TP1 : Algorithmes gloutons

Le but de ce TP est d'implanter des algorithmes gloutons vus en cours. On fournit une trame de code pour chaque exercice dans les fichiers `TP1Exo1.cc` *et* `TP1Exo2.cc`*, ainsi qu'un* `Makefile` *(commun aux deux exercices).*

Exercice 1. *Choix optimal de cours*

Les dates de débuts et de fin de cours sont des entiers de 1 à 100. On stocke un ensemble de n cours dans un tableau `int cours[n][2]`, où pour $i = 0, \ldots, n-1$ la variable `cours[i][0]` contient la date de début du cours i et `cours[i][1]` sa date de fin.

1. Compléter la fonction `GenererCours(int n, int cours[][2])` qui remplit le tableau `cours` en choisissant pour le cours i avec $i = 0, \ldots, n-1$ une date de début choisie au hasard entre 1 et 90 et une durée du cours choisie au hasard entre 1 et 10.
2. Compléter la fonction `TrierCours(int n, int cours[][2])` qui trie le tableau `cours` par dates de fin de cours croissantes. **Bien tester** votre algorithme sur un petit nombre de cours.
   Remarque : dans un premier temps on pourra utiliser un tri à bulles, si on le souhaite, puis implanter un tri plus rapide si le temps le permet.
3. Compléter la fonction `int ChoixCoursGlouton(int n, int cours[][2], int choix[][2])` qui implante l'algorithme ChoixCoursGlouton du cours. Le tableau `int choix[n][2]` sera intialisé à 0 et contiendra les cours choisis après l'appel de la fonction `ChoixCoursGlouton`. De plus, celle-ci renverra le nombre de ces cours choisis. **Tester et vérifier** votre algorithme. On pourra notamment essayer l'exemple `cours2` fourni dans le fichier `TP1Exo1.cc` pour obtenir l'affichage suivant :

   ```
   Test non aléatoire :
     Cours non tries : [76,78] [12,17] [13,15] [19,28] [12,20] [43,45] [44,45]
   [1,8] [68,78] [85,88]
     Cours triés par dates de fin croissantes : [1,8] [13,15] [12,17] [12,20]
   [19,28] [43,45] [44,45] [76,78] [68,78] [85,88]
     Nombre de cours choisis : 6
     Liste des cours choisis :[1,8] [13,15] [19,28] [43,45] [76,78] [85,88]
   ```

4. Écrire une fonction `Alarmes` qui permet de résoudre le problème des alarmes incendies : on veut faire sonner le moins possible d'alarmes incendies, tout en garantissant qu'au moins une alarme retentit pendant chaque cours. *Pour cette question, vous devez étudier le problème (et trouver l'algorithme glouton), déterminer les structures de données à utiliser, implanter votre solution, et* ***effectuer des tests****. Aucune trame de code n'est fournie, même pour les tests.*
   Comparer le nombre d'alarmes minimales et le nombre maximal de cours trouvé par ChoixCoursGlouton. Expliquer (voire démontrer !) ce que vous observez.

Exercice 2. SETCOVER *en dimension 2*

1. Compiler et exécuter le fichier fourni, comprendre son fonctionnement (notamment le remplissage du tableau `coordMaisons` et observer le fichier `Maison.ps` produit.
2. Implanter la fonction `bool Couvre(int i, int j, const int coordMaisons[][2])` qui retourne vrai si, et seulement si, les maisons i et j se situent à moins de `dcouv` l'une de l'autre. *La valeur de la variable* `dcouv` *est fixée par* `#define dcouv 100` *dans le fichier* `TP1Exo2.cc`.
3. Implanter la fonction `int ChoixProchaineMaison(int n, const int coordMaisons[][2], const int dejaCouvert` qui retourne l'indice de la prochaine maison sur laquelle placer un émetteur. Le tableau `dejaCouverte[n+1]` contient pour chaque i de 0 à $n-1$ un marqueur 0 ou 1 pour indiquer si la $i^{\text{ème}}$ maison est déjà couverte. La valeur `dejaCouverte[n]` est quant à elle le nombre de maisons déjà couvertes.
4. Implanter la fonction `void ChoixEmetteurs(int n, const int coordMaisons[][2], int emetteurs[])` qui trouve un ensemble d'émetteurs qui couvre toutes les maisons, avec l'algorithme vu en cours. *Pour tester votre fonction, appeler le programme produit avec* `./TP1Exo2 N` *où* `N` *est un entier, pour générer N maisons aléatoires et obtenir une couverture. Les fichiers* `Maisons.ps` *et* `Emetteurs.ps` *produits doivent être semblables à ceux de la figure 1.*

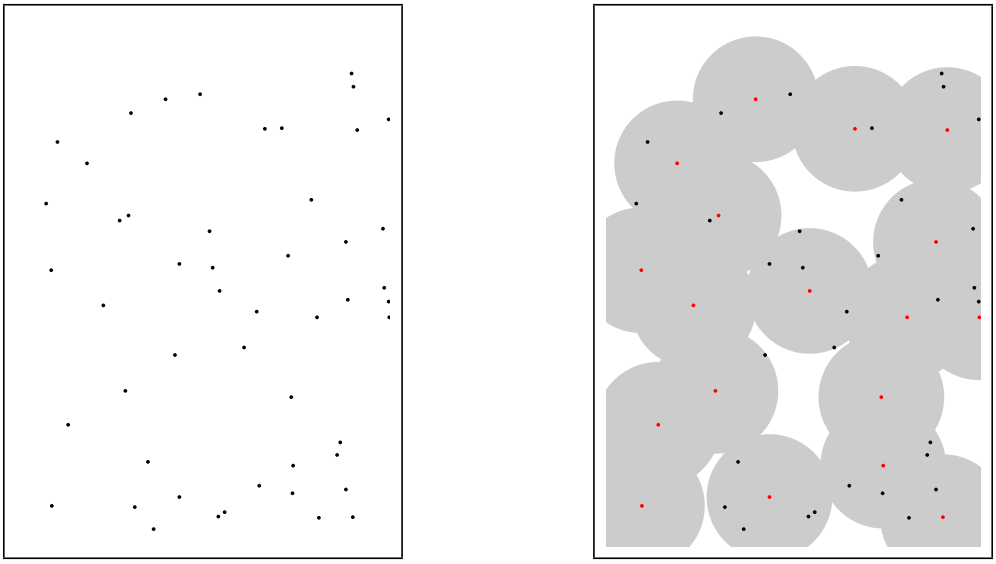

FIGURE 1 – À gauche, les maisons dans le plan, à droite les émetteurs placés (en rouge) et leur couverture (en gris).

5. Implanter la fonction `ChoixEmetteursOpt` qui effectue le même travail que `ChoixEmetteurs` mais avec un algorithme qui renvoie une solution optimale. *Indications :*
   On pourra effectuer une recherche exhaustive : en s'aidant de la fonction `suivant` *fournie, parcourir toutes les possibilités de placement d'émetteurs, et garder la meilleure.*
   Ne pas hésiter à écrire une ou plusieurs fonctions auxiliaires pour faciliter la programmation !
6. Rechercher un exemple où l'algorithme glouton donne un résultat non optimal.
7. *(bonus)* Améliorer la fonction `ChoixEmetteursOpt` afin de la rendre **la plus rapide possible**.