الگوریتم ژنتیک

تکنیک جستجو در علم رایانه برای یافتن راه‌حل تقریبی برای بهینه‌سازی مدل ،ریاضی و مسائل جستجو است. الگوریتم ژنتیک نوع خاصی از الگوریتم‌های تکامل است که از تکنیک‌های زیست‌شناسی فرگشتی مانند وراثت، جهش زیست‌شناسی و اصول انتخابی داروین برای یافتن فرمول بهینه جهت پیش‌بینی یا تطبیق الگواستفاده می‌شود. الگوریتم‌های ژنتیک اغلب گزینه خوبی برای تکنیک‌های پیش‌بینی بر مبنای رگرسیون هستند.

در مدل‌سازی الگوریتم ژنتیک یک تکنیک برنامه‌نویسی است که از تکامل ژنتیکی به عنوان یک الگوی حل مسئله استفاده می‌کند. مسئله‌ای که باید حل شود دارای ورودی‌هایی می‌باشد که طی یک فرایند الگوبرداری شده از تکامل ژنتیکی به راه‌حلها تبدیل می‌شود سپس راه حلها به عنوان کاندیداها توسط تابع ارزیاب (Fitness Function) مورد ارزیابی قرار می‌گیرند و چنانچه شرط خروج مسئله فراهم شده باشد الگوریتم خاتمه می‌یابد. الگوریتم ژنتیک چیست؟ بطور کلی یک الگوریتم مبتنی بر تکرار است که اغلب بخش‌های آن به صورت فرایندهای تصادفی انتخاب می‌شوند که این الگوریتم‌ها از بخش‌های تابع برازش، نمایش، انتخاب وتغییر تشکیل می‌شوند.

مقدمه

الگوریتم‌های ژنتیک یکی از الگوریتم‌های جستجوی تصادفی است که ایده آن برگرفته از طبیعت می‌باشد. الگوریتم‌های ژنتیک برای روش‌های کلاسیک بهینه‌سازی در حل مسائل خطی، محدب و برخی مشکلات مشابه بسیار موفق بوده‌اند ولی الگوریتم‌های ژنتیک برای حل مسائل گسسته و غیر خطی بسیار کاراتر می‌باشند. به عنوان مثال می‌توان به مسئله فروشنده دوره‌گرد اشاره کرد. در طبیعت از ترکیب کروموزوم‌های بهتر، نسل‌های بهتری پدید می‌آیند. در این بین گاهی اوقات جهش‌هایی نیز در کروموزوم‌ها روی می‌دهد که ممکن است باعث بهتر شدن نسل بعدی شوند. الگوریتم ژنتیک نیز با استفاده از این ایده اقدام به حل مسائل می‌کند. روند استفاده از الگوریتم‌های ژنتیک به صورت زیر می‌باشد:

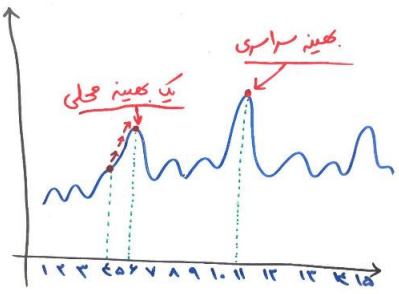
الف) معرفی جواب‌های مسئله به عنوان کروموزوم
ب) معرفی تابع برازندگی (فیت نس)
ج) جمع‌آوری اولین جمعیت
د) معرفی عملگرهای انتخاب
ه) معرفی عملگرهای تولید مثل

در الگوریتم‌های ژنتیک ابتدا به‌طور تصادفی یا الگوریتمیک، چندین جواب برای مسئله تولید می‌کنیم. این مجموعه جواب را جمعیت اولیه می‌نامیم. هر جواب را یک کروموزوم می‌نامیم. سپس با استفاده از عملگرهای الگوریتم ژنتیک پس از انتخاب کروموزوم‌های بهتر، کروموزوم‌ها را باهم ترکیب کرده و جهشی در آن‌ها ایجاد می‌کنیم. در نهایت نیز جمعیت فعلی را با جمعیت جدیدی که از ترکیب و جهش در کروموزوم‌ها حاصل می‌شود، ترکیب می‌کنیم.

پس از اختراع اتومبیل به تدریج و در طی سال‌ها اتومبیل‌های بهتری با سرعت‌های بالاتر و قابلیت‌های بیشتر نسبت به نمونه‌های اولیه تولید شدند. طبیعی است که این نمونه‌های متأخر حاصل تلاش مهندسان طراح جهت بهینه‌سازی طراحی‌های قبلی بوده‌اند. اما دقت کنید که بهینه‌سازی یک اتومبیل، تنها یک «اتومبیل بهتر» را نتیجه می‌دهد.
اما آیا می‌توان گفت اختراع هواپیما نتیجه همین تلاش بوده‌است؟ یا فرضاً می‌توان گفت فضاپیماها حاصل بهینه‌سازی طرح اولیه هواپیماها بوده‌اند؟
پاسخ اینست که گرچه اختراع هواپیما قطعاً تحت تأثیر دستاوردهای‌های صنعت اتومبیل بوده‌است؛ اما به هیچ وجه نمی‌توان گفت که هواپیما صرفاً حاصل بهینه‌سازی اتومبیل یا فضاپیما حاصل بهینه‌سازی هواپیماست. در طبیعت هم عیناً همین روند حکم‌فرماست. گونه‌های متکامل‌تری وجود دارند که نمی‌توان گفت صرفاً حاصل تکامل تدریجی گونه قبلی هستند.
در این میان آنچه شاید بتواند تا حدودی ما را در فهم این مسئله یاری کند مفهومیست به نام تصادف یا جهش.
به عبارتی طرح هواپیما نسبت به طرح اتومبیل یک جهش بود و نه یک حرکت تدریجی. در طبیعت نیز به همین گونه‌است. در هر نسل جدید بعضی از خصوصیات به صورتی کاملاً تصادفی تغییر می‌یابند سپس بر اثر تکامل تدریجی که پیشتر توضیح دادیم در صورتی که این خصوصیت تصادفی شرایط طبیعت را ارضا کند حفظ می‌شود در غیر این‌صورت به شکل اتوماتیک از چرخه طبیعت حذف می‌گردد.
در واقع می‌توان تکامل طبیعی را به این‌صورت خلاصه کرد: جستجوی کورکورانه (تصادف یا Blind Search) + بقای قوی‌تر.
حال ببینیم که رابطه تکامل طبیعی با روش‌های هوش مصنوعی چیست. هدف اصلی روش‌های هوشمندِ به کار گرفته شده در هوش مصنوعی، یافتن پاسخ بهینه مسائل مهندسی است. به عنوان مثال اینکه چگونه یک موتور را طراحی کنیم تا بهترین بازدهی را داشته باشد یا چگونه بازوهای یک ربات را متحرک کنیم تا کوتاه‌ترین مسیر را تا مقصد طی کند (دقت کنید که در صورت وجود مانع یافتن کوتاه‌ترین مسیر دیگر به سادگی کشیدن یک خط راست بین مبدأ و مقصد نیست) همگی مسائل بهینه‌سازی هستند.
روش‌های کلاسیک ریاضیات دارای دو اشکال اساسی هستند. اغلب این روش‌ها نقطه بهینه محلی (Local Optima) را به عنوان نقطه بهینه کلی در نظر می‌گیرند و نیز هر یک از این روش‌ها تنها برای مسئله خاصی کاربرد دارند. این دو نکته را با مثال‌های ساده‌ای روشن می‌کنیم.

بهینه محلی و بهینه کلی

به شکل زیر توجه کنید. این منحنی دارای دو نقطه ماکزیمم می‌باشد؛ که یکی از آن‌ها ماکزیمم محلی است.



حال اگر از روش‌های بهینه‌سازی ریاضی استفاده کنیم مجبوریم تا در یک بازه بسیار کوچک مقدار ماکزیمم تابع را بیابیم. مثلاً از نقطه ۱ شروع کنیم و تابع را ماکزیمم کنیم. بدیهی است اگر از نقطه ۱ شروع کنیم تنها به مقدار ماکزیمم محلی

دست خواهیم یافت و الگوریتم ما پس از آن متوقف خواهد شد. اما در روش‌های هوشمند، به ویژه الگوریتم ژنتیک به دلیل خصلت تصادفی آن‌ها حتی اگر هم از نقطه ۱ شروع کنیم باز ممکن است در میان راه نقطه A به صورت تصادفی انتخاب شود که در این صورت ما شانس دست‌یابی به نقطه بهینه کلی (Global Optima) را خواهیم داشت.

در مورد نکته دوم باید بگوییم که روش‌های ریاضی بهینه‌سازی اغلب منجر به یک فرمول یا دستورالعمل خاص برای حل هر مسئله می‌شوند. در حالی که روش‌های هوشمند دستورالعمل‌هایی هستند که به صورت کلی می‌توانند در حل هر مسئله‌ای به کار گرفته شوند. این نکته را پس از آشنایی با خود الگوریتم بیشتر و بهتر خواهید دید.

نحوه عملکرد الگوریتم ژنتیک روش کار الگوریتم ژنتیک به‌طور فریبنده‌ای ساده، قابل درک و به‌طور قابل ملاحظه‌ای روشی است که ما معتقدیم حیوانات آنگونه تکامل یافته‌اند. هر فرمولی که از طرح داده شده بالا تبعیت کند فردی از جمعیت فرمول‌های ممکن تلقی می‌شود. الگوریتم ژنتیک در انسان متغیرهایی که هر فرمول داده‌شده را مشخص می‌کنند به عنوان یکسری از اعداد نشان داده‌شده‌اند که معادل DNA آن فرد را تشکیل می‌دهند. موتور الگوریتم ژنتیک یک جمعیت اولیه این‌گونه است که هر فرد در برابر مجموعه‌ای از داده‌ها مورد آزمایش قرار می‌گیرد و مناسبترین آن‌ها باقی می‌مانند؛ بقیه کنار گذاشته می‌شوند. مناسبترین افراد با هم جفتگیری (جابجایی عناصر DNA) و (تغییر تصادفی عناصر DNA) کرده و مشاهده می‌شود که با گذشت از میان تعداد زیادی از نسلها، الگوریتم ژنتیک به سمت ایجاد فرمول‌هایی که دقیقتر هستند، میل می‌کنند. در فرمول نهایی برای کاربر انسانی قابل مشاهده خواهد بوده و برای ارائه سطح اطمینان نتایج می‌توان تکنیک‌های آماری متعارف را بر روی این فرمول‌ها اعمال کرد که در نتیجه جمعیت را کلاً قویتر می‌سازند. الگوریتم ژنتیک درمدل سازی مختصراً گفته می‌شود که الگوریتم ژنتیک یک تکنیک برنامه‌نویسی است که از تکامل ژنتیکی به عنوان یک الگوی حل مسئله استفاده می‌کند. مسئله‌ای که باید حل شوددارای ورودی‌هایی می‌باشد که طی یک فرایند الگو برداری شده از تکامل ژنتیکی به راه حلها تبدیل سپس راه حلها به عنوان کاندید توسط تابع ارزیاب (fitness function) مورد ارزیابی قرار گرفته و چنانچه شرط خروج مسئله فراهم باشد الگوریتم خاتمه می یابد. در هر نسل، مناسبترین‌ها انتخاب می‌شوند نه بهترین‌ها. یک راه‌حل برای مسئله مورد نظر، با یک لیست از پارامترها نشان داده می‌شود که به آن‌ها کروموزوم یا ژنوم می‌گویند. کروموزوم‌ها عموماً به صورت یک رشته ساده از داده‌ها نمایش داده می‌شوند، البته انواع ساختمان داده‌های دیگر هم می‌توانند مورد استفاده قرار گیرند. در ابتدا چندین مشخصه به صورت تصادفی برای ایجاد نسل اول تولید می‌شوند. در طول هر نسل، هر مشخصه ارزیابی می‌شود و ارزش تناسب (fitness) توسط تابع تناسب اندازه‌گیری می‌شود.

گام بعدی ایجاد دومین نسل از جامعه است که بر پایه فرایندهای انتخاب، تولید از روی مشخصه‌های انتخاب شده با عملگرهای ژنتیکی است: اتصال کروموزوم‌ها به سر یکدیگر و تغییر.

برای هر فرد، یک جفت والد انتخاب می‌شود. انتخاب‌ها به گونه‌ای‌اند که مناسبترین عناصر انتخاب شوند تا حتی ضعیفترین عناصر هم شانس انتخاب داشته باشند تا از نزدیک شدن به جواب محلی جلوگیری شود. چندین الگوی انتخاب وجود دارد: چرخ منگنه‌دار (رولت)، انتخاب مسابقه‌ای (Tournament) ،... .

معمولاً الگوریتم‌های ژنتیک یک عدد احتمال اتصال دارد که بین ۰،۶ و ۱ است که احتمال به وجود آمدن فرزند را نشان می‌دهد. ارگانیسم‌ها با این احتمال دوباره با هم ترکیب می‌شوند. اتصال ۲ کروموزوم فرزند ایجاد می‌کند، که به نسل بعدی اضافه می‌شوند. این کارها انجام می‌شوند تا این که کاندیدهای مناسبی برای جواب، در نسل بعدی پیدا شوند. مرحله بعدی تغییر دادن فرزندان جدید است. الگوریتم‌های ژنتیک یک احتمال تغییر کوچک و ثابت دارند که معمولاً درجه‌ای در حدود

۰،۰۱ یا کمتر دارد. بر اساس این احتمال، کروموزوم‌های فرزند به‌طور تصادفی تغییر می‌کنند یا جهش می‌یابند، مخصوصاً با جهش بیت‌ها در کروموزوم ساختمان داده‌مان.
این فرایند باعث به وجود آمدن نسل جدیدی از کروموزوم‌هایی می‌شود، که با نسل قبلی متفاوت است. کل فرایند برای نسل بعدی هم تکرار می‌شود، جفت‌ها برای ترکیب انتخاب می‌شوند، جمعیت نسل سوم به وجود می‌آیند و ... این فرایند تکرار می‌شود تا این که به آخرین مرحله برسیم.

شرایط خاتمه الگوریتم‌های ژنتیک عبارتند از:

- به تعداد ثابتی از نسل‌ها برسیم.
- بودجه اختصاص داده‌شده تمام شود (زمان محاسبه/پول).
- یک فرد (فرزند تولید شده) پیدا شود که مینیمم (کمترین) ملاک را برآورده کند.
- بیشترین درجه برازش فرزندان حاصل شود یا دیگر نتایج بهتری حاصل نشود.
- بازرسی دستی.
- ترکیبهای بالا.

مثال عملی

در این مثال می‌خواهیم مسئله ی ۸ وزیر را بوسیله ی این الگوریتم حل کنیم. هدف مشخص کردن چیدمانی از ۸ وزیر در صفحهٔ شطرنج است به نحوی که هیچ‌یک همدیگر را تهدید نکند. ابتدا باید نسل اولیه را تولید کنیم. صفحه شطرنج ۸ در ۸ را در نظر بگیرید. ستونها را با اعداد ۰ تا ۷ و سطرها را هم از ۰ تا ۷ مشخص می‌کنیم. برای تولید حالات (کروموزومها) اولیه به صورت تصادفی وزیرها را در ستونهای مختلف قرار می‌دهیم. باید در نظر داشت که وجود نسل اولیه با شرایط بهتر سرعت رسیدن به جواب را افزایش می‌دهد (اصالت نژاد) به همین خاطر وزیر i ام را در خانهٔ تصادفی در ستون i ام قرار می‌دهیم (به جای اینکه هر مهره‌ای بتواند در هر خانه خالی قرار بگیرد). با اینکار حداقل از برخورد ستونی وزیرها جلوگیری می‌شود. توضیح بیشتر اینکه مثلاً وزیر اول را بطور تصادفی در خانه‌های ستون اول که با ۰ مشخص شده قرار می‌دهیم تا به آخر. =i۰،۰۱، ...۷ حال باید این حالت را به نحوی کمی مدل کرد. چون در هر ستون یک وزیر قرار دادیم هر حالت را بوسیلهٔ رشته اعدادی که عدد k ام در این رشته شمارهٔ سطر وزیر موجود در ستون i ام را نشان می‌دهد. یعنی یک حالت که انتخاب کردیم می‌تواند به صورت زیر باشد: ۶۷۲۰۳۴۲۲ که ۶ شمارهٔ سطر ۶ است که وزیر اول که شمارهٔ ستونش ۰ است می‌باشد تا آخر. فرض کنید ۴ حالت زیر به تصادف تولید شده‌اند. این چهار حالت را به عنوان کروموزومهای اولیه بکار می‌گیریم.

۱.) ۶۷۲۰۳۴۲۰
۲.) ۷۰۰۶۳۳۵۴
۳.) ۱۷۵۲۲۰۶۳
۴.)۴۳۶۰۲۴۷۱

حال نوبت به تابع برازش fitness function می‌رسد. تابعی را که در نظر می‌گیریم تابعی است که برای هر حالت تعداد برخوردها (تهدیدها) را در نظر می‌گیرد. هدف صفر کردن یا حداقل کردن این تابع است. پس احتمال انتخاب کروموزومی برای تولید نسل بیشتر است که مقدار محاسبه شده توسط تابع برازش برای آن کمتر باشد. (روشهای دیگری نیز برای

انتخاب وجود دارد) مقدار برازش برای حالات اولیه به صورت زیر می‌باشد: (مقدار عدد برازش در جلوی هر کروموزوم (با رنگ قرمز)همان تعداد برخوردهای وزیران می‌باشد)

۱. ۶۷۲۰۳۴۲۰(← ۶
۲. ۷۰۰۶۳۳۵۴(← ۸
۳. ۱۷۵۲۲۰۶۳(← ۲
۴. ۴۳۶۰۲۴۷۱(← ۴

در اینجا کروموزومهایی را انتخاب می‌کنیم که برازندگی کمتری دارند. پس کروموزوم ۳ برای crossover با کروموزومهای ۴ و ۱ انتخاب می‌شود. نقطهی ترکیب را بین ارقام ۶ و ۷ در نظر می‌گیریم.

۴ و ۳:

۱. ۱۷۵۲۲۰۷۱(
۲. ۴۳۶۰۲۴۶۳(

۱ و ۳:

۱. ۱۷۵۲۲۰۲۰(
۲. ۶۷۲۰۳۴۶۳(

حال نوبت به جهش می‌رسد. در جهش باید یکی از ژن‌ها تغییر کند.

فرض کنید از بین کروموزومهای ۵ تا ۸ کروموزوم شمارهٔ ۷ و از بین ژن چهارم از ۲ به ۳ جهش یابد. پس نسل ما شامل کروموزومهای زیر با امتیازات نشان داده شده می‌باشد: (امتیازات تعداد برخوردها می‌باشد)

۱. ۶۷۲۰۳۴۲۰(← ۶
۲. ۷۰۰۶۳۳۵۴(← ۸
۳. ۱۷۵۲۲۰۶۳(← ۲
۴. ۴۳۶۰۲۴۷۱(← ۴
۵. ۱۷۵۲۲۰۷۱(← ۶
۶. ۴۳۶۰۲۴۶۳(← ۴
۷. ۱۷۵۳۲۰۲۰(← ۴
۸. ۶۷۲۰۳۴۶۳(← ۵

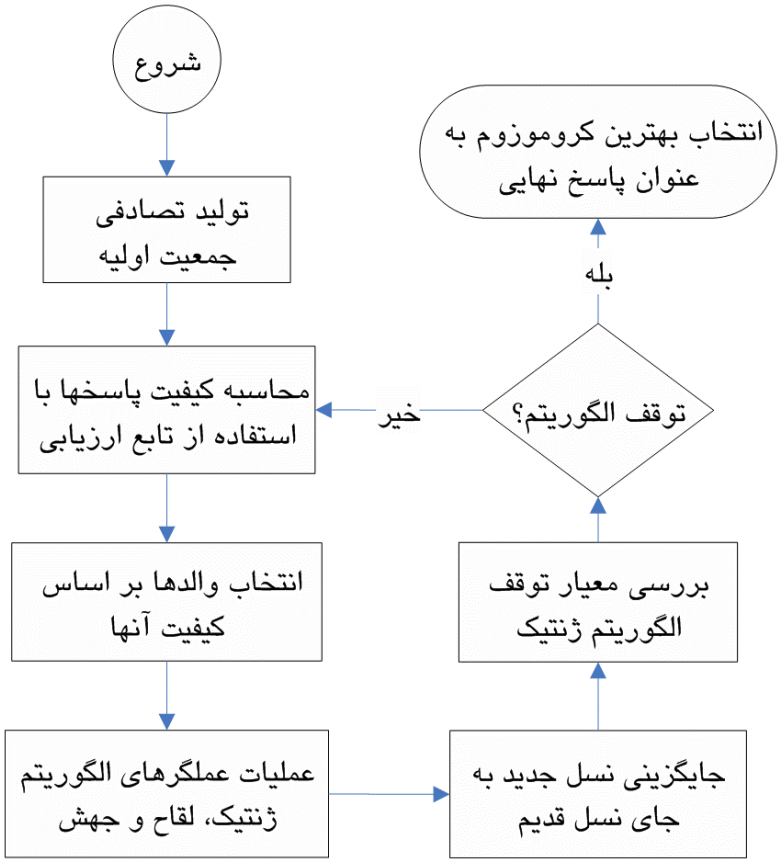
کروموزوم ۳ کاندیدای خوبی برای ترکیب با ۶ و ۷ می‌باشد. (فرض در این مرحله جهشی صورت نگیرد و نقطهٔ اتصال بین ژنهای ۱ و ۲ باشد)

۱. ۶۷۲۰۳۴۲۰(← ۶
۲. ۷۰۰۶۳۳۵۴(← ۸
۳. ۱۷۵۲۲۰۶۳(← ۲
۴. ۴۳۶۰۲۴۷۱(← ۴
۵. ۱۷۵۲۲۰۷۱(← ۶

۶. (۴۳۶۰۲۴۶۳ ← ۴
۷. (۱۷۵۳۲۰۳۰ ← ۴
۸. (۶۷۲۰۳۴۶۳ ← ۵
۹. (۱۳۶۰۲۴۶۳ ← ۲
۱۰. (۴۷۵۲۲۰۶۳ ← ۲
۱۱. (۱۷۵۲۲۰۲۰ ← ۴
۱۲. (۱۷۵۲۲۰۶۳ ← ۲

کروموزومهای از ۹ تا ۱۲ را نسل جدید می‌گوییم. بطور مشخص کروموزوم‌های تولید شده در نسل جدید به جواب مسئله نزدیکتر شده‌اند. با ادامهٔ همین روند پس از چند مرحله به جواب مورد نظر خواهیم رسید.

فلو چارت این الگوریتم



مزایا و معایب استفاده از الگوریتم‌های ژنتیک

این نکته که الگوریتم‌های ژنتیک خوب یا بد هستند، تا حد زیادی به مساله مرتبط می‌شود، به این معنی که در بعضی از کاربردها خوب عمل می‌کنند و در بعضی از کاربردها با توجه به اینکه الگوریتم‌های کلاسیک بهتری برای آنها تعریف شده است، ضعیف‌تر عمل می‌کنند.

البته این نکته را نباید فراموش کنیم که این الگوریتم‌ها پارامترهای بسیار زیادی دارند که با تنظیم صحیح این پارامترها می‌توانیم نتایج بسیار متفاوت و در مواردی نتایج بسیار حیرت‌آوری بدست بیاوریم.

به طور کلی مزایا و معایب زیر را می‌توانیم برای این الگوریتم‌ها معرفی کنیم:

- **مزایا**
  - این الگوریتم‌ها همیشه یک جواب نسبتاً خوب پیدا خواهد کرد
  - در هر مرحله از کار می‌توانیم الگوریتم را متوقف کنیم. در این حالت نیز یک جواب خواهیم داشت، بدیهی است که با پیشرفت کار قاعدتاً جواب بهتری خواهیم داشت.
  - براحتی می‌توانیم این الگوریتم‌ها را بصورت موازی بر روی چند پردازنده اجرا کنیم
- **معایب**
  - یک جواب خوب پیدا می‌کنند ولی ممکن است جواب بهینه را پیدا نکنند
  - به حافظه و محاسبات زیادی نیاز دارند
  - در مورد اینکه جواب پیدا شده چقدر خوب است و آیا جواب بهتری وجود دارد، نمی توانیم هیچگونه ادعائی داشته باشیم
  - پشتوانه ریاضی ضعیفی دارند
  - در دو بار اجرای مختلف، جوابهای متفاوتی دریافت می‌کنیم

پیاده سازی به زبان پایتون

```
import random as rn
def CreatCH(n):  # تابع تولید  یک کروموزوم
        ch=[rn.randint(1, n) for i in range(n)]
        return ch
def Fitnes(p): # تابع  محاسبه برازندگی یک کروموزوم و اضافه کردن به ابتدای کروموزوم
        n=len(p)
        f=0
        for i in range(n-1):
            for j in range(i+1,n):
                if (p[i]==p[j]) or (abs(p[i]-p[j])==abs(i-j)):
                    f+=1
        p.insert(0, f)
        return p
def Crossover(p1,p2):  # تابع  لقاح (تولید فرزندان)
    n=len(p1)
    r=rn.randint(1, n-1)
    ch1=p1.copy()
    ch2=p2.copy()
    ch1[r:]=p2[r:]
    ch2[r:]=p1[r:]
    return ch1,ch2
```

```
def CreatPop(n,npop): # تابع تولید جمعیت اولیه
    population=[CreatCH(n) for i in range(npop)]
    return population
def Mution(p): # تابع جهش
    r=rn.randint(0, len(p)-1)
    p[r]=rn.randint(1, len(p))
    return p
```

 # شروع الگوریتم ژنتیک

```
n=8
npop=100
population=CreatPop(n, npop) # ایجاد جمعیت اولیه
for i in range(npop):
    population[i]=Fitnes(population[i]) # محاسبه برازندگی جمعیت اولیه
population.sort() # مرتب سازی بر اساس برازندگی
best=population[0]
```

 # حلقه اصلی

```
while best[0]!=0:
    Newpop=[]
    for i in range(0,npop,2):
        ch1,ch2=Crossover(population[i][1:], population[i+1][1:])
        ch1=Mution(ch1)
        ch2=Mution(ch2)
        ch1=Fitnes(ch1)
        ch2=Fitnes(ch2)
        Newpop.append(ch1)
        Newpop.append(ch2)
    population=population+Newpop
    population.sort()
    population=population[:npop]
    best=population[0]
    print(best)
```