Exercice 1. Algorithme de Karatsuba

On souhaite calculer le produit de deux entiers x et y. On suppose que x et y ont $2n$ chiffres en base 2. On peut donc écrire :

$$x = a2^n + b$$

$$y = c2^n + d$$

Avec $a, b, c, d < 2^n$.

1. Montrer qu'on peut calculer le produit xy en effectuant seulement 3 multiplications à n chiffres et expliciter ces 3 multiplications (en considérant que les multiplications par 2^k sont négligeables).
2. Écrire une fonction `int taille(int x)` renvoyant le nombre de chiffres de `x` en base 2.
3. Écrire une fonction `int karatsuba(int x, int y, int n)` qui calcule le produit de `x` et `y` en utilisant l'algorithme de Karatsuba, où `x` et `y` ont n chiffres en base 2.
4. En déduire une fonction `int mult(int x, int y)` qui calcule le produit de `x` et `y` en utilisant l'algorithme de Karatsuba.
5. Montrer que la complexité de cette méthode est en $\mathrm{O}(n^{\log_2(3)})$.
   Est-ce mieux que la méthode naïve ?
6. Expliquer comment adapter cette méthode au produit de deux polynômes.

Exercice 2. Suites graphiques

Une suite décroissante est dite *graphique* s'il existe un graphe simple (sans arête multiple ni boucle sur un sommet) dont les degrés des sommets correspondent à cette suite.

1. Les suites suivantes sont-elles graphiques ?
   - (3, 3, 2, 2)
   - (5, 3, 3, 2)
   - (3, 3, 1, 1)
2. Trouver deux graphes différents correspondants à la suite $(3, 2, 2, 2, 1)$.
3. Soit $n \geq 2$ et $(d_1, ..., d_n)$ une suite décroissante. Montrer l'équivalence des deux propriétés suivantes :
   a) La suite $(d_1, ..., d_n)$ est graphique.
   b) La suite $(d_2 - 1, d_3 - 1, ..., d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+2}, , d_{d_1+3}, ..., d_n)$ est graphique.

   Pour le sens direct, on pourra montrer par l'absurde l'existence d'un graphe $G = (V, E)$ tel que $V = \{v1, ..., vn\}$, $\deg(v_i) = d_i$ et tel que v_i soit adjacent aux sommets v_2, v_3, ..., v_{d_1+1}.
4. Deduire de ce résultat un graphe correspondant à la suite $(4, 4, 3, 2, 2, 1)$.

Exercice 3. Minimum de fenêtre glissante

Soit `a` un tableau de n entiers et $k \in \mathbb{N}$. L'objectif de cet exercice est de créer un tableau `a_min` de taille $n - k$ tel que `a_min[i]` soit le minimum des `a[j]` pour $j \in [\![i, i + k]\!]$.

1. Quelle serait la complexité de la méthode naïve ?
2. Écrire un meilleur algorithme en pseudo-code, en utilisant une file de priorité. Quelle serait la complexité ?

Dans la suite, on va implémenter un algorithme utilisant une file à deux bouts `q` (*deque* ou *double-ended queue* en anglais) qui va contenir des éléments de `a`. Une file à deux bouts possède les opérations suivantes :

- `add_right(q, x)` : ajoute l'élément x à la fin de `q`.
- `add_left(q, x)` : ajoute l'élément x au début de `q`.
- `peek_right(q)` : renvoie l'élément à la fin de `q`.
- `peek_left(q)` : renvoie l'élément au début de `q`.

- `pop_right(q)` : supprime l'élément à la fin de `q`.
- `pop_left(q)` : supprime l'élément au début de `q`.
- `is_empty(q)` : détermine si `q` est vide.

3. Donner des implémentations possibles pour une file à deux bouts et discuter de leurs complexités. Implémenter une file à deux bouts en C (on écrira seulement `add_right`, `peek_right`, `pop_right`).

À l'itération i, l'algorithme va considérer `a[i]` avec l'invariant de boucle suivant : `q` contient, dans l'ordre, des indices $i_1, ..., i_p$ de `a` (où i_1 est l'élément au début de `q` et i_p l'élément de fin) tels que :

- $i_1 \leq i_2 \leq ... \leq i_p$.
- `a[i_1]` $\leq$ `a[i_2]` $\leq ... \leq$ `a[i_p]`.
- $i_1 > i - k$ (on considère seulement les k derniers éléments).

4. Avec ces notations, que vaut `a_min[i]`?
5. Comment mettre à jour `q` à l'itération $i + 1$?
6. En déduire une fonction `int* minimum_sliding(int* a, int n, int k)` renvoyant `a_min`.
7. Quelle est la complexité de `minimum_sliding`?

Exercice 4. Algorithme de Ford-Fulkerson

On considère un graphe orienté $\overrightarrow{G} = (V, \overrightarrow{E})$ avec une **capacité** $c : \overrightarrow{E} \longrightarrow \mathbb{R}^+$.
Si $A \subseteq V$, on définit :

- $A^+ = \{(u, v) \in \overrightarrow{E} \mid u \in A, v \notin A\}$ (« arcs sortants de A »)
- $A^- = \{(u, v) \in \overrightarrow{E} \mid u \notin A, v \in A\}$ (« arcs rentrants dans A »)

Si $v \in V$, on définit $v^+ = \{v\}^+$ et $v^- = \{v\}^-$.
Si $f : \overrightarrow{E} \longrightarrow \mathbb{R}^+$ et $\overrightarrow{B} \subseteq \overrightarrow{E}$, on définit :

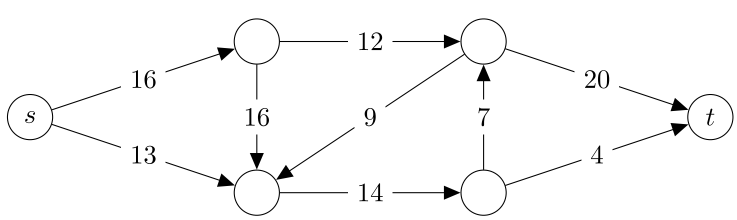
$$f(\overrightarrow{B}) = \sum_{\overrightarrow{e} \in B} f(\overrightarrow{e})$$

Un **flot** est une fonction $f : \overrightarrow{E} \longrightarrow \mathbb{R}^+$ telle que :

- $\forall \overrightarrow{e} \in \overrightarrow{E} \,:\, 0 \leq f(\overrightarrow{e}) \leq c(\overrightarrow{e})$
- $\forall v \in V - \{s, t\} \,:\, f(v^-) = f(v^+)$ (la somme des flots rentrants dans un sommet est égal à la somme des flots sortants).

La **valeur** d'un flot f est définie par $|f| = f(s^+)$. L'objectif de cet exercice est de trouver un flot de valeur maximum dans $\overrightarrow{G}$.

1. Dans le graphe ci-dessous, on a représenté la capacité sur chaque arc. Donner le flot de de plus grande valeur que vous réussissez à trouver dans ce graphe, en explicitant la quantité de flot sur chaque arc. On ne demande aucune justification.



2. Montrer que tout graphe muni d'une capacité possède un flot (on cherchera à définir un flot très simple).
3. Soit $\overrightarrow{P}$ un chemin de s à t et c la capacité minimum des arcs de $\overrightarrow{P}$. On définit $f : \overrightarrow{E} \longmapsto \mathbb{R}^*$ qui vaut c sur chaque arc de $\overrightarrow{P}$ et 0 partout ailleurs. Justifier que f est un flot.
4. L'algorithme suivant permet de construire un flot en ajoutant itérativement un chemin de s à t :

Algorithme : Ford-Fulkerson

$f \leftarrow$ flot nul
Tant que $\exists$ un chemin $\overrightarrow{P}$ de s à t, dont les arcs sont tous de capacité > 0 :
    $c \longleftarrow$ minimum des capacités de $\overrightarrow{P}$
    Diminuer de c la capacité des arcs de $\overrightarrow{P}$
    Augmenter le flot f de c, le long des arcs de $\overrightarrow{P}$

Appliquer l'algorithme de Ford-Fulkerson sur le graphe de la 1ère question.

5. On suppose que toutes les capacités sont entières. Montrer que l'algorithme de Ford-Fulkerson termine et donner la complexité dans le pire cas.

Une **coupe** de $\overrightarrow{G}$ est un ensemble $S \subseteq V$ contenant s mais pas t. La capacité d'une coupe S est la somme $c(S^+)$ des capacités des arcs sortants de S.

6. Soit S une coupe. Montrer que $f(S^+) \leq c(S^+)$ et $f(S^+) = |f|$.
7. Soit f un flot et S une coupe vérifiant $f(S^+) = c(S^+)$. Montrer que :
   - f est un flot de valeur maximum
   - S une coupe de capacité minimum
8. Montrer que si l'algorithme de Ford-Fulkerson termine, le flot obtenu est un flot maximum.
9. Quel méthode connaissez-vous pour trouver un chemin dans l'algorithme de Ford-Fulkerson ? Implémenter algorithme de Ford-Fulkerson en OCaml, avec l'une de ces méthodes.

Exercice 5. Arbre de segments

Soit `a` un tableau d'entiers.
On souhaite concevoir, à partir de `a`, une structure de donnée `t` permettant de réaliser efficacement les opérations suivantes :

- `min_range i j t` : renvoie le minimum des éléments de `a` entre les indices `i` et `j`.
- `set i e t` : met à jour la structure pour que `a.(i)` soit remplacé par `e`.

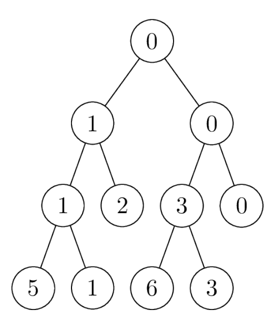
1. Proposer une solution naïve et donner sa complexité.

Dans la suite, on va utiliser un arbre de segments :

Définition : Arbre de segments

Un arbre de segments (pour un tableau `a`) est un arbre binaire dont :
- Les feuilles sont les éléments de `a`
- Chaque noeud est étiqueté par un triplet `(m, i, j)` tel que son sous-arbre contienne les feuilles `a.(i), ... , a.(j)` et `m` est le minimum de ces valeurs.

Par exemple, voici un arbre de segments obtenu à partir du tableau `[|5; 1; 2; 6; 3; 0|]`, où on a représenté seulement les minimums (premiers éléments de chaque triplet) :



Ainsi, les feuilles sont bien les éléments du tableau `[|5; 1; 2; 6; 3; 0|]` et chaque noeud correspond à un minimum sur une certaine plage du tableau.
Remarque : Il y a d'autres arbres de segments possibles pour le même tableau.
On utilisera le type suivant :

```
type tree = E | N of int * int * int * tree * tree
```

Ainsi, un sous-arbre `N(m, i, j, g, d)` possède `a.(i)`, `a.(i + 1)`, ..., `a.(j)` comme feuilles, de minimum `m` et de sous-arbres `g`, `d`.

2. Écrire une fonction `make : int array -> tree` qui construit un arbre de segments à partir d'un tableau d'entiers. On fera en sorte que l'arbre construit soit de hauteur logarithmique en la taille du tableau.
3. Écrire une fonction `set i e t` qui met à jour `t` en remplaçant `a.(i)` par `e`.
   Quelle est la complexité de cette fonction?
4. Écrire une fonction `min_range i j t` renvoyant le minimum des éléments de `a` entre les indices `i` et `j`.
5. Montrer que la complexité de `min_range i j t` est $O(\log(n))$, où n est la taille de `a`.
6. On s'intéresse à un autre problème : calculer efficacement une somme d'éléments entre les indices `i` et `j`, dans un tableau. Adapter les fonctions précédentes pour y parvenir.