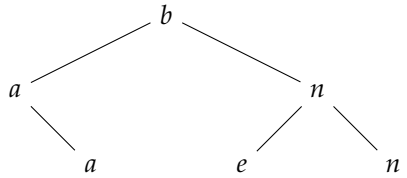
DS n° 02 — corrigé

I Classes sylvestres

I.1 Préliminaires

Q 1) On obtient l'arbre :



Q 2) On a $\mathsf{Syl}(T_1) = \{cabd, cadb, cdab\}$.

Q 3) On suit la définition en faisant bien attention aux inégalités et en respectant au mieux les notations de l'énoncé.

OCaml
```
let rec insertion_lettre a t =
  match t with
  | V -> N (V, a, V)
  | N (tg, r, td) ->
     if a < r then N (insertion_lettre a tg, r, td)
     else N (tg, r, insertion_lettre a td)
```

Q 4) Encore une fois on suit directement la définition en conservant les notations de l'énoncé.

OCaml
```
let rec insertion_mot u t =
  match u with
  | [] -> t
  | a :: v -> insertion_mot v (insertion_lettre a t)
```

Q 5) On trouve le mot « garnement ». Attention à ne pas confondre avec l'ordre infixe qui aurait donné « rangement ».

Q 6) On montre ce résultat par induction sur les arbres. On remarque également que pour tous mots $u, v \in \Sigma^*$ et pour tout arbre T on a $T \leftarrow uv = (T \leftarrow u) \leftarrow v$ ce que l'on peut montrer par induction sur u.

- si $T = \circ$, alors $\mathsf{pref}(T) = \varepsilon$ et $\circ \leftarrow \varepsilon = \circ = T$;
- supposons le résultat établi pour deux ABR T_g et T_d et soit $r \in \Sigma$ tel que $T = (T_g, r, T_d)$ soit un ABR. Alors $\mathsf{pref}(T) = r \cdot \mathsf{pref}(T_g) \cdot \mathsf{pref}(T_d)$. Dès lors, $\circ \leftarrow \mathsf{pref}(T) = (\circ \leftarrow r) \leftarrow \mathsf{pref}(T_g) \cdot \mathsf{pref}(T_d) = ((\circ, r, \circ) \leftarrow \mathsf{pref}(T_g)) \leftarrow \mathsf{pref}(T_d)$ par définition de l'insertion d'un mot dans un arbre et la remarque ci-dessus.
  - ⋆ sachant que toutes les étiquettes de T_g sont strictement inférieures à r, on en déduit que $(\circ, r, \circ) \leftarrow \mathsf{pref}(T_g) = (\circ \leftarrow \mathsf{pref}(T_g), r, \circ) = (T_g, r, \circ)$ par hypothèse d'induction ;
  - ⋆ sachant que toutes les étiquettes de T_d sont supérieures ou égales à r, on en déduit que $(T_g, r, \circ) \leftarrow \mathsf{pref}(T_d) = (T_g, r, \circ \leftarrow T_g) = (T_g, r, T_d)$ par hypothèse d'induction.

  On a bien montré que $\circ \leftarrow \mathsf{pref}(T) = T$.

On conclut par induction.

I.2 Contruction des classes sylvestres

Q 7) On a $abba \bowtie ba = \{ababba, abbaba, abbbaa, baabba, bababa, babbaa\}$.

Q 8) On peut utiliser la fonction `List.map` :

OCaml
```
let ajout_lettre a lst = List.map (fun v -> a :: v) lst
```

Q 9) On suit la définition, en respectant au mieux les notations de l'énoncé.

OCaml
```
let rec melange u v =
  match u, v with
  | [], w | w, [] -> [w]
  | a :: x, b :: y ->
     ajout_lettre a (melange x v) @
       ajout_lettre b (melange u y)
```

Q 10) Il s'agit d'un calcul de produit cartésien. Il faut faire attention ici à ne pas faire trop d'appel récursifs inutiles pour limiter le temps d'exécution et l'apparition de doublons.

OCaml
```
let rec melange_ens l m =
  match l, m with
  | [], _ | _, [] -> []
  | u :: l_suite, v :: m_suite ->
     melange u v @
       melange_ens [u] m_suite @
       melange_ens l_suite m
```

Q 11) Il suffit ici de suivre la définition. On prend bien garde au cas de base, car $\mathrm{Syl}(\circ) = \{\varepsilon\}$ et non $\varnothing$.

OCaml
```
let rec sylvestre t =
  match t with
  | V -> [[]]
  | N (tg, r, td) ->
    ajout_lettre r (melange_ens (sylvestre tg)
                                (sylvestre td))
```

I.3 Caractérisation des classes sylvestres

Q 12)
- Réflexivité : $u \sim_S u$, en utilisant la suite de mot réduite à $w_0 = u$;
- Symétrie : la définition de S-adjacence est clairement symétrique de par la présence du « ou » dans sa définition. Dès lors, si $u \sim_S v$ via la suite de mots $w_0, w_1, \dots, w_n$, alors $v \sim_S u$ via la suite de mots $w_n, w_{n-1}, \dots, w_0$;
- Transitivité : supposons $u \sim_S v$ via la suite $x_0, \dots, x_n$ et $v \sim_S w$ via la suite $y_0, \dots, y_m$. Posons $z_0, \dots, z_{n+m}$ la suite de mot définie par :
  - ⋆ si $i \in [\![0, n]\!]$, $z_i = x_i$;
  - ⋆ sinon, $z_i = y_{i-n}$;

  Notons que $z_0 = x_0 = u$, $z_{n+m} = y_m = w$ et $z_n = x_n = y_0 = v$. De plus, pour $i \in [\![0, n+m-1]\!]$, z_i est bien S-adjacent à z_{i+1} (car $x_n = y_0$). On en déduit que $u \sim_S w$.

Q 13) On raisonne par induction :
- si $T = \circ$ alors il n'y a rien à faire puisque l'hypothèse n'est pas vérifiée ;
- supposons le résultat vrai pour deux sous-arbres T_g et T_d (attention, on ne sait pas si ces sous-arbres contiennent au moins b donc on ne peut pas utiliser la conclusion de l'hypothèse pour l'instant) et soit $T = (T_g, r, T_d)$ avec $r \in \Sigma$. Distinguons selon la valeur de r :
  - ⋆ si $c < r$, comme T contient la valeur b et que T est un ABR, alors $b \leq c < r$ apparait dans T_g. Par hypothèse d'induction sur T_g et vu l'insertion on a $T \leftarrow ac = (T_g \leftarrow ac, r, T_d) = (T_g \leftarrow ca, r, T_d) = T \leftarrow ca$;
  - ⋆ on raisonne symétriquement si $r \leq a$;
  - ⋆ sinon, $a < r \leq c$; dès lors, $T \leftarrow ac = (T_g \leftarrow a, r, T_d \leftarrow c) = T \leftarrow ca$ (pas besoin de l'hypothèse d'induction ici).

  Dans tous les cas, $T \leftarrow ac = T \leftarrow ca$.

Q 14) Notons qu'il suffit de montrer ce résultat pour deux mots u et v qui sont S-adjacents et de conclure par définition de la S-équivalence et décomposition de l'insertion. Soient alors u et v qui sont S-adjacent et trois lettres $a < b \leq c$ et trois mots $x, y, z \in \Sigma^*$ tels que $u = xbyacz$ et $v = xbycaz$ (sans perte de généralité, quitte à inverser les rôles de u et v). Alors $\circ \leftarrow u = ((\circ \leftarrow xby) \leftarrow ac) \leftarrow z = ((\circ \leftarrow xby) \leftarrow ca) \leftarrow z = \circ \leftarrow v$. La deuxième égalité découle de la question précédente, car $\circ \leftarrow xby$ contient la lettre b et est un ABR.

II Circuits eulériens

Q 15) Oui/non/non.

Q 16) Notons $x_0 x_1 \dots x_n$ un cycle eulérien du graphe, avec donc $x_0 = x_n$. Chaque sommet de ce chemin est incident à un nombre pair d'arêtes dans ce chemin (la précédente et la suivante de chaque occurrence de ce sommet en identifiant pour x_0 sa première et dernière occurrence). Comme chaque arête apparaît exactement une fois, le degré d'un sommet dans le graphe est précisément le nombre d'arêtes incidentes dans ce cycle eulérien et est donc pair.

Q 17) On calcule le degré de chaque sommet et on vérifie qu'il est bien pair. Le programme est à la page suivante.

Q 18) Comme le graphe possède au moins deux sommets, il existe un sommet $v \in S \setminus \{v_0\}$. Comme le graphe est connexe, il existe un chemin de v_0 à v. La première arête de ce chemin est incidente à v_0.

C

```c
bool is_eulerian(void) {
  for (int i = 0; i < N; i++) {
    int d = 0;
    for (int j = 0; i < N; j++) {
      if (graph[i][j] != NULL) {d++;}
    }
    if (d % 2 != 0) {return false;}
  }
  return true;
}
```

Q 19) Suivont l'indication et raisonnons par l'absurde en supposant qu'il n'existe pas de cycle passant par v_0. Montrons que pour tout $k \in \mathbb{N}$ on peut construire un chemin $v_0 - v_1 - \cdots - v_k$ comportant des arêtes deux à deux distinctes et sans repasser par v_0. Pour $k = 0$ le chemin v_0 convient. Pour $k = 1$, d'après la question précédente, il existe un sommet v_1 incident à v_0 et le chemin $v_0 - v_1$ convient donc. Supposons le résultat établi pour $k \geq 1$ et notons $v_0 - v_1 - \cdots - v_k$ un tel chemin. Comme $v_k \neq v_0$, le sommet v_k est incident à un nombre impair d'arêtes dans ce chemin (toutes deux à deux distinctes). Comme le degré de v_k est pair, il existe une arête $v_k - v_{k+1}$ qui n'est pas dans ce chemin. De plus, $v_{k+1} \neq v_0$ sinon il y aurait un cycle passant par v_0. On a donc construit un chemin convenable de longueur $k + 1$. Ce résultat est donc vrai pour tout $k \in \mathbb{N}$, c'est-à-dire que l'on dispose d'un chemin d'arêtes deux à deux distinctes arbitrairement grand, ce qui est absurde puisque le nombre d'arêtes du graphe est fini.

Q 20) On avance tant que l'on n'est pas revenu au début.

OCaml

```ocaml
edge *roud_trip(int start) {
  edge *first = any_edge_from(start);
  edge *curr = first;
  while (e->dst != start) {
    edge *next = any_edge_from(e->dst);
    connect(curr, next);
    curr = next;
  }
  connect(curr, first);
  return first;
}
```

Il ne faut pas oublier de bien renvoyer la toute première arête et de connecter la dernière arête du cycle avec la première.

Q 21) On parcourt les arêtes du cycle à la recherche d'un sommet ayant encore des arcs sortants.

OCaml

```ocaml
edge *find_vertex_with_edge(edge *c) {
  if (has_edge(c->src)) {return c;}
  edge *e = c->next;
  while (e != c) {
    if (has_edge(e->src)) {return e;}
    e = e->next;
  }
  return NULL;
}
```

Q 22) On suit l'algorithme.

OCaml

```ocaml
edge *eulerian_cycle(int start) {
  edge *cycle = round_trip(start);
  edge *next = find_vertex_with_edge(cycle);
  while (next != NULL) {
    join(next, round_trip(next->src));
    next = find_vertex_with_edge(next);
  }
  return cycle;
}
```

Q 23) Les fonctions `has_edge` et `any_edge_from` ont chacune une complexité en $\mathcal{O}(N)$. La fonction `round_trip` a un coût de $\mathcal{O}(N)$ pour chaque arc qu'elle construit. La fonction `find_vertex_with_edges` parcourt dans le pire cas le cycle en entier pour un coût $\mathcal{O}(N)$ pour chaque arête du cycle. On note $C_1, C_2, \ldots, C_k$ les tailles successives des cycles construits par `round_trip`. On a donc $E = C_1 + C_2 + \cdots + C_k$. Le coût de la i^e itération (qui consiste donc à construire un cycle puis à trouver un éventuel sommet avec des arcs restants) est donc

$$\mathcal{O}(N \cdot C_i + N \cdot (C_1 + C_2 + \cdots + C_i)) = \mathcal{O}(NE)$$

d'où un total de

$$\mathcal{O}(\sum_{i=1}^{k} N \cdot E) = \mathcal{O}(NE^2)$$

car $k \leq E$. On peut vérifier que cette complexité est atteinte par exemple par une succession de cycles de longueur 3.

Q 24) Soit un graphe admettant un chemin eulérien du sommet a au sommet b. On ajoute un nouveau sommet c et deux arêtes $a - c$ et $c - b$. On a alors un cycle eulérien et donc uniquement des sommets de degré pair. En supprimant le sommet c et les deux arêts $a - c$ et $c - b$ on a bien au plus deux sommets de degré impair (aucun dans le cas $a = b$, c'est-à-dire lorsque le chemin était un cycle). Pour la réciproque on peut procéder exactement de la même façon. Si le graphe a deux sommets de degrés impairs a et b, on ajoute un nouveau sommet c et deux arêtes $a - c$ et $c - b$. On a alors uniquement des sommets de degré pair et donc un cycle eulérien, que l'on peut construire avec l'algorithme de Hierholzer. En supprimant les arêtes $a - c - b$, on obtient un chemin eulérien de a à b. C'est bien là un algorithme qui construit un chemin eulérien. Et dans le cas où le graphe n'a aucun sommet de degré impair, on a un cycle eulérien par les questions précédentes, donc un chemin eulérien.

Q 25) On construit un graphe ayant un sommet pour chaque pièce, ainsi qu'un sixième sommet pour l'extérieur. On relie ensuite deux sommets dès qu'il y a une porte entre les deux. Il y a plusieurs arêtes entre deux sommets ce qui n'est pas un problème en soi : on peut rajouter un sommet intermédiaire sur chacune de ces arêtes pour les distinguer (sommets de degré 2). On se retrouve alors avec quatre sommets de degré impair (trois de degré 5 et un de degré 9). En vertu de la question précédente, il n'existe pas de chemin eulérien pour ce graphe et donc pas de chemin traversant les cinq pièces.

— FIN DU SUJET —