1er semestre — 2020-CMS-ICS — LUCAS JUNG

I. Python 3 - Programming

Base Types: int 12 0 -4, float 9,23 0,0 -1,7e-6 (x10^-6), bool True False, str "Hi\nYou" 'L\'' (line break, escape)

Container Types: list [1,5,"x"], tuple (1,5) "x", 1.5, 11 ("mot",) (ordered), dict {"key":"value"} {} ({k:v}), dict(a=3, b=2, k="v") {1:3, 3:"t", 3.14:"n"}, set {"k1","k2"} set() (immutable ← frozenset)

Identifiers: ^[a-zA-Z_][a-zA-Z0-9_]* a toto x7 y-max Big_H ~~8y and for~~

Assignment: x=1,2+sin(y) a=b=c=0 y,x=-1,9.2 a,b=b,a (swap values) a,*b=seq *a,b=seq x+=3 (x=x+3) (-=, *=, /=, //=, ...) x=None del x

Conversions: int("15")→15 int(15,7)→15, float("11,5e3")→11500.0, round(1.56,1)→1.5, round(1.57)→2 (int)

bool(x)→False, True (null, empty, None, False), str(x)→"..." (representation), chr(64)→'@', ord('@')→64, list("abc")→['a','b','c'], dict([(3,"th"),(1,"on")])→{1:'on', 3:'th'}, set(["on","tw"])→{'on','tw'}, "w i h".split()→['w','i','h'] (␣ default)

"1,2,3".split(",")→["1","2","3"], ":".join(["hi","12"])→"hi:12" (any char)

Sequence Container Indexing: (lst, tpl, str): lst=[a,b,c,d,e], len(lst)→5, lst[0] (first), lst[-1] (last)

index → -5 -4 -3 -2 -1 / 0 1 2 3 4; slice → 0 1 2 3 4 5 / -5 -4 -3 -2 -1

[start slice : end slice : step] (sub-sequence), lst[1:-1]→[b,c,d] lst[::-1]→[e,d,c,b,a] lst[::2]→[a,c,e] lst[:] (shallow copy)

Maths, Logic: + - * / // % ** (Operators; % quotient, reste; a^b), a and b a or b not a True False

Conditional Statement: if age<=18: ... elif age>65: ... else: ..., if bool(x)==True: ⇔ if x: (False: not x)

Exceptions: raise exce., try: normal except Exception as e: catch err. else: if no err. finally: all cases (before return)

Conditional Loop: while cond.: statement block (as long as cond. True), pass continue (next iteration) break (exit), else: normal exit

Iterative Loop: for var. in seq.: (for each item in container, on keys if dict.), for i,v in enumerate(lst):, range(start, end, step) (/range(len(seq.)), 0 default, 1) range(5)→0,1,2,3,4 range(2,12,3)→2,5,8,11 range(20,5,-5)→20,15,10

Display, Input: print(f"Sum:{1+2}\"",1,f"{1+2=}", sep="␣", end="\n"), s=input("Hey!") ⇔ input(prompt="...") ("Sum:3" 1 1+2=3; default; type string)

Function: def fct(x,y,*args,a=3,b=5,**kwargs): """Doc""" return None, global nonlocal (name, params, positional args (*seq), named params (**dict); default; scope: outside func., outer func) (args: call, params: definition)

Operations on Containers: len(c), min(c), max(c), sum(c), sorted(c), val in/not in c, enumerate(c)→(i,v) (on keys for dict.)

copy.copy() copy.deepcopy() (import copy), zip(c1,c2,...) [tuples for same index], all(c) any(c) (if ... are True), | Ordered sequences: (lists, tuples, str) reversed(c), c.index(val), c*5, c1+c2, c.count(val)

Operations on Lists: L.append(val), L.extend(seq), L.insert(i,v), L.remove(val) (first occurence), L.pop(i) (default last; remove and return), L.sort(), L.reverse(), L.copy()→shallow (elem. have same id), [expression for item in iterable if cond.==True]

Operations on Dict.: d.clear(), del d[k], d.update(d2) add, d.keys() d.values() d.items()→(k,v), d.pop(k)→v

Operations on Sets: Operators: | union s1.union(*s2), & intersection s1.intersection(*s2), - difference (sym. diff.: ^, (S1∪S2)\(S1∩S2)) s1.difference(*s2), < <= > >= (inclusion) (is subset)

Operations on Strings: s.startswith(prefix), s.endswith(suffix), s.count(sub), s.upper(), s.lower(), s.title()→upper first letter (str[s:e] (..., start, end))

s.find(sub) (index first letter; -1 if none), s.index(sub) (index error), s.strip(char) (white char at begin/end, "␣" default), s.center(length, char) (total (returned), "␣" default), s.split(sep, maxsplit) ("␣" default, -1 (all)), s.rsplit(...) (depuis droite), s.swapcase(), s.rfind(...)

Formating: "modele {} {}".format(x,y), f"{x} {x=}"→1 x=1, "%s a %s"%(1,2) →1 à 2, "{0}{0}{1}".format(1,2)→"112", sep.join(list)

II. Algorithmic

Base: entrée → algorithme → sortie / programme implémentation spécifique d'un algo

Correctitude: invariant de boucle: 1) Initialisation 2) Maintenance 3) Terminaison (expliciter au début puis la cond. d'arrêt valeure)

ex.: - Somme [a→n], S=somme [0→i] S=0: somme [0→0] $S=S_{i-1}+i=S[0\to i]$ S=somme [0→n]

- maximum [0:n], max [0:i] max [0:1] (liste de 1 el) $\max(\max_{i-1}, i)=\max(0,i)$ max(0,n)

boucles while: montrer que la boucle se termine ⇒ variant de boucle décroissant et cond. pour une certaine valeure

ex.: reste division entière de a par b: (while r ≥ b) r décroit strictement de b à chaque itération. ⇒ pendant la boucle: a-r est mutiple de b, à la fin de la boucle a-r est multible de b et 0≤r<b (0 car sinon la boucle aurait dû s'arreter avant)

Complexité: $F=O(g) \Leftrightarrow \exists C,N>0$ tq. $\forall n>N$ $F(n)\le C\cdot g(n)$ (ex.: $2n\le 2n^2$ $\forall n\ge 1$ ⇒ N=1 C=2 ⇒ $2n=O(2n^2)$)

$an+b\le(a+b)n$ $\forall n\ge 1$ ⇒ C=a+b N=1 ⇒ $an+b=O(n)$ et $n^p=O(n^q)$ $(p\le q)$ O: "majorant"

$F=\Omega(g)\Leftrightarrow \exists C,N>0$ tq. $\forall n>N$ $F(n)\ge C\cdot g(n)$ Ω: "minorant", $F=\Theta(g)\Leftrightarrow \exists C_1,C_2,N>0$ tq. $\forall n>N$ $C_1\cdot g(n)\le F(n)\le C_2\cdot g(n)$

Θ: "bornée" ex.: $an+b=\Theta(n)$ $an^2+bn+c=\Theta(n^2)$ $P(n^q)=\Theta(n^q)$, T(n): temps de parcours au pire des cas

récursion: "diviser pour regner" plusieurs appel au sein de la fonction jusqu'à atteindre valeure retournée.

Tri : Tri par sélection $\Theta(n^2)$: (sur place)

répéter len(L) fois[1] : trouver le minimum de L[i:][2] et le placer à la position i[3]

```
def tri_selection(L):
    n = len(L)
1   for i in range(n):
        m = L[i]
        m_index = i
        for j in range(i+1, n):        # recherche minimum L[i:]
            if L[j] < m:
                m = L[j]
                m_index = j
3       L[i], L[m_index] = L[m_index], L[i]   # (swap!)
```

ex. : [64, 25, 12, 22, 11] → 11 25 12 22 64 → 11 12 25 22 64 → 11 12 22 25 64 → 11 12 22 25 64

Tri par Insertion $\Theta(n^2)$: (sur place)

répéter len(L) fois[1] : tant que l'élem. précédant est plus petit que l'élément courant i continuer de comparer avec celui d'avant. Insérer l'élem. i à la position trouvée. (échanger tant qu'il est plus petit[2] que le précédant) jeu de carte (décale les élem.)

```
def tri_insertion(L):
    n = len(L)
1   for i in range(n):
        j = i
2       while j>0 and L[j] < L[j-1]:
            L[j], L[j-1] = L[j-1], L[j]
            j -= 1
```

ex. : [12, 11, 13, 5, 6] → 12 11 13 5 6 → 11 12 13 5 6 → 5 11 12 13 6 → 5 6 11 12 13

Tri par Fusion (récursif) $\Theta(n \cdot \log_2(n))$: ($\Theta(n)$ de stockage)

Divise la liste en ~2[1] si sa taille est + grande que 1[2] puis trier ces deux sous listes de la même manière[3]. Fusionner ces 2 sous-listes une fois quelles ont été triées.[4] (divise virtuellement)

```
def tri_fusion(L, bas, haut):
2   if haut - bas > 0:
1       milieu = (bas + haut) // 2      # (partie entière len(L)/2)
3       tri_fusion(L, bas, milieu)
        tri_fusion(L, milieu+1, haut)
4       fusion(L, bas, milieu, haut)
def fusion(L, bas, milieu, haut):       # ← fusionne 2 ss-listes triées en Θ(n)
    L1 = L[bas : milieu+1]
    L2 = L[milieu+1 : haut+1]
    L1.append(float('inf'))
    L2.append(float('inf'))
    L1_index = L2_index = 0
    for i in range(bas, haut+1):
        if L1[L1_index] <= L2[L2_index]:
            L[i] = L1[L1_index]
            L1_index += 1
        else:
            L[i] = L2[L2_index]
            L2_index += 1
```

ex. : 1[38,27,43,3,9,82,10]
2[38,27,43] 10[3,9,82,10]
3[38] 5[27,43] 11[3,9] 15[82,10]
6[27] 7[43] 12[3] 13[9] 16[82] 17[10]
4[38] 8[27,43] 14[3,9] 18[10,82]
9[27,38,43] 19[3,9,10,82]
20[3,9,10,27,38,43,82]

x : ordre des étapes (récursif)

Tri à bulles $\Theta(n^2)$: (sur place)

Échange les éléments adjacents s'ils sont dans le mauvais ordre[3] jusqu'à ce que la liste soit triée. (répète len(L) fois[1] : Fais remonter le max(L[:len(L)-i][2] à la position len(L)-i-1)

```
def tri_bulles(L):
    n = len(L)
1   for i in range(n):
2       for j in range(n-i-1):          # -1 car on utilise l'élém. L[j+1] donc le dernier ⚠
3           if L[j] > L[j+1]:
                L[j], L[j+1] = L[j+1], L[j]
```

ex. : [5,1,4,2,8] → 1 5 4 2 8 → 1 4 5 2 8 → 1 4 2 5 8 → 1 4 2 5 8 → 1 2 4 5 8 → 1 2 4 5 8 → 1 2 4 5 8

Temps : (moyen) : • Constant : L[i] (w/r), L.append(e), L.pop() → Last, D[K] (w/r), D.pop(K) (del D[K]) • linéaire : L.insert(i,e), L.pop(i), L.remove(x), del L[i:j:K]

max(L) → $\Theta(n)$, L.index(e) → $\Theta(n)$, (recherche binaire, dichotomie) L_triée.index(e) → $\Theta(\log_2(n))$

trier une liste : $\Omega(n \cdot \log_2(n))$

Graphes : G = (V, E) (sommets, arrêtes), E = (u, v) (paire de sommets, ordonnée pour les graphes dirigés), n : nb. de sommets, m : nb. d'arrêtes, $0 \le m \le n(n-1)$ G dirigé, $0 \le m \le \frac{n}{2}(n-1)$ G non dirigé, matrice d'adjacence $\Theta(n^2)$ espace

ex. :

départ ↓ 0 1 2 3 ← arrivée
0 [0 1 0 0],
1 [0 0 1 1],
2 [0 0 0 1],
3 [0 0 1 0]

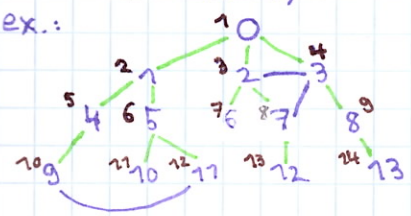
, liste d'adjacence : {0:[1], 1:[2,3], 2:[3], 3:[2]} ↳ $\Theta(n+m)$ espace

, cycle : chemin d'un sommet u à lui même (graphe acyclique : n'a pas de cycle), connexe : il existe un chemin qui relie toute paire de sommets (tous sommets est atteignable peu importe le sommet de départ), arbre : graphe connexe et acyclique, arbre couvrant d'un graphe : un graphe T qui a les même sommets que G, un ss-ens. des arrêtes de G tq. T soit un arbre, arbre couvrant minimal : somme des poids des arêtes de T est minimale (si G est connexe et non pondéré : tous les arbres couvrants sont de même poids, tout arbre couvrant est minimal) DFS et BFS produisent des arbres couvrants minimals qui reflètent le parcours.

BFS/chemin/arbre (liste d'ajacence) $\Theta(n+m)$: Breadth-First Search à partir d'un sommet s explore d'abord les sommets à distance 1, puis à distance 2, ... (comme un liquide)

```
From collections import deque
def BFS(G,s):
    n=len(G)
    a_parcourir=deque([s])
    vu=[0 for v in range(n)]
    vu[s]=1
    chemin=[[] for v in range(n)]
    chemin[s]=[s]
    T={v:[] for v in range(n)}
    while a_parcourir:
        sommet=a_parcourir.popleft()
        for u in G[sommet]:
            if not vu[u]:
                a_parcourir.append(u)
                vu[u]=1
                chemin[u]=chemin[sommet].copy()
                chemin[u].append(u)
                T[sommet].append(u)
                T[u].append(sommet)
        print(sommet)
    return chemin, T
```

ex.:



0→1→2→3→4→5→6→7→8→9→10→11→12→13

Pile/Deque: ajouter des éléments à la Fin ou supprimer le premier élément, (collections.deque), temps cte. ($\Theta(1)$): d.append(e), d.pop()→last, d.appendleft(e), d.popleft()→First

Recherche binaire: (dichotomie) position de x dans L triée.

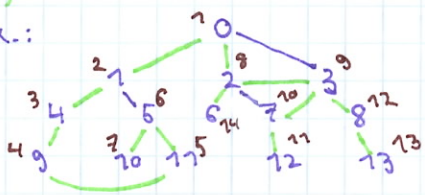
```
def recherche_binaire(L,x):
    n=len(L)
    bas=0
    haut=n-1
    while haut>=bas:
        milieu=(bas+haut)//2
        if L[milieu]==x:
            return milieu
        elif L[milieu]>x:
            haut=milieu-1
        else:
            bas=milieu+1
    return None
```

DFS/arbre (liste d'ajacence) $\Theta(n+m)$: (récursif) Depth-First Search à partir d'un sommet s explore en allant le plus loin possible avant de revenir en arrière. (explorateur)

```
def DFS(G,s):
    if not vu[s]:
        vu[s]=1
        print(s)
        for v in G[s]:
            if not vu[v]:
                T[s].append(v)
                T[v].append(s)
                DFS(G,v)

G={0:[...],...}
n=len(G)
vu=[0 for v in range(n)]
T={v:[] for v in range(n)}
DFS(G,0)
print(T)
```

ex.:



0→1→4→9→11→5→10→2→3→7→12→8→13→6

```
def DFS_prime(G,s):
    n=len(G)
    a_parcourir=[s]
    vu=[0 for i in range(n)]
    vu[s]=1
    while a_parcourir:
        sommet=a_parcourir.pop()
        for u in G[sommet]:
            if not vu[u]:
                a_parcourir.append(u)
                vu[u]=1
        print(sommet)
```

0→3→8→13→7→12→2→6→1→5→11→9→10→4

Ordre de croissance: $\{\log_2(n), \log_2(n^3)\}, (\log_2(n))^2, \sqrt{n}, \sqrt{n}\cdot\log_2(n), n, n\cdot\log_2(n), n^2$

ex.: G={0:[1,2,3], 1:[0,4,5], 2:[0,3,6,7], 3:[0,2,7,8], 4:[1,9], 5:[1,10,11], 6:[2], 7:[2,3,12], 8:[3,13], 9:[4,11], 10:[5], 11:[5,9], 12:[7], 13:[8]} (non orienté, connexe)

ICS - CMS - Résumé

2ème semestre

Equations non-linéaires: Bolzano: F continue et $F(a)\cdot F(b)<0$, sécante: $n_{min} > \log_2\left(\frac{b-a}{\varepsilon}\right)-1$

$x_n = \frac{a_n F(b_n) - b_n F(a_n)}{F(b_n) - F(a_n)}$ erreur: $\frac{|b_n - a_n|}{2} < \varepsilon$ (bissection $x_n = \frac{a_n + b_n}{2}$), point fixe: picard →

$g(x) = \lambda F(x) + x$, k-contractante: $|F'(x)| < 1 \ \forall x \in I$, $x_{n+1} = g(x_n)$ $x_1 = g(x_0)$ erreur:

selon $O_x \rightarrow |x_{n+1} - x_n| < \varepsilon$ selon $O_y \rightarrow |F(x_n)| < \varepsilon$, newton → $g(x) = x - \frac{F(x)}{F'(x)}$, newton-corde →

$g(x) = x - \frac{F(x)}{F'(x_0)}$, parallèle → $g(x) = x - \frac{F(x)}{\lambda}$, sécante → $x_{n+1} = x_n - \frac{F(x_n)(x_n - x_{n-1})}{F(x_n) - F(x_{n-1})}$

Calcul intégral: Lagrange: $L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$ $L(x) = \sum_{i=0}^{n} y_i L_i(x)$, non-composite: sur $[a,b]$,

composite: sur n partitions de $[a,b]$, partitions régulières: $h = \frac{b-a}{n}$ non-régulières:

↳ $I = \sum I_n$

$h = x_{i+1} - x_i$, rectangles: à gauche → $F(a)\cdot h$ à droite → $F(b)\cdot h$ point milieu → $F\left(\frac{a+b}{2}\right)\cdot h$,

trapèze: $\frac{F(a)+F(b)}{2}\cdot h$, simpson: $\left(\frac{1}{6}F(a) + \frac{4}{6}F\left(\frac{a+b}{2}\right) + \frac{1}{6}F(b)\right)\cdot h$, erreur relative:

$\varepsilon = |I_n - I_{n+1}|$, erreur absolue: $\varepsilon = \frac{|I_n - I_{n-1}|}{I_n}$

Equations différentielles: Cauchy $y'(t) = F(t, y(t)) \ \forall t \in I$, $y(t_0) = y_0$, $I = [t_0, T]$ en

N partitions: $h_n = t_{n+1} - t_n$ ou $h = \frac{T - t_0}{N}$, Euler progressif: $u_{n+1} = u_n + h\cdot F(t_n, y(t_n))$

$u_0 = y_0$, Euler rétrograde: $u_{n+1} = u_n + h\cdot F(t_{n+1}, u_{n+1})$ ← recherche zéro