

Stockage et exploitation de tables de routage

Résumé

L'objectif de ce projet est d'implanter, évaluer et comparer plusieurs manières de stocker et d'exploiter la table de routage d'un routeur.

Table des matières

1	Cahier des charges	2
1.1	Routeur	2
1.2	Table de routage	2
1.3	Routeur simple	3
1.4	Routeur avec cache	3
1.4.1	Politique de gestion du cache	3
1.4.2	Cohérence du cache	3
1.4.3	Représentation	4
2	Travail à réaliser	5
2.1	Structure de la table de routage	5
2.2	Structure des paquets	5
2.3	Structure des résultats	6
2.4	Commandes	6
2.5	Arguments de la ligne de commande	7
2.6	Exemple d'utilisation du programme	8
2.7	Compléments sur la mise en œuvre	8
3	Exigences pour la réalisation du projet	9
4	Livrables et échéances	9
5	Principaux critères de notation	12
6	Barème indicatif pour l'évaluation	13

1 Cahier des charges

1.1 Routeur

Un routeur est un élément d'un réseau qui a pour objectif de transmettre les paquets qu'il reçoit sur une interface d'entrée vers la bonne interface de sortie en fonction des informations qui sont stockées dans sa table de routage.

1.2 Table de routage

Lorsqu'un routeur reçoit un paquet, il consulte sa table de routage qui lui indique sur quelle interface le paquet doit être émis. La table de routage est organisée en ligne. Chaque ligne contient :

- une adresse destination (4 octets, entier de 0 à 255 donc 32 bits) ;
- un masque (une adresse IP composée de 1 puis seulement de 0) ;
- un indicateur qui signale si l'utilisation d'une passerelle est nécessaire ;
- l'adresse de la passerelle (nœud intermédiaire) vers laquelle le paquet doit être envoyé ;
- l'interface de sortie à utiliser.

La table de routage contient généralement au moins deux lignes :

- la première qui indique les nœuds qui sont directement accessibles (sans passerelle) ;
- la dernière qui correspond à la route par défaut (adresse destination 0.0.0.0) qui est utilisée lorsque l'adresse IP ne correspond à aucune autre route.

Voici un exemple de table de routage.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
147.127.16.0	0.0.0.0	255.255.240.0	U	0	0	0	eth0
0.0.0.0	147.127.18.200	0.0.0.0	UG	0	0	0	eth0

Le nœud consulte sa table de routage lors de l'émission de chaque paquet. Il compare l'adresse de destination du paquet au couple adresse destination et masque des lignes de la table. La ligne qui convient (les bits des deux adresses sont égaux pour les bits à 1 du masque) est utilisée pour envoyer le paquet. Notons que si plusieurs lignes conviennent, c'est celle qui a le masque le plus long qui est utilisée. La ligne ainsi sélectionnée contient l'interface à utiliser.

Lorsque la ligne indique une passerelle, il pourrait être nécessaire de consulter de nouveau la table de routage pour déterminer l'interface correspondant à la passerelle. Nous supposons que l'interface est toujours donnée sur la ligne.

Dans ce projet, nous considérerons simplement les informations suivantes de la table de routage : la destination, le masque et l'interface. Toutes les autres informations sont ignorées, en particulier l'information concernant la passerelle.

Prenons un exemple avec la table de routage suivante.

Destination	Masque	Interface
147.127.0.0	255.255.0.0	eth1
147.127.18.0	255.255.255.0	eth0

Pour la destination 147.127.18.12, la table de routage contient deux routes possibles. Celle qui est utilisée est celle qui correspond au masque le plus long donc la deuxième ligne. L'interface à utiliser est donc eth0.

Pour la destination 147.127.20.85, seule la première route est possible. C'est donc l'interface eth1 qui est utilisée.

1.3 Routeur simple

Une première implantation simple et naïve d'un routeur consiste à utiliser une structure de données qui conserve toutes les routes. Le principe est de pouvoir retrouver l'interface de sortie à utiliser en fonction de la destination (l'adresse IP contenue dans l'entête du paquet à router).

On utilisera une liste chaînée pour stocker toutes les routes même si d'autres types de structures de données pourraient être utilisés.

1.4 Routeur avec cache

Pour améliorer l'efficacité du routeur, on utilise un cache. Le cache conserve un sous-ensemble des informations de la table de routage, celles qui ont des chances d'être utilisées dans le futur. Généralement, on conserve les dernières informations utilisées. L'idée est que si une information a été utilisée, elle a de bonnes chances d'être utilisée de nouveau.

1.4.1 Politique de gestion du cache

Pour rester efficace, un cache ne peut pas être trop grand. En conséquence pour pouvoir stocker une nouvelle information, il arrive un moment où il faut supprimer une information du cache. Plusieurs politiques peuvent alors être envisagées. Elles définissent la donnée qui sera supprimée du cache :

- FIFO (*First In, First Out*) : la donnée la plus ancienne du cache (la première insérée) ;
- LRU (*Least Recently Used*) : la donnée la moins récemment utilisée ;
- LFU (*Least Frequently Used*) : la donnée la moins utilisée.

1.4.2 Cohérence du cache

Une propriété importante du cache consiste à savoir si la donnée trouvée dans le cache est valide ou non. On parle de cohérence du cache. Dans notre cas, modifier la table de routage nécessiterait d'invalider des routes dans le cache. Comme nous ne traitons pas les changements de table de routage, ce problème ne se pose pas. Pour garantir la validité des données du cache, il suffit de s'assurer que les données insérées sont correctes. Ceci est moins trivial qu'il y paraît au premier abord. Prenons un exemple et considérons la table de routage suivante :

Destination	Masque	Interface
147.127.0.0	255.255.0.0	eth1
147.127.18.0	255.255.255.0	eth0

On suppose le cache initialement vide et une demande de route pour la destination 147.127.25.12. Seule la première route correspond. On pourrait donc ajouter cette route dans le cache. Supposons maintenant que le routeur reçoive une demande de route pour la destination 147.127.18.85. La recherche dans le cache fournit une seule route qui conduit à utiliser l'interface eth1. Cependant, d'après la table de routage complète, l'interface à prendre est eth0. En effet, les deux routes correspondent et c'est celle du masque le plus long qu'il faut utiliser. Dans cet exemple, la route qu'il aurait fallu mettre en cache était :

```
147.127.25.0    255.255.255.0    eth1
```

Le principe est donc de mettre en cache la route obtenue en utilisant la destination demandée, l'interface trouvée dans la table de routage et le masque le plus long qui existe pour « la destination de la route sélectionnée ». Prenons deux autres exemples.

Exemple 1. Soit la table suivante :

```
147.127.127.0    255.255.255.0    eth0
147.128.0.0      255.255.0.0      eth1
0.0.0.0          0.0.0.0          eth2
```

Le cache étant initialement vide, on reçoit le paquet pour l'adresse 147.5.5.5. C'est la règle par défaut qui s'applique. On insère dans le cache la règle 147.5.5.0 255.255.255.0 eth2. Il serait possible de faire plus général (et donc de diminuer les risques de défaut de cache), en insérant 147.5.0.0 255.255.0.0 eth2 ou même 147.0.0.0 255.192.0.0 eth2 puisqu'un masque de 10 bits suffit à être discriminant par rapport aux deux autres règles.

Exemple 2. Ajoutons une règle avec un masque plus long (25 bits) à l'exemple initial.

```
147.127.0.0      255.255.0.0      eth1
147.127.18.0     255.255.255.0    eth0
212.19.8.128     255.255.255.128    eth2
```

Si le cache est vide, et qu'on effectue une demande de route pour la destination 147.127.25.12, c'est toujours la règle 147.127.25.0 255.255.255.0 eth1 qui est insérée dans le cache.

1.4.3 Représentation

Le cache peut être organisé sous la forme d'un tableau, sous la forme d'une liste chaînée, etc. La représentation recommandée est en fait un arbre préfixe ou *trie* (*prefix tree* ou *trie* en anglais). Le mot est l'adresse IP et ses caractères sont les bits qui constituent sa représentation binaire (en base 2). Ici les lettres possibles sont donc au nombre de deux 0 ou 1. Lorsque le préfixe est suffisant pour discriminer une information, l'information est rangée dans le nœud correspondant comme le montre la figure 1.

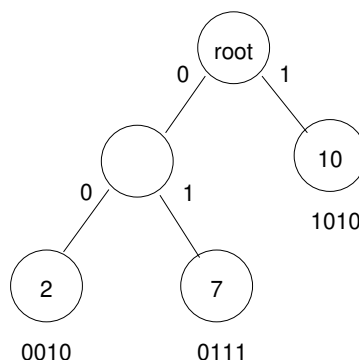


FIGURE 1 – Exemple d’arbre préfixe contenant 2, 7 et 10

2 Travail à réaliser

Vous devez implanter les différents algorithmes présentés. Vous aurez ainsi deux programmes à écrire suivant que le cache est représenté par une liste chaînée (L) ou un arbre préfixe (A). La table de routage sera toujours stockée avec une liste chaînée (L).

Le nom du programme est `routeur_L[LA]` : `routeur_LA` est un routeur avec une liste chaînée pour la table de routage et un arbre préfixe pour le cache ; `routeur_LL` est un routeur avec une liste chaînée pour la table de routage et une liste chaînée pour le cache.

Ces programmes n’auront aucune interaction avec l’utilisateur. Ils seront pilotés en utilisant les paramètres de la ligne de commande.

2.1 Structure de la table de routage

La table de routage sera donnée dans un fichier texte contenant exactement une route par ligne. Chaque ligne contient la destination, suivi du masque puis de l’interface. Ces trois informations sont séparées par au moins un caractère blanc. La destination et le masque sont donnés sous la forme de 4 entiers séparés par des points. L’interface est une chaîne de caractères.

Voici le contenu du fichier `table.txt` contenant une table de routage définissant cinq routes.

```

147.127.16.0 255.255.240.0 eth0
147.127.18.0 255.255.255.0 eth1
147.127.0.0 255.255.255.0 eth2
212.0.0.0 255.0.0.0 eth3
0.0.0.0 0.0.0.0 eth0
    
```

2.2 Structure des paquets

Les paquets à router seront fournis dans un fichier pour lequel chaque ligne correspond à l’adresse IP de destination du paquet (seule information qui nous intéresse).

```

212.212.212.212
147.127.18.80
    
```

```
147.127.18.85
147.127.19.1
147.127.20.20
147.127.32.32
```

2.3 Structure des résultats

Pour chaque paquet traité, le routeur donnera dans un fichier les informations sur son routage : adresse IP destination du paquet et interface utilisée. Les deux apparaîtront sur la même ligne séparées par un et un seul espace. Voici le contenu du fichier pour les paquets et la table de routage précédents.

```
212.212.212.212 eth3
147.127.18.80 eth1
147.127.18.85 eth1
147.127.19.1 eth0
147.127.20.20 eth0
147.127.32.32 eth0
```

2.4 Commandes

Dans le fichier des paquets, on pourra insérer des commandes qui serviront à vérifier le bon comportement du programme. Ces commandes sont définies par un mot, écrit seul sur une ligne. Pour chaque commande reconnue, son nom sera affiché sur le terminal suivi du numéro de ligne dans le fichier de cette commande. Sur les lignes suivantes seront affichés les résultats de la commande (voir ci-après). On laissera une ligne vide avant d'afficher le nom de la commande.

Les commandes sont les suivantes :

- table : afficher toutes les routes de la table de routage.
- cache : afficher toutes les routes du cache.
- stat : afficher toutes les statistiques relatives au cache et à sa politique.
- fin : arrêter le traitement des paquets.

Voici un exemple de paquets à router et de commandes. On veut afficher la table de routage avant le traitement du premier paquet et le cache après avoir routé le premier paquet, puis le deuxième paquet. On arrête là le traitement des paquets sans aller plus loin.

```
table
212.212.212.212
cache
147.127.18.80
cache
fin
147.127.18.85
147.127.19.1
147.127.20.20
147.127.32.32
```

Le résultat de l'exécution ressemblera alors à ce qui suit.

```

table (ligne 1)
147.127.16.0 255.255.240.0 eth0
147.127.18.0 255.255.255.0 eth1
147.127.0.0 255.255.255.0 eth2
212.0.0.0 255.0.0.0 eth3
0.0.0.0 0.0.0.0 eth0

cache (ligne 3)
212.0.0.0 255.0.0.0 eth3

cache (ligne 5)
212.0.0.0 255.0.0.0 eth3
147.127.18.0 255.255.255.0 eth1

fin (ligne 6)

```

Le fichier résultat contiendra alors :

```

212.212.212.212 eth3
147.127.18.80 eth1

```

2.5 Arguments de la ligne de commande

Ces programmes proposeront les options suivantes. Ces options peuvent apparaître dans n'importe quel ordre. La même option peut apparaître plusieurs fois. C'est alors sa dernière apparition qui donne sa valeur. Voici les options :

-c <taille>

Définir la taille du cache. <taille> est la taille du cache. La valeur 0 indique qu'il n'y a pas de cache. La valeur par défaut est 10.

-p FIFO|LRU|LFU

Définir la politique utilisée pour le cache (par défaut FIFO) ;

-s

Afficher les statistiques (nombre de défauts de cache, nombre de demandes de route, taux de défaut de cache). C'est l'option activée par défaut.

-S

Ne pas afficher les statistiques.

-t <fichier>

Définir le nom du fichier contenant les routes de la table de routage. Par défaut, on utilise le fichier `table.txt`.

-q <fichier>

Définir le nom du fichier contenant les paquets à router. Par défaut, on utilise le fichier `paquets.txt`.

`-r <fichier>`

Définir le nom du fichier contenant les résultats (adresse IP destination du paquet et interface utilisée). Par défaut, on utilise le fichier `resultats.txt`.

Si le programme est lancé avec les options `-s -c 10 -p FIFO -S -p LRU -c 50` on aura alors un cache de taille 50, une politique LRU et pas de statistiques affichées.

2.6 Exemple d'utilisation du programme

On pourra utiliser le programme de la manière suivante.

```
routeur_LL -c 15 -t table.txt -q paquets.txt -r resultats.txt
```

Comme les noms des fichiers sont les noms par défaut, la commande suivante est équivalente à la précédente.

```
routeur_LL -c 15
```

2.7 Compléments sur la mise en œuvre

Adresse IP. Une adresse IP est généralement notée comme dans le sujet sous la forme de 4 octets en base 10 (de 0 à 255) séparés par des points. Elle est donc composée de 32 bits (4 fois 8 bits) sous la forme d'un entier. En ada, on peut définir le type Adresse IP comme suit.

```
type T_Adresse_IP is mod 2 ** 32;
```

Ce sont des entiers naturels pour lesquels les calculs se feront modulo l'entier indiqué, `2 ** 32` dans notre cas.

Enfin, on peut appliquer les opérateurs `and` et `or` sur ces types. Ils réalisent alors des opérations bit à bit. Par exemple si `IP1` est une adresse IP, on peut savoir si son bit de poids fort (le plus à gauche) est à 1 en faisant `IP1 and 2 ** 31 /= 0`.

On peut aussi décaler les bits vers la gauche en multipliant par 2 ou vers la droite en divisant par 2. Voir le programme `exemple_adresse_ip.adb` pour des exemples et plus d'explications.

Manipulation de fichiers. Le programme `exemple_fichiers.adb` montre comment manipuler des fichiers en Ada : ouverture en écriture (`Create`) ou en lecture (`Open`). Dans les deux cas, il faut fermer le fichier quand on a fini de s'en servir (`Close`). Les opérations classiques de lecture (`Get`) ou écriture (`Put`) s'appliquent aussi sur les fichiers (en met le descripteur du fichier en premier paramètre).

Pour lire une chaîne de caractères, il est conseillé d'utiliser la fonction `Get_Line` qui retourne une `Unbounded_String`. On peut utiliser `Trim` pour supprimer les caractères blancs en début et en fin de la chaîne (paramètre `Both`).

1. On pourrait aussi faire `IP1 / 2 ** 31 /= 0` (décaler 31 bits à droite).

Ada consomme les caractères de fin de ligne sans qu'on puisse y avoir accès (`End_Of_Line` permet de savoir si elle a été consommée) et gère un curseur sur le fichier. On peut accéder au numéro de ligne du curseur (`Line`) et de colonne (`Col`). La fonction `End_Of_File` permet de savoir si la fin de fichier a été atteinte. Ceci peut poser des problèmes si on utilise un `Get` sur un entier ou un réel. C'est ce qu'illustre le fichier `exemple2.txt` qui contient deux lignes vides à la fin (deux retours à la ligne). Dans le programme on part donc sur la lecture d'une nouvelle ligne et donc la lecture d'un entier alors qu'il n'y en a pas. L'exception `End_Error` se produit alors.

Ligne de commande. Le module `Ada.Command_Line` donne accès à la ligne de commande. Il a déjà été utilisé dans le mini-projet 2. Il est aussi utilisé dans `exemple_fichiers.adb`.

3 Exigences pour la réalisation du projet

1. Le projet se fera en équipes de 3 du même groupe de TD (pas forcément du même groupe de TP). Il pourra y avoir une ou deux équipes de 2.
2. Le langage utilisé est le langage Ada **limité aux concepts vus en cours, TD ou TP et aux éléments présentés dans ce sujet et les exemples fournis**.
3. Les programmes devront compiler et fonctionner sur les machines Linux de l'N7.
4. L'ensemble des concepts du cours devront être mis en œuvre, en particulier :
 - Écriture des spécifications pour tous les programmes et sous-programmes
 - Conception en utilisant la méthode des raffinages
 - Justification des choix des types de données manipulés
 - Conception de modules réutilisables : encapsulation, généricité, TAD...
 - Définition des tests

4 Livrables et échéances

- lundi 1er décembre : **mise en ligne du sujet**.
- 6 décembre : **raffinages du routeur simple**.

Ce livrable prendra la forme d'un document partagé (Google Doc) appelé PIM-PR3-XXX (XXX est le nom de votre équipe). On copiera ce document². Il doit être partagé entre tous les membres de l'équipe et l'enseignant de TD.

Il doit contenir le raffinement du programme principal dans le cas d'un retour simple (sans cache). Il faut bien sûr traiter les arguments de la ligne de commande et exploiter les fichiers (lecture et écriture). Notons que ce raffinement sera donc commun à tous les programmes à écrire.

2. « ce document » est un lien sur lequel il faut cliquer pour accéder au document. Ensuite, il faut le copier en faisant « *Fichier / Créer une copie* ».

La grille d'évaluation des raffinages, déjà utilisée dans le mini-projet 1, devra être complétée.

Le but des raffinages est de structurer la solution en proposant une décomposition fonctionnelle du problème posé. Les actions et expressions complexes deviendront certainement des sous-programmes qu'il faudra regrouper en modules.

Ainsi, une fois les raffinages faits, on listera les modules de l'application et les sous-programmes de ces modules.

On les fera apparaître dans la grille « qui fait quoi ? » qui devra être mise à jour au fur et à mesure de l'avancement du projet.

- à chaque séance de projet, **point d'avancement** avec l'enseignant de TD.
- samedi 20 décembre : **remise d'une version 1 du projet avec un routeur simple** (sans cache) qui montre que tous les aspects du projet sont traités (analyse de la ligne de commande, lecture et écriture des fichiers, routage des paquets par le routeur). La grille « Qui fait quoi ? » doit impérativement être complétée. En fait, elle doit l'être au fur et à mesure du déroulement du projet. Vous devez aussi valider régulièrement, avec un message clair, vos avancées sur le projet et les pousser sur Gitlab.

À cette même date, vous devez aussi rendre les **raffinages du routeur avec cache**. Le principe sera d'ajouter une nouvelle section dans le document partagé et de reprendre la décomposition des actions complexes qui doivent être modifiées suite à l'ajout du cache et de ses trois politiques.

- lundi 12 janvier : rendu des livrables via Gitlab :
 1. les sources du projet dans le dossier `src`.
 2. le **manuel utilisateur** (`manuel.pdf`). Il décrit comment utiliser les programmes développés. Il est illustré avec des copies d'écran.
 3. le **rapport** du projet (`rapport.pdf`).

Le **rapport** doit au moins contenir les informations suivantes :

- un résumé qui décrit l'objectif et le contenu du rapport (10 lignes maxi),
- une introduction qui présente le problème traité³ et le plan du document,
- l'architecture de l'application en modules,
- la présentation des principaux choix réalisés,
- la présentation des principaux algorithmes et types de données,
- la démarche adoptée pour tester le programme,
- les difficultés rencontrées et les solutions adoptées en justifiant vos choix (en particulier quand vous avez envisagé plusieurs solutions),
- l'organisation de l'équipe (qui a fait quoi, etc.),
- un bilan technique donnant un état d'avancement du projet et les perspectives d'amélioration / évolution,

3. La présentation du problème doit être concise car les enseignants connaissent le sujet !

- en annexe, un bilan *personnel* et *individuel* : intérêt, temps passé, temps passé à la conception, temps passé à l'implantation, temps passé à la mise au point, temps passé sur le rapport, enseignements tirés de ce projet, etc.

Remarque : Bien sûr, les modifications faites par rapport aux premiers livrables (raffinages, structures de données, interfaces des modules, programmes de tests) devront être indiquées et argumentées dans le rapport.

- mardi 13 janvier : Remise des **livrables pour l'oral**, décrits dans la rubrique suivante : « Recette du projet » :
 - le **script de la démonstration** (demo.txt ou demo.pdf)
 - et la **présentation orale** (presentation.pdf).

— 14-15 janvier : **Recette du projet**

La recette du projet durera environ 20 minutes par équipe. Elle sera organisée en 3 phases : une démonstration, une présentation orale et un échange avec l'enseignant.

La démonstration durera 5 minutes. Vous avez la main. Il s'agit de montrer que vos programmes répondent au cahier des charges. Cette démonstration doit être préparée. Le **script de la démonstration** décrit le déroulé de la démonstration et explique ce que vous allez montrer. Il fait parti des livrables.

La présentation dure 5 minutes également. Il s'agit de nous expliquer les principaux choix faits dans votre projet : architecture en modules, structures de données, principaux algorithmes, avancement du projet... Un support de **présentation** doit être préparé et rendu (presentation.pdf). En 5 minutes, l'ensemble du projet ne pourra pas être présenté. C'est à vous de choisir ce qu'il est pertinent de présenter pour que l'enseignant comprenne le travail réalisé par l'équipe.

L'échange dure environ 10 minutes. Il s'agit de questions/réponses, discussions, tests supplémentaires...

L'ordre de passage sera communiqué via Moodle.

- Samedi 17 janvier : Date limite pour mettre à jour le code.

Le code pourra être modifié jusqu'à cette date. Dans ce cas, il sera nécessaire de décrire ces modifications en annexe du rapport dans une rubrique « Dernières modifications ».

Attention cependant, ces modifications n'influeront qu'à la marge sur la note. Ce délai n'est pas là pour terminer le projet mais corriger de petites erreurs qui auraient été identifiées pendant la recette du projet.

Remarque : Les équipes de 2 ne traiteront pas le cache LFU.

5 Principaux critères de notation

Voici quelques uns des critères qui seront pris en compte lors de la notation du projet :

- le respect du cahier des charges,
- la qualité des raffinages,
- la facilité à comprendre la solution proposée,
- la pertinence des modules définis,
- la pertinence des sous-programmes définis,
- la qualité des sous-programmes : ils ne doivent pas être trop longs, ne pas faire trop de choses, ne pas avoir trop de structures de contrôle imbriquées. Dans ces cas, il faut envisager de découper le sous-programme !
- la bonne utilisation de la programmation par contrat (en particulier les préconditions et les invariants de type) et de la programmation défensive,
- la pertinence des tests réalisés sur les sous-programmes
- la pertinence des types utilisateurs définis,
- la pertinence de l'architecture logicielle adoptée,
- l'absence de code redondant,
- le choix des identifiants,
- les commentaires issus des raffinages,
- la bonne utilisation des commentaires,
- le respect de la solution algorithmique (les raffinages) dans le programme,
- la présentation du code (le programme doit être facile à lire et à comprendre),
- l'utilisation des structures de contrôle adéquates,
- la validité du programme,
- la robustesse du programme,
- la bonne utilisation de la mémoire dynamique (valgrind),
- le respect des contraintes imposées par PIM,
- l'utilisation régulière de Gitlab avec des messages de validation significatifs,
- les grilles d'évaluation des raffinages et la grille « qui fait quoi ? ».

Attention : Cette liste n'est pas limitative !

6 Barème indicatif pour l'évaluation

1. Raffinages (4 points)
 - respect du formalisme présenté
 - qualité des raffinages
 - pertinence
 - respectés dans le code
2. Réalisation (7 points)
 - organisation en modules
 - caractère réutilisable des modules (généricité, complétude des opérations)
 - modélisation des données (types utilisés)
 - algorithmes
 - qualité du code
3. Fonctionnement du programme (5 points)
 - couverture du sujet
 - tests réalisés (y compris unitaires)
4. Démonstration (1 points)
 - préparation de la démonstration
 - couverture du sujet
5. Présentation orale (1 points)
 - qualité de la présentation
 - couverture du sujet
6. Rapport (2 points)
 - structure générale (introduction, conclusion, plan)
 - qualité informative du rapport (raffinages, types, état d'avancement, difficultés)
 - qualité de la présentation (forme, grammaire, orthographe...)