DM 1

À rendre le 8 novembre en TD.

- Exercice 1 - Découpage électoral. Le parti des programmeurs dynamiques (PPD) se présente aux élections sur deux circonscriptions qui regroupent à elles deux un total de n zones (avec n pair). Chaque zone a m électeurs. Le nombre de voix du PPD dans une zone $i \in [n]$ (on note $[n] := \{1, \ldots, n\}$) est estimé à v_i, avec $0 \leq v_i \leq m$. Le PPD peut choisir à sa guise le regroupement des zones en deux circonscriptions de tailles égales. Un *découpage gagnant* est une partition de $[n]$ en deux parties sur lesquelles le PPD à la majorité absolue.

Précisément, c'est une partie $I \subseteq [n]$ de taille $n/2$ telle que $\sum_{i \in I} v_i > m.n/4$ et $\sum_{i \in [n] \setminus I} v_i > m.n/4$.

Par exemple pour $n = 4$ et $m = 100$, si les estimations de vote sont $46, 60, 55, 44$, un découpage gagnant est 46+55 et 60+44 (regroupant donc les zones 1 et 3 et les zones 2 et 4).

1. Proposer une instance ayant 6 zones telle qu'il existe un unique découpage gagnant (on choisira ici $m = 100$).
2. Proposer un algorithme en $O(n^3 m)$ qui calcule tous les nombres possibles de voix des sous-ensembles de $n/2$ zones.
3. Comment décider s'il existe un découpage gagnant en temps polynomial?

- Exercice 2 - Indépendant. On se donne un graphe $G = (V, E)$ ainsi qu'une fonction de poids ω positive définie sur les arêtes. Un sous-ensemble d'arêtes F est *indépendant* si chaque composante connexe de F possède au plus un cycle (i.e. est un arbre ou un arbre plus une arête).

1. Montrer que si G a n sommets, alors un indépendant a au plus n arêtes.
2. Soit F un indépendant. On dit qu'un sommet $v \in V$ est *saturé dans* F s'il appartient à une composante de F qui possède un cycle. Montrer que si e est une arête de $E \setminus F$ qui ne relie pas deux sommets saturés, alors $F \cup \{e\}$ est indépendant.
3. Montrer que l'ensemble des indépendants forme un matroide.
4. Proposer un algorithme qui calcule un indépendant de poids maximum (on indiquera surtout comment les composantes connexes sont mises à jour).

- Exercice 3 - Arbre binaire. On se donne un arbre binaire A de racine r ayant n noeuds tel que chaque noeud i (interne ou feuille) possède une valeur $v_i > 0$. On note p la *hauteur* de A, c'est à dire la distance maximale de la racine à une feuille. Pour les problèmes suivants, décrire un algorithme de résolution aussi performant que possible, montrer sa validité, et calculer sa complexité.

1. Calculer un sous-ensemble X de noeuds de valeur totale maximale sans relation de parenté directe (i.e. pour toute paire de noeuds x, y de X le parent de y n'est pas x).

2. Calculer un sous-ensemble X de noeuds de valeur totale maximale sans relation de descendance directe (i.e. tout chemin de la racine à une feuille contient au plus un élément de X).

3. Calculer un minimum local de A (dont la valeur est inférieure ou égale à celles de son enfant gauche, son enfant droit, et de son parent, si applicable).

4. (Bonus) Calculer un minimum local dans la grille $n \times n$ en temps $O(n)$.

- Exercice 4 - Rendu de la monnaie. On considère le problème du rendu de la monnaie. Étant donne une somme à rendre $S \in \mathbb{N}$ et un système monétaire $P := \{p_1 > p_2 > \ldots > p_n\}$ avec $\forall i \in [n], p_i \in \mathbb{N}$ et $p_n := 1$, on cherche à minimiser le nombre de pièces pour rendre la somme S en utilisant autant de fois que l'on souhaite chaque pièce. Formellement, cela revient à minimiser $\sum_{i=1}^{n} \lambda_i$ sous la contrainte $\sum_{i=1}^{n} \lambda_i p_i = S$ et $\forall i \in [n], \lambda_i \in \mathbb{N}$. On appellera $N_P(S)$ ce minimum.

1. Calculer $N_P(S)$ à la main pour :
   (a) $S = 9$ et $P = \{5, 2, 1\}$.
   (b) $S = 6$ et $P = \{4, 3, 1\}$.

2. Exprimer $N_P(S)$ en fonction des $N_P(S - p_i)$. En déduire un algorithme de programmation dynamique qui résoud le problème en temps $O(Sn)$.

L'algorithme précédent est malheureusement seulement *pseudo-polynomial*, car l'entier S, codé en binaire, est de taille $\log(S)$. L'objectif à présent est de trouver une solution polynomiale en n et la taille de S (la taille des p_i étant bornée) dans certains cas. Nous allons nous intéresser à l'approche gloutonne qui consiste à prendre la plus grande pièce tant que c'est possible.

1. Prouver que si $\{\lambda_i^G\}_{i \in [m]}$ est la solution donnée par l'algorithme glouton, alors $\forall i \in [n-1], \lambda_{i+1}^G < \frac{p_i}{p_{i+1}}$.

2. Pour les systèmes monétaires P suivants, prouver que l'algorithme glouton est correct ou exhiber un contre-exemple S (les méthodes numériques pour trouver un tel contre-exemple sont autorisées):
   (a) $P = \{5, 2, 1\}$.
   (b) $P = \{2^{m-i}\}_{i \in [m]}$.
   (c) $P = \{200, 149, 33, 1\}$.
   (d) (Bonus) $P = \{F_{m-i+1}\}_{i \in [m]}$ pour $F_1 = 1, F_2 = 2, F_{k+2} = F_{k+1} + F_k$.

3. On appelle les systèmes monétaires pour lesquels l'algorithme glouton est correct des systèmes *canoniques*. Prouver que si P n'est pas canonique, alors il existe un contre-exemple S de taille inférieure à $p_1 \sum_{i=2}^{n} p_i$.

4. Conclure: donner un algorithme en temps polynomial en n et $\log(S)$ (la taille des p_i étant bornée par une constante C) qui vérifie que le système monétaire P est canonique et qui, le cas échéant, résoud le problème du rendu de la monnaie sur S.