

محمد رضا جلی ۹۴۳۱۰۳۵
تمرین دوم برنامه نویسی هوش مصنوعی

نحوه ی مدلسازی :

از یک کلاس search که تمامی حالات مختلف search در آن به صورت تابع قرار داده شده استفاده شده. این توابع به هیچ وجه نباید به نحوه ی پیاده سازی کلاس های action , state , problem ارتباط داشته باشند.

سپس هر مسئله را با استفاده از کلاس های action , state , problem مدل شده است که هر کلاس های action , state از کلاس های پدرشان ارث برده اند که کد دسته بندی شود و کار ها مشخص شود . کلاس problem نیز از یک interface , ایملمنت کرده است.

در ضمن در تمامی الگوریتم های سرچ فرض شده که استتیت بهتر استتیتی است که مقدار f کمتری داشته باشد. و در تمامی مسایل استتیتی که $f=0$ داشته باید بهترین استتیت ممکن یا همان استتیت goal است.

مسئله ی ۱ : رنگ آمیزی گراف

مدل hill climbing معمولی :

```
Problem2.java State2.java Action2.java Main.java
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem1();
4         Search search = new Search();
5         search.hillClimbing(problem);
6     }
7 }
```

Problems @ Javadoc Declaration Console

```
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_101-b12/Contents/Home/bin/java
Number of maximum colors? :
3
Number of nodes (n) :
6
insert your graph as a matrix :
0 1 1 1 1 0
1 0 1 0 0 0
1 1 0 1 0 0
1 0 1 0 1 0
1 0 0 1 0 0
0 0 0 0 0 0
Number of observed nodes : 25
Number of extended nodes : 1
Path states : ( 0 2 2 1 2 0 ) ( 0 1 2 1 2 0 )
Path actions : ( node:1 , color:1 )
Path cost : 1.0
Final state worth : 0.0
```

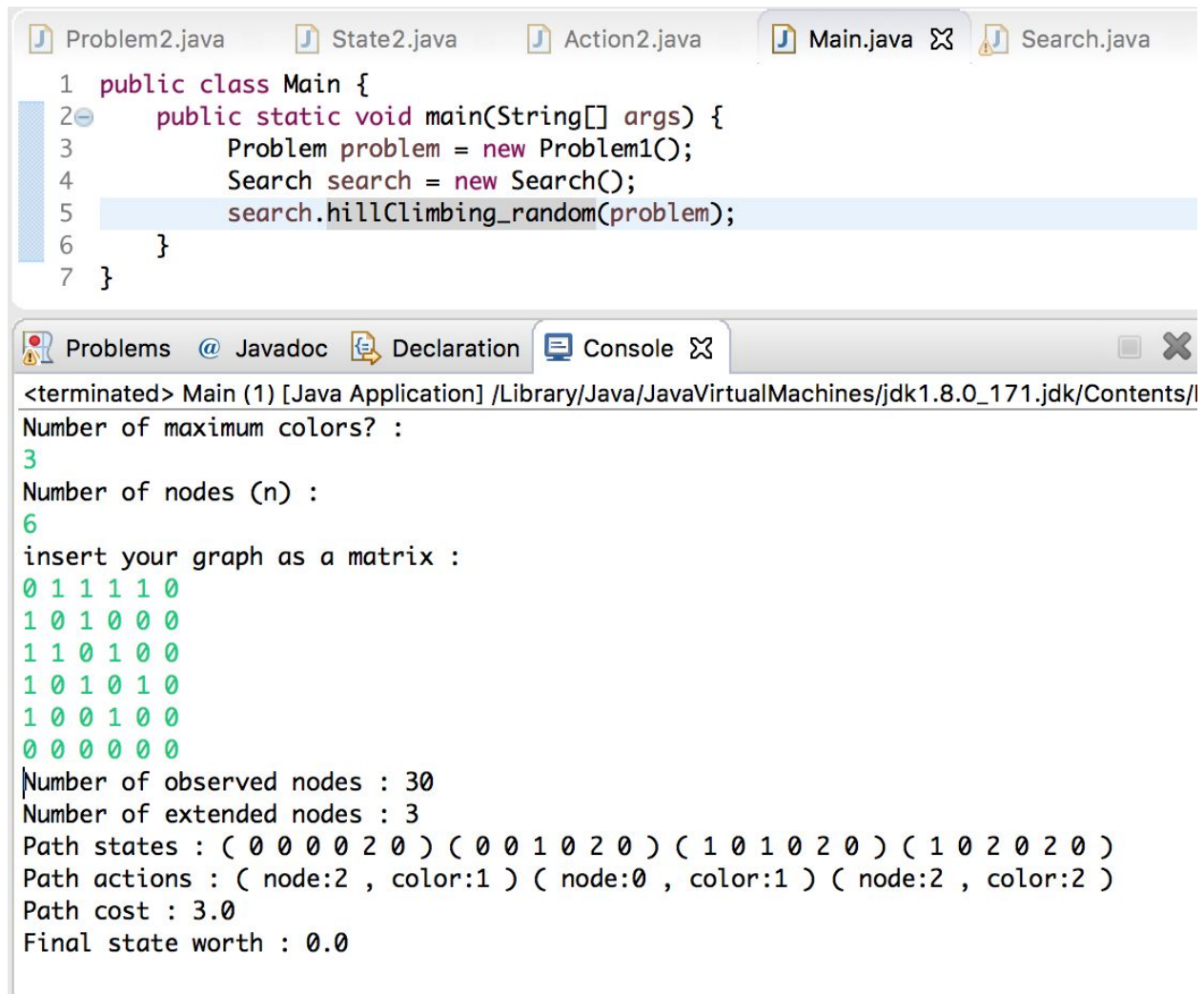
: hill climbing first choice مدل

```
Problem2.java State2.java Action2.java Main.java Search.java
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem1();
4         Search search = new Search();
5         search.hillClimbing_firstChoice(problem);
6     }
7 }
```

Problems @ Javadoc Declaration Console

<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/H
Number of maximum colors? :
3
Number of nodes (n) :
6
insert your graph as a matrix :
0 1 1 1 1 0
1 0 1 0 0 0
1 1 0 1 0 0
1 0 1 0 1 0
1 0 0 1 0 0
0 0 0 0 0 0
Number of observed nodes : 27
Number of extended nodes : 3
Path states : (0 0 0 2 1 1) (1 0 0 2 1 1) (1 2 0 2 1 1) (1 2 0 2 0 1)
Path actions : (node:0 , color:1) (node:1 , color:2) (node:4 , color:0)
Path cost : 3.0
Final state worth : 0.0

مدل hill climbing random :



The screenshot shows an IDE with several tabs: Problem2.java, State2.java, Action2.java, Main.java, and Search.java. The Main.java tab is active, displaying the following code:

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Problem problem = new Problem1();  
4         Search search = new Search();  
5         search.hillClimbing_random(problem);  
6     }  
7 }
```

Below the code editor is a console window with the following output:

```
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/  
Number of maximum colors? :  
3  
Number of nodes (n) :  
6  
insert your graph as a matrix :  
0 1 1 1 1 0  
1 0 1 0 0 0  
1 1 0 1 0 0  
1 0 1 0 1 0  
1 0 0 1 0 0  
0 0 0 0 0 0  
Number of observed nodes : 30  
Number of extended nodes : 3  
Path states : ( 0 0 0 0 2 0 ) ( 0 0 1 0 2 0 ) ( 1 0 1 0 2 0 ) ( 1 0 2 0 2 0 )  
Path actions : ( node:2 , color:1 ) ( node:0 , color:1 ) ( node:2 , color:2 )  
Path cost : 3.0  
Final state worth : 0.0
```

مدل hill climbing random restart:

```
Problem2.java State2.java Action2.java Main.java Search.java
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem1();
4         Search search = new Search();
5         search.hillClimbing_randomRestart(problem);
6     }
7 }
```

Problems Javadoc Declaration Console

```
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home
Number of maximum colors? :
3
Number of nodes (n) :
6
insert your graph as a matrix :
0 1 1 1 1 0
1 0 1 0 0 0
1 1 0 1 0 0
1 0 1 0 1 0
1 0 0 1 0 0
0 0 0 0 0 0
Number of observed nodes : 37
Number of extended nodes : 2
Path states : ( 0 1 0 0 2 1 ) ( 0 1 2 0 2 1 ) ( 0 1 2 1 2 1 )
Path actions : ( node:2 , color:2 ) ( node:3 , color:1 )
Path cost : 2.0
Final state worth : 0.0
```

توضیحات مسئله ۱:

با توجه به سادگی مسئله و کم بودن حالات مینیمم هر ۴ نوع الگوریتم hill climbing در بیشتر موارد بهترین استتیت ممکن را پیدا میکند.

الگوریتم first choice , random تعداد نود مشاهده‌ی شده‌ی کمتری دارند و قاعدتا دیرتر نیز به جواب میرسند همچنین احتمال گیر کردن در مینیمم محلی در آنها کمتر است.

احتمال گیر کردن در مینیمم محلی برای hill climbing معمولی از تمام موارد دیگر بیشتر است و همچنین بعد از hill climbing random restart بیشترین تعداد نود مشاهده شده را نیز دارد.

مسئله ۲: جدول حروف

الگوریتم simulated annealing با استفاده از تابع سرد کردن خطی

```
1 public class Main {  
2     public static void main(String[] args) {  
3         Problem problem = new Problem2();  
4         Search search = new Search();  
5         search.simulatedAnnealing(problem); //linear  
6     }  
7 }
```

Problems @ Javadoc Declaration Console

<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_1

number of rows:
4
number of columns:
3
Table:
a p t
m l b
k l o
u o c
number of words:
4
Words:
cool cat talk go
Number of observed nodes : 2772
Number of extended nodes : 1879
Best Answer :
o o l
c p b
a l u
t k m

Path cost : 0.0
Final state worth : 3.0

الگوریتم simulated annealing با استفاده از تابع سرد کردن نمایشی

```
Problem2.java State2.java Action2.java Main.java Search
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem2();
4         Search search = new Search();
5         search.simulatedAnealing(problem); //exponential
6     }
7 }
```

Problems @ Javadoc Declaration Console

<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/C

number of rows:
4
number of columns:
3
Table:
a p t
m l b
k l o
u o c
number of words:
4
Words:
cool talk cat go
Number of observed nodes : 2126
Number of extended nodes : 643
Best Answer :
m b u
p l o
k l o
t a c

Path cost : 0.0
Final state worth : 3.0

الگوریتم simulated annealing با استفاده از تابع سرد کردن لگاریتمی

```
Problem2.java State2.java Action2.java Main.java S
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem2();
4         Search search = new Search();
5         search.simulatedAnnealing(problem); //logarithmic
6     }
7 }
```

Problems @ Javadoc Declaration Console

<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171

number of rows:
4
number of columns:
3
Table:
a p t
m l b
k l o
u o c
number of words:
4
Words:
cool talk cat go
Number of observed nodes : 30359
Number of extended nodes : 9142
Best Answer :
p t b
c a l
o o k
u l m

Path cost : 0.0
Final state worth : 3.0

مقایسه ی حالات مختلف الگوریتم simulated annealing

خطی : $T = T - 0.005$

از ۱۰ اجرای آخر ۶ اجرا بهترین جواب ممکن را داد.

نمایی : $T = T * (1 - 0.005)$

از ۱۰ اجرای آخر ۳ اجرا بهترین جواب ممکن را داد.

لگاریتمی : $T = (10 + 0.0) / \text{Math.log}(t + 10)$;

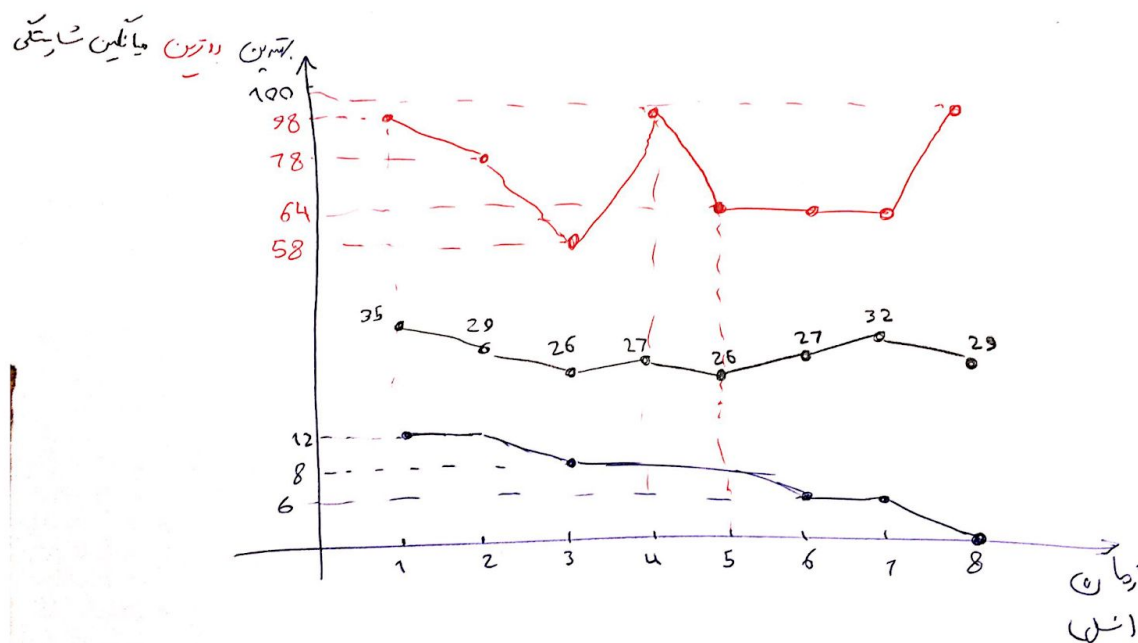
از ۱۰ اجرای آخر هر ۱۰ اجرا بهترین جواب ممکن را داد.

همان طور که مشخص است تابع سرد کننده ی لگاریتمی تضمین بالاتری برای رسیدن به جواب بهینه میدهد و بنابراین تابع بهتری است.

از طرف دیگر تابع نمایی سریع تر به جواب میرسد. و تعداد نود مشاهده ی شده ی کمتری دارد.

مسئله ی ۳ : کیبورد سازی

الف) نمودار بهترین بدترین میانگین هر نسل در طول زمان:



ب) تاثیر کاهش یا افزایش احتمال جهش:

هر چه احتمال جهش بیشتر باشد با تعداد نسل های کمتری به جواب بهینه میرسیم.

احتمال ۰.۱ : ۹۶ نسل

```
Problem2.java State2.java Action2.java Main.java Search.java
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem3();
4         Search search = new Search();
5         search.geneticSearch(problem); //mutation probability = 0.1
6     }
7 }
```

```
Problems Javadoc Declaration Console
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home
[65]( 12.0 , 48.0 , 22.692307692307693 )
[66]( 12.0 , 48.0 , 21.76923076923077 )
[67]( 12.0 , 24.0 , 20.923076923076923 )
[68]( 12.0 , 24.0 , 20.76923076923077 )
[69]( 12.0 , 100.0 , 24.307692307692307 )
[70]( 12.0 , 100.0 , 24.0 )
[71]( 16.0 , 100.0 , 23.46153846153846 )
[72]( 20.0 , 100.0 , 23.46153846153846 )
[73]( 20.0 , 98.0 , 23.384615384615383 )
[74]( 20.0 , 98.0 , 23.384615384615383 )
[75]( 20.0 , 54.0 , 22.076923076923077 )
[76]( 20.0 , 54.0 , 22.076923076923077 )
[77]( 20.0 , 24.0 , 20.46153846153846 )
[78]( 20.0 , 24.0 , 21.23076923076923 )
[79]( 20.0 , 24.0 , 21.23076923076923 )
[80]( 20.0 , 24.0 , 20.923076923076923 )
[81]( 20.0 , 24.0 , 21.23076923076923 )
[82]( 20.0 , 48.0 , 22.153846153846153 )
[83]( 20.0 , 24.0 , 21.076923076923077 )
[84]( 20.0 , 24.0 , 21.076923076923077 )
[85]( 12.0 , 64.0 , 22.46153846153846 )
[86]( 16.0 , 64.0 , 22.307692307692307 )
[87]( 20.0 , 64.0 , 23.384615384615383 )
[88]( 20.0 , 52.0 , 21.692307692307693 )
[89]( 16.0 , 64.0 , 22.0 )
[90]( 16.0 , 52.0 , 21.846153846153847 )
[91]( 16.0 , 52.0 , 22.307692307692307 )
[92]( 16.0 , 52.0 , 22.46153846153846 )
[93]( 16.0 , 52.0 , 22.615384615384617 )
[94]( 20.0 , 70.0 , 22.692307692307693 )
[95]( 20.0 , 70.0 , 23.692307692307693 )
[96]( 0.0 , 54.0 , 22.076923076923077 )

Answer : ( a b c d f g h s m o q r v , e i j k n p l t u w x y z )
```

احتمال ۰.۵ : ۸ نسل

```
Problem2.java State2.java Action2.java Main.java Search.j
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem3();
4         Search search = new Search();
5         search.geneticSearch(problem); //mutation probability = 0.5
6     }
7 }
```

```
Problems Javadoc Declaration Console
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Co
Number of Generations : 8

best , worst , average value in each generation :
[1]( 12.0 , 98.0 , 35.61538461538461 )
[2]( 12.0 , 78.0 , 29.615384615384617 )
[3]( 8.0 , 58.0 , 26.692307692307693 )
[4]( 8.0 , 100.0 , 27.46153846153846 )
[5]( 8.0 , 64.0 , 26.846153846153847 )
[6]( 6.0 , 64.0 , 27.846153846153847 )
[7]( 6.0 , 64.0 , 32.23076923076923 )
[8]( 0.0 , 100.0 , 29.384615384615383 )

Answer : ( a b c h k m o q r s u v , d e f g i j l n p t x y z )
```

احتمال ۱ : ۳ نسل

```

1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem3();
4         Search search = new Search();
5         search.geneticSearch(problem); //mutation propability = 1
6     }
7 }

```

<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Co
 Number of Generations : 3

best , worst , average value in each generation :
 [1](12.0 , 98.0 , 35.61538461538461)
 [2](12.0 , 108.0 , 39.69230769230769)
 [3](0.0 , 120.0 , 29.692307692307693)

Answer : (a b c e k o p t u v w x y , d f g h i j l m n s q r z)

ج) تاثیر افزایش یا کاهش تعداد برش های (تعداد فرزندان در هر نسل) هرچه تعداد ایجاد فرزندان بیشتر باشد تعداد نسل ها برای رسیدن به جواب کمتر میشود.

تعداد بچه های هر نسل : نصف جمعیت اولیه

```

1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem3();
4         Search search = new Search();
5         search.geneticSearch(problem); //number of children in each generation = population/2
6     }
7 }

```

<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home/bin/java (Tir 16, 1397
 Number of Generations : 2

best , worst , average value in each generation :
 [1](12.0 , 98.0 , 35.61538461538461)
 [2](0.0 , 78.0 , 33.23076923076923)

Answer : (a c e o q m t u v w x y z , b d f g h i j k l s n p r)

تعداد بچه های هر نسل : ۱/۸ جمعیت اولیه

```
Problem2.java State2.java Action2.java Main.java Search.java State1.java
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem3();
4         Search search = new Search();
5         search.geneticSearch(problem); //number of children in each generation = population/8
6     }
7 }
```

Problems @ Javadoc Declaration Console

<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home/bin/java (Tir 16, 1397)

```
[556]( 8.0 , 58.0 , 22.615384615384617 )
[557]( 10.0 , 46.0 , 21.384615384615383 )
[558]( 10.0 , 46.0 , 21.384615384615383 )
[559]( 10.0 , 46.0 , 20.615384615384617 )
[560]( 10.0 , 46.0 , 21.307692307692307 )
[561]( 10.0 , 72.0 , 25.076923076923077 )
[562]( 10.0 , 72.0 , 22.0 )
[563]( 10.0 , 78.0 , 23.307692307692307 )
[564]( 10.0 , 78.0 , 24.153846153846153 )
[565]( 10.0 , 72.0 , 21.76923076923077 )
[566]( 10.0 , 72.0 , 21.153846153846153 )
[567]( 10.0 , 34.0 , 18.23076923076923 )
[568]( 10.0 , 46.0 , 19.307692307692307 )
[569]( 10.0 , 46.0 , 19.846153846153847 )
[570]( 10.0 , 46.0 , 20.153846153846153 )
[571]( 10.0 , 46.0 , 21.0 )
[572]( 10.0 , 46.0 , 20.846153846153847 )
[573]( 10.0 , 58.0 , 20.615384615384617 )
[574]( 10.0 , 58.0 , 20.846153846153847 )
[575]( 10.0 , 58.0 , 20.846153846153847 )
[576]( 10.0 , 58.0 , 21.384615384615383 )
[577]( 10.0 , 58.0 , 21.615384615384617 )
[578]( 16.0 , 58.0 , 22.153846153846153 )
[579]( 16.0 , 24.0 , 20.76923076923077 )
[580]( 16.0 , 24.0 , 20.615384615384617 )
[581]( 16.0 , 24.0 , 20.384615384615383 )
[582]( 14.0 , 24.0 , 20.307692307692307 )
[583]( 16.0 , 24.0 , 20.615384615384617 )
[584]( 8.0 , 24.0 , 20.307692307692307 )
[585]( 8.0 , 72.0 , 22.615384615384617 )
[586]( 8.0 , 72.0 , 24.53846153846154 )
[587]( 8.0 , 72.0 , 22.153846153846153 )

Answer : ( a b c d i j k q r s h u w , e f g t l m n o p v x y z )
```

د)تاثیر اندازه ی جمعیت :

هر چه تعداد جمعیت اولیه بیشتر شود تعداد نسل ها کمتر میشود.

جمعیت اولیه : ۵ نفر -> ۲۵ نسل

```

Problem2.java State2.java Action2.java Main.java Search.java
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem3();
4         Search search = new Search();
5         search.geneticSearch(problem); //number of population = 5
6     }
7 }

Problems @ Javadoc Declaration Console
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home
Number of Generations : 25

best , worst , average value in each generation :
[1]( 12.0 , 98.0 , 35.61538461538461 )
[2]( 12.0 , 96.0 , 38.0 )
[3]( 12.0 , 78.0 , 34.61538461538461 )
[4]( 4.0 , 68.0 , 31.384615384615383 )
[5]( 4.0 , 68.0 , 28.46153846153846 )
[6]( 4.0 , 68.0 , 26.384615384615383 )
[7]( 4.0 , 68.0 , 26.76923076923077 )
[8]( 4.0 , 68.0 , 24.846153846153847 )
[9]( 4.0 , 68.0 , 24.076923076923077 )
[10]( 4.0 , 68.0 , 23.615384615384617 )
[11]( 4.0 , 68.0 , 26.76923076923077 )
[12]( 4.0 , 68.0 , 28.53846153846154 )
[13]( 4.0 , 68.0 , 29.076923076923077 )
[14]( 6.0 , 62.0 , 26.384615384615383 )
[15]( 6.0 , 62.0 , 25.153846153846153 )
[16]( 6.0 , 62.0 , 24.923076923076923 )
[17]( 6.0 , 78.0 , 26.923076923076923 )
[18]( 6.0 , 78.0 , 26.307692307692307 )
[19]( 6.0 , 78.0 , 30.615384615384617 )
[20]( 6.0 , 72.0 , 28.76923076923077 )
[21]( 6.0 , 42.0 , 22.307692307692307 )
[22]( 8.0 , 46.0 , 24.46153846153846 )
[23]( 4.0 , 86.0 , 24.615384615384617 )
[24]( 4.0 , 94.0 , 26.76923076923077 )
[25]( 0.0 , 68.0 , 25.53846153846154 )

Answer : ( a c e g k l m o p q t v w , b d f h i j n r s u x y z )

```

جمعیت اولیه : ۲۶ نفر -> ۶ نسل

```

Problem2.java State2.java Action2.java Main.java Search.java
1 public class Main {
2     public static void main(String[] args) {
3         Problem problem = new Problem3();
4         Search search = new Search();
5         search.geneticSearch(problem); //number of population = 26
6     }
7 }

Problems @ Javadoc Declaration Console
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_171.jdk/Contents/Home
Number of Generations : 6

best , worst , average value in each generation :
[1]( 12.0 , 98.0 , 35.61538461538461 )
[2]( 10.0 , 76.0 , 35.15384615384615 )
[3]( 10.0 , 98.0 , 34.53846153846154 )
[4]( 10.0 , 98.0 , 34.46153846153846 )
[5]( 6.0 , 98.0 , 31.23076923076923 )
[6]( 0.0 , 128.0 , 33.30769230769231 )

Answer : ( a f h v j l m o p q r s u , b c d e g k n t i w x y z )

```