A 2008 INFO. MP

ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DE TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE (FILIÈRE TSI)

CONCOURS D'ADMISSION 2008

ÉPREUVE D'INFORMATIQUE

Filière MP
Durée de l'épreuve : 3 heures.
L'usage de calculettes est autorisé. L'utilisation d'un ordinateur est interdite.

Sujet mis à disposition des concours : ENSAE ParisTech, TELECOM SudParis (ex INT), TPE-EIVP
Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :

INFORMATIQUE - MP

L'énoncé de cette épreuve comporte 10 pages.

RECOMMANDATIONS AUX CANDIDATS

- **Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.**
- **Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.**
- **Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.**

COMPOSITION DE L'ÉPREUVE

L'épreuve comporte deux problèmes indépendants :

- un problème sur les automates, pages 2 et 3 ;
- un problème d'algorithmique et programmation, pages 3 à 10.

1. Problème sur les automates (temps conseillé : 40 mn)

Deux constructions d'un automate reconnaissant l'intersection de deux langages reconnaissables

Un **alphabet** Σ est un ensemble fini d'éléments appelés **lettres**. Un **mot** sur Σ est une suite finie de lettres de Σ. La **longueur** d'un mot m, notée $|m|$, est le nombre de lettres qui le composent ; par exemple, la longueur du mot « abac » vaut 4. Le **mot vide** est le mot de longueur nulle et est noté ε. On désigne par Σ^* l'ensemble des mots sur Σ, y compris le mot vide. Un **langage** sur Σ est une partie de Σ^*.

Un **automate** A est décrit par une structure $<\Sigma, Q, T, I, F>$, où :

- Σ est un alphabet ;
- Q est un ensemble fini et non vide appelé **ensemble des états** de A ;
- $T \subseteq Q \times \Sigma \times Q$ est appelé l'**ensemble des transitions** ; étant donnée une transition $(p, x, q) \in T$, on dit qu'elle va de l'état p à l'état q et qu'elle est d'étiquette x ; on pourra la noter $p \xrightarrow{x} q$;
- $I \subseteq Q$ est appelé **ensemble des états initiaux** de A ;
- $F \subseteq Q$ est appelé **ensemble des états finals** de A.

Un **calcul** c de A est une suite de la forme $p_0 \xrightarrow{x_1} p_1 \xrightarrow{x_2} p_2 \ldots \xrightarrow{x_k} p_k$ où, pour $1 \leq i \leq k$, $p_{i-1} \xrightarrow{x_i} p_i$ est une transition ; p_0 est l'**origine** du calcul, p_k son **extrémité**. L'**étiquette** de c est le mot $x_1x_2 \ldots x_k$ formé par la suite des étiquettes des transitions successives.

Un calcul d'origine p, d'extrémité q et d'étiquette m peut être noté $p \xrightarrow{m} q$. Un calcul $p \xrightarrow{m} q$ de A est dit **réussi** si on a $p \in I$ et $q \in F$. Un mot m de Σ^* est **reconnu** par A s'il est l'étiquette d'un calcul réussi. Le **langage reconnu** par A, noté $L(A)$, est l'ensemble des mots reconnus par A. Un langage est dit **reconnaissable** s'il existe un automate qui le reconnaît.

Un état q d'un automate A est dit **accessible** s'il existe dans A au moins un calcul dont l'origine est un état initial et dont l'extrémité est q.

L'automate A est dit **déterministe** si I contient exactement un élément et si, pour tout $p \in Q$ et tout $x \in \Sigma$, il existe au plus un état $q \in Q$ avec $(p, x, q) \in T$. L'automate A est dit **complet** si, pour tout $p \in Q$ et tout $x \in \Sigma$, il existe au moins un état $q \in Q$ avec $(p, x, q) \in T$.

Pour les exemples de ce problème, on utilisera l'alphabet $\Sigma\_ex = \{a, b\}$.

❒ 1 – Soit L_1_ex le langage sur Σ_ex des mots qui commencent par a. Représenter graphiquement un automate déterministe et complet noté A_1_ex qui reconnaît L_1_ex, c'est-à-dire vérifiant $L(A_1\text{\_ex}) = L_1\_ex$; cet automate aura trois états nommés 1, 2, 3. L'état 1 sera l'unique état initial et l'état 2 l'unique état final.

❒ 2 – Soit L_2_ex le langage sur Σ_ex des mots dont la longueur est multiple de 3. Représenter graphiquement un automate déterministe et complet noté A_2_ex qui reconnaît L_2_ex, c'est-à-dire vérifiant $L(A_2\text{\_ex}) = L_2\_ex$; cet automate aura trois états nommés 4, 5, 6. L'état 4 sera l'unique état initial et l'unique état final.

Les questions ❒ 3 et ❒ 4 sont consacrées à une première méthode pour calculer un automate reconnaissant l'intersection de deux langages reconnaissables.

❒ 3 – On considère deux langages L_1 et L_2 sur un alphabet quelconque Σ. On suppose que L_1 est reconnu par un automate $A_1 = <\Sigma, Q_1, T_1, I_1, F_1>$; on suppose que L_2 est reconnu par un automate $A_2 = <\Sigma, Q_2, T_2, I_2, F_2>$. Montrer que le langage $L = L_1 \cap L_2$ est reconnu par un automate $A = <\Sigma, Q, T, I, F>$ pour lequel :

- $Q = Q_1 \times Q_2$;
- $I = I_1 \times I_2$;
- $F = F_1 \times F_2$.

En particulier, on précisera l'ensemble T des transitions de A.

❒ 4 – En utilisant la construction de la question précédente, représenter graphiquement un automate noté A_3_ex reconnaissant $L_1\_ex \cap L_2\_ex$. On ne représentera que les états accessibles de cet automate.

La suite du problème propose d'appliquer une seconde façon de calculer un automate reconnaissant l'intersection de deux langages reconnaissables.

❒ 5 – On considère un langage L sur un alphabet quelconque Σ. On suppose que L est reconnu par un automate déterministe et complet $A = <\Sigma, Q, T, \{q_0\}, F>$, où q_0 est l'unique état initial de A. Montrer que le langage complémentaire $\overline{L}$ de L, constitué des mots de Σ^* qui ne sont pas dans L, est reconnu par un automate $A' = <\Sigma, Q, T', \{q_0\}, F'>$ déterministe et complet ; on précisera les ensembles T' et F' ; on justifiera la réponse. En déduire un automate déterministe et complet $A_1'\_ex$ qui reconnaît $\overline{L_1}$ et un automate déterministe et complet $A_2'\_ex$ qui reconnaît $\overline{L_2}$; ces deux automates seront décrits par leur représentation graphique.

Soient deux automates $A_1 = <\Sigma, Q_1, T_1, I_1, F_1>$ et $A_2 = <\Sigma, Q_2, T_2, I_2, F_2>$ reconnaissant respectivement des langages L_1 et L_2 ; on suppose que les ensembles Q_1 et Q_2 sont disjoints ; l'union de ces automates est par définition l'automate $A = <\Sigma, Q_1 \cup Q_2, T_1 \cup T_2, I_1 \cup I_2, F_1 \cup F_2>$. On admet que cet automate A reconnaît le langage $L_1 \cup L_2$.

❒ 6 – On note A_4_ex l'automate obtenu par union de $A_1'\_ex$ et $A_2'\_ex$. Représenter graphiquement un automate déterministe et complet A_5_ex obtenu en appliquant à A_4_ex un algorithme classique de déterminisation. On ne représentera que les états accessibles de cet automate.

❒ 7 – Déduire de l'automate A_5_ex obtenu à la question ❒ 6, un automate reconnaissant $L_1\_ex \cap L_2\_ex$.

2. Problème d'algorithmique et programmation (temps conseillé : 2 h 20 mn)

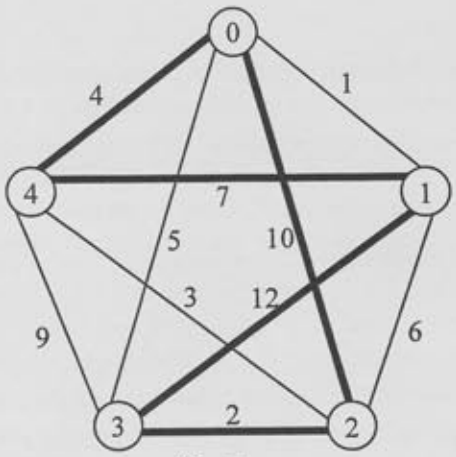
On s'intéresse au **problème du voyageur de commerce** dans un graphe complet symétrique valué.

Préliminaire concernant la programmation : il faudra écrire des fonctions ou des procédures à l'aide d'un langage de programmation qui pourra être soit **Caml**, soit **Pascal**, tout autre langage étant exclu. **Indiquer en début de problème le langage de programmation choisi ; il est interdit de modifier ce choix au cours de l'épreuve**. Certaines questions du problème sont formulées différemment selon le langage de programmation ; cela est indiqué chaque fois que nécessaire. Par ailleurs, pour écrire une fonction ou une procédure en langage de programmation, le candidat pourra définir des fonctions ou des procédures auxiliaires qu'il explicitera ou faire appel à d'autres fonctions ou procédures définies dans les questions précédentes.

Dans l'énoncé du problème, un même identificateur écrit dans deux polices de caractères différentes désigne la même entité, mais du point de vue mathématique pour la police écrite en italique (par exemple : n) et du point de vue informatique pour celle écrite en romain (par exemple : n).

Un **graphe** est défini par un ensemble fini, noté X, d'éléments appelés **sommets** et par l'ensemble de ses **arêtes** ; une arête $\{x, y\}$ est une paire de sommets distincts x et y qui sont les **extrémités** de cette arête. Un graphe est **complet** si toute paire de sommets distincts est une arête. Le nombre de sommets d'un graphe s'appelle l'**ordre** du graphe ; tous les graphes considérés dans ce problème sont d'ordre au moins 3. Si un graphe est d'ordre n, **ses sommets sont nommés 0, 1, 2, ..., $n-1$** dans ce problème. Les graphes considérés ici seront des graphes complets **valués**, ce qui signifie qu'on attribue à chaque arête un entier naturel nommé **poids** de cette arête. Dans tout ce problème, les graphes considérés seront d'ordre au plus 100 et les poids ne dépasseront pas 100.

Un graphe complet valué d'ordre n peut être représenté par une matrice carrée symétrique de dimension $n \times n$ dont les cases sont indicées par (i, j) avec $0 \leq i \leq n-1$ et $0 \leq j \leq n-1$; on notera *poids* cette matrice indiquant dans la case d'indice i et j le poids $poids(\{i, j\})$ de l'arête $\{i, j\}$; par convention, cette matrice a des zéros sur la diagonale. On représente ci-après un graphe complet valué G_ex et la matrice $poids\_ex$ correspondante.



G_ex

i \ j	0	1	2	3	4
0	0	1	10	5	4
1	1	0	6	12	7
2	10	6	0	2	3
3	5	12	2	0	9
4	4	7	3	9	0

poids_ex

Soit G un graphe complet d'ordre n. Soit $(t_0, t_1, t_2, \ldots, t_{n-1})$ une permutation de l'ensemble X des sommets de G. Par définition, le **tour T induit par cette permutation** est un graphe ayant X comme ensemble de sommets et dont l'ensemble des arêtes est : $\{\{t_i, t_{i+1}\} \mid 0 \leq i \leq n-2\} \cup \{t_{n-1}, t_0\}$. Par exemple, le tour *T_ex* induit par la permutation (0, 2, 3, 1, 4) est représenté en gras ci-dessus.

On appelle **poids d'un tour T d'un graphe complet valué G**, et on note ***poids*(G, T)**, la somme des poids des arêtes de T dans G :

$$poids(G, T) = \sum_{i=0}^{n-2} poids(\{t_i, t_{i+1}\}) + poids(\{t_{n-1}, t_0\}).$$

Ainsi : $poids(G\_ex, T\_ex) = 10+2+12+7+4 = 35$.

Le problème auquel on s'intéresse ici est celui de la détermination d'un tour T de G tel que *poids* (G, T) soit minimum. On dit alors qu'il s'agit d'un **tour de poids minimum**.

On appelle dans ce problème **chaîne** à p sommets d'un graphe complet G une suite ordonnée $C = (c_0, c_1, c_2, \ldots, c_{p-1})$ de p sommets distincts de G ; en notant n l'ordre de G, on a alors nécessairement $1 \leq p \leq n$. Le poids d'une telle chaîne vaut :

$$poids(G, C) = \sum_{i=0}^{p-2} poids(\{c_i, c_{i+1}\}).$$

Par exemple, pour la chaîne $C\_ex = (0, 3, 1)$ du graphe G_ex, $poids(G\_ex, C\_ex) = 5 + 12 = 17$.

On pourra sans ambiguïté évoquer le poids d'une arête, ou le poids d'un tour, ou le poids d'une chaîne.

Indications pour la programmation

Caml : On définit l'identificateur suivant :

```
let MAX_POIDS = 100;;
```

On supposera que les poids des arêtes des graphes traités ne dépassent jamais la valeur `MAX_POIDS`.

La matrice des poids est codée par un tableau à deux dimensions (un vecteur de vecteurs) $n \times n$ (où n est l'ordre du graphe) ; la matrice *poids_ex* représentant le graphe *G_ex* est ainsi codée par :

```
let poids_ex = [|[|0; 1; 10; 5; 4|]; [|1; 0; 6; 12; 7|];
 [|10; 6; 0; 2; 3|];[|5; 12; 2; 0; 9|];[|4; 7; 3; 9; 0|]|];;
```

On accède alors au poids de l'arête $\{i, j\}$ du graphe *G_ex* par : `poids_ex.(i).(j)`.

Un tour est codé par un tableau (vecteur) de dimension égale à l'ordre du graphe et contenant la suite des sommets d'une permutation induisant ce tour. Le tour *T_ex* peut être défini par :

```
let T_ex = [|0; 2; 3; 1; 4|];;
```

De même, une chaîne est codée par un tableau contenant la suite des sommets de la chaîne ; on pourrait coder la chaîne $C\_ex = (0, 3, 1)$ par :

```
let C_ex = [|0; 3; 1|];;
```

On remarque donc qu'une matrice de poids, ou bien un tour, ou bien une chaîne sont codés avec des tableaux ayant la stricte dimension nécessaire, et non avec des tableaux « surdimensionnés ». Pour

connaître la dimension d'un tableau, on peut utiliser la fonction `vect_length`. Par exemple, avec les définitions ci-dessus, `vect_length poids_ex` vaut 5 et `vect_length C_ex` vaut 3.
Un sous-ensemble S de l'ensemble des sommets d'un graphe G d'ordre n est codé par un tableau de longueur n donnant la fonction caractéristique de cet ensemble ; ainsi, en notant `S` ce tableau et i un entier compris entre 0 et $n-1$, `S.(i)` vaut 1 si le sommet i appartient à S et vaut 0 si le sommet i n'appartient pas à S.

Fin des indications pour Caml

Pascal : On utilise les définitions suivantes.

```
const
        MAX_SOMMETS = 100;
        MAX_POIDS = 100;

type Matrice = array [0 .. MAX_SOMMETS - 1, 0 .. MAX_SOMMETS - 1] of Integer;
type Table = array [0 .. MAX_SOMMETS - 1] of Integer;
```

La constante `MAX_SOMMETS` donne une borne supérieure de l'ordre des graphes qui pourront être traités. Par ailleurs, on supposera que les poids des arêtes des graphes traités ne dépassent jamais la valeur de la constante `MAX_POIDS`.
Le type `Matrice` sert à coder la matrice des poids du graphe considéré. La matrice *poids_ex* représentant le graphe *G_ex* est ainsi codée avec une variable `poids_ex` de type `Matrice` par :

```
poids_ex[0, 1] := 1;  poids_ex[1, 0] := 1;  poids_ex[0, 2] := 10;
poids_ex[2, 0] := 10; poids_ex[0, 3] := 5;  poids_ex[3, 0] := 5;
poids_ex[0, 4] := 4;  poids_ex[4, 0] := 4;  poids_ex[1, 2] := 6;
poids_ex[2, 1] := 6;  poids_ex[1, 3] := 12; poids_ex[3, 1] := 12;
poids_ex[1, 4] := 7;  poids_ex[4, 1] := 7;  poids_ex[2, 3] := 2;
poids_ex[3, 2] := 2;  poids_ex[2, 4] := 3;  poids_ex[4, 2] := 3;
poids_ex[3, 4] := 9;  poids_ex[4, 3] := 9;  poids_ex[0, 0] := 0;
poids_ex[1, 1] := 0;  poids_ex[2, 2] := 0;  poids_ex[3, 3] := 0;
poids_ex[4, 4] := 0;
```

Si la matrice des poids d'un graphe, de type `Matrice`, s'appelle `poids`, on peut accéder au poids de l'arête $\{i, j\}$ par `poids[i, j]`.
Le type `Table` sera utilisé dans les cas suivants :

- pour coder un tour, on utilise un tableau de type `Table` contenant, à partir de l'indice 0, la suite des sommets d'une permutation induisant ce tour ;
- pour coder une chaîne, on utilise un tableau de type `Table` contenant, à partir de l'indice 0, la suite des sommets de cette chaîne ;
- un sous-ensemble S de l'ensemble des sommets d'un graphe G d'ordre n ($n \leq$ `MAX_SOMMETS`) est codé par un tableau de type `Table` donnant la fonction caractéristique de cet ensemble ; ainsi, en notant `S` ce tableau et i un entier compris entre 0 et $n-1$, `S[i]` vaut 1 si le sommet i appartient à S et vaut 0 si le sommet i n'appartient pas à S.

Fin des indications pour Pascal

Première partie : questions préliminaires

❒ 8 – On considère un graphe G complet d'ordre n. Donner le nombre de tours différents de G. Une méthode pour déterminer un tour de poids minimum consiste à dresser la liste exhaustive de tous les tours possibles, à calculer le poids de chacun des tours et à retenir un tour de poids minimum. On suppose que n vaut 50 ; dire si cette méthode est raisonnable d'un point de vue pratique en justifiant brièvement la réponse.

❒ 9 – Il s'agit de calculer le poids d'un tour dans un graphe complet valué donné.

Caml : Écrire en Caml une fonction nommée `poids_tour` telle que, si `poids` est la matrice donnant les poids des arêtes d'un graphe complet valué G et `T` un tableau codant un tour de G, `poids_tour poids T` renvoie le poids du tour considéré.

Pascal : Écrire en Pascal une fonction nommée `poids_tour` telle que, si `poids`, de type `Matrice`, contient les poids des arêtes d'un graphe complet valué G, si `n` est un entier donnant

l'ordre de G et si `T`, de type `Table`, code un tour de G, `poids_tour(poids, n, T)` renvoie le poids du tour considéré.

❒ 10 – Indiquer la complexité de la fonction `poids_tour` écrite dans la question précédente.

❒ 11 – Soit G un graphe complet valué d'ensemble de sommets X. On considère un sous-ensemble S de X ; on suppose que S est non vide et différent de X. Soit x un sommet de X n'appartenant pas à S. Il s'agit de déterminer un sommet $p(x)$ de S tel que, pour tout sommet s de S, on ait : $poids(\{x, p(x)\}) \leq poids(\{x, s\})$; si plusieurs sommets sont candidats, on prend pour $p(x)$ celui dont le nom est le plus petit. On dira alors que $p(x)$ est le sommet de S **le plus proche** de x.

Caml : Écrire en Caml une fonction nommée `plus_proche` telle que,

- si `poids` est la matrice donnant les poids des arêtes d'un graphe complet valué G d'ensemble de sommets X,
- si `S` est un tableau codant un sous-ensemble S de X non vide et différent de X,
- si `x` est un entier correspondant à un sommet x de G n'appartenant pas à S,

alors `plus_proche poids S x` renvoie un entier correspondant au sommet de S le plus proche de x.

Pascal : Écrire en Pascal une fonction nommée `plus_proche` telle que,

- si `poids`, de type `Matrice`, contient les poids des arêtes d'un graphe complet valué G d'ensemble de sommets X,
- si `n` est un entier donnant l'ordre de G,
- si `S`, de type `Table`, code un sous-ensemble S de X non vide et différent de X,
- si `x` est un entier correspondant à un sommet x de G n'appartenant pas à S,

alors `plus_proche(poids, n, S, x)` renvoie un entier correspondant au sommet de S le plus proche de x.

❒ 12 – Indiquer la complexité de la fonction `plus_proche` écrite dans la question précédente.

Deuxième partie : l'heuristique du plus proche voisin

On cherche dans cette partie à définir une méthode pour construire « rapidement » un tour de « faible poids » sans être nécessairement de poids minimum. Une telle méthode s'appelle une heuristique.

L'heuristique décrite ci-après s'appelle l'**heuristique du plus proche voisin**. On dispose d'un graphe G complet valué d'ordre n et on souhaite déterminer un tour de « faible poids ». Pour construire un tel tour T, on détermine comme suit une permutation $(t_0, t_1, t_2, \ldots, t_{n-1})$ des sommets de G induisant ce tour. On choisit d'abord un sommet quelconque du graphe et on le note t_0. On cherche alors, parmi les sommets de G autres que t_0, le sommet le plus proche de t_0. On note t_1 un tel sommet. On continue ainsi : quand, pour $1 \leq i \leq n-1$, on a déterminé successivement les sommets $t_1, t_2, \ldots, t_{i-1}$, on cherche le sommet t_i de G le plus proche de t_{i-1} parmi les sommets n'appartenant pas à l'ensemble $\{t_0, t_1, t_2, \ldots, t_{i-1}\}$. Le résultat de l'heuristique du plus proche voisin dépend donc uniquement du choix du **sommet de départ**, c'est-à-dire de t_0.

❒ 13 – Indiquer le tour obtenu par l'heuristique du plus proche voisin si on l'applique au graphe *G_ex* en choisissant le sommet 0 comme sommet de départ. Préciser le poids de ce tour.

❒ 14 – Indiquer le tour obtenu par l'heuristique du plus proche voisin si on l'applique au graphe *G_ex* en choisissant le sommet 1 comme sommet de départ. Préciser le poids de ce tour.

❒ 15 – L'objectif de cette question est de programmer l'heuristique du plus proche voisin.

Caml : Écrire en Caml une fonction nommée `tour_plus_proche` telle que,

- si `poids` est la matrice donnant les poids des arêtes d'un graphe complet valué G d'ordre n,
- si `depart` est une valeur entière vérifiant $0 \leq$ `depart` $\leq n-1$,

alors `tour_plus_proche poids depart` renvoie un tableau de longueur n contenant un codage du tour obtenu par l'heuristique du plus proche voisin appliquée au sommet de nom `depart`.

Pascal : Écrire en Pascal une procédure nommée `tour_plus_proche` telle que,

- si `poids`, de type `Matrice`, contient les poids des arêtes d'un graphe complet valué G,
- si `n` est un entier donnant l'ordre de G,
- si `depart` est un entier vérifiant $0 \leq$ `depart` $\leq n-1$,
- si `tour` est de type `Table`,

alors `tour_plus_proche(poids, n, depart, tour)` remplit le tableau `tour` pour qu'il contienne un codage du tour obtenu par l'heuristique du plus proche voisin appliquée au sommet de nom `depart`.

❒ 16 – Calculer la complexité de l'algorithme de la question précédente.

❒ 17 – À l'aide d'un graphe complet valué d'ordre 4 et en choisissant un sommet de départ de façon appropriée, montrer que l'heuristique du plus proche voisin ne détermine pas toujours un tour de poids minimum.

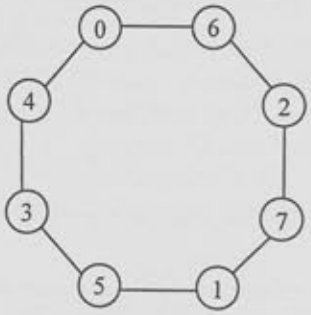
Troisième partie : méthode par amélioration itérative

On considère une transformation d'un tour en un autre ; on appelle **transformation 2-opt** cette transformation ; elle est définie comme suit. On note $T = (t_0, t_1, t_2, \ldots, t_{n-1})$ une permutation induisant un tour T. Pour définir une transformation 2-opt de T, on précise deux entiers i et j vérifiant : $0 \leq i \leq n-3$, $i+2 \leq j \leq n-1$ et $(i,j) \neq (0, n-1)$; on définit plus précisément une transformation notée *2-opt*(T, i, j) ; on pose $i' = i+1$ et $j' = (j+1)$ modulo n ; le tour $T' =$ *2-opt*(T, i, j) est le tour dont les arêtes sont obtenues à partir de celles de T :

- en retirant de T les arêtes $\{t_i, t_{i'}\}$ et $\{t_j, t_{j'}\}$;
- en ajoutant les arêtes $\{t_i, t_j\}$ et $\{t_{i'}, t_{j'}\}$.

Les paramètres i et j de la transformation 2-opt ne sont donc pas des sommets mais des indices de sommets relativement à T ; on rappelle que les indices dans T commencent à 0.

❒ 18 – Dans un graphe complet d'ordre 8, on considère le tour T induit par la permutation $T = (3, 4, 0, 6, 2, 7, 1, 5)$. On pose $T' =$ *2-opt*$(T, 1, 6)$. Dessiner le tour T comme ci-contre et représenter le tour T' sur la même figure par un tracé qui se différencie de celui de T (autre couleur, trait plus gras...). Indiquer explicitement une permutation induisant T' ; on remarquera que cette permutation peut commencer par 3, 4, ... ; c'est ce début qui sera retenu.



❒ 19 – Il s'agit d'écrire en langage de programmation le calcul d'un tour obtenu par une transformation 2-opt à partir d'un tour donné.

Caml : Écrire en Caml une fonction nommée `deux_opt` telle que,

- si `T` est un tableau codant un tour T d'un graphe G d'ordre n,
- si `i` et `j` sont deux entiers vérifiant $0 \leq$ `i` $\leq n-3$, `i` $+2 \leq$ `j` $\leq n-1$ et `(i, j)` $\neq (0, n-1)$,

alors `deux_opt T i j` transforme le tableau `T` pour que celui-ci code le tour obtenu par la transformation *2-opt*(T, i, j).
La fonction ne vérifiera pas que les entiers `i` et `j` respectent les inégalités indiquées.

Pascal : Écrire en Pascal une procédure nommée `deux_opt` telle que,

- si `T`, de type `Table`, code un tour T d'un graphe G d'ordre n,
- si `i` et `j` sont deux entiers vérifiant $0 \leq$ `i` $\leq n-3$, `i` $+2 \leq$ `j` $\leq n-1$ et `(i, j)` $\neq (0, n-1)$,

alors `deux_opt(T, i, j)` transforme le tableau `T` pour que celui-ci code le tour obtenu par la transformation *2-opt*(T, i, j).
L'ordre n du graphe n'intervient pas dans l'écriture de cette fonction.
La procédure ne vérifiera pas que les entiers `i` et `j` respectent les inégalités indiquées.

❒ 20 – Indiquer l'ordre de grandeur de la complexité dans le pire des cas de la transformation `deux_opt` programmée dans la question précédente.

Pour déterminer dans un graphe G un tour de « faible poids », on peut envisager l'heuristique suivante : on choisit un tour initial qu'on note T ; tant qu'il existe une transformation 2-opt transformant T en un tour de poids **strictement** plus petit, on choisit une telle transformation qu'on applique à T pour obtenir un nouveau tour qu'on substitue à T et qu'on nomme encore T. Ainsi, quand l'algorithme se termine, il n'existe aucune transformation 2-opt permettant de transformer le tour final T en un tour de poids strictement plus petit. Cette méthode est qualifiée de **méthode par amélioration itérative**.

❒ 21 – On considère le graphe *G_ex* ; appliquer la méthode par amélioration itérative décrite ci-dessus au tour initial induit par la permutation (0, 1, 2, 3, 4).

❒ 22 – On considère encore le graphe *G_ex* ; appliquer la méthode par amélioration itérative décrite ci-dessus au tour initial induit par la permutation (1, 0, 4, 2, 3). On pourra commencer par redessiner le graphe de façon que le tour considéré soit le « tour extérieur » du dessin du graphe.

❒ 23 – Indiquer si le tour obtenu à l'issue de la méthode par amélioration itérative est ou non toujours de poids minimum. Justifier la réponse.

Quatrième partie : méthode par séparation et évaluation

On souhaite dans cette partie développer une méthode qui détermine un tour de poids minimum. Pour cela, on va d'abord définir puis utiliser l'**évaluation d'une chaîne**.

Soit une chaîne $C = (c_0, c_1, c_2, \ldots, c_{p-1})$ constituée de p sommets distincts d'un graphe complet valué G d'ordre n ; en supposant $1 \leq p < n$, on utilise les notations ci-dessous :

- U est l'ensemble des sommets de G qui ne sont pas dans C ; autrement dit, $U = X - \{c_0, c_1, c_2, \ldots, c_{p-1}\}$;
- pour $x = c_0$ ou $x = c_{p-1}$, on considère une arête la plus légère parmi les arêtes dont une extrémité est x et l'autre est dans U ; *eval1*(G, C, x) désigne le poids de cette arête ;
- pour $x \in U$, on considère deux arêtes les plus légères parmi celles dont une extrémité est x et l'autre (distincte de x) est dans $U \cup \{c_0, c_{p-1}\}$; *eval2*(G, C, x) désigne la somme des poids de ces deux arêtes ;
- $eval(G, C) = poids(G, C) + \left\lceil \frac{1}{2}\left(eval1(G,C,c_0) + eval1(G,C,c_{p-1}) + \sum_{x \in U} eval2(G,C,x) \right) \right\rceil$, où, si r représente un nombre réel, $\lceil r \rceil$ représente la partie entière par excès de r, c'est-à-dire le plus petit entier supérieur ou égal à r.

❒ 24 – Dans le graphe *G_ex*, on considère la chaîne *C_ex* = (0, 3, 1) ; calculer *eval*(*G_ex*, *C_ex*).

On dit qu'un tour contient une chaîne $C = (c_0, c_1, c_2, \ldots, c_{p-1})$ s'il est induit par une permutation $(t_0, t_1, t_2, \ldots, t_{p-1}, t_p, \ldots, t_{n-1})$ avec $t_i = c_i$ pour tout i vérifiant $0 \leq i \leq p-1$.

❒ 25 – On considère un graphe complet valué G d'ordre n, un entier p vérifiant $1 \leq p < n$, une chaîne $C = (c_0, c_1, c_2, \ldots, c_{p-1})$ de G et un tour T contenant la chaîne C. Montrer la relation : $poids(G, T) \geq eval(G, C)$.

L'objectif est maintenant de construire un algorithme récursif permettant de déterminer un tour de poids minimum dans un graphe complet valué.

Dans cette fin de problème, on code tous les tours avec des permutations commençant par le sommet 0 et, pour les chaînes $(c_0, c_1, c_2, \ldots, c_{p-1})$ considérées, on a toujours $c_0 = 0$.

On définit un algorithme *tour_min* qui a pour données :

- la matrice des poids d'un graphe G d'ordre n,
- une chaîne $C = (c_0, c_1, c_2, \ldots, c_{p-1})$ avec $1 \leq p \leq n$,
- un tableau nommé *meilleur_tour* codant un tour.

S'il existe parmi les tours contenant la chaîne C un tour de poids strictement inférieur à celui de *meilleur_tour*, l'algorithme, appliqué aux données précédentes, remplace le contenu du tableau *meilleur_tour* par une permutation induisant un tour de poids minimum parmi les tours de G contenant la chaîne C. Le principe de cet algorithme est décrit ci-dessous :

- si $p = n$, la chaîne C définit une permutation induisant un tour ; on remplace si nécessaire le contenu de *meilleur_tour* par celui de la chaîne C et l'algorithme se termine ;
- sinon, on calcule *eval*(G, C) ; si cette évaluation est supérieure ou égale au poids de *meilleur_tour*, il n'existe aucun tour contenant C qui soit de poids strictement inférieur à celui de *meilleur_tour* et dans ce cas l'algorithme se termine ;
- si l'algorithme ne s'est pas terminé ci-dessus, on résout le problème en appliquant l'algorithme *tour_min* successivement à chacune des $n - p$ chaînes $(c_0, c_1, c_2, \ldots, c_{p-1}, x)$ prolongeant la chaîne C d'un sommet x ne figurant pas dans C.

❒ 26 – Détailler l'application de l'algorithme *tour_min* au graphe *G_ex* avec la chaîne (0, 3) et le tableau *meilleur_tour* qui contient initialement la permutation (0, 1, 2, 3, 4).

Les questions suivantes ont pour objectif la programmation de l'algorithme *tour_min*.

❒ 27 – On considère un graphe complet valué G d'ensemble de sommets X, un sous-ensemble S de X non vide et différent de X, et un sommet x n'appartenant pas à S. Par définition, la fonction *somme_deux_plus_legeres* calcule le minimum de la somme des poids de deux arêtes distinctes dont une extrémité est x et l'autre est dans S. Il s'agit de programmer cette fonction en utilisant nécessairement la fonction `plus_proche` qui a fait l'objet de la question ❒ 11.

Caml : Écrire en Caml une fonction nommée `somme_deux_plus_legeres` telle que,

- si `poids` est la matrice donnant les poids des arêtes d'un graphe complet valué G d'ensemble de sommets X,
- si `S` est un tableau codant un sous-ensemble S de X non vide et différent de X,
- si `x` est un entier correspondant à un sommet x de G n'appartenant pas à S,

alors `somme_deux_plus_legeres poids S x` renvoie la valeur de la fonction *somme_deux_plus_legeres* appliquée au graphe G, à l'ensemble S et à x.

Pascal : Écrire en Pascal une fonction nommée `somme_deux_plus_legeres` telle que :

- si `poids`, de type `Matrice`, contient les poids des arêtes d'un graphe complet valué G d'ensemble de sommets X,
- si `n` est un entier donnant l'ordre de G,
- si `S`, de type `Table`, code un sous-ensemble S de X non vide et différent de X,
- si `x` est un entier correspondant à un sommet x de G n'appartenant pas à S,

alors `somme_deux_plus_legeres(poids, n, S, x)` renvoie la valeur de la fonction *somme_deux_plus_legeres* appliquée au graphe G, à l'ensemble S et à x.

❒ 28 – Il s'agit de programmer le calcul de l'évaluation d'une chaîne.

Caml : Écrire en Caml une fonction nommée `eval` telle que,

- si `poids` est la matrice donnant les poids des arêtes d'un graphe complet valué G,
- si `C` est un tableau codant une chaîne C ne contenant pas tous les sommets de G,

alors `eval poids C` renvoie la valeur de *eval*(G, C).

Pascal : Écrire en Pascal une fonction nommée `eval` telle que,

- si `poids`, de type `Matrice`, contient les poids des arêtes d'un graphe complet valué G,

- si `n` est un entier donnant l'ordre de G,
- si `C`, de type `Table`, code une chaîne C ne contenant pas tous les sommets de G,
- si `p` est un entier donnant le nombre de sommets de C,

alors `eval(poids, n, C, p)` renvoie la valeur de *eval*(G, C).

❒ 29 – Il s'agit de programmer l'algorithme récursif *tour_min*.

Caml : Écrire en Caml une fonction récursive nommée `tour_min` telle que,

- si `poids` est la matrice donnant les poids des arêtes d'un graphe complet valué G,
- si `C` est un tableau codant une chaîne C,
- si `meilleur_tour` est un tableau codant un tour *meilleur_tour* de G,

alors `tour_min poids C meilleur_tour` modifie le tableau `meilleur_tour` selon ce qui est indiqué plus haut concernant l'algorithme *tour_min* appliqué à G, C et *meilleur_tour*.

Pascal : Écrire en Pascal une procédure récursive nommée `tour_min` telle que,

- si `poids`, de type `Matrice`, contient les poids des arêtes d'un graphe complet valué G,
- si `n` est un entier donnant l'ordre de G,
- si `C`, de type `Table`, contient les sommets d'une chaîne C,
- si `p` est un entier donnant le nombre de sommets de C,
- si `meilleur_tour`, de type `Table`, code un tour *meilleur_tour* de G,

alors `tour_min(poids, n, C, p, meilleur_tour)` modifie le tableau `meilleur_tour` selon ce qui est indiqué plus haut concernant l'algorithme *tour_min* appliqué à G, C et *meilleur_tour*.

❒ 30 – Proposer une méthode pour déterminer un tour de plus petit poids dans un graphe complet valué en utilisant les algorithmes étudiés dans ce problème. On ne demande pas d'écrire cette méthode en langage de programmation.