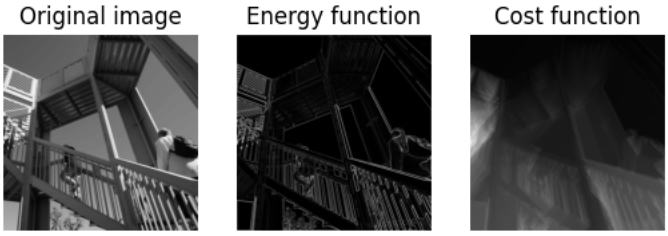
Heimaverkefni 4. Dæmi 5

Fill in the code below and answer the questions at the end.

```
import numpy as np
from scipy import ndimage, datasets
import matplotlib.pyplot as plt

def energy(img):
    # Calculate energy (importance) matrix using the Sobel convolutional operator
    #
    # Parameters:  img : Image (numpy array)
    #
    # Returns:     E:    Energy matrix (numpy array)

    # Compute gradient in x- and y- directions using the Sobel kernel
    fx = ndimage.sobel(img,axis=0,mode='constant')
    fy = ndimage.sobel(img,axis=1,mode='constant')

    return np.sqrt(fx**2 + fy**2)

def minimize(E):
    n, m = E.shape
    C = np.zeros((n, m)) # Cost matrix
    P = np.zeros((n, m), dtype=int) # Parent pointers, use int for indices

    # Initialize the first row of C with the first row of E
    C[0, :] = E[0, :]

    for i in range(1, n):
        for j in range(m):
            # Handle the left edge
            if j == 0:
                idx_min = np.argmin(C[i-1, j:j+2])
                C[i, j] = E[i, j] + C[i-1, j+idx_min]
                P[i, j] = j+idx_min
            # Handle the right edge
            elif j == m-1:
                idx_min = np.argmin(C[i-1, j-1:j+1])
                C[i, j] = E[i, j] + C[i-1, j-1+idx_min]
                P[i, j] = j-1+idx_min
            # Handle the general case
            else:
                idx_min = np.argmin(C[i-1, j-1:j+2])
                C[i, j] = E[i, j] + C[i-1, j-1+idx_min]
                P[i, j] = j-1+idx_min

    # Adjust P to store relative parent position (-1, 0, or 1)
    for i in range(1, n):
        for j in range(m):
            P[i, j] -= j

    return C, P

def get_path(C, P):
    n, m = C.shape
    path = []

    # Find the column index with the minimum cost in the bottom row
    min_cost_index = np.argmin(C[-1])

    # Initialize the current position with the min cost index at the bottom row
    current_position = min_cost_index

    # Start from the bottom row and move upwards
    for i in range(n-1, -1, -1):
        path.append(current_position)
        # Update the current position based on the parent pointer
        # Note: Since we are moving upwards, we subtract the parent pointer value
        # to get the next column index in the path
        if i > 0:  # No parent for the first row, so skip updating position when i == 0
            current_position += P[i, current_position]

    # Reverse the path to start from the top to the bottom
    path.reverse()

    return path


def vertical_cut(img, path):
```

```
def vertical_cut(img, path):
    # Cuts an image across a given path
    # Parameters: img : Image (numpy array)
    #             path : The path where the cut is performed (list)
    #                    path[0] corresponds to the bottom row
    #
    # Returns:    img_cut: Image after removing pixels along the path

    # NOTE: Feel free to rewrite this function
    n, m = img.shape
    img_cut = np.zeros((n, m - 1))
    for i in reversed(range(n - 1)):
        img_cut[i, 0:path[-i]] = img[i, 0:path[-i]]
        img_cut[i, path[-i]:] = img[i, (path[-i]+1):]

    return img_cut

def overlay_path(img, path):
    # Overlays the path on the original image for (visualization purposes only)
    # Same parameters as for vertical_cut

    n = img.shape[0]
    img_path = img.copy()
    img_path[n - 1, path[0]] = 1
    for i in reversed(range(n - 1)):
        img_path[i, path[-i]] = 1

    return img_path

# Load test image
img = datasets.ascent()
img = img / 255.0 # Convert to floats in the range [0,1]
n, m = img.shape
print("Image size: ", img.shape)

# Compute energy and perform minimization
E = energy(img)
C, P = minimize(E)
path = get_path(C, P)

f, (ax1, ax2, ax3) = plt.subplots(1, 3)
ax1.imshow(img,cmap=plt.cm.gray)
ax1.set_title('Original image')
ax1.set_axis_off()
ax2.imshow(E, cmap=plt.cm.gray)
ax2.set_title('Energy function')
ax2.set_axis_off()
ax3.imshow(C, cmap=plt.cm.gray) # C is currently a zero matrix
ax3.set_title('Cost function')
ax3.set_axis_off()
```

```
Image size:  (512, 512)
```




```
# Visualize a single cut

# NOTE: Does not work until the minimize and get_path functions
#       have been completed

img_path = overlay_path(img, path)
plt.imshow(img_path, cmap=plt.cm.gray)
plt.show()

# Perform the cutout and display the modified image
# There should be practically no difference
img_cut = vertical_cut(img, path)
plt.imshow(img_cut, cmap=plt.cm.gray)
print("Size of cut image:", img_cut.shape)
plt.show()
```

Additional tasks

Now repeat the deletion several times, e.g. 512/4=128 times (25% reduction in horizontal size).

- Are the results acceptable
- Compare with the naive strategy of removing 128 equally spaced columns from the original matrix (it is convenient to use numpy.delete for this operation).
- Assume that the energy-based algorithm removes unwanted pieces of an image, e.g., a face of a person. Can you suggest a simple strategy to address this issue?
- Suggest a stretegy for *stretching* an image, using a similar strategy

Greining reiknirita – Heimadæmi 4

1. Þú ert tölvan (⋆)

Lengsta vaxandi hlutruna (e. subsequence) rununnar $A[1 \dots n]$ er hlutruna sem hefur þann eiginleika að $A[i] < A[k]$ $fyrir$ $i < k$.

Dæmi: Lengsta vaxandi hlutruna $1, 4, 1, 4, 2, 1, 3, 5, 6, 2, 3, 7$ er runan $1, 2, 3, 5, 6, 7$.

Ef $L(i)$ táknar lengstu vaxandi hlutrunu $A[i \dots n]$ sem inniheldur stakið $A[i]$ þá má reikna L með rakningarvenslunum $(i < k \leq n)$

$$L(i) = max\{L(k) + 1 \mid i < k \leq n \; og \; A[i] < A[k]\} \cup \{1\}$$

fyrir $i = n, n-1, \dots, 1$. Max-aðgerðin skilar stærsta staki í menginu. Með því að bæta stakinu 1 við mengið þarf ekki að hafa áhyggjur af því að taka max af tómu mengi. Ennfremur felur það í sér grunnskilyrðið $L(n) = 1$. Með því að halda bókhald um hvaða gildi á k gefa hæsta $L(k)$ má rekja sig til baka í lokin og endurskapa lengstu vaxandi hlutrunu. Þetta má t.d. gera með því að útbúa fylki P sem er jafnstórt L sem heldur utan um niðurstöður úr max aðgerðunum, þ.e. hvaða gildi á k vinnur "max-keppnina" í hvert skipti2

a) **Ákvarðið gildin á L fyrir rununa $A = 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8$.**
b) **Útbúið fylki P sem heldur utan um niðurstöður úr max-aðgerðunum og notið það til að ákvarða sjálfa rununa.**

SVAR:

Röðum þessu upp í tölfu þá er þetta skilningsríkt:

index	1	2	3	4	5	6	7	8	9	10	11	12
A	3	1	4	1	5	9	2	6	5	3	5	8
L	5	5	4	5	3	1	4	2	2	3	2	1
P	2	2	4	6	7	0	9	11	11	10	11	0

Nú út frá þessu getum við séð LIS sem er [3,4,5,6,8].

2. Teningar (⋆)

(Gamalt prófdæmi) Setjið fram reiknirit sem byggir á kvikri bestun sem ákvarðar fjölda leiða til að fá summuna m þegar n sex hliða teningum er kastað.

SVAR:

i) Undirverkefnum í orðum: Skilgreinum $f(m,n)$ sem fjölda möguleika til að fá summu m þegar við notum 6 hliða tening n sinnum.

ii) Rakningarvenslum og rökstyðja hvernig þau eru fengin (hafið í huga ákvörðun/ágiskun sem tekin er í hverju skrefi): Getum sett fram að til að fá summu m með n teningum, þá skoðum við fjölda leiða til að ná $m-1$ með $n-1$ teningum og leggjum allar mögulegar útkomur i af síðasta tengnum þ.e. 1-6. Þá eru rakningarvenslin $f(m,n) = f(m-1,n-1) + f(m-2,n-1) + \ldots + f(m-6,n-1)$. Hér er hver liður í summunni sami og ein möguleg útkoma frá síðasta teningi og eftir stendur summa sem þarf frá hinum teningum.

iii) Röð undirverkefna: Leysum undirverkefni með því að byrja á minnsta undirverkefni eða $f(m,1)$ fyrir öll m frá 1-6, gerum þetta þar til við komum að $f(m,n)$.

iv) Grunntilviki:
1) n = 0 og m != 0 þá fyrir öll $m > 0$, $f(m,0) = 0$
2) n = 0 og m = 0 þá $f(0,0) = 1$
3) $f(m,1) = 1$ fyrir m frá 1-6 (ein útkoma með að kasta einum tening)
4) $f(m,1) = 0$ fyrir $m > 6$ og $m < 1$ (ekki hægt að fá með einum tenging).
5) $f(m,n) = 0$ fyrir öll $m < n$
6) $f(m,n) = 0$ fyrir öll $m > 6n$

v) Hvernig lausn á upphaflegu verkefni er fengin: Þegar við erum búinn að fylla í töflu með endurkvæmu rakningarvenslunum þá mun gildið á $f(m,n)$ gefa okkur fjölda leiða til að fá summu m með n teningum.

vi) Hvernig hægt er að koma í veg fyrir endurtekna útreikninga með því að geyma milliniðurstöður í viðeigandi gagnagrind: Komum í veg fyrir endurtekinn útreikning með því að geyma niðurstöður í töflu og vísa í geymdu gildin í stað þess að endureikna þau.

vii) Tímaflækju og rökstyðja hvernig hún er fengin: $O(n * m * 6)$ eða margliðu tímaflækja, því fyrir hvert stak í töflunni $f(m,n)$ hvert skipti sem framkvæmum við 6 útreikninga fyrir hverja hlið tenings.. getum því sagt $O(n * m)$

3. Einmenningsspil (⋆⋆)

(Gamalt prófdæmi) Athugum einmenningsspil sem er spilað á $1 \times n$ reita borði. Leikandinn raðar 2×1 kubbum á borðið þannig að kubbarnir eru láréttir. Það má setja eins marga kubba og maður vill á borðið nema hver reitur á borðinu er í mesta lagi þakinn af einum kubb og allir kubbarnir eru inni á borðinu. Í hverjum reit er heiltala sem getur verið bæði jákvæð eða neikvæð og táknar hún hversu mikið maður græðir eða tapar á að þekja viðkomandi reit. Stig í leiknum eru summa allra talna í reitum sem eru þakktir ($(7 - 3) + (3 + 4) = 11$ í eftirfarandi dæmi).

-4	7	-3	2	3	4	-3	2

Setjið fram reiknirit sem byggir á kvikri bestun sem ákvarðar mesta mögulega fjölda stiga sem hægt er að fá. Inntak í reikniritið er listi n heiltalna. Hver er tímaflækja reikniritsins?

SVAR:

i) <u>Undirverkefnum í orðum:</u> Fyrir hvert undirverkefni $S(i)$ þá ákvörðum við hæsta stig sem hægt er að fá með því að setja flísar á borðið frá fyrstu stöðu upp í stöðu i.

ii) <u>Rakningarvenslum og rökstyðja hvernig þau eru fengin (hafið í huga ákvörðun/ágiskun sem tekin er í hverju skrefi):</u> Við getum sett flís í stöðu i og $i - 1$ eða ekki, ef við setjum flís þá bætum við gildunum á i og $i - 1$ við útkomu úr undirverkefni $S(i - 2)$ það er vegna þess að við getum sett í stöðu $i - 1$ með annarri flís. Ef við setjum ekki flís í i er útkoman $S(i - 1)$ Rakningarvenslin eru því: $S(i) = max(S(i - 1), S(i - 2) + board[i] + board[i - 1])$ fyrir öll $i \geq 3$

iii) <u>Röð undirverkefna:</u> Leysum þau í vaxandi röð og byrjum frá minnstu borða stærð og fyllum upp í n.

iv) <u>Grunntilviki:</u>
1. S(0) = 0 (ekkert borð til að setja flísar á)
2. S(1) = 0 (2*1 flís passar ekki á 1*1 borð)
3. S(2) = board[0] + board[1] (setja flís á borð)

v) <u>Hvernig lausn á upphaflegu verkefni er fengin:</u> Gildið á $S(n)$ mun gefa okkur maximum stig sem hægt er að fá fyrir borðið.

vi) <u>Hvernig hægt er að koma í veg fyrir endurtekna útreikninga með því að geyma milliniðurstöður í viðeigandi gagnagrind:</u> Geymum útkomu af hverju undirverkefni $S(i)$ í fylki svo hvert undireverkefni er útreiknað aðeins einusinni. Þetta fylki hagar sér eins og memoization tafla.

vii) <u>Tímaflækju og rökstyðja hvernig hún er fengin:</u> Tímaflækjan er línuleg eða $O(n)$ það er því hvert undirvandamál er reiknað einusinni og er alltaf tvær samlagningar eða einn samanburður.

4. Öl er böl (⋆⋆)

Ruddabrugg ehf verður með m bjórtunnur á lager í næsta mánuði. Fyrirtækið hefur fengið n pantanir frá mögulegum kaupendum þar sem i-ta pöntun er viljayfirlýsing um að kaupa a_i tunnur fyrir p_i þúsund (heildarverð, ekki per tunnu). Verðið getur verið bæði jákvætt eða neikvætt . Sérhver pöntun er annaðhvort afgreidd að fullu eða hún er ekki afgreidd og einungis er hægt að afgreiða hverja pöntun einu sinni. Ruddabrugg þarf ekki að selja allar tunnurnar en þá þarf fyrirtækið að borga s þúsund í förgunarkostnað fyrir hverja óselda tunnu. Ruddabrugg vill hámarkja hagnaðinn, þ.e. tekjur mínus gjöld.

Setjið fram kvikt bestunarreiknirit til að ákvarða hámarkshagnað í næsta mánuði. Ábending: Bakpokaverkefnið.

SVAR:

i) Undirverkefnum í orðum: Skilgreinum $P(j,k)$ sem hámarks hagnað fyrir fyrstu j pantanir og k bjórtunnur eftir.

ii) Rakningarvenslum og rökstyðja hvernig þau eru fengin (hafið í huga ákvörðun/ágiskun sem tekin er í hverju skrefi): Hér þurfum við að hafa í huga ef við afgreiðum pöntum j (þ.e. ef við eigum nóg af brjótunnum k) eða ekki. Því eru 2 rakningarvensl.

$$P(j,k) = max(P(j-1,k), P(j-1,k - aj) + pj) \; ef \; k \geq a\,j$$

$$P(j,k) = P(j-1,k) \; ef \; k < aj$$

iii) Röð undirverkefna: Leysum undirverkefni í vaxandi röð á pöntunum j og fyrir hvert j í minnkandi tunnum k.

iv) Grunntilviki:

1. $P(0,k) = -s * k$ fyrir öll k
2. $P(j,0) = 0$ fyrir öll j

v) Hvernig lausn á upphaflegu verkefni er fengin: Maximum hagnaður fyrir næsta mánuð er $P(n,m)$ eftir að búið er að skoða allar n pantanir og m tunnur í byrjun.

vi) Hvernig hægt er að koma í veg fyrir endurtekna útreikninga með því að geyma milliniðurstöður í viðeigandi gagnagrind: Geymum gildi á hverju $P(j,k)$ í töflu svo hvert undirverkefni er aðeins reiknað einusinni. Taflan hagar sér eins og memoization tafla eða kvik bestunar tafla.

vii) Tímaflækju og rökstyðja hvernig hún er fengin: Tímaflækjan er $O(n * m)$ eða margliðu tímaflækja. Vegna þess að við höfum n pantanir og þurfum mögulega að finna maximum hagnað fyrir hvert k frá m niður í 0. Þetta tekur allt fasta tíma. Samanburð eða samlagningu.

5. Myndvinnsla með kvikri bestun (⋆ ⋆ ⋆)

SVAR: