DS n° 06 — corrigé

I Problème du voyageur de commerce

Q 1) Le circuit $ACBDA$ de poids 5 est une solution optimale.

Q 2) Pour $n \leq 2$ il n'y a aucun circuit hamiltonien. Pour $n \geq 3$, sur un graphe complet les circuits hamiltoniens correspondent exactement aux permutations des sommets. Il y en a $n!$.[1] Si la fonction de pondération est constante, par exemple constante égale à 1, tous les circuits hamiltoniens ont même poids et ce sont donc tous des solutions au problème du voyageur de commerce.

Q 3) La difficulté est ici de bien comprendre les structures de données proposées.

C

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

La complexité de cette fonction est linéaire en la taille du chemin.

Q 4) Le problème de décision associé est :

INSTANCE : un graphe G complet pondéré dans $\mathbb{N}^*$ et un seuil $s \in \mathbb{N}$;

QUESTION : existe-t-il dans G un circuit hamiltonien de poids au plus s.

Un certificat pour une instance (G, s) de ce problème de décision est un circuit hamiltonien de poids au plus s, qui est bien de taille polynomiale en la taille de l'instance. On peut vérifier qu'un certificat correspond bien à un chemin hamiltonien d'un graphe, en temps polynomial, calculer son poids avec la fonction polynomiale précédente et vérifier, en temps polynomial, que ce poids est inférieur à s.

Q 5) À une instance $(G = (S, A), a, b)$ du problème du chemin hamiltonien avec[2] $a \neq b$ on associe une instance $G' = (S', A')$ du problème du circuit hamiltonien en ajoutant un nouveau sommet $s \notin S$ relié uniquement aux sommets a et b, c'est-à-dire que l'on pose $S' = S \sqcup \{s\}$ et $A' = A \sqcup \{\{s, a\}, \{s, b\}\}$. Cette transformation se fait bien en temps polynomial. Si G est une instance positive pour le problème du chemin hamiltonien alors il existe un chemin hamiltonien $a - \cdots - b$ de G et alors $s - a - \ldots b - s$ est un circuit hamiltonien pour G'. Inversement, si G' admet un circuit hamiltonien, comme celui-ci passe par s qui n'est relié qu'à a et b dans G' ce circuit peut s'écrire $s - a - \cdots - b - s$ avec $a - \cdots - b$ un chemin de G' passant exactement une fois par tous les sommets de $S' \setminus \{s\}$ qui est donc un chemin hamiltonien de G entre a et b.

Q 6) À une instance $G = (S, A)$ du problème du circuit hamiltonien on associe l'instance $(G' = (S, \mathcal{P}_2(S), w), |S|)$ du problème du voyageur de commerce en construisant un graphe complet à partir des sommets de G dont les arêtes de G ont un poids 1 et les autres un poids de 2, c'est-à-dire que l'on pose comme fonction de pondération :

$$w : \begin{array}{ccc} \mathcal{P}_2(S) & \longrightarrow & \mathbb{N}^* \\ a & \longmapsto & \begin{cases} 1 & \text{si } a \in A \\ 2 & \text{sinon} \end{cases} \end{array}$$

Cette transformation est bien polynomiale en la taille de l'instance. S'il existe un circuit hamiltonien dans G alors ce même circuit est un circuit hamiltonien de G' de poids $|S|$ qui est bien inférieur ou

1. Ou $\frac{(n-1)!}{2}$ si on s'intéresse aux circuits abstraits et non à leurs représentations.
2. Si $a = b$ la réduction est immédiate avec la fonction identité.

égal au seuil $|S|$. Inversement, s'il existe dans G' un circuit hamiltonien de de poids inférieur ou égale à $|S|$, comme il comporte $|S|$ arêtes de poids au moins 1, toutes les arêtes sont de poids exactement 1 et ce circuit hamiltonien est un circuit de G.

Q 7) L'idée est d'associer à chaque sommet $s \in S$ du graphe orienté trois sommets s^-, s^0, s^+ ; les arcs entrants sur s se branchant sur s^- et les arcs sortant de s se branchant sur s^+. Formellement, à un graphe $G = (S, A)$ orienté on associe le graphe $G' = (S', A')$ non orienté avec

$$G' = \bigcup_{s \in S} \{s^-, s^0, s^+\} \qquad A' = \bigcup_{(s,t) \in S} \{\{s^+, t^-\}\}$$

À une instance (G, a, b) du problème du chemin hamiltonien orienté on associe donc l'instance (G', a^-, b^+) du problème du chemin hamiltonien non orienté. Cette transformation est bien polynomiale. Si $a - x \cdots - b$ est un chemin hamiltonien de G alors $a^- - a^0 - a^+ - x^- - x^0 - x^+ - \ldots b^- - b^0 - b^+$ est un chemin hamiltonien de G'. Inversement, si $a^- - \cdots - b^+$ est un chemin hamiltonien dans G', alors le deuxième sommet doit être a^0, sinon celui-ci devrait être visité juste après a^+ et on ne pourrait pas ensuite continuer le chemin. Le troisième sommet est alors nécessairement a^+. Celui-ci est donc ensuite relié à un certain x^- qui est relié à x^0 puis à x^+, pour les mêmes raisons, et ainsi de suite. Le chemin $a - x - \cdots - b$ associé dans G est bien un un chemin orienté de G et il visite chaque sommet exactement une fois.

Q 8) On propose respectivement les chemins :

- $e_1 - e_2 - e_3 - s_3 - s_2 - s_1$
- $e_1 - s_1$ et $e_2 - e_3 - s_3 - s_2$
- $e_1 - s_1$, $e_2 - s_2$ et $e_3 - s_3$

Q 9) On suppose qu'il existe un modèle ν de la formule et on va constuire un chemin de v_1 à v_{m+1} qui passe au moins une fois par chaque A_i. Pour $k \in [\![1, m]\!]$ on construit un chemin entre v_k et v_{k+1} dans G_k^+ si $\nu(y_k) = V$ et dans G_k^- si $\nu(y_k) = F$ en entrant dans chaque A_{k_i} par e_p et en sortant directement par s_p. En concaténant ces chemins d'intérieur disjoint on obtient un chemin élémentaire de v_1 à v_{m+1}. Comme ν est un modèle de la formule, chaque clause est satisfaite par ν et donc visitée au moins une fois par ce chemin. Certains sommets des A_k peuvent cependant ne pas avoir été visités (ce sont ceux correspondant aux littéraux non satisfaits par ν), mais d'après la question précédente on peut toujours construire un détour qu'il y ait un ou deux chemins passant par ce gadget pour en visiter tous les sommets, sans changer les points d'entrée et de sortie. En prolongeant ainsi ce chemin, on obtient un chemin hamiltonien de v_1 à v_{m+1}.

Q 10) Considérons un chemin hamiltonien de v_1 à v_{m+1}. Ce chemin passe exactement une fois par chaque v_k, pour $k \in [\![1, m]\!]$. Le successeur du sommet v_k est soit dans G_k^+, auquel cas on pose $\nu(y_k) = V$ soit dans G_k^- auquel cas on pose $\nu(y_k) = F$, soit c'est directement v_{k+1} auquel cas on pose arbitrairement $\nu(y_k) = V$ par exemple. Montrons que ν est un modèle de la formule. Pour une clause C_i pour $i \in [\![1, n]\!]$, comme le chemin est hamiltonien, A_i est parcouru par le chemin et appartient donc au moins à un certain G_k parcouru à partir d'un sommet v_k. Si A_i est dans G_k^+, alors la clause C_i contient le littéral y_k et comme $\nu(y_k) = V$ la clause est satisfaite par ν. De même, si A_i st dans G_k^- alors la clause C_i contient le littéral $\neg y_k$ et comme $\nu(y_k) = F$ la clause est également satisfait par ν. Finalement, ν satisfait toutes les clauses de la formule et en est donc un modèle.

Q 11) On vient de montrer que 3-SAT se réduit au problème du chemin hamiltonien orienté, la réduction précédente étant bien polynomiale. Comme 3-SAT est NP-difficile on en déduit que le problème du chemin hamiltonien orienté est NP-difficile. Par transitivité des réductions polynomiales et par ce qui précède, les problèmes du circuit hamiltonien et du voyageur de commerce sont donc NP-difficiles. Le problème du voyageur de commerce étant également NP, celui-ci est NP-complet. Le problème du circuit hamiltonien se réduisant en temps polynomial au problème du voyageur de commerce qui est NP, celui-ci est également NP et finalement NP-complet.

Q 12) Un circuit hamiltonien de G est un circuit hamiltonien de G' comportant n arêtes de poids 1 donc de poids n.

Q 13) Si G ne possède pas de circuit hamiltonien alors tout circuit hamiltonien de G' doit prendre une arête qui n'est pas dans G donc de poids $n(1+\varepsilon)+1$. Le poids de ce circuit est donc au moins de $(n-1)\times 1 + n(1+\varepsilon)+1 = n(2+\varepsilon)$.

Q 14) Supposons qu'il existe une $1+\varepsilon$ approximation A du problème du voyageur de commerce et construisons un algorithme polynomial pour résoudre le problème du circuit hamiltonien, ce qui impliquerait $P = NP$ puisque ce problème est NP-complet. Pour une instance $G = (S, A)$ du problème du circuit hamiltonien on commence par construire, en temps polynomial, le graphe G' ci-dessus et on exécute l'algorithme d'approximation A. Si celui-ci trouve une solution de poids inférieur à $n(1+\varepsilon)$ on affirme qu'il existe un circuit hamiltonien et dans le cas contraire on affirme qu'il n'en existe pas. Cet algorithme résout bien le problème du circuit hamiltonien, en temps polynomial, d'après les deux questions précédentes. En effet, s'il existe un circuit hamiltonien dans G alors la solution optimale est de poids $p^* = n$ et l'algorithme d'approximation donne une solution de poids inférieur à $(1+\varepsilon)p^* = n(1+\varepsilon)$. Inversement, s'il n'existe pas de circuit hamiltonien dans G, tout solution est de poids supérieur à $n(2+\varepsilon)$ et l'algorithme d'approximation ne peut trouver de solution de poids inférieur.

Q 15) Pas de réelles difficultés mais il faut faire attention tout de même.

C

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

Q 16) Il n'est pas précisé si on peut utiliser une structure de données unir et trouver ou s'il est nécessaire de la réimplémenter. On implémente rapidement une version sans heuristiques :

C

```c
int* creer(int n) {
  int* p = malloc(n * sizeof(int));
  for (int i = 0; i < n; i++) p[i] = i;
  return p;
}

int trouver(int* p, int x) {
  while (x != p[x]) x = p[x];
  return x;
}

void unir(int* p, int x, int y) {
  p[trouver(p, x)] = trouver(p, y);
}
```

On peut enfin écrire l'algorithme de Kruskal. On pourrait s'arrêter après $|S|-1$ ajouts, ce qui permettrait d'améliorer la complexité dans le meilleur cas.

C

```c
struct Graphe kruskal(struct Graphe g) {
  int nb_aretes = g.V * (g.V - 1) / 2;
  struct Arete* aretes = liste_aretes(g);
  struct Graphe t = alloue_graphe(g.V);
  for (int k = 0; k < g.V * g.V; k++) {t.adj[k] = 0;}
  int* p = creer(g.V);
  tri_aretes(aretes, nb_aretes);
  for (int i = 0; i < nb_aretes; i++) {
    struct Arete a = aretes[i];
    if (trouver(p, a.s1) != trouver(p, a.s2)) {
      t.adj[a.s1 * g.V + a.s2] = 1;
      t.adj[a.s2 * g.V + a.s1] = 1;
      unir(p, a.s2, a.s2);
    }
  }
  free(p);
  free(aretes);
  return t;
}
```

La complexité de cet algorithme est en $\mathcal{O}(|S|^2)$ pour obtenir la liste des arêtes et pour l'initialisation, $\mathcal{O}(|A|\log|A|) = \mathcal{O}(|S|^2\log|S|)$ pour le tri des arêtes, puis $\mathcal{O}(|A|)$ opérations unir et trouver qui, avec l'implémentation naïve ici, sont en $\mathcal{O}(|S|)$, donc un $\mathcal{O}(|S|^3)$, le reste est en temps constant. Avec une l'heuristique de l'union par rang, les opérations unir et trouver se font en $\mathcal{O}(\log|S|)$ et on obtient au total une complexité en $\mathcal{O}(|A|\log|A|) = \mathcal{O}(|S|^2\log|S|)$, puisque le graphe est complet, dominée par le coût du tri des arêtes.

Q 17) La terminaison ne pose pas de problème. Pour la correction, on montre que « le sous-graphe T en cours de construction est un sous-graphe acyclique d'un arbre couvrant de poids minimal de G » est un invariant. L'initialisation est immédiate. Pour la conservation on n'ajoute une arête à T que si celle-ci ne crée pas de cycle et le caractère acyclique est conservé. Si T est un sous-arbre d'un arbre couvrant T^* de poids minimal et que l'arête a que l'on ajoute est également dans T^* alors l'invariant est conservé. Sinon, ajouter a à T^* crée un cycle. Comme on s'apprêtait à ajouter a à T, ses extrémités ne sont pas reliées dans T. Donc il existe sur ce cycle se trouve une arête a' de T^* qui n'a pas encore été considérée (sinon elle aurait été ajoutée), et donc $p(a) \leq p(a')$. On a donc T qui est un sous-graphe de $T^* - a' + a$ qui est un arbre couvrant dont le poids est inférieur ou égal au poids de T^* qui est de poids minimal, donc $T^* - a' + a$ est un arbre couvrant de poids minimal et l'invariant est conservé.

Q 18) Pas de difficulté.

C

```c
int degre(struct Graphe g, int i) {
  int d = 0;
  for (int j = 0; j < g.V; i++) {
    if (g.adj[i * g.V + j] != 0) d++;
  }
  return d;
}
```

Q 19) On alloue ici un tableau qui sera éventuellement trop grand, mais ce n'est pas un problème. On pourrait faire une première passe pour calculer la bonne taille avant de l'allouer.

C

```c
int* sommets_impairs(struct Graphe g, int* nb_sommets) {
  *nb_sommets = 0;
  int* sommets = malloc(g.V * sizeof(int));
  for (int i = 0; i < g.V; i++) {
    if (degre(g, i) % 2 == 1) {
      sommets[*nb_sommets] = i;
      (*nb_sommets)++;
    }
  }
  return sommets;
}
```

Q 20) Le nombre de sommets impairs d'un graphe est pair par le lemme des poignées de mains. Le sous-graphe $G_{|I}$ est un graphe complet avec un nombre pair de sommet donc tout couplage maximal est parfait. L'ensemble des couplages parfait est fini non vide donc admet un plus petit élément. Donc $G_{|I}$ admet au moins un couplage parfait de poids minimal.

Q 21) Chaque sommet de degré impair de T est couplé avec exactement un autre sommet de degré impair de T, donc les sommets de degré impair de T sont de degré pair dans H. Le degré des autres sommets est inchangé. Tous les sommets de H sont donc de degré pair. De plus T est un arbre couvrant de H qui est donc connexe. La propriété admise dans l'énoncé permet de conclure.

Q 22) On pourrait utiliser tableau de booléen[3] ou une table de hachage pour tester efficacement si un sommet a déjà été vu, mais on va au plus rapide avec une approche quadratique. L'important dans ce contexte est de proposer une solution polynomiale. On commence par écrire une fonction auxilaire permettant de savoir si un sommet est déjà dans un chemin.

C

```c
bool vu(struct Chemin c, int s) {
  for (int k = 0; k < c.longueur; k++) {
    if (c.l_sommets[k] == s) {return true;}
  }
  return false;
}
```

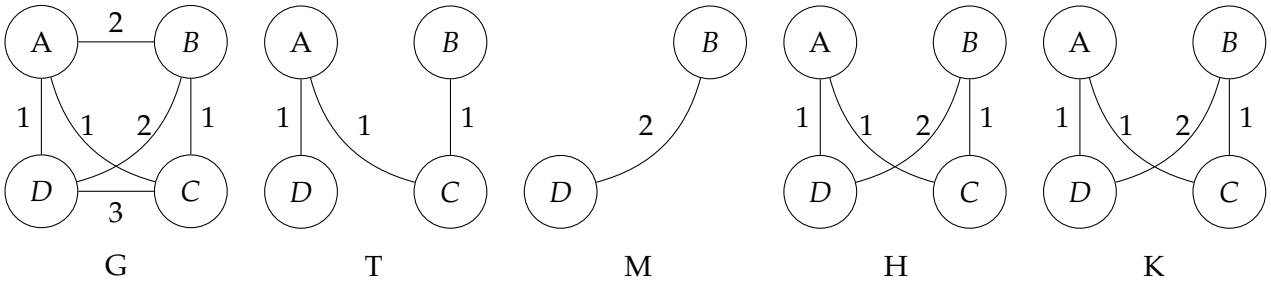
Encore une fois, on va allouer un tableau éventuellement trop grand pour le nouveau chemin.

C

```c
struct Chemin euler_to_hamilton(struct Chemin c) {
  struct Chemin h = alloue_chemin(c.longueur);
  h.longueur = 0;
  for (int i = 0; i < c.longueur; i++) {
    if (!vu(h, c.l_sommets[i])) {
      h.l_sommets[h.longueur] = c.l_sommets[i];
      h.longueur++;
    }
  }
  // Il faut quand même reboucler sur le premier
  h.l_sommets[h.longueur] = c.l_sommets[0];
  h.longueur++;
  return h;
}
```

3. C'est ce qui m'aurait semblé le plus simple si on avait eu en argument le graphe, ce qui aurait permi de connaître le nombre de sommets. Ici il faudrait faire une première passe sur le chemin pour avoir cette information ou allouer un tableau de la taille du chemin eulérien par exemple.

Q 23) On obtient les étapes suivantes :



Q 24) Il suffit de suivre l'algorithme :

C

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

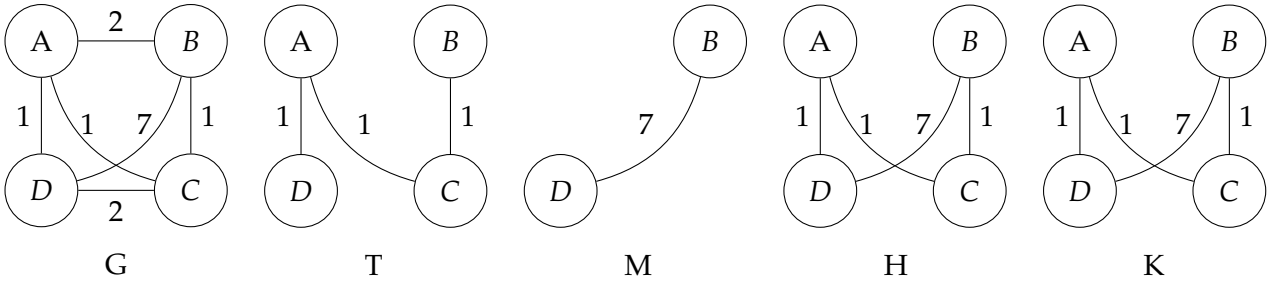
Q 25) H est un multigraphe connexe (car T est connexe) dont les sommets sont ceux de G. Un circuit eulérien de H passe donc au moins une fois par tous les sommets de g. Le circuit sans doublons qui en est extrait est donc un circuit hamiltonien de G. Rappelons que G est complet et qu'il est donc toujours possible de court-circuiter des sommets par des arêtes de G.

Q 26) Toutes les étapes de l'algorithme de Christophides se font en temps polynomiale, soit cela est admis par l'énoncé, soit il s'agit de fonctions que nous avons programmées, et cela de manière polynomiale. La complexité de cette algorithme d'approximation est donc polynomiale.

Q 27) Soit U une solution optimale au problème du voyageur de commerce, T l'arbre couvrant de U obtenu et notons U_I le circuit hamiltonien de $G_{|I}$ induit par U sur le sous-graphe des sommets I de degré impair dans T, c'est-à-dire le chemin formé par les sommets de I dans l'ordre d'apparition dans U. Il s'agit bien d'un chemin hamiltonien de $G_{|I}$ puisque le graphe est complet et par l'inégalité triangulaire, les raccourcis empruntés en sautant les sommets de degré pair de T ne peuvent que faire décroître le coût. Ainsi $c(U_I) \leq c(U)$. Considérons la partition (A_1, A_2) des arêtes de U_I (qui est de longueur paire) en en prenant une arête sur deux pour l'un et une arête sur deux pour l'autre. Chacune de ces deux partitions induit un couplage parfait C_1 et C_2 du graphe $G_{|I}$. On a donc $c(M) \leq c(A_1)$ et $c(M) \leq c(A_2)$. Ainsi, on a $c(M) \leq \min(c(A_1), c(A_2)) \leq 0.5c(U_I) \leq 0.5c(U)$.

Q 28) Notons S la solution construite par l'algorithme et R le circuit eulérien construit par l'algorithme. Ce circuit emprunte toutes les arêtes de H et son poids est donc $c(R) = c(H) = C(T) + C(M)$, puisque les arêtes de H sont exactement celles de T et de M. Puisque le graphe vérifie l'inégalité triangulaire, les raccourcis utilisés pour construire S à partir de R ne peuvent que faire décroitre le poids et on a donc $c(S) \leq c(R) = c(T) + c(M) \leq c(T) + \frac{1}{2}c(U) < c(U) + \frac{1}{2}c(U) = \frac{3}{2}c(U)$. On en déduit que cet algorithme est une $\frac{3}{2}$ approximation du problème du voyageur de commerce. *Avec la définition douteuse du sujet qui semble demande une inégalité stricte, il faut montrer l'inégalité stricte à la question 27.*

Q 29) Considérons le graphe G suivant avec les différentes étapes de l'algorithmes indiquées :



Une solution optimale pour le graphe G est de poids 6 mais la solution trouvée par l'algorithme de Christofides est $10 > \frac{3}{2}6$ et ce n'est donc plus une $\frac{3}{2}$ approximation. En choisissant un poids assez grand plutôt que 7 on peut montrer qu'il ne s'agit pas d'une α approximation, et ce quel que soit $\alpha > 1$. Nous avons d'ailleurs montré que sous l'hypothèse $P \neq NP$ il n'existait pas de telle approximation pour le cas général.