Question 10. Soit u un mot reconnu par A ; il existe donc un chemin étiqueté par u qui mène de q_0 à un état acceptant. Si ce chemin ne passe pas par q il est encore présent dans S(A, q) et u est reconnu par S(A, q).

S'il passe par q il contient un ou plusieurs chemins de la forme $q' \xrightarrow{x} q \xrightarrow{y} q''$. Or ces chemins sont remplacés dans S(A, q) par des chemins de la forme $q' \xrightarrow{x} r \xrightarrow{y} q''$, donc le mot u est bien reconnu par S(A, q).

Réciproquement, considérons un mot u reconnu par S(A, q), et un chemin acceptant étiqueté par u.
Si ce chemin ne contient que des sommets de A ce chemin est toujours présent dans A et u est reconnu par A.

Dans le cas contraire il contient un ou plusieurs chemins de la forme $q' \xrightarrow{x} r \xrightarrow{y} q''$ avec $r \notin$ A. Mais ces états r ne possèdent qu'un transition entrante et une transition sortante donc par construction on doit avoir $q' \in$ A et $q'' \in$ A, et ainsi la transition $q' \xrightarrow{x} q \xrightarrow{y} q''$ existe dans A, ce qui montre que A reconnait u.

Question 11. Considérons un automate A qui reconnait le langage P(L) (rationnel d'après la question 7). D'après la question 8 il n'existe pas de transition de la forme (q, x, q) où q est un état utile. La question 10 permet de construire progressivement un automate A′ qui reconnait P(L) et qui possède la propriété suivante : « tout état qui n'est ni initial ni acceptant n'est extrémité que d'une unique transition et n'est à l'origine que d'une unique transition ».
Considérons maintenant un état utile r de A′ qui n'est ni initial ni acceptant. Il existe deux uniques états q et q' et deux transitions de la forme (q, x, r) et (r, y, q'). Transformons ces deux transitions en (q, y, r) et (r, x, q'). L'automate A″ ainsi construit reconnait le langage ϕ(P(L)), ce qui prouve que ce dernier est rationnel.

Question 12. Soit A un automate qui reconnait L, F l'ensemble de ses états acceptants. On note F′ l'ensemble des états q pour lesquels il existe une transition étiquetée par x et menant à un état de F. Alors l'automate A′ = $(\Sigma, Q, T, \{q_0\}, F')$ est un automate qui reconnait le langage M(L, x), qui est donc rationnel.

Question 13. Soit $u \in$ I(L). Alors u s'écrit $u = va$ avec $v \in$ M(I(L), a) ou $u = wb$ avec $w \in$ M(I(L), b).
Ainsi, I(L) = M(I(L), a)a + M(I(L), b)b, l'inclusion réciproque étant évidente.
Mais les mots v et w sont de longueurs paires donc d'après la question 1, $\phi(u) = \phi(v)a$ ou $\phi(u) = \phi(w)b$ selon les cas. On a donc

$$\phi(\text{I(L)}) = \phi(\text{M(I(L)}, a))a + \phi(\text{M(I(L)}, b))b.$$

Question 14. D'après la question 7, I(L) est un langage rationnel ; d'après la question 12 il en est de même des langages M(I(L), a) et M(I(L), b). Mais ces langages ne contiennent que des mots de longueurs paires donc P(M(I(L), a)) = M(I(L), a) et P(M(I(L), b)) = M(I(L), b) et d'après la question 11, les langages ϕ(M(I(L), a)) et ϕ(M(I(L), b)) sont rationnels. D'après la question 13 le langage ϕ(I(L)) est donc rationnel.
Enfin, d'après la question 11 le langage ϕ(P(L)) est rationnel donc ϕ(L) = ϕ(I(L)) + ϕ(P(L)) est rationnel.

Question 15. L'égalité L = $\phi(\phi$(L)) montre que si ϕ(L) est rationnel il en est de même de L.

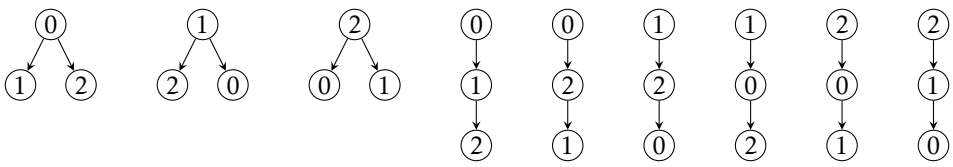
Question 16. On définit la fonction :

```
let rec phi = function
  | []      -> []
  | [x]     -> [x]
  | x::y::q -> y::x::(phi q) ;;
```

Partie II. Algorithmique

II.1 D'un codage racine-fils-frères d'un arbre au codage de Prüfer

Question 17. Il y a 9 arbres à trois nœuds et étiquetés consécutivement ; il sont représentés ci-dessous.



Question 18.

```
let calculer_peres racine fils freres =
  let n = vect_length fils in
  let peres = make_vect n (-1) in
  let rec aux p i =
    peres.(i) <- p ;
    if freres.(i) <> -1 then aux p freres.(i) ;
    if fils.(i) <> -1 then aux i fils.(i)
  in aux racine fils.(racine) ;
  peres ;;
```

Question 19. Chaque case des tableaux **`fils`** et **`freres`** est comparé une fois à la valeur -1 donc le coût de cette fonction est proportionnel au nombre de nœuds : la complexité est linéaire.

Question 20.

```
let calculer_arites fils freres =
  let n = vect_length fils in
  let arites = make_vect n 0 in
  let rec aux i = function
    | -1 -> ()
    | j  -> arites.(i) <- arites.(i) + 1 ; aux i freres.(j)
  in
  for i = 0 to n-1 do
    if fils.(i) <> -1 then aux i fils.(i) done ;
  arites ;;
```

Question 21. Le coût de la fonction ci-dessus est proportionnel à la somme des valeurs du tableau **`arites`** c'est-à-dire à $n-1$: la complexité est linéaire.

Question 22.

```
let rec inserer table nb d = match nb with
  | 0                          -> table.(0) <- d
  | _ when table.(nb-1) >= d -> table.(nb) <- d
  | _                          -> table.(nb) <- table.(nb-1) ;
                                  inserer table (nb-1) d ;;
```

Question 23. Le coût de cette fonction est proportionnel au nombre d'éléments strictement inférieurs à d : la complexité est un $O(nb)$.

Question 24. Le codage de Prüfer de l'arbre A_3 est : $9,1,6,7,1,3,7,9,9,3$.

Question 25.

```
let calculer_prufer racine fils freres =
  let n = vect_length fils in
  let peres = calculer_peres racine fils freres in
  let arites = calculer_arites fils freres in
  let feuilles = make_vect n (-1) in
  let nb = ref 0 in
  let prufer = make_vect (n-1) (-1) in
  for i = n-1 downto 0 do
    if fils.(i) = -1 then (feuilles.(!nb) <- i ; incr nb)
  done ;
  for k = 0 to n-2 do
    let f = feuilles.(!nb-1) in
    let p = peres.(f) in
    prufer.(k) <- p ;
    arites.(p) <- arites.(p) - 1 ;
    if arites.(p) = 0 then inserer feuilles !nb p else decr nb
  done ;
  prufer ;;
```

Le vecteur **feuilles** contient la liste triée des feuilles de l'arbre dont il faut calculer le codage de Prüfer ; la référence **nb** contient le nombre de feuilles de ce dernier. La première itération engendre la liste des feuilles initiales, la seconde applique l'algorithme décrit par l'énoncé.

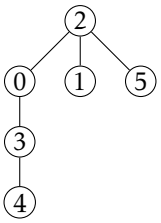
Question 26. Les opérations effectuées dans la première itération sont de coût constant donc le coût de celle-ci est linéaire.
Dans la seconde itération, seule la fonction **inserer** n'est pas de coût constant mais linéaire, on peut donc affirmer que le coût total de la fonction est un $O(n^2)$. Cependant, au cours de l'algorithme le nombre b d'éléments dans le tableau des feuilles n'excède jamais le nombre de feuilles initial b_0, ce qui permet d'affirmer que le coût est plus précisément un $O(b_0 n)$.

II.2 D'un codage de Prüfer d'un arbre à un codage racine-fils-freres

Question 27. Chaque élément qui apparaît dans le tableau **prufer** est le père d'un nœud qui a été supprimé lors de l'application de l'algorithme de calcul du codage de Prüfer ; son arité est donc égale à son nombre d'occurrences dans ce tableau.

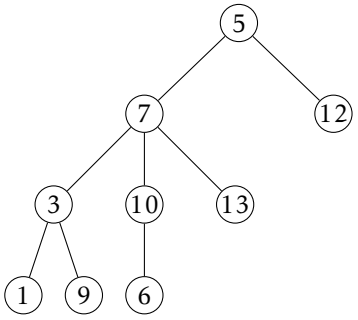
```
let calculer_arites_par_Prufer prufer =
  let n = vect_length prufer + 1 in
  let arites = make_vect n 0 in
  for i = 0 to n-2 do
    arites.(prufer.(i)) <- arites.(prufer.(i)) + 1
  done ;
  arites ;;
```

Question 28. D'après la question précédente, les feuilles de l'arbre A sont les éléments de $[\![0,5]\!]$ qui n'apparaissent pas dans le codage de Prüfer : dans le cas présent il s'agit des nœuds 1, 4 et 5.
2 est donc le père de 1, et $3,0,2,2$ est le codage de Prüfer de l'arbre A′ étiqueté par $\{0,2,3,4,5\}$.
Les feuilles de A′ sont 4 et 5 donc 3 est le père de 4 et $0,2,2$ est le codage de Prüfer de l'arbre étiqueté par $\{0,2,3,5\}$.
Les feuilles de cet arbre sont 3 et 5 donc 0 est le père de 3 et $2,2$ est le codage de Prüfer de l'arbre étiqueté par $\{0,2,5\}$.
Les feuilles de cet arbre sont 0 et 3 donc 2 est le père de 0 et 2 est le codage de Prüfer de l'arbre étiqueté par $\{2,5\}$.
Ce dernier arbre a pour père 2 et pour fils 5 donc l'arbre recherché A est l'arbre :



Question 29. La démarche n'est pas différente de celle décrite à la question précédente. À chaque étape on détermine la liste des feuilles, à savoir les étiquettes non encore utilisées et qui ne figurent pas dans le codage de Prüfer. La plus petite a pour père le premier terme du codage. On détermine ainsi successivement que :

- 3 est le père de 1 ;
- 10 est le père de 6 ;
- 3 est le père de 9 ;
- 7 est le père de 3 ;
- 7 est le père de 10 ;
- 5 est le père de 12 ;
- 7 est le père de 13 ;
- 5 est la racine de l'arbre et le père de 7.



Question 30. Nous allons appliquer une démarche un peu différente pour coder la fonction demandée : nous allons calculer le tableau des arités à l'aide de la fonction écrite à la question 27 et maintenir à jour le vecteur des feuilles disponibles en utilisant la fonction **inserer**. Ainsi, pour chaque valeur de la suite de Prüfer on connaîtra la plus petite des étiquettes des feuilles disponibles, ce qui nous permettra de remplir le tableau **fils** ou **freres**. On supprime ensuite cette feuille en diminuant d'une unité l'arité de son père, et si cette dernière passe à 0 on introduit cette valeur dans le tableau des feuilles.

```
let calculer_arbre prufer fils freres =
  let n = vect_length fils in
  fill_vect fils 0 n (-1) ;
  fill_vect freres 0 n (-1) ;
  let arites = calculer_arites_par_Prufer prufer in
  let feuilles = make_vect n (-1) in
  let nb = ref 0 in
  for i = 0 to n-1 do
    if arites.(i) = 0 then (inserer feuilles !nb i ; incr nb)
  done ;
  let rec aux k = function
    | j when freres.(j) = -1 -> freres.(j) <- k
    | j                      -> aux k freres.(j) in
  for k = 0 to n-2 do
    let p = prufer.(k) and f = feuilles.(!nb-1) in
    decr nb ;
    if fils.(p) = -1 then fils.(p) <- f else aux f fils.(p) ;
    arites.(p) <- arites.(p) - 1 ;
    if arites.(p) = 0 then (inserer feuilles !nb p ; incr nb)
  done ;
  prufer.(n-2) ;;
```

Question 31. Montrons par récurrence sur $n = |E|$ que Pr est injective.

- Si $n = 1$ un seul arbre est possible donc le résultat est évident.
- Si $n > 1$, supposons le résultat acquis au rang $n-1$, et considérons deux arbres A et B ayant même codage de Prüfer. Le plus petit entier x absent du codage est la plus petite étiquette des feuilles de ces deux arbres. Notons A' et B' les arbres obtenus en supprimant cette feuille. Alors A' et B' ont même codage de Prüfer donc par hypothèse de récurrence ils sont égaux. Or x a même père dans A et dans B (le premier terme du codage), donc A = B.

Montrons par récurrence sur $n = |E|$ que Pr est surjective.

- Si $n = 1$ la seule suite possible est la suite vide, qui code l'arbre réduit à sa racine et dont l'étiquette est l'unique valeur de E.
- Si $n > 1$, supposons le résultat acquis au rang $n-1$, et considérons $P \in \mathcal{S}(E)$. Soit x le plus petit des éléments de E qui ne soit pas dans P, soit p le premier terme de P, et soit P' la suite des autres termes de P. Par hypothèse de récurrence P' code un arbre A' étiqueté par $E \setminus \{x\}$. Considérons l'arbre A obtenu en ajoutant à A' un fils étiqueté par x au nœud p. Alors A est un arbre dont le codage de Prüfer est la suite P.

Ceci prouve que Pr est une bijection de $\mathcal{A}(E)$ sur $\mathcal{S}(E)$.

Question 32. Deux ensembles finis en bijection ont même cardinal donc $\operatorname{card}\mathcal{A}(E) = \operatorname{card}\mathcal{S}(E) = n^{n-1}$.