

# SWERC NoteBook

SaintGermainDesPrés : Mathilde Bonin, Eyal Cohen, Hugo Demaret

March 20, 2022

## 1 Configuration

### 1.1 C/C++

## 2 Parcours de graphes

### 2.1 Implémentation des graphes

#### 2.1.1 C/C++

#### 2.1.2 Python

### 2.2 DFS - Depth First Search

---

```
1 #version iterative pour éviter 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)
```

---

### 2.3 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: #evaluate 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
```

---

- 2.4 Topological Sort
- 2.5 Composantes connexes
- 2.6 Composantes bi-connexe
- 2.7 Composantes fortement connexe
- 2.8 2-SAT
- 2.9 Postier Chinois
- 2.10 Chemin eulérien
- 2.11 Chemin le plus court
  - 2.11.1 Poids positif ou nul - Dijkstra
  - 2.11.2 Poids arbitraire - Bellman-Ford
  - 2.11.3 Floyd-Warshall

## 3 Points et polygones

### 3.1 Points

#### 3.1.1 Points

---

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

---

#### 3.1.2 Cross-product

---

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

---

#### 3.1.3 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]
```

---

### 3.2 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]
```

---

### 3.3 Aire d'un polygone

### 3.4 Paire de points les plus proches

## 4 Ensembles

### 4.1 Rendu de monnaie

### 4.2 Sac à dos

### 4.3 k-somme

## 5 Calculs

### 5.1 PGCD

---

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

---

### 5.2 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
```

---

### 5.3 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
```

---