محمدامین پورزارع CA2 810199392

قسمت دوم : ژنتیک

- بخش یک: مشخص کردن مفاهیم اولیه

در ابتدا باید مفاهیم اولیه را برای خود مشخص کنیم. یعنی تعیین کنیم که کروموزوم ما به چه شکل میباشد و از چه ژن هایی تشکیل شده است. کروموزوم ما یک لیست از کاراکتر ها میباشد که یک عبارت ریاضی رندوم را درون آن نگه داری میکنیم به شکلی که طول آن به اندازه ی تعداد جایگاهی است که در ورودی آن را تعریف میکنیم و ژن های آن یا operand هستند و یا operator. همانطور که میدانیم operand ها و operator ها در ورودی مشخص میشوند و واضح است که در کروموزوم یکی در میان میباشند.

- بخش دو: تولید جمعیت اولیه

در این بخش باید جمعیت اولیه ی مان را تشکیل دهیم. به عبارتی دیگر عمل initialization را انجام میدهیم.

باید توجه کنیم که جمعیت اولیه ی ما باید کاملا به طور رندوم انتخاب شوند. پس عبارت های ریاضی ای داریم که operand ها و operator های آن کاملا به طور تصادفی انتخاب شده اند. در این مسئله تعداد جامعه را ۱۰۰ گذاشته ایم.

● بخش سه: پیاده سازی و مشخص کردن تابع معیار سازگاری

در این بخش ما باید به هر کروموزوم یک fitness نسبت بدهیم. کاربرد fitness این است که میتوانیم کروموزوم های برتر را شناسایی کنیم و احتمال بیشتری برای آنها قائل باشیم که به جمعیت مرحله ی بعد بروند. در ابتدا یک تابع به اسم d(x) تعریف میکنیم که نشان دهنده ی تفاضل عبارت ریاضی داخل کروموزوم ما با جواب مد نظر که در ورودی داده میشود میباشد. پس d(x) تمام کروموزوم ها را بدست میاوریم و fitness را به این شکل تعریف میکنیم که fitness هر کروموزوم به شکل زیر میباشد :

```
fitnesses[i] = 1 / (1 + difference)
```

به علاوه ی یک میکنیم که هیچ وقت با مخرج مساوی با صفر برخورد نکنیم. در ادامه باید کاری کنیم که کروموزوم هایی که اختلاف کمتری با جواب مدنظرمان دارند احتمال بیشتری برای انتقال به جمعیت مرحله ی بعد داشته باشند. پس احتمال هر کروموزوم را به شکل زیر محاسبه میکنیم :

```
probability[i] = fitnesses[i] / sumOfFitnesses
```

که این یعنی هر کروموزومی که fitness بالاتری داشته باشد از احتمال بیشتری برای انتقال به مرحله ی بعد برخورد دار میباشد.

در مرحله ی بعد احتمالات تجمعی را بدست میاوریم و سپس به اندازه ی سایز جمعیتمان عدد رندوم تولید میکنیم و کروموزوم هارا با توجه به عدد رندوم انتخاب میکنیم و به مرحله ی بعد میفرستیم.

● بخش چهار: پیاده سازی crossover و mutation و تولید جمعیت بعدی

در این مرحله باید crossover و mutation را پیاده سازی کنیم که ابتدا کروموزوم های انتقال داده شده در مرحله قبل را میگیریم و به طور رندوم تعدادی از آنهارا انتخاب میکنیم و عمل کراس اور را بر روی انها انجام میدهیم به شکلی که دو کروموزوم را از یک جا قطع میکنیم و تکه هارا جابجا میکنیم.

سپس در مرحله ی بعد mutation را به شکلی انجام میدهیم که به طوری رندوم تعدادی از ژن های بعضی از کروموزوم هارا به طور رندوم تغییر میدهیم.

● بخش پنج: ایجاد الگوریتم ژنتیک روی مسئله

در این بخش با توجه به عملیات گفته شده در بالا الگوریتم را بر روی کروموزوم ها اجرا میکنیم و نتیجه را بدست میاوریم.

● بخش شش: سوالات

1 - جمعیت اولیه ی بسیار کم یا بسیار زیاد چه مشکلاتی را به وجود می آورند؟

در صورتی که جمعیت اولیه کم باشد تعداد کروموزوم های ما کم میباشد و حلقه ای که برای الگوریتم genetic پیاده سازی کرده ایم به تعداد زیادی میچرخد و اجرا میشود. با توجه به اینکه عمل های crossover و mutation را بر روی کروموزوم هایمان انجام میدهیم تا بتوانیم با تولید کروموزوم های مختلف در هر مرحله تا حد توان به رسیدن به جواب نزدیک شویم. اما با کم بودن جمعیت اولیه روند ما کند میشود و تعداد زیادی باید اعمال را روی انها اجرا کنیم. در صورتی که جمعیت ما زیاد باشد تعداد کروموزوم ها زیاد میشود و این باعث میشود در مراحلی مانند crossover و mutation که بر روی کروموزوم ها تغییراتی انجام میدهیم و کروموزوم ها را بررسی میکنیم تعداد زیادی کروموزوم را پیمایش کنیم و این باعث کند شدن برنامه ی ما میشود.

برای مثال:

جمعیت کم در برنامه = ۱۰

جمعیت خوب در برنامه = ۱۰۰

جمعیت زیاد در برنامه = ۱۰۰۰۰

2 - اگر تعداد جمعیت در هر دوره افزایش یابد، چه تاثیری روی دقت و سرعت الگوریتم می گذارد؟

اگر تعداد جمعیت در هر مرحله زیاد شود قطعا سرعت ما کمتر میشود چون باعث میشود کروموزوم های بیشتری تولید شود و حافظه ی مصرفی ما بیشتر میشود. اما دقت الگوریتم ممکن است بیشتر شود اما نیازی به این کار نیست چون قرار است جمعیت را همگرا کنیم و در هر مرحله تعداد کروموزوم های بدتر کمتر شوند.

3 - تاثیر هر یک از عملیات های crossover و mutation را بیان و مقایسه کنید. آیا می توان فقط یکی از آنها را استفاده کرد ؟ چرا ؟

در کل دو عمل crossover و mutation به این منظور انجام میشود که کروموزوم ها را به کروموزوم های بهتری تبدیل کنیم. به این صورت که عمل crossover را انجام میدهیم به طوری که ۲ یا چند کروموزوم را ترکیب میکنیم و کروموزوم های جدیدی میسازیم و عمل mutation هم به این شکل میباشد که به صورت رندوم تعدادی از ژن های کروموزوم ها را به طور رندوم انتخاب میکنیم و مقدار آنهارا تغییر میدهیم. این عملیات میتواند شکل های مختلفی داشته باشد و الگوریتم های متفاوتی برای آن موجود است.

در کل این دو عمل به این منظور است که در کروموزوم ها تغییری ایجاد کنیم که در مرحله ی بعد که قرار بود با احتمالات خوب بودن آنهارا بررسی کنیم کروموزوم هایی که بدتر هستند را به مرور حذف کنیم. میتوان از یکی از دو عمل crossover یا mutation استفاده کرد و در اخر هم جواب بگیریم اما قطعا این کار بهینه نمیباشد و اگر از هر دوی انها استفاده کنیم در وقت بسیار صرفه جویی میشود.

4 - به نظر شما چه راهکارهایی برای سریعتر به جواب رسیدن در این مسئله ی خاص وجود دارد؟

برای سریع تر به جواب رسیدن میتوانیم از الگوریتم های پیشرفته تر در دو عمل crossover و mutation بهره ببریم. از طرفی دیگر میتوانیم پارامترهایی که در برنامه ی مان داریم را تغییر دهیم. پارامتر ها حاوی جمعیت اولیه و mutation_rate و crossover_rate و چیز های دیگر میباشد. کار دیگری هم که میتوان کرد این است که fitness هر کروموزوم را هوشمندانه تر محاسبه کنیم.

5 - با وجود استفاده از این روش ها، باز هم ممکن است که کروموزوم ها پس از چند مرحله دیگر تغییر نکنند. دلیل این اتفاق و مشکلاتی که به وجود می آورد را شرح دهید. برای حل آن چه پیشنهادی می دهید؟ (راه حلهای خود را امتحان کنید و بهترین آن ها را روی پروژه خود پیاده سازی کنید).

عمل mutation دقیقا کارش همین است که نگذارد این اتفاق رخ دهد. یعنی حتی اگر در بد ترین حالت همه ی کروموزوم هایمان با هم برابر شده بودند و crossover دیگر تاثیری بر روی آنها نداشت اینجا است که mutation ژن های کروموزوم هارا تغییر میدهد و تنوع کروموزوم هارا در هر مرحله بیشتر میکند. پس با تکرار این عملیات این مشکل حل میشود.

6 - چه راه حلی برای تمام شدن برنامه در صورتی که مسئله جواب نداشته باشد پیشنهاد می دهید؟

یکی از کار هایی که میتوانیم برای این کار انجام دهین این است که تعداد نسل هایمان را محدود کنیم و تا یک جای مشخص جلو برویم

قسمت دوم : بازی

در این پروژه ابتدا تابع minimax را پیاده سازی کردیم و سپس یک heuristic برای evaluation function ارائه کردیم که در بخش سوالات به صورت مفصل به آن میپردازیم. و در آخر هم از هرس آلفا و بتا استفاده میکنیم تا استست های کم تری را پیمایش بکنیم و در زمان صرفه جویی کنیم.

حال به بررسی نتایج اجرای برنامه میپردازیم.

میانگین زمان اجرا بر حسب میلی ثانیه است.

	شانس پیروزی	تعداد نودهای مورد بررسی	میانگین زمان اجرا
Depth = 1	%99.4	46	0.629643898010232
Depth = 3	%94.4	5260	42.84261035919171
Depth = 5	%76.6	421164	4,371.746314452688

تست های زمان اجرا برای تست با depth = 1 :

Test 1 : time execution = 62.83092498779297 ms

result = {red': 99, 'blue': 1'}

Test 2 : time execution = 62.80611991882324 ms

result = {'red': 100, 'blue': 0}

Test 3 : time execution = 63.52066993713379 ms

result = {'red': 100, 'blue': 0}

Test 4 : time execution = 63.83061408996582 ms

result = {'red': 99, 'blue': 1}

Test 5 : time execution = 61.83362007141113 ms

result = {'red': 99, 'blue': 1}

تست های زمان اجرا برای تست با depth = 3 :

Test 1 : time execution = 4308.717250823975 ms

result = {'red': 96, 'blue': 4}

Test 2 : time execution = 4308.417797088623 ms

result = {'red': 93, 'blue': 7}

Test 3 : time execution = 4231.712579727173 ms

result = {'red': 95, 'blue': 5}

Test 4 : time execution = 4282.234907150269 ms

result = {'red': 94, 'blue': 6}

Test 5 : time execution = 4290.222644805908 ms

result = {'red': 94, 'blue': 6}

تست های زمان اجرا برای تست با depth = 5 :

Test 1 : time execution = 218799.85785484314 ms

result = {red': 40, 'blue': 10'}

Test 2 : time execution = 219029.85785484314 ms

result = {red': 38, 'blue': 12'}

Test 3 : time execution = 217932.23145821738 ms

result = {red': 37, 'blue': 13'}

حال در مرحله ی بعد هرس آلفا و بتا رو به کدمون اضافه میکنیم و معیار ها را بررسی میکنیم.

	شانس پیروزی	تعداد نودهای مورد بررسی	میانگین زمان اجرا
Depth = 1	% 99.6	42	0.45672745647674
Depth = 3	% 94.2	2,570	18.3299084663391
Depth = 5	% 75	61,195	547.407661676405
Depth = 7	% 64	828,825	8,823.81315563201

تست های زمان اجرا برای تست با depth = 1 :

Test 1 : time execution = 238.3637282383749 ms

result = {red': 498, 'blue': 2'}

تست های زمان اجرا برای تست با depth = 3 :

Test 1 : time execution = 9364.954233169556 ms

result = {red': 471, 'blue': 29'}

تست های زمان اجرا برای تست با depth = 5 :

Test 1 : time execution = 109481.53233528137 ms

result = {red': 150, 'blue': 50'}

تست های زمان اجرا برای تست با depth = 7 :

Test 1 : time execution = 444610.51759529114 ms

result = {red': 33, 'blue': 17'}

Test 2 : time execution = 437770.79796791077 ms

result = {red': 31, 'blue': 19'}

● سوالات

سوال ۱: یک heuristic خوب چه ویژگی هایی دارد؟ علت انتخاب heuristic شما و دلیل برتری آن نسبت به تعدادی از روش های دیگر را بیان کنید.

یک heuristic خوب به گونه ای میباشد که تخمین نزدیکی در مقایسه با هزینه ی واقعی داشته باشد. از طرفی باید به شکلی باشد که بهینه باشد و زمان بیش از اندازه ای مصرف نکند. heuristic انتخاب شده در برنامه ی من به شکلی عمل میکرد که تمام صفحه ی بازی را بررسی میکرد و مثلث هایی را بررسی میکرد که هر سه ضلع آن یک رنگ بودند و مثلث هایی که دو ضلع آن یک رنگ و ضلع دیگر آن از رنگ دیگری بود و مثلث هایی که دو ضلع آن از یک رنگ و در ضلع سوم آن هیچ یالی وجود نداشت و خالی بود را بررسی میکرد. این معیار ها میزان خوب بودن وضعیت را برای ما نشان میدهد.

سوال ۲: آیا میان عمق الگوریتم و پارامترهای حساب شده روابطی می بینید؟ به طور کامل بررسی کنید که عمق الگوریتم چه تاثیراتی بر روی شانس پیروزی، زمان و گر ههای دیده شده می گذارد.

با توجه به نتایجی که در بالا به دست آمده است مشاهده میکنیم که هر چقدر که عمق الگوریتم زیاد میشود شانس پیروزی ما کم تر میشود. دلیل آن این است که وقتی که عمق کم را بررسی میکنیم زمین بازی را در تعداد مراحل بعدی کمتری بررسی میکنیم. از آنجایی که حریف ما رندوم بازی میکند وقتی عمق را زیاد کنیم و فرض میکنیم که حریف اوپتیمال بازی میکند این برای ما مشکل ساز میشود بعضی اوقات چون حریف ما

به شکل رندوم حرکت ها را انتخاب میکند و ما در اصل بهتر غافل گیر میشویم.

هر چقدر عمق بیشتر شود گره های دیده شده و زمان برنامه بیشتر میشوند که دلیل آن مشخص است و ناشی از آن است که نود های بیشتری را ما بررسی میکنیم.

سوال ۳: وقتی از روش هرس کردن استفاده می کنید، برای هر گره درخت، فرزندانش به چه ترتیبی اضافه می شوند؟ آیا این ترتیب اهمیت دارد؟ چرا این ترتیب را انتخاب کردید؟

از آنجایی که ما نمیدانیم بیشترین مقدار و کمترین مقدار نود ها در کدام طرف فرزندان یک نود قرار دارند پس تفاوتی ندارد که ما از کدام طرف فرزندان را بررسی کنیم. صرفا در روش هرس کردن درخت نود هایی که مطمئن میشویم از یک مقدار بیشتر و یا کمتر میشوند را هرس میکنیم.