Projet OS01 : relaxation lagrangienne pour le problème du voyageur de commerce

1. Principe général de la relaxation lagrangienne

Soit un programme mathématique primal très général (P) avec un vecteur x de n variables et m contraintes. La fonction-objectif f et les m fonctions des contraintes $g_i(x)$ sont quelconques. S est un sous-ensemble de $\mathbb{R}^n$ défini par d'autres contraintes. Par exemple, pour un programme linéaire en nombres entiers, f et les g_i sont linéaires et $S = \mathbb{N}^n$.

$$(P) \begin{cases} \min z = f(x) \\ g_i(x) \leq 0 \text{ ou } = 0, i = 1 \dots m \\ x \in S \end{cases}$$

La relaxation lagrangienne (RL) consiste à multiplier les $g_i(x)$ par des poids π_i et à les ajouter à z. Les poids sont appelés *multiplicateurs de Lagrange* ou *variables duales*. Ce sont des réels positifs ou nuls pour des contraintes d'inégalité et de signe quelconque pour des égalités. En notant $g(x)$ le vecteur des $g_i(x)$ on obtient la *fonction de Lagrange* $L(x, \pi) = f(x) + \pi . g(x)$. La *fonction duale* $L(\pi)$ donne le minimum de la fonction de Lagrange pour π fixé, c'est-à-dire $L(\pi) = \min_{x \in S} L(x, \pi)$. On appelle *problème lagrangien* le calcul de la fonction duale pour π fixé.

Propriété 1. $L(\pi) \leq f(x)$ pour tout vecteur π et tout solution réalisable x de (P).

En particulier, $L(\pi)$ est une borne inférieure pour le coût z^* de la solution optimale x^* du primal (P). Une idée naturelle est donc de chercher le vecteur π^* qui donne la meilleure borne inférieure, ce qui revient à résoudre le problème appelé *dual de* (P) :

(D) $\max_\pi \{\min_{x \in S} L(x, \pi)\}$ ou, en utilisant la fonction duale : $\max_\pi L(\pi)$

Pour beaucoup de problèmes à variables continues, même non linéaires, le saut de dualité ou *duality gap* $f(x^*) - L(\pi^*)$ est nul et on peut donc résoudre le dual à la place du primal.

Pour les PLNE, le gap est rarement nul mais les bornes inférieures obtenues sont très bonnes. Ces bornes peuvent servir pour supprimer des nœuds dans des méthodes arborescentes ou évaluer la qualité d'une heuristique. De plus, la solution du dual est souvent quasi-entière et on peut la réparer pour déduire une bonne solution du primal : c'est le principe des *heuristiques lagrangiennes*.

Pour appliquer la RL à un PLNE difficile, il faut choisir les contraintes à relaxer et celles qu'on garde pour définir S. Plus on relaxe de contraintes, plus la borne inférieure obtenue est faible. Mais si on ne relaxe pas assez, le PLNE obtenu reste difficile. Une bonne tactique est de relaxer juste assez de contraintes pour que le calcul de $L(\pi)$ pour π fixé soit facile, c'est-à-dire de complexité polynomiale.

2. Résolution du dual par une méthode de sous-gradient

Propriété 2. La fonction duale est concave et linéaire par morceaux.

Cette propriété ne suppose rien sur la convexité ou la concavité de f et des g_i. Grâce à la concavité, il suffit d'utiliser une méthode de montée pour arriver au maximum global du dual. Cependant, la linéarité par morceaux empêche l'utilisation des techniques classiques de gradient puisque L n'a pas de gradient aux points anguleux. On utilise alors une méthode d'augmentations successives basés sur des *sous-gradients*, qui généralisent la notion de gradient aux fonctions non partout différentiables.

Définition 1. Soit une fonction f à n variables et un point $\bar{x}$ de $\mathbb{R}^n$. Un sous-gradient de f en $\bar{x}$ est une direction d'augmentation de f, c'est-à-dire un vecteur γ tel que : $\forall x, f(x) \leq f(\bar{x}) + \gamma^T.(x - \bar{x})$. En particulier, si f est différentiable en $\bar{x}$ le seul sous-gradient est le gradient classique $\nabla f(\bar{x})$.

A première vue il n'est pas évident de trouver un sous-gradient pour la fonction duale, mais le résultat classique suivant montre que c'est un sous-produit de la résolution du problème lagrangien.

Propriété 3. Soit y la solution du problème lagrangien pour π fixé : $L(\pi) = f(y) + \pi.g(y) = \min_{x \in S}\{f(x) + \pi.g(x)\}$. Alors $g(y)$ est un sous-gradient de la fonction duale en π.

Malheureusement, la fonction peut ne pas augmenter pour certains sous-gradients. Il est donc inutile de chercher à maximiser L dans la direction d'un seul sous-gradient. En pratique, on choisit un sous-gradient, on fait un petit déplacement λ dans sa direction, puis on utilise un nouveau sous-gradient au point obtenu. La structure générale d'un algorithme de sous-gradient est alors :

1. initialiser le vecteur de multiplicateurs π et la borne inférieure LB à zéro
2. initialiser le numéro d'itération j à zéro
3. **répéter**
4. $\quad j \leftarrow j + 1$
5. calculer le minimum de la fonction duale en π :
   $L(\pi) = f(y) + \pi.g(y) = \min_{x \in S}\{f(x) + \pi.g(x)\}$,
   ($\gamma(\pi) = g(y)$ est un sous-gradient de L en π d'après la propriété 3)
6. **si** $L(\pi) > LB$ **alors** $LB \leftarrow L(\pi)$ **fin si**
7. **si** $\|\gamma(\pi)\| > \epsilon$ **alors**
8. déterminer le pas λ_j pour cette itération
9. calculer le point suivant : $\pi \leftarrow \pi + \lambda_j.\gamma(\pi)/\|\gamma(\pi)\|$
10. pour toute contrainte d'inégalité i, forcer π_i à zéro s'il est négatif
11. mettre à jour le pas λ
12. **fin si**
13. **jusqu'à** $(j = j_{max})$ ou $(\|\gamma(\pi)\| < \epsilon)$ ou $(LB = UB)$ ou (y est primal-réalisable).

La fonction duale peut temporairement diminuer ou osciller, c'est pourquoi il vaut mieux mettre à jour la borne inférieure *LB* (*lower bound*) à chaque fois qu'elle est améliorée (ligne 6).

Il y a quatre cas d'arrêt ligne 13. Le premier est un nombre maximum d'itérations j_{max} quand la convergence est trop lente. Le deuxième est quand la norme du sous-gradient devient inférieure à une précision ϵ : la solution du dual est alors optimale mais pas forcément pour le primal à cause du saut de dualité. Le troisième cas est l'atteinte d'une borne supérieure *UB* (*upper bound*) fournie par une heuristique : on sait alors que la borne et l'heuristique sont optimales. Enfin, pour certains problèmes, la solution y du problème lagrangien peut respecter toutes les contraintes de (*P*). Elle est donc optimale pour (*P*) et on peut stopper la méthode de sous-gradient.

Si l'objectif de (*P*) a des valeurs entières, on peut stopper dans le troisième cas quand $UB - LB < 1$. Ainsi, si *UB* = 100 et *LB* atteint 99.01, il ne peut exister de solution avec un coût entier *UB* < 100 et UB est optimale. Notez que LB est rarement entier à cause des multiplicateurs à valeurs réelles.

Voici trois exemples de règles classiques pour mettre à jour le pas λ_j utilisé à l'itération j :

R1 - Pas constant

C'est la règle la plus simple : les λ_j sont tous égaux à une constante positive λ, par exemple $\lambda = 1$.

R2 - Règle de la série divergente

Polyak (1967) a montré que $L(\pi)$ converge vers l'optimum si les λ_j forment une suite divergente :

$$\lambda_j \to 0 \ (j \to +\infty) \ \text{ et } \ \sum_{j=1}^{\infty} \lambda_j = +\infty$$

Un exemple simple est $\lambda_j = 1/j$. La convergence avec cette règle est parfois très lente.

R3 - Règle de la série convergente

Cette règle proposée par Shor (1968) marche bien en pratique mais peut converger vers un point non optimal. On utilise $\lambda_j = \sigma.\alpha^j$, avec σ une constante positive et $0 < \alpha < 1$. Pour éviter une convergence vers un point non optimal, on a intérêt à prendre α assez proche de 1 et σ assez grand.

3. Relaxation lagrangienne pour le problème du voyageur de commerce

Il s'agit de la première application de la RL à l'optimisation combinatoire (Held et Karp, 1971). Avant, la méthode n'était utilisée qu'en optimisation non linéaire. Le problème de voyageur de commerce (*traveling salesman problem* ou TSP) est défini sur un graphe non orienté complet G de n nœuds, avec une matrice C $n \times n$ donnant les coûts des arêtes. Il faut trouver un cycle de coût minimal qui part du dépôt (nœud 1), visite une fois chaque autre nœud, et revient au dépôt.

On appelle *arbre recouvrant* de G (*spanning tree*) un graphe sans cycle qui connecte tous les nœuds. Le calcul d'un arbre recouvrant est possible avec l'algorithme de Prim (1956). A chaque itération, A est l'ensemble des arêtes de l'arbre, S l'ensemble des nœuds connectés et h le coût total :

1. partir d'un arbre réduit au nœud 1 : $S \leftarrow \{1\}, A \leftarrow \emptyset, h \leftarrow 0$
2. **pour** itération $\leftarrow 1$ **à** $n - 1$ **faire**
3. déterminer l'arête $[i, j]$ de coût minimal, avec $i \in S$ et $j \notin S$
4. $h \leftarrow h + C_{ij}$
5. $S \leftarrow S \cup \{j\}$
6. $A \leftarrow A \cup \{[i, j]\}$
7. **fin pour**

Le *degré* $D(i)$ d'un nœud i dans un graphe est le nombre d'arêtes incidentes à i. Un *1-arbre* (*1-tree*) est un arbre recouvrant des nœuds 2 à n, plus deux arêtes incidentes au nœud 1. Il est facile de voir qu'un 1-arbre est formé d'un cycle passant par le nœud 1, avec éventuellement des arbres greffés sur ce cycle. Une solution du TSP est un cas particulier de 1-arbre, où tous les nœuds sont sur le cycle et ont donc tous un degré égal à 2.

Par conséquent, le coût d'un 1-arbre optimal est une borne inférieure du coût optimal pour le TSP. De plus, si le 1-arbre optimal a tous ses nœuds de degré 2, c'est aussi la solution optimale du TSP. Pour calculer un 1-arbre optimal il suffit d'adapter l'algorithme de Prim :

1. calculer un arbre recouvrant sur les nœuds 2 à n avec l'algorithme de Prim (partir du nœud 2).
2. ajouter les deux plus petites arêtes incidentes au nœud 1.

Dans la suite, les TSP à traiter sont définis par n points du plan avec des coordonnées (X_i, Y_i) à stocker dans des réels en double précision (*double* en C). Des exemples sont disponibles dans Moodle sous forme de fichiers-texte avec le format suivant : n sur la première ligne, puis une ligne pour chaque nœud avec les deux coordonnées. Quand le coût optimal est connu, il est sur la première ligne après n. Le plus petit fichier contient $n = 7$ nœuds et s'appelle *small.txt*.

Le coût C_{ij} d'une arête $[i, j]$ est sa longueur, calculée sous forme d'un réel en double précision, et on cherche un cycle de longueur totale minimale. Vous pouvez utiliser C/C++, Delphi, Visual Basic ou Matlab pour programmer les algorithmes. Pour simplifier la programmation en C/C++, Delphi et Visual Basic, on définira une constante $nmax$ pour le nombre maximum de nœuds et on déclarera des tableaux indicés de 1 à $nmax$, même si on utilise seulement les indices 1 à n.

Tâche 1

Préparer trois sous-programmes : $Read\_TSP\ (Filename, n, X, Y)$ pour lire les données contenues dans un fichier-texte de nom donné, $Random\_TSP\ (Xmax, Ymax, n, X, Y)$ pour générer en mémoire un TSP aléatoire (n nœuds tirés au sort dans un rectangle de largeur $Xmax$ et de hauteur $Ymax$), et $Compute\_C\ (n, X, Y, C)$ pour calculer la matrice des longueurs C.

Tâche 2

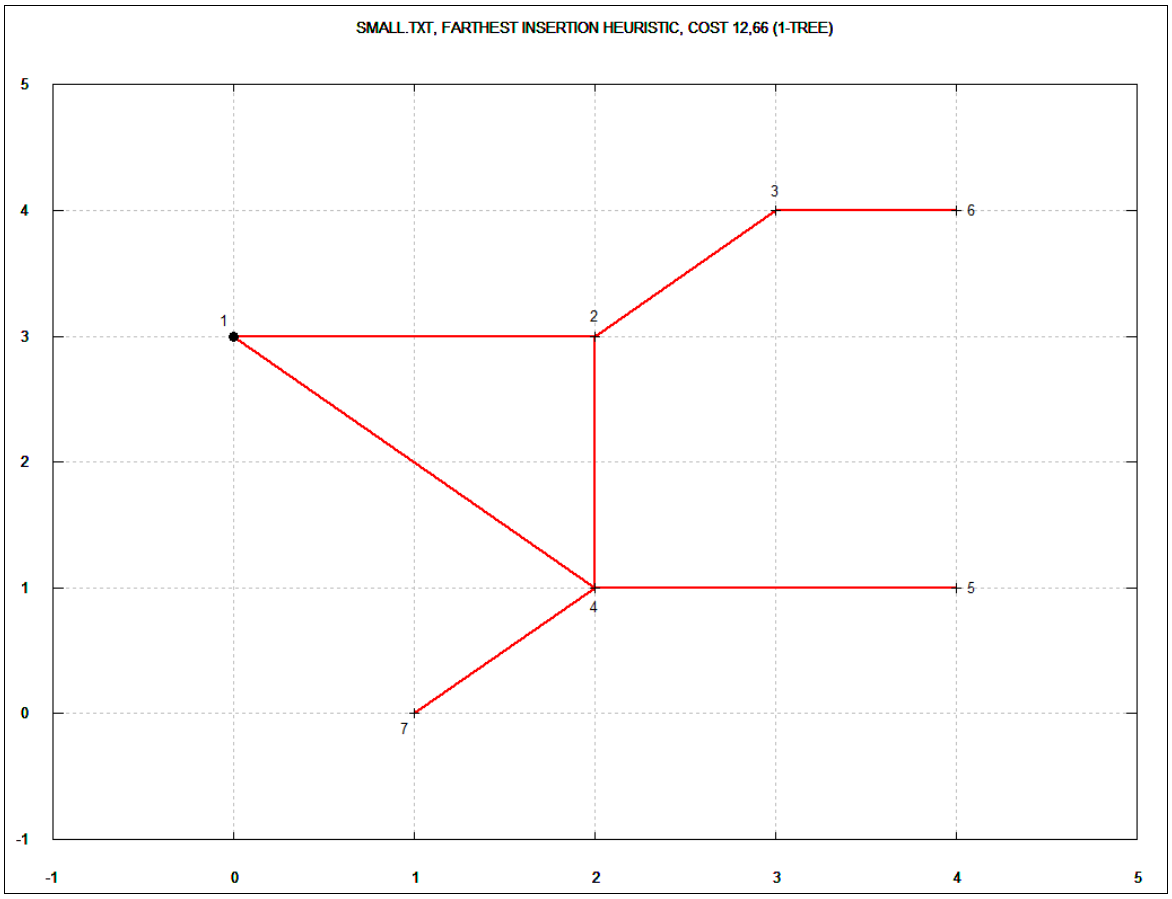
Implémenter l'heuristique $Farthest\_Insertion\ (n, C, UB, Cycle)$ qui donne de bons résultats sur les TSP euclidiens. $Cycle$ contient le cycle trouvé, avec le dépôt 1 au début et à la fin, et UB est le coût du cycle. L'heuristique part d'un cycle réduit à une boucle sur le dépôt : $Cycle = (1,1)$. A chaque itération, elle détermine le nœud i non visité le plus loin des nœuds du cycle, puis elle insère i dans le cycle pour augmenter le moins possible la longueur totale. Tester l'heuristique sur *small.txt*.

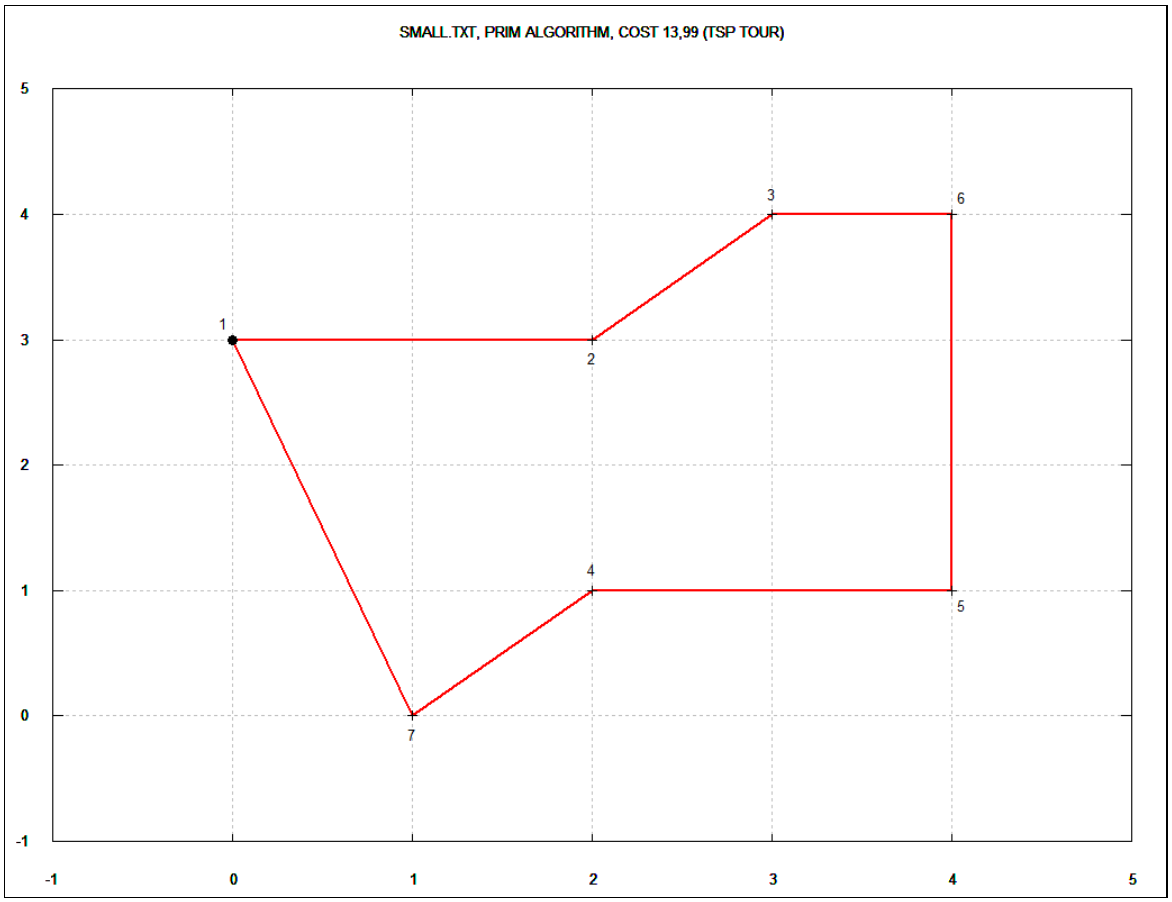
Tâche 3

Programmer l'algorithme de calcul du 1-arbre optimal sous forme d'un sous-programme $One\_Tree\ (n, C, h, A, D)$. Les résultats sont h le coût du 1-arbre trouvé, A la liste des arêtes (par exemple un tableau à deux colonnes avec une ligne par arête) et D un vecteur donnant le degré de chaque nœud dans le 1-arbre. Tester l'algorithme sur *small.txt*.

Pour vérifier vos résultats, voici le contenu de *small.txt* et les résultats des deux algorithmes :

7 13.99
1 0 3
2 2 3
3 3 4
4 2 1
5 4 1
6 4 4
7 1 0


SMALL.TXT, FARTHEST INSERTION HEURISTIC, COST 12,66 (1-TREE)
5
4
3
2
1
0
-1
-1
0
1
2
3
4
5
1
2
3
6
4
5
7


SMALL.TXT, PRIM ALGORITHM, COST 13,99 (TSP TOUR)
5
4
3
2
1
0
-1
-1
0
1
2
3
4
5
1
2
3
6
4
5
7

La borne inférieure donnée par le 1-arbre optimal peut être fortement améliorée par la relaxation lagrangienne. En effet, le TSP peut être modélisé par le PLNE suivant, dans lequel T est l'ensemble des 1-arbres possibles, E l'ensemble des $m = n.(n-1)/2$ arêtes du graphe et $\Gamma(i)$ l'ensemble des nœuds voisins du nœud i dans le graphe. Ce modèle utilise une variable binaire x_{ij} par arête $[i,j]$, égale à 1 si l'arête est choisie dans le 1-arbre. Ce PLNE signifie simplement qu'on cherche un 1-arbre de coût minimal dont tous les nœuds ont un degré égal à 2.

$$\begin{cases} \min\ z = \sum_{[i,j]\in E} c_{ij}.x_{ij} \\ \forall i : \sum_{j\in\Gamma(i)} x_{ij} = 2 \\ x \in T \subseteq \{0,1\}^m \end{cases}$$

La fonction duale a l'expression suivante, avec des multiplicateurs non contraints en signe :

$$L(\pi) = \min_{x\in T}\left\{\sum_{[i,j]\in E} c_{ij}.x_{ij} + \sum_{i=1,n} \pi_i.\left(\sum_{j\in\Gamma(i)} x_{ij} - 2\right)\right\}$$

Si on arrive à calculer la fonction duale pour π fixé, le 1-arbre obtenu y nous donne un sous-gradient en π d'après la propriété 3. La composante n° i de ce sous-gradient est tout simplement :

$$\gamma_i = \sum_{j\in\Gamma(i)} x_{ij} - 2 = D(i) - 2$$

C'est-à-dire le degré du nœud i dans le 1-arbre y, moins deux.

A priori, on ne voit pas comment calculer la fonction duale, mais on peut la réécrire comme suit :

$$\begin{aligned} L(\pi) &= \min_{x\in T}\left\{\sum_{[i,j]\in E} c_{ij}.x_{ij} + \sum_{i=1,n} \pi_i.\left(\sum_{j\in\Gamma(i)} x_{ij} - 2\right)\right\} \\ &= \min_{x\in T}\left\{\sum_{[i,j]\in E} c_{ij}.x_{ij} + \sum_{i=1,n} \pi_i.\left(\sum_{j\in\Gamma(i)} x_{ij}\right)\right\} - 2\sum_{i=1,n}\pi_i \\ &= \min_{x\in T}\left\{\sum_{[i,j]\in E} (c_{ij} + \pi_i + \pi_j)\right\} - 2\sum_{i=1,n}\pi_i \end{aligned}$$

La dernière ligne est obtenue en remarquant que si l'arête $[i,j]$ est prise ($x_{ij} = 1$), alors π_i et π_j sont ajoutés au coût total. Notez que la somme des multiplicateurs à droite est une constante qui n'intervient pas dans la minimisation. Finalement, le calcul revient à déterminer un 1-arbre de coût minimal avec des coûts $c'_{ij} = c_{ij} + \pi_i + \pi_j$. Il suffit d'appeler One_Tree avec la matrice C' des coûts modifiés puis de soustraire au coût du 1-arbre obtenu deux fois la somme des multiplicateurs.

Tâche 4

Implémenter la méthode de sous-gradient pour le TSP en la testant avec la règle R1 et $\lambda = 1$. Tester sur le petit problème *small.txt* : en une itération, le 1-arbre se transforme en une solution optimale du TSP, la même que celle trouvée par $Farthest\_Insertion$. On a donc la preuve que l'heuristique a donné une solution optimale.

Tâche 5

Evaluation de la méthode sur des problèmes-tests. Pour chaque fichier-test disponible sur Moodle, donner la valeur de *LB* pour les règles R1, R2, R3, la valeur de *UB*, et l'écart entre les deux en %. Pour R2, utiliser $\lambda_j = 1/j$. Pour R3, utiliser $\sigma = 50$ et $\alpha = 0.9$. Attention! A part *small.txt*, ces problèmes viennent de la TSPLIB (Reinelt, 1995) et nécessitent des longueurs arrondies au plus proche entier, en faisant $c_{ij} = \lfloor c_{ij} + 0.5 \rfloor$. On demande seulement les coûts, il est inutile de détailler les solutions.

Tâche 6

Evaluation statistique. Générer aléatoirement 100 problèmes à 30 nœuds avec $Random\_TSP$ et préciser pour chaque règle l'écart moyen entre UB et LB en % et le nombre de problèmes où le 1-arbre est une solution optimale du TSP.

Travail à rendre

Ce projet est à faire en formant des groupes de 3 étudiants. Il faut m'envoyer les résultats par e-mail au plus tard le jour de l'examen. Il suffira d'écrire un petit rapport en Word ou PDF, contenant les listings des programmes et les résultats des tests des tâches 5 et 6. Il est inutile de répéter l'énoncé du projet et de donner les résultats pour les autres tâches.