الف)

در کلاس state یک آرایه ۸ خانه ای (با فرض در نظر نگرفتن ۰ (برای راحتی از اندیس ۰ استفاده نکرده ام)) برای نشان دادن شماره ستون هر وزیر وجود دارد (شماره سطر هر وزیر ثابت و برابر شماره ی خود وزیر است.)

x[2] = 3

یعنی وزیر ۲ام در سطر ۲ و ستون شماره ۳ قرار دار.

```
public class State8Queen extends State{
	int[] x = new int[9];
}
```

تابع هدف: خروجی این تابع برابر با تعداد conflict های موجود است یعنی چه تعداد از وزیر ها همدیگر را میتوانند بزنند. بعضی از حالت ها ۲بار حساب شده برای بهینه بودن الگوریتم.

```
public double f(State state_inter) {
		State8Queen state = (State8Queen)state_inter;

		int conflicts = 0;
		for (int i = 1; i <= 8 ; i++) {
			for (int j = 1; j <= 8; j++) {
				if (j!=i && (state.x[j] == state.x[i] || (Math.abs(state.x[j] - state.x[i]) == Math.abs(j - i)) ) ) {
					conflicts++;
				}
			}
		}
		return conflicts;
	}
```

نحوه ی ایجاد یک حالت تصادفی:یکی از اکشن های تابع actions را به صورت تصادفی به عنوان حالت بعدی انتخاب میکند. اگر به جواب نرسید یعنی از استیت فعلی بدتر بود اکشن دیگری را انتخاب میکند. هرگز ۱ اکشن ۲بار انتخاب نمیشود.

```
while (observeNode.contains(false) && bestAction == null) {
				int index = (int) (Math.random() * actions.size());
				while (observeNode.get(index)) {
					index = (index + 1) % actions.size();
				}
				State neighbour = problem.result(p, actions.get(index));
				double neighbourWorth = problem.f(neighbour);
				observedNodes++;
				observeNode.set(index, true);
				if (neighbourWorth < pCost) {
					pCost = neighbourWorth;
					bestNeighbour = neighbour;
					bestAction = actions.get(index);
				}
			}
```

ب)randomRestart

```
finalState[0]: 4.0
finalState[1]: 2.0
finalState[2]: 4.0
finalState[3]: 2.0
finalState[4]: 2.0
finalState[5]: 4.0
finalState[6]: 0.0
Number of observed nodes : 321
Number of extended nodes : 4
Path states : ( [1]:4 [2]:3 [3]:4 [4]:1 [5]:7 [6]:7 [7]:6 [8]:7 ) ( [1]:4 [2]:3 [3]:4 [4]:1 [5]:7 [6]:2 [7]:6 [8]:7 ) ( [1]:5 [2]:3 [3]:4 [4]:1 [5]:7 [6]:2
Path actions : ( q[6] -> 2 ) ( q[1] -> 5 ) ( q[2] -> 8 ) ( q[8] -> 3 )
Path cost : 4.0
Final state worth : 0.0
```

finalState[0]: 4.0
finalState[1]: 2.0
finalState[2]: 4.0
finalState[3]: 2.0
finalState[4]: 2.0
finalState[5]: 4.0
finalState[6]: 0.0
Number of observed nodes : 321
Number of extended nodes : 4
Path states : ([1]:4 [2]:3 [3]:4 [4]:1 [5]:7 [6]:7 [7]:6 [8]:7) ([1]:4 [2]:3 [3]:4 [4]:1 [5]:7 [6]:2 [7]:6 [8]:7) ([1]:5 [2]:3 [3]:4 [4]:1 [5]:7 [6]:2 [7]:6 [8]:7) ([1]:5 [2]:8 [3]:4 [4]:1 [5]:7 [6]:2 [7]:6 [8]:7) ([1]:5 [2]:8 [3]:4 [4]:1 [5]:7 [6]:2 [7]:6 [8]:3)
Path actions : (q[6] -> 2) (q[1] -> 5) (q[2] -> 8) (q[8] -> 3)
Path cost : 4.0
Final state worth : 0.0

به طور مشخص finalState هایی که غیر ۰ هستند بهینه ی محلی بوده و الگوریتم از آنها عبور کرده است.

ج)
Number of Generations : 599

best , worst , average value in each generation :
[1](10.0 , 22.0 , 14.0)
[2](10.0 , 22.0 , 13.0)
[3](10.0 , 14.0 , 12.5)
[4](10.0 , 16.0 , 12.75)
[5](10.0 , 16.0 , 13.0)
[6](10.0 , 16.0 , 12.0)
[7](10.0 , 16.0 , 11.75)
[8](10.0 , 12.0 , 10.75)
[9](10.0 , 16.0 , 11.25)
[10](10.0 , 18.0 , 12.25)
[11](10.0 , 16.0 , 11.75)
[12](10.0 , 18.0 , 12.75)
[13](8.0 , 14.0 , 11.25)
[14](8.0 , 16.0 , 12.0)
[15](8.0 , 22.0 , 12.75)
[16](8.0 , 18.0 , 12.25)
[17](8.0 , 18.0 , 12.5)
[18](8.0 , 14.0 , 12.0)
[19](8.0 , 14.0 , 11.75)
[20](8.0 , 14.0 , 11.25)

[21](8.0 , 14.0 , 11.0)
[22](8.0 , 14.0 , 9.25)
[23](8.0 , 16.0 , 11.0)
[24](8.0 , 16.0 , 10.25)
[25](8.0 , 16.0 , 10.5)
[26](8.0 , 14.0 , 11.25)
[27](10.0 , 14.0 , 11.75)
[28](10.0 , 18.0 , 12.5)
[29](10.0 , 18.0 , 12.5)
[30](10.0 , 18.0 , 13.25)
[31](10.0 , 18.0 , 12.75)
[32](10.0 , 18.0 , 12.5)
[33](10.0 , 18.0 , 13.0)
[34](10.0 , 18.0 , 13.0)
[35](10.0 , 18.0 , 13.25)
[36](10.0 , 20.0 , 12.75)
[37](8.0 , 20.0 , 12.5)
[38](8.0 , 20.0 , 12.5)
[39](8.0 , 16.0 , 11.75)
[40](8.0 , 16.0 , 11.25)
[41](8.0 , 16.0 , 11.25)
[42](8.0 , 12.0 , 10.25)
[43](8.0 , 14.0 , 10.75)
[44](8.0 , 18.0 , 11.0)
[45](8.0 , 14.0 , 9.75)
[46](8.0 , 16.0 , 10.25)
[47](8.0 , 16.0 , 11.5)
[48](8.0 , 16.0 , 11.25)
[49](8.0 , 14.0 , 11.0)
[50](8.0 , 14.0 , 12.0)
[51](8.0 , 16.0 , 11.75)
[52](8.0 , 16.0 , 11.25)
[53](6.0 , 16.0 , 10.25)
[54](6.0 , 16.0 , 10.0)
[55](6.0 , 14.0 , 9.0)
[56](6.0 , 14.0 , 8.5)
[57](6.0 , 10.0 , 8.0)
[58](6.0 , 10.0 , 8.25)
[59](6.0 , 10.0 , 8.0)
[60](6.0 , 16.0 , 8.75)
[61](6.0 , 16.0 , 9.5)
[62](8.0 , 16.0 , 10.5)
[63](8.0 , 16.0 , 10.5)
[64](8.0 , 16.0 , 10.5)
[65](8.0 , 16.0 , 10.5)
[66](8.0 , 18.0 , 11.25)
[67](8.0 , 20.0 , 11.25)
[68](8.0 , 20.0 , 10.25)
[69](8.0 , 20.0 , 11.0)
[70](8.0 , 12.0 , 9.5)
[71](8.0 , 12.0 , 9.5)
[72](8.0 , 12.0 , 9.5)
[73](8.0 , 16.0 , 10.5)
[74](6.0 , 16.0 , 10.0)

[75](8.0 , 16.0 , 10.0)
[76](8.0 , 16.0 , 11.0)
[77](6.0 , 16.0 , 10.75)
[78](6.0 , 14.0 , 10.0)
[79](6.0 , 14.0 , 10.5)
[80](6.0 , 14.0 , 11.0)
[81](6.0 , 14.0 , 10.25)
[82](6.0 , 12.0 , 9.25)
[83](6.0 , 12.0 , 9.75)
[84](6.0 , 12.0 , 9.0)
[85](6.0 , 16.0 , 11.5)
[86](6.0 , 16.0 , 10.75)
[87](8.0 , 16.0 , 11.25)
[88](10.0 , 16.0 , 12.0)
[89](8.0 , 16.0 , 11.75)
[90](8.0 , 16.0 , 11.25)
[91](8.0 , 14.0 , 11.0)
[92](8.0 , 14.0 , 11.0)
[93](8.0 , 12.0 , 10.5)
[94](10.0 , 18.0 , 11.75)
[95](10.0 , 12.0 , 11.0)
[96](10.0 , 20.0 , 12.0)
[97](10.0 , 12.0 , 11.0)
[98](10.0 , 14.0 , 11.25)
[99](10.0 , 12.0 , 11.0)
[100](10.0 , 12.0 , 11.5)
[101](10.0 , 12.0 , 11.5)
[102](10.0 , 12.0 , 11.5)
[103](10.0 , 12.0 , 11.25)
[104](10.0 , 20.0 , 12.75)
[105](10.0 , 20.0 , 14.25)
[106](12.0 , 20.0 , 15.25)
[107](12.0 , 20.0 , 14.5)
[108](8.0 , 16.0 , 13.0)
[109](8.0 , 16.0 , 13.25)
[110](8.0 , 16.0 , 12.75)
[111](8.0 , 16.0 , 13.25)
[112](10.0 , 16.0 , 12.75)
[113](8.0 , 16.0 , 12.5)
[114](8.0 , 16.0 , 12.25)
[115](10.0 , 18.0 , 13.75)
[116](10.0 , 18.0 , 13.75)
[117](12.0 , 18.0 , 14.0)
[118](12.0 , 16.0 , 13.75)
[119](12.0 , 18.0 , 14.25)
[120](12.0 , 14.0 , 13.25)
[121](12.0 , 16.0 , 13.75)
[122](12.0 , 18.0 , 14.0)
[123](10.0 , 20.0 , 14.5)
[124](10.0 , 18.0 , 13.25)
[125](10.0 , 18.0 , 13.0)
[126](10.0 , 18.0 , 13.25)
[127](6.0 , 18.0 , 12.0)
[128](6.0 , 18.0 , 11.25)

[129](6.0 , 14.0 , 10.75)
[130](6.0 , 14.0 , 11.25)
[131](10.0 , 16.0 , 12.0)
[132](10.0 , 14.0 , 12.0)
[133](10.0 , 14.0 , 12.0)
[134](8.0 , 12.0 , 11.0)
[135](8.0 , 14.0 , 11.5)
[136](8.0 , 16.0 , 13.0)
[137](8.0 , 16.0 , 12.25)
[138](8.0 , 18.0 , 13.0)
[139](8.0 , 18.0 , 13.5)
[140](8.0 , 16.0 , 11.75)
[141](8.0 , 14.0 , 11.0)
[142](8.0 , 16.0 , 11.25)
[143](8.0 , 14.0 , 10.75)
[144](8.0 , 12.0 , 10.25)
[145](6.0 , 12.0 , 10.0)
[146](6.0 , 20.0 , 12.75)
[147](6.0 , 20.0 , 12.25)
[148](6.0 , 20.0 , 10.75)
[149](6.0 , 20.0 , 11.5)
[150](10.0 , 14.0 , 10.75)
[151](8.0 , 16.0 , 10.5)
[152](8.0 , 16.0 , 11.0)
[153](8.0 , 16.0 , 11.0)
[154](8.0 , 16.0 , 11.0)
[155](8.0 , 16.0 , 11.75)
[156](8.0 , 14.0 , 10.75)
[157](10.0 , 16.0 , 12.25)
[158](10.0 , 26.0 , 13.75)
[159](8.0 , 26.0 , 12.25)
[160](8.0 , 26.0 , 12.75)
[161](10.0 , 26.0 , 14.0)
[162](10.0 , 26.0 , 14.0)
[163](10.0 , 18.0 , 12.0)
[164](10.0 , 18.0 , 12.75)
[165](10.0 , 18.0 , 12.0)
[166](10.0 , 18.0 , 11.5)
[167](10.0 , 14.0 , 11.0)
[168](10.0 , 14.0 , 11.0)
[169](10.0 , 14.0 , 11.75)
[170](10.0 , 14.0 , 11.25)
[171](10.0 , 14.0 , 11.0)
[172](8.0 , 14.0 , 11.0)
[173](8.0 , 14.0 , 10.75)
[174](8.0 , 12.0 , 10.0)
[175](8.0 , 16.0 , 11.0)
[176](8.0 , 16.0 , 10.75)
[177](8.0 , 12.0 , 10.25)
[178](8.0 , 12.0 , 10.25)
[179](8.0 , 16.0 , 10.75)
[180](8.0 , 16.0 , 11.25)
[181](8.0 , 16.0 , 11.25)
[182](8.0 , 16.0 , 12.0)

[183](8.0 , 22.0 , 13.75)
[184](8.0 , 22.0 , 13.0)
[185](8.0 , 16.0 , 11.5)
[186](8.0 , 14.0 , 11.25)
[187](10.0 , 14.0 , 12.5)
[188](10.0 , 14.0 , 12.5)
[189](10.0 , 14.0 , 12.5)
[190](10.0 , 20.0 , 13.5)
[191](10.0 , 20.0 , 12.75)
[192](10.0 , 20.0 , 12.75)
[193](10.0 , 20.0 , 13.0)
[194](10.0 , 20.0 , 14.0)
[195](10.0 , 20.0 , 13.75)
[196](10.0 , 20.0 , 12.75)
[197](10.0 , 20.0 , 14.25)
[198](10.0 , 14.0 , 12.0)
[199](8.0 , 14.0 , 11.75)
[200](6.0 , 16.0 , 11.25)
[201](6.0 , 16.0 , 9.75)
[202](6.0 , 16.0 , 9.5)
[203](6.0 , 12.0 , 8.25)
[204](6.0 , 12.0 , 8.25)
[205](6.0 , 10.0 , 7.5)
[206](6.0 , 8.0 , 6.75)
[207](6.0 , 8.0 , 7.0)
[208](6.0 , 8.0 , 6.75)
[209](6.0 , 8.0 , 6.75)
[210](6.0 , 10.0 , 7.25)
[211](6.0 , 10.0 , 7.5)
[212](6.0 , 12.0 , 7.75)
[213](6.0 , 12.0 , 8.5)
[214](6.0 , 12.0 , 8.5)
[215](6.0 , 12.0 , 8.0)
[216](6.0 , 12.0 , 8.0)
[217](6.0 , 10.0 , 7.5)
[218](6.0 , 14.0 , 8.5)
[219](6.0 , 14.0 , 8.75)
[220](6.0 , 14.0 , 10.25)
[221](6.0 , 14.0 , 10.75)
[222](6.0 , 14.0 , 9.25)
[223](6.0 , 18.0 , 10.25)
[224](6.0 , 18.0 , 10.75)
[225](6.0 , 18.0 , 10.75)
[226](6.0 , 14.0 , 9.5)
[227](6.0 , 14.0 , 10.25)
[228](8.0 , 14.0 , 10.75)
[229](8.0 , 14.0 , 10.75)
[230](8.0 , 16.0 , 11.5)
[231](8.0 , 16.0 , 11.75)
[232](8.0 , 16.0 , 13.0)
[233](8.0 , 16.0 , 13.0)
[234](12.0 , 16.0 , 13.25)
[235](12.0 , 16.0 , 13.0)
[236](12.0 , 16.0 , 12.75)

[237](12.0 , 16.0 , 13.0)
[238](12.0 , 16.0 , 13.0)
[239](12.0 , 16.0 , 14.0)
[240](12.0 , 16.0 , 14.0)
[241](8.0 , 16.0 , 14.0)
[242](8.0 , 16.0 , 12.75)
[243](8.0 , 16.0 , 12.0)
[244](8.0 , 20.0 , 12.0)
[245](6.0 , 20.0 , 11.0)
[246](8.0 , 20.0 , 11.5)
[247](8.0 , 20.0 , 11.0)
[248](8.0 , 20.0 , 12.0)
[249](8.0 , 20.0 , 12.0)
[250](8.0 , 20.0 , 11.75)
[251](8.0 , 18.0 , 11.25)
[252](8.0 , 20.0 , 11.75)
[253](8.0 , 18.0 , 10.0)
[254](8.0 , 12.0 , 9.75)
[255](8.0 , 12.0 , 10.75)
[256](8.0 , 12.0 , 10.0)
[257](8.0 , 18.0 , 10.75)
[258](8.0 , 12.0 , 9.0)
[259](8.0 , 12.0 , 9.0)
[260](8.0 , 12.0 , 9.25)
[261](8.0 , 12.0 , 9.0)
[262](8.0 , 12.0 , 9.5)
[263](8.0 , 12.0 , 10.5)
[264](8.0 , 12.0 , 11.0)
[265](8.0 , 14.0 , 11.5)
[266](12.0 , 14.0 , 12.25)
[267](12.0 , 14.0 , 12.5)
[268](12.0 , 16.0 , 13.0)
[269](12.0 , 14.0 , 12.5)
[270](12.0 , 18.0 , 12.75)
[271](12.0 , 18.0 , 13.25)
[272](12.0 , 18.0 , 13.5)
[273](12.0 , 18.0 , 13.5)
[274](12.0 , 18.0 , 13.5)
[275](12.0 , 18.0 , 13.5)
[276](12.0 , 14.0 , 12.5)
[277](12.0 , 16.0 , 12.75)
[278](8.0 , 16.0 , 11.75)
[279](8.0 , 16.0 , 12.25)
[280](8.0 , 14.0 , 11.5)
[281](8.0 , 14.0 , 11.0)
[282](6.0 , 14.0 , 9.5)
[283](6.0 , 12.0 , 8.75)
[284](6.0 , 12.0 , 9.25)
[285](6.0 , 12.0 , 8.75)
[286](6.0 , 12.0 , 9.25)
[287](8.0 , 20.0 , 11.5)
[288](6.0 , 20.0 , 11.5)
[289](6.0 , 16.0 , 10.25)
[290](6.0 , 18.0 , 11.5)

[291](6.0 , 12.0 , 9.0)
[292](6.0 , 18.0 , 10.25)
[293](6.0 , 12.0 , 9.5)
[294](6.0 , 12.0 , 9.5)
[295](6.0 , 12.0 , 9.25)
[296](6.0 , 12.0 , 9.25)
[297](6.0 , 10.0 , 8.5)
[298](6.0 , 12.0 , 8.75)
[299](6.0 , 14.0 , 9.5)
[300](6.0 , 14.0 , 10.0)
[301](6.0 , 14.0 , 11.5)
[302](6.0 , 14.0 , 10.5)
[303](6.0 , 14.0 , 9.75)
[304](6.0 , 18.0 , 11.5)
[305](6.0 , 18.0 , 10.75)
[306](4.0 , 12.0 , 8.25)
[307](4.0 , 10.0 , 7.25)
[308](6.0 , 14.0 , 8.75)
[309](6.0 , 16.0 , 10.25)
[310](6.0 , 16.0 , 9.5)
[311](6.0 , 16.0 , 7.75)
[312](6.0 , 16.0 , 8.25)
[313](6.0 , 16.0 , 8.5)
[314](6.0 , 16.0 , 8.75)
[315](6.0 , 16.0 , 10.0)
[316](6.0 , 16.0 , 10.75)
[317](6.0 , 14.0 , 10.75)
[318](6.0 , 12.0 , 9.5)
[319](6.0 , 12.0 , 9.25)
[320](6.0 , 12.0 , 9.0)
[321](6.0 , 12.0 , 9.5)
[322](8.0 , 12.0 , 10.25)
[323](8.0 , 12.0 , 9.75)
[324](8.0 , 18.0 , 10.75)
[325](8.0 , 18.0 , 11.0)
[326](8.0 , 18.0 , 10.5)
[327](8.0 , 18.0 , 11.75)
[328](8.0 , 18.0 , 11.5)
[329](8.0 , 18.0 , 11.5)
[330](8.0 , 12.0 , 10.25)
[331](6.0 , 14.0 , 10.25)
[332](10.0 , 16.0 , 11.5)
[333](6.0 , 14.0 , 10.0)
[334](6.0 , 10.0 , 9.5)
[335](6.0 , 10.0 , 9.25)
[336](6.0 , 10.0 , 9.5)
[337](6.0 , 12.0 , 9.0)
[338](6.0 , 14.0 , 10.0)
[339](6.0 , 14.0 , 10.25)
[340](8.0 , 16.0 , 11.0)
[341](8.0 , 16.0 , 12.25)
[342](8.0 , 16.0 , 13.0)
[343](6.0 , 16.0 , 12.0)
[344](6.0 , 16.0 , 11.25)

[345](6.0 , 16.0 , 10.75)
[346](8.0 , 16.0 , 11.0)
[347](8.0 , 18.0 , 10.75)
[348](8.0 , 16.0 , 10.25)
[349](8.0 , 16.0 , 10.5)
[350](8.0 , 16.0 , 9.75)
[351](8.0 , 10.0 , 8.75)
[352](8.0 , 10.0 , 8.25)
[353](8.0 , 10.0 , 8.25)
[354](8.0 , 10.0 , 8.5)
[355](8.0 , 14.0 , 9.0)
[356](8.0 , 14.0 , 9.5)
[357](8.0 , 10.0 , 8.25)
[358](8.0 , 14.0 , 9.25)
[359](8.0 , 12.0 , 9.0)
[360](6.0 , 10.0 , 8.25)
[361](6.0 , 12.0 , 8.75)
[362](8.0 , 14.0 , 11.0)
[363](8.0 , 14.0 , 11.0)
[364](10.0 , 16.0 , 12.25)
[365](6.0 , 14.0 , 11.5)
[366](10.0 , 14.0 , 12.0)
[367](8.0 , 14.0 , 11.25)
[368](8.0 , 18.0 , 12.0)
[369](8.0 , 18.0 , 12.5)
[370](8.0 , 16.0 , 11.5)
[371](8.0 , 16.0 , 11.0)
[372](8.0 , 18.0 , 11.0)
[373](8.0 , 18.0 , 11.75)
[374](8.0 , 16.0 , 12.0)
[375](8.0 , 16.0 , 12.25)
[376](8.0 , 16.0 , 12.5)
[377](8.0 , 16.0 , 12.5)
[378](8.0 , 16.0 , 12.0)
[379](10.0 , 14.0 , 11.75)
[380](10.0 , 14.0 , 11.75)
[381](10.0 , 16.0 , 12.5)
[382](10.0 , 14.0 , 12.25)
[383](10.0 , 14.0 , 12.5)
[384](10.0 , 14.0 , 11.75)
[385](10.0 , 14.0 , 11.25)
[386](10.0 , 14.0 , 11.5)
[387](10.0 , 20.0 , 12.5)
[388](8.0 , 20.0 , 12.25)
[389](6.0 , 20.0 , 12.75)
[390](10.0 , 20.0 , 13.5)
[391](8.0 , 20.0 , 13.75)
[392](8.0 , 20.0 , 13.0)
[393](8.0 , 20.0 , 12.5)
[394](8.0 , 20.0 , 12.25)
[395](8.0 , 14.0 , 11.75)
[396](10.0 , 22.0 , 13.25)
[397](6.0 , 14.0 , 11.5)
[398](6.0 , 14.0 , 10.25)

[399](8.0 , 16.0 , 11.0)
[400](8.0 , 14.0 , 10.5)
[401](8.0 , 14.0 , 9.75)
[402](8.0 , 14.0 , 9.75)
[403](8.0 , 14.0 , 9.5)
[404](6.0 , 16.0 , 9.25)
[405](6.0 , 16.0 , 9.0)
[406](6.0 , 16.0 , 8.75)
[407](6.0 , 16.0 , 9.0)
[408](6.0 , 12.0 , 9.0)
[409](4.0 , 12.0 , 8.0)
[410](4.0 , 12.0 , 7.75)
[411](4.0 , 12.0 , 7.75)
[412](4.0 , 8.0 , 7.0)
[413](4.0 , 8.0 , 6.75)
[414](4.0 , 10.0 , 7.0)
[415](4.0 , 14.0 , 8.25)
[416](4.0 , 14.0 , 7.75)
[417](4.0 , 10.0 , 6.75)
[418](6.0 , 10.0 , 7.5)
[419](6.0 , 10.0 , 7.75)
[420](8.0 , 12.0 , 9.5)
[421](6.0 , 12.0 , 9.0)
[422](6.0 , 12.0 , 9.5)
[423](8.0 , 18.0 , 11.25)
[424](8.0 , 12.0 , 10.25)
[425](8.0 , 14.0 , 11.25)
[426](8.0 , 14.0 , 11.25)
[427](8.0 , 14.0 , 11.0)
[428](6.0 , 18.0 , 11.0)
[429](8.0 , 18.0 , 11.75)
[430](6.0 , 12.0 , 10.75)
[431](6.0 , 14.0 , 11.25)
[432](6.0 , 22.0 , 12.5)
[433](6.0 , 22.0 , 11.75)
[434](10.0 , 22.0 , 13.0)
[435](10.0 , 22.0 , 12.5)
[436](10.0 , 14.0 , 11.0)
[437](8.0 , 14.0 , 11.0)
[438](8.0 , 14.0 , 10.75)
[439](8.0 , 14.0 , 11.0)
[440](8.0 , 14.0 , 10.5)
[441](8.0 , 14.0 , 10.5)
[442](8.0 , 14.0 , 11.0)
[443](8.0 , 14.0 , 11.0)
[444](10.0 , 16.0 , 12.25)
[445](8.0 , 14.0 , 11.25)
[446](8.0 , 14.0 , 10.75)
[447](8.0 , 12.0 , 10.0)
[448](8.0 , 12.0 , 11.0)
[449](8.0 , 16.0 , 11.75)
[450](8.0 , 14.0 , 11.5)
[451](8.0 , 14.0 , 10.25)
[452](6.0 , 14.0 , 9.75)

[453](6.0 , 14.0 , 10.0)
[454](6.0 , 14.0 , 9.5)
[455](4.0 , 14.0 , 8.5)
[456](4.0 , 10.0 , 7.5)
[457](4.0 , 10.0 , 7.0)
[458](4.0 , 10.0 , 7.0)
[459](4.0 , 16.0 , 8.75)
[460](4.0 , 16.0 , 9.5)
[461](4.0 , 10.0 , 8.75)
[462](6.0 , 12.0 , 10.0)
[463](6.0 , 14.0 , 10.5)
[464](8.0 , 16.0 , 11.0)
[465](8.0 , 12.0 , 10.0)
[466](6.0 , 14.0 , 9.75)
[467](6.0 , 18.0 , 10.75)
[468](8.0 , 18.0 , 12.0)
[469](8.0 , 14.0 , 12.25)
[470](8.0 , 14.0 , 11.75)
[471](12.0 , 14.0 , 12.25)
[472](12.0 , 14.0 , 12.5)
[473](12.0 , 18.0 , 13.25)
[474](10.0 , 18.0 , 13.0)
[475](12.0 , 18.0 , 13.5)
[476](12.0 , 18.0 , 13.75)
[477](12.0 , 18.0 , 14.0)
[478](8.0 , 16.0 , 12.75)
[479](8.0 , 20.0 , 12.25)
[480](8.0 , 14.0 , 11.25)
[481](8.0 , 14.0 , 11.5)
[482](8.0 , 18.0 , 12.75)
[483](8.0 , 18.0 , 12.0)
[484](8.0 , 18.0 , 11.25)
[485](8.0 , 18.0 , 10.75)
[486](6.0 , 18.0 , 10.75)
[487](6.0 , 12.0 , 10.25)
[488](6.0 , 12.0 , 8.75)
[489](6.0 , 14.0 , 9.75)
[490](4.0 , 12.0 , 9.25)
[491](4.0 , 12.0 , 8.25)
[492](6.0 , 12.0 , 10.25)
[493](6.0 , 14.0 , 10.5)
[494](6.0 , 14.0 , 10.75)
[495](6.0 , 12.0 , 9.75)
[496](6.0 , 14.0 , 10.25)
[497](6.0 , 14.0 , 10.0)
[498](8.0 , 14.0 , 10.5)
[499](8.0 , 20.0 , 12.25)
[500](6.0 , 20.0 , 11.5)
[501](6.0 , 20.0 , 12.0)
[502](8.0 , 16.0 , 11.25)
[503](8.0 , 14.0 , 11.0)
[504](8.0 , 14.0 , 11.75)
[505](8.0 , 14.0 , 12.0)
[506](8.0 , 14.0 , 12.5)

[507](8.0 , 16.0 , 13.0)
[508](8.0 , 16.0 , 13.25)
[509](8.0 , 16.0 , 13.5)
[510](4.0 , 20.0 , 12.0)
[511](4.0 , 20.0 , 11.75)
[512](8.0 , 20.0 , 11.75)
[513](8.0 , 20.0 , 12.5)
[514](8.0 , 20.0 , 13.0)
[515](8.0 , 20.0 , 14.0)
[516](8.0 , 20.0 , 12.0)
[517](8.0 , 20.0 , 12.25)
[518](8.0 , 20.0 , 11.0)
[519](8.0 , 20.0 , 10.5)
[520](8.0 , 12.0 , 8.75)
[521](8.0 , 10.0 , 8.75)
[522](8.0 , 12.0 , 9.0)
[523](6.0 , 12.0 , 8.75)
[524](6.0 , 10.0 , 8.0)
[525](6.0 , 10.0 , 8.0)
[526](6.0 , 10.0 , 8.25)
[527](8.0 , 10.0 , 8.5)
[528](6.0 , 10.0 , 8.25)
[529](6.0 , 10.0 , 8.25)
[530](6.0 , 10.0 , 8.5)
[531](6.0 , 14.0 , 9.5)
[532](6.0 , 18.0 , 11.0)
[533](8.0 , 18.0 , 12.25)
[534](8.0 , 20.0 , 14.25)
[535](6.0 , 20.0 , 14.0)
[536](8.0 , 20.0 , 14.25)
[537](10.0 , 20.0 , 13.75)
[538](12.0 , 20.0 , 15.0)
[539](12.0 , 20.0 , 15.0)
[540](12.0 , 18.0 , 13.75)
[541](10.0 , 18.0 , 12.75)
[542](10.0 , 14.0 , 12.0)
[543](8.0 , 16.0 , 12.25)
[544](8.0 , 14.0 , 11.75)
[545](8.0 , 16.0 , 12.25)
[546](12.0 , 16.0 , 12.75)
[547](12.0 , 16.0 , 12.75)
[548](12.0 , 14.0 , 12.75)
[549](12.0 , 14.0 , 12.75)
[550](10.0 , 14.0 , 12.5)
[551](10.0 , 14.0 , 12.0)
[552](10.0 , 16.0 , 12.25)
[553](8.0 , 20.0 , 12.75)
[554](8.0 , 20.0 , 11.5)
[555](8.0 , 16.0 , 11.75)
[556](8.0 , 16.0 , 11.75)
[557](8.0 , 16.0 , 11.75)
[558](6.0 , 16.0 , 11.5)
[559](8.0 , 16.0 , 13.0)
[560](8.0 , 22.0 , 14.5)

[561](8.0 , 20.0 , 14.0)
[562](8.0 , 16.0 , 13.0)
[563](8.0 , 16.0 , 12.75)
[564](8.0 , 16.0 , 12.0)
[565](8.0 , 16.0 , 11.75)
[566](8.0 , 14.0 , 10.75)
[567](8.0 , 14.0 , 11.0)
[568](8.0 , 14.0 , 11.5)
[569](8.0 , 14.0 , 12.25)
[570](6.0 , 16.0 , 11.0)
[571](6.0 , 16.0 , 9.5)
[572](6.0 , 16.0 , 10.5)
[573](6.0 , 16.0 , 10.0)
[574](4.0 , 16.0 , 9.0)
[575](4.0 , 14.0 , 7.5)
[576](4.0 , 14.0 , 7.75)
[577](4.0 , 14.0 , 7.5)
[578](4.0 , 12.0 , 7.5)
[579](4.0 , 12.0 , 6.75)
[580](4.0 , 10.0 , 6.0)
[581](4.0 , 14.0 , 6.25)
[582](4.0 , 14.0 , 7.0)
[583](4.0 , 8.0 , 6.0)
[584](4.0 , 12.0 , 6.75)
[585](4.0 , 12.0 , 6.75)
[586](4.0 , 8.0 , 6.5)
[587](6.0 , 14.0 , 8.0)
[588](4.0 , 14.0 , 7.75)
[589](4.0 , 14.0 , 7.25)
[590](4.0 , 14.0 , 8.0)
[591](4.0 , 14.0 , 7.5)
[592](4.0 , 14.0 , 7.25)
[593](4.0 , 14.0 , 7.75)
[594](4.0 , 14.0 , 8.75)
[595](4.0 , 14.0 , 8.5)
[596](4.0 , 20.0 , 11.0)
[597](4.0 , 20.0 , 11.25)
[598](6.0 , 20.0 , 11.0)
[599](0.0 , 20.0 , 9.25)

Answer : ([1]:5 [2]:1 [3]:8 [4]:6 [5]:3 [6]:7 [7]:2 [8]:4)

برای حل مسیله ۵۹۹ نسل تولید شده و در آخر میبینید که به بهترین حالت ممکن رسیده است.

تابع هدف: خروجی این تابع برابر با تعداد conflict های موجود است یعنی چه تعداد از وزیر ها همدیگر را میتوانند بزنند. بعضی از حالت ها ۲بار حساب شده برای بهینه بودن الگوریتم.

```
@Override
public double f(State state_inter) {
    State8Queen state = (State8Queen)state_inter;

    int conflicts = 0;
    for (int i = 1; i <= 8 ; i++) {
        for (int j = 1; j <= 8; j++) {
            if (j!=i && (state.x[j] == state.x[i] || (Math.abs(state.x[j] - state.x[i]) == Math.abs(j - i)) ) ) {
                conflicts++;
            }
        }
    }
    return conflicts;
}
```

جهش:
یکی از وزیر ها به صورت رندوم یک ستون رندوم را اختیار میکند.

```
@Override
public State mutation(Problem problem) {
    State8Queen mutatedChild = new State8Queen(x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8]);
    int rand1 = (int) (Math.random() * 8) + 1;
    int rand2 = (int) (Math.random() * 8) + 1;
    mutatedChild.x[rand1] = rand2;
    return mutatedChild;
}
```

بازترکیبی:
۴ وزیراول استیت پدر و ۴ وزیر دوم استیت مادر برای فرزند انتخاب شده است.

```
@Override
public State crossOver(Problem problem_inter, State mother_inter) {
    Problem8Queen problem = (Problem8Queen)problem_inter;
    State8Queen mother = (State8Queen)mother_inter;

    State8Queen child = new State8Queen(this.x[1], this.x[2], this.x[3], this.x[4], mother.x[5], mother.x[6], mother.x[7], mother.x[8]);
    return child;
}
```

نحوه ی انتخاب والدین:

همانطور که از کد مشخص است هر استیت که ارزش بیشتری داشته باشد شانس بیشتری برای والد شدن در اختیار دارد که این مسیله با آرایه ی probailityArray هندل شده است. یه رندوم روی این آرایه زده میشود و پدر و مادر تعیین میشوند.

probabilityArray چگونه ساخته میشود؟

به هر استیت یک ارزش داده میشود که برای بهینه سازی از میانگین دور شده است سپس به تعداد ارزش این استیت این استیت به آرایه ی probabilityArray اضافه میشود پس اگر روی این آرایه رندوم زده شود شانس بیشتری برای انتخاب شدن دارد.

```
private ArrayList<State> makesChildren(Problem problem, ArrayList<State> population, int m,
        double mutationPropability) { // m is number of children , select 2m parent from n (that can be
                                      // repetitive(one guy can be father of more than 1 child))

    ArrayList<Integer> propabilityArray = new ArrayList<Integer>();
    for (int i = 0; i < population.size(); i++) {
        double value = problem.maximumPossibleF() - problem.f(population.get(i));
        if (value < problem.maximumPossibleF() - problem.borderF()) {
            value /= 2;
        } else {
            value *= 2;
        }
        for (int j = 0; j < value; j++) {
            propabilityArray.add(i);
        }
    }

    ArrayList<State> children = new ArrayList<State>();
    double p = 0.1; // if a state is a repetitive add to population with this p
    while (children.size() < m) { // with propability select who has lowest f
        int random1 = (int) (Math.random() * propabilityArray.size());
        int index1 = propabilityArray.get(random1);
        int random2 = (int) (Math.random() * propabilityArray.size());
        int index2 = propabilityArray.get(random2);
        State parent1 = population.get(index1);
        State parent2 = population.get(index2);
        State child = reProduce(problem, parent1, parent2, mutationPropability);
        if (!children.contains(child)) {
            boolean valid = true;
            for (int i = 0; i < children.size(); i++) {
                if (problem.equal(child, children.get(i))) {
                    valid = false;
                    break;
                }
            }

            if (valid) {
                children.add(child);
            } else if (Math.random() < p) {
                children.add(child);
            }
        }
    }
    return children;

}
```

نحوه انتخاب بازماندگان:
همانطور که از کد مشخص است هر استیت که ارزش بیشتری داشته باشد شانس بیشتری برای زنده ماندن در اختیار دارد که این مسیله با آرایه ی probaillityArray هندل شده است. یه رندوم روی این آرایه زده میشود وبازمانده ها تعیین میشوند.
مثل انتخاب والدین

```
private ArrayList<State> selectTheBests(Problem problem, ArrayList<State> population, ArrayList<State> children) {
    int n = population.size();

    ArrayList<State> survivors = new ArrayList<State>();

    ArrayList<Integer> propabilityArray = new ArrayList<Integer>();
    for (int i = 0; i < population.size(); i++) {
        double value = problem.maximumPossibleF() - problem.f(population.get(i));
        if (value < problem.maximumPossibleF() / 2) {
            value /= 2;
        } else {
            value *= 2;
        }
        for (int j = 0; j < value; j++) {
            propabilityArray.add(i);
        }
    }
    for (int i = 0; i < children.size(); i++) {
        double value = problem.maximumPossibleF() - problem.f(children.get(i));
        if (value < problem.maximumPossibleF() / 2) {
            value /= 2;
        } else {
            value *= 2;
        }
        for (int j = 0; j < value; j++) {
            propabilityArray.add(i + population.size());
        }
    }

    while (survivors.size() != population.size()) { // with propability select who has lowest f
        int random = (int) (Math.random() * propabilityArray.size());
        int index = propabilityArray.get(random);
        if (index < population.size()) {
            if (!survivors.contains(population.get(index))) {
                survivors.add(population.get(index));
            }
        } else { // is a index of children
            index = index - population.size();
            if (!survivors.contains(children.get(index))) {
                survivors.add(children.get(index));
            }
        }
    }

    return survivors;
```

(د

Number of Generations : 100

best , worst , average value in each generation :
[1](8.0 , 24.0 , 14.9)
[2](8.0 , 24.0 , 13.7)
[3](8.0 , 20.0 , 12.5)
[4](6.0 , 20.0 , 12.4)
[5](6.0 , 18.0 , 11.2)
[6](8.0 , 18.0 , 10.9)
[7](6.0 , 14.0 , 10.2)
[8](8.0 , 16.0 , 10.5)
[9](6.0 , 16.0 , 11.0)
[10](6.0 , 20.0 , 10.7)
[11](6.0 , 18.0 , 11.0)
[12](8.0 , 14.0 , 11.0)
[13](8.0 , 18.0 , 11.9)
[14](8.0 , 24.0 , 12.7)
[15](8.0 , 18.0 , 12.3)
[16](6.0 , 20.0 , 11.1)
[17](6.0 , 18.0 , 11.7)
[18](6.0 , 18.0 , 12.2)
[19](8.0 , 18.0 , 11.7)
[20](8.0 , 14.0 , 11.4)
[21](8.0 , 24.0 , 13.0)
[22](10.0 , 22.0 , 13.5)
[23](8.0 , 18.0 , 13.3)
[24](10.0 , 22.0 , 14.2)
[25](8.0 , 20.0 , 12.5)
[26](6.0 , 16.0 , 11.3)
[27](6.0 , 18.0 , 11.3)
[28](6.0 , 16.0 , 10.1)
[29](6.0 , 18.0 , 10.3)
[30](6.0 , 14.0 , 10.1)
[31](6.0 , 20.0 , 10.3)
[32](6.0 , 16.0 , 10.3)
[33](6.0 , 16.0 , 10.3)
[34](8.0 , 16.0 , 11.4)
[35](6.0 , 16.0 , 11.0)
[36](8.0 , 22.0 , 11.6)
[37](6.0 , 18.0 , 12.3)
[38](6.0 , 20.0 , 12.6)
[39](8.0 , 18.0 , 12.7)
[40](8.0 , 18.0 , 12.1)
[41](8.0 , 20.0 , 12.6)
[42](6.0 , 18.0 , 11.7)
[43](6.0 , 20.0 , 11.2)
[44](8.0 , 24.0 , 12.2)

[45](6.0 , 24.0 , 12.3)
[46](6.0 , 20.0 , 11.7)
[47](6.0 , 20.0 , 11.8)
[48](4.0 , 18.0 , 10.9)
[49](6.0 , 20.0 , 12.5)
[50](6.0 , 22.0 , 12.5)
[51](6.0 , 20.0 , 11.7)
[52](10.0 , 20.0 , 11.8)
[53](8.0 , 16.0 , 11.6)
[54](6.0 , 18.0 , 10.5)
[55](6.0 , 16.0 , 11.0)
[56](6.0 , 16.0 , 10.9)
[57](6.0 , 16.0 , 10.9)
[58](6.0 , 18.0 , 11.5)
[59](6.0 , 16.0 , 11.1)
[60](8.0 , 18.0 , 12.0)
[61](8.0 , 20.0 , 11.9)
[62](10.0 , 18.0 , 12.4)
[63](10.0 , 18.0 , 13.4)
[64](8.0 , 18.0 , 13.1)
[65](8.0 , 18.0 , 12.0)
[66](8.0 , 18.0 , 12.8)
[67](8.0 , 20.0 , 13.3)
[68](6.0 , 22.0 , 13.2)
[69](6.0 , 20.0 , 11.8)
[70](8.0 , 18.0 , 12.6)
[71](6.0 , 24.0 , 13.4)
[72](8.0 , 24.0 , 13.1)
[73](10.0 , 16.0 , 11.8)
[74](8.0 , 22.0 , 11.6)
[75](6.0 , 22.0 , 12.4)
[76](8.0 , 22.0 , 12.1)
[77](6.0 , 16.0 , 10.7)
[78](6.0 , 20.0 , 12.3)
[79](6.0 , 26.0 , 12.1)
[80](6.0 , 16.0 , 11.1)
[81](8.0 , 16.0 , 9.8)
[82](6.0 , 16.0 , 10.8)
[83](8.0 , 16.0 , 11.5)
[84](6.0 , 16.0 , 10.3)
[85](4.0 , 18.0 , 9.7)
[86](6.0 , 16.0 , 10.2)
[87](4.0 , 18.0 , 10.5)
[88](6.0 , 16.0 , 10.9)
[89](4.0 , 14.0 , 9.4)
[90](6.0 , 18.0 , 10.7)
[91](2.0 , 16.0 , 10.2)
[92](6.0 , 14.0 , 10.4)
[93](6.0 , 18.0 , 10.5)
[94](6.0 , 18.0 , 10.8)
[95](6.0 , 18.0 , 10.7)
[96](4.0 , 14.0 , 8.7)
[97](6.0 , 14.0 , 9.9)
[98](6.0 , 14.0 , 9.4)

[99](6.0 , 18.0 , 11.1)
[100](6.0 , 18.0 , 10.7)

Answer : ([1]:5 [2]:3 [3]:1 [4]:7 [5]:2 [6]:8 [7]:6 [8]:1)

جمعیت ۲۰ نفر و تعداد تولید نسل ۱۰۰ و تعداد فرزند در هر مرحله ۵۰
میبینید که به جواب بهینه نرسیده ایم
چرا که بهترین جواب دارای f=2 است.