ds5.py

```python
#Exemple simplifié
ligne1='ABCD'
ligne2='AEFG'
ligne3='DHIJKG'
metro=[ligne1,ligne2,ligne3]

## Question 1
def tri(l):
    n=len(l)
    for i in range(n):
        rg,m = i, l[i]
        for j in range(i,n):
            if l[j]<m:
                rg,m = j,l[j]
        l[rg],l[i]=l[i],l[rg]

def init_sommet(metro):
    S=[]
    for l in metro:
        for g in l:
            if not g in S:
                S.append(g)
    tri(S)
    return S

S = init_sommet(metro)

## Question 2
def indice(nom, S):
    for k in range(len(S)):
        if nom == S[k]:
            return k

def init_arrete(metro):
    S=init_sommet(metro)
    mat = [[None]*len(S) for i in range(len(S))]
    for num in range(len(metro)):
        l=metro[num]
        for a in range(1,len(l)):
            i = indice(l[a-1],S)
            j = indice(l[a],S)
            mat[i][j] = num+1
            mat[j][i] = num+1
    return mat

M = init_arrete(metro)

## Question 3
from collections import deque

def chemin(depart, arrive):
    file = deque()
    deja_vu = [False]*len(S)
    avant = {}
    file.append(depart)
    while len(file)>0:
        g = file.popleft()
        if g == arrive:
            break
        deja_vu[g]=True
        for k in range(len(S)):
            if not deja_vu[k] and M[g][k] is not None:
                file.append(k)
                avant[k] = g
    ch = deque([arrive])
```

```
66|     g = arrive
67|     while g!=depart:
68|         g=avant[g]
69|         ch.appendleft(g)
70|     return list(ch)
71|
72| ## Question 4
73| def trajet(lgare):
74|     lnum = []
75|     lnom = []
76|     num = 0
77|     for k in range(len(lgare)-1):
78|         n = M[lgare[k]][lgare[k+1]]
79|         if num!=n:
80|             lnum.append(num)
81|             lnom.append(S[k])
82|             num=n
83|     lnom.append(S[lgare[-1]])
84|     return lnum,lnom
85|
86| ## Question 5
87| Le parcours en largeur va explorer les trajets en 1 étape puis en 2 étapes puis
en 3 étapes.
88| Donc le première fois où l'algorithme atteindra l'arrivée le trajet sera le plus
court possible en nombre d'étape.
89| Avec un parcours en profondeur, l'algorithme recherchera la gare en parcourant un
maximun de gares du réseau.
90|
91| ## Question 6
92| L'algorithme de Dijkstra n'a pas d'avantage ici car toutes les distances dans le
graphe valent 1 donc la profondeur équivaut déjà à la distance. Dans ce cas BFS et
Dijkstra sont donc équivalent.
93| Pour appliquer l'algorithme de A*, il faudrait pouvoir créer un distance
heuristique entre deux gares quelconques. Par exemple les coordonées GPS des gares
permettraient de savoir la distance à pied entre deux de ces gares même si il n'y pas
de ligne qui les relient.
```