Numéro d'inscription

Numéro de table

Né(e) le

Nom : La Frite

Prénom : Jojo

Emplacement QR Code

Filière : PC

Session : 2023

Épreuve de : INFORMATIQUE

Consignes

- *Remplir soigneusement l'en-tête de chaque feuille avant de commencer à composer*
- *Rédiger avec un stylo non effaçable bleu ou noir*
- *Ne rien écrire dans les marges (gauche et droite)*
- *Numéroter chaque page (cadre en bas à droite)*
- *Placer les feuilles A3 ouvertes, dans le même sens et dans l'ordre*

PC5IN

DOCUMENT RÉPONSE

Ce Document Réponse doit être rendu dans son intégralité.

Q1 - Taille du mot en bits. Dimensions en pixels d'une image. Taille en bits du fichier image

- Pour représenter $m \in [\![0,255]\!]$, il faut 8 bits, soit un octet.
- $P = \frac{21}{2,54} \times 300 \times \frac{29,7}{2,54} \times 300 \approx 2480 \times 3507 = 8697360$ pixels
- Pour obtenir le nombre de bits, il faut tenir compte des trois canaux :

  $P \times 3 \times 8 = 208736640$ bits soit ≈ 26 Moctets.

Q2 - Complexité de la fonction `conversion_gris(imgè:array)->array`

Les instructions du corps de boucle sont effectuées en un temps constant. Donc, si on ne tient pas compte de dimension et initialise, la complexité est en $O(n_0 n_1)$

NE RIEN ÉCRIRE DANS CE CADRE

Q3 - Fonction `binarisation(imgG:array, seuil:int)->array`

```
def binarisation(imgG, seuil):
    n0, n1 = dimension(imgG)
    img = initialise(n0, n1, 0)
    for i in range(n0):
        for j in range(n1):
            if imgG[i, j] > seuil:
                img[i, j] = 255
    return img
```

Q4 - Choix de la fonction `bilineaire(im:array, x:float, y:float)->int`

```
def bilineaire(im:array,x:float,y:float)->int:
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x1][y1])
    b = lineaire(y,y0,y1,im[x1][y0],im[x0][y1])
    c = lineaire(x,x0,x1,a,b)
    return int(c)
```

OK

```
def bilineaire(im:array,x:float,y:float)->int:
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x0][y1])
    b = lineaire(y,y0,y1,im[x1][y0],im[x1][y1])
    c = lineaire(x,x0,x1,a,b)
    return int(c)
```

```
def bilineaire(im:array,x:float,y:float)->int:
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x0][y1])
    b = lineaire(y,y0,y1,im[x0][y0],im[x1][y0])
    c = lineaire(x,x0,y1,a,b)
    return int(c)
```

```
def bilineaire(im:array,x:float,y:float)->int:
    x0 = int(x)
    x1 = x0+1
    y0 = int(y)
    y1 = y0+1
    a = lineaire(y,y0,y1,im[x0][y0],im[x0][y1])
    b = lineaire(y,y0,y1,im[x1][y0],im[x0][y1])
    c = lineaire(y,y0,y1,a,b)
    return int(c)
```

* from math import pi, cos, sin

Q5 - Fonction `rotation(im:array, angle:float)->array`

```
def rotation(im:array, angle:float)->array:
    p,q,_ = dimension(im)
    imr = .initialise.(p,q,255) # par défaut en blanc
    angr = ..angle..*..pi../180 # on suppose : from math import pi
    matR = [[.cos(angr)., .sin(angr)], [.-sin(angr), cos(angr)]] # (*)
    for ni in range(...p.....):
        for nj in range(.q.....):
            x, y = prod_matrice_vecteur(matR, [ni - p//2, nj - q//2])
            x = x + .p//2.....
            y = y + .q//2.....
            if  0.<=.int(x).<.p-1..and...0.<=.int(y)<q-1 :
                ...imr[ni,nj]..=.bi_lineaire(im, x, y)
    return imr
```

Q6 - Étude de la fonction `lineaire`

- L'image codée en uint8 prend 8x moins de place.
- 19 - 23 = -5 qui n'est pas représentable en uint8.

C'est un dépassement de capacité. Les résultats du linéaire ne sont donc pas toujours corrects.

Q7 - Fonction `histo_lignes(im:array)->list`

```
def histo_lignes(im):
    h = []
    for ligne in im:
        c = 0
        for pixel in ligne:
            if pixel == 0:
                c+=1
        h.append(c)
    return h
```

Q8 - Fonction `detecter_lignes(liste:list)->list`

```
def detecter_lignes(liste :list)->list :
    lignes = []
    i = 0
    deb = -1 #contient -1 tant qu'on parcourt des lignes de pixel blanc
    while i<m :
        #début d'une suite de lignes contenant des pixels noirs
        if liste[i]!=0 and deb == -1 :
            deb = i
        #fin d'une suite de lignes contenant des pixels noirs
        elif liste[i]==0 and deb != -1 :
            lignes.append((deb, i-1))
            deb = -1
        i=i+1
    return lignes
```

Q9 - Justification de la terminaison et nom de la méthode. Nombre d'itérations nécessaires

- Il s'agit d'un algorithme dichotomique.
- L'intervalle $[a, b]$ de recherche est divisé par 2 à chaque itération. Donc au bout de k itérations on aura $\frac{b-a}{2^k} < \varepsilon$ et l'algorithme termine.
- Nombre d'itérations nécessaires $\lceil \log_2\left(\frac{b-a}{\varepsilon}\right) \rceil$

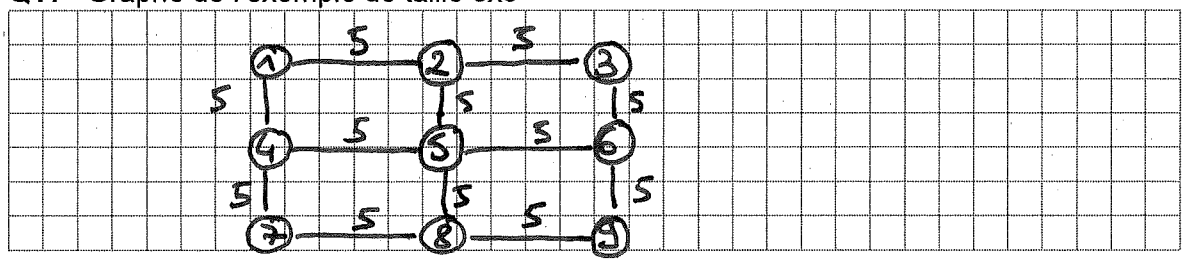
Q10 - Fonction `rotation_auto(im:array, a:float, b:float)->array`

```
def nb_zeros(im:array, angle:float)->int:
    imr = rotation(im, angle)
    ligne = histo_ligne(imr)
    f=ligne.count(0)
    return f
```

```
def rotation_auto(im:array, a:float, b:float)->array:
    c = (a+b)/2
    fc = nb_zeros(im,c)
    while b-a > 0.1:#plus grand que 0.1 degré
        ac = (a+c)/2
        fac = nb_zeros(im,ac)
        cb = (c+b)/2
        fcb = nb_zeros(im,cb)
        maxi = max(fac,fc,fcb)
        if fac == maxi:
            b = c
            c = ac
            fc = fac
        elif fc == maxi:
            a = ac
            b = cb
        else:
            a = c
            c = cb
            fc = fcb
    return rotation(im,(b+a)/2)
```

NE RIEN ÉCRIRE DANS CE CADRE

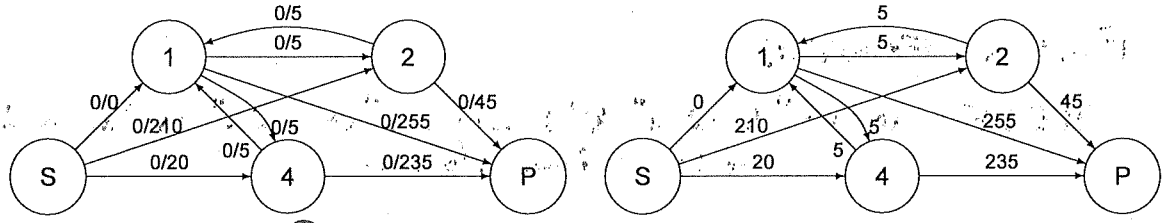
Q11 - Graphe de l'exemple de taille 3x3



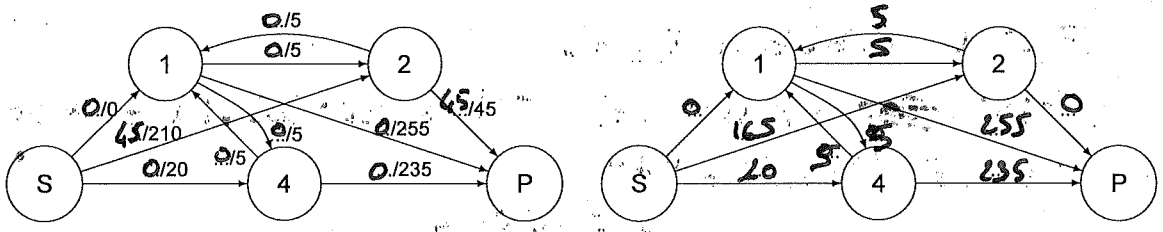
Q12 - Compléter la partie supérieure de la matrice de capacités

	S	1	2	3	4	5	6	7	8	9	P
S	0	0	210	190	20	100	200	10	5	255	0
1	-	0	5	0	5	0	0	0	0	0	255
2	-	-	0	5	0	5	0	0	0	0	45
3	-	-	-	0	0	0	5	0	0	0	65
4	-	-	-	-	0	5	0	5	0	0	235
5	-	-	-	-	-	0	5	0	5	0	155
6	-	-	-	-	-	-	0	0	0	5	55
7	-	-	-	-	-	-	-	0	5	0	245
8	-	-	-	-	-	-	-	-	0	5	250
9	-	-	-	-	-	-	-	-	-	0	0
P	-	-	-	-	-	-	-	-	-	-	0

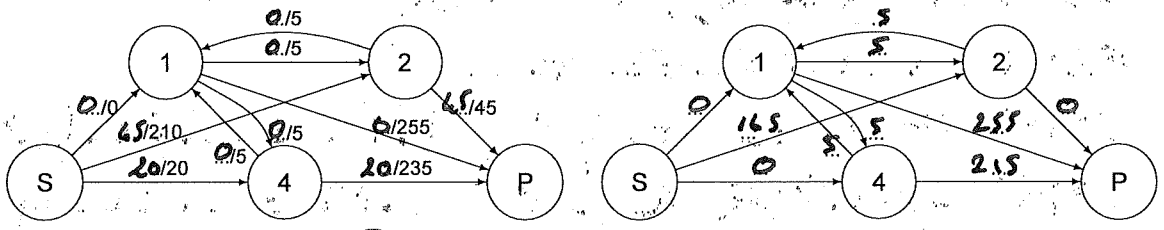
Q13 - Compléter les graphes de flot à gauche et résiduel à droite. Pour chaque étape, préciser le chemin choisi et la valeur de l'augmentation du flot



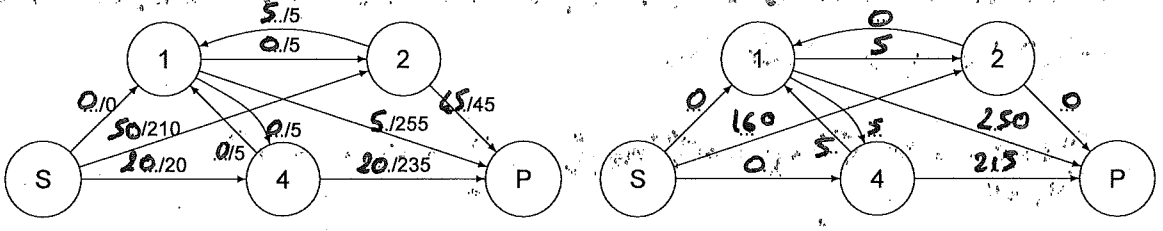
Chemin choisi : S 2 P



Chemin choisi : S 4 P



Chemin choisi : S 2 1 P



Q14 - Ensembles A et B. Flot maximal. Coupe choisie et capacité de coupe

A = {S, 2} B = {1, 4, P} flux max = 70
= 50+20
Coupe {(S,1), (S,4), (2,P), (2,1)} → coupe de flux 70.

Q15 - Correspondance ensembles A et B. Analyse

A correspond aux pixels blancs, B aux noirs.
Le résultat est bon, la lettre t est
reconnaissable et nette.

Q16 - Requête SQL

```
SELECT id
FROM FONTES
WHERE nom = "Zurich" AND style = "romain" AND taille <= 16
AND taille >= 10.
```

Q17 - Requête SQL

```
SELECT fichier
FROM CARACTERES
JOIN [illegible] SYMBOLES ON SYMBOLES.id = CARACTERES
.id_symbole
WHERE label = "A"
```

Q18 - Requête SQL

```
SELECT (label), COUNT(*)   → à retirer
FROM CARACTERES AS c
JOIN FONTES AS f ON f.id = c.id_fonte
JOIN SYMBOLES AS s ON s.id = c.id_symbole
WHERE nom = "Zurich" AND style = "romain" AND taille >= 10 AND taille <= 16
GROUP BY label.
```

Q19 - Contenu des variables `car`, `num`, `var`, `ind`. Retour de la fonction

`car` : ["Zurich LightRst", "mgnumb", "iapng"] `var` : "majuscules"

`num` : "10" `ind` : 0

Retour de la fonction : K

Q20 - Fonction `lire_donnees_ref(fichier_car_ref:list)->dict`

```
def lire_donnees_ref(fichiers_car_ref):
    c_ref = {}
    for nom_f in fichiers_car_ref:
        symb = lire_symbole_fichier(nom_f)
        if symb in c_ref:
            c_ref[symb].append(imread(nom_f))
        else:
            c_ref[symb] = [imread(nom_f)]
    return c_ref
```

Q21 - Fonction `distance(im1:array, im2:array)->float`

```
def distance(im1, im2):
    n0, n1, _ = dimension(im1)
    d = 0
    for i in range(n0):
        for j in range(n1):
            d += (im1[i,j] - im2[i,j])**2
    return d**(1/2)
```

NE RIEN ÉCRIRE DANS CE CADRE

Q22 - Fonction `calcul_distances(carac_ref:dict, carac_test:array)->dict`

```
def calcul_distances(carac_ref, carac_test):
    d = {}
    for s in carac_ref:
        d[s] = []
        for img in carac_ref[s]:
            d[s].append(distance(img, carac_test))
    return d
```

Q23 - Méthode de tri performante envisageable et complexité temporelle

Tri fusion

$O(n \log n)$ dans tous les cas

Q24 - Compléter les 3 zones manquantes

```
def Kvoisins(distances :dict,K:int)->list :
    voisins = [(float("inf"),"") for k in range(K)]
    for lettre in distances :
        d = distances[lettre]
        for j in range( len(d)..... ):
            if voisins[K-1][0] > d[j] :
                k = len(voisins)-1
                while k>0 and voisins[k-1][0] > d[j]
                    voisins[k] = voisins[k-1]
                    k = k - 1
                voisins[k] = [d[j], lettre]
    return voisins
```

Q25 - Complexité en fonction de n et K. Comparaison

Dans le pire cas, toute la liste voisins est parcourue à chaque fois. Les opérations de la boucle sont de complexité constante. Donc la complexité est $O(Kn)$.

Cela est intéressant si K est petit devant n.

Q26 - Fonction `symbole_majoritaire(voisins:list)->str`

```
def symbole_majoritaire(voisins):
    S = {}
    for v in voisins:
        s = v[1]
        if s not in S:
            S[s] = 1
        else:
            S[s] += 1
    M = 0 ; classe = None
    for s in S:
        if S[s] > M:
            M = S[s]
            classe = s
    return classe.
```

Q27 - Commentaires

Le nombre de voisins K n'apparaît pas un facteur améliorant. Par contre, le nombre d'un nombre d'images de référence semble avoir un impact sur la classification.

FIN