**Geneticky algoritmus** je optimalizačný algoritmus inšpirovaný prírodou. Na začiatku sa vygeneruje niekoľko **jedincov (chromozómov)**, ktorí každý reprezentuje riešenie daného problému. Jedinci sú reprezentovaní napr. reťazcom znakov. Generovanie na začiatku môže byť náhodné, alebo s použitím heuristiky, ktorá vygeneruje čo najlepších jedincov, ak je taká heuristika dostupná.

K danému problému musíme mať tzv. **fitness funkciu**, ktorá každého jedinca ohodnocuje. Množina všetkých jedincov sa nazýva **populácia**. Kvalita populácie je daná priemernou populáciou jedincov a postupom času by sa mala zvyšovať.

Z populácie sa vyberú najlepší jedinci (buď tzv. **elitizmom** – vyberú sa iba najlepší, alebo turnajovým výberom, ktorý umožňuje populácií prekonať lokálne optimá). Víťazní jedinci sa priradia do párov. Tzv. **krížením** vzniknú dva nové jedince tým, že sa zvolí náhodné (jedno alebo viac) miesto kríženia (chiazma), v ktorom sa podreťazce rodičov vymenia. Do ďalšej iterácie postupujú buď iba jedinci z novej generácie, alebo aj potomkovia aj rodičia.

Ojedinele sa potomok náhodným spôsobom mierne upraví – toto sa nazýva **mutácia**. Mutácie slúži na prekonanie lokálnych optím.

Algoritmus sa opakuje dovtedy, kým nenájde optimálne riešenie (v mnohých prípadoch sa optimálnosť riešenia nedá zistiť), alebo po dosiahnutí istej hodnoty fitness funkcie alebo po dosiahnutí istého počtu iterácií.

**Algoritmus optimalizácie mravčou kolóniou** je optimalizačný algoritmus inšpirovaný prírodou. Mravce v ozajstnej kolónií chodia náhodne okolo mraveniska a hľadajú potravu. Mravce majú tendenciu ísť smerom najväčšieho množstva **feromónu**. Keď mravec nájde potravu začne ju nosiť naspäť do mraveniska a začne pri tom vypúšťať veľké množstvo feromónu, v závislosti od veľkosti zdroja potravy. Týmto ku zdroju potravy naviguje aj ostatných mravcov.

Na podobnom princípe feromónov fungujú aj umelé mravce v ACO. Narozdiel od ozajstných mravcov majú zrak, pamäť a pohybujú sa synchrónne a v diskrétnom čase, feromóny nevypúšťajú stále, ale iba po ceste späť.

Konkrétnym problémom riešiteľným týmto algoritmom je **problém obchodného cestujúceho (TSP)**, kde musíme nájsť najkratšiu cestu, ktorá prechádza každým mestom a následne sa vráti do začiatočného mesta. Tento problém nie je bežnými metódami prehľadávania stavového priestoru riešiteľný v rozumnom čase (časová zložitosť je faktoriálna).

Na riešenie problému si vyberieme fixný počet mravcov, každého z nich vložíme do náhodného mesta. Na každú cestu medzi mestami dáme rovnaké a veľmi mále množstvo feromónu (aby sme nedelili nulou). Každý mravec si v aktuálnej iterácií pamätá, v ktorých mestách už bol, tým pádom do žiadneho nepríde dvakrát (okrem toho keď na konci príde do mesta z ktorého začal). Mravec musí v danej iterácií navštíviť všetky mestá, ignoruje priame cesty k mestám, kde už bol. Ak má na výber cestu do viacerých miest, v ktorých ešte nebol vyberá si podľa toho aká dlhá je tá cesta a ako veľa feromónov na nej je. Po tom ako každý mravec navštívi každé mesto tak ide rovnakou cestou späť a dáva na tú cestu feromóny. Tým, že každý mravec má rovnaké množstvo feromónu, tak sa na kratšie cesty dá viac feromónu.

V ďalšej iterácií zase dáme mravcov do iných náhodných miest a opakujeme. Algoritmus sa končí po dosiahnutí istej hodnoty dĺžky cesty alebo po fixnom počte iterácií.

% je cislo 0

zero(X) :- X =:= 0. % alebo zero(0).

% je cislo kladne

positive(X) :- X > 0.

% je cislo parne

even(X) :- 0 =:= (X mod 2).

% je cislo kladne aj parne

positiveEven(X) :- positive(X), even(X).

% je cislo kladne alebo nula (nezaporne)

nonNegative(X) :- zero(X), !.

nonNegative(X) :- positive(X).

% dlzka zoznamu

len([], 0).

len([\_|T], OuterLength) :- length(T, InnerLength), OuterLength is InnerLength + 1.

% patri do zoznamu

member(Item, [H|\_]) :- H = Item, !.

member(Item, [\_|T]) :- member(Item, T).

% hodnota na indexe zoznamu

valueOnIndex([H|\_], Index, Item) :- Index =:= 0, !, Item is H.

valueOnIndex([\_|T], Index, Item) :- NewIndex is Index -1, valueOnIndex(T, NewIndex, Item).

% index prveho konkretneho prvku

indexOf([H|\_], Item, Index) :- Item =:= H, Index is 0, !.

indexOf([\_|T], Item, OuterIndex) :- indexOf(T, Item, InnerIndex), OuterIndex is InnerIndex + 1.

% index najvacsieho prvku

argmax([Num], Num, 0).

argmax([H|T], Max, NewIndex) :- max(T, TailMax, \_), H > TailMax, Max is H, NewIndex is 0.

argmax([H|T], Max, NewIndex) :- max(T, TailMax, Index), H =< TailMax, Max is TailMax, NewIndex is Index + 1.

% mnozina zo zoznamu

list2set([], []).

list2set([LH|LT], S) :- member(LH, LT), !, list2set(LT, S).

list2set([LH|LT], [LH|S]) :- list2set(LT, S).

% scitanie poslednych dvoch cisel

sumLast([A, B], [C]) :- C is A + B.

sumLast([H|T], [H | NewTail]) :- sumLast(T, NewTail).

% zjednotenie dvoch mnozin

union(S1, [], S1).

union(S1, [H|T], R) :- member(H, S1), !, union(S1, T, R).

union(S1, [H|T], [H|R]) :- union(S1, T, R).

% prienik dvoch mnozin

intersect(\_, [], []).

intersect(S1, [H|T], [H|R]) :- member(H, S1), !, intersect(S1, T, R).

intersect(S1, [\_|T], R) :- intersect(S1, T, R).

% obsahuje zoznam iba atomicky hodnoty (ziadne podzoznamy)

atom([], 0).

atom([H|T], X) :- islist(H), !, atom(H, Y), atom(T, Z), X is Y + Z.

atom([\_|T], Y) :- atom(T, X), Y is X + 1.

islist([]).

islist([\_|\_]).

% median v usporiadanom zozname lichej dlzky

median(L, Value) :- len(L, Length), Index is Length // 2, getMedian(L, Index, Value).

getMedian([H|\_], 0, H).

getMedian([\_|T], OuterIndex, Value)

:- getMedian(T, InnerIndex, Value), OuterIndex is InnerIndex + 1, !.