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 $[\![1 \dots n]\!]$ 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 $C' = zav_k \dots v_\ell bz$ est hamiltonien si et seulement si le chemin $C = av_k \dots v_\ell b$ 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 $\textsc{CiH}(I) = \text{DVC}(I')$:

— $\textsc{CiH}(I) = 1 \implies \text{DVC}(I') = 1$: 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.

— $\text{DVC}(I) = 1 \implies \textsc{CiH}(I') = 1$: 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.

.

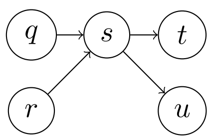
Pour la suite, il faut une réduction polynomiale. La construction précédente se fait clairement en un temps quadratique en $|I|$, d'où :

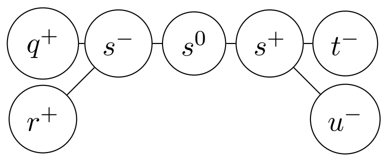
$$\boxed{\textsc{CiH} \leqslant_{\mathbf{P}} \text{DVC}}$$

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' :



devient


Il reste à montrer que $\textsc{ChHO}(I) = \textsc{ChH}(I')$:

— 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 :

$$\boxed{\textsc{ChHO} \leqslant_{\mathbf{P}} \textsc{ChH}}$$

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++){
```

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

Q 19.

```
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 :

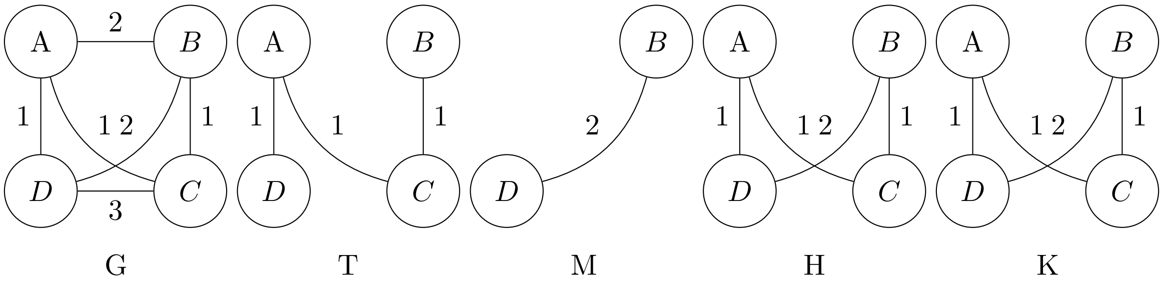
$$\boxed{\text{Il existe un circuit eulérien dans } H}$$

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.

```
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` : $O(n_a)$.