Centrale Informatique MPI 2023
Un corrigé

1 Problème du voyageur de commerce

Q 1. Le circuit $\boxed{\text{BCADB}}$ est un circuit hamiltonien visiblement minimal, de poids 1+1+1+2 = $\boxed{5}$.

Q 2. Pour $n \geqslant 3$, à chaque permutation de $[\![1 \ldots n]\!]$ correspond un circuit ; Il y en a donc $n!$ en distinguant le point de départ (n possibilités) et le sens de parcours (2 possibilités). Sans cette distinction, il y en a $2n$ fois moins , soit $\boxed{(n-1)!/2}$.

Tous les circuits sont de poids minimal si toutes les arêtes ont le même poids. En prenant un $\boxed{\text{poids de } 1}$ pour chaque arête, chaque circuit aura un poids n.

Q 3.

```c
int poids_chemin(struct Graphe g, struct Chemin c){
    int poids = 0, i, j;
    for(int k=1; k<c.longueur; k++){ // (|c|-1) arêtes
      i = c.l_sommets[k-1];
      j = c.l_sommets[k];
      poids += g.adj[i * g.V + j];
    }
    return poids;
}
```

Le temps d'exécution croit comme la longueur du chemin : $\boxed{T(g,c) = O(|c|)}$.

Q 4. Le problème de Décision du Voyageur de Commerce (DVC dans la suite) se définit ainsi :

— **Instance :** Un graphe non orienté G, un entier s.

— **Question :** Existe-il un circuit hamiltonien de poids p dans G, tel que $p \leqslant s$?

D'après la question **Q 3**, La vérification d'un chemin se fait en temps polynomial, donc

$$\boxed{\text{DVC} \in \text{NP}}$$

Q 5. Soit la réduction suivante : $I = (G(V,E), a, b)$ étant une instance du problème du chemin hamiltonien (CHH dans la suite), une instance $I' = G'(V', E')$ du problème du cycle hamiltonien (CIH dans la suite) est construite ainsi :

— $V' = V \cup \{z\}$, z étant un nouveau sommet.

— $E' = E \cup \{\{a,z\}, \{b,z\}\}$

Il est clair que le circuit $C' = zav_k \ldots v_\ell bz$ est hamiltonien si et seulement si le chemin $C = av_k \ldots v_\ell b$ hamiltonien, car bza est le seul moyen de visiter z.

Pour la suite, il faut une réduction polynomiale. La construction précédente se fait clairement en un temps linéaire en $|I|$, donc CHH se réduit pronominalement à CIH :

$$\boxed{\text{CHH} \leqslant_{\mathbf{P}} \text{CIH}}$$

Q 6. Soit la réduction suivante :

$I = G(V,E)$ étant une instance du problème du cycle hamiltonien, l'instance $I' = (G', s)$ du problème de décision du voyageur de commerce est construite ainsi :

— G' est le graphe complet ayant les mêmes sommets que G.

— le poids de chaque arête a de G' vaut 0 si $a \in E$, et 1 sinon.

— $s = 0$.

Reste à montrer que CIH$(I) = $ DVC(I') :

— CIH$(I) = 1 \implies$ DVC$(I') = 1$: Si C est un chemin hamiltonien dans G, alors c'est aussi un chemin hamiltonien de G', de poids nul par construction. donc I' est une instancer positive de DVC.

— DVC$(I) = 1 \implies$ CIH$(I') = 1$: Si C est un chemin hamiltonien dans G' de poids nul, alors il ne passe que par des arêtes de poids nul, soit des arêtes de G. donc C est hamiltonien dans G.

.

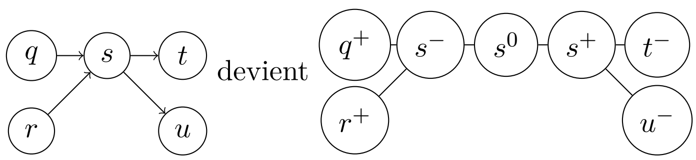
Pour la suite, il faut une réduction polynomiale. La construction précédente se fait clairement en un temps quadratique en $|I|$, d'où :

$$\boxed{\text{CIH} \leqslant_{\mathbf{P}} \text{DVC}}$$

Q 7. Soit $I = (G(V, E), a, b)$ une instance du problème du chemin hamiltonien orienté (CHHO dans la suite). Soit $I' = (G'(V', E'), a^-, b^+)$ l'instance du problème du chemin hamiltonien (non orienté) construite ainsi :

— Pour chaque sommet s de V sont créés trois sommets s^0, s^+ et s^- dans V', et deux arêtes $\{s^-, s^0\}$ et $\{s^0, s^+\}$ dans E'.

— Pour chaque arc (u, v) de E est créée l'arête $\{u^+, v^-\}$ non orientée dans E' :



Il reste à montrer que CHHO$(I) = $ CHH(I') :

— Il est clair que si $as_k \dots s_\ell b$ est hamiltonien dans G, alors $a^-a^0a^+s_k^-s_k^0s_k^+ \dots s_\ell^-s_\ell^0s_\ell^+b^-b^0b^+$ est hamiltonien dans G'.

— Si maintenant G' possède un chemin hamiltonien C' allant de a^- à b^+ : En partant de a^-, C' doit continuer par a^0, sinon ce sommet ne sera jamais visité par la suite. Puis de a_0 il doit atteindre a^+, car a^0 est de degré 2. Il doit ensuite continuer vers un sommet s_j^- car il n'existe pas d'arête $a^+s_j^+$ par construction, et le retour sur a^0 est impossible sur un chemin hamiltonien. Le même raisonnement peut se faire à partir de s_j^- : Le chemin est donc une succession de sous-chemins de la forme $s_k^-s_k^0s_k^+$, soit $a^-a^0a^+s_k^-s_k^0s_k^+ \dots s_\ell^-s_\ell^0s_\ell^+b^-b^0b^+$. Dès lors le chemin $C = as_k \dots s_\ell b$ associé est également hamiltonien dans G.

Cette construction est linéaire en la taille de I, d'où :

$$\boxed{\text{CHHO} \leqslant_{\mathbf{P}} \text{CHH}}$$

Q 8. Il suffit d' exhiber ces chemins :

1. $e_1e_2e_3s_3s_2s_1$.
2. e_1s_1 et $e_2e_3s_3s_2$.
3. e_1s_1, e_2s_2 et e_3s_3.

Q 9. La question est ambigüe. Une formulation plus explicite est : « Montrer que pour tout **modèle** de la formule, il existe un chemin hamiltonien orienté de v_1 à v_{m+1} dans le graphe G ».

Soit un premier chemin construit en passant pour tout i :

— par le graphe G_i^+ si $x_i = 1$ dans le modèle, ou par le graphe G_i^- si $x_i = 0$;

— par l'arête $e_{p_i} \to s_{p_i}$ dans chaque A_k.

Comme la formule est satisfaite, toutes les clauses le sont et donc chaque A_k est traversé entre une et trois fois. Il reste alors, en vertu de **Q 8** à réarranger le chemin dans chaque A_k pour passer une et une seule fois par chaque sommet. Le chemin ainsi construit est bien un chemin hamiltonien.

Q 10. La question est ambigüe. Une formulation plus explicite est « Montrer que pour chaque chemin hamiltonien orienté de v_1 à v_{m+1} dans le graphe G, on peut associer un **modèle** pour la formule ».

Comme il ne peut y avoir deux arcs sortants du même sommet sur un chemin hamiltonien, une valuation σ est définie sans ambiguïté ainsi : $[\![x_i]\!]_\sigma = 1$ si l'arc sortant de v_i emprunte G_i^+, $[\![x_i]\!]_\sigma = 0$ s'il emprunte G_i^-. Chaque A_k est ainsi atteint par au moins un arc car le chemin est hamiltonien, et cet arc rend la clause vraie selon σ par construction. Toutes les clauses étant satisfaites, σ est un modèle pour la formule.

Q 11. La construction de graphe proposée étant de complexité $O(mn)$, **Q 9** et **Q 10** permettent d'établir que $\boxed{\text{3SAT} \leqslant_{\mathbf{P}} \text{ChHO}}$.

Les questions précédentes permettent finalement de conclure que

$$\text{3SAT} \leqslant_{\mathbf{P}} \text{ChHO} \leqslant_{\mathbf{P}} \text{ChH} \leqslant_{\mathbf{P}} \text{CiH} \leqslant_{\mathbf{P}} \text{DVC}.$$

Comme 3SAT est NP-complet, et que de plus DVC $\in$ NP d'après **Q 3** (ainsi que CiH) , Il vient :

$$\boxed{\text{Les problèmes de décision du voyageur de commerce et du circuit hamiltonien sont NP-complets}}$$

Q 12. Le circuit hamiltonien de G est aussi un circuit hamiltonien de G', de poids n puisque ses n arêtes ont un poids 1 dans G'.

Q 13. Si G ne possède pas de circuit hamiltonien, alors tout circuit hamiltonien C de G' aura au moins une arête qui n'est pas dans G. D'où poids$(c) \geqslant (n-1) + n(1+\varepsilon) + 1 = n(2+\varepsilon)$. CQFD.

Q 14. Soit A un éventuel algorithme **polynomial** permettant de trouver une $1+\varepsilon$ approximation de VC. Sur le graphe G', $p^* = n$, A trouve donc une solution de poids inférieur à $n(1+\varepsilon)$ en temps polynomial. $n(1+\varepsilon) < n(2+\varepsilon)$, donc d'après la contraposée du résultat de **Q 13**, G' possède un cycle hamiltonien. Ceci résout donc un problème de NP en temps polynomial, donc P=NP.

Finalement, A existe $\implies$ P = NP. Soit, par contraposition :

$$\boxed{\text{P} \neq \text{NP} \implies \text{il n'existe pas de } 1+\varepsilon \text{ approximation au problème du voyageur de commerce}}$$

Q 15.

```c
struct Arete* liste_aretes(struct Graphe g){
    int n = g.V, k = 0;
    struct Arete* liste = malloc(n*(n-1)/2*sizeof(struct Arete));
    for(int i=0;i<n;i++){
        for(int j=i+1;j<n;j++){
            liste[k].s1 = i;
            liste[k].s2 = j;
            liste[k].p = g.adj[i*n + j];
            k++;
        }
    }
    return liste;
}
```

Q 16. On implémente généralement l'algorithme de KRUKSAL à l'aide d'une structure **union-find** : Sont donc supposées définies `bool find(int i)` et `void union_(int i,int j)` (`union` est réservé en C).

Il vient :

```
struct Graphe kruksal (struct Graphe g){
    int n = g.V;
    struct Graphe gk = alloue_graphe(n);
    int i, j, k = 0;
    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
           gk.adj[i*n + j] = 0; //initialisation
        }
    }
    struct Arete* liste = liste_aretes(g);
    tri_aretes(liste, n*(n-1)/2);
    for (int na = 1 ; na < n; na++){ // n-1 noeuds dans l'arbre
        struct Arete a = liste[k];
        i = a.s1;
        j = a.s2;
        if (find(i) != find(j)){
            gk.adj[i*n + j] = a.p;
            union_(i,j);
        }
        k++;
    }
    return gk;
```

Q 17. Soit n le nombre de sommets du graphe. Le graphe `g` est connexe par hypothèse, donc `liste` contient au moins $n-1$ arêtes.

1. `kruskal` **termine en renvoyant un arbre couvrant** : En effet, la boucle `for` continue tant que $n-1$ arêtes n'ont pas été sélectionnées, qui est exactement le nombre d'arêtes d'un arbre couvrant d'un graphe connexe. Si la liste d'arêtes était épuisée avant la fin de la boucle `for`, c'est qu'au moins une arête ne formant pas de cycle a été omise. Ceci est impossible puisque toutes les arêtes ne formant pas de cycle sont sélectionnées.
2. `kruskal` **renvoie un arbre couvrant minimal** : On considère pour cela l'invariant de boucle suivant : « les arêtes sélectionnées sont un sous-ensemble d'un arbre couvrant minimal M de `g` ». C'est vrai initialement car `gk` est vide au départ. Lorsqu'on ajoute une arête a :

   — Si l'arête est dans M l'invariant est conservé.

   — Sinon $M \cup \{a\}$ possède nécessairement n arêtes et donc un cycle contenant a et une autre arête a' non testée (sinon elle eut été sélectionnée,car sans a elle ne forme pas de cycle). Donc le poids de a est inférieur ou égal à celui de a' et $M' = (M \backslash \{a'\}) \cup \{a\}$ est encore minimal (en fait a et a' ont même poids). Donc $a \in M'$ minimal et l'invariant est conservé.

   Donc :

$$\boxed{\text{KRUSKAL termine et renvoie un arbre couvrant minimal}}$$

Q 18.

```
int degre(struct Graphe g, int i){
    int n = g.V, d = 0;
    for(int j=0;j<n;j++){
```

```c
        if(g.adj[i*n+j]>0) d++ ;
    }
    return d;
}
```

Q 19.

```c
int* sommets_impairs(struct Graphe g, int* nb_sommets){
    int n = g.V;
    *nb_sommets = 0;
    int* sommets = malloc(n*sizeof(int));
    for(int i=0; i<n; i++){
        if (degre(g,i) %2 > 0){
            sommets[*nb_sommets]=i;
            (*nb_sommets)++;
        }
    }
    return sommets;
}
```

Q 20. En appelant J l'ensemble des nœuds de degré pair, le lemme des poignées de main stipule :

$$\sum_{v\in I}\deg(v) = -\sum_{v\in J}\deg(v) + 2|E|$$

D'où $\sum_{v\in I}\deg(v)$ est pair, et donc $|I|$ est pair.
Comme toutes les villes sont reliées entre elles, le graphe G du problème est complet. $G_{|I} = (I, \{\{x,y\} \in E, (x,y) \in I^2\})$ l'est donc aussi, et donc il existe bien des couplages parfaits dans $G_{|I}$. L'un d'eux est de poids minimal car c'est un ensemble fini.

$$\boxed{G_{|I} \text{ contient un couplage de poids minimal}}$$

Q 21. Chaque sommet de degré impair dans T est complété par une unique arête de M pour former H, donc tous les sommets de H sont de degré pair. D'où :

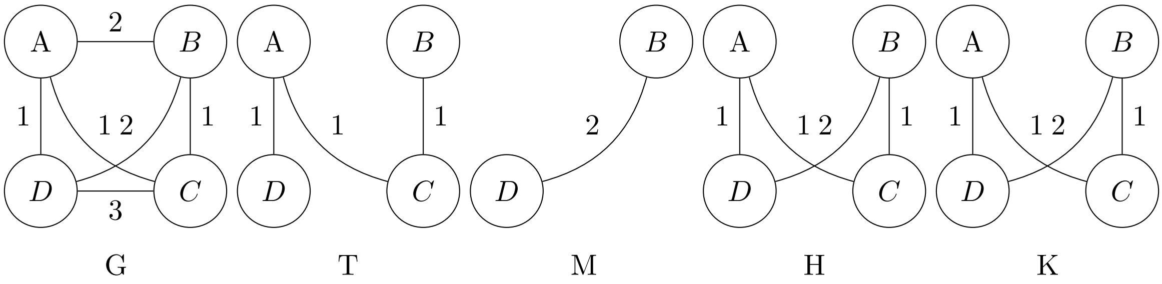
$$\boxed{\text{Il existe un circuit eulérien dans } H}$$

Q 22. Il suffit de parcourir le circuit en « sautant » les sommets déjà visités, sauf le premier et le dernier qui sont identiques par définition d'un circuit.

```c
struct Chemin euler_to_hamilton(struct Chemin c){
    int ns = c.longueur;
    struct Chemin ch = alloue_chemin(ns); // chemin hamiltonien
    int ich = 0;                          // indice pour ch
    struct Chemin vu = alloue_chemin(ns); // pour gérer les doublons
    for (int i = 0; i < ns; i++){
        vu.l_sommets[i] = 0;
    }
    for (int i = 0; i < ns; i++){
        int s = c.l_sommets[i];
        if (vu.l_sommets[s] == 0){
            ch.l_sommets[ich++] = s;
            vu.l_sommets[s] = 1;
        }
    }
```

```
    ch.l_sommets[ich++] = c.l_sommets[0]; // le dernier
    libere_chemin(vu);
    ch.longueur = ich;
    return ch;
}
```

Q 23.



Le circuit eulérien est ici sans doublon, le circuit K est à la fois eulérien et hamiltonien car la dernière étape est sans effet.

Q 24.

```
struct Chemin christofides(struct Graphe g){
    struct Graphe t = kruksal(g);
    int ns;
    int *i = sommets_impairs(t, &ns);
    struct Graphe gi = graphe_induit(g, ns, i);
    struct Graphe m = couplage(gi);
    struct Multigraphe h = multigraphe(m, t);
    struct Chemin ce = eulerien(h);
    struct Chemin ch = euler_to_hamilton(ce);
    libere_graphe(t);
    libere_graphe(gi);
    libere_graphe(m);
    libere_multigraphe(h);
    libere_chemin(ce);
    return (ch);
}
```

Q 25. D'après **Q 21**, `eulerien` renvoie un cycle eulérien. Reste à montre que `euler_to_hamilton` fonctionne correctement. Le chemin `ce` étant eulérien, il passe par toutes les arêtes de `h` , qui contient toutes les arêtes de `t`. Il contient donc toute les sommets de `t`, donc de `g`. En supprimant tous les doublons (sauf le dernier qui ferme le circuit), on obtient bien un chemin hamiltonien de `g`.

Q 26. Soit respectivement n_s et n_a le nombre de sommets et d'arêtes de G. Les complexités des différentes fonctions sont :

- `kruskal` : $O(n_a log(n_a))$. Le tri est en effet la tâche la plus complexe avec une structure union-find efficace.
- `sommets_impairs` : $O(n_s)$.
- `graphe_induit` : $O(n_s)$ car $G_{|I}$ est plus petit que G.
- `couplage` : polynomiale d'après l'énoncé (L'algorithme de Lawler est en $O(n_a^3)$).
- `multigraphe` : $O(n_s + n_a)$.
- `eulerien` : $O(n_a)$.
- `hamiltonien` : $O(n_a)$.

Comme ces fonctions sont appelées en séquence , le résultat final est aussi polynomial en la taille de G.

Q 27. T est de poids inférieur ou égal au poids de U, car U privé d'une arête est un arbre couvrant, et T est un arbre couvrant de poids minimal.

Q 28. Soit $C = i_1 i_2 \dots i_{2p} i_1$ le circuit qui relie les sommets de l'ensemble I des sommets de degré impair dans T, dans l'ordre où on les rencontre en parcourant U. d'après l'inégalité triangulaire, $c(C) \leqslant c(U)$. Soit maintenant les deux couplages complémentaires $M_1 = \{\{i_1, i_2\}, \dots, \{i_{2p-1}, i_{2p}\}\}$ et $M_2 = \{\{i_2, i_3\}, \dots, \{i_{2p}, i_1\}\}$.

Par construction, $c(M_1) + c(M_2) = c(C)$. Soit enfin $c_{\min} = \min(c(M_1), c(M_2))$. Il vient $c(M) \leqslant c_{\min} \leqslant \frac{1}{2}c(C) \leqslant \frac{1}{2}c(U)$, ce qui achève la preuve.

Q 29. Comme les arêtes de la solution K renvoyée par l'algorithme de Christofides sont choisies dans $T \cup M$, $c(K) \leqslant c(T) + c(M)$. Donc $c(K) \leqslant \frac{3}{2}c(U)$.

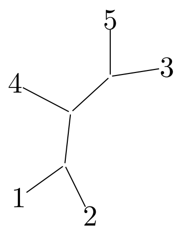
$\boxed{\text{l'algorithme de Christofides est une } \frac{3}{2} \text{ approximation du problème du voyageur de commerce}}$

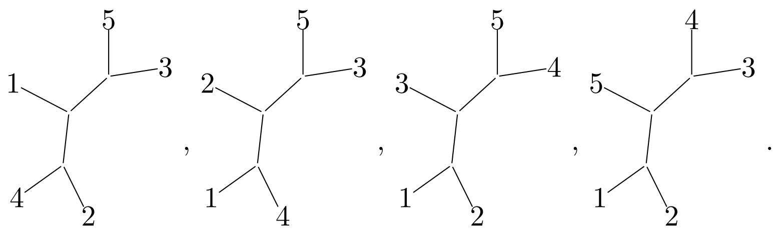
Q 30. Ce n'est pas le cas sans l'inégalité triangulaire . L'exemple de la **Figure 1** en donnant un poids 100 pour l'arête $\{B, D\}$ fournit un contre exemple : Le circuit ACBDA trouvé par l'algorithme est de poids 103, alors que $c(ABCDA) = 8$ est optimal, et $103 > \frac{3}{2} \times 8$.

2 Espaces d'arbres

Q 31. Passer de l'arbre 1 à l'arbre 2 peut se faire en échangeant les sous arbre contenant 2 et 4 par un mouvement NNI sur la branche entre e et f.

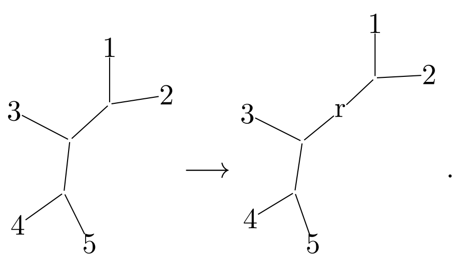
Il y a 2 NNI possibles autour des 2 branches internes, soit 4 voisins en tout (à déformation près). La feuille 4 de l'arbre 1 peut être permutée avec n'importe laquelle des 4 autres :

l' arbre 1 : 

a pour voisins 

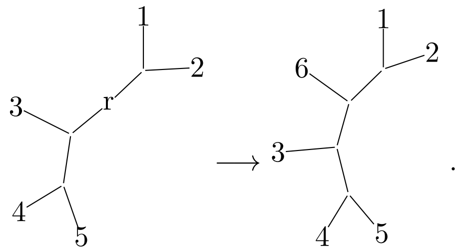
Q 32. Soit N et B le nombre de nœuds et de branches. Comme ce sont des arbres, $B = N - 1$. les feuilles étant de degré 1 et les nœuds internes de degré 3, le lemme de la poignée de main impose $n + 3(N - n) = 2(N - 1)$. D'où $\boxed{2n - 2 \text{ nœuds et } 2n - 3 \text{ branches}}$ par arbre.

Q 33. Pour créer un arbre raciné à n feuilles, il suffit de partir d'un arbre non raciné, de couper un branche en deux pour y insérer une racine(r) comme ci-dessous :



Comme il y a $2n - 3$ branches par arbre non raciné, on a bien $\boxed{|RB(n)| = (2n - 3)|B(n)|}$.

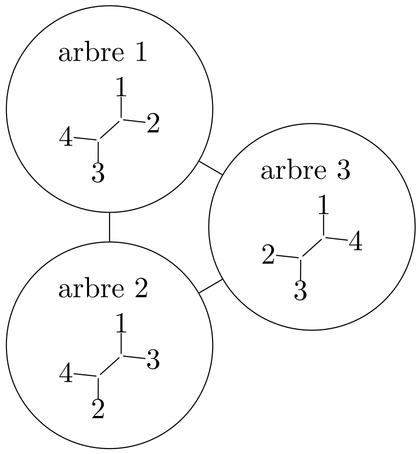
De même on passe d'un arbre raciné à un arbre non raciné en ajoutant une feuille à la racine :



D'où $\boxed{\forall n \geqslant 3, |RB(n)| = |B(n-1)|}$.

En regroupant ces deux équations, il vient $|B(n)| = (2n-5)|B(n-1)|$. Soit $|B(n)| = (2n-5) \times (2n-7) \times \ldots \times 1$ ou encore $\boxed{\forall n \geqslant 3, |B(n)| = \prod_{i=0}^{n-3}(2k+1)}$.

Q 34. $G_{\text{NNI}}(4)$ est un graphe à $3 \times 1 = 3$ sommets :



Q 35. Il y a $2n-3$ branches dans un arbre de $B(n)$, dont n sont reliées à des feuilles, soit $n-3$ branches internes. Pour chacune de celle-ci, il y a 2 mouvements NNI possibles. Ainsi, un arbre possède $\boxed{2(n-3) \text{ voisins dans } G_{\text{NNI}}(n)}$, si $n \geqslant 3$.

Q 36. Une chenille est un arbre pour lequel aucun nœud n'est relié à trois branches internes.

— Appelons **tête** un nœud interne dont deux branches portent des feuilles, **articulation** un nœud relié à trois branches internes. Appelons **patte** un sous-arbre connecté à cette articulation. Soit e une branche liée à une articulation . Un mouvement NNI par rapport à e rapproche nécessairement une patte d'une tête. Tant que l'articulation existe, on peut déplacer celle ci vers la tête. Lorsque la patte atteint la tête, elle disparaît. On peut ainsi faire disparaître toutes les articulations pour obtenir une chenille.

— Reste à remarquer que l'on peut passer d'une chenille à l'autre par NNI, car :
  — à chaque chenille est associée une permutation de $[\![1 \ldots n]\!]$ correspondant au chemin qui la définit ;
  — un mouvement NNI qui échange 2 feuilles sur une chenille correspond à une transposition ;
  — toute permutation se décompose en transpositions.

D'où : $\boxed{G_{\text{NNI}}(n) \text{ est connexe}}$

Q 37.

— Soit e et f les deux branches d'une transformation SPR. Tant que e et f ne sont pas adjacentes, il existe des branches internes les reliant, et donc un mouvement NNI permettant rapprocher e de f. La transformation SPR correspond à la suite des mouvements NNI qui font « glisser » e jusqu'à f.

— Un mouvement NNI est alors un mouvement SPR où e et f sont séparés par exactement une branche, comme en **Q 31**.

— Il en résulte que $G_{\mathrm{NNI}}(n)$ est un sous-graphe de $G_{\mathrm{SPR}}(n)$, ayant les même sommets. Comme $G_{\mathrm{NNI}}(n)$ est connexe, $\boxed{G_{\mathrm{SPR}}(n) \text{ est connexe}}$.

Q 38. Les sommets de $G_{\mathrm{NNI}}(5)$ sont les arbres de $B(5)$, il y a donc $5 \times 3 \times 1 = \boxed{15 \text{ sommets}}$ d'après **Q 32**. Et d'après **Q 35**, chaque arbre à $2(5-3) = \boxed{4 \text{ voisins}}$.

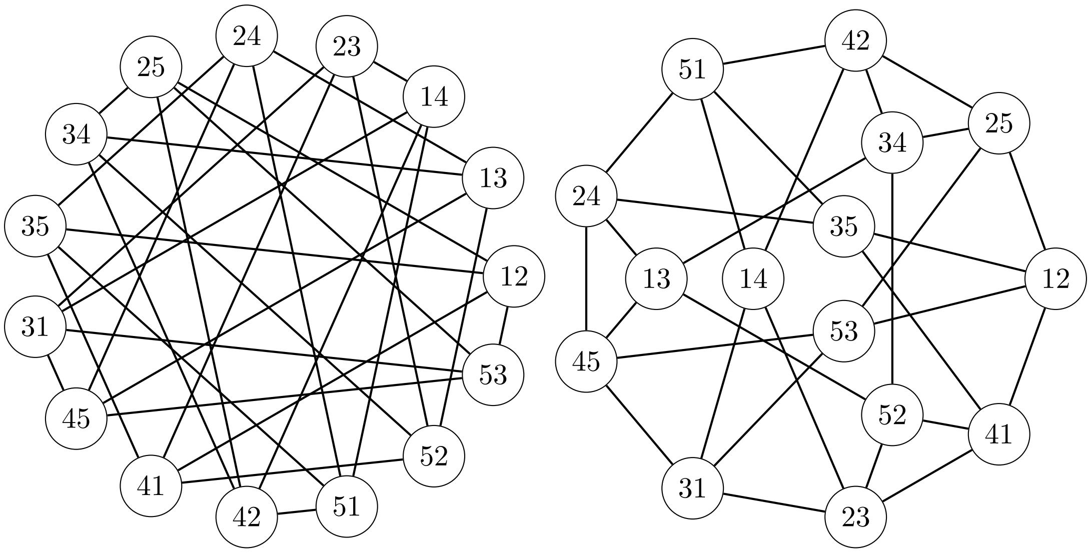
Q 39. Comme tout les arbres de $B(5)$ se déduisent les uns des autres par permutation de feuilles, il suffit de raisonner sur un arbre particulier, l'arbre 5 dans la suite. Pour simplifier l'exposé, soit a l'étiquette centrale, associée au seul nœud n'ayant qu'un feuille (le 3 dans l'arbre 5). Tout mouvement NNI échange a et une autre feuille b, noté $a \leftrightarrow b$.

— Les cycles de longueur 3 correspondent aux suites de mouvements NNI qui échangent a avec les deux feuilles initialement du même coté de a. ($3 \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow 3$ et $3 \leftrightarrow 4 \leftrightarrow 5 \leftrightarrow 3$ pour l'arbre 5). Il y a deux cycles de longueur 3 pour chaque arbre, les deux sens de parcours représentant le même cycle.

— Les cycles de longueur 5 correspondent aux suites de mouvements NNI qui donnent à a les 5 valeurs possibles. Il faut pour cela changer de coté à chaque fois . ($3 \leftrightarrow 1 \leftrightarrow 4 \leftrightarrow 2 \leftrightarrow 5 \leftrightarrow 3$, $3 \leftrightarrow 1 \leftrightarrow 5 \leftrightarrow 2 \leftrightarrow 4 \leftrightarrow 3$, $3 \leftrightarrow 2 \leftrightarrow 4 \leftrightarrow 1 \leftrightarrow 5 \leftrightarrow 3$, et $3 \leftrightarrow 2 \leftrightarrow 5 \leftrightarrow 1 \leftrightarrow 4 \leftrightarrow 3$ pour l'arbre 5). Il y a quatre cycles de longueur 5 pour chaque arbre.

— Pour 2 échanges successifs, il y a 4 choix pour le premier, puis 3 pour le second pour ne pas retomber sur l'arbre initial, soit 12 arbres accessibles en 2 mouvements. Comme il y a 15 arbres dans $G_{\mathtt{NNI}}(5)$, deux $(15-12-1)$ arbres sont inaccessibles en 2 mouvements, mais le sont en 3 mouvements.

Q 40. Pour simplifier le graphe, un arbre est dénoté par un nombre à deux chiffres ac : a représente l'étiquette centrale, c le voisin de $(a-2) \mod 5 + 1$ (les arbres 1,2 et 5 sont respectivement notés 45,24 et 31). Il suffit de tracer soigneusement les 6 arêtes reliant 12 à 25 et 35, 13 à 24 et 34, 14 à 23 et 31 : les 24 autres se tracent par rotation de $\dfrac{2k\pi}{5}$ sur la figure de gauche. La figure de droite représente le même graphe, les cycles y étant plus facile à observer.



On observe bien (à droite) que chaque nœud est inclus dans quatre pentagones, deux triangles, et seuls deux nœuds sont inaccessibles en deux « pas » : (42⇝41 et 42⇝45 par exemple).

Q 41. et Q 42, Q 43, Q 44 : `egaux` est correcte par unicité de la représentation des arbres.

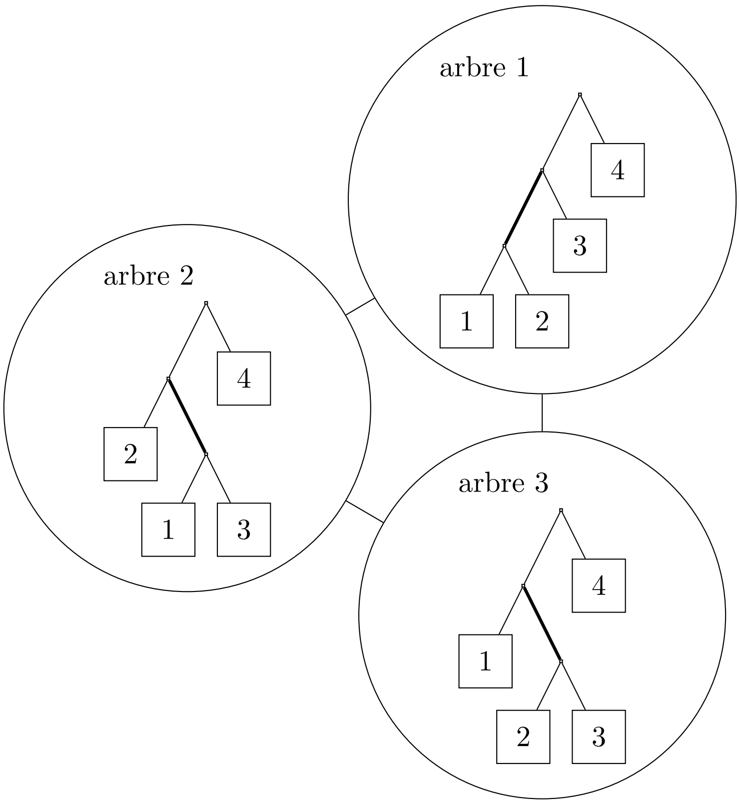
```
let rec feuilles = function
  Feuille x ->[x]
  |Noeud (a, b) -> feuilles a @ feuilles b

let rec degres (g:graphe) = match g with
  [] ->[]
  |(_,l)::q-> List.length l :: degres q

let  rec egaux a b = match (a, b) with
  Feuille x, Feuille y  -> x = y
  |Noeud (ag, ad), Noeud(bg, bd) -> egaux ag bg && egaux ad bd
  |_ -> false

let rec appartient a = function
  [] -> false
  |b::q -> egaux a b || appartient a q
```

Q 45. Le graphe de la **Q 34** est repris ci-dessous en version arborescente normalisée. Chaque étiquette représente ici un sous-arbre et la branche pour le mouvement NNI est en gras.



La partie de l'arbre contenant le sous-arbre $\boxed{4}$ est toujours considérée fixe.

Il faut distinguer les transformations NNI, selon que l'échange se fasse selon une branche père-fils gauche (NNIG) ou père-fils droit (NNID).

Pour passer de arbre 2 ou arbre 3 vers arbre 1 sur la figure précédente (NNID), il faut savoir ordonner $\boxed{1}$ et $\boxed{2}$ dans arbre 1 pour garantir que la plus grande étiquette soit à droite. Les transformations inverses (NNIG) sont sans ambigüité.

La fonction auxiliaire `nmax` détermine la plus grande étiquette d'un arbre (au fond à droite). Puis `nnig`, `nnid`, `nni` réalisent respectivement les transformations associées, et les deux quand c'est possible. Enfin `voisinsNNI` explore récursivement toutes les branches internes de l'arbre :

```
let rec nmax = function Feuille x -> x | Noeud(_, a)-> nmax a
let orderNoeud (a, b) = if nmax a < nmax b then Noeud(a, b) else Noeud(b, a)
let  nnig (a, b) c = [Noeud(b, Noeud(a, c)); Noeud(a, Noeud (b, c))]
let  nnid a (b, c) = [Noeud(b, Noeud(a, c)); Noeud(orderNoeud(a, b), c)]

let nni abr =
    (match abr with Noeud (Noeud(a, b),c)-> nnig (a, b) c |_ -> [])
    @ (match abr with Noeud (a, Noeud(b, c))-> nnid a (b, c) |_ -> [])

let rec voisinsNNI = function
  Noeud (g, d) ->
    List.map (fun g -> Noeud (g, d)) (nni g @ voisinsNNI g)
    @ List.map (fun d -> Noeud (g, d)) (nni d @ voisinsNNI d)
  |_ -> []
```

Q 46.

```
let rec chenille n =
  if n = 1 then Feuille 1 else
    Noeud (chenille (n-1), Feuille n)
```

Q 47. et Q 48.

```
let rec insere arbre (graphe:graphe) =
  match graphe with
    |[] -> [arbre,[]]
    |(s,l)::q when egaux s arbre -> graphe
    |(s,l)::q  -> (s,l) :: insere arbre q

let rec relie (graphe:graphe) arbre1 arbre2 =
  match graphe with
    |[] -> graphe
    |(s,l)::q when egaux s arbre1 && not (appartient arbre2  l)
                -> (arbre1, arbre2::l) :: relie q arbre1 arbre2
    |(s,l)::q when egaux s arbre2 && not (appartient arbre1  l)
                -> (arbre2, arbre1::l) :: relie q arbre1 arbre2
    |sl::q  -> sl :: relie q arbre1 arbre2
```

Q 49. Les fonctions précédentes sont inutiles, car `voisinsNNI a` est la liste d'adjacence de `a` :

```
let rec construction fait afaire = match afaire with
  |[] -> []
  |arbre :: q when not (appartient arbre fait)->
      let vois = voisinsNNI arbre in
          (arbre, vois) :: construction (arbre :: fait) (q @ vois)
  |_ :: q  -> construction fait q

let grapheNNI n = construction [] [chenille n]
```

`construction` termine car :

- le second cas de filtrage (`when not (appartient arbre fait)`) est appelé au plus $|B(n)|$ fois ;
- le troisième cas de filtrage fait diminuer strictement $|$`afaire`$|$.

Et `construction` est correct car $G_{\texttt{NNI}}(n)$ est connexe, donc tous les voisins sont visités et la fonction renvoie le graphe complet.

Q 50. Supprimer la feuille $n-1$ avec sa branche adjacente d'un arbre quelconque de S_i, puis la rajouter sur une branche quelconque de cet arbre (n choix possibles), c'est exactement la définition d'un mouvement SPR entre deux arbres quelconques de S_i. Deux arbres quelconques sont donc reliées. Donc $\boxed{G_{\texttt{SPR}}(n+1)_{|S_i} \text{ est complet}}$.

Q 51. $n \geqslant 4$ car il n'y a pas d'arêtes dans $G_{\texttt{SPR}}(3)$. Soit t_j l'arbre obtenu par SPR en utilisant deux branches nommées e et f, à partir de t_i. Soit t'_i et t'_j les arbres obtenus à partir de t_i et t_j par addition de l'étiquette (n+1) sur une branche g . Alors l'opération SPR sur les branches e et f appliquée à t'_i donne t'_j. Il y a donc au moins $2n-3$ arêtes (au moins 5) entre S_i et S_j. $\boxed{S_i \text{ est relié à } S_j \text{ dans } G_{\texttt{SPR}}(n+1) \text{ par des arêtes}}$.

Q 52.

— **Initialisation :** $G_{\texttt{SPR}}(4) = G_{\texttt{NNI}}(4)$ est clairement hamiltonien (voir **Q 34**).

— **Hérédité :** Soit $n \geqslant 5$ et $G_{\texttt{SPR}}(n-1)$ hamiltonien. Par construction, $B(n) = \bigcup_{i=1}^{n} S_i$, et les S_i sont disjoints. On commence par connecter les S_i entre eux à l'aide des arêtes exhibées à la question précédente, en formant un cycle : c'est possible car $G_{\texttt{SPR}}(n-1)$ hamiltonien. puis les $G_{\texttt{SPR}}(n)_{|S_i}$ étant complets, il est simple de fermer le cycle dans chaque S_i.

$$\text{donc } \boxed{G_{\texttt{SPR}}(n) \text{ est hamiltonien}}$$