پروژه کامپیوتری دوم

آرین باستانی 810100088

قسمت اول)

1- با تابع imread() میتوان اطلاعات یک تصویر را در ماتریس سه بعدی ذخیره کرد، که دو بعد آن همان ابعاد تصویر و بعد دیگر آن اطلاعات رنگ (همان rgb) است.

```
[file, path] = uigetfile({'*.jpg;*.bmp;*.png;*.tif'}, 'Choose an Image');
image = imread([path, file]);
```

شکل متغیر image

2- تصویر پس از ریسایز کردن به شکل زیر است:

شکل پس از ریسایز

3- از آنجایی که عکس خاکستری خواهد بود و میخواهیم سه مقدار برای شدت رنگهای رنگ قرمز، سبز و آبی را به یک تک مقدار رنگ تبدیل کنیم، پس از خاکستری کردن عکس ماتریس سه بعدی به یک ماتریس دو بعدی تبدیل میشود که به ازای هر پیکسل یک مقدار برای رنگ را در بر دارد.

بنابراین خروجی این تابع یک ماتریس به ابعاد عکس ورودی خواهد بود و کافیست با دو حلقه، برای هر پیکسل طبق فرمول داده شده، مقدار عددی رنگ جدید را محاسبه کنیم.

```
function gray_image=mygrayfun(image)
    gray_image=zeros(size(image,1),size(image,2));
    for i=1:size(image,1)
        for j=1:size(image,2)
            gray_image(i,j)=0.299*image(i,j,1)+0.578*image(i,j,2)+0.114*image(i,j,3);
        end
    end
    gray_image=uint8(gray_image);
end
```

تابع *mygrayfun*

خروجی تابع *mygrayfun*

4- برای تعیین این مرز، از تابع graythresh و یک ضریب که با آزمون و خطا با عکس ها بدست آوردم استفاده کردم:

```
threshold = 247 * graythresh(gray_image);
```

تعیین مرز

در این تابع نیز باید با استفاده از دو حلقه، به ازای هر پیکسل چک کنیم. اگر مقداری که این پیکسل برای رنگ خاکستری گرفته بود کمتر از threshold بود، مقدار متناظر با رنگ خاکستری این پیکسل را صفر میکنیم و در غیر این صورت برابر با عدد یک قرار میدهیم.

```
function binary_image=mybinaryfun(gray_image,threshold)
    binary_image=zeros(size(gray_image,1),size(gray_image,2));
    for i=1:size(gray_image,1)
        for j=1:size(gray_image,2)
            if(gray_image(i,j)<threshold)
                binary_image(i,j)=0;
            else
                binary_image(i,j)=1;
            end
        end
    end
    binary_image=logical(binary_image);
end
```

تابع mybinaryfun

و از آنجایی که ما المنت های مورد نظر را به رنگ سفید میخواهیم باید پس از استفاده از آن، صفر ها را یک و یکهارا صفر کنیم که این امر با اپراتور ~ امکان پذیر است.

```
binary_image=~mybinaryfun(gray_image,threshold);
```

متمم گرفتن از خروجی

عکس سیاه و سفید شده

5- برای پیاده سازی از ایده ی bfs استفاده کردم.

ابتدا تغییرات افقی و عمودی را در دو متغیر ذخیره کردم. که شامل چهار جهت اصلی و چهار جهت بین اینها میباشد. برای مثال برای همسایه ی سمت راست تغییرات افقی برابر +1 و تغییرات عمودی برابر 0 میباشد.

سپس با استفاده از یک حلقه، پیکسل هایی که مقدار آنها یک بود را در یک متغیر جدید ذخیره کردم تا آنها را چک کرده و پیکسل های پیوسته به هم را پیدا کنم.

```
function cleanimage = myremovecom(binaryimage, n)
    adjacency_row = [ 0, 0, -1, -1, -1, 1, 1, 1];
    adjacency_col = [-1, 1, 0, -1, 1, 0, -1, 1];

    object_set = {};
    visited = zeros(size(binaryimage));

    [rows, cols] = size(binaryimage);
    ones = [];
    for i = 1:cols
        for j = 1:rows
            if binaryimage(j, i) == 1
                ones = [ones [j; i]];
            end
        end
    end
```

مقداردهی اولیه و پیدا کردن پیکسل های یک

و بقیه ی پیاده سازی دقیقا مانند bfs خواهد بود.

بنابراین با یک حلقه ی اصلی در پیکسل های یک، چک میکنیم اگر این پیکسل قبلا ویزیت شده بود که به پیکسل بعدی میرویم و در غیر اینصورت این پیکسل را به پیکسل های ویزیت شده اضافه میکنیم و این پیکسل را به عنوان پیکسلی که میخواهیم همسایه هایش را چک کنیم ذخیره میکنیم.

سپس در ادامه ی داخل این حلقه، همسایه های این پیکسل را چک میکنیم و هرکدام که 1 بود، آن را به همسایه های آبجکت مورد نظر، آبجکت هایی که میخواهیم همسایه هایش را بررسی کنیم، و به پیکسل های ویزیت شده اضافه میکنیم.(باید چک کنیم که این همسایه ها از صفحه بیرون نزنند)

و سپس این چک کردن همسایه ها را، برای تمام همسایه های آبجکت مورد نظر انجام میدهیم و آنقدر اینکار را تکرار میکنیم تا دیگر همسایه ی فعلی، همسایه ی جدید دیگری نداشته باشد.

و سپس در حلقه ی اصلی به یک پیکسلی خواهیم رفت که جزو آبجکت های قبلی نبوده است. هر بار در حلقه ی اصلی، یک آبجکت جدا را شناسایی میکنیم.

```
for k = 1:size(ones, 2)
    current_object = [];
    object_queue = [];

    if visited(ones(1, k), ones(2, k))
        continue;
    else
        visited(ones(1, k), ones(2, k)) = 1;
        object = ones(:, k);
        object_queue = ones(:, k);
    end

    while ~isempty(object_queue)
        currentX = object_queue(1, 1);
        currentY = object_queue(2, 1);
        object_queue(:, 1) = [];

        for m = 1:size(adjacency_row, 2)
            newX = currentX + adjacency_row(m);
            newY = currentY + adjacency_col(m);

            if is_valid_coordinate(newX, newY, [rows, cols]) && ...
               ~visited(newX, newY) && ...
               binaryimage(newX, newY) == 1

                   visited(newX, newY) = 1;
                   object_queue = [object_queue [newX; newY]];
                   object = [object [newX; newY]];
            end
        end
    end

    object_set = [object_set object];
end
```

پیدا کردن آبجکت های تصویر

```
function is_valid = is_valid_coordinate(x, y, imageSize)
    is_valid = (x > 0) && (x <= imageSize(1)) && (y > 0) && (y <= imageSize(2));
end
```

تابع چک برای داخل صفحه بودن همسایه

حالا که آبجکت هارا شناسایی کردیم، کافیست آنهایی که سایزشان کمتر از مرز تعیین شده در ورودی تابع است را از تصویر حذف کنیم(مقدار آن را صفر کنیم).

```
    toErase = false(size(binaryimage));
    for p = 1:size(object_set, 2)
        if size(object_set{p}, 2) < n
            indices = sub2ind(size(binaryimage), object_set{p}(1, :), object_set{p}(2, :));
            toErase(indices) = 1;
        end
    end

    cleanimage = binaryimage;
    cleanimage(toErase) = 0;
end
```

حذف آبجکت های کوچک

برای استفاده کردن از این تابع، باید ابتدا با دادن یک عدد کوچک به عنوان مرز حذف کردن، آبجکت های کوچک که نویز هستند را حذف کنیم.

سپس در یک متغیر جدید با دادن یک عدد بزرگ به تابع، پس زمینه ی پلاک را بدیست آورده و سپس از تصویر حذف میکنیم.

```
noiseless_img = myremovecom(binary_image, 400);
background = myremovecom(binary_image, 2300);
clean_image = logical(noiseless_img - background);
```

حذف آبجکت های اضافی

تصویر تمیز شده

6- در وهله ی اول برای شناسایی آبجکت ها دقیقا مانند تابع قبلی عمل میکنیم.

```
function [labeled_objcs, max_num]=mysegmentation(image)
    adjacency_row = [ 1, 1, 1, 0, 0, -1, -1, -1];
    adjacency_col = [-1, 0, 1, -1, 1, -1, 0, 1];

    object_set = {};
    visited = zeros(size(image));

    [rows, cols] = size(image);
    ones = [];
    for i = 1:cols
        for j = 1:rows
            if image(j, i) == 1
                ones = [ones [j; i]];
            end
        end
    end

    for k = 1:size(ones, 2)
        current_object = [];
        object_queue = [];

        if visited(ones(1, k), ones(2, k))
            continue;
        else
            visited(ones(1, k), ones(2, k)) = 1;
            object = ones(:, k);
            object_queue = ones(:, k);
        end
```

شناسایی آبجکت ها قسمت اول

```
    while ~isempty(object_queue)
        currentX = object_queue(1, 1);
        currentY = object_queue(2, 1);
        object_queue(:, 1) = [];

        for m = 1:size(adjacency_row, 2)
            newX = currentX + adjacency_row(m);
            newY = currentY + adjacency_col(m);

            if is_valid_coordinate(newX, newY, [rows, cols]) && ...
               ~visited(newX, newY) && ...
               image(newX, newY) == 1

                  visited(newX, newY) = 1;
                  object_queue = [object_queue [newX; newY]];
                  object = [object [newX; newY]];
            end
        end
    end

    object_set = [object_set object];
end
```

شناسایی آبجکت ها قسمت دوم

7- ابتدا دیتا ست را لود کرده و سپس با تعیین یک عدد مرزی مناسب برای تشخیص کاراکتر ادامه میدهیم.

```
map_list = dir('p1\MapSet');
mapset = cell(2, size(map_list, 1) - 2);
for i = 3:size(map_list)
    name = fullfile('p1\MapSet', map_list(i).name);
    mapset{1, i - 2} = imread(name);
    mapset{2, i - 2} = map_list(i).name(1);
end

corr_threshold = 750;
recognized_chars = '';
```

لود مپ ست و تعیین مرز

سپس با یک حلقه در آبجکت های بدست آمده، به ازای هر آبجکت ابتدا تصویر این آبجکت را بصورت جداگانه در یک متغیر ذخیره میکنیم. سپس این تصویر را با ابعاد دیتاست هم سایز میکنیم.

حالا این آبجکت را با تمام دیتاست correlation میگیریم و در یک متغیر ذخیره میکنیم.

و اکنون بین این مقدار های ذخیره شده، ماکسیمم میگیریم؛ اگر این ماکسیمم از عدد مرزی تعیین شده بیشتر بود یعنی این کاراکتر تصویر را تشخیص داده ایم و کار تمام است.

```
for i = 1:max_num
    [row, col] = size(labeled_objects);
    current_label = [];
    for j = 1:col
        for k = 1:row
            if labeled_objects(k, j) == i
                current_label = [current_label [k; j]];
            end
        end
    end
    current_object = clean_image(min(current_label(1, :)):max(current_label(1, :)),...
        min(current_label(2, :)):max(current_label(2, :)));
    current_object = imresize(current_object, [42 24]);
    corr = zeros(1, size(mapset, 2));
    for j = 1:size(mapset, 2)
        bit_xor = ~bitxor(current_object, mapset{1, j});
        corr(j) = sum(bit_xor, 'all');
    end
    [max_corr, corr_index] = max(corr);
    if max_corr > corr_threshold
        recognized_chars = [recognized_chars mapset{2, corr_index}];
    end
end
```

تشخیص کاراکتر داخل تصویر

```
file = fopen('output.txt', 'wt');
fprintf(file, '%s\n', recognized_chars);
fclose(file);
fprintf('%s\n', recognized_chars);
```

پرینت جواب

تصویر تمیز شده اول

```
>> p1
DL5CH8855
```

خروجی تصویر اول

تصویر تمیز شده دوم

```
>> p1
DL2CAD0311
```

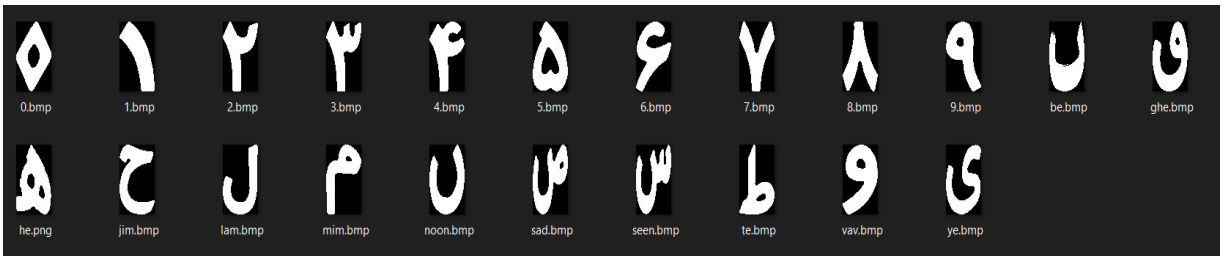
خروجی تصویر دوم

تصویر تمیز شده سوم

```
>> p1
UP14CB7145
```

خروجی تصویر سوم

قسمت دوم)



دیتابیس تهیه شده

این قسمت هم دقیقا مانند کد قسمت قبل است. با این تفاوت که در این دیتابیس اسم ها بیش از یک کاراکتر هستند و کافیست فقط در زمان سیو کردن نام دیتا در متغیر، از اولین کاراکتر تا قبل از کاراکتر نقطه را سیو کنیم(بعد از نقطه، فرمت فایل نوشته شده است).

```
[file, path] = uigetfile({'*.jpg;*.bmp;*.png;*.tif'}, 'Choose an Image');
image = imread([path, file]);

image = imresize(image, [300 500]);

gray_image = mygrayfun(image);

threshold = 247 * graythresh(gray_image);

binary_image = ~mybinaryfun(gray_image, threshold);

noiseless_img = myremovecom(binary_image, 500);
background = myremovecom(binary_image, 7000);
clean_image = logical(noiseless_img - background);

figure;
imshow(clean_image);

[labeled_objects, max_num] = mysegmentation(clean_image);

map_list = dir('p2\MapSet');
mapset = cell(2, size(map_list, 1) - 2);
for i=3 : size(map_list)
    filename = fullfile('p2\MapSet', map_list(i).name);
    mapset{1,i-2} = imread(filename);
    name = map_list(i).name;
    for j = 1 : length(name)
        if(name(j) == '.')
            break;
        end
    end
    mapset{2,i-2} = name(1:j-1);
end
```

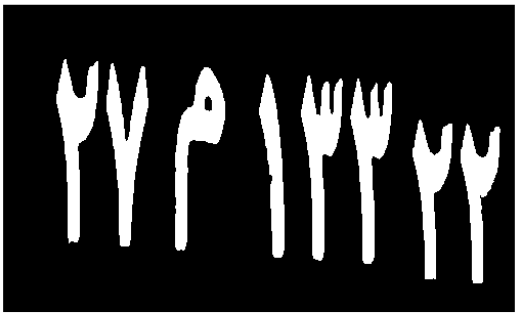
قسمت اول کد

```
corr_threshold = 750;
recognized_chars = '';

for i = 1:max_num
    [row, col] = size(labeled_objects);
    current_label = [];
    for j = 1:col
        for k = 1:row
            if labeled_objects(k, j) == i
                current_label = [current_label [k; j]];
            end
        end
    end
    current_object = clean_image(min(current_label(1, :)):max(current_label(1, :)),...
        min(current_label(2, :)):max(current_label(2, :)));
    current_object = imresize(current_object, [84 48]);
    corr = zeros(1, size(mapset, 2));
    for j = 1:size(mapset, 2)
        bit_xor = ~bitxor(current_object, mapset{1, j});
        corr(j) = sum(bit_xor, 'all');
    end
    [max_corr, corr_index] = max(corr);
    if max_corr > corr_threshold
        recognized_chars = [recognized_chars mapset{2, corr_index}];
    end
end

file = fopen('output.txt', 'wt');
fprintf(file, '%s\n', recognized_chars);
fclose(file);
fprintf('%s\n', recognized_chars);
```

قسمت دوم کد



شکل تمیز شده پلاک

```
>> p2
27mim13322
```

خروجی پلاک

قسمت سوم)

ابتدا در یک تابع، پلاک را از عکس جدا کرده و سپس دقیقا کارهای قسمت قبل را انجام میدهیم.

در این تابع، از قاب آبی رنگ سمت چپ قاب پلاک برای جدا کردن آن از عکس استفاده میکنیم.

از آنجایی که ممکن است این عکس از فاصله های مختلف از پلاک گرفته شده باشد، این قاب آبی را در سایز های مختلف ذخیره کرده و با قسمت های مختلف عکس correlation میگیریم و قسمتی از عکس که جوابش ماکسیمم شد، قاب پلاک ما است.

برای سایز بندی های مختلف، از سایز قاب آبی رنگ از 0 تا 34 تا کم کردم و اول در یک حلقه به ازای هرکدام از این قاب های آبی، ماکسیمم ضرب داخلی را ذخیره کردم؛ سپس در این 35 تا عدد، ماکسیمم همان جواب نهایی برای مکان قاب است.

گوشه ی سمت راست و بالای قاب به عنوان مختصات قاب پلاک به همراه سایز طولی و عرضی خروجی داده میشود و با این مقادیر، مختصات دقیق قاب پلاک به دست خواهد آمد.

دقت شود که برای ضرب داخلی گرفتن این تصویر، باید برای مقادیر قرمز، سبز و آبی این ضرب داخلی را جدا حساب کنیم و سپس میانگین بگیریم.

و همچنین در نهایت باید چک کنیم که سمت راست پلاک از تصویر بیرون نزند(ممکن است سایز طولی بیش از حد مجاز شود) و اگر این سایز بیش از حد بود، قاب پلاک را تا انتهای سمت راست تصویر در نظر میگیریم.

و همچنین دقت شود که این قاب بدست آمده را در تابع، به چپ و راست و بالا و پایین کمی اضافه کردم تا چیزی از قاب از دست نرود.

برای تمیزی کد، این کارهارا بین توابع مختلف پخش کردم که کار هر تابع بدین صورت است:

- my_corr:
  این تابع ضرب داخلی تصویر و قاب آبی رنگ را محاسبه میکند(به روش میانگین گیری گفته شده)
- best_corr:
  این تابع بین مقادیر ضرب داخلی داده شده ماکسیمم را پیدا کرده(یعنی مکان مناسب در تصویر را پیدا میکند) و سپس مختصات مورد نیاز و سایز ها را بدست می آورد.
- best_pos:
  به ازای هر کدام از 35 تا از قاب های آبی ضرب داخلی میگیرد(با استفاده از تابع my_corr) و سپس ماکسیمم آنها را بدست می آورد.
- find_plate:
  این تابع اصلی است. و در آن قاب آبی را لود کرده؛ 35 تا سایز های مختلف را ساخته و با استفاده از تابع های گفته شده، مختصات و سایزها را بدست می آوریم.

- max_x:
  این تابع برای چک کردن این است که قاب پلاک از تصویر بیرون زده یا خیر و مقدار سایز طولی صحیح را برمیگرداند.

```
[file, path] = uigetfile({'*.jpeg;*.jpg;*.bmp;*.png;*.tif'}, 'Choose an Image');
image = imread([path, file]);
image = imresize(image,[NaN 800]);

plate = find_plate(image);
max_x = find_max_x(image, plate);
image = image(plate(1) : plate(1) + plate(3), plate(2) : max_x, : );

image = imresize(image, [300 500]);

gray_image = mygrayfun(image);

threshold = 249*graythresh(gray_image);

binary_image = ~mybinaryfun(gray_image, threshold);

noiseless_img = myremovecom(binary_image, 450);
background = myremovecom(binary_image, 9000);
clean_image = logical(noiseless_img - background);

figure;
imshow(clean_image)

[labeled_objects, max_num] = mysegmentation(clean_image);

map_list = dir('p2\MapSet');
mapset = cell(2, size(map_list, 1) - 2);
```

Codes 1

```
for i=3 : size(map_list)
    filename = fullfile('p2\MapSet', map_list(i).name);
    mapset{1,i-2} = imread(filename);
    name = map_list(i).name;
    for j = 1 : length(name)
        if(name(j) == '.')
            break;
        end
    end
    mapset{2,i-2} = name(1:j-1);
end

 corr_threshold = 3100;
 recognized_chars = '';
```

Loading mapset & initialize threshold

```
for i = 1:max_num
    [row, col] = size(labeled_objects);
    current_label = [];
    for j = 1:col
        for k = 1:row
            if labeled_objects(k, j) == i
                current_label = [current_label [k; j]];
            end
        end
    end
    current_object = clean_image(min(current_label(1, :)):max(current_label(1, :)),...
        min(current_label(2, :)):max(current_label(2, :)));
    current_object = imresize(current_object, [84 48]);
    corr = zeros(1, size(mapset, 2));
    for j = 1:size(mapset, 2)
        bit_xor = ~bitxor(current_object, mapset{1, j});
        corr(j) = sum(bit_xor, 'all');
    end
    [max_corr, corr_index] = max(corr);
    if max_corr > corr_threshold
        recognized_chars = [recognized_chars mapset{2, corr_index}];
    end
end
```

Redognize chars

```
file = fopen('output.txt', 'wt');
fprintf(file, '%s\n', recognized_chars);
fclose(file);
fprintf('%s\n', recognized_chars);
```

Print output

```
function plate = find_plate(image)
    bluestrip = imread('p3\bluestrip_big.png');

    changed_bluestrips = cell(1,35);
    for i=1:35
        changed_bluestrips{i}=imresize(bluestrip,[NaN size(bluestrip,2)-i+1]);
    end

    pos = best_pos(image, changed_bluestrips);

    plate = [pos(1) - 10 pos(2) - 10 pos(3) + 20 pos(4) * 14];
    figure
    imshow(image);
    rectangle('Position',[plate(2) plate(1)...
        plate(4) plate(3)], 'EdgeColor', 'g', 'LineWidth', 2);
end

function pos = best_pos(image, changed_bluestrips)
    current_corr=0;
    for i=1:35
        [maxcorrval,rect] = my_corr(changed_bluestrips{1,i},image);
        if(maxcorrval>current_corr)
            current_corr=maxcorrval;
            pos=rect;
        end
    end
end
```

Find_plate & best_pos functions

```
function [corr, coordinate] = my_corr(changed_blu, image)
    r_corr = normxcorr2(changed_blu(...
        : , : , 1), image( : , : , 1));
    g_corr = normxcorr2(changed_blu(...
        : , : , 2), image( : , : , 2));
    b_corr = normxcorr2(changed_blu(...
        : , : , 3), image( : , : , 3));

    all_corrs = (r_corr + g_corr + b_corr) / 3;

    [corr, coordinate] = best_corr(changed_blu, all_corrs);
end

function [corr, coordinate] = best_corr(changed_blus, all_corrs)
    [corr,indx] = max(abs(all_corrs( : )));
    [col,row] = ind2sub(size(all_corrs), indx);
    left_col = col - size(changed_blus, 1);
    up_row = row - size(changed_blus, 2);
    coordinate = [left_col, up_row, size(...
        changed_blus, 1), size(changed_blus, 2)];
end

function max_x = find_max_x(image, plate)
    max_x = plate(2)+plate(4);
    if(max_x > size(image, 2))
        max_x = size(image, 2);
    end
end
```

my_corr & best_corr & max_x functions

تصویر اولیه 1

قاب تمیز شده 1

```
>> p3
34ye77410
```

خروجی 1

تصویر اولیه 2

قاب تمیز شده 2

```
>> p3
62ghe37846
```

خروجی 2