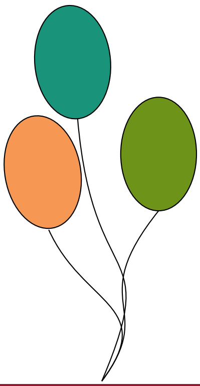
SWERC NoteBook

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

```
#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)
```

4.2 BFS - Breadth First Search

```
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
```

4.3 Topological Sort

4.4 Composantes connexes

4.5 Composantes bi-connexe

4.6 Composantes fortement connexe

4.6.1 Kosaraju

```
def kosaraju_dfs(graph,nodes,order,sccp):
    times_seen = [-1] * len(graph)
    for start in nodes:
        if times_seen[start] == -1:
            to_visit = [start]
            times_seen[start] = 0
            sccp.append([start])
            while to_visit:
                node = to_visit[-1]
                children = graph[node]
                if times_seen[node] == len(children):
                    to_visit.pop()
```

```
                order.append(node)
            else:
                child = children[times_seen[node]]
                times_seen[node] += 1
                if times_seen[child] == -1:
                    times_seen[child] = 0
                    to_visit.append(child)
                    sccp[-1].append(child)
def reverse(graph):
    rev_graph = [[] for node in graph]
    for node in range(len(graph)):
        for neighbour in graph[node]:
            rev_graph[neighbour].append(node)
    return rev_graph
def kosaraju(graph):
    n = len(graph)
    order = []
    sccp = []
    kosaraju_dfs(graph, range(n), order, [])
    kosaraju_dfs(reverse(graph),order[::-1], [], sccp)
    return sccp[::-1]
```

4.7 2-SAT

```
def vertex(lit):
    if lit > 0:
        return 2 * (lit - 1)
    else:
        return 2 * (-lit -1) +1
def two_sat(formula):
    n = max(abs(clause[p]) for p in (0,1) for clause in formula)
    graph = [[] for node in range(2*n)]
    for x,y in formula:
        graph[vertex(-x)].append(vertex(y))
        graph[vertex(-y)].append(vertex(x))
    sccp = kosaraju(graph)
    comp_id = [None] * (2*n)
    affectations = [None] * (2*n)
    for component in sccp:
        rep = min(component)
        for vtx in component:
            comp_id[vtx] = rep
            if affectations[vtx] == None:
                affectations[vtx] = True
                affectations[vtx ^ 1] = False
    for i in range(n):
        if comp_id[2*i] == comp_id[2*i+1]:
            return None
    return affectations[::2]
```

4.8 Postier Chinois

4.9 Chemin eulérien

4.9.1 Dirigé

```
def eulerian_tour_directed(graph):
    P = []
    Q = [0]
    R = []
    next = [0] * len(graph)
    while Q:
        node = Q.pop()
```

```
        P.append(node)
        while next[node] < len(graph[node]):
            neighbour = graph[node][next[node]]
            next[node] += 1
            R.append(neighbour)
            node = neighbour
        while R:
            Q.append(R.pop())
    return P
```

4.9.2 Non Dirigé

```
def eulerian_tour_undirected(graph):
    P = []
    Q = [0]
    R = []
    next = [0] * len(graph)
    seen = [set() for _ in graph]
    while Q:
        node = Q.pop()
        P.append(node)
        while next[node] < len(graph[node]):
            neighbour = graph[node][next[node]]
            next[node] += 1
            if neighbour not in seen[node]:
                seen[neighbour].add(node)
                R.append(neighbour)
                node = neighbour
        while R:
            Q.append(R.pop())
    return P
```

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

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

5.1.2 Cross-product

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

5.1.3 Direction

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

5.2 Enveloppe convexe

Complexité : $\mathcal{O}(n \log(n))$

```
def andrew(S):
    S.sort()
    top = []
    bot = []
    for p in S:
        while len(top) >= 2 and not left_turn(p,top[-1],top[-2]):
            top.pop()
        top.append(p)
        while len(bot) >= 2 and not left_turn(bot[-2],bot[-1],p):
            bot.pop()
        bot.append(p)
    return bot[:-1] + top[:0:-1]
```

5.3 Aire d'un polygone

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

```
def area(p):
    A = 0
    for i in range(len(p)):
        A += p[i-1][0] * p[i][1] - p[i][0] * p[i-1][1]
    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.

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

6.2 Sac à dos

Problème NP-Complet.

```
def knapsack(p, v, cmax):
    n = len(p)
    Opt = [[0] * (cmax + 1) for _ in range(n+1)]
    Sel = [[False] * (cmax + 1) for _ in range(n+1)]
    #cas de base
    for cap in range(p[0], cmax +1):
        Opt[0][cap] = v[0]
        Sel[0][cap] = True
    # cas d'induction
    for i in range(1,n):
        for cap in range(cmax+1):
            if cap >= p[i] and Opt[i-1][cap - p[i]] + v[i] > Opt[i-1][cap]:
```

```python
                Opt[i][cap] = Opt[i-1][cap-p[i]] + v[i]
                Sel[i][cap] = True
            else:
                Opt[i][cap] = Opt[i-1][cap]
                Sel[i][cap] = False
    cap = cmax
    sol = []
    for i in range(n-1, -1, -1):
        if Sel[i][cap]:
            sol.append(i)
            cap -= p[i]
    return (Opt[n-1][cmax], sol)
```

6.3 k-somme

7 Calculs

7.1 PGCD

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

7.2 Coefficients de Bézout

```python
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
```

7.3 Coefficients binomiaux

```python
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
```