```
(*f - given function | x1,x2,y1,y2 - restrictons of the given function |
 rad - ''radius'' of searched area per point*)
(*sizePath - amount of steps for the algorithm to be taken |
 sizeAround - amount of randomly chosen points in the searched area*)
tabuSearch[f_, x1_, x2_, y1_, y2_, rad_, sizePath_, sizeAround_] :=
 Module[{ xa = x1, xb = x2, ya = y1, yb = y2, r = rad, path, around, tabu,
 moduł
   activeMinimum, tmpMinimum, inTabu, tmp, limiter, ultimateMinimum},
  (*array that saves the points chosen by algorithm*)
  path = CreateDataStructure["DynamicArray"];
         stwórz strukturę danych
  (*array that takes the random
   points around the chosen point restricted by value r*)
  around = Table[0, {i, 1, sizeAround}];
           tabela
  (*replaces the r in case the alghoritm would like to escape function limits*)
  limiter = Table[0, {i, 1, 4}];
            tabela
  (*array that saves the tabu points*)
  tabu = CreateDataStructure["DynamicArray"];
         stwórz strukturę danych
  (*plot of the given function f*)
  plott = ContourPlot[f, {x, x1, x2}, {y, y1, y2}];
          wykres konturowy
  (*first point is chosen randomly*)
  around[[1]] = {Random[Real, {xa, xb}], Random[Real, {ya, yb}]};
                 liczba rzeczywista              liczba rzeczywista
  path["Append", around[[1]]];
       dołącz na końcu
  tmpMinimum = around[[1]];
  activeMinimum = tmpMinimum;
  ultimateMinimum = tmpMinimum;

  (*loop that determines how long should the algorithm compute*)
  For[k = 2, k ≤ sizePath, k++,
  dla
   (*picking random points with respect to the limiter*)
   For[i = 2, i ≤ sizeAround, i++,
   dla
    If[around[[1, 1]] - r < xa, limiter[[1]] = Abs[xa - around[[1, 1]]], limiter[[1]] = r];
    operator warunkowy                         wartość bezwzględna
    If[around[[1, 1]] + r > xb, limiter[[2]] = Abs[xb - around[[1, 1]]], limiter[[2]] = r];
    operator warunkowy                         wartość bezwzględna
    If[around[[1, 2]] - r < ya, limiter[[3]] = Abs[ya - around[[1, 2]]], limiter[[3]] = r];
    operator warunkowy                         wartość bezwzględna
    If[around[[1, 2]] + r > yb, limiter[[4]] = Abs[yb - around[[1, 2]]], limiter[[4]] = r];
    operator warunkowy                         wartość bezwzględna

    around[[i]] = {Random[Real, {around[[1, 1]] - limiter[[1]], around[[1, 1]] + limiter[[2]]}],
                   liczba rzeczywista
      Random[Real, {around[[1, 2]] - limiter[[3]], around[[1, 2]] + limiter[[4]]}]};];
      liczba rzeczywista
```

```
          liczba rzeczywista
(*end of picking random points with respect to the limiter*)

(*loop that validates the random points*)
For[i = 2, i ≤ sizeAround, i++,
dla
 (*checking if a point is in the tabu list*)
 inTabu = False;
          fałsz
 For[j = 1, j < tabu["Length"], j++,
 dla                  długość
  tmp = tabu["Part", j];
             część
  If[ around⟦i, 1⟧ < tmp⟦1⟧ + r && around⟦i, 1⟧ > tmp⟦1⟧ - r &&
  operator warunkowy
    around⟦i, 2⟧ < tmp⟦2⟧ + r && around⟦i, 2⟧ > tmp⟦2⟧ - r , inTabu = True];];
                                                          prawda


 (*if not, we check if the point takes a lower value than the previous one*)
 If[inTabu == False && (f /. {x → tmpMinimum⟦1⟧, y → tmpMinimum⟦2⟧}) >
 operator war⋯ fałsz
    (f /. {x → around⟦i, 1⟧, y → around⟦i, 2⟧}), tmpMinimum = around⟦i⟧;];

 (*if the point has the lowest
  value in this run it saves it to an activeMinimum*)
 If[inTabu == False && (f /. {x → activeMinimum⟦1⟧, y → activeMinimum⟦2⟧}) >
 operator war⋯ fałsz
    (f /. {x → around⟦i, 1⟧, y → around⟦i, 2⟧}), activeMinimum = around⟦i⟧; ];
];
(*end of the loop that validates the random points*)

(*appending chosen point to a tabu list*)
tabu["Append", tmpMinimum];
     dołącz na końcu

(*Diversification - if the algorithm cannot find a lower value
  in given area it chooses new starting point somewhere on the graph,
if it can it goes on. Also saves the lowest point of all runs*)
If[tmpMinimum == around⟦1⟧, path["Append",
operator warunkowy                dołącz na końcu
  {Random[Real, {xa, xb}], Random[Real, {ya, yb}]}];
          liczba rzeczywista     liczba rzeczywista
 If[(f /. {x → ultimateMinimum⟦1⟧, y → ultimateMinimum⟦2⟧}) > (f /.
 operator warunkowy
     {x → activeMinimum⟦1⟧, y → activeMinimum⟦2⟧}), ultimateMinimum = activeMinimum;
  activeMinimum = path["Part", k];], path["Append", tmpMinimum]];
                       część               dołącz na końcu

(*restarts auxiliary values*)
tmpMinimum = path["Part", k];
                  część
```

```
     around⟦1⟧ = path["Part", k];
                      └część

   ];
   (*end of the loop that determines how long should the algorithm compute*)

   (*assigning algorithm output to present it on plot*)
   p = Table[0, {i, 1, sizePath}];
      └tabela
   For[i = 1, i ≤ sizePath, i++,
   └dla
    p [[i]] = path["Part", i];
                   └część
   ];
   Print["the approximate minimum point of given function is", ultimateMinimum,
   └drukuj
    " and equals ", f /. {x → ultimateMinimum⟦1⟧, y → ultimateMinimum⟦2⟧}];
   points = Table[Show[plott, ListLinePlot[Take[p, aa], Mesh → All,
            └tabela └pokaż      └wykres liniowy li⋯ └weź      └siatka └wszystko
       PlotStyle → {PointSize[0.02], Red}]], {aa, 1, sizePath}];
       └styl grafiki └rozmiar kropki  └czerwony
   ListAnimate[points]
   └animuj liste
  ]

Clear[x];
└wyczyść
g1 = x^2 + y^2;
tabuSearch[g1, -5, 5, -5, 5, 0.5, 30, 500]

g3 = Sin[x]^3 + Cos[y]^2;
tabuSearch[g3, -10, 10, -10, 10, 1, 40, 500]

g4 = Sin[x]^4 + Cos[y]^3 + x^2 ;
tabuSearch[g4, -10, 10, -10, 10, 1, 40, 500]
```