Développement : analyse d'un algorithme de dichotomie sur les listes (leçons 12 et 13)

Problème : Recherche de position d'insertion

Étant donnés une liste strictement croissante de n nombres entiers $S = (a_i)_{0 \leq i < n}$ et un nombre entier b, déterminer le plus grand indice $0 < k \leq n$ tel que $b > a_{k-1}$ (ou $k = 0$ si $b \leq a_0$).

Algorithme

1. On pose $g = 0$, $d = n - 1$ les bornes initiales de l'intervalle de recherche.
2. Si $d < g$, la réponse au problème est l'indice g.
3. Sinon, en posant $m = \lfloor (d + g)/2 \rfloor$:
    1. Si $b \leq a_m$, on pose $d = m - 1$ et on reprend au point 2.
    2. Sinon, on pose $g = m + 1$ et on reprend au point 2.

Implémentations

Implémentation itérative en Python :

```
def indice_insertion(lst, b):
    g, d = 0, len(lst)
    while g < d:
        m = (g + d) // 2
        if b <= lst[m]:
            d = m
        else:
            g = m+1
    return g
```

Implémentation récursive :

```
def indice_insertion(lst, b):
    # fonction auxiliaire (récursive)
    def aux(g, d):
        if g == d:
            return g
        m = (g + d) // 2
        if b <= lst[m]:
            return aux(g, m)
        else:
            return aux(m+1, d)

    # appel principal
    return aux(0, len(lst))
```

Preuve de terminaison

Considérons les valeurs successives de g et d à chaque exécution de l'étape 2 de l'algorithme sur une instance $(S = (a_i)_{0 \leq i < n}, b)$ donnée. Les valeurs successives de $d - g$ forment une suite strictement décroissante d'entiers naturels, qui ne peut donc être infinie, ce qui implique la terminaison de l'algorithme. En effet, en supposant $g \neq d$ (le cas où $g = d$ est trivial), notons g' et d' les nouvelles valeurs de g et d calculées dans le corps de la boucle. Comme $g < d$ on a $g \leq m < d$. Selon la valeur de a_m on a soit $g' = m + 1 > g$ et $d' = d$, soit $g' = g$ et $d' = m < d$, ce qui implique que $0 \leq d' - g' < d - g$.

Preuve de correction

Considérons une instance $(S = (a_i)_{0 \leq i < n}, b)$ du problème et appelons k le résultat associé. Soit ℓ le nombre de passages par l'étape 2 de l'algorithme (qui est fini comme nous l'avons démontré), posons $(g_i)_{0 \leq i \leq \ell}$ et $(d_i)_{0 \leq i \leq \ell}$ les suites de valeurs respectives de g et d à chaque exécution de cette étape. Nous allons établir par récurrence sur i la propriété $P(i) : g_i \leq k \leq d_i$.

Premièrement, $P(0)$ est vraie car $g_0 = 0$, $d_0 = n$ et par définition $0 \leq k \leq n$. Supposons à présent $P(i)$ pour tout $0 \leq i < \ell$ et montrons $P(i + 1)$. Par hypothèse de récurrence, $g_i \leq k \leq d_i$. On calcule $m = \lfloor (g + d)/2 \rfloor$. Deux cas se présentent :

1. Si $b \leq a_m$, comme S est croissante et par définition de k on a nécessairement $k \leq m$:

   - si $m = 0$, nécessairement $k = 0$,
   - si $m > 0$ et $a_{m-1} < a_m$, nécessairement $k = m$,
   - si $m > 0$ et $a_{m-1} = a_m$, nécessairement $k < m$.

   Or $d_{i+1} = m$ et $g_{i+1} = g_i$, donc $g_{i+1} = g_i \leq k \leq m = d_{i+1}$.

2. Si $b > a_m$, nécessairement $k > m$. Or $g_{i+1} = m + 1$ et $d_{i+1} = d_i$, donc $g_{i+1} = m + 1 \leq k \leq d_i = d_{i+1}$.

On déduit de ce qui précède que $P(i)$ est vraie pour tout $0 \leq i \leq \ell$, et en particulier que $g_\ell \leq k \leq d_\ell$, ce qui conclut la preuve puisque $g_\ell = d_\ell$.

Analyse de complexité

On souhaite démontrer que l'exécution de l'algorithme A sur toute instance (S, b), avec S de longueur n, effectue dans tous les cas de l'ordre de $\log_2 n$ opérations élémentaires.

Plaçons-nous à l'étape 2 de l'algorithme A, pour des valeurs quelconques de g et d, $0 \leq g \leq d \leq n$. Nous allons montrer par récurrence complète sur $d - g$ que l'algorithme fournit un résultat en un nombre d'étapes ℓ tel que $\lfloor x \rfloor \leq \ell \leq \lceil x \rceil$, avec $x = \log_2(d - g + 1)$.

Si $d = g$, on vérifie que $\ell = \log_2(1) = 0$. Sinon, supposons la propriété vraie à tout rang strictement inférieur à $d - g$. Comme $g \neq d$ la condition de l'étape 2 n'est pas vérifiée, on effectue donc une nouvelle itération. Appelons respectivement g' et d' les valeurs de g et d au passage suivant par l'étape 2. On distingue deux cas :

- Si $d - g + 1 = 2^p$ pour un certain $p \geq 0$, un simple calcul montre que $d' - g' + 1 = 2^{p-1}$ dans tous les cas. Par hypothèse de récurrence, on a donc $\lfloor \log_2(2^{p-1}) \rfloor \leq \ell - 1 \leq \lceil \log_2(2^{p-1}) \rceil$, soit $\ell = p$.

- Sinon, soit p l'unique entier tel que $2^{p-1} < d - g + 1 < 2^p$. On montre facilement que $(d-g)/2 \leq d' - g' + 1 \leq (d-g)/2 + 1$, ce qui implique que $2^{p-2} \leq d' - g' + 1 \leq 2^{p-1}$. Par hypothèse de récurrence, on obtient $\lfloor \log_2(2^{p-2}) \rfloor \leq \ell - 1 \leq \lceil \log_2(2^{p-1}) \rceil$, soit enfin $p - 1 \leq \ell \leq p$.

La propriété est donc vraie pour toutes valeurs initiales $0 \leq g \leq d \leq n$. En prenant $g = 0$ et $d = n$, on en déduit que l'exécution complète de l'algorithme effectue $\lceil \log_2(n+1) \rceil$ accès aux éléments de S dans le pire cas. Dans le cas particulier où $n = 2^p - 1$ pour un certain p, on note que l'algorithme effectue exactement p comparaisons dans tous les cas.

Optimalité

On raisonne maintenant sur la classe de *tous* les algorithmes permettant de résoudre le problème, du moins tous ceux qui s'inscrivent dans le modèle de calcul utilisé.

L'ensemble des exécutions de tout algorithme de ce type sur une suite S à n éléments peut être représenté par un arbre binaire, couramment appelé *arbre de comparaisons*, dont chaque noeud interne correspond à la comparaison d'un élément de S et d'un autre nombre et dont chaque feuille représente un résultat possible. Dans un tel arbre, la longueur de chaque branche représente le nombre de comparaisons effectuées par l'algorithme sur une instance. La longueur des plus longues branches correspond donc à la complexité au pire (en nombre de comparaisons) sur les suites de taille n.

On appelle *hauteur* d'un arbre la longueur (le nombre d'arêtes) de sa branche la plus longue. Un résultat facile à démontrer est que tout arbre binaire possédant m feuilles est nécessairement de hauteur au moins $\lceil \log_2 m \rceil$. Intuitivement, cela est dû au fait qu'un arbre binaire de hauteur h possède au plus 2^h feuilles.

Dans le cas présent, l'arbre de comparaison de tout algorithme résolvant ce problème possède au moins $n+1$ feuilles, chacune représentant un résultat possible k entre 0 et n. Par conséquent, une de ses branches est nécessairement de longueur supérieure ou égale à $\lceil \log_2(n+1) \rceil$. Cela permet d'en déduire que *tout* algorithme résolvant ce problème *doit* effectuer au moins $\lceil \log_2(n+1) \rceil$ comparaisons dans le pire cas. Puisque l'algorithme A, comme nous l'avons vu précédemment, effectue *au plus* $\lceil \log_2(n+1) \rceil$ accès à S, il est dit *optimal*. C'est une caractéristique fondamentale de l'algorithme A et de ses variantes, qui en fait un cas d'école d'analyse d'algorithmes.