Name: Mary-Lynn Hayek (El) -13 May 2022-

Final documents to take to T.G final exam:

Important links to use:

|  |  |
| --- | --- |
| Link | Related chapter |
| https://replit.com | All of them (code in python) |
| https://www.desmos.com/calculator | Graphic calculator |
| <https://graphonline.ru/en/?msclkid=04c56f7ac61f11ec95d0d0c1327d609a> | Find the shortest Path in a graph |
| [https://networkx.org/documentation](https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.shortest_path.html#networkx.algorithms.shortest_paths.generic.shortest_path) | Network X documentation |

**Chapitre: Markov Chains:**

**Partie cours (summary):**

|  |  |
| --- | --- |
| **Important formulas** | **What they help get:** |
|  | **Pour démontrer que c’est une chaine de markov:**  **Le future de mon processus est indépendant du passé mais conditionné au présent.Pm(ij) c’est la probabilité de passage de i vers j avec un chemin de longueur m.** |
|  | Pi[n] contient les probabilités de la chaîne après K sauts.  **Pi(i)= En régime stationnaire: Fréquence de passage a l’état i.**   1. Proportion du temps pendant lequel la CM est a l’état i. 2. La probabilité de trouver la chaîne a l’état i en un instant quelconque. |
| Pi[3] = Pi[0] \* P^3 |  |
|  | **Equation d’équilibre** |
| E( | **Temps moyen de retour ,**L’inverse de la probabilité stationnaire est le temps de retour. |
|  | **Condition de stationnarité (Vecteur de probabilité)** |
| E[si]= | **Temps de séjour:**   1. **--> 1-Pii** 2. **-->Pii(1-Pii) , Avec Pii un element de la matrice de transition.** 3. **->(1-Pii)**   **On peut directement trouver le temps de séjour d’un état du vecteur Pi.** |
|  | **Processus de naissance et de mort**  Donc elle suit une loi géométrique de paramètre (1-Pii).  De moyenne **E[Si]=(1/1-Pii)**  **Naissance fo2**  **Mort ta7et**  **Equation d’équilibre:**    Donc,    **Comment trouver la probabilité stationnaire:** On trouve les équations d’équilibre:  **On exprime toutes les probabilités en fonction de**  4 = /1-p  3 = /1-p  2 = /1-p  1 = = /1-p (depend des valeurs)  Pour trouver Pi[0]= on réalise la somme des Pi, on factorise: |
|  | **Temps de premier passage a un état i** |
|  | **Temps de retour:** On prend i comme origine |
| **Remarques:**   1. **La somme des arêtes sortants d’un nœud est = 1.** 2. **La somme des elements d’une ligne dans la matrice de transition est = 1.** 3. **Pm(ij) est la probabilité de passer de i vers j avec un chemin de longueur m.** 4. **Probabilité d’une trajectoire: P=** 5. 5) = [ 0.1 0 0.1 0.8 ]   **Ca signifie le suivant: a l’instant 5 la probabilité de la trouver en 1 est de 10%, en 2 est de 0% etc… La somme de ces elements est toujours égale a 1.**  **Etat récurrent:** état répété tout le temps. La probabilité que le temps de retour soit inférieur a l’infini est = 1. (Une fois dans cet état on y retournera sûrement après un temps fini).    **Etat transient (transitoire) :** La probabilité de ne pas retourner a cet état est fini > 0. Le temps de retour est de moyenne finie. Condition nécessaire:  **Probabilité stationnaire d’un état transient est nulle. Donc on calcule la P.S.**  **Probabilité stationnaire d’un état récurrent est non-nulle**  **État périodique:** Période = 1. Si on a une **boucle** donc **apériodique.**  - **i conduit a j** => **Symbole :** i -> j: **Terme utilisé:** j est accessible a partir de i).  - **i <-> j** => **Terme utilisé:** i est j communiquent. Existe un chemin de i a j et de j i. Alors i et j de meme nature. (Récurrents, transients, périodiques de meme période).  - Les états **récurrents** sont les états qui restent dans la probabilité stationnaire, ceux qui possèdent une probabilité stationnaire nulle sont les états transients.  - Pour une chaîne de Markov quelconque si on trouve un vecteur pi vérifiant les équations d’équilibre les états avec pi[i]> 0 sont **récurrents**.  - Pour une CM **irréductible**, elle est **récurrente** ssi elle admet une probabilité stationnaire. Si non, elle est transiente.  - **Une CM finie et irréductible est récurrente admet une proba stationnaire:** ejbare ykoun fiya at least 1 état récurrent ta ykoun 3enda ma7all trou7 3le   1. **Fermé:** Eza fetet 3a this state aw sous-ensemble de la CM bi battil fiyye odhar menno. Pij = 0.   **Irréductible:** Tous les elements bi albo bi waddo 3a baad. Ma droure tkoun une composante fortement connexe. (Mais les composantes fortement connexes sont irréductibles).  Dans une CM **irréductible** tous les états sont de meme nature.  **- État absorbant:** il forme un ensemble fermé la 7alo.  On peut avoir une CM irréductible et transiente si le nombre d’état est infini.  **- CM ergodique:** Une CM **irréductible** (son graphe est fortement connexe), **récurrente** et **apériodique** (n’a pas de période) est dite **ergodique**.  **- CM périodique:** Quand une CM est apériodique elle va converger vers les probabilités stationnaires.     1. Inessential states sont les états transients. (Un état transient est un état dont la proba enno erja3 3le is low). 2. **La probabilité des états transients a l’infini est nulle.** 3. **La probabilité des états absorbants a l’infini est = 1.** 4. Le vecteur pi[0] est déterminé par l’état (sommet initiale). 5. Communication classes (composantes fortements connexes). 6. Dans une CM irréductible, tout le graphe forme **une seule composante fortement connexe**, or quand on a plusieurs composante fortement connexe la CM n’est plus irréductible. 7. La périodicité dans un état transient n’a pas de meaning. 8. Comment lire la matrice élevée a l’infini d’un processus de naissance et de mort?     Probabilité de perte sachant que j’ai 1 dollars est 0.9797 (P11).  Probabilité de perte sachant que j’ai 2 dollars est 0.9492 (P21).  Probabilité de gain sachant que j’ai 2 dollars est 0.0508 (P28). --> [ 1-P(perte) ] car somme d’une ligne = 1.  Probabilité de gain sachant que j’ai 1 dollars est 0.0203 (P18).  **Équilibre de probabilité:** | |

**Exemples pour m’aider:**

|  |  |
| --- | --- |
|  | - 1 est transient.  - 2, 3, 4, 5, 6 sont récurrents  - CM non-irréductible, On peut la réduire.  - {4, 5, 6} => Ensemble fermé et irréductible.  - {1, 2} => irréductible et non-fermée  - {3, 4} => irréductible et fermée.  - {2, 3} => irréductible et fermée.  - Les ensembles T={1}, C1={2,3}, C2={4,5,6} |
|  | La probabilité d’être a l’état 2 après 3 sauts est de 1/3. |
|  | **Eza talab ntalli3 pi(1), pi(2) etc…. Trouver la probabilité stationnaire avec la méthode du vecteur propre et prendre ces elements (no need to do the full calcul).** |
|  | **- Il n’y a pas de transition entre l’état 3 et l’état 5:** cad dans la matrice il y a un 0 sur la 3ème ligne et la 5ème colonne.  - **L’état 5 est absorbant:** cad P(5,5)=1 , loop 3a 7alo. Men 5 ma fiye rou7 wala ma7all. Donc les valeur sur la ligne 5 sont toutes nulles sauf (5,5)=1.  - **Le sous-ensemble {1,3,4,5}:** cad on entre dans cet ensemble et on est incapable de sortir menno. Donc ma fina nrou7 des états 1,3,4 et 5 3ala be2e l états, et dans ce cas on a l’état 2 seulement. Donc toute la colonne de 2 est nulle.  - **Quand la chaine est a l’état 3, elle y reste un temps moyen valant 4:**  cad le temps de séjour la CM a l’état 3 est de 4. **E[S3]=1/1-P(3,3)=4** donc P(3,3) = 3/4. Et sachant que la somme des elements d’une ligne = 1 on trouve le reste.  **- Probabilité d’une trajectoire: 1,4,1,4,1 sachant que la chaine est initialement en 1 vaut 0.0004:**    Or P(41)= 0.1 on la prend de la matrice.  P14 \* P41 \* P14 \* P41 =  Alors \* 0.001 = 0.0004 donc  - **Etant en 2, la probabilité que la chaine soit a l’état 4 après deux sauts vaut 0.14:** P( = 4 / = P( = 4 /  Produit scalaire de la ligne 2 \* la colonne 4: 0.7\*0.2+0+0+  Or matrix power sur python pour la trouver directement. |
| **Exo 2: Calcul de probabilité stationnaire de la CM:**  On a la CM suivante avec sa matrice de transition P. | On cherche un vecteur Pi=[ Pi(1) Pi(2) ] qui vérifie cette équation:    **Explication:**  Pi[1] c’est la première multiplication du vecteur Pi avec la matrice de transition P.  Pi[1]=vecteur Pi\*Colonne 1 de P  Pi[2]=vecteur Pi\*colonne 2 de P.  Pi[3]=vecteur Pi\*colonne 3 de P.  Pi[4]=vecteur Pi\*colonne 4 de P.  Etc… |
| Trouver le vecteur Pi de la chaîne de Markov suivante: | Pi=[Pi(1) Pi(2)]\*Matrice P  Le nombre des elements du vecteur Pi sont égales au nombre de sommets dans le graphe. (nombre total d’états).  **Steps:**   1. Formuler le vecteur Pi. 2. Multiplication matricielle 3. Formulation du système d’équations 4. Ajout de l’équation de normalisation 5. Résolution du système. (Analytiquement ou bien sur python). |
| **Start:** Room 1  **Destination:** Room 5  **Find the number of steps needed to reach the destination** | E(nombre de sauts pour arriver a 5/X0=1)=  S(i)=1+ |

Text

Description automatically generated with medium confidence**Q1:** Calculating the Stationary Distribution: Principe adopted: (Probabilité stationnaire):



But : Déterminer l’état de la chaîne de Markov a l’infini. Step by step:

1. Code sur Python : Libraries to include :

|  |
| --- |
| import numpy as np  from numpy.linalg import matrix\_power  from numpy import linalg as LA  np.set\_printoptions(precision=5,suppress=True) |

1. Code sur Python : Création la matrice de transition :

|  |
| --- |
| P=[[0,0.3,0.4,0.3],\  [0,0.2,0,0.8],\  [0,1,0,0],\  [0,0.5,0.5,0]]  P=np.array(P)  print(P) |

1. Code sur Python : Voir l’état de la chaîne de Markov après 2 sauts (Passage de l’état i a l’état j en 2 sauts):

**N.B :** La position P7[1][3] est P (2,4) dans la vie réelle (on commence de 0 sur Python) :

|  |
| --- |
| P2=matrix\_power(P,2)  print(‘Proba a 7 sauts:’, P2[1][3])  #P(X2=4/X0=2)=Probabilité de passage de 2 vers 4 en 2 sauts (P24)^2 produit scalaire L2C4 |

Code sur Python : Probabilité que le rat étant dans la piece 4, soit dans la piece 1 après 10 sauts.

|  |
| --- |
| P10=matrix\_power(P,10)  print(‘P10:\n’,P10)  print(‘P(10)(4,1)=’,P10[3][0])  #Cette méthode peut nous aider a trouver l’état de la CM a l’infini et determiner le vecteur Pi si **ergodique:**  P1000=matrix\_power(P, 1000)  print(‘La matrice a l’infini:\n’, P1000)  Remarque: Pour démontrer qu’un état est absorbant on trouve P1000 et si toutes les probabilités seront nulles sauf celle de l’état 5. Car 5 est absorbant et que i conduit a 5 quelque soit i. (5 ma bet wadde la ma7all). |

Code sur Python : État initiale pi(0): Case impaire,Quelle est la probabilité de la trouver dans la case 4 après 10 sauts.

|  |
| --- |
| pi\_0=np.array([1/3,0,1/3,0,1/3,0])  pi\_10=pi\_0 @ P10  print("pi\_10 a partir des cases impaires=\n",pi\_10) |

Code sur Python : Le vecteur Pi a l’instant 6 : (Loi marginale de l’instant 6) :

|  |
| --- |
| Pi\_0=np.array([ 1/3 ,0 ,1/3 ,0 ,1/3 ,0 ])  Pi\_6=Pi\_0@matrix\_power(P,6)  print(‘Loi marginale de 6’, Pi\_6) |

Code sur Python : Montrer qu’une CM est irréductible :

|  |
| --- |
| Méthode 1:   1. Include NetworkX 2. Call the function: isConnected()   Méthode 2:   1. Elevation au carré 2. Voir si des 0 existent, sinon donc irréductible. |

Code sur Python : Trouver une valeur moyenne :

|  |
| --- |
| Q=[….]  qMoy=np.dot(Q,pStat)  Print(‘Quantite moyenne’, qMoy) |

Code sur Python : Démontrer si une CM est périodique :

|  |
| --- |
| #Les puissances successives de P vont réaliser un switching dans les 0.  #Si elle est **apériodique** donc elle converge vers la probabilité stationnaire.  print(matrix\_power(P,10000))  print(matrix\_power(P,10001))  #Voir si **periodique** ou **aperiodique**:  mat1=matrix\_power(P,10000)  mat2=matrix\_power(P,10001)  mat3=mat1-mat2  if(len(mat3[mat3>0.0000001])):  print('periodique \n')  else:  print('aperiodique \n') |

Code sur Python : Trouver la **probabilité stationnaire** avec la méthode du vecteur propre (Pi(1), Pi(2) etc..)

|  |
| --- |
| **#Methode 1: Vecteur propre:**  val, vec = LA.eig(np.matrix.transpose(P))  print('val',val,'\nvect: \n',vec,'\n')  pStat=vec[:,0]  **#assuming the eigen value 1 is the first value in val**  pStat=pStat/np.sum(pStat)  print('Stationary probabilities by eigen values method: \n',pStat)  **#Méthode 2: Elever la matrice P a une puissance élevée: (Applicable si CM ergodique):**  Pinf=matrix\_power(P, 1000)  print('La matrice a l’infini:\n', Pinf)  pStat=Pinf[0:]  print ('pStat:', pStat) |

Code sur Python : Trouver la fermeture transitive irréductible :

|  |
| --- |
| o=len(P)  Mb=np.mat(P,dtype=bool)  I=np.eye(o,o,dtype=bool)  TC=(Mb+I)\*\*(o-1)  print('transitive closure of P:\n', TC\*1,'\n') |

Code sur Python : Si on a le vecteur Pi(0) : Dessiner les éléments Pi(n) :

|  |
| --- |
| pi\_0=[1,0]  for i in range(20):  print('pi\_',i,'=',pi\_0 @matrix\_power(P,i)) |

Code sur Python : Si on a le vecteur Pi(0) : trouver le vecteur Pi(10)

|  |
| --- |
| pi(0)=np.array([ 1/3 ,0 ,1/3 ,0 ,1/3 ,0 ])  Pi(10)=pi(0) @ P(10)  Print(“pi(10) a partir des cases impaires=\n”,pi(10)) |

Code sur Python : Trouver la fermeture transitive de la matrice P:

|  |
| --- |
| o=len(P)  Mb=np.mat(P,dtype=bool)  I=np.eye(o,o,dtype=bool)  Tc=(Mb+I)\*\*(o-1)  print('Fermeture transitive\n', Tc\*1) |

Code sur Python : déterminer les états transients:

|  |
| --- |
| P=[[0.99,0.01,0,0],\  [0.005, 0.99, 0.005,0],\  [0, 0, 0.1, .9],\  [0, 0, 1,0]]  P=np.array(P)  print(P)  print(matrix\_power(P,10))  print(matrix\_power(P,100))  print(matrix\_power(P,1000))  print(matrix\_power(P,10000))  print(matrix\_power(P,10001)) |

**Chapitre: Programmation Linéaire:**

|  |  |
| --- | --- |
| **CODE** | **FONCTION** |
| import numpy as np  from numpy.linalg import matrix\_power  from scipy.optimize import linprog | Packages to import to replit. |
| #ex1  c=[-4,-12] #maximization on met -  A=[[-1,0],  [0,-1],  [1,0],  [0,1],  [1,2]]  b=[0,0,1000,500,1750]  res=linprog(c,A\_ub=A,b\_ub=b)  print(res) | **Définir:**   1. Le vecteur colonne c qui contient toutes les constantes de la fonction objective. 2. La matrice A qui contient les coefficients de mes variables des contraintes d’inégalités. (Forme standard g(x). 3. Le vecteur colonne b contient les constantes ( Syntaxe: contraintes < **constante qu’on place dans la matrice B** ). |

**Faire attentions aux signes!! NEVER FORGET LES 2 CONTRAINTES X1>0 ET X2>0 !!!!**

**Puisque la fonction linprog minimise:**

**- Ajouter le signe (-) si le but de notre exercice est de maximiser notre fonction.**

**- Laisser le signe + si le but est de minimiser.**

|  |  |
| --- | --- |
| Dans la création de la matrice A: on a 2 méthodes pour la remplir: use the first one: | Explication syntaxe: |
| **Adopter la méthode des bounds:**  **#objective function : On veut minimiser le cout donc on n’ajoute pas de (-) dans c**  c=[2,24,13,9,20,19]  A=[[-110,-205,-160,-160,-420,-260],\  [-4,-32,-13,-8,-4,-14],\  [-2,-12,-54,-285,-22,-80]]  b=[-2000,-55,-800]  bound=[(0,4),(0,3),(0,2),(0,3),(0,5),(0,2)]  res = linprog(c, A\_ub=A, b\_ub=b, bounds=bound)  print(res) | Bound (a,b) signifie:  1st variable >0 et <4 etc… |

**Après résolution sur Python, on a : Output of the command:**

x=[ 4, 0, 0.41, 3,2.41, 0]

Le coût total de la solution minimale= **fun** = 88.6 USD

**Formulation du problème:**

|  |
| --- |
| **Formuler un problème de Recherche Opérationnelle: c’est de définir trois composantes:**   1. **Variables de décision**.x=[x1,…,xn], Le but c’est de trouver les meilleurs valeurs de ces variables. 2. **Fonction objectif** à maximiser ou à minimiser. (x --> f(x)). 3. **Contraintes:** qui vont changer le domaine de définition de la fonction objective.     **Arg max f(x) signifie:** trouver la valeur des variables x qui la maximisent la valeur de f(x).  **Fonction standard est g(x): La forme standard nous permet de formuler le code sur Python.**  x1 >=0 [La forme standard: g1(x1,x2)= - x1 <= 0]  x2 >=0 [La forme standard: g2(x1,x2)= - x2 <= 0]  x1<=1000 [La forme standard: g3(x1,x2)= x1 - 1000 <= 0] |

**Méthode Graphique:**

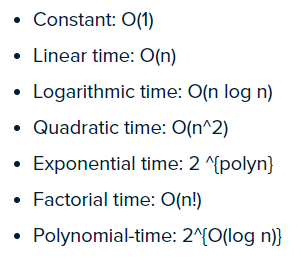
|  |
| --- |
| 1. Open desmos 2. Représenter les contraintes sous-forme de droite dans un plan 2D. 3. Determiner le domaine admissible 4. Voir les intersections des droites 5. **Solution optimale:** Gain élevé, changer la valeur après l’égalité de facon a rester dans le domaine admissible (dernier point de contacte). 6. Je cherche un point de décision dans ce plan. C’est un plan, fonction de x1 et x2. Mais ayant des contraintes. Qui se trouvent dans le plan aussi. 7. On rempli toutes les contraintes graphiquement, afin de déterminer la région de décision.   C’est la **région admissible** (feasible region). C’est notre domaine de définition réel.   1. La solution optimale est le point intersection qui donne un gain maximal. Déplacer par translation jusqu’au dernier point de contact avec le point admissible (le point de contact est la décision optimale). 2. **Pour voir la solution optimale: On prend the highest coordinates, je les remplace dans f(x) et on trouve la valeur de f(x). (Voir toutes les combinaisons des coordonnées des intersections).** 3. **Interpretation graphique:**   - Le plan de décision est l'espace R"  - Les contraintes sont des hyperplans de R"  - Le domaine admissible est un domaine convexe (polytope)  - La solution optimal est un sommet de ce polytope. |

**Chapitre: Complexity:**

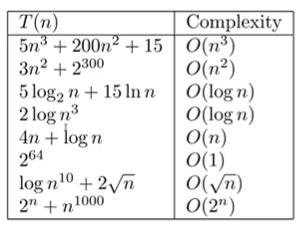
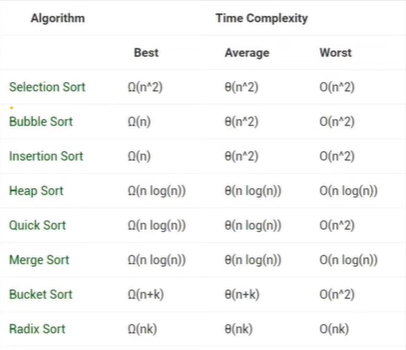
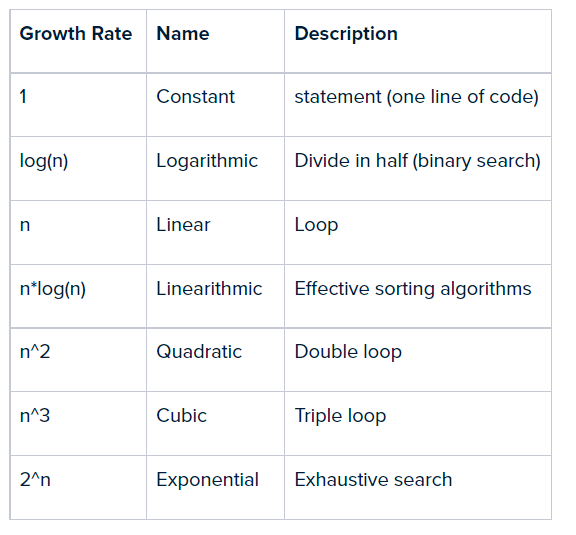
**What is a complexity?**

1. Algorithmic complexity is **a measure of how long an algorithm would take to complete given an input of size n.** C’est la façon d’étudier le temps exécution des algorithmes. Unite: Nombre d’instruction. Etude asymptotique. (Temps, performance etc..). (Approximatif). C’est alors le nombre d’instruction (fonction de la taille des données en entrees) qu’il doit exécuter pour arriver a la solution.

|  |  |
| --- | --- |
| **Algorithm / Function** | **Complexity measure** |
| **Dijkstra's Algorithm** | * 1. IMG_256 |
| **Prim algorithm** | * 1. **O ( ( V + E ) l o g V )** |
| **Kruskal** | * 1. **O ( E log V )** |
| **Bellman-Ford** | * 1. **O(VE)** |

1. **Quelques règles générales:**
2. Complexity is equal to a constant O(1) quand on n’est pas dependant d’une variable n.
3. Linear time complexity O(n) when running time increases with the input
4. Quand the size of the input data reduces in each step : Complexity = O(log(n))
5. When we have nested loops: O(n^2) for 2 nested loops
6. O(2^n) means that an algorithm where the growth is \*2 every time the input data is added.

**Some famous algorithms:**



**Les étapes de calcul:**

1. **Drop lower terms** (ex: T(n)= 2\*n^2 + 3\*n + 1) we drop the one in blue.
2. **Drop constant multipliers** (ex: T(n)=2\*n^2) we drop the coefficient.
3. **Alors la réponse finale est** T(n)=O(n^2)

**Note:**

**On réalise une: !!!!!!!!!!!!!!!**

**Addition: Si outside of the loop**

**Multiplication: Si dans la meme loop.**

**When we have a for() loop look at the max value it can take and place it in the O().**

**En resume:**

1. Split your algorithm into operations
2. Calculate the Big O of each operation
3. Add the Big O from each operation
4. Strip out the constants
5. Find the highest order term

**Les complexités en ordre croissant: Pour savoir laquelle choisir:**



Pour calculer un element de la matrice produit, il nous faut n produit et (n-1) sommes donc 2n-1 operations. **Au total:** n^2(2n-1) = 2n^2-n = O(n^3)

**Space complexity:** The amount of space taken by a command. The more time efficiency we have the less space it takes (Inversement proportionels).

|  |  |
| --- | --- |
| Examples | Complexity Value |
|  | Constante: O(1) |
|  | Linear: O(n) |
| Let us consider the following machine:   1. Single processor 2. 32 Bits 3. Sequential execution 4. 1 time unit for arithmetic and logical operations 5. 1 time unit for assignment and return statements   If we write the following code on the machine above:  Code #1:    Code #2: | **Code #1: Sum of 2 elements:**  Complexity is : O(2) = O(1) since it’s a constant  **Explanation:**  1 unit for the sum (+)  1 unit for the return  **Code #2: Find the sum of all elements of a list/array:**  **Complexity is:** 1+2\*(n+1)+2\*n+1=O(n)  **Code #3: Find the sum of all elements of a matrix:**  Complexity is: O(n^2) (2 nested loops, 1 pour les lignes et l’autre pour les colonnes). |
| **A regular for() loop:** | **Time complexity is:** O(n)  **Explanation:**  The loop runs (n) times and (x=y+z) is 1 expression donc constant time to execute= T(n)=n\*Constante. On peut supprimer la constante donc O(n). |
| **Nested Loops:** | Time complexity is: O(n^2)  **Explanation:**  **The expressions:** x=y+z : constant time.  **First loop:** O(n) runs (n) times  **Second loop:** O(n) runs (n) times |
| **Sequential statements:** | We work statement by statement:  Complexity: C1+C2\*n+C3\*n = O(n)  We can ignore the constants C[1a3].  Faire attention que les for() loops ne sont pas nested donc ca ne va pas être au carré! C’est sequential (wara ba3d). |
| Conditional statements: **if(): O(1)** | Si la if() condition contient une for() loop par exemple, la complexité va être = O(n), et si la else statement contient des nested for() loops alors la complexité sera O(n^2) pour cette partie. Donc en total on a :  O(n) et O(n^2): La complexité totale sera: O(n^2).  **On prend la valeur la plus grande ! ! ! !** |
|  | **Complexity is: T(n)=O(1)\*O(n)=O(n)**  **Line 2: Loop of size (n)**  **Line 3: 1 operation inside the loop** |
|  | **for () loop: O(n)**  **if () statement : O(1)**  **1 line of command: O(1)**  **Total complexity: 2\*(n)=O(n) (On supprime le coeff)** |
|  | **Complexity is:**  **O(1)+O(n)\*[O(1)+O(1)]+O(1)=2+2\*n=O(n)**  **Explication:**  **Line 1: Regular expression: O(1)**  **Line 2: for() loop : O(n)**  **Line 3: if() statement: O(1)**  **Line 4: Regular expression: O(1)**  **Line 7: Regular expression: O(1)** |
|  | **Complexity is: O(n)**  **Explication:**  **Line 1: Regular expression: O(1)**  **Line 2: for() loop de size n/2: O(n/2)**  **Line 4: Regular expression: O(1)**  **Line 6: for() loop de size 100: O(100)**  **Line 8: Regular expression: O(1)**  **Donc:**  **O(1+n/2 \*1+100\*1) = O(n)**  **On supprime les petits termes et les coefficients.** |
|  | **O(g(n))=O(n)\*O(f)** |
|  | **O(g)=O(n)\*O(n)\*O(f)=n\*n\*O(f)** |
|  | **O(g)=**(+n)/2 |
| **Fermeture transitive** | I + M --> n^2  (N-1) produits de matrices => (n-1)\*(2n^2 - n^2)  **Au total:**  Ou  O(Log(n)\*n^3) |
|  | Attention au size maximale dans les loops!  Meme loop=multiplication  Hors de la loop=Addition |

**Chapitre: NetworkX:**

**TP : NetworkX:**

|  |  |
| --- | --- |
| CODE | EXPLANATION |
| import networkx as nx  import matplotlib.pyplot as plt  import scipy as sp  from collections import deque  from numpy import \* | **Librairies to import:** |
| cG=nx.complete\_graph(7) | #graphe complet |
| G.add\_node('A')  G.add\_node(1)  G.add\_nodes\_from([2, 5, ‘A’, 'B']) | 1. Ajouter le sommet A dans le graphe G. 2. Ajouter le sommet 1 dans le graphe G. 3. Ajouter plusieurs sommets en meme temps |
| G = nx.Graph() | Définir G comme un graphe |
| nx.draw(cG) | Draw the graph cG |
| plt.savefig('completeGraph.png')  plt.show()  plt.close() | 1. Save the graph as a .png file 2. Show the graph (plot) 3. Close the plot |
| print(cG.order()) | Print l’ordre du graphe cG |
| print(cG.nodes()) | Print la liste des sommets |
| Print(cG.degree(2)) | Print le degré du sommet 2 (returns a dictionary-like container) |
| nx.draw(G,with\_labels=True) | Dessiner le graphe avec des poids visibles. |
| print("Sommets de G:", list(G.nodes()))  print("Ordre du graphe :", G.order())  print("degres :", G.degree)  print("degre de A :", G.degree('A'))  print("Voisins de 6", list(G.neighbors(6))) | Respectivement:   1. Print les sommets du graphe G. 2. Print l’ordre du graphe G. 3. Print le degre du sommet A 4. Print les voisins du sommet 6 |
| l = list(nx.bfs\_edges(G, 1))  print(‘largeur’, l) | Parcours en largeur |
| l = list(nx.dfs\_edges(G, 1))  print('profondeur', l) | Parcours en profondeur |
| print("The graph is connected?", nx.is\_connected(G)) | To see if the graphe is connected |
| print( "what is the nbre of connected components?",nx.number\_connected\_components(G)) | Print le nombre composantes connexes |
| print("Connected component containing vertex 1",nx.node\_connected\_component(G, 1)) | Voir les composantes connexes contenant le sommet 1. |
| print('components', list(nx.connected\_components(G))) | Print les composantes connexes |
| comp = [G.subgraph(c) for c in nx.connected\_components(G)]  G1=comp[0]  print("order of G1", G1.order()) | Create components as graphs |
| M = nx.adjacency\_matrix(G)  print('adjacency matrix of G:\n', M)  Ou  M=nx.attr\_matrix(G,rc\_order=[1,2,3,4,5,6,'A','B','C','D'])  print('adjacency matrix of G:', M)  print (M\*\*3) | Création de la matrice d’adjacence  Pour élever la matrice M au cube. |
| n = G.order()  I = eye(n, n)  TC = (I+M)\*\*(n-1)  TC\_b = mat(TC, dtype=bool)  print(I)  print(TC)  print(TC\_b)  print(TC\_b\*1) | La fermeture transitive du graphe G. |
| G=nx.read\_edgelist('./graph.edges',nodetype=str)  nx.draw(G,with\_labels=True)  nx.write\_edgelist(G, "graph.edges\_out.txt")  nx.write\_graphml\_lxml(G, "./graph.graphml")  plt.show()  plt.close() | Création d’un graphe a partir d’un fichier .txt |
| def pf(G,s) :  F=[] #file vide  order=1 #ordre de visite  marque={}  for i in G.nodes() :  marque[i]=-1;  marque[s]=order  F.append(s)  while F :  x=F.pop(0)  for y in G.neighbors(x) :  if marque[y]==-1 :  F.append(y)  order=order+1  marque[y]=order  return marque  print("pf(G,1)",pf(G,'1')) | Les algorithmes  #file implémentée par liste  #Cette fonction retourne l'ordre du parcours en largeur du graphe |
| def pf2(G,s) :  #F=[] #file vide  F=deque(); #file vide  order=1 #ordre de visite  marque={}  for i in G.nodes() :  marque[i]=-1;  marque[s]=order  F.append(s)    while F :  #x=F.pop(0)  x=F.popleft()  for y in G.neighbors(x) :  if marque[y]==-1 :  F.append(y)  order=order+1  marque[y]=order  return marque | #file implémentée plus efficacement par un deque |
| nx.write\_graphml(G,'./gg.graphml') | Save graph in graphml file |
| #F=[] #file vide  F=deque();  marque={}  pred={}  for i in G.nodes() :  marque[i]=-1;  pred[1]=s  marque[s]=1  F.append(s)  while F :  #x=F.pop(0)  x=F.popleft()  for y in G.neighbors(x) :  if marque[y]==-1 :  F.append(y)  marque[y]=1  pred[y]=x  elif marque[y]==1 and pred[x]!=y:  return True  return False  print(isCyclic(G,'1')) | def isCyclic(G,s) : |
| s = 0  for i in G.nodes:  s += G.degree(i)  print(‘somme des degres: ’, s)  print(s == 2 \* G.number\_of\_edges()) | #Ma baarif eza 5assa bi théorème du 1st chapitre (|x| = 2|A|)  Somme des degrés |

|  |  |
| --- | --- |
| **Graphe orienté et pondéré** | |
| wG=nx.read\_edgelist('./wG.txt',create\_using=nx.DiGraph(),nodetype=str,data=(('weight',int),))  print(wG['H1']['1']['weight'])  nx.draw(wG,with\_labels=True)  plt.show()  plt.close() | 1. Création d’un graphe orienté a partir d’un fichier .txt nommé wG.txt |
| print(nx.shortest\_path(wG, source='H1', target='V', weight='weight', method='dijkstra')) | Print the shortest path de H1 vers V en appliquant l’algorithme de Dijkstra. |
| print(nx.dijkstra\_path(wG, 'H1', 'V'))  print(nx.dijkstra\_path\_length(wG, 'H1', 'V')) | Networkx algo Dijkstra |
| length = nx.shortest\_path\_length(wG, source='H1', target='V', weight='weight')  print(length)  print(nx.dijkstra\_path(wG, 'H1', 'V'))  print(nx.dijkstra\_path\_length(wG, 'H1', 'V')) | Idem |
| G=nx.read\_edgelist('./as\_rel.txt',data=(('relation',str),))  print("Is connected? ", nx.is\_connected(G))  print("nbr of ASs is :", G.order())  G=nx.read\_edgelist('./as\_rel.txt',data=(('relation',str),))  print("Is connected? ", nx.is\_connected(G))  print("nbr of ASs is :", G.order())  histDeg=nx.degree\_histogram(G)  print(histDeg)  print("maximum degree in the graph", len(histDeg)-1)  plt.bar(range(len(histDeg)),histDeg)  plt.xscale('log')  plt.bar(range(len(histDeg)),histDeg) | **#====== Analyse graphe Internet ====**  Hist as in histogramme |
| plt.xscale('log')  plt.bar(range(len(histDeg)),histDeg)  plt.savefig('histo.png') | #bar chart |
| histDeg=nx.degree\_histogram(G)  avg\_deg=0  for i in range(len(histDeg)):  avg\_deg=avg\_deg+i\*histDeg[i]  avg\_deg=avg\_deg/G.order()  print('degre moyen du graphe:',avg\_deg) | #degre moyen du graphe= (somme i\*histogramme(i))/ordre |
| print(sum(histDeg)) #==G.order()  print(G.order()) | #ordre du graphe |
| print("la ponderation entre le sommet H1 et 1 est : ",wG['H1']['1']['weight']) | Le poids entre 2 sommets d’un graphe |
| histDeg=nx.degree\_histogram(G)  print(histDeg) | Combien de sommets on le degré = 1?  #combien de sommet on un degre =1 : 6672 = histDeg[1] |
| EG=nx.erdos\_renyi\_graph(G.order(),4.2/G.order())  histDeg2=nx.degree\_histogram(EG)  plt.bar(range(len(histDeg2)),histDeg2) | Si on a le degré moyen de 4.2 par exemple et il nous demande de représenter le graphe.  #pour avoir un graphe de degré moyen de 4.2, il faut avoir p=4.2/(ordre-1)  #car ordre\*p=4.2  #plus spécifiquement ordre-1 car chaque sommet est connecte a ordre-1 autre sommet |
| def dijkstraBW(G,s):  dist={}  pred={}  for i in G.nodes():  dist[i]=0  pred[i]=s  dist[s]=np.inf  Y=list(G.nodes())  while Y:  i=max(Y,key=dist.get)  for j in G.neighbors(i):  if min(dist[i],G[i][j]['weight']) > dist[j]:  dist[j]=min(dist[i],G[i][j]['weight'])  pred[j]= i  Y.remove(i)  return dist,pred  Gbw=nx.read\_edgelist('./BW.txt',create\_using=nx.Graph(),nodetype=str,data=(('weight',int),))  [d,p]=dijkstraBW(Gbw,'S')  print(d)  print(p) | #dijkstra bande passante |
| cyclomatique=G.number\_of\_edges()-G.number\_of\_nodes()+nx.number\_connected\_components(G) | No idea what this is bass gonna check it later |
| EG=nx.erdos\_renyi\_graph(20,0.4)  print(EG.number\_of\_edges())  nx.draw(EG,with\_labels=True)  plt.show()  plt.close() | Graph randomly generated  #by default:directed=False;  #20 sommets  #20 sommets implique 400 paires de sommets  #(400\*p)/2 est le nombre moyen d’arêtes pour un graphe non oriente |
| EG=nx.erdos\_renyi\_graph(20,0.4,directed=True)  print(EG.number\_of\_edges())  #by default:directed=False;  #20 sommets  #20 sommets implique 400 paires de sommets  #(400\*p) est le nombre moyen d’arêtes pour un graphe oriente | #graphe randomly generated  (Directed) |

**Exemple supplémentaire:Programmation linéaire: Contraintes : x1>0 et x1<4 le code sera**

c=[2, 24, 13, 9, 20, 19] **#on minimize donc pas de signe (-), coeff de f(x)**

**#Méthode 1: Regular A matrix, les 6 premières lignes pour l contrainte x1…6>0 donc -x1…6<0 et les 6 autres lignes pour la seconde limite x1<4 par exemple donc coeff de x1 ici est 1 non pas -1:**

A=[[-1, 0, 0, 0, 0, 0],\

[0, -1, 0, 0, 0, 0],\

[0, 0, -1, 0, 0, 0],\

[0, 0, 0, -1, 0, 0],\

[0, 0, 0, 0, -1, 0],\

[0, 0, 0, 0, 0, -1],\

[1, 0, 0, 0, 0, 0],\

[0, 1, 0, 0, 0, 0],\

[0, 0, 1, 0, 0, 0],\

[0, 0, 0, 1, 0, 0],\

[0, 0, 0, 0, 1, 0],\

[0, 0, 0, 0, 0, 1],\

[-110, -205, -160, -160, -420, -260],\

[-4, -32, -13, -8, -4, -14],\

[-2, -12, -54, -285, -22, -80]]

**#Les limites x1…6>0 et x1<40**

b=[0, 0, 0, 0, 0, 0, 4, 3, 2, 3, 5, 2, -2000, -55, -800]

res1=linprog(c,A\_ub=A,b\_ub=b)

print('Methode 1:',res1)