Bonyolultságelmélet gyakorlat – 02
Tárigény

Recap: tárigény

- a **tárigényt** szintén az input leírásához szükséges bitek n számának függvényében adjuk meg, szintén csak nagyságrendben („O-ban”)
- magát az inputot és az outputot **nem** kell beleszámolni, ha az inputot **csak olvassuk**
- ha egy `int` változóba futás közben a belekerülő maximális érték A, akkor az ő tárigénye $O(\log A)$
- ha a kódban nincs függvényhívás, akkor a tárigény a lokális változók tárigényének összege lesz (plusz az inputból jövő tag, ha írunk az inputba is)

1. feladat.

Elemezzük az alábbi kód tárigényét!

```
int isPrime(int A) {
  for (int B := 2; B < A; B++) {
    if (A % B == 0) return 0
  }
  return 1
}
```

2. feladat.

Elemezzük az alábbi kód tárigényét!

```
int tombMax(int[] T) {
  if (T.length == 0) return 0
  int max = T[0]
  for (int i := 1; i < T.length; i++ ) {
    if (max < T[i]) max := T[i];
  }
  return max;
}
```

3. feladat.

Elemezzük az alábbi kód tárigényét!

```
int tombMax(int[] T) {
  if (T.length == 0) return 0
  int maxIndex = 0
  for (int i := 1; i < T.length; i++ ) {
    if (T[maxIndex] < T[i]) maxIndex := i;
  }
  return T[maxIndex];
}
```

Recap: függvényhívás

Ha az elemzendő kódunkban szerepel függvényhívás, akkor a tárigénynél minden egyes függvényhívásos lépésben

- kiszámítjuk az **argumentumok** lemásolásához szükséges tárat
- plusz, magának a hívott függvénynek a tárigényét, ezt a kettőt összeadjuk

és (mivel függvényhívás után a hívott függvény „visszaadja a stacket”) ha **több** függvényhívás is szerepel egy függvénytörzsben, akkor a szükséges memória ezeknek a függvényhívásoknak számított memória **maximuma**, **plusz** a függvény lokális változóinak tárigénye lesz

4. feladat.

Elemezzük az alábbi kód `main` függvényének tárigényét!

```
1  int f(int A) {
2    int M := 0
3    for (int i := 0; i < A; i++) {
4      M := 1 - M
5    }
6    return M
7  }
8
9  int main(int[] T) {
10   int count := 0
11   for (int i := 0; i < T.length; i++) {
12     int A := f(T[i])
13     count := count + A
14   }
15   return count
16 }
```

Recap: rekurzió

Rekurzív függvények esetében ha a függvényünkben csak egyetlen rekurzív hívás van, akkor a tárigénye a (lokális változók tárigénye + a rekurzív hívásban az argumentum leírásához szükséges bitek száma), **szorozva a rekurzió maximális mélységével**.

5. feladat.

Elemezzük az alábbi kód tárigényét!

```
1  int sum(int A) {
2    if (A <= 0) return 0
3    int B := sum(A - 1)
4    return A + B
5  }
```

6. feladat.

Az alábbi kód inputja egy **gráf**, szomszédsági mátrixszal reprezentálva: `G[i][j]=1`, ha van i-ből j-be él, `0`, ha nincs, `N` pedig a csúcsok száma (tehát `G` egy `N x N`-es mátrix). Elemezzük a tárigényét!

```
1 int haromszog(int N, int[][] G) {
2   for (int i := 0; i < N; i++)
3     for (int j := 0; j < N; j++)
4       for (int k := 0; k < N; k++)
5         if (G[i][j] && G[j][k] && G[i][k] ) return 1;
6   return 0;
7 }
```

---

1. feladat megoldása.

Az A input változó értékét csak olvassuk, így az innen jövő n bitet nem kell beleszámolni a tárigénybe.

A kódban szerepel még egy B lokális változó, melynek értéke legfeljebb A lesz, tehát neki a tárigénye $O(\log A)$, ami $O(n)$ (hiszen mint korábban is, az input mérete $n = O(\log A)$.

Mivel a függvény kódjában nem szerepel további függvényhívás, és az inputot csak olvassa, így a tárigénye a lokális változóinak az összesített tárigényével egyezik meg, ami tehát $O(n)$.

---

2. feladat megoldása.

A függvény az inputban érkező `T` tömböt csak olvassa, ezért annak az n bitjét nem kell beleszámoljuk a tárigénybe.

A kódban nincs másik függvényre hívás, ezért a tárigény a lokális változók össz tárigénye lesz.

Két lokális változónk van: `i` és `max`. Kettejük közül `i` **értéke** legfeljebb `T.length`, amiről előző gyakorlaton láttuk, hogy $O(n)$. Tehát ennek a változónak a tárigénye a benne tárolt legnagyobb érték **logaritmusa** lesz: $O(\log n)$.

A `max` változó **értéke** legfeljebb annyi lesz, mint az input tömb legnagyobb elemének az értéke, erről előző gyakorlaton láttuk, hogy $O(2^n)$. Ennek a változónak a tárigénye tehát $O(\log 2^n) = O(n)$.

Az egész függvény tárigénye tehát $O(\log n) + O(n) = O(n)$, azaz **lineáris**.

---

3. feladat megoldása.

Ebben a kódban az input szintén read-only, nincs benne függvényhívás, így csak a `maxIndex` és az `i` lokális változók tárigényét kell összeadjuk; `i` továbbra is 0 és `T.length` között vesz fel értéket, tehát neki a tárigénye $O(\log(\text{T.length})) = O(\log n)$, de ezúttal `maxIndex`ben nem egy tömb**elemet**, hanem egy tömb**indexet** tárolunk, így ő is 0 és `T.length` közti értéket vesz fel, azaz neki is $O(\log n)$ a tárigénye.

A teljes kód tárigénye tehát $O(\log n) + O(\log n) = O(\log n)$, **logaritmikus**.

4. feladat megoldása.

Nézzük előbb a `main` függvényt. A `T` tömböt csak olvassa (a 12. sorban az `f(T[i])` hívásnál ugyanúgy, mint a legtöbb progamozási nyelven, a `T[i]` argumentumot **érték szerint** adjuk át, azaz **lemásoljuk** a call stackre, és úgy hívjuk meg az `f` függvényt, ezért még ha az `f` függvény változtatná is az input argumentumának az értékét, az már az eredeti `T` tömbre nem hat vissza!)

Foglalkozzunk először a lokális változókkal: `count`, `i` és `A`. Az biztos, hogy közülük az `i` értéke 0 és `T.length` közt veszi fel az értékét, `T.length` továbbra is $O(n)$, tehát az `i` változó tárigénye $O(\log n)$.

Ahhoz viszont, hogy `A` és `count` tárigényét megbecsüljük, tudnunk kell, hogy mekkora érték kerülhet bele legfeljebb, itt `A` esetében ehhez meg kell vizsgáljuk az `f` függvényt, hogy lássuk, mit adhat vissza. (Egyelőre az `f` függvény tárigényével ehhez még nem kell foglalkoznunk, csak azzal, hogy milyen intervallumba eshet az ő outputja.)

Azt látjuk `f`-ben, hogy az `M` változót fogja visszaadni, ami 0-ról indul és a 4. sorban kaphat új értéket. Ez a sor 0-ból 1-et, 1-ből 0-t fog készíteni, tehát `f` mindenképp vagy 0-t, vagy 1-et ad vissza.

Így az `A` változóba a `main` függvény 12. sorában vagy 0, vagy 1 kerül, máshol nem kap értéket, azaz az ő tárolására elég $O(1)$ bit.

A `count` változó pedig 0-ról indul és egy legfeljebb $O(n)$-szer lefutó ciklusban mindig hozzáadunk (ezek szerint) vagy 0-t, vagy 1-et, így az értéke $O(n)$ lesz, tehát az ő tárigénye $O(\log n)$.

Eddig ott tartunk, hogy a `main` függvény **lokális** változóinak tárigénye $O(\log n) + O(1) + O(\log n) = O(\log n)$, de ezzel így még nem vagyunk kész, mert a függvényben a 12. sorban van egy **függvényhívás**, amivel foglalkoznunk kell!

A függvényhívások tárigénye (mint a legtöbb imperatív programozási nyelvben) a következőből áll össze:

- a hívott függvény argumentumait (most `T[i]`-t) lemásoljuk a call stackbe
- eztán meghívva a függvényt (most `f`-et) ő is lefoglalja a memóriában **pluszban** a számára kellő lokális memóriát (most: helyet foglal az `M` és az `i` változóinak)
- mikor pedig visszatér az eredménnyel, felszabadítja az argumentumként kapott és a saját lokális területét is, helyükre az eredmény kerül
- visszatérés után egy további függvényhívás pedig az így felszabadított memóriaterületen fog operálni (ha odafér).

Ebben a példában az `f` függvény ha kap egy `A` értéket, akkor az `M` lokális változójának már láttuk, hogy $O(1)$ a tárigénye, az `i` lokális változójának pedig $O(\log A)$ (mert benne 0 és A közti érték lesz), benne már nincs további függvényhívás, így `f` tárigénye $O(\log A)$, ha A-t kapja inputként.

Vagyis, a 12. sorban lefutó kódnak a tárigénye:

- függvényhívás miatt a `T[i]` argumentum beleírása a call stackbe ad egy $O(n)$-es tárigényt (előző gyakon láttuk, hogy a `T[i]`-ben levő érték $O(2^n)$, ennek tárigénye $O(\log 2^n) = O(n)$

- magának `f`-nek a futásához pedig $O(n)$ további tárra van szükség (mert tárigénye $O(\log A)$, ha az A számot kapja inputként, híváskor $A = T[i]$ ami $O(2^n)$, ezért az $O(\log A) = O(\log 2^n) = O(n)$ lesz

így tehát az `f(T[i])` kiértékeléséhez szükséges tár $O(n)+O(n) = O(n)$ lesz, ezt kell hozzáadjuk a `main` függvény lokális változóinak tárigényéhez és kapjuk, hogy a `main` kód teljes tárigénye $O(\log n) + O(n) = O(n)$.

---

5. feladat megoldása.

A `sum` függvényben szerepel egy lokális `B` változó. Hogy ennek a tárigényét megbecsüljük, szükségünk van magára a `sum` függvényre, hogy mekkora értéket adhat vissza, ha az inputja `A-1`. Szerencsére ez ebben az esetben nem nehéz, mert látjuk, hogy az implementáció az első `A` pozitív egész szám összegét adja vissza, ami $O(A^2)$, ekkora érték kerülhet a `B` lokális változóba, így annak tárigénye $O(\log A^2) = O(\log A)$ (mert $\log A^2 = 2 \log A$ és $O(2 \log A) = O(\log A)$).

Hozzá kell még adnunk a `sum(A - 1)` függvényhívásból jövő memóriaigényt, az argumentumnak, `A - 1`-nek az értéke persze $O(A)$, így az argumentum lemásolása további $O(\log A)$ tárat igényel.

Még hozzá kell mindehhez adnunk a `sum(A - 1)` **rekurzív** függvényhívás tárigényét, azaz a `sum(A)` hívás tárigényéhez szükségünk van a `sum(A - 1)` hívás tárigényére és azt kapjuk az eddigiekből, hogy

`sum(A)` tárigénye $O(\log A)$ (a lokális változókra és az argumentum másolásra ezt számoltuk eddig) **plusz** `sum(A - 1)` tárigénye; ha `A <= 0`, akkor pedig $O(1)$ (olyankor nincs rekurzív hívás, hanem egyből visszaadunk egy 0-t).

Ha ezt kifejtjük, akkor `sum(A - 1)` tárigénye $O(\log(A-1))$ plusz `sum(A - 2)` tárigénye lesz, ha azt is tovább fejtjük egész amíg az argumentum le nem szalad 0-ig, azt kapjuk, hogy `sum(A)` összes tárigénye $O(\log A)+O(\log(A-1))+O(\log(A-2))+\ldots+O(\log 1)+O(1)$ lesz. Itt látunk $O(A)$ tagot az összeadásban, mindegyikre jó az $O(\log A)$ felső korlát, tehát a teljes tárigény $O(A \cdot \log A)$ lesz.

Már csak annyi dolgunk van, hogy ezt átírjuk az input n méretének függvényébe: az $n = O(\log A)$-ból $A = O(2^n)$-t kapjuk, amit ha visszaírunk a fentibe, megkapjuk az $O(n \cdot 2^n)$ tárigényt.

6. feladat megoldása.

A kódban nincs rekurzív hívás, így csak a három lokális változó, `i,j,k` tárigényét kell összeadjuk.

Mindhárom változó 0 és N közti értéket vehet fel, így az összes tárigényük $O(\log N)+O(\log N)+O(\log N) = O(\log N)$.

Ezt kell még kifejeznünk az input leírásához szükséges bitek n számában; a mátrix minden eleme 1-bites, van belőlük N^2 darab, ez összesen eddig $O(N^2)$ bit, plusz még a bejövő N szám ábrázolásához kellő $O(\log N)$ bit, tehát $n = O(N^2) + O(\log N) = O(N^2)$. Ebből azt kapjuk, hogy $N = O(\sqrt{n})$ és így az $O(\log N)$ tárigény az n függvényében $O(\log(\sqrt{n})) = \mathbf{O}(\log \mathbf{n})$ (mert $\log \sqrt{n} = \log n^{1/2} = \frac{1}{2} \log n$).