Exercice 1. Mots

Question 1 –

$$\boxed{\text{Ce sont tous les mots de la forme } a^n \text{ ou } b^n \text{ avec } n \in \mathbb{N}.}$$

En effet, supposons que $u \neq \epsilon$ soit un mot vérifiant la propriété. Alors tous les facteurs de longueur 1 de u sont des préfixes de u. En d'autres termes, toutes les lettres de u sont égales à u_1.

Question 2.a – $\boxed{\text{Oui, c'est vrai.}}$ En effet, soient u' (resp. v') le préfixe de longueur maximale de u (resp. v). Alors par hypothèse $u' = v'$. De plus, on a toujours $v' = v$ et $u' = u$.

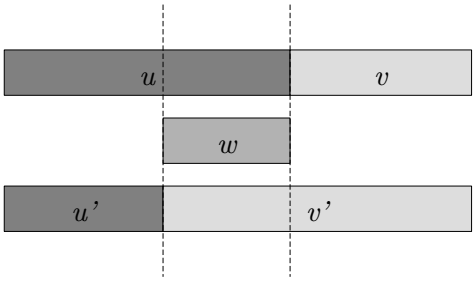
Question 2.b – $\boxed{\text{Non ce n'est plus vrai.}}$ Par exemple les mots d'une lettre $u = a$ et $v = b$ ont tous les deux pour ensemble des préfixes propres $\{\epsilon\}$.

Question 3 – Montrons par récurrence sur $|u|$ que $\boxed{a = b \text{ et } u \in \{a\}^*}$.

⋆ Initialisation. Si $|u| = 0$, alors $u = \epsilon \in \{a\}^*$ et $a = au = ub = b$.

⋆ Hérédité. Supposons la propriété vraie au rang n et montrons la au rang $n+1$. Soit $u \in \Sigma^{n+1}$ tel que $a \cdot u = u \cdot v$. Comme, $|a| \leqslant |u|$, a est un préfixe de u, c'est à dire qu'il existe $v \in \Sigma^n$ tel que $u = av$. Ainsi, $aav = avb$ et donc $av = vb$. Par l'hypothèse de récurrence, on a $a = b$ et $v \in \{a\}^*$. D'où $u = av \in \{a\}^*$.

Question 4 – On suppose que $|u| \leqslant |u'|$ (l'autre cas est symétrique). Comme $u \cdot v = u' \cdot v'$ et $|u| \leqslant |u'|$, u est un préfixe de u', c'est à dire qu'il existe $w \in \Sigma^*$ tel que $u' = u \cdot w$. On obtient $u \cdot v = u \cdot w \cdot v'$. Donc $v = w \cdot v'$



Exercice 2. Langages

Question 1 – On le montre par double implication.

($\Rightarrow$) On suppose $\epsilon \in L$. Alors, pour tout $u \in L$, on a $u = \epsilon \cdot u \in L \cdot L$, d'où le résultat.

($\Leftarrow$) Comme L est non vide, on peut considérer $u \in L$ un mot tel que $|u|$ est minimal parmi les éléments de L. On a $u \in L \subset L \cdot L$. Il existe donc $(v, w) \in L^2$ tels que $u = v \cdot w$. On a donc $|u| = |v| + |w| \geqslant 2|u|$. Donc $|u| \leqslant 0$, c'est à dire $u = \epsilon$

Question 2 – On a :

$$\boxed{\text{pref}(L_1 \cdot L_2) = \begin{cases} \varnothing \text{ si } L_2 = \varnothing \\ \text{pref}(L_1) \cup \Big(L_1 \cdot \text{pref}(L_2)\Big) \text{ sinon} \end{cases}}$$

Montrons le cas $L_2 \neq \varnothing$ par double inclusion.

($\subset$) Soit $u \in \text{pref}(L_1 \cdot L_2)$. Alors, u est un préfixe d'un mot $v_1 \cdot v_2$ où $v_1 \in L_1$ et $v_2 \in L_2$. On traite deux cas :

- Si $|u| \leqslant |v_1|$, alors $u \in \text{pref}(v_1)$
- Si $|u| > |v_1|$, alors u s'écrit $u = v_1 \cdot w$ avec w un préfixe de w_2 et donc $u \in L_1 \cdot \text{pref}(L_2)$.

($\supset$) On vérifie facilement que $\text{pref}(L_1) \subset \text{pref}(L_1 \cdot L_2)$ et $L_1 \cdot \text{pref}(L_2) \subset \text{pref}(L_1 \cdot L_2)$.

Question 3 – Soit L une solution de l'équation.

$\star$ Montrons par récurrence sur $n \in \mathbb{N}$ que :

$$(E_n) : L = A^n L \cup \bigcup_{k=0}^{n-1} A^k B$$

On remarque que (E_0) est l'équation $L = \{\epsilon\} \cdot L \cup \varnothing$ qui est vraie. Supposons que (E_n) soit vraie et montrons (E_{n+1}). En utilisant (E_n) et l'équation à résoudre :

$$\begin{aligned} L &= A^n L \cup \bigcup_{k=0}^{n-1} A^k B \\ &= A^n \Big(AL \cup B \Big) \cup \bigcup_{k=0}^{n-1} A^k B \\ &= A^{n+1} L \cup A^n B \cup \bigcup_{k=0}^{n-1} A^k B \\ &= A^{n+1} L \cup \bigcup_{k=0}^{n} A^k B \end{aligned}$$

D'où (E_{n+1}).

$\star$ Montrons $L = A^* B$ par double inclusion.

($\supset$) Pour tout $n \in \mathbb{N}$, (E_{n+1}) est vraie, donc $A^n B \subset L$. Ainsi :

$$A^* B = \left(\bigcup_{n \in \mathbb{N}} A^n \right) \cdot B = \bigcup_{n \in \mathbb{N}} \Big(A^n \cdot B \Big) \subset L$$

($\subset$) Soit $w \in L$. On pose $n = |w|$. Tout mot $w' \in A^{n+1} L$ vérifie $|w'| \geqslant n + 1$ (car $\epsilon \notin A$) et donc $w' \neq w$. Ainsi, d'après (E_{n+1}), on a $w \in \bigcup_{k=0}^{n} A^k B \subset A^* B$.

$\boxed{A^* B \text{ est l'unique solution de l'équation}}$

Exercice 3. Mots de Fibonacci

On a :

$f_0 = \epsilon$ $\quad$ $f_1 = a$ $\quad$ $f_2 = b$ $\quad$ $f_3 = ba$
$f_4 = bab$ $\quad$ $f_5 = babba$ $\quad$ $f_6 = babbabab$ $\quad$ $f_7 = babbababbabba$

Question 1 – Montrons par récurrence sur $n \geqslant 3$ que :

Si n est impair, alors les deux dernières lettres de f_n sont ba.
Si n est pair, alors les deux dernières lettres de f_n sont ab.

$\star$ Initialisation. Pour $n = 3$ et $n = 4$, on a $f_3 = ba$ et $f_4 = bab$ d'où le résultat.

$\star$ Hérédité. Soit $n \geqslant 5$. On suppose le résultat vrai pour tout $n' < n$ et on le montre pour n. Par définition, on a $f_n = f_{n-1} \cdot f_{n-2}$. Or, n et $n-2$ ont la même parité et $n - 2 \geqslant 3$. On peut donc utiliser l'hypothèse de récurrence pour conclure.

Question 2 – Montrons le par récurrence sur $n \geqslant 3$.

$\star$ Initialisation. Pour $n \in \{3, 4, 5\}$, on a $g_3 = \epsilon$, $g_4 = b$ et $g_5 = bab$ d'où le résultat.

$\star$ Hérédité. Soit $n \geqslant 6$. On suppose le résultat vrai pour tout $n' < n$ et on le montre pour n. On a :

$$f_n = f_{n-1} \cdot f_{n-2} = f_{n-2} \cdot f_{n-3} \cdot f_{n-2}$$

Si n est pair :

$$g_n = g_{n-2} \cdot ab \cdot g_{n-3} \cdot ba \cdot g_{n-2}$$

Si n est impair :

$$g_n = g_{n-2} \cdot ba \cdot g_{n-3} \cdot ab \cdot g_{n-2}$$

Dans les deux cas, on obtient un palindorme par hypothèse de récurrence.

Exercice 4.

On le montre par double implication.

($\Rightarrow$) Soient x, y tels que $u = x \cdot y$ et $v = y \cdot x$. On pose $w = x$, alors :

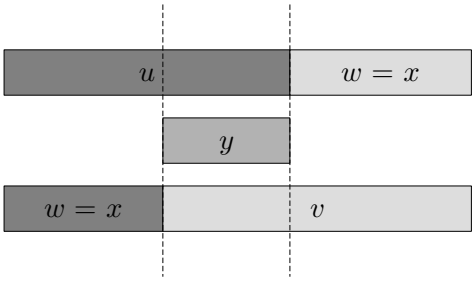
$$u \cdot w = (x \cdot y) \cdot x = x \cdot (y \cdot x) = w \cdot v$$

($\Leftarrow$) Supposons qu'il existe un mot w tel que $u \cdot w = w \cdot v$.

$\star$ Si $|w| \leqslant |u|$. Comme w est un préfixe de u, il existe $y \in \Sigma^*$ tel que $u = w \cdot y$. De plus :

$$w \cdot y \cdot w = u \cdot w = w \cdot v$$

Ainsi $y \cdot w = v$, d'où le résultat en posant $x = w$.



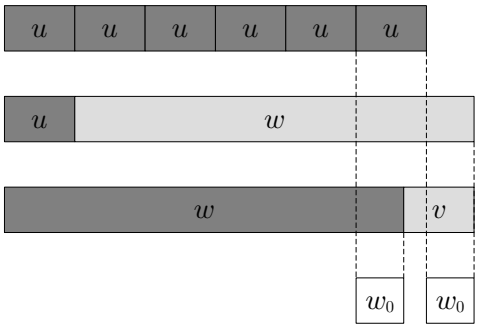
$\star$ Sinon, on a $|w| > |u|$. Soit $w_0 \in \Sigma^*$ de taille minimale parmi les mots vérifiant $u \cdot w_0 = w_0 \cdot v$.

Supposons par l'absurde que $|w_0| \geqslant |u|$, alors u est un préfixe de w_0, c'est à dire qu'il existe $w_0' \in \Sigma^*$ tel que $w_0 = u \cdot w_0'$ avec $|w_0'| < |w_0|$ (car u est non vide). De plus :

$$u \cdot u \cdot w_0' = u \cdot w_0 = w_0 \cdot v = u \cdot w_0' \cdot v$$

Ainsi, $u \cdot w_0' = w_0' \cdot v$ avec $|w_0'| < |w_0|$ ce qui contredit la minimalité de w_0.

En résumé, on a $u \cdot w_0 = w_0 \cdot v$ et $|w_0| < |u|$. On est donc ramenés au cas précédent, ce qui permet de conclure.



Exercice 5. Code sur un alphabet

Question 1 –

⋆ $\boxed{L_1 \text{ n'est pas un code.}}$ En effet :

$$(ba)(a)(bba) = (baab)(ba)$$

⋆ $\boxed{L_2 \text{ est un code.}}$ En effet, supposons qu'il existe des mots x_i et y_j tels que :

$$x_1 \dots x_p = y_1 \dots y_q$$

Supposons par l'absurde qu'il existe un indice tel que $x_i \neq y_i$. On note k le plus petit de ces indices et donc :

$$x_k \dots x_p = y_k \dots y_q$$

Au vu des éléments de L_2, on a deux possibilités :

$$x_k = aa \text{ et } y_k = aab \text{ ou inversement.}$$

Sans perte de généralité, on peut supposer qu'on est dans le premier cas. Le mot x_{k+1} commence nécessairement par b, donc $x_{k+1} = bba$. Ainsi, y_{k+1} commence nécessairement par ba ce qui est impossible.

⋆ $\boxed{L_3 \text{ n'est pas un code.}}$ En effet :

$$(ab)(baa)(baa)(baa) = (abba)(ab)(aabaa)$$

⋆ $\boxed{L_4 \text{ est un code.}}$ En effet, soient x_i et y_j des mots de L_4 tels que :

$$x_1 \dots x_p = y_1 \dots y_q$$

Supposons par l'absurde qu'il existe un indice tel que $x_i \neq y_i$. On note k le plus petit de ces indices et donc :

$$x_k \dots x_p = y_k \dots y_q$$

Au vu des éléments de L_4, on a quatre possibilités :

$$x_k = b \text{ et } y_k = baa \text{ (ou inversement)} \qquad \text{ou} \qquad x_k = ab \text{ et } y_k = abaa \text{ (ou inversement)}$$

On se place dans le premier cas (les autres se traitent de manière similaire). Montrons par itération finie sur $i \in [\![k+1, \min(p,q)]\!]$ que $x_i = y_j = aaaa$:

→ Initialisation. Pour $i = k+1$, on remarque que $w_0 = x_k \dots x_p = y_k \dots y_q$ a pour préfixes bx_{k+1} et baa. Ainsi, la seule possibilité est que $x_{k+1} = aaaa$.
Par suite, $baaaa$ et $baay_{k+1}$ sont des préfixes de w_0. La seule possibilité est que $y_{k+1} = aaaa$.

→ Hérédité. Soit $i \in [\![k+1, \min(p,q)-1]\!]$. Si la propriété est vraie au rang i, alors $ba^{4(i-k)}x_{i+1}$ et $ba^{4(i-k)+2}$ sont des préfixes de w_0 et donc $x_{i+1} = aaaa$. Par suite, $y_{i+1} = aaaa$.

On aboutit alors à la contradiction $x_k \dots x_p \neq y_k \dots y_q$ en calculant les tailles de ces deux mots :

- Si $p \leqslant q$ alors :

$$|y_k \dots y_q| \geqslant |y_k \dots y_p| = 3 + 4(p-k) > 1 + 4(p-k) = |x_k \dots x_p|.$$

- Sinon, $p \geqslant q+1$ et :

$$|x_k \dots x_p| \geqslant |x_k \dots x_{q+1}| = 5 + 4(q-k) > 3 + 4(q-k) = |y_k \dots y_q|.$$

Question 2 – Si $\epsilon \in L$, alors l'égalité $\epsilon = \epsilon \cdot \epsilon$ contredit directement la définition de code.

Question 3 – Il s'agit de montrer que pour tout $(p, q) \in \mathbb{N}^2$, si $u^p = u^q$, alors $p = q$. Il suffit de remarquer que si $u^p = u^q$, alors $p|u| = |u^p| = |u^q| = q|u|$ et donc $p = q$ car $|u| \neq 0$.

Question 4 – On le montre par double implication

($\Rightarrow$) Montrons la contraposée : l'égalité $uv = vu$ contredit directement la définition de code.

($\Leftarrow$) Montrons la contraposée par récurrence sur $|u| + |v|$: si $\{u, v\}$ n'est pas un code, alors $uv = vu$.

Soit $n \geqslant 2$. On suppose la propriété vraie pour tout $n' < n$ et on la montre au rang n. Soit u, v distincts non vides tels que $\{u, v\}$ n'est pas un code. Soient $(x_i)_i$ et $(y_j)_j$ des mots de $\{u, v\}$ tels que :

$$x_1 \dots x_p = y_1 \dots y_q$$

et tels qu'il existe un indice tel que $x_i \neq y_i$. On note k le plus petit de ces indices et donc :

$$x_k \dots x_p = y_k \dots y_q$$

Quitte à renommer, on a $|u| \leqslant |v|$, $x_k = u$ et $y_k = v$, donc v est un préfixe de u. Ainsi, il existe v' tel que $v = uv'$. On a plusieurs cas :

- Si v' est vide, alors $v = u$ et donc $uv = vu$ (CQFD)
- Si $v' = u$, alors $v = uu$ et donc $uv = vu$ (CQFD)
- Sinon, peut appliquer l'hypothèse de récurrence sur u et v'.

Dans ce dernier cas, on obtient :

$$u \cdot x_{k+1} \dots x_p = u \cdot v' \cdot y_{k+1} \dots y_q \qquad \text{et donc} \qquad x_{k+1} \dots x_p = v' \cdot y_{k+1} \dots y_q$$

Comme les x_i et y_i appartiennnent à $\{u, uv'\}$, il existe des mots x'_i et des mots y'_j de $\{u, v'\}$ tels que :

$$x'_1 \dots x'_r = y'_1 \dots y'_s \qquad \text{où} \qquad x'_1 = u \text{ et } y'_1 = v'$$

Ainsi, $\{u, v'\}$ n'est pas un code et donc $uv' = v'u$ par l'hypothèse de récurrence. On peut conclure :

$$uv = uuv' = uv'u = vu$$

Exercice 6. Mots de Bruijn

Question 1 –

Pour $k = 1$, on a $\Sigma^1 = \{a, b\}$, donc $u = ab$ convient.
Pour $k = 2$, on a $\Sigma^2 = \{aa, ab, ba, bb\}$, donc $u = aabb$ convient.
Pour $k = 3$, on a $\Sigma^2 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$, donc $u = aaababbb$ convient.

Question 2 – Si u est un mot de Bruijn, alors par définition, Σ^k est en bijection avec $[\![0, |u| - 1]\!]$. Ainsi :

$$\boxed{|u| = |\Sigma|^k}$$

Question 3 – Commençons par vérifier que G_k satisfait les hypothèses du théorème d'Euler :

- Le graphe est fortement connexe. En effet, soient $u = u_1 \dots u_k$ et $v = v_1 \dots v_k$ deux mots. Pour chaque $i \in [\![0; k]\!]$, on définit le mot w_k par :

$$w_k = (u_1 \dots u_{k-i}) \cdot (v_{k-i+1} \dots v_k)$$

  Alors $(w_0, w_1, \dots, w_k)$ est un chemin de u à v dans G_k.
- Pour tout mot $u = u_1 \dots u_k$, les successeurs (resp. prédécesseurs) de u dans G_k sont tous les mots de la forme $u_1 \dots u_{k-1}c$ (resp. $cu_2 \dots u_k$) avec $c \in \Sigma$. Ainsi, le degré entrant et le degré sortant de u sont tous les deux égaux à $|\Sigma|$.

Par le théorème d'Euler, G_k admet un cycle passant une et une seule fois par chaque arc.

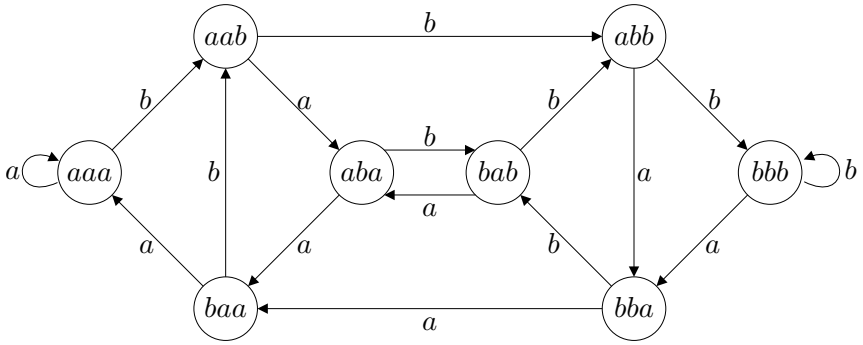
Pour construire un mot de Bruijn B d'ordre $k + 1$:

- On associe à chaque arc (u, v) de G_k une étiquette égale à la dernière lettre de v.
- On construit un cycle γ passant une et une seule fois par chaque arc de G_k.
- Le mot B est alors la suite des étiquettes des arcs parcourus dans l'ordre de γ.

Montrons que B est bien un mot de Bruijn d'ordre $k+1$. Le mot B a la bonne taille puisqu'il y a $|\Sigma| \times |V_k| = |\Sigma|^{k+1}$ arcs dans G_k. Il suffit donc de montrer que tout $u \in \Sigma^{k+1}$ est un facteur circulaire de B.

On remarque que dans le graphe G_k, si $(w_0, w_1, w_2 \dots, w_k)$ est un chemin de longueur k, alors les k arcs de ce chemin sont étiquetés par les lettres de w_k. Pour tout $u = u_1 \dots u_{k+1} \in \Sigma^{k+1}$, le cycle γ passe $|\Sigma|$ fois par $u' = u_1 \dots u_k \in V_k$ et chaque éléments de Σ étiquette l'un des arcs sortants de u'. En particulier, l'un des arcs sortants de u' noté a est étiqueté par u_{k+1}. Si on note γ' le seul sous-chemin de γ de longueur $k+1$ dont le dernier arc est a, alors la suite des étiquettes des arcs parcourus dans l'ordre de γ' est u. Ainsi, u est un mot circulaire de B.

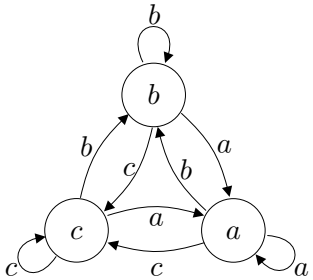
Question 4 – Pour $\Sigma = \{a, b\}$, le graphe G_3 est :



En partant du sommet aaa, on obtient le mot de Bruijn d'ordre 4 :

$$abbbbaababbabaaa$$

Pour $\Sigma = \{a, b, c\}$, le graphe G_1 est :



En partant du sommet a, on obtient le mot de Bruijn d'ordre 2 :

$$accbbabca$$

Question 5 –

→ On représente un mot par une liste d'entiers.

→ Attention, la liste est dans l'ordre inverse par rapport au mot : le premier (rep. dernier) élément de la liste est la dernière (resp. première) lettre du mot.

→ L'algorithme serait plus efficace si les mots étaient représentés par des listes doublement chaînées et non des listes simplement chaînées.

→ Le graphe G_k est représenté par listes d'adjacences. Elles sont stockées dans une table de hachage qui à chaque mot de taille k associe la liste de ses successeurs dans G_k.

Étape 1. Génération des mots de taille k.

```
(* Prend en entrée la liste de tous les mots de taille k et renvoie la liste de
   tous les mots de taille k+1 *)
let rec ajout_int2 (li: mot list) n = match n with
  | 0 -> []
  | n -> (List.map (fun l -> n :: l) li) @ (ajout_int2 li (n-1));;
```

```
let rec tous_mots n k: mot list = match k with
  | 0 -> [[]]
  | k -> let li = tous_mots n (k-1) in
         ajout_int2 li n;;
```

Étape 2. Création du graphe G_k.

```
(* Supprime la dernière lettre d'un mot *)
let rec suppr_dernier (li: mot): mot = match li with
  | [] -> failwith "suppr_dernier: liste vide"
  | e :: [] -> []
  | e :: q -> e :: suppr_dernier q;;
```

```
(* Renvoie la liste des voisins d'un mot *)
let vois n (m : mot): mot list =
  let m1 = suppr_dernier m in
  List.init n (fun c -> (c+1) :: m1);;
```

```
let make_graph n k: graphe =
  let li_mots = tous_mots n k in
  let g = Hashtbl.create (List.length li_mots) in
  List.iter (fun s -> Hashtbl.add g s (vois n s)) li_mots;
  g;;
```

Étape 3. Algorithme de Hierholzer.

```
(* Cette fonction renvoie un cycle de g entre le sommet s0 et lui-même.
   Hypothèse: s0 a au moins un voisin dans g.
   Attention: modifie g en supprimant les arcs empruntées *)
let find_cycle (g: graphe) (s0: mot): mot list  =
  let rec parc (s: mot) =
    let vois = Hashtbl.find g s in
    let t = List.hd vois in
    Hashtbl.replace g s (List.tl vois);
    if t = s0 then [s; s0] else s :: (parc t)
  in
  parc s0;;
```

```
(* Cette fonction prend entrée un chemin c et coupe ce chemin en trois
   parties: c1, m, c2 où:
   - m est le premier sommet du chemin ayant encore un successeur dans g. Si
     aucun sommet du chemin n'a de successeur, la fonction renvoie m = None.
   - c1 contient les sommets avant m dans c.
   - c2 contient les sommets après m dans c. *)
let rec split_chemin (g: graphe) (c: mot list):
          mot list * (mot option) * mot list = match c with
  | [] -> [], None, []
  | m1 :: q -> match Hashtbl.find g m1 with
               | [] -> let c1, m2 ,c2 = split_chemin g q in
                       m1 :: c1, m2, c2
               | _ -> [], Some m1, q;;
```

```
let hierholzer (g: graphe) k =
  let cycle0 = [List.init k (fun _ -> 1)] in
  let rec ajout (cycle: mot list) = match split_chemin g cycle with
    | c, None, [] when c = cycle -> cycle
    | _, None, _ -> failwith "hierholzer: ne devrait pas arriver"
    | c1, Some m, c2 -> ajout (c1 @ (find_cycle g m) @ c2)
  in
  ajout cycle0;;
```

Étape 4. Construction du mot de Bruijn

```
let make_mot_bruijn n k: mot =
  if k < 1 then failwith "make_mot_bruijn: k < 1";
  if k = 1 then List.init n (fun i -> i+1) else
    let g = make_graph n (k-1) in
    let cycle = hierholzer g (k-1) in
    List.map List.hd (List.tl cycle);;
```