Ressources nécessaires

Vous pouvez choisir le langage de votre choix. Les exemples du cours sont rédigés en **Python 3**. Si vous souhaitez les utiliser tels quels :

- Python 3.x (la distribution standard depuis le site officiel, ou miniconda)
- un IDE (thonny, pycharm, pyzo)

Biblio

CORMEN Thomas, LEISERSON Charles, RIVEST Ronald et STEIN Clifford, *Algorithmique*, 3e éd., Dunod, 2010

Un premier chiffrement

On effectue une permutation sur les lettres :

$$A \mapsto F$$
$$B \mapsto P$$
$$C \mapsto A$$
$$D \mapsto \ldots$$

Un premier chiffrement

Exercice

Rédigez les fonctions suivantes :

- `cipher(s)` *qui génère une clef aléatoire et code le texte* `s` *à partir de celle-ci.*

Un premier chiffrement

Exercice

Rédigez les fonctions suivantes :

- `cipher(s)` *qui génère une clef aléatoire et code le texte* `s` *à partir de celle-ci.*
- `kill(t,u)` *qui à partir d'un texte chiffré* `t` *retrouve le texte original* `s` *dont on connaît le court extrait* $u \subset s$.

Solution (1)


```
from random import shuffle
def cipher(s):
    key = [chr(i+65) for i in range(26)]
    shuffle(key)
    out = ""
    for c in s:
        ascii = ord(c)
        if ascii > 64 and ascii < 91:
            out += key[ascii-65]
        else:
            out += c
    return out

print(cipher("BONNE NUIT LES PETITS"))
```

Solution (2)


```
from random import shuffle
def kill(t,u):
    out, key = t, [chr(i+65) for i in range(26)]
    while u not in out:
        shuffle(key)
        out = ""
        for c in t:
            ascii = ord(c)
            if ascii > 64 and ascii < 91:
                out += key[ascii-65]
            else:
                out += c
    return out
print(kill("QJFFO_FMGX_ZOH_BOXGXH","PETITS"))
```

Analyse

Combien parviennent à faire tourner leur programme en moins d'une minute ?

Analyse

Combien parviennent à faire tourner leur programme en moins d'une minute ?

Il faut au pire des cas
26 ! = 403 291 461 126 605 635 584 000 000 essais.

Analyse

Combien parviennent à faire tourner leur programme en moins d'une minute ?

Il faut au pire des cas
26 ! = 403 291 461 126 605 635 584 000 000 essais.

Ce code est cependant très vulnérable à une attaque statistique (cf Edgar Poe)

Un meilleur chiffrement

CHAQUE FOIS QU UN HOMME
BVABVA BVAB VA BV ABVAB
EDBSGF … NG

Un meilleur chiffrement

Exercice

Rédigez les fonctions suivantes :

- `cipher2(s,k)` *qui génère une clef aléatoire de taille* `k` *et code le texte* `s` *à partir de celle-ci.*

Un meilleur chiffrement

Exercice

Rédigez les fonctions suivantes :

- `cipher2(s,k)` *qui génère une clef aléatoire de taille* `k` *et code le texte* `s` *à partir de celle-ci.*
- `kill2(t,k,u)` *qui à partir d'un texte chiffré* `t` *avec une clef inconnue de taille* `k` *retrouve le texte original* `s` *dont on connaît le court extrait* $u \subset s$.

Solution (1)


```
from random import randrange
def cipher2(s,k):
    key = [randrange(1,27) for i in range(k)]
    out = ""
    i = 0
    for c in s:
        out += chr((ord(c)-65+key[i])%26+65)
        i = (i+1)%k
    return out
print(cipher2("OHMONBATEAUQUILESTBEAU",4))
```

Solution (2)


```
from random import randrange
def kill2(t,k,u):
    out = t
    while not u in out :
        key = [randrange(1,27) for i in range(k)]
        out,i = "",0
        for c in t:
            out += chr((ord(c)-65+key[i])%26+65)
            i = (i+1)%k
    return out
print(kill2("LBJNKVXSBURPRCIDPNYDXO",4,"ESTBEAU"))
```

Analyse

Prenez une phrase `s` assez longue (cent caractères). A partir de quelle valeur de `k` le programme met-il trop de temps à répondre ?

Analyse

Prenez une phrase `s` assez longue (cent caractères). A partir de quelle valeur de `k` le programme met-il trop de temps à répondre ?

La complexité du programme est 26^k donc il est peu probable que vous puissiez dépasser une dizaine de caractères pour la clef.

Bilan

La complexité d'un problème peut être une garantie de sécurité en cryptanalyse. Cependant, dans presque tous les autres domaines, c'est surtout une donnée à maîtriser pour rendre un algorithme efficace.

Tout au long de ce module nous allons voir qu'une bonne connaissance de l'algorithmique est infiniment plus efficace que le choix d'un langage ou une maîtrise parfaite de sa syntaxe.

Mais d'abord revenons sur un dernier exemple de cryptographie, réellement utilisé aujourd'hui celui-là.

RSA

a) On pose `p = 8191` et `q = 127`. Calculez `phi = (p-1)(q-1)`
b) Trouvez deux nombres entiers `a,b` tels que :

- `a` est premier avec `phi`,
- `a*b % phi = 1`.

c) Traduisez une lettre sous forme de nombre, puis chiffrez le nombre obtenu en le passant à l'exposant `a` modulo `n`.
d) Vérifiez qu'on peut bien déchiffrer le nombre obtenu en le passant à l'exposant `b` modulo `n`, puis retrouvez la lettre originale.

Solution (1)


```
from math import gcd
from random import randrange
#a,b)
p,q = 8191,127
phi = (p-1)*(q-1)
n = p*q
a = randrange(1,phi)
while not (gcd(a,phi) == 1):
    a += 1 %phi
r,b,v,r2,b2,v2 = a,1,0,phi,0,1
while not (r2 == 0):
    q = r//r2
    r,b,v,r2,b2,v2 = r2,b2,v2,r-q*r2,b-q*b2,v-q*v2
while b < 0:
    b += phi
```

Solution (2)


```
print(a,b)

#c)
c = 'H'
x = ord(c)**a % n
print(x)

#d)
y = chr(x**b % n)
print(y)
```

Exemple introductif


```
def factorec(n):
    return factorec(n-1)*n if n>1 else 1
print(factorec(6))
```

Warning I : non-terminaison


```
def badrec(x):
    return badrec(x/2) if x > 0 else 1
print(badrec(3))
```

Exponentielle rapide


```
def fastexp(a,b):
    if b == 0:
        return 1
    if b%2 == 0:
        return fastexp(a,b//2)**2
    else:
        return a*fastexp(a,b//2)**2
```

Exponentielle rapide

Exercice

Implémentez l'exponentielle rapide et comparez son temps d'exécution à celui d'une exponentielle définie par une simple séquence de multiplications $2^a = 2 \times 2 \times \ldots \times 2$ *(a fois).*

Quelle est la valeur maximale de *a* que vous pouvez calculer dans chacun des cas ?

Warning II : explosion

Exercice

Proposez une fonction récursive qui calcule le 100e terme de la suite de Fibonacci.

Warning II : explosion


```
def fibonacci(n):
    (a,b) = (1,1) if n == 1 else fibonacci(n-1)
    return b, a+b

print(fibonacci(100)[0])
```

ET SURTOUT PAS :

```
def fibonacci(n):
    x = 1 if n <= 1 else fibonacci(n-1)+fibonacci(n-2)
    return x

print(fibonacci(100))
```

Sac à dos

Le problème du sac à dos est de trouver comment remplir de façon optimale un sac avec des items de volumes et de valeurs différents.

Sac à dos

Le problème du sac à dos est de trouver comment remplir de façon optimale un sac avec des items de volumes et de valeurs différents.

On s'intéresse à une variante simplifiée : vérifier s'il existe une façon de prendre un sous-ensembles d'items de volume total exactement W.

Sac à dos

Exercice

Proposez un programme récursif `sacados(liste,i)` qui vérifie s'il existe une sous-liste de la liste `liste` dont la somme vaut `i`.

Sac à dos


```
def sacados(liste,i) :
    if len(liste) == 0:
        return i == 0
    else :
        return (sacados(liste[1:],i)
                or sacados(liste[1:],i-liste[0]))
print(sacados([2, 7, -1, 9],16))
print(sacados([2, 7, -1, 9],4))
```

Sac à dos

Il est aussi possible de faire en non-récursif, par exemple en utilisant la bijection entre 2^X et $[0, 2^{|X|}]$ - c'est-à dire l'écriture binaire.

Parfois la récursivité...

Exercice

Pour les deux programmes suivants :

- *expliquez ce qu'ils font (fonctionnellement)*
- *comparez leur temps de calcul*
- *détaillez pas à pas leur exécution*

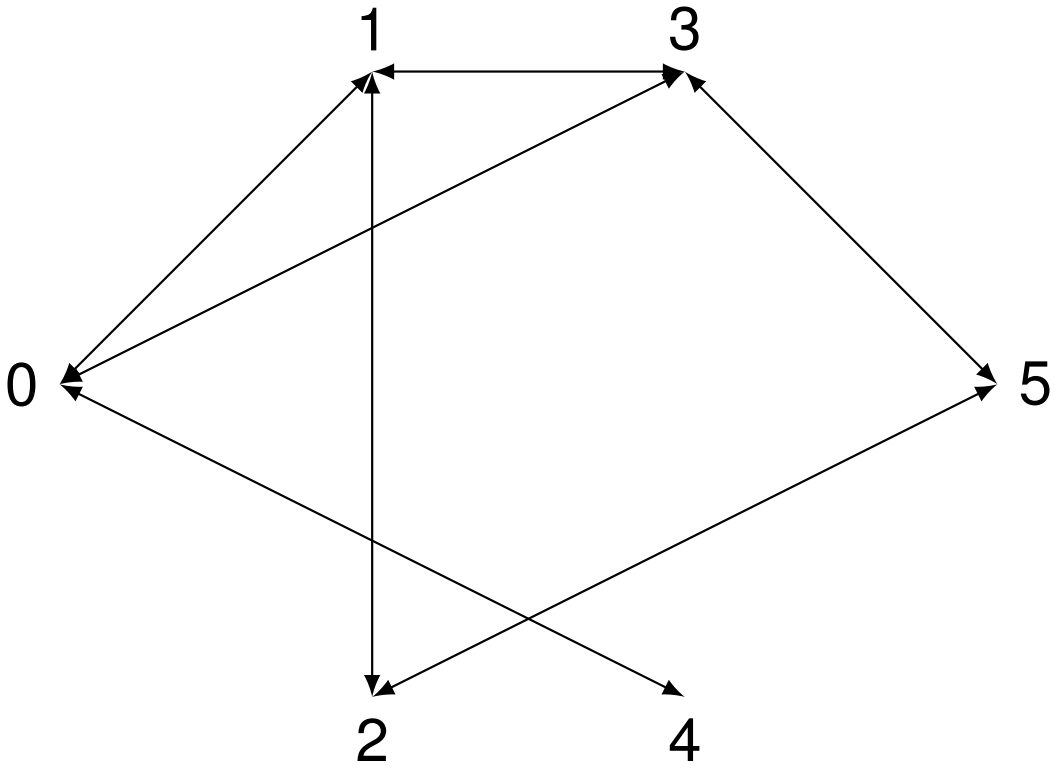
Parfois la récursivité...


```
def f(n,m):
    if n == 0 or m == 0:
        return n+m
    return f(n-1,m) + f(n,m-1)
print(f(30,30))
```

Parfois la récursivité...


```
m = 30
n = 30
liste = [[0 for i in range(m+1)] for j in range(n+1)]
liste[0] = [i for i in range(m+1)]
for j in range(1,n+1):
    liste[j][0] = j
    for i in range(1,m+1):
        liste[j][i] = liste[j-1][i] + liste[j][i-1]
print(liste[30][30])
```

Plus court chemin



Plus court chemin


```
voisines = [ [1,3,4], [0,2,3] , [1,5] ,
             [0,1,5] , [0] , [2,3] ]
chemins = [[[0]]]
for i in range(1,6):
    chemins.append( [c+[v] for c in chemins[i-1]
                    for v in voisines[c[-1]]] )
print(chemins[1])
print(chemins[2])
```

Plus court chemin


```
voisines = [ [1,3,4], [0,2,3] , [1,5] ,
             [0,1,5] , [0] , [2,3] ]
chemins = [[0]]
arrive, i = [], 0
while arrive == []:
    i += 1
    chemins.append( [c+[v] for c in chemins[i-1]
                    for v in voisines[c[-1]]] )
    arrive = [c for c in chemins[i]
              if c[-1] == 5]
print(arrive)
```

Plus court chemin

Exercice

Modifiez le programme précédent de façon à ce qu'il n'énumère que les chemins qui ne font pas d'aller-retours ou de boucles.

Plus court chemin


```
voisines = [ [1,3,4], [0,2,3] , [1,5] ,
             [0,1,5] , [0] , [2,3] ]
chemins = [[[0]]]
arrive, i = [], 0
while arrive == []:
    i += 1
    chemins.append( [c+[v] for c in chemins[i-1]
        for v in voisines[c[-1]] if not v in c] )
    arrive = [c for c in chemins[i]
              if c[-1] == 5]
print(arrive)
print(chemins)
```

Plus court chemin


```
voisines = [ [1,3,4], [0,2,3] , [1,5] ,
             [0,1,5] , [0] , [2,3] ]
def chemin(k,maxl):
    if maxl == 0 :
        trouve = (k == 0)
    else :
        trouve = any(chemin(i,maxl-1)
                     for i in voisines[k])
    if trouve :
        print(k)
    return trouve

chemin(5,2)
```

Plus court chemin


```
voisines = [ [1,3,4], [0,2,3] , [1,5] ,
             [0,1,5] , [0] , [2,3] ]
plusCourt = [None]*6
plusCourt[0] = [0]
for step in range(1,6):
    for ville in range(1,6):
        if plusCourt[ville] == None:
            for v in voisines[ville]:
                c = plusCourt[v]
                if c != None and len(c) == step:
                    plusCourt[ville] = c+[ville]
print(plusCourt[5])
```

Plus court chemin

Exercice

Modifiez le programme précédent de façon à ce qu'il renvoie l'ensemble des plus courts chemins et non un seul.

Plus court chemin


```
voisines = [ [1,2,3,4], [0,2,3] , [0,1,5] ,
             [0,1,5] , [0] , [2,3] ]
plusCourts = [[] for i in range(6)]
plusCourts[0] = [[0]]
for step in range(1,6):
    for ville in range(1,6):
        if plusCourts[ville] == []:
            for v in voisines[ville]:
                c = plusCourts[v]
                if c != [] and len(c[0]) == step:
                    for x in c:
                        plusCourts[ville].append(
                            x+[ville])
print(plusCourts[5])
```

Linéarité

Exercice

Vérifiez sur quelques exemples que le temps nécessaire pour effectuer une séquence de multiplications d'entiers croît proportionnellement avec la taille de la séquence.

Linéarité

```
from time import time
for i in range(10):
    t0 = time()
    for j in range(10**i):
        x = 2*3
    print(i,time()-t0)
```

Polynomialité

Exercice

Vérifiez que pour la multiplication de deux matrices, le temps augmente plus vite. Selon quelle loi ?

Instrument de mesure

La fonction time() reste limitée :

- Dépendance à l'environnement
- Pas idéal pour scaler

Instrument de mesure

La fonction time() reste limitée :

- Dépendance à l'environnement
- Pas idéal pour scaler

On veut plutôt compter le nombre d'opérations élémentaires.

Sous-linéarité

Exercice

Avec une variable compteur, comparez le nombre d'opérations effectuées dans une exponentielle rapide vs une exponentielle naïve.

Sous-linéarité


```
def fastexp(a,b):
    if b == 0:
        return (1,0)
    (ex,c) = fastexp(a,b//2)
    if b%2 == 0:
        return (ex**2,c+1)
    else:
        return (a*ex**2,c+1)
print(fastexp(0.999999,10**8)[1])
f = 1
for i in range(1,10**8):
    f *= 0.999999
print(i)
```

Exemples d'algorithmes

Exponentielle rapide : $O(\log n)$
Tri rapide, exponentielle naïve : $O(n)$
Tri par insertion : $O(n^2)$
Multiplication matricielle naïve : $O(n^3)$
Énumération des sous-ensembles : $O(2^n)$
Voyageur de commerce, Coloration : $O(2^n)$
Énumération des permutations : $O(n!)$

Ordres de grandeur

Taille	$n \log n$	n^3	2^n
$n = 20$	60	8000	1048576
$n = 50$	196	125000	1125899907000000
$n = 100$	461	1000000	1267650600000000000000000000

$\Longrightarrow$ Intuition d'une frontières entre algorithmes polynomiaux et algorithmes exponentiels.

Méthodes de calcul

- des blocs successifs ont leur complexité qui s'additionne.
- une boucle a une complexité égale à la somme de ses itérations.
- en particulier, une boucle de taille constante a une complexité égale à sa taille multipliée par la complexité d'une itération.

Méthodes de calcul

Exercice

Calculez la complexité de l'algorithme suivant :

```
p = 100
for i in range(p):
    for j in range(i):
        l = [i+j+k for k in range(p)]
```

Méthodes de calcul

Exercice

Calculez la complexité de l'algorithme suivant :

```
def sumdivs(n):
    return sum([i for i in range(1,n) if n%i == 0])

p = 1000
amic_num = [-1 for i in range(p)]
for i in range(p):
    if amic_num[i] == -1:
        j = sumdivs(i)
        if i != j and sumdivs(j) == i:
            amic_num[i] = i
            amic_num[j] = j
        else :
            amic_num[i] = 0
            if j < 1000 and amic_num[j] == -1:
                amic_num[j] = 0
print([i for i in amic_num if i>0])
```

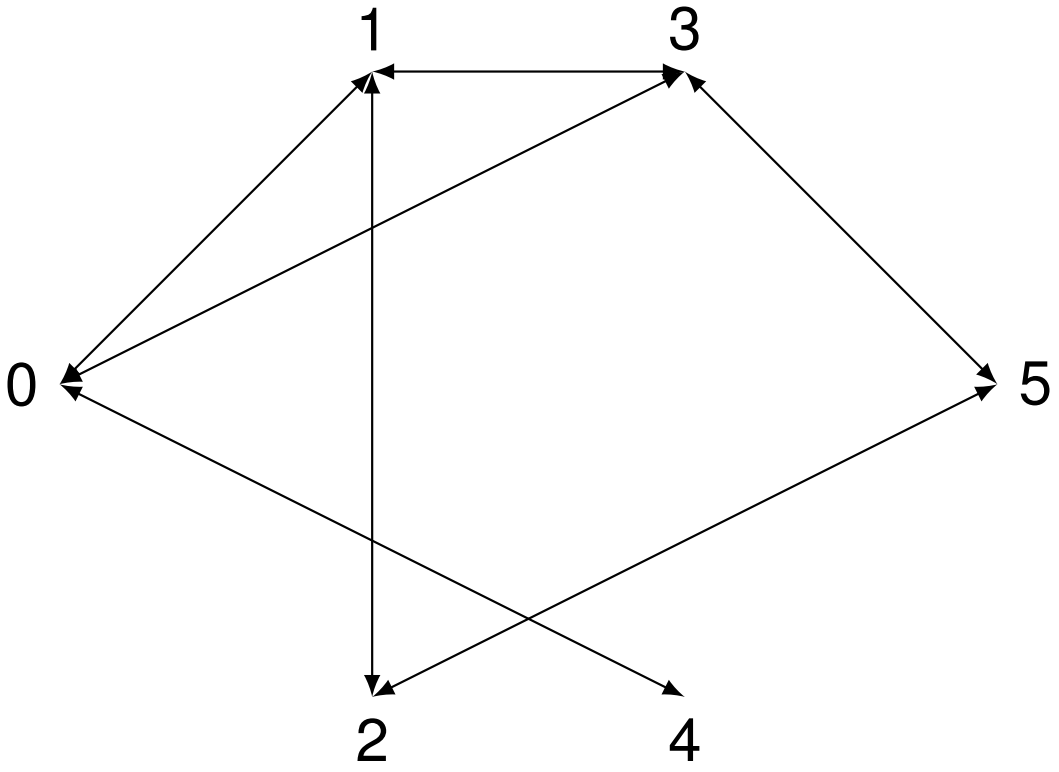
Méthodes de calcul

Exercice

Calculez la complexité de l'algorithme suivant :

```
from math import log
def dec2bin(n):
    m = int(log(n)/log(2))
    liste = [0 for i in range(m+1)]
    for i in range(m+1):
        liste[i] = n%(2**(i+1))//(2**i)
    return liste
def enum_bin(p):
    for n in range(1,p):
        print(dec2bin(n))
```

Dominating set



Dominating set

Étant donné un graphe $G(V, E)$, on cherche un sous-ensemble D de taille minimale tel que $N[D] = V$.

Dominating set

Exercice

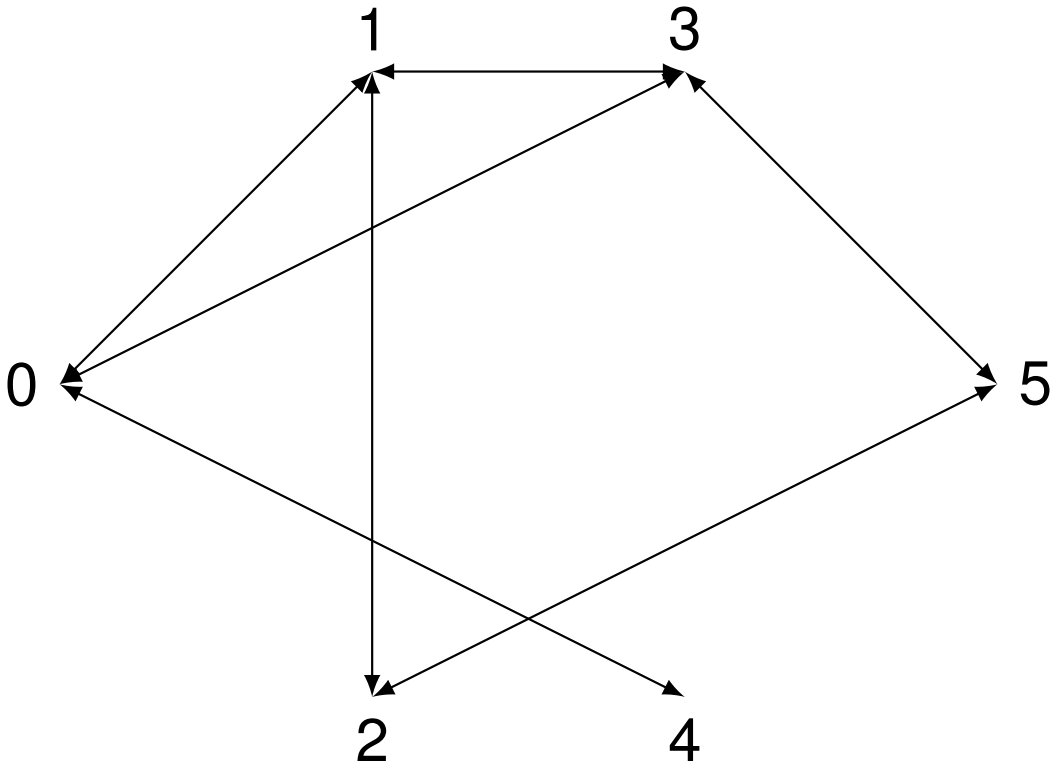
Proposez un algorithme qui trouve un Dominating Set

Dominating set

Exercice

Trouvez un exemple dans lequel votre algorithme n'est pas optimal

Coloring



Coloring

Étant donné un graphe $G(V, E)$, on cherche une partition de V en un nombre minimum de stables. Ce nombre $\chi(G)$ est appelé nombre chromatique du graphe et la partition une coloration minimale.

Coloring

Exercice

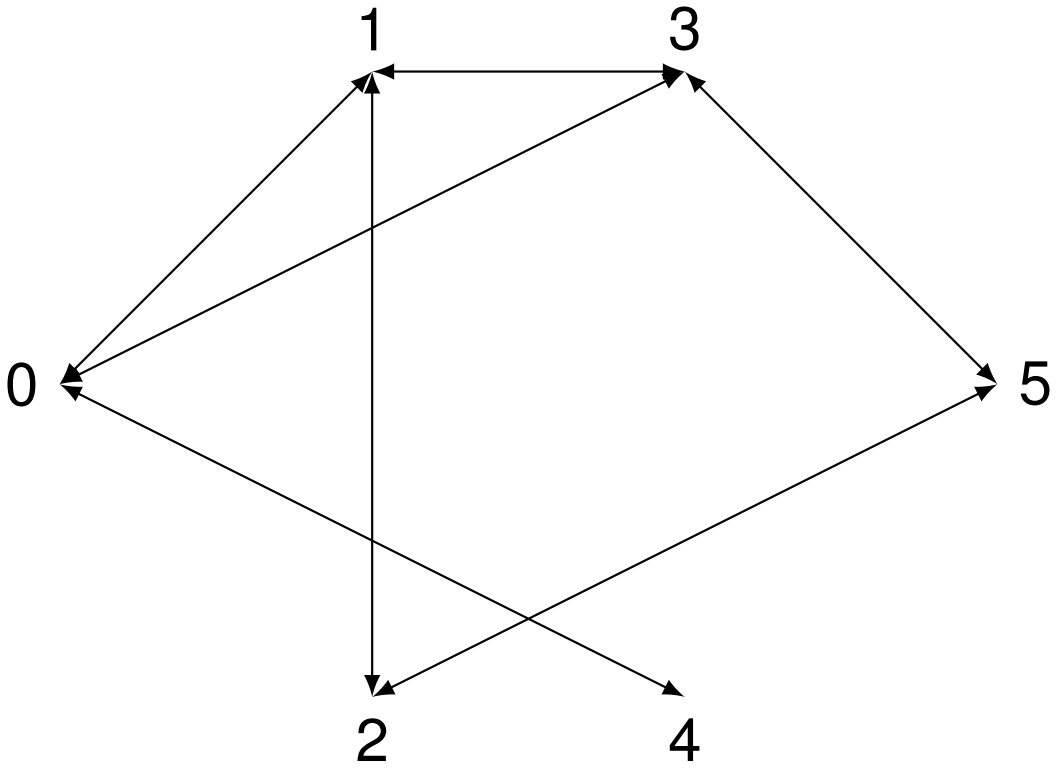
Proposez un algorithme qui trouve une Coloration

Coloring

Exercice

Trouvez un exemple dans lequel votre algorithme n'est pas optimal

TSP



TSP

Étant donné un graphe $G(V, E)$, on cherche une permutation σ sur V telle que pour tout $i \in V$, $(\sigma(i), \sigma(i+1)) \in E$.

TSP

Exercice

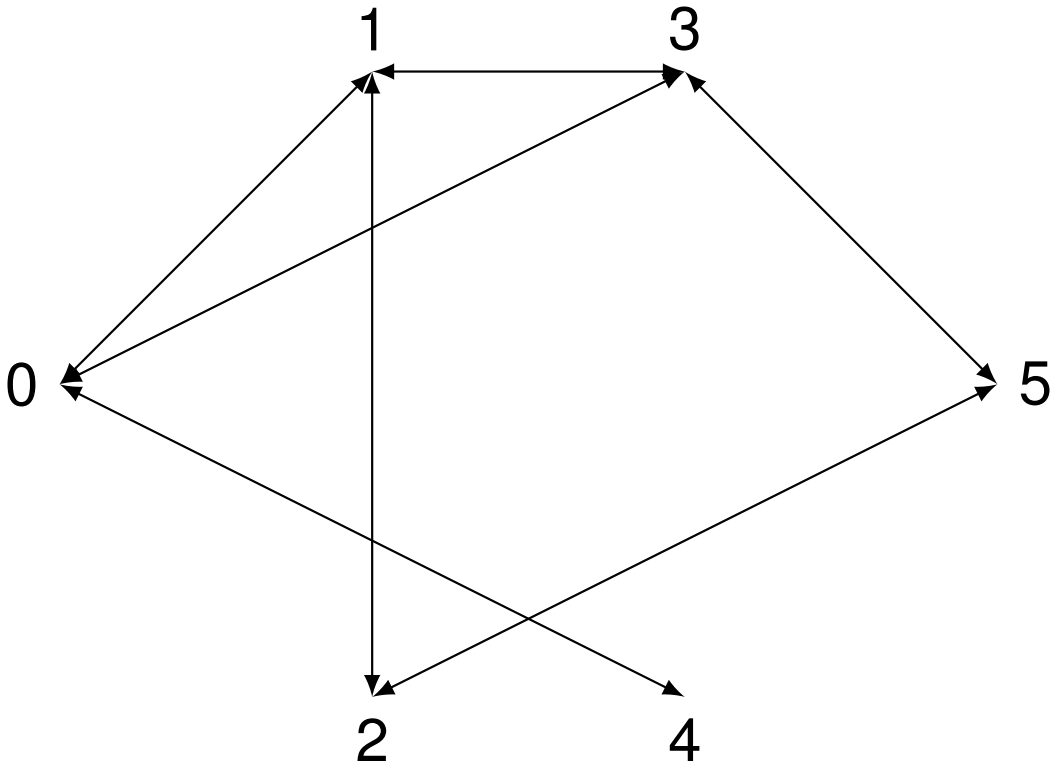
Proposez un algorithme qui trouve un chemin hamiltonien

TSP

Exercice

Pourquoi votre algorithme ne compile-t-il pas ?

Independent Set



Independent Set

Étant donné un graphe $G(V, E)$, on cherche un sous-graphe totalement déconnecté (également appelé ensemble stable) de taille maximale.

Independent Set

Exercice

Proposez un algorithme qui trouve un Independent Set

Independent Set

Exercice

Trouvez un exemple dans lequel votre algorithme n'est pas optimal

Independent Set et Branching

$$\alpha(V) = \max\{\alpha(V \setminus \{v\}), \alpha(V \setminus N[v]) + 1\}$$

TSP et Programmation Dynamique

On reformule ainsi le TSP :

$$P(U,i) = argmin_{\sigma \in S_U} \left(\sum_{j \in U \setminus \{i\}} w(\sigma(j), \sigma(j+1)) \right)$$

D'où la récurrence suivante :

$$P(U,i) = \min_{j \in U} \left(P(U \setminus \{i\}, j) + w(e_{ij}) \right)$$