

Informatique et Programmation

Cours 9

Jean-Jacques Lévy

`jean-jacques.levy@inria.fr`

`http://jeanjacqueslevy.net/prog-py`

Plan

- algorithme glouton (**Dijkstra**)
- algorithme glouton (marche du cavalier)
- exploration exhaustive (sac à dos)
- exploration exhaustive (les 8 reines)
- programmation dynamique (plus courts chemins, **Warshall**)
- programmation dynamique (plus longue chaîne commune)

dès maintenant: **télécharger Python 3 en** `http://www.python.org`

Recap

- mots clés en Python (déjà vus en rouge)

```
>>> help()
help> keywords
```

Here is a list of the Python keywords. Enter any keyword to get more help.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

Python ++

- rupture de boucle

- break arrête la boucle
- continue saute à l'itération suivante

```
for i in range (10):  
    if i * 2 == 4 :  
        break  
    print (i, end = " ")
```

➡ 0 1

```
for i in range (10):  
    if i * 2 == 4 :  
        continue  
    print (i, end = " ")
```

➡ 0 1 3 4 5 6 7 8 9

- instruction vide

- pass ne fait rien

```
if 4 * 2 == 4 :  
    pass  
else :  
    print ("8 is not 4")
```

➡ 8 is not 4

Python ++

- traitement des exceptions

- try: début d'un bloc avec exception possible
- except IOError: récupère l'exception IOError
- except: récupère toutes les exceptions
- finally: pour le traitement normal **et** le traitement exceptionnel

- alias de module

- as déclare un alias pour un module (par exemple si le nom est trop long)

```
def lire_lignes (nom) :  
    try:  
        f = open (nom, 'r')  
        return f.read().splitlines()  
    except IOError:  
        print("Fichier '%s' inexistant." % nom)  
lire_lignes('abc')
```

➡ Fichier 'abc' inexistant.

```
import random as r  
r.choice (['a', 'b', 'c'])
```

➡ 'a'

Recap

- mots clés en Python (déjà vus en rouge)

```
>>> help()
help> keywords
```

Here is a list of the Python keywords. Enter any keyword to get more help.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

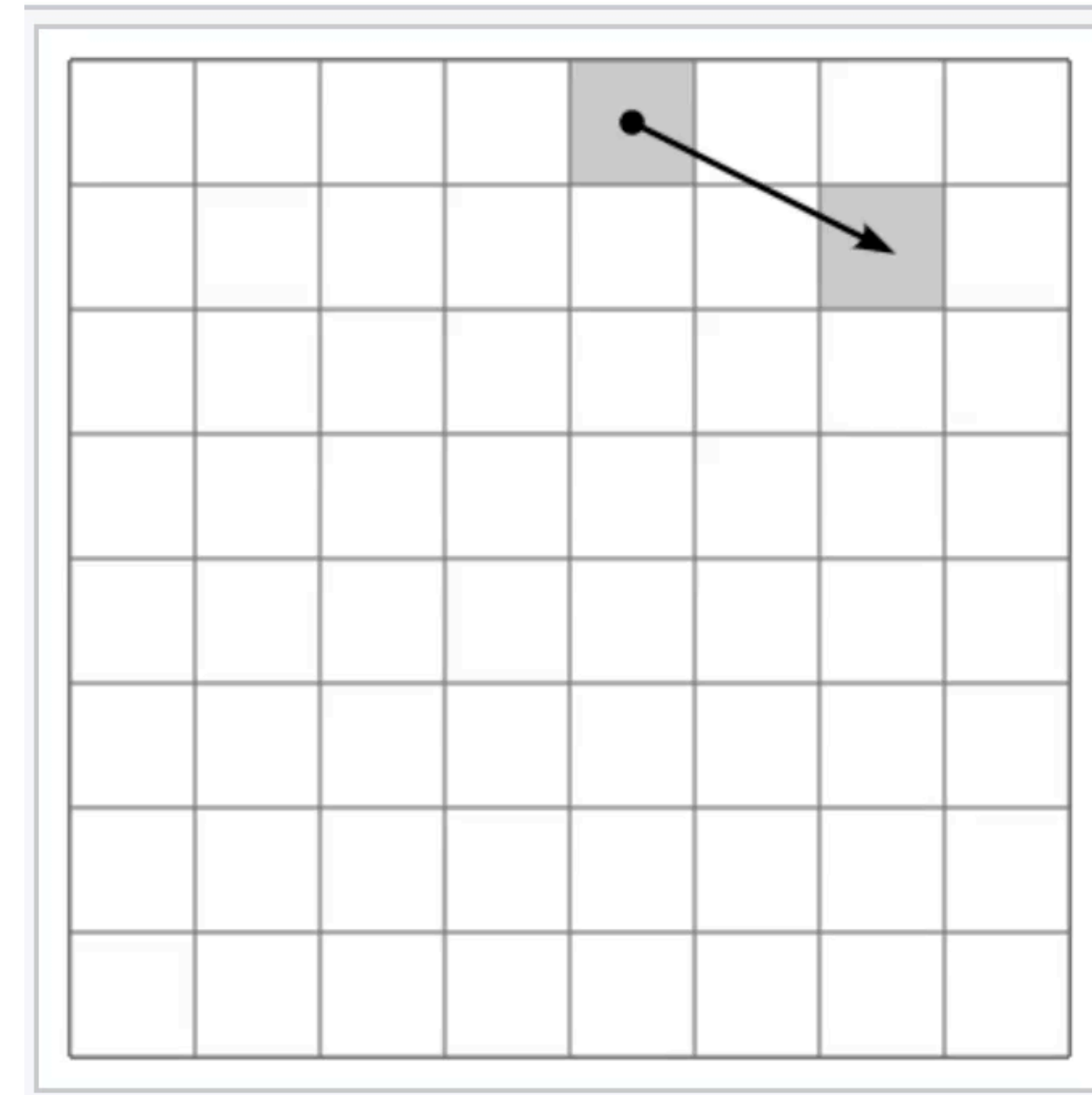
Exploration

on distingue 3 méthodes d'exploration

- algorithmes gloutons
 - un choix local permet d'obtenir la solution globale
- exploration exhaustive
 - on parcourt les solutions globales jusqu'à trouver la bonne solution
 - retours arrière possibles (*backtracking*)
- programmation dynamique
 - on mémorise tous les résultats partiels pour obtenir la solution globale
 - demande de la mémoire supplémentaire

Algorithme glouton

- plus court chemin dans un graphe (**Dijkstra**)
[on cherche le minimum local — cf cours 8]
- arbre de recouvrement minial dans un graphe non-orienté valué (**Kruskal**, **Prim**)
- allocation de ressources, etc
- marche du cavalier pour couvrir toutes les cases d'un échiquier



Marche du cavalier

- marche du cavalier pour couvrir toutes les cases d'un échiquier
- on se déplace vers la case où il y aura le moins de déplacements possibles pour le cavalier

```
LIBRE = -1
```

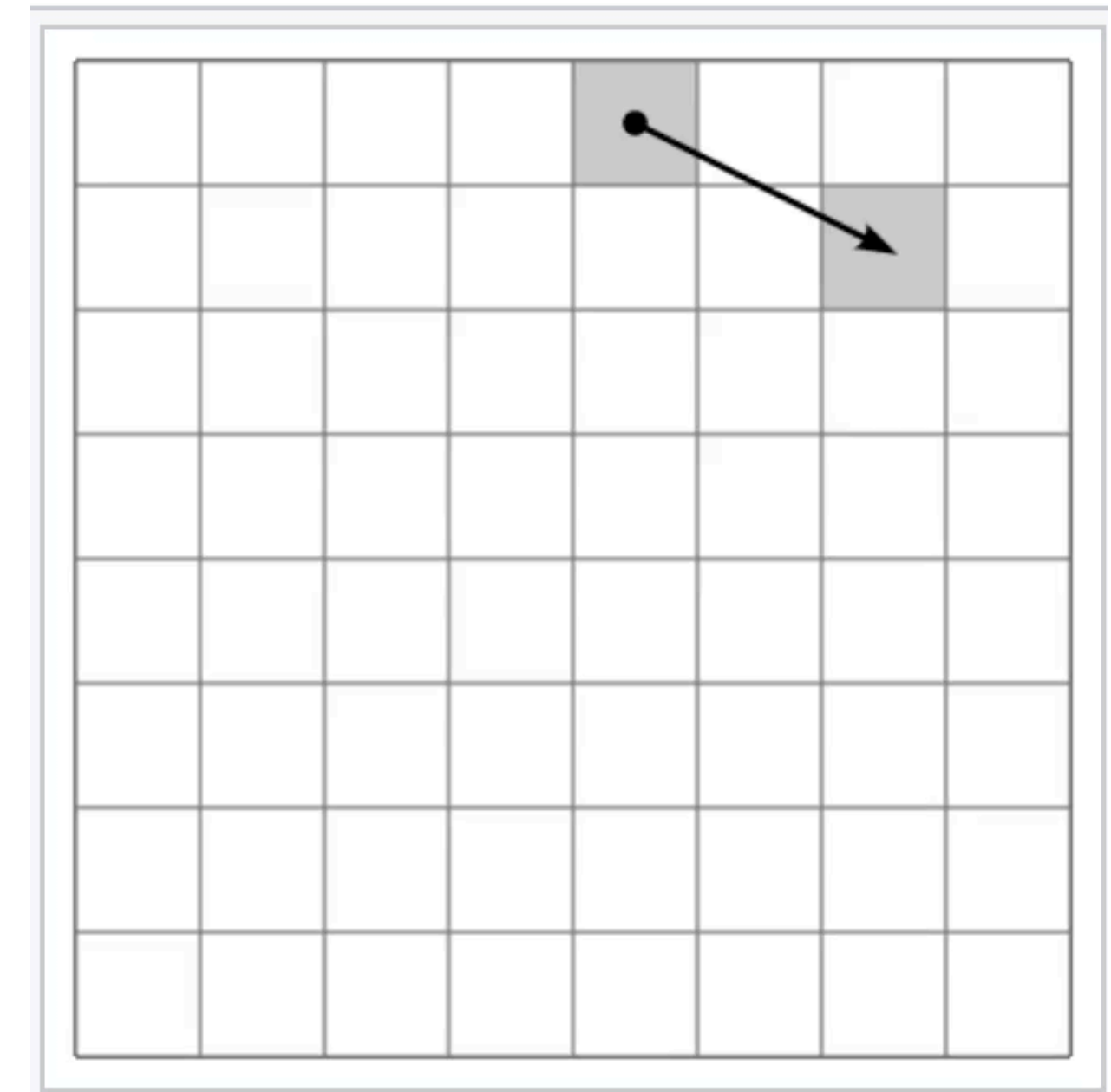
```
dX = [ 2, 1, -1, -2, -2, -1, 1, 2 ]
```

```
dY = [ 1, 2, 2, 1, -1, -2, -2, -1 ]
```

```
def marche (a, i, j, k) :  
    a[i][j] = k  
    p = caseMinCoupsJouables (a, i, j)  
    if p == (i, j) :  
        return k  
    else:  
        return marche (a, p[0], p[1], k+1)
```

```
def une_solution (n, i, j) :  
    a = new_matrix (n, n, LIBRE)  
    if marche (a, i, j, 1) < n*n :  
        print ("pas de solution!")  
    print_matrix (a)
```


une_solution (8, 0, 4)



47	14	61	32	1	16	19	34
64	31	46	15	60	33	2	17
13	48	57	62	45	18	35	20
30	63	42	53	56	59	40	3
49	12	55	58	41	44	21	36
26	29	52	43	54	39	4	7
11	50	27	24	9	6	37	22
28	25	10	51	38	23	8	5

Marche du cavalier

- marche du cavalier pour couvrir toutes les cases d'un échiquier
- on se déplace vers la case où il y aura le moins de déplacements possibles pour le cavalier

```
LIBRE = -1
dX = [ 2, 1, -1, -2, -2, -1, 1, 2 ]
dY = [ 1, 2, 2, 1, -1, -2, -2, -1 ]

def nbDeCoupsJouables (a, i, j) :
    if not coupJouable (a, i, j) :
        return infini
    r = 0
    for k in range (len(dX)) :
        if coupJouable (a, i + dX[k], j + dY[k]) :
            r = r + 1
    return r
```

```
def coupJouable (a, i, j) :
    return 0 <= i < len(a) and \
           0 <= j < len(a[0]) and \
           a[i][j] == LIBRE

def caseMinCoupsJouables (a, i, j) :
    i1 = i; j1 = j;
    min = infini
    for k in range(len(dX)) :
        nk = nbDeCoupsJouables (a, i+dX[k], j+dY[k])
        if nk < min :
            i1 = i+dX[k]; j1 = j + dY[k]
            min = nk
    return (i1, j1)
```

Recherche exhaustive

- problème du sac à dos (ranger le maximum d'objets dans un sac)
- voyageur de commerce et tous les problèmes NP
- les 8 reines (placer 8 reines sur un échiquier sans qu'elle ne soit en prise par une autre reine)

```
def conflit (i1, j1, i2, j2) :  
    return (i1 == i2) or (j1 == j2) or \  
        (abs (i1 - i2) == abs (j1 - j2))
```



teste si la reine en (i2, j2) peut
prendre la reine en (i1, j1)

	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

Les 8 reines

- les 8 reines (placer 8 reines sur un échiquier sans qu'elle ne soit en prise par une autre reine)
on explore les solutions avec possibles retours arrière (*backtracking*)

```
def compatible (i, j, pos) :  
    for k in range (i) :  
        if conflit (k, pos[k], i, j) :  
            return False  
    return True
```

```
def reines (n, i, pos) :  
    if i >= n :  
        imprimerSolution (pos);  
    else :  
        for j in range (n) :  
            if compatible (i, j, pos) :  
                pos[i] = j  
                reines (n, i+1, pos)
```

```
def nReines (n) :  
    pos = n * [0]  
    reines (n, 0, pos)
```

ici *backtracking* si pas de solution à partir de ligne $i+1$ (on passe alors à la colonne suivante)

	0	1	2	3	4	5	6	7
0	♔							
1					♔			
2								♔
3						♔		
4			♔					
5							♔	
6		♔						
7				♔				

pos == [0, 4, 7, 5, 2, 6, 1, 3]

Les 8 reines

- impression de la solution

```
def print_matrix (a) :  
    for line in a :  
        for elt in line :  
            print (elt, end = ' ' )  
        print ()  
  
def new_matrix (m, n, v) :  
    a = [ ]; z = n * [v]  
    for i in range(m): a.append (z.copy())  
    return a  
  
def imprimerSolution (pos) :  
    n = len (pos)  
    a = new_matrix (n, n, ".")  
    for i in range (n) :  
        a[i][pos[i]] = "R"  
    print_matrix (a)
```



	0	1	2	3	4	5	6	7
0	♔							
1					♔			
2								♔
3						♔		
4			♔					
5							♔	
6		♔						
7				♔				

pos == [0, 4, 7, 5, 2, 6, 1, 3]

```
R . . . . . . .  
. . . . R . . .  
. . . . . . . R  
. . . . . R . .  
. . R . . . . .  
. . . . . R .  
. R . . . . .  
. . . R . . . .
```

Programmation dynamique

- plus courts chemins entre tous les noeuds d'un graphe (Warshall)
- plus longue sous-chaîne de caractères commune entre 2 chaînes (diff)
- algorithme de Warshall (le graphe est représenté par une matrice d'adjacence)
le résultat est la matrice des plus courtes distances (n^3 opérations)

```
import sys; infini = sys.maxsize

def chemins_les_plus_courts (G) :
    n = len (G)
    r = copy.deepcopy(G)
    for k in range(n) :
        for i in range(n) :
            for j in range(n):
                r[i][j] = min (r[i][j],  r[i][k] + r[k][j])
    return r
```

Exercice Modifier la fonction pour retourner tous les chemins de distance minimale.

Programmation dynamique

- plus longue sous-séquence commune entre 2 chaînes de caractères (commande Unix diff)

[on mémorise les solutions partielles — $m \times n$ opérations]

```
GAUCHE = 1
HAUT = 2
DIAG = 3
```

```
def new_matrix (m, n, v) :
    a = [ ]; z = n * [v]
    for i in range(m): a.append (z.copy())
    return a
```

```
def longueurSSC (u, v) :
    m = len(u); n = len(v)
    lg = new_matrix (m+1, n+1, 0)
    p = new_matrix (m+1, n+1, 0)
    for i in range(1, m+1):
        for j in range(1, n+1):
            if u[i-1] == v[j-1]:
                lg[i][j] = 1 + lg[i-1][j-1]
                p[i][j] = DIAG;
            elif lg[i][j-1] > lg[i-1][j]:
                lg[i][j] = lg[i][j-1]
                p[i][j] = GAUCHE
            else :
                lg[i][j] = lg[i-1][j]
                p[i][j] = HAUT
    return (lg[m][n], p)
```

u = 'abcadefg'

v = 'fbcexyg'

lg

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1
0	0	1	2	2	2	2	2
0	0	1	2	2	2	2	2
0	0	1	2	2	2	2	2
0	0	1	2	3	3	3	3
0	1	1	2	3	3	3	3
0	1	1	2	3	3	3	4

p

0	0	0	0	0	0	0	0
0	2	2	2	2	2	2	2
0	2	3	1	1	1	1	1
0	2	2	3	1	1	1	1
0	2	2	2	2	2	2	2
0	2	2	2	2	2	2	2
0	2	2	2	3	1	1	1
0	3	2	2	2	2	2	2
0	2	2	2	2	2	2	3

Programmation dynamique

- plus longue sous-séquence commune entre 2 chaînes de caractères (commande Unix diff)

[on mémorise les solutions partielles — $m \times n$ opérations]

```
def ssc (u, v) :  
    m = len(u); n = len(v)  
    lg_p = longueurSSC (u, v)  
    lg = lg_p[0]; p = lg_p[1]  
    r = ''; i = m; j = n;  
    while lg > 0 :  
        if p[i][j] == DIAG :  
            r = u[i-1] + r  
            i = i - 1; j = j - 1;  
            lg = lg - 1  
        elif p[i][j] == GAUCHE :  
            j = j - 1  
        else :  
            i = i - 1  
    return r
```

```
>>> ssc ('abcadefg', 'fbcexyg')  
'bceg'
```

u = 'abcadefg'

lg								
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1
0	0	1	2	2	2	2	2	2
0	0	1	2	2	2	2	2	2
0	0	1	2	2	2	2	2	2
0	0	1	2	3	3	3	3	3
0	1	1	2	3	3	3	3	3
0	1	1	2	3	3	3	4	

v = 'fbcexyg'

p								
0	0	0	0	0	0	0	0	0
0	2	2	2	2	2	2	2	2
0	2	3	1	1	1	1	1	1
0	2	2	3	1	1	1	1	1
0	2	2	2	2	2	2	2	2
0	2	2	2	2	2	2	2	2
0	2	2	2	3	1	1	1	1
0	3	2	2	2	2	2	2	2
0	2	2	2	2	2	2	2	3

à faire

- analyses lexicale et syntaxique
- modularité et programmation objet
- programmation graphique
- algorithmes géométriques
- calculs flottants et méthodes numériques
- programmation de plusieurs fils de calcul
- assertions et logique des programmes
- introduction à l'informatique théorique
- etc

vive l'informatique

et

la programmation !