1.ZAD.

```
FIND_PATH (BT a, SUMA)
    PUTOVI = []
    DFS (a.root, SUMA, [], PUTOVI)
    RETURN PUTOVI

DFS (NODE, SUMA, PUT, PUTOVI)
    if (node == NULL):
        return
    put.append (node.v)
    if (node.left == NULL AND node.right == NULL)
        if SUM(PUT) == SUMA:
            putovi.append (PUT)
    else:
        DFS (node.left, SUMA, PUT, PUTOVI)
        DFS (node.right, SUMA, PUT, PUTOVI)
    put.pop()
```

Za ovaj algoritam koristio sam dvije funkcije. Prva funkcija koristi se za poziv DFS na određenom stablu uz određenu sumu. Dok druga funkcija radi sva računanje i odabir prikladne putove.

Kod započinje pretragu od korijena stabla.

Ako je dani node null funkcija se ne izvršava. Za ostale slučajeve nadodajemo vrijednost noda u listu put. Ako dani node je list došli smo do kraja te grane i uspoređujemo sumu svih vrijednosti unutar liste put s traženom vrijednosti sume, te nadodajemo taj put u listu putova. Ako pak nismo došli do lista algoritam se poziva rekurzivno na lijevo i desno dijete.

Neovisno o tome jeli za danu granu pronađena odgovarajuća suma kada smo došli do lista, brišemo ~~nju~~ vrijednost iz liste jer vrijednost tog lista ne može utjecati na vrijednost neke druge grane koja ima isti parent node.

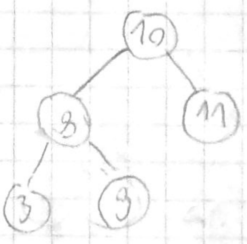
VSA: FIND_PATH je $O(1)$

DFS: zbroj svih vrijednosti u listi put $O(\lg n)$

DFS cijelog stabla prolazi kroz sve nodove stabla pretpostavimo li da ih ima n:

$O(n)$

FIND_PATH(a, 21)

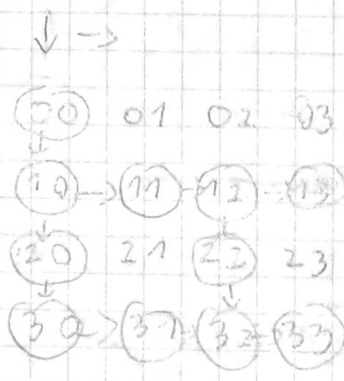


[[10, 8, 3], [10, 11]]

[10, ~~8~~, ~~3~~, 3, ~~11~~

2.ZAD. n x m matrica

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



```
FIND_PATH (n, m)
    for i in range (0, n)
        for j in range (0, m)
            NODOVI.PUSH_BACK (Node (i, j))
    PUTOVI = []
    PRETRAGA (0, 0, [], PUTOVI, NODOVI)
    RETURN (MIN (PUTOVI[i].length, i = 0 .. PUTOVI.length))

PRETRAGA (X, Y, PUT, PUTOVI, NODOVI)
if (x > m-1 or x < 0 or y > n-1 or y < 0)
    return
put.append (Node (x, y))
if (x == m-1 and y == n-1)
    putovi.append (put)
else
    if (node (x+1, y) == 0)
        PRETRAGA (x+1, y, put, putovi, nodovi)
    if (node (x, y+1) == 0)
        PRETRAGA (x, y+1, put, putovi, nodovi)
    if (node (x+1, y) != 0 and node (x, y+1) != 0)
        put.pop()
```

VSA: $O(n m)$: u najgorem slučaju proći ćemo

OPIS ALGORITMA 2. ZAD.

Find_path algoritam služi za poziv algoritma pretrage na početnim koordinatama labirinta (0,0) uz dvije prazne liste put i putovi te liste nodova. Find_path vraća put najkraće duljine iz liste putovi. Algoritam pretrage radi na principu da gleda sve nodove koji imaju donji ili desni dijete koje ima vrijednost nula, te ih nadodaje na put. Sve dok x,y koordinate noda ne budu jednake n-1,m-1 ili ne premaše granice matrice.

OPIS NAD A FUNKCIJE PRETRAGA: Prvi uvjet provjerava nalazi li se node unutar matrice, ako se ne nalazi izvršavanje se prekida. Nadodajemo novi node unutar puta. Provjeravamo je li taj node krajnji node, ako je nadodajemo put u putovi. Inače gledamo postoji li desni ili donji node te rekurzivno pozivamo algoritam na te nodove. Ako niti desni niti donji node ne postoje popamo zadnji node iz liste.

ZAD 4.

NAD_PAL(S)

DFS(SSI, R, NAD_PAL, S)

ZAD.4.

```
NAJ_PAL (S)
    NAJ_PAL_S = ""
    for i in range (len(S))
        DFS(1, i, NAJ_PAL_S, S)
        if (i < n-1 and S[i] = S[i+1])
            DFS (1, i+1, NAJ_PAL_S, S)
    return (NAJ_PAL_S)
```

```
DFS ( ST, K, NAJ_PAL_S, S)
    if (K - ST + 1 <= LEN(NAJ_PAL_S))
        RETURN
    if (S[ST..K+1] == S[ST..K+1][::-1]
        NAJ_PAL_S = S[ST..K+1]
    if ( ST > 0 and K < len(S)-1 and
        S[ST-1] == S[K+1])
        DFS (ST-1, K+1)
```

OPIS ALGORITMA: Funkcija NAJ_PAL poziva DFS nad jednim ili dva središnja elementa te vraća najdulji palindrom. DFS radi na principu da uspoređuje elemente oko tog središnjeg elementa, dok ne pronađe najdulji palindrom.

VSA: $O(n^2)$ jer DFS u worst-case će biti $O(n)$, a sam poziv DFS-a se nalazi unutar for petlje.

OPIS KODA DFS: Pozivamo funkciju na STARTNOM i KRAJNJEM INDEXU, STRINGOM NAJ_PAL_S koji predstavlja najdulji palindrom i S koji je sam string kojeg pretražujemo. Početni uvjet gleda jeli uopće moguće da u tom rasponu indexa se nalazi dulji palindrom od sadašnjeg. Drugi uvjet provjerava jeli jednaki niz od S u rasponu od [S, K+1] i njegov niz u obrnutom poretku. Ako jesu oni postaju najdulji palindrom. Zadnji uvjet provjerava možemo li rekurzivno pozvati DFS na veću okolinu oko središnjeg elementa