IFT2125 automne 2016 - Devoir 3

Philippe Caron

16 novembre 2016

1 Mergesort 5

Commençons par la fonction **merge5()**. Avant la boucle, l'ajout des sentinelles et la détermination de la longueur sont faits en temps constant, puis dans la boucle, le programme fait toujours 5 comparaison avant d'avoir déterminer le bon cas, où il fait deux opération, disons donc environ 7 opération **nlen** fois (voir code), où **nlen** correspond à la somme des longueur de chaque tableau. Soit **n** le nombre d'élément par tableau, on peut donc dire que :

$$t_{merge5}(n) = k + 5n \cdot 7 = k + 35n \in O(n)$$

La fonction **mergesort5()** quant à elle est également constante quand **n** ≤ 1, sinon lorsqu'elle divise la liste en sous liste, on peut imaginer que chaque séparation est $O(n)$ ainsi que chaque appel à **merge5()**, on a donc 10 appel de $O(n)$ ce qui est aussi dans $O(n)$. Il ne nous reste qu'à savoir combien de fois **mergesort5()** est appelée. Ceci correspond essentiellement au temps que cela prend pour que le paramètre **n** vaille 0. Puisqu'on divise en 5 à chaque fois, on trouve le nombre avec $\log_5 n$ et donc on sait que l'algorithme est $O(n \log n)$.

$$t_{mergesort5}(n) = \log_5 n \cdot O(n) \in O(n \log n)$$

2 Roule

(a) arbre d'appel et fonction

```
roule(a, 143)
   roule(a, 136)
      roule(a, 128)
         roule(a, 8)
            roule(a, 1) = a
            roule(a, 2) = a^2
         roule(a^2, 16)
            roule(a^2, 4)
               roule(a^2, 2) = a^4
               roule(a^4, 2) = a^8
            roule(a^8, 4)
               roule(a^8, 2) = a^16
               roule(a^16, 2) = a^32
      roule(a, 8)
         roule(a, 1) = a
         roule(a, 2) = a^2
   [a^32 * a^2 = a^34]
   roule(a, 7)
      roule(a, 4)
         roule(a, 2) = a^2
         roule(a^2, 2) = a^4
      roule(a, 3)
         roule(a, 2) = a^2
         roule(a, 1) = a
      [a^2 * a = a^3]
   [a^4 * a^3 = a^7]
[a^34 * a^7 = a^41]
```

On conclu donc que $roule(a, 143) = a^{41}$

(b) récurrence

Le nombre d'appel peut-être donné par la fonction suivante :

$$t(k) = \begin{cases} 1 & \text{si } k = 0 \\ 2 \cdot t(k-1) & \text{sinon} \end{cases} \tag{1}$$

(c) fonction

On peut définir $t(k)$ explicitement :

$$t(k) = 2^k \tag{2}$$

Il est donc certain que si on dit $f(n) = 2^n$, alors $t(k) \in \Theta f(n)$.

3 RSA

On connait les valeurs suivantes :

$$\begin{aligned} p &= 311 \\ q &= 47 \\ z &= 14617 \\ n &= 13 \\ m &= 123 \end{aligned}$$

On trouve ϕ :

$$\phi = (p-1)(q-1) = 310 \times 46 = 14260$$

puis on trouve s avec une simple boucle en python :

```
>>> for x in range(14617):
...     if (13 * x) % 14260 == 1:
...             print(x)
...
1097
```

ensuite on calcule c :

$$c = m^n \mod z = 123^{13} \mod 14617 = 1925$$

puis on confirme :

$$c^s \mod z = 1925^{1097} \mod 14617 = 123$$

Le message est restauré correctement les valeurs trouvées sont donc les bonnes.

4 Distance minimale

Voici l'algorithme :

```
FindMin(P, n) :
    min ← ∞
    for i = 0 to n :
        for j = i + 1 to n :
            if {distance entre P[i] et P[j]} < min then :
                min ← {distance entre P[i] et P[j]}
    return min

CenterMin(P, d) :
    {trie selon y le tableau de points P}
    min ← d
    for i = 0 to len(P) :
        for j = i + 1 until len(P) or until {distance entre P[i] et P[j]} < min :
            if {distance entre P[i] et P[j]} < min then :
                min ← {distance entre P[i] et P[j]}
    return min

MinDistRec(P, n) :
    if n ≤ 3 then :
        return FindMin(P, n)
    else :
        P1 ← P[0 : n/2]
        P2 ← P[n/2 : n]
        d1 = MinDistRec(P1, n/2)
        d2 = MinDistRec(P2, n − n/2)
        d = min(d1, d2)
        {conserve les points de P à l'intérieur d'une distance ± d du centre : (P[n/2 − 1] + P[n/2])/2}
        return min(d, CenterMin(P[−d : +d], d))

MinDist(P) :
    {trie selon x le tableau de points P}
    return MinDistRec(P, len(P))
```

On peut voir que cet algorithme est $\notin \Omega(n^2)$ en analysant chacune des fonctions séparément. D'abord la première (FindMin) est $O(n^2)$, mais elle n'est jamais utilisée avec un $n > 3$ on peut donc considérer qu'elle est effectuée en temps constant. La seconde (CenterMin) est très similaire à FindMin alors on pourrait croire qu'elle est aussi $O(n^2)$ or c'est faux puisqu'il est prouvé que la seconde boucle ne dépasse jamais 6 itérations.Le trie est donc l'opération la plus demandante avec $O(n \log n)$. Similairement, la fonction principale demande un trie elle aussi que l'on peut considérer $O(n \log n)$. L'ordre de l'algorithme sera donc déterminé par le nombre de fois que MinDistRec est exécutée, et puisqu'on divise par deux à chaque fois, on sait que c'est $O(\log n)$.

L'algorithme final est donc $O(n(\log n)^2)$, ce qui est plus petit que $O(n^2)$.

```python
import math

def merge5(S, T, U, V, W):
   R = list()
   s = t = u = v = w = 0
   nlen = len(S) + len(T) + len(U) + len(V) + len(W)
   S.append(float("inf"))
   T.append(float("inf"))
   U.append(float("inf"))
   V.append(float("inf"))
   W.append(float("inf"))
   for x in range(nlen):
      if S[s] < T[t]:
         if S[s] < U[u]:
            if S[s] < V[v]:
               if S[s] < W[w]:
                  R.append(S[s])
                  s += 1
               else:
                  R.append(W[w])
                  w += 1
            else:
               if V[v] < W[w]:
                  R.append(V[v])
                  v += 1
               else:
                  R.append(W[w])
                  w += 1
         else:
            if U[u] < V[v]:
               if U[u] < W[w]:
                  R.append(U[u])
                  u += 1
               else:
                  R.append(W[w])
                  w += 1
            else:
               if V[v] < W[w]:
                  R.append(V[v])
                  v += 1
               else:
                  R.append(W[w])
                  w += 1
      else:
         if T[t] < U[u]:
            if T[t] < V[v]:
               if T[t] < W[w]:
                  R.append(T[t])
                  t += 1
               else:
                  R.append(W[w])
                  w += 1
            else:
               if V[v] < W[w]:
                  R.append(V[v])
                  v += 1
               else:
                  R.append(W[w])
                  w += 1
         else:
            if U[u] < V[v]:
```

```python
                if U[u] < W[w]:
                    R.append(U[u])
                    u += 1
                else:
                    R.append(W[w])
                    w += 1
            else:
                if V[v] < W[w]:
                    R.append(V[v])
                    v += 1
                else:
                    R.append(W[w])
                    w += 1
    return R


def mergesort5(A, n):
    if n <= 1:
        return A
    else:
        p = 0
        gap = math.ceil(n / 5)
        B = A[p : p + gap] ; p += gap
        C = A[p : p + gap] ; p += gap
        D = A[p : p + gap] ; p += gap
        E = A[p : p + gap] ; p += gap
        F = A[p : p + gap]
        return merge5(
            mergesort5(B, len(B)),
            mergesort5(C, len(C)),
            mergesort5(D, len(D)),
            mergesort5(E, len(E)),
            mergesort5(F, len(F)))


# TEST
a = [11, 3, 4, 7, 2, 9, 10, 8, 1, 5, 6]
mergesort5(a, len(a));
```