- Fiche de TP2 : Algorithmes gloutons -

Une trame de code en C est disponible pour les exercices.

- Choix optimal d'activités -

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

- Exercice 1 - Entrées/sorties sur un tableau d'activités -

Écrire une fonction `GenererActivite(int n, int activites[][2])` qui remplit le tableau `activites` en choisissant pour l'activité i avec $i = 0, \ldots, n-1$ une date de début choisie au hasard entre 1 et 90 et une durée de l'activité choisie au hasard entre 1 et 10.

- Exercice 2 - Tri des activités -

Utiliser un algorithme de tri pour trier le le tableau `activites` par dates de fin d'activités croissantes. Tester bien votre algorithme sur un petit nombre d'activités.
Remarque : dans un premier temps on pourra utiliser un tri à bulles, si on le souhaite, puis implémenter un tri plus rapide si le temps le permet.

- Exercice 3 - Algorithme glouton de choix d'activités -

Écrire la fonction `int ChoixGloutonActivites(int n, int activites[][2], int choix[][2])` qui implémente l'algorithme CHOIX-GLOUTON-D-ACTIVITES du cours. Le tableau `int choix[n][2]` sera intialisé à 0 et contiendra les activités choisies après l'appel de la fonction `ChoixGloutonActivites`. De plus, celle-ci reverra le nombre de ces activités choisies. Tester et vérifier votre algorithme. On pourra notamment essayer l'exemple `activites2` proposé pour obtenir :

```
Entrer le nombre d'activités: 10
Activites non tries:
[76,78] [12,17] [13,15] [19,28] [12,20] [43,45] [44,45] [1,8] [68,78] [85,88]
Activites tries 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 d'activites choisies: 6
Liste de ces activites:
[1,8] [13,15] [19,28] [43,45] [76,78] [85,88]
```

- SET-COVER en dimension 2 -

- Exercice 4 - Affichage et distance -

Compiler et exécuter le fichier fourni, comprendre son fonctionnement (notamment le remplissage du tableau `coordMaison` et observer le fichier *Maison.ps* produit. Implémenter la petite fonction `bool Couvre(int i, int j, int coordMaison[][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` étant fixée à 100 dans le fichier *AffichageMaison.cc*.

- Exercice 5 - Implémentation du choix glouton -

Compiler et exécuter le fichier fourni, comprendre son fonctionnement (notamment le remplissage du tableau `coordMaison`) et observer le fichier *Maison.ps* produit.

Implémenter les fonctions `int ChoixMaison(int n, int coordMaison[][2],int dejaCouverte[])`, qui retourne notamment l'indice de la maison suivante à choisir, et `void MettreAJour(int n, int indiceNew, int emetteur[], int dejaCouverte[], int coordMaison[][2])`.

- Exercice 6 - Résultats-

Décommenter la dernière partie du code et visualiser le fichier *Emetteur.ps* créé. Un exemple de ce qui est attendu est donné ci-dessous.
En fonction du nombre d'émetteurs choisis, faire afficher une borne inférieure sur le nombre optimal de terminaux à choisir.

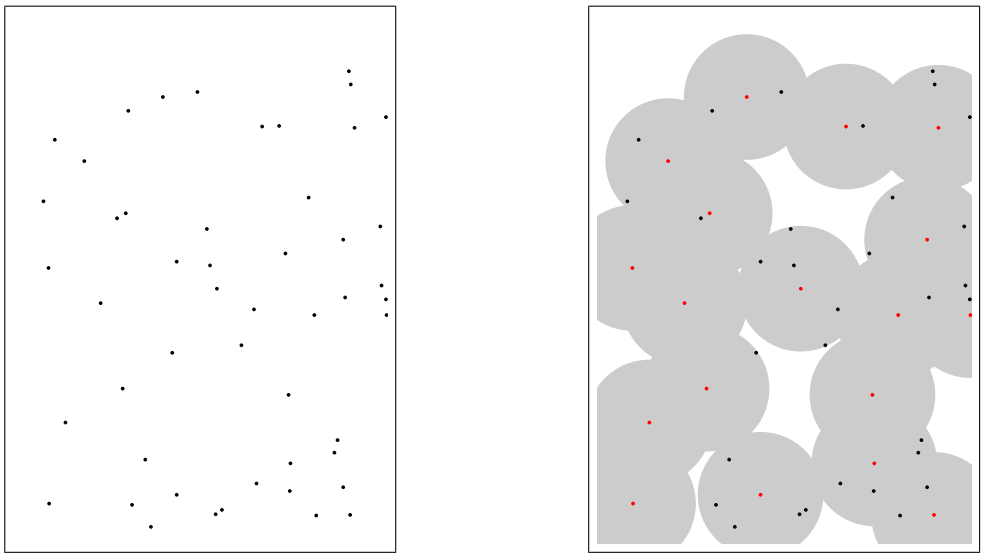

FIGURE 1 – À gauche, les maisons dans le plan, à droite les émetteurs placés (en rouge) et leur couverture (en gris).