Il s'agit de la fiche professeur, voir également le fichier PRESENTATION.pdf
Groupe B2G1A : R.Amrouche - O.Hupé - P.Mercier - G.Pons

Recherche dichotomique

Objectif :
Le programme au BO indique dans la partie contenu : « recherche dichotomique dans un tableau trié ». Il se peut que la notion de dichotomie ait été vue en mathématiques en classe de seconde (exemple d'algorithme : pour une fonction dont le tableau de variations est donné, algorithmes d'approximation numérique d'un extremum (balayage, dichotomie)) mais cela ne suffit bien sûr pas à la considérer comme acquise.

L'algorithme est intéressant par lui-même mais il va servir de prétexte à découvrir d'autres notions largement utilisées et qui seront approfondies par la suite :
- étude du coût d'un algorithme,
- preuve de la terminaison d'un programme à l'aide de la notion de variant,
- preuve de la correction d'un algorithme à l'aide la notion d'invariant.

Malgré l'ordre du programme, il peut être opportun de faire cette séance avant les tris puisqu'elle montrera que la recherche est bien plus rapide sur un tableau trié* pour peu qu'il soit conséquent.
** Il faut néanmoins tenir compte du coût du tri du tableau et donc avoir besoin de faire de nombreuses recherches.*

Enfin, même si la notion de récursivité n'est pas au programme de première, ce peut être l'occasion d'aborder cette notion qui sera vue en terminale.

Modalités de mise en œuvre :
Environnement python,
Dictionnaires,
(Variante : cartes avec des mots).

Pré-requis :
Boucles, conditions, tableaux en algorithmique
Implémentation en langage python
Bibliothèques spécifiques à python étudiées dans d'autres disciplines (matplotlib).

Compétences :
Anticiper (terminaison d'un algorithme),
Evaluer (notion de complexité),
Décomposer (simplifier le problème)
Abstraire (concevoir un algorithme)

Minutage :
Voir page 4 du fichier présentation

1. **Mise en situation (activité débranchée) – 10 min**

Les élèves cherchent la page où un mot est présent sur un dictionnaire selon diverses stratégies.
(Pourquoi pas avec un jeu de cartes, avec un mot sur chaque carte. Discussion sur les stratégies avec les élèves : dans l'ordre en retournant les cartes, au hasard, dans l'ordre inverse. La notion de tri doit apparaître.)
On recommence avec les mots dans le dictionnaire.
- Le 1er groupe recherche le mot en commençant par la 1ère lettre de l'alphabet (stratégie linéaire)

- Le 2nd groupe partage le dictionnaire en 2 parties égales, élimine la partie où le mot n'est pas, et recommence (stratégie dichotomique)
- Pourquoi ne pas envisager une méthode aléatoire si les élèves y pensent.

Résultats :
La stratégie dichotomique gagne quasiment chaque fois (sauf si le mot commence par « a »). Pourquoi ?
Les élèves doivent montrer que la stratégie linéaire oblige à évaluer chaque mot alors que la dichotomie évite de passer par tous les mots.
D'autres stratégies possibles :
Les élèves peuvent nous proposer le hasard qui peut être plus rapide que la stratégie linéaire. Problème : les valeurs peuvent revenir plusieurs fois.
Les élèves proposent de commencer directement à la page de la lettre correspondante aux mots ce qui revient a une pseudo-recherche dichotomique. (on a éliminé d'emblée toutes les pages ne commençant pas par l'initiale du mot.(-→ on sort du contexte de l'activité).

2. Le nombre secret – Algorithme papier et implémentation python - 45 min

Afin de quantifier plus facilement les programmes, on passe à une liste de nombres ordonnée à la place des mots, ce qui facilite l'implémentation en Python (pas de dictionnaire à créer).

Question : vous devez réaliser un algorithme (sur papier) permettant à l'ordinateur de deviner le nombre compris entre 1 et n que vous avez choisi. Vous pouvez utiliser l'une ou l'autre des stratégies vues plus haut (ou les deux si vous êtes rapides).
Les élèves passent sur les ordinateurs pour une implémentation en Python.

Bilan : solutions données afin d'avoir un algorithme commun pour la classe.

3. Exploitation des résultats - Notion de complexité – 30 min

Le but n'est pas d'être exhaustif mais d'introduire la notion de complexité avec un exercice simple.
On complètera cette notion de complexité(n^2…) lors des prochaines séances sur les tris.

On va chercher avec quelle vitesse on obtient les résultats. Pour cela ouvrir lineairedicho.py
- exécuter successivement les fonctions : lineaire_print, dicho_print et observer le défilement,
- exécuter successivement les fonctions : lineaire_etapes, dicho_etapes et et comparer le nombre de boucles exécutées
- exécuter successivement les fonctions : lineaire_time, dicho_time et comparer les temps d'exécution,
- exécuter la fonction : graphes, qu'observez-vous ?

Ces courbes sont réalisées dans le pire cas, si les élèves posent la question, on peut distinguer le meilleur cas, du cas moyen du pire cas.

Bilan :

Les algorithmes de recherche n'ont pas tous la même efficacité on parle de complexité. Certains sont plus efficaces que d'autres comme la recherche dichotomique consistant à rechercher l'élément dans un sous-ensemble réduit de moitié à chaque étape
Grâce à la méthode par dichotomie, par exemple pour un ensemble de 10000 éléments, la machine trouvera à coup sûr le nombre secret en 14 tests maximum(2^{14}=16384).

Ce nombre de tests qui dépend de la taille n de l'ensemble est le nombre de fois qu'il faut diviser n par 2 pour obtenir un nombre inférieur ou égal à 1. (résolution de l'équation et notation $\log_2(n)$ en Terminale)

4. Terminaison et correction des algorithmes – 1h

Ouvrir le programme « test_terminaison » : ce programme vous parait-il satisfaisant ?

On peut supposer que les élèves vont dire que oui, éventuellement faire un test et se réjouir d'avoir trouvé un bon algorithme pour répondre à notre problème de deviner un nombre.

Mais le travail n'est pas fini ! En effet, deux questions importantes se posent :

- notre algorithme va-t-il se terminer dans tous les cas ?

- notre algorithme est-il correct, c'est-à-dire qu'il va bien nous donner le nombre secret ?

Il faut que les élèves fassent bien la différence entre un algorithme correct et un programme qui compile.

Tout ce que l'on risque c'est de perdre au jeu ou d'avoir un programme qui ne s'arrête pas : la question peut sembler secondaire dans notre cas. Mais changeons de point de vue : le programme qui envoie les instructions de pilotage automatique d'un airbus va-t-il se terminer et dire à l'avion ce qu'il faut faire ? Et va-t-il lui donner les bonnes instructions ?

Il s'agit de faire comprendre l'importance de ces notions à travers des exemples concrets.

ě Pourquoi un programme ne se terminerait-il pas ?

ě Que proposeriez-vous pour répondre à ces questions ?

On peut supposer que les élèves imaginent de réaliser des tests. Les laisser faire…

On peut s'apercevoir que le cas 10000 ne trouve pas de solution car il boucle indéfiniment…mais il n'est pas du tout sûr que les élèves aient testé ce cas. Le notion de test doit être complétée.

Montrer la terminaison de la recherche dichotomique à l'aide d'un variant de boucle :.

Le problème de non terminaison ne peut venir ici que de la boucle. Il n'y a pas de difficulté avec la boucle « pour » qui s'effectue un nombre fini de fois. Mais pour la boucle « tant que », il est nécessaire d'étudier sa condition de sortie.

Pour démontrer la terminaison d'une boucle, il faut par exemple trouver une quantité, fonction des paramètres de la boucle, qui est un entier positif et qui décroît strictement à chaque itération. Cette quantité est appelée **variant de boucle**.

ě Quel pourrait être ici le variant à tester ?

Il existe de nombreuses variantes en fonction de l'algorithme réalisé par les élèves.
Dans notre algorithme, il faut que m soit égal au nombre cherché pour sortir de la boucle, nombre qui appartient forcément au tableau. Si le tableau ne contient qu'un nombre, c'est donc forcément celui-là.
On va donc montrer que le nombre d'éléments du tableau diminue jusqu'à la taille 1.
Taille du tableau lors de l'itération numéro n : L_n
A l'étape n, $L_n = fin_n - début_n$
Si m = nbre_secret, on sort de la boucle et l'algorithme se termine.
Si m > nbre_secret, $fin_{n+1} \leftarrow m$, la taille du tableau diminue (divisée par 2 à l'entier près), puisqu'on conserve la première moitié.
Si m < nbre_secret, $debut_{n+1} \leftarrow m$, la taille du tableau diminue (divisée par 2) puisqu'on conserve la deuxième moitié.
Donc à l'étape n+1, $L_{n+1} < L_n$,
Nous aurons au pire un tableau de taille 1 donc trouvé le nombre secret et la boucle « tant que » s'arrête.
Par conséquent l'algorithme se termine.

Montrer la correction de la recherche dichotomique à l'aide d'un invariant de boucle :

Pour démontrer qu'un algorithme produit le résultat attendu, on utilise un invariant de boucle, c'est-à-dire une propriété qui est vérifiée avant d'entrer dans la boucle. Si cette propriété est vérifiée avant une itération, elle est vérifiée après celle-ci. Lorsqu'elle est vérifiée en sortie de boucle, elle permet (combinée avec la condition de sortie de boucle) d'en déduire que le programme produit le résultat attendu.

La démonstration va se faire en 3 étapes :

- On montre que l'invariant de boucle est vrai avant la première itération de la boucle, c'est la phase d'initialisation.

- On montre que si l'invariant de boucle est vrai avant une itération de la boucle, il le reste avant l'itération suivante, c'est la phase de conservation.

- Enfin, on montre qu'après les passages dans la boucle, l'invariant fournit une propriété utile qui aide à montrer la correction de l'algorithme.

ĕQuel invariant de boucle pourrait-on rechercher dans l'algorithme de dichotomie ?

Pour cela, montrons que la propriété P_i suivante est un invariant de la boucle « tant que ».
P_i : « Le nombre recherché appartient au tableau délimité par les indices début et fin avant l'entrée dans la boucle.

Initialisation
Avant la boucle « tant que », le tableau de nombres est complet et on s'est assuré que le nombre secret y figurait donc il est évident que P_1 est vraie.

Conservation :
Supposons que P_i est vraie pour un entier naturel i : avant le tour de boucle i, si le nombre est dans le tableau, il se trouve entre les positions debut et fin.
Le nombre secret est comparé avec la valeur en position médiane (début + fin) /2.
S'il est strictement inférieur à milieu, fin devient milieu et le nombre secret appartient donc toujours au nouveau sous-tableau.
S'il est supérieur ou égal à milieu, début devient milieu et le nombre secret appartient donc également toujours au nouveau sous-tableau.
Dans tous les cas, si le nombre appartient au tableau initial, il appartiendra nécessairement au sous tableau.
Ainsi avant le tour de boucle i + 1, la propriété P_{i+1} sera vraie, c'est donc une propriété qui se conserve, un invariant de boucle.

Terminaison :
Nous avons précédemment vu que la taille du tableau allait diminuer, jusqu'à atteindre 1. Comme on vient de montrer qu'à chaque tour de boucle, le nombre mystère est dans le tableau, soit il a été trouvé lors d'un tour de boucle, soit la liste sera de taille 1, contiendra le nombre secret qui sera le dernier de la liste et l'algorithme est correct.

5. Exercices complémentaires

Préparer la notion de récursivité vue en terminale et programmer la dichotomie en récursif. Voir présentation.

Variante : voir dossier activité Bloc2

Travail sur convertisseur analogique/numérique

Il serait souhaitable ici ou plus loin de faire tester sur le cas de données nombreuses (base de données).

Exemple d'exercices :

Terminaison : extrait sujet 0

QCM 7.1

A désignant un entier, lequel des codes suivants ne termine pas ?

Réponses possibles

A

```
i = A + 1
while i < A:
   i = i-1
```

B

```
i = A + 1
while i < A:
   i = i + 1
```

C

```
i = A - 1
while i < A:
   i = i - 1
```

D

```
i = A - 1
while i < A:
   i = i + 1
```

Exercice :

Prouvez que cet algorithme fonctionne.

```
# n est un entier
def preuve1 (n) :
    b = 0
    for i in range (0,n) :
        b = b + 2
    return b

# b vaut le double de a
```