

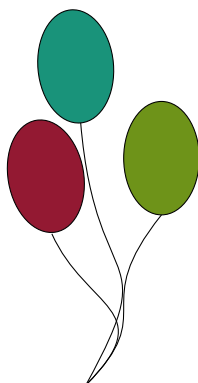
SWERC NoteBook

Équipe SaintGermainDesPrés

Mathilde BONIN, Eyal COHEN, Hugo DEMARET

Avril 2022

Ensemble d'algorithmes et techniques de programmation



Université Paris Cité
UFR de Mathématiques-Informatique
2021-2022

1 Configuration

1.1 C/C++

2 Chaînes de caractères

3 Séquences

4 Parcours de graphes

4.1 DFS - Depth First Search

```
1 #version iterative pour eviter la recursion limit de python
2 def dfs_iterative(graph,start,seen):
3     seen[start] = True
4     to_visit = [start]
5     while to_visit:
6         node = to_visit.pop()
7         for neighbour in graph[node]:
8             if not seen[neighbour]:
9                 seen[neighbour] = True
10                to_visit.append(neighbour)
```

4.2 BFS - Breadth First Search

```
1 from collections import deque
2 def bfs(graph, start=0):
3     to_visit = deque()
4     dist = [float('inf')] * len(graph)
5     prec = [None] * len(graph)
6     dist[start] = 0
7     to_visit.appendleft(start)
8     while to_visit: #evaluer a faux si vide
9         node = to_visit.pop()
10        for neighbour in graph[node]:
11            if dist[neighbour] == float('inf'):
12                dist[neighbour] = dist[node] + 1
13                prec[neighbour] = node
14                to_visit.appendleft(neighbour)
15    return dist, prec
```

- 4.3 Topological Sort
- 4.4 Composantes connexes
- 4.5 Composantes bi-connexe
- 4.6 Composantes fortement connexe
- 4.7 2-SAT
- 4.8 Postier Chinois
- 4.9 Chemin eulérien
- 4.10 Chemin le plus court
 - 4.10.1 Poids positif ou nul - Dijkstra
 - 4.10.2 Poids arbitraire - Bellman-Ford
 - 4.10.3 Floyd-Warshall

5 Points et polygones

5.1 Points

5.1.1 Points

```
1 point = [x,y]
```

5.1.2 Cross-product

```
1 def cross_product(p1, p2):
2     return p1[0] * p2[1] - p2[0] * p1[1]
```

5.1.3 Direction

```
1 def left_turn(a,b,c):
2     return (a[0]-c[0]) * (b[1]-c[1]) - (a[1]-c[1]) * (b[0]-c[0]) > 0
3     # If floats are used, instead of 0 test if in [0-10E-7,0+10E-7]
```

5.2 Enveloppe convexe

Complexité : $\mathcal{O}(n \log(n))$

```
1 def andrew(S):
2     S.sort()
3     top = []
4     bot = []
5     for p in S:
6         while len(top) >= 2 and not left_turn(p,top[-1],top[-2]):
7             top.pop()
8         top.append(p)
9         while len(bot) >= 2 and not left_turn(bot[-2],bot[-1],p):
10            bot.pop()
11        bot.append(p)
12    return bot[:-1] + top[:0:-1]
```

5.3 Aire d'un polygone

Uniquement pour les polygones simples. Réduire à des composantes simples sinon. $A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i x_{i+1} - x_{i+1} y_i)$

```
1 def area(p):
2     A = 0
3     for i in range(len(p)):
4         A += p[i-1][0] * p[i][1] - p[i][0] * p[i-1][1]
5     return A/2
```

5.4 Points entiers dans un polygone

5.4.1 Sur le contour

5.4.2 Dans le polygone

Théorème de Pick : $P = n_i + \frac{n_b}{2} - 1$

5.5 Paire de points les plus proches

6 Ensembles

6.1 Rendu de monnaie

Problème NP-Complet.

```
1 def coin(x, R):
2     b = [False] * (R+1)
3     b[0] = True
4     for xi in x:
5         for s in range(xi, R+1):
6             b[s] |= b[s - xi]
7     return b[R]
```

6.2 Sac à dos

Problème NP-Complet.

```
1 def knapsack(p, v, cmax):
2     n = len(p)
3     Opt = [[0] * (cmax + 1) for _ in range(n+1)]
4     Sel = [[False] * (cmax + 1) for _ in range(n+1)]
5     #cas de base
6     for cap in range(p[0], cmax + 1):
7         Opt[0][cap] = v[0]
8         Sel[0][cap] = True
9     # cas d'induction
10    for i in range(1,n):
11        for cap in range(cmax+1):
12            if cap >= p[i] and Opt[i-1][cap - p[i]] + v[i] > Opt[i-1][cap]:
13                Opt[i][cap] = Opt[i-1][cap-p[i]] + v[i]
14                Sel[i][cap] = True
15            else:
16                Opt[i][cap] = Opt[i-1][cap]
17                Sel[i][cap] = False
18    cap = cmax
19    sol = []
20    for i in range(n-1, -1, -1):
21        if Sel[i][cap]:
22            sol.append(i)
23            cap -= p[i]
24    return (Opt[n-1][cmax], sol)
```

6.3 k-somme

7 Calculs

7.1 PGCD

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

7.2 Coefficients de Bézout

```
1 def bezout(a,b):
2     if b == 0:
3         return (1,0)
4     else:
5         u,v = bezout(b,a%b)
6         return (v, u - (a//b) *v)
7 def inv(a,p):
8     return bezout(a,p)[0]%p
```

7.3 Coefficients binomiaux

```
1 def binom(n,k,p):
2     prod = 1
3     for i in range(k):
4         prod = (prod * (n-i)) // (i+1) %p
5     return prod
6 #Enlever le p et mod p pour sans modulo
```
