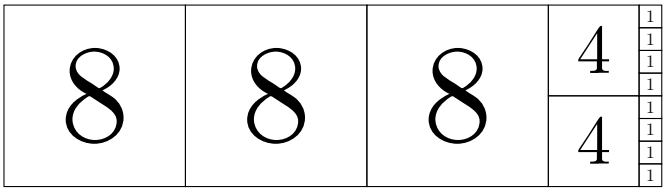
Rešitev tretje domače naloge (Ploščičar)

Iterativna rešitev

Pri tej nalogi bomo na veliko delali s potencami števila 2, zato najprej omenimo, da lahko 2^n v javi jedrnato in učinkovito izračunamo z izrazom `1 << n`. (Operator `<<` je bitni pomik v levo, zato je rezultat izraza `1 << n` dvojiško število, sestavljeno iz enice in n ničel. V desetiškem sistemu je to število enako 2^n.)

Za začetek odmislimo višino in se ukvarjajmo samo s polaganjem ploščic vzdolž širine. Osredotočimo se na tretji primer v besedilu naloge. Za zapolnitev pasu širine 29 potrebujemo tri ploščice s stranico 2^3, dve ploščici s stranico 2^2, nič ploščic s stranico 2^1 in osem ploščic s stranico 2^0, torej skupaj 13 ploščic:



Število ploščic, ki jih potrebujemo za zapolnitev pasu širine w, pri čemer ima največja ploščica stranico 2^p, lahko v splošnem izračunamo po sledečem postopku:

- Število ploščic s stranico 2^p dobimo preprosto z izrazom $n_p = \lfloor w \,/\, 2^p \rfloor$.
- Preostanek širine ($w' = w \bmod 2^p$) poskusimo napolniti s ploščicami s stranico 2^{p-1}. Njihovo število izračunamo kot $n_{p-1} = 2\,\lfloor w' \,/\, 2^{p-1} \rfloor$. Faktor 2 potrebujemo zato, ker so ploščice s stranico 2^{p-1} za polovico nižje od tistih s stranico 2^p.
- Preostanek širine ($w'' = w' \bmod 2^{p-1}$) poskusimo napolniti s ploščicami s stranico 2^{p-2}. Njihovo število izračunamo kot $n_{p-2} = 4\,\lfloor w'' \,/\, 2^{p-2} \rfloor$.
- Postopek ponavljamo, dokler ne zapolnimo celotne širine.

Opisani postopek bomo realizirali s pomočjo metode

`private static long napolniSirino(int sirina, int ploscica)`,

ki vrne število ploščic, potrebnih za zapolnitev pasu širine `sirina`, pri čemer je stranica največje ploščice dolga kvečjemu `1 << ploscica`.

```
private static long napolniSirino(int sirina, int ploscica) {
    int faktor = 1;
    long stPloscic = 0;
    while (sirina > 0) {
        stPloscic += faktor * (sirina / (1 << ploscica));
        sirina %= 1 << ploscica;
        ploscica--;
        faktor *= 2;
    }
    return stPloscic;
}
```

Pravokotnik velikosti $h \times w$ pa zapolnimo s sledečim postopkom:

- Izračunamo stranico največje možne ploščice. Recimo, da znaša 2^r.
- S pomočjo ploščic s stranicami 2^r, 2^{r-1}, ... zapolnimo pas širine w. Recimo, da v ta namen potrebujemo s ploščic.
- Izračunamo, koliko takih pasov lahko zložimo v pravokotnik ($\lfloor h/2^r \rfloor$) in koliko ploščic za to porabimo ($s \lfloor h \,/\, 2^r \rfloor$).
- Izračunamo, koliko višine nam še preostane ($h \bmod 2^r$).
- Postopek ponavljamo, dokler ne zapolnimo celotne višine.

Postopek realiziramo z metodo

`private static long napolni(int visina, int sirina, int ploscica)`,

ki vrne število ploščic, potrebnih za zapolnitev pravokotnika s podano višino in širino, pri čemer je največja možna stranica ploščice enaka `1 << ploscica`:

```
private static long napolni(int visina, int sirina, int ploscica) {
    long stPloscic = 0;
    while (visina > 0) {
        ploscica = najvecjaMoznaStranica(ploscica, Math.min(visina, sirina));
        long stPloscicVPasu = napolniSirino(sirina, ploscica);
        stPloscic += stPloscicVPasu * (visina / (1 << ploscica));
        visina %= 1 << ploscica;
    }
    return stPloscic;
}
```

Metoda

`private static int najvecjaMoznaStranica(int zacVelikost, int prostor)`

ugotovi, kako veliko ploščico lahko postavimo v kvadrat s stranico dolžine `prostor`, pri čemer stranica ploščice ne sme biti daljša od `1 << zacVelikost`. Metoda vrne dvojiški logaritem iskane dolžine stranice:

```
private static int najvecjaMoznaStranica(int zacVelikost, int prostor) {
    int velikost = zacVelikost;
    while ((1 << velikost) > prostor) {
        velikost--;
    }
    return velikost;
}
```

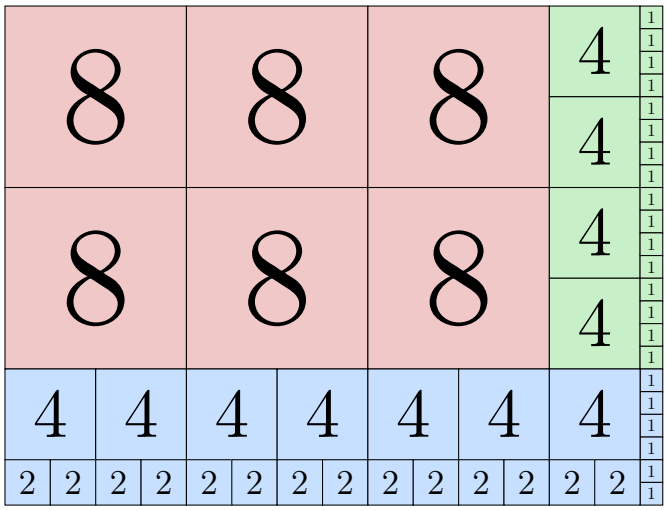
Manjka nam le še metoda `main`:

```
import java.util.Scanner;

public class Ploscice {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int visina = sc.nextInt();
        int sirina = sc.nextInt();
```

```
        int ploscica = sc.nextInt();
        System.out.println(napolni(visina, sirina, ploscica));
    }

    private static long napolni(int visina, int sirina, int ploscica) { ... }
    private static int najvecjaMoznaStranica(int zacVelikost, int prostor) { ... }
    private static long napolniSirino(int sirina, int ploscica) { ... }
}
```

Rekurzivna rešitev

Rekurzija nas privede do presenetljivo elegantne rešitve. Naša rešitev zapolni steno tako, da najprej zapolni rdeč, nato zelen in nazadnje moder pravokotnik:



Rdeč pravokotnik je sestavljen zgolj iz največjih možnih ploščic, zelenega in modrega pa lahko zapolnimo na enak način kot celotno steno (torej lahko tudi vsakega od njiju razdelimo na tri dele, in to na enak način kot celotno steno).

Pri rdečem pravokotniku moramo izračunati stranico največje ploščice (označimo jo s p) glede na začetno velikost in razpoložljivi prostor. Če sta h in w višina celotne stene, potem je število ploščic v rdečem pravokotniku enako $\lfloor h \,/\, p \rfloor \lfloor w \,/\, p \rfloor$. Število ploščic v zelenem pravokotniku (njegova višina znaša $h - (h \bmod p)$, širina pa $(w \bmod p)$) in modrem pravokotniku (njegova višina znaša $(h \bmod p)$, širina pa w) pa izračunamo na enak način kot število ploščic v celotni steni.

```
import java.util.Scanner;

public class Ploscice {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int visina = sc.nextInt();
        int sirina = sc.nextInt();
        int ploscica = sc.nextInt();
        System.out.println(napolni(visina, sirina, ploscica));
    }
```

```
    private static long napolni(int visina, int sirina, int ploscica) {
        if (visina == 0 || sirina == 0) {   // robni pogoj
            return 0;
        }
        ploscica = najvecjaMoznaStranica(ploscica, Math.min(visina, sirina));
        int p = 1 << ploscica;

        return ((long) (visina / p)) * ((long) (sirina / p)) +
                napolni(visina - visina % p, sirina % p, ploscica) +
                napolni(visina % p, sirina, ploscica);
    }

    private static int najvecjaMoznaStranica(int zacVelikost, int prostor) {
        int velikost = zacVelikost;
        while ((1 << velikost) > prostor) {
            velikost--;
        }
        return velikost;
    }
}
```