

Rekurzija 2

```

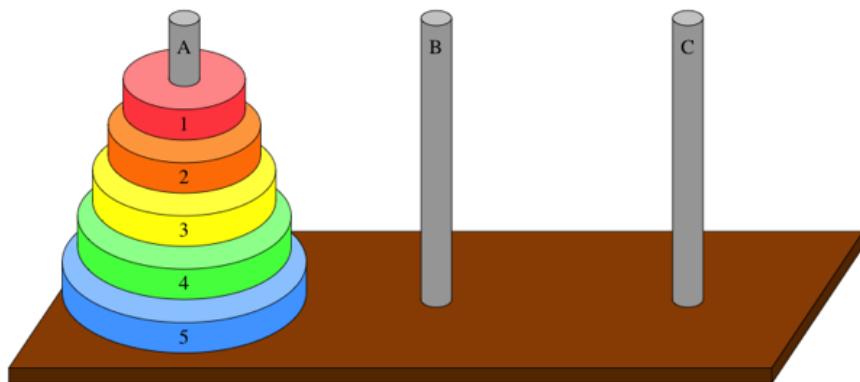
int f(int *p[10]) {...}
int main(int argc, char *args[]) {...}

int p; // spremenljivka tipa int
int p[10]; // p je tabela 10 intov
int *p[10]; /* je isto kot */ int *(p[10]); // p je tabela 10 kazalcev na int
int (*p)[10]; // p je pointer na tabelo 10 intov

```

Towers of Hanoi (Hanojski stolpi)

- odmikamo na sredino ($n-1$ obročev)
- če jih odmikamo sodo (**n-1 je sodo**), gre prvi na C
- če jih odmikamo liho (**n-1 je liho**), gre prvi na B
- število potez je $2^n - 1$
- osnovno nalogu razgradimo na dele za rekurzijo



5 obročev: A \rightarrow C preko B

- 4 obroče A \rightarrow B preko C (obroč 1 gre na C)
- 1 obroč A \rightarrow C
- 4 obroče B \rightarrow C (obroč 1 gre na A)

```

#include <stdio.h>

int steviloDiskov;

void hanoi(int n, char zaceten, char pomozen, char koncen){
    printf("zac: %c, pom: %c, konc: %c\n", zaceten, pomozen, koncen);
    if(n == 1){
        printf("\tPREMIK: %c -> %c\n", zaceten, koncen);
        return;
    }
    printf("v odmikanje diska %d: ", n-1);
}

```

```

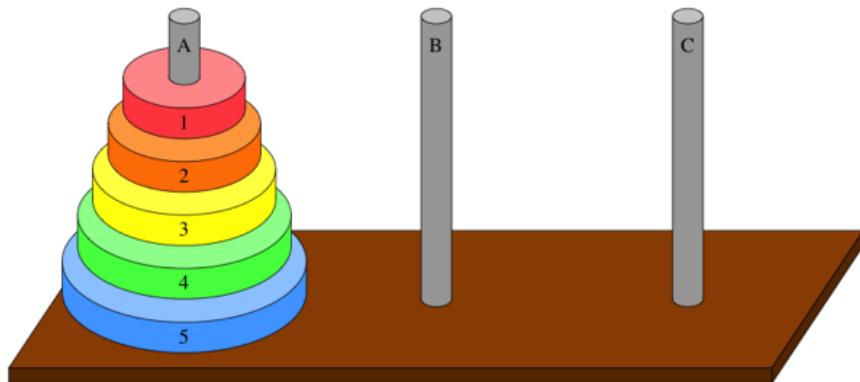
hanoi(n-1, zaceten, koncen, pomozen); // odmikanje diskov (od spodaj navzgor
gremo, izvaja se od zgoraj navzdol - DELOVANJE PODOBNO SKLADU, SAJ SO FUNKCIJE NA
SKLADU)

hanoi(1, zaceten, pomozen, koncen); // premik na ciljno pozicijo

printf("PREMIK NAZAJ\n");
hanoi(n-1, pomozen, zaceten, koncen); // premik odmaknjениh diskov na že
odmaknjene večje
}

int main()
{
    int n;
    printf("Enter the number of disks: ");
    scanf("%d", &n);
    steviloDiskov = n;
    hanoi(n, 'A', 'B', 'C');
    return 0;
}

```



Razlaga delovanja (po izpisih):

```

zac: A, pom: B, konc: C // PRVOTNI KLIC
v odmikanje diska 4: zac: A, pom: C, konc: B // NA SKLAD DAMO FUNKCIJE ZA ODMIK
n-1 DISKOV
v odmikanje diska 3: zac: A, pom: B, konc: C
v odmikanje diska 2: zac: A, pom: C, konc: B
v odmikanje diska 1: zac: A, pom: B, konc: C // VSE NALOŽENE NA SKLAD
    PREMIK: A -> C // IZVAJAMO IZ SKLADA (odmik diska 1)
zac: A, pom: C, konc: B
    PREMIK: A -> B // IZVAJAMO IZ SKLADA (odmik diska 2)
PREMIK NAZAJ // PREMIK MANJŠIH NAZAJ NA VEČJE, SAJ SMO TE ŽE UMAKNILI NA
POMOŽEN POLOŽAJ
zac: C, pom: A, konc: B
    PREMIK: C -> B
zac: A, pom: B, konc: C // IZVAJAMO IZ SKLADA (odmik diska 3)
    PREMIK: A -> C
PREMIK NAZAJ
zac: B, pom: A, konc: C

```

```
v odmikanje diska 1: zac: B, pom: C, konc: A
    PREMIK: B -> A
zac: B, pom: A, konc: C
    PREMIK: B -> C
PREMIK NAZAJ
zac: A, pom: B, konc: C
    PREMIK: A -> C
zac: A, pom: C, konc: B      // IZVAJAMO IZ SKLADA (odmik diska 4)
    PREMIK: A -> B
PREMIK NAZAJ
zac: C, pom: A, konc: B
v odmikanje diska 2: zac: C, pom: B, konc: A
v odmikanje diska 1: zac: C, pom: A, konc: B
    PREMIK: C -> B
zac: C, pom: B, konc: A
    PREMIK: C -> A
PREMIK NAZAJ
zac: B, pom: C, konc: A
    PREMIK: B -> A
zac: C, pom: A, konc: B
    PREMIK: C -> B
PREMIK NAZAJ
zac: A, pom: C, konc: B
v odmikanje diska 1: zac: A, pom: B, konc: C
    PREMIK: A -> C
zac: A, pom: C, konc: B
    PREMIK: A -> B
PREMIK NAZAJ
zac: C, pom: A, konc: B
    PREMIK: C -> B
zac: A, pom: B, konc: C      // PREMIK SPODNJEGA (5) NA KONČNI CILJ
    PREMIK: A -> C
PREMIK NAZAJ
zac: B, pom: A, konc: C
v odmikanje diska 3: zac: B, pom: C, konc: A
v odmikanje diska 2: zac: B, pom: A, konc: C
v odmikanje diska 1: zac: B, pom: C, konc: A
    PREMIK: B -> A
zac: B, pom: A, konc: C
    PREMIK: B -> C
PREMIK NAZAJ
zac: A, pom: B, konc: C
    PREMIK: A -> C
zac: B, pom: C, konc: A
    PREMIK: B -> A
PREMIK NAZAJ
zac: C, pom: B, konc: A
v odmikanje diska 1: zac: C, pom: A, konc: B
    PREMIK: C -> B
zac: C, pom: B, konc: A
    PREMIK: C -> A
PREMIK NAZAJ
zac: B, pom: C, konc: A
    PREMIK: B -> A
```

```

zac: B, pom: A, konc: C
    PREMIK: B -> C
PREMIK NAZAJ
zac: A, pom: B, konc: C
v odmikanje diska 2: zac: A, pom: C, konc: B
v odmikanje diska 1: zac: A, pom: B, konc: C
    PREMIK: A -> C
zac: A, pom: C, konc: B
    PREMIK: A -> B
PREMIK NAZAJ
zac: C, pom: A, konc: B
    PREMIK: C -> B
zac: A, pom: B, konc: C
    PREMIK: A -> C
PREMIK NAZAJ
zac: B, pom: A, konc: C
v odmikanje diska 1: zac: B, pom: C, konc: A
    PREMIK: B -> A
zac: B, pom: A, konc: C
    PREMIK: B -> C
PREMIK NAZAJ
zac: A, pom: B, konc: C
    PREMIK: A -> C

```

Kot lahko opazimo, je rekurzija razbitje problema na podprobleme in nato reševanje le teh tako, da postavimo funkcije na sklad v napačnem vrstnem redu in ko se sproži usatavitveni pogoj se te funkcije iz sklada izvedejo v pravem vrstnem redu

...naredi da funkcija vrne število potez

...naredi da vrne kateri obroč premaknemo kam

...naredi da izpiše:

5 4

3

2 1

Množenje matrik

n matrik: $m_1 \times m_2, m_2 \times m_3, m_3 \times m_4, \dots, m_n \times m_{n+1}$ (mali m-ji so dimenzijs matrik)

$M_1 \quad M_2 \quad M_3 \quad \dots \quad M_n$

$m \times m, m \times m \rightarrow m \times m$...skupaj... **m^3 množenj**

$m_1 \times m_2, m_2 \times m_3 \rightarrow m_1 \times m_3$...skupaj... **$m_1 m_2 m_3$ množenj** (vrstice x stolpci x stolci V Drugi)

$$(AB)C = A(BC)$$

$$((M_1 M_2) M_3) M_4$$

$$(M_1 M_2) (M_3 M_4)$$

(M₁ (M₂ M₃)) M₄

...

$M_1 \cdot M_2 \cdot M_3 \cdot \dots \cdot M_m$ vredenju na $(\underbrace{M_1 \cdot M_2 \cdot M_3 \dots M_i}_{l \text{ množenj skalarjev}}) \cdot (\underbrace{M_{i+1} \cdot M_{i+2} \cdot M_{i+3} \dots M_n}_{d \text{ množenj skalarjev}})$

$\underbrace{\overbrace{M_1 \times M_{i+1}}^{\text{velikost: } M_1 \times M_{i+1}} \cdot \overbrace{M_{i+1} \times M_{n+1}}^{\text{velikost: } M_{i+1} \times M_{n+1}}}_{M_1 \cdot M_{i+1} \cdot M_{n+1}}$

skupnoj: $(M_1 \cdot M_{i+1} \cdot M_{n+1}) + l + d$

$(M_1 \cdot M_2 \cdot M_3) \cdot M_4$

\vdots

$\begin{matrix} 0 & 1 & \dots & n \\ M_1, M_2, \dots, M_{i+1}, \dots, M_{n+1} \end{matrix}$ ← indeksi
← rednosti

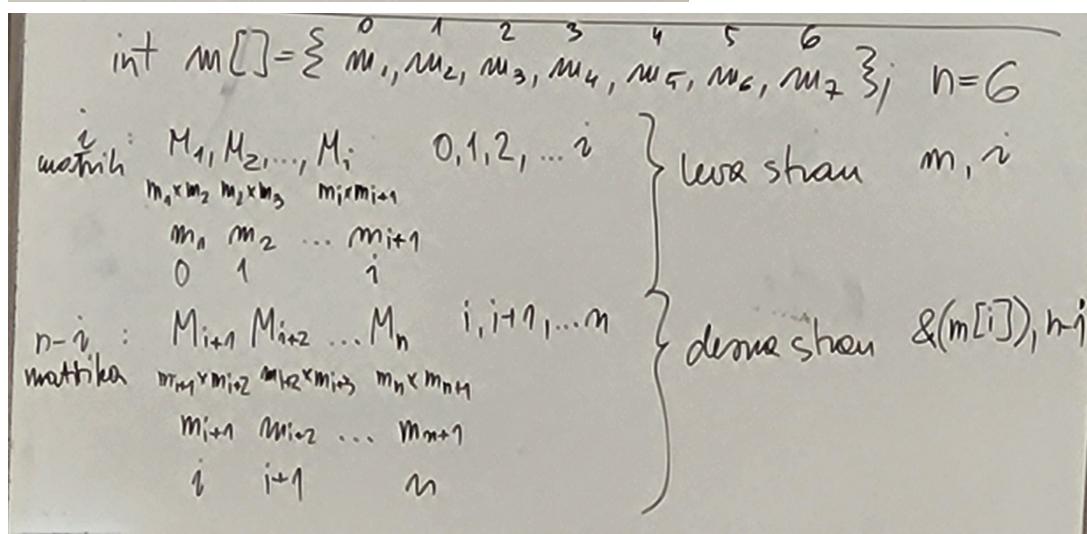
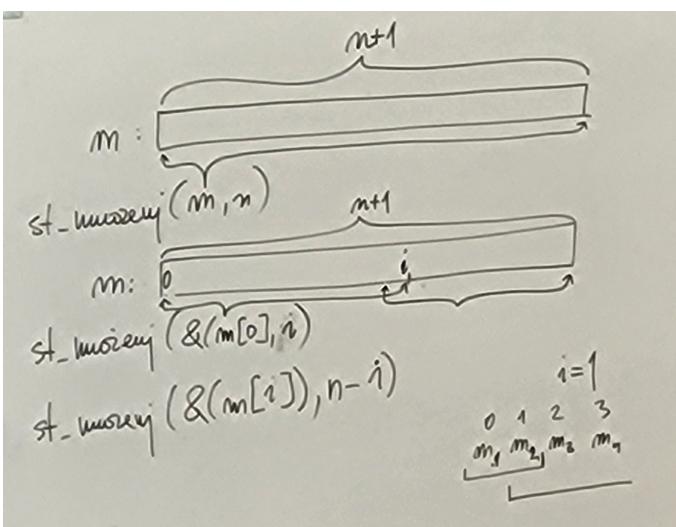
int $m[] = \{ \dots \}$

int m_j

```
#include <limits.h>

int st_mnozenj(int m[], int n){
    if(n == 1)
        return 0;

    int min = INT_MAX;
    for(int i = 1; i <= n-1; i++){ // i je število matrik na levi strani
        int l = st_mnozenj(&(m[0]), i); // &(m[0]) je isto kot m
        int d = st_mnozenj(&(m[i]), n-i);
        int vsehPriI = m[0] * m[i] * m[n] + l + d;
        if(vsehPriI < min){
            min = vsehPriI;
        }
    }
    return min;
}
```

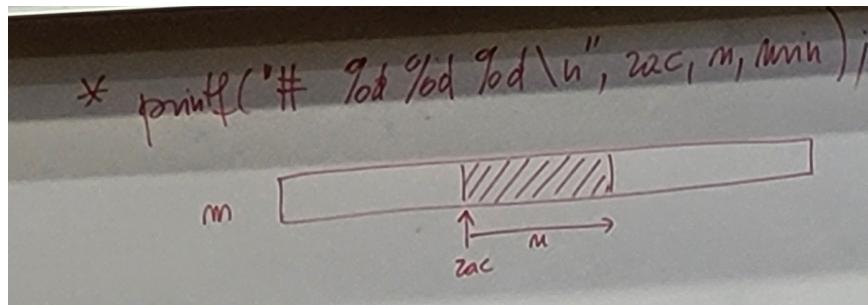


```

int m[] = {...};
int n;
int memo[1000][1000];

int st_mnozenj(int zac, int n){
    if(n == 1)
        return 0;
    if(memo[zac][n] != 0){
        return memo[zac][n];
    }
    int min = INT_MAX;
    for(int i = 1; i <= n-1; i++){
        int l = st_mnozenj(zac, i);
        int d = st_mnozenj(zac+i, n-i);
        int vsehPriI = m[zac] * m[zac+i] * m[zac+n] + l + d;
        if(vsehPriI < min){
            min = vsehPriI;
        }
    }
    printf("# %d %d %d\n", zac, n, min);
    memo[zac][n] = min;
    return min;
}

```



```
$ ./matrike | grep '#' | sort | wc
$ ./matrike | grep '#' | sort -u | wc
// z memoizacijo lahko prihranimo toliko časa
$ time ./matrike
```

- s permutacijami:

$$\begin{aligned}
 & M_1 \cdot M_2 \cdot M_3 \cdot M_4 \\
 & M_1^3(M_2^2(M_3 \cdot M_4)) \\
 & M_1^3((M_2 \cdot M_3)^2 M_4) \\
 & (M_1^2 M_2)^3(M_3 \cdot M_4) \\
 & (M_1 \cdot M_2)^3(M_3^2 M_4) \\
 & ((M_1 \cdot M_2)^2 M_3)^3 M_4 \\
 & (M_{12}(M_2 \cdot M_3))^3 M_4
 \end{aligned}$$