(v_2, v_i) \\ \vdots \\ \sum_{i=1}^{|G|}(\sigma_i - \sigma_{|G|})e(v_{|G|}, v_i) \end{bmatrix}$$

$$= - \begin{bmatrix} \mathrm{Soc}(v_1) \\ \mathrm{Soc}(v_2) \\ \mathrm{Soc}(v_3) \\ \vdots \\ \mathrm{Soc}(v_{|G|}) \end{bmatrix}$$

Notre problème consiste à rendre tous les soldes positifs, donc :

$$s(G) + M \cdot \vec{\sigma} \geq 0$$

où 0 est le vecteur nul $\Longrightarrow \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

avec s l'ensemble des soldes du village

3/

- **(a)** Ces coordonnées seront les ensembles des soldes obtenus par la fonction socle et qui sont des solutions du problème des soldes positifs.
- **(b)** Si cette face du polytope représente une configuration du jeu (comme une distribution de pain sur les villageois), alors un point ayant la distance maximale à cette face pourrait correspondre à une configuration du jeu où les ressources sont dans une position très éloignée de cette face, c'est-à-dire dans une configuration qui maximise le nombre de charte ou mendiance necessaire pour l atteindre.

4/Soit un polytope convexe $\mathcal{P} \subseteq \mathbb{R}^n$ défini par un système d'inégalités linéaires :

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\},$$

où $A \in \mathbb{R}^{m \times n}$ est une matrice et $b \in \mathbb{R}^m$ est un vecteur. Le nombre de points à coordonnées entières dans $\mathcal{P}$ est donné par :

$$|\mathcal{P} \cap \mathbb{Z}^n|.$$

Dans le cadre du problème de Painville, chaque point entier correspond à une distribution de soldes où :

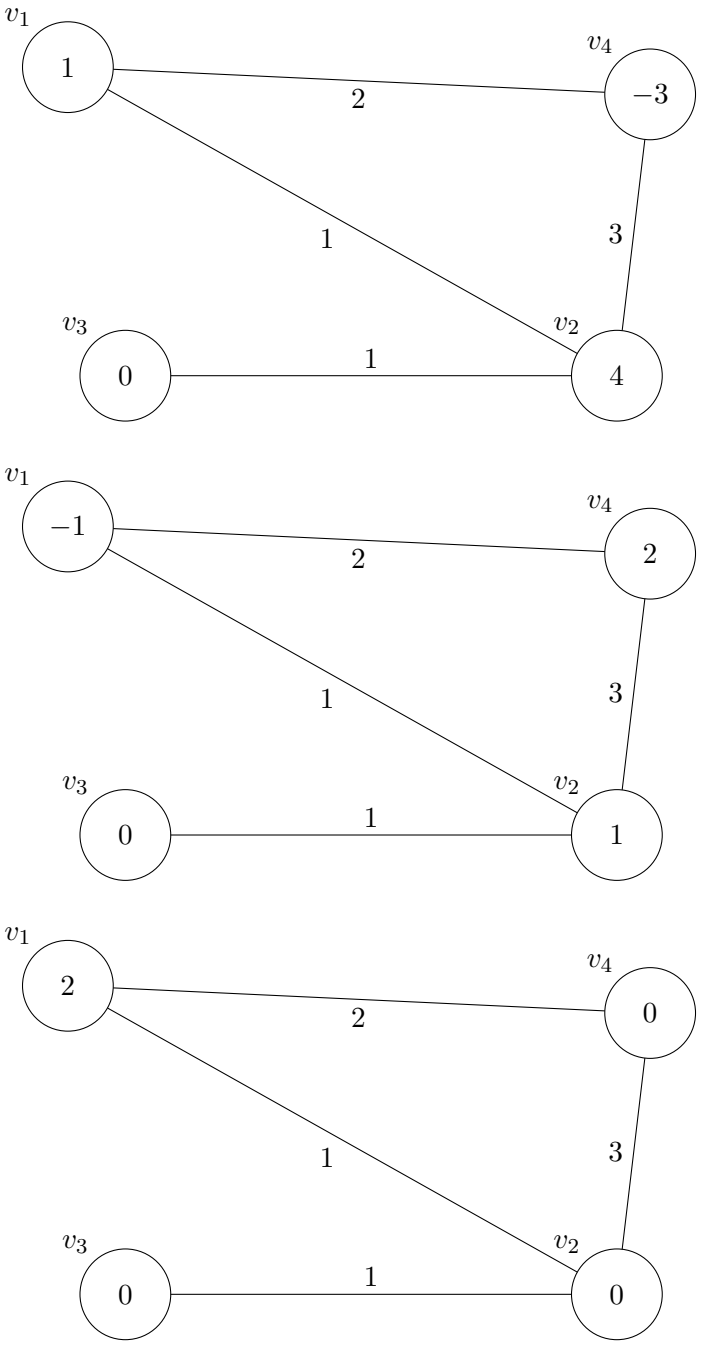
- Les coordonnées représentent les soldes des villageois.
- Les contraintes garantissent qu'aucun villageois ne possède un solde strictement négatif.

Le nombre de points à coordonnées entières dans le polytope $\mathcal{P}$ représente le nombre de manières possibles de redistribuer le pain de façon à satisfaire les contraintes imposées par le graphe. La relation entre le nombre du point a coordonnee entieres et le volume du polytope est souvent liee a la conjecture du volume de Ehrhart , une conjecture qui nous donne un nombre maximal de points lattices , ou a coordonnee entiere , ou :

$$Vol(k)$$

5.1.4 Algorithme

1/on a fait la mendience de v_4 puis mendience de v_1



2/Un algorithme mendiant est un algorithme qui prend la dette de chaque villageois et augmente son solde par des opérations de mendience .

- Si la fonction solde est résoluble : cette fonction va diminuer la dette totale du village après chaque mendicité, en convergeant vers une solution à la fin.
- Si la fonction solde n'est pas résoluble : nous montrerons que tous les villageois effectueront des opérations de mendicité dans l'algorithme mendiant, ce qui entraînera un cycle, et donc l'absence de solution.
  demo:
  Soit s_i le solde du villageois v_i après l'exécution de l'algorithme.

On a :

$$0 \leq s_i \leq \deg(v_i) - 1,$$

car la mendicité s'arrête lorsque $s_i \geq 0$,
et le maximum de s_i est donc $\deg(v_i) - 1$.
Ainsi, chaque solde s_i peut être exprimé par un nombre fini de valeurs.
Après un grand nombre d'opérations de mendicité, s_i reviendra à sa position initiale.
Cela implique que l'algorithme mendiant entre dans un cycle sans jamais atteindre une solution.

3/On suppose qu il existe 2 solutino differente notéé ω_1 et ω_2 . sans perte de généralité , il existe un villageois v_k qui fait la mendience en ω_2 plus que ω_1
Notons le nombre d'opérations de mendicité de $\omega_2 commesuit$: $\{m_1, m_2, m_3, \ldots, m_j\}$.Selon notre hypothèse, il existe un K tel que

$$m_k < \omega_1(v_k),$$

où $\omega_1(v_k)$ représente la mendicité de ω_1 sur v_k.

Nommons $\bar{\omega_2}$ l'ensemble des mendicités défini par :

$$\bar{\omega_2} = \{m_1, m_2, m_3, \ldots, m_{k-1}, 0, 0, \ldots, 0\}.$$

On a donc $\bar{\omega_2} \leq \omega_1$, ce qui implique que :

$$S_{\bar{\omega_2}}(v_k) = s_i(v_k) + \sum_{v_j \in V} \big(m_i \cdot e(v_i, v_k)\big) - \deg(v_k) \cdot m_k.$$

En utilisant la relation $\bar{\omega_2} \leq \omega_1$, on peut écrire :

$$S_{\bar{\omega_2}}(v_k) \geq s_i(v_k) + \sum_{v_j \in V} \big(\omega_1(v_i) \cdot e(v_i, v_k)\big) - \deg(v_k) \cdot \omega_1(v_k) = S_{\omega_1}(v_k) \geq 0$$

5.1.5 Soldes w-stable

1/ Avant de commencer notre démonstration, on introduit une nouvelle façon de voir les graphes : Un arbre couvrant d'un graphe G est un sous-graphe de G qui couvre tous les sommets de G avec le minimum des arêtes possibles.

On va dès maintenant définir un ordre de notre arbre :

$$V_1, V_2, V_3, \ldots, V_n$$

et w = (V_1)pour A, B $\in$ Div(G)

on dit que $A > B$ si :

$$\mathrm{Deg}(A) > \mathrm{Deg}(B) \quad \text{ou}(\quad \mathrm{Deg}(A) = \mathrm{Deg}(B) \quad \text{et} \quad A(V_i) < B(V_i)$$

$$\text{pour le plus petit } i \text{ tel que } A(V_i) \neq B(V_i))$$

* Soient $A, B \in \mathrm{Div}(G)$ et $w \in V$. si B rt Résultat de la charité sur un ensemble non vide $S \in V/\{w\}$ tel que $A > B$.

Preuve : On considère un arbre couvrant d'ordre $V_1, V_2, \ldots, V_n$ avec $w = V_1$.

Prenons V_j avec j le plus petit indice tel que V_j est adjacent avec un élément dans S.

Après la charité sur S, on a :

$$s_A(V_i) = s_B(V_i) \quad \text{pour tout } i \in [|1, j-1|],$$

mais pour j,

$$s_A(V_j) < s_B(V_j).$$

Donc, par définition,

$$A > B.$$

et

$$w = v_1$$

2/ Maintenant on montre l'existence de l'opération F pour tout fonction de sable $F(S)$ et W-stable, ainsi cette fonction est unique.

1. Existence Considérons un arbre couvrant où $W = V$. Si on fait la charité sur un S et $V/\{W\}$, on obtient A_1 selon * qui precede dans q1 ,

$$A > A_1 > A_2 > A_3 > \cdots$$

Mais puisque V est un ensemble fini, donc il existe un A_i où on ne peut pas refaire des charités. À cet endroit, on déclare que F est W-stable.

2. Unicité Supposons deux opérations F et F', W-stable. Prenons $m = \max_{v \in V}\{C(V)\}$ (où $C(V)$ est le nombre de c

$$S = \{V, C(V) = m\}.$$

Si $S = V$, alors la charité est faite sur tout le graphe, donc $F = F'$.

Si $S \neq V$, il suppose en plus que $q \notin S$ ($q \in S$, on change $C(q)$ par l'opposé de la charité sur tout le village).
Donc : $v \in S$ et $u \notin S$ alors $C(V) - C(u) \geq 1$, et donc pour $v \in S$ et $u \notin S$.

$$0 \leq F'(V) = F(V) - \sum_{u \in \mathrm{Adj}(V)} (C(V) - C(u))\, e(v,u) \leq F(V) - \mathrm{outdeg}(V),$$

où outdeg(V) est le nombre d'arêtes de V avec tous les sommets $u \notin S$.

Cette inégalité est vraie parce que :

$$\sum_{u \in \mathrm{Adj}(V)} (C(V) - C(u)) \geq \mathrm{outdeg}(V),$$

car

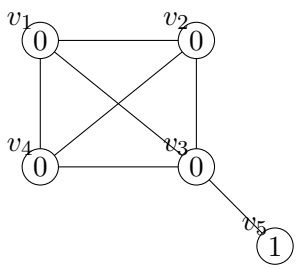
$$C(V) - C(u) \geq 1 \quad \text{pour tout} \quad u \notin S.$$

Donc :

$$0 \leq F(V) - \mathrm{outdeg}(V),$$
$$\Rightarrow F(V) \geq \mathrm{outdeg}(V).$$

Mais puisque cela est vrai, si on fait une opération de charité sur tout les éléments de S, leurs sables ne va pas diminuer, mais F est W-stable, ce qui est une contradiction. Donc F est unique.

3 Pour la question, j'ai trouvé un simple contre-exemple :



On considère $K_4 \cup \{V_5\}$.

Si on a :

$$s_1 = s_2 = s_3 = s_4 = 0 \quad \text{et} \quad s_5 = 1,$$

on peut enlever 3 arêtes et laisser le graphe connexe.

Or :

$$1 < 4,$$

et on peut faire une opération de charité sur V_5 (prenant W un autre sommet).

Donc, notre graphe n'est pas W-stable.
or on peut bien enlever 3 aretes sans que le graphe ne soit pas connexe , et la somme des scores est egale a 1 ¡ 4

la condition que la somme des scores soit inferieur a d , nomme le genus , nous donne l existance des configuration non resolvable , et si la somme est superieur a d , cela implique que quelque soit la configuration (repartition des soldes) , elle est resolvable ,soit par algorithme mendiant ou w-stabilité.

5.2 Bibliographie :

1. Most-important-resource 1
2. Most-important-resource 2
3. Most-important-resource 3
4. A lot of PDFs that I used but can't find now :)