Zadanie 4 – praktyczne

In[1]:=

```
(*1. Generatory obrotów sześcianu o 90° wokół osi Z,Y i X*)
(*Obrót o 90° wokół osi Z:(x,y,z)→(-y,x,z)*)
obrótZ = Cycles[{{1, 2, 3, 4}, {5, 6, 7, 8}}];
         cykle

(*Obrót o 90° wokół osi Y:(x,y,z)→(z,y,-x)*)
obrótY = Cycles[{{1, 5, 6, 2}, {4, 8, 7, 3}}];
         cykle

(*Obrót o 90° wokół osi X:(x,y,z)→(x,-z,y)*)
obrótX = Cycles[{{1, 4, 8, 5}, {2, 3, 7, 6}}];
         cykle

(*Tworzenie grupy generowanej przez obroty*)
grupa = PermutationGroup[{obrótZ, obrótY, obrótX}];
        grupa permutacji
elementyGrupy = GroupElements[grupa];
                elementy grupy

(*2. Wszystkie kolorowania 8 wierzchołków dwoma barwami*)
wszystkieKolorowania = Tuples[{0, 1}, 8];
                       krotki

(*Słowniki do śledzenia odwiedzonych kolorowań i reprezentantów orbity*)
odwiedzoneWszystkie = <||>;
reprezentanci = {};

(*3. BFS wyznaczający reprezentantów orbity dla każdego kolorowania*)
Do[kolorowanie = wszystkieKolorowania[[i]];
rób
  If[! KeyExistsQ[odwiedzoneWszystkie, kolorowanie], orbita = <||>;
  op··· istnieje klucz?
   kolejka = {kolorowanie};
   While[kolejka =!= {}, obecne = First[kolejka];
   podczas                      pierwszy
    kolejka = Rest[kolejka];
              bez pierwszego elementu
    If[! KeyExistsQ[orbita, obecne], AssociateTo[orbita, obecne → True];
    op··· istnieje klucz?            dodaj to stowarzyszenia      prawda
     Do[nowe = Permute[obecne, InversePermutation[elementyGrupy[[j]]]];
     rób       permutuj        permutacja odwrotna
      If[! KeyExistsQ[orbita, nowe], AppendTo[kolejka, nowe]];,
      op··· istnieje klucz?          dołącz na końcu do wartości zmiennej
      {j, Length[elementyGrupy]}];];];
          długość
   (*Dodajemy znalezione reprezentanty i oznaczamy odwiedzone*)
   AppendTo[reprezentanci, kolorowanie];
   dołącz na końcu do wartości zmiennej
```

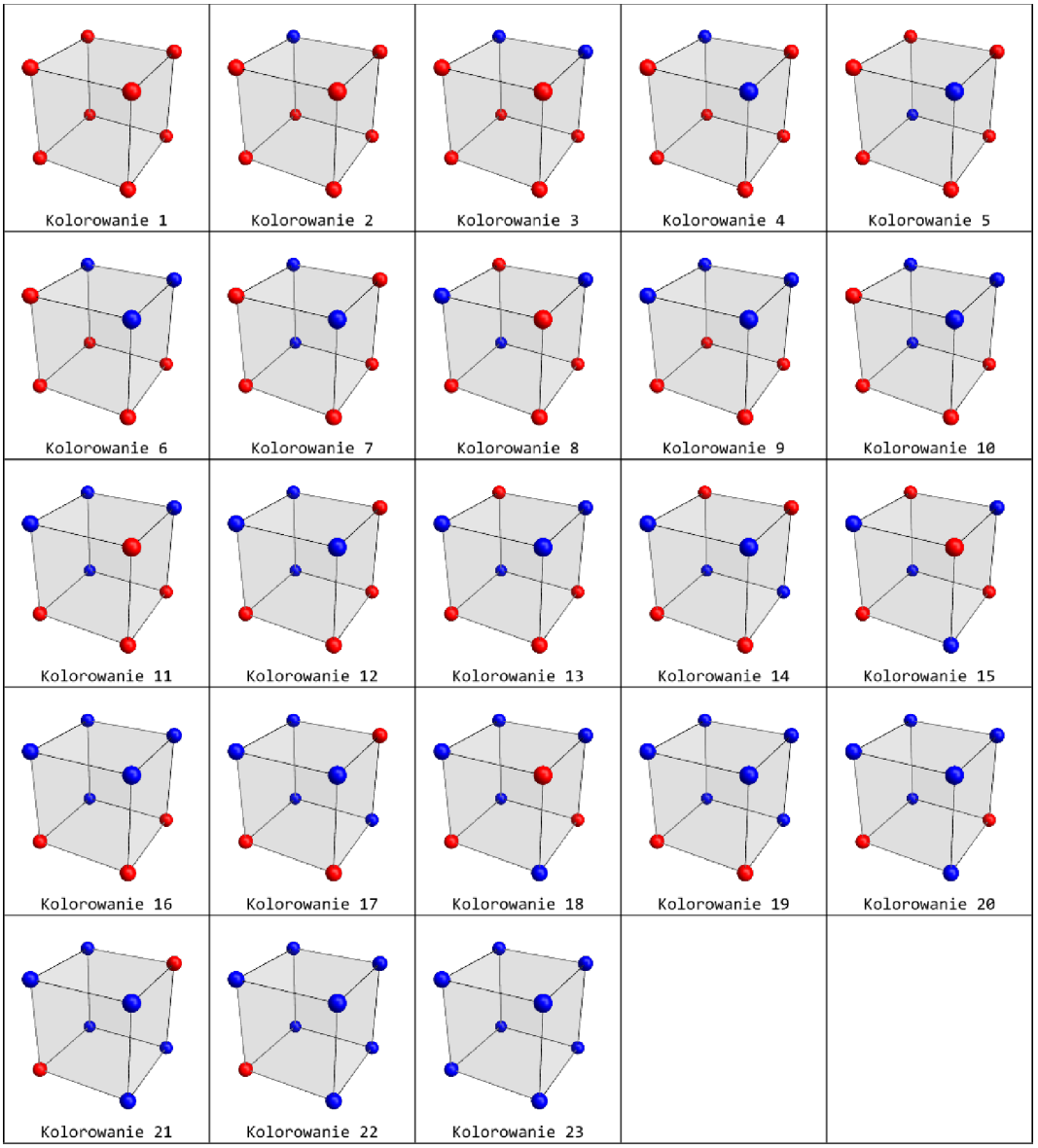
```
    Do[AssociateTo[odwiedzoneWszystkie, c → True], {c, Keys[orbita]}];];,
    {i, Length[wszystkieKolorowania]}];

(*4. Wyświetlenie liczby reprezentantów i samych kolorowań*)
Print["Liczba nieekwiwalentnych kolorowań: ", Length[reprezentanci]];
Print["Reprezentanci orbity (posortowani według liczby jedynek):"];
posortowaniReprezentanci = SortBy[reprezentanci, Total];
Do[Print[posortowaniReprezentanci[[k]]], {k, Length[posortowaniReprezentanci]}];

(*5. Funkcja rysująca sześcian z pokolorowanymi wierzchołkami*)
RysujSzescian[kolorowanie_] := Module[
    {wierzcholki3D, sciany, krawedzieIndeksy, scianyPolygon, krawedzieLines, kolory},
    (*Współrzędne 3D ośmiu wierzchołków*)wierzcholki3D = {{-1, -1, -1}, {1, -1, -1},
      {1, 1, -1}, {-1, 1, -1}, {-1, -1, 1}, {1, -1, 1}, {1, 1, 1}, {-1, 1, 1}};
    (*Ściany sześcianu jako czwórki indeksów wierzchołków*)
    sciany = {{1, 2, 3, 4}, (*dolna*){5, 6, 7, 8}, (*górna*){1, 2, 6, 5}, (*przód*)
      {2, 3, 7, 6}, (*prawa*){3, 4, 8, 7}, (*tył*){4, 1, 5, 8}  (*lewa*)};
    (*Krawędzie jako pary indeksów*)krawedzieIndeksy = {{1, 2}, {2, 3}, {3, 4},
      {4, 1}, {5, 6}, {6, 7}, {7, 8}, {8, 5}, {1, 5}, {2, 6}, {3, 7}, {4, 8}};
    scianyPolygon = Polygon /@ (wierzcholki3D[[#]] & /@ sciany);
    krawedzieLines = Line /@ (wierzcholki3D[[#]] & /@ krawedzieIndeksy);
    (*Kolory:0→Red,1→Blue*)kolory = kolorowanie /. {0 → Red, 1 → Blue};
    Graphics3D[{{Opacity[0.1], Gray, scianyPolygon},
      {Gray, Thickness[0.005], krawedzieLines}, MapThread[
       {#2, Specularity[White, 50], Sphere[#1, 0.15]} &, {wierzcholki3D, kolory}]},
     Boxed → False, Lighting → "Neutral", ViewPoint → {1.3, -2.4, 1.5},
     ImageSize → 150]];

(*6. Generowanie i wyświetlanie wizualizacji reprezentantów*)
wizualizacje = Table[Labeled[RysujSzescian[posortowaniReprezentanci[[i]]],
    Row[{"Kolorowanie ", i}], Bottom], {i, Length[posortowaniReprezentanci]}];
siatka = Partition[wizualizacje, UpTo[5]];
```

podział na rozdzielone bloki bez do

```
Print[Grid[siatka, Frame → All, FrameStyle → LightGray]];
```

drukuj krata ramka ws··· styl ramki jasnoszary

```
Liczba nieekwiwalentnych kolorowań: 23

Reprezentanci orbity (posortowani według liczby jedynek):

{0, 0, 0, 0, 0, 0, 0, 0}

{0, 0, 0, 0, 0, 0, 0, 1}

{0, 0, 0, 0, 0, 0, 1, 1}

{0, 0, 0, 0, 0, 1, 0, 1}

{0, 0, 0, 1, 0, 1, 0, 0}

{0, 0, 0, 0, 0, 1, 1, 1}

{0, 0, 0, 1, 0, 1, 0, 1}

{0, 0, 0, 1, 1, 0, 1, 0}

{0, 0, 0, 0, 1, 1, 1, 1}

{0, 0, 0, 1, 0, 1, 1, 1}

{0, 0, 0, 1, 1, 0, 1, 1}

{0, 0, 0, 1, 1, 1, 0, 1}

{0, 0, 0, 1, 1, 1, 1, 0}

{0, 0, 1, 1, 1, 1, 0, 0}

{0, 1, 0, 1, 1, 0, 1, 0}

{0, 0, 0, 1, 1, 1, 1, 1}

{0, 0, 1, 1, 1, 1, 0, 1}

{0, 1, 0, 1, 1, 0, 1, 1}

{0, 0, 1, 1, 1, 1, 1, 1}

{0, 1, 0, 1, 1, 1, 1, 1}

{0, 1, 1, 1, 1, 1, 0, 1}

{0, 1, 1, 1, 1, 1, 1, 1}

{1, 1, 1, 1, 1, 1, 1, 1}
```



In[18]:= **ClearAll**

wyczyść wszystko

Out[18]=

ClearAll

Zadanie 5 - praktyczne

In[19]:=
```
(*1. Wszystkie funkcje logiczne z 3 wejściami*)
allFuncs = Tuples[{0, 1}, 8]; (*2^8=256 możliwych funkcji*)
           krotki

(*2. Permutacje zmiennych wejściowych*)
permutations = Permutations[{1, 2, 3}];
               lista permutacji

(*3. Lista wszystkich wejść:kombinacje binarne trzech zmiennych*)
inputs = Tuples[{0, 1}, 3];
         krotki

(*4. Zdefiniuj funkcję,która permutuje wejścia i zmienia funkcję logiczną*)
permuteFunc[func_, perm_] :=
  Module[{reorderedInputs, indices}, reorderedInputs = inputs[[All, perm]];
  moduł                                                wszystko
    (*permutujemy zmienne*)indices = FromDigits[#, 2] + 1 & /@ reorderedInputs;
                                     liczba zadana przez ciąg cyfr
   func[[#]] & /@ indices];

(*5. Zbiór unikalnych klas równoważności względem permutacji*)
equivalenceClasses =
  Union[Table[Sort[permuteFunc[f, #] & /@ permutations], {f, allFuncs}]];
  suma··· tabela  sortuj

(*6. Wynik końcowy:liczba klas równoważności*)
Length[equivalenceClasses]
długość
```

Out[24]=
```
80
```