SWERC NoteBook

SaintGermainDesPrés : Mathilde Bonin, Eyal Cohen, Hugo Demaret

March 19, 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

```python
#version iterative pour eviter la recursion limit de python
def dfs_iterative(graph,start,seen):
    seen[start] = True
    to_visit = [start]
    while to_visit:
        node = to_visit.pop()
        for neighbour in graph[node]:
            if not seen[neighbour]:
                seen[neighbour] = True
                to_visit.append(neighbour)
```

2.3 BFS - Breadth First Search

```python
from collections import deque
def bfs(graph, start=0):
    to_visit = deque()
    dist = [float('inf')] * len(graph)
    prec = [none] * len(graph)
    dist[start] = 0
    to_visit.appendleft(start)
    while to_visit: #evalue a faux si vide
        node = to_visit.pop()
        for neighbour in graph[node]:
            if dist[neighbour] == float('inf'):
                dist[neighbour] = dist[node] + 1
                prec[neighbour] = node
                to_visit.appendleft(neighbour)
    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

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def subtract(self, p):
      return Point(self.x - p.x, self.y - p.y)
    def __str__(self):
        return '(' + str(self.x) + ', ' + str(self.y) + ')'
```

3.1.2 Cross-product

```
def cross_product(p1, p2):
  return p1.x * p2.y - p2.x * p1.y
```

3.1.3 Direction

```
def direction(p1, p2, p3):
  return cross_product(p3.subtract(p1), p2.subtract(p1))
# checks if p3 makes left turn at p2
def left(p1, p2, p3):
  return direction(p1, p2, p3) < 0
# checks if p3 makes right turn at p2
def right(p1, p2, p3):
  return direction(p1, p2, p3) > 0
# checks if p1, p2 and p3 are collinear
def collinear(p1, p2, p3):
  return direction(p1, p2, p3) == 0
```

3.2 Enveloppe convexe

3.2.1 Marche de Jarvis

```
def jarvis_march(points):
    a = min(points, key = lambda point: point.x)
    index = points.index(a)
    l = index
    result = []
    result.append(a)
    while (True):
        q = (l + 1) % len(points)
        for i in range(len(points)):
           if i == l:
               continue
           d = direction(points[l], points[i], points[q])
           if d > 0 or (d == 0 and distance_sq(points[i], points[l]) > distance_sq(points[q], points[l])):
               q = i
        l = q
        if l == index:
           break
        result.append(points[q])
    return result
```

3.2.2 Graham Scan

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

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

5.2 Coefficients de Bézout

```
def bezout(a,b):
    if b == 0:
        return (1,0)
    else:
        u,v = bezout(b,a%b)
        return (v, u - (a//b) *v)
def inv(a,p):
    return bezout(a,p)[0]%p
```

5.3 Coefficients binomiaux

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