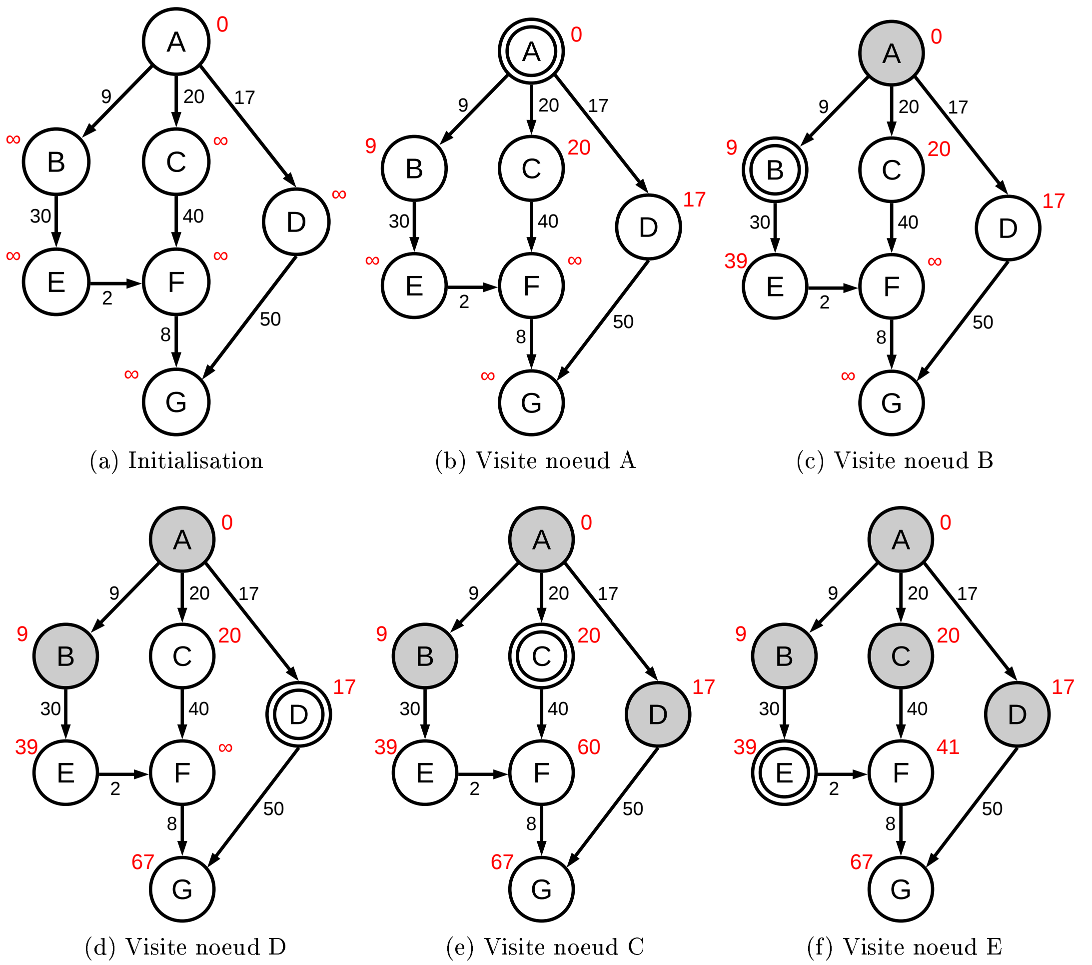
TP 6 - Corrigé
Algorithme de Dijkstra

Les solutions données dans ce corrigé ne sont bien sûr que des propositions, et sont sans nul doute perfectibles.

2 Pseudo-algorithme

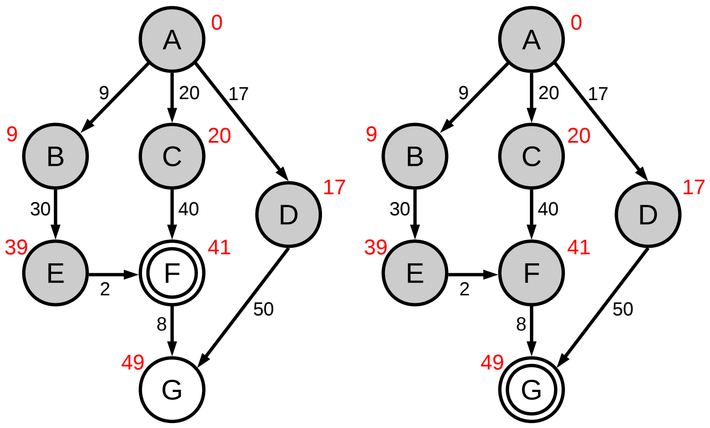
Q1 Voir figures 1 et 2.



(a) Initialisation (b) Visite noeud A (c) Visite noeud B

(d) Visite noeud D (e) Visite noeud C (f) Visite noeud E

FIGURE 1 – Application de l'algorithme de Dijkstra



(a) Visite noeud F (b) Visite noeud d'arrivée : arrêt

FIGURE 2 – Application de l'algorithme de Dijkstra (suite)

Q2 Ci-dessous l'algorithme de Dijkstra retournant le plus court chemin, en pseudo-code. On a simplement rajouté la ligne 17 et les lignes 21 à 29.

```
1:  procedure DIJKSTRA(G,depart,arrivee)
2:      noeud_visites ← ∅
3:      pour chaque noeud n de G faire
4:          distance_min[n] ← +∞
5:      fin pour
6:      distance_min[depart] ← 0
7:      tant que noeuds_visites ne contient pas tous les noeuds de G faire
8:          noeud_courant ← noeud non visité de distance minimale
9:          noeud_visites ← noeud_visites ∪ {noeud_courant}
10:         si noeud_courant = arrivee alors
11:             quitter boucle
12:         fin si
13:         pour chaque arc sortant (successeur,longueur_arc) de noeud_courant faire
14:             distance ← distance_min[noeud_courant] + longueur_arc
15:             si distance < distance_min[successeur] alors
16:                 distance_min[successeur] ← distance
17:                 predecesseur[successeur] ← noeud_courant
18:             fin si
19:         fin pour
20:     fin tant que
```

```
    chemin ← []
    si arrivee a un prédecesseur alors
        noeud_courant ← arrivee
        ajouter arrivee au chemin
        tant que noeud_ courant a un prédecesseur faire
            ajouter noeud_courant en tête du chemin
            noeud_courant ← predecesseur[noeud_courant]
        fin tant que
    fin si
    retourner chemin, distance_min[arrivee]
fin procedure
```

3 Implémentation

3.1 Représentation d'un graphe

3.1.1 Matrice d'adjacence

Q3 La matrice d'adjacence correspondant au graphe est la suivante.

$$\begin{pmatrix} -1 & 9 & 20 & 17 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & 30 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 40 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 50 \\ -1 & -1 & -1 & -1 & -1 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 8 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

On peut l'écrire en Python comme suit.

```
matrice_graphe = [
[-1, 9, 20, 17, -1, -1, -1],
[-1, -1, -1, -1, 30, -1, -1],
[-1, -1, -1, -1, -1, 40, -1],
[-1, -1, -1, -1, -1, -1, 50],
[-1, -1, -1, -1, -1, 2, -1],
[-1, -1, -1, -1, -1, -1, 8],
[-1, -1, -1, -1, -1, -1, -1]]
```

3.1.2 Liste d'adjacence

Q4 On donne ci-dessous la liste des arcs sortants de chaque noeud.

Arcs sortants de :

0. $[(1, 9), (2, 20), (3, 17)]$
1. $[(4, 30)]$
2. $[(5, 40)]$
3. $[(6, 50)]$
4. $[(5, 2)]$
5. $[(6, 9)]$
6. [] (liste vide)

On peut l'écrire en Python de la manière suivante.

```
graphe = [
[(1,9), (2,20), (3,17)],
[(4,30)],
[(5,40)],
[(6,50)],
[(5,2)],
[(6,9)],
[]]
```

3.2 Structures de données utiles

3.2.3 Implémentation

Q5 Il n'y a ici rien à faire.

Listing 1 – arcs_sortants

```
def arcs_sortants(graphe, noeud):
    return graphe[noeud]
```

Q6 Ci-dessous le contenu des différentes variables après chaque itération.

Itération	`noeuds_visites`	`distance_min`	`file_priorite`
0	$\emptyset$	$\{0 : 0\}$	$[(0, 0)]$
1	$\{0\}$	$\{0 : 0, 1 : 9, 2 : 20, 3 : 17\}$	$[(1, 9), (3, 17), (2, 20)]$
2	$\{0, 1\}$	$\{0 : 0, 1 : 9, 2 : 20, 3 : 17, 4 : 39\}$	$[(3, 17), (2, 20), (4, 17)]$
3	$\{0, 1, 3\}$	$\{0 : 0, 1 : 9, 2 : 20, 3 : 17, 4 : 39, 6 : 67\}$	$[(2, 20), (4, 17), (6, 67)]$
4	$\{0, 1, 2, 3\}$	$\{0 : 0, 1 : 9, 2 : 20, 3 : 17, 4 : 39, 5 : 60, 6 : 67\}$	$[(4, 39), (5, 60), (6, 67)]$
5	$\{0, 1, 2, 3, 4\}$	$\{0 : 0, 1 : 9, 2 : 20, 3 : 17, 4 : 39, 5 : 41, 6 : 67\}$	$[(5, 41), (6, 67)]$
6	$\{0, 1, 2, 3, 4, 5\}$	$\{0 : 0, 1 : 9, 2 : 20, 3 : 17, 4 : 39, 5 : 41, 6 : 49\}$	$[(6, 49)]$
7	$\{0, 1, 2, 3, 4, 5, 6\}$	$\{0 : 0, 1 : 9, 2 : 20, 3 : 17, 4 : 39, 5 : 41, 6 : 49\}$	$[]$

Q7 Il suffit presque de traduire ligne à ligne le pseudo-algorithme en Python. Il faut juste prendre garde au fait qu'on ne peut pas tester directement si la file est vide, et considérer que la distance à un noeud est infinie s'il n'a pas d'entrée dans le dictionnaire.

Listing 2 – Pseudo-algorithme de Dijkstra

```
def dijkstra(G, depart, arrivee):
    file_priorite = PriorityQueue()
    file_priorite.insert(depart, 0) # File de priorite initialisee avec le noeud de depart
    dict_distance_min = {depart: 0} # Dictionnaire contenant la plus petite distance calculee vers chaque noeud
    noeuds_visites = set() # Ensemble des noeuds deja visites
    while True:
        entree = file_priorite.pop()
        if entree is None: # La file est vide, le graphe a ete entierement parcouru, l'arrivee n'est pas accessible
            break
        distance, noeud = entree
        if noeud not in noeuds_visites:
            noeuds_visites.add(noeud)
            if noeud == arrivee: # On a trouve le plus court chemin vers l'arrivee, on peut s'arreter
                break
            for successeur, longueur_arc in arcs_sortants(G, noeud):
                # On compare la distance minimale connue vers le successeur avec la longueur du plus court chemin vers le noeud courant + la longueur de l'arc allant du noeud courant vers le successeur.
                nouvelle_distance = distance+longueur_arc
                distance_min = dict_distance_min.get(successeur)
```

```
                    if (distance_min is None or nouvelle_distance < distance_min):
                        # On met a jour la distance minimale connue, et la file de priorite
                        dict_distance_min[successeur] = nouvelle_distance
                        file_priorite.insert(successeur, nouvelle_distance)

    distance = distance_min.get(arrivee)
    return distance
```

Q8 Là aussi c'est une traduction du pseudo-algorithme. La seule différence est encore qu'on ne peut savoir que la file est vide qu'après avoir essayé d'en retirer un élément.

Listing 3 – Algorithme de Dijkstra - Avec retour du plus court chemin

```
def dijkstra(G, depart, arrivee):
    file_priorite = PriorityQueue()
    file_priorite.insert(depart, 0)
    dict_distance_min = {depart: 0}
    dict_predecesseur = {} # Dictionnaire contenant le meilleur noeud predecesseur pour le plus court chemin
    noeuds_visites = set()
    while True:
        entree = file_priorite.pop()
        if entree is None:
            break
        distance, noeud = entree
        if noeud not in noeuds_visites:
            noeuds_visites.add(noeud)
            if noeud == arrivee:
                break
            for successeur, longueur_arc in arcs_sortants(G, noeud):
                nouvelle_distance = distance+longueur_arc
                distance_min = dict_distance_min.get(successeur)
                if (distance_min is None or nouvelle_distance < distance_min):
                    # On met a jour egalement le predecesseur
                    dict_distance_min[successeur] = nouvelle_distance
                    dict_predecesseur[successeur] = noeud
                    file_priorite.insert(successeur, nouvelle_distance)

    distance = distance_min.get(arrivee)
```

```
    chemin = []
    if distance is not None:
        # Reconstruction du plus court chemin
        noeud = arrivee
        while noeud is not None:
            chemin = [noeud] + chemin
            noeud = dict_predecesseur.get(noeud)
    return (chemin,distance)
```

4 Application au redimensionnement d'images

4.3 Énergie d'une image

Q9 La fonction `deriv_x` calcule pour chaque pixel, à part ceux du bord gauche et droit, la norme de la différence entre le pixel à sa gauche et celui à sa droite, et retourne la matrice correspondante : l'élément (x, y) contient la différence entre le pixel $(x + 1, y)$ et le pixel $(x - 1, y)$. On appelle souvent cette matrice l'énergie de l'image.

L'énergie du pixel (x, y) est un bon candidat pour la longueur des arcs allant vers le pixel (x, y) : en effet, un chemin empruntant un tel arc mettra en contact, une fois supprimé, les pixels $(x - 1, y)$ et $(x + 1, y)$. Il est donc logique que cette longueur soit d'autant plus grande que cette différence est forte.

Q10 Il suffit de faire proprement une disjonction des cas.

Listing 4 – arcs_sortants - Pour une image

```
def arcs_sortants(graphe, noeud):
    image,energy = graphe
    x,y = noeud
    if y < 0:
        # Du noeud au-dessus de l'image sortent les arcs vers tous les du bord haut de l'image
        return [((dx,0),0) for dx in range(1,image.shape[1]-1)]
    elif y >= image.shape[0]-1:
        # D'un pixel du bord bas de l'image sort un arc vers le noeud en dessous de l'image
        return [((0,image.shape[0]),0)]
    elif x <= 0:
        # Du bord gauche de l'image ne sort aucun arc
        return []
    elif x == 1:
        # Du pixel juste a droite du bord gauche de l'image il n'y a que deux arcs (pas d'arc vers le bord gauche)
        return [((x,y+1),energy[y+1,x]), ((x+1,y+1),energy[y+1,x+1])]
    elif x >= image.shape[1]-1:
        # Du bord droit de l'image ne sort aucun arc
```

```
        return []
    elif x == image.shape[1]-2:
        # Du pixel juste a gauche du bord droit de l'image il n'
           y a que deux arcs (pas d'arc vers le bord droit)
        return [((x-1,y+1),energy[y+1,x-1]), ((x,y+1),energy[y
           +1,x])]
    else:
        # D'un pixel loin des bords de l'image sortent 3 arcs,
           vers le pixel en bas a gauche, en bas, et en bas a
           droite
        return [((x-1,y+1),energy[y+1,x-1]), ((x,y+1),energy[y
           +1,x]), ((x+1,y+1),energy[y+1,x+1])]
```

Q11 Le graphe est le couple `(image,energy)`, le noeud de départ est le noeud au-dessus de l'image, i.e. `(0,-1)` et le noeud d'arrivée est le noeud en-dessous de l'image, i.e. `(0,hauteur)`.

Listing 5 – Suppresion d'un chemin

```
height = image.shape[0]
width = image.shape[1]
energy = deriv_x(image)
seam,d = dijkstra((image,energy),(0,-1),(0,height))
tracer_chemin(image,seam)
plt.clf()
plt.axis([0,width-1,0,height-1])
plt.gca().invert_yaxis() # pour que l'image soit a l'endroit
plt.imshow(image)
```

Q12 Pas de difficulté particulière ici, il suffit d'utiliser la fonction `supprimer_chemin` déjà fournie et la fonction `pause` du module `pyplot` pour mettre à jour la fenêtre au fur et à mesure.

Listing 6 – Réduction de la largeur par seam-carving

```
width = image.shape[1]
height = image.shape[0]
for i in range(100):
    energy = deriv_x(image)
    seam,d = dijkstra((image,energy),(0,-1),(0,height))
    tracer_chemin(image,seam)
    plt.clf()
    plt.axis([0,width-1,0,height-1])
    plt.gca().invert_yaxis()
    plt.imshow(image)
    plt.pause(10**-15)
    image = supprimer_chemin(image,seam)
```