Instituto Politécnico Nacional

Escuela Superior de Cómputo

Genetic Algorithms

Reporte Séptima Práctica:
“Algoritmos de cruza Pt2”

Grupo 3CM5
Romero Godinez José Enrique

Introducción:

Continuando con los algoritmos de curza expuestos en los apuntes del Dr. Coello nos encontramos con 5 aplicados a la representación entera de los individuos, estos algoritmos se utilizan en esta representación debido a que se basan en los diferentes alelos de los padres y como susutituirlos en la descendencia, caso que no podría ser utilizado en la representación binaria ya que sólo tienen dos tipos diferentes de alelos.

El primero de estos algoritmos es el denominado Order Crossover, propuesto por Davis, consiste en tomar una subcadena del primer padre y pasársela tal y como está al primer hijo, después tomamos el segundo padre y le quitamos la subcadena previamente tomada, para finalmente rellenar al primer hijo con los alelos restantes en el segundo padre y así alternando los padres .

El segundo algoritmo es el de Partially Mapped Crossover, el cual es muy similar a la cruza de dos puntos que discutimos la práctica pasada,propuesto por Goldberg y Lingle, funciona de la siguiente manera: se toman dos puntos y se parte a los padres en estas posiciones, a los hijos se les pasa la subcadena que se encuentra entre los dos puntos seleccionados, posteriormente a los padres se les “elimina” los alelos que se encuentran en las subcadenas tomadas, se rellenan los hijos con los alelos restantes de los padres y al final se hace un mapeo para completar a los hijos con los valores que falten sin repetición.

El tercer algoritmo es el de Position Based Crossover, propuesto por Syswerda, es una adaptación de la cruza uniforme, donde se toman n puntos de cruza y se copian los alelos del padre al hijo que esten en estos puntos, posteriormente se “eliminan” del segundo padre estos alelos y se rellena al hijo con los alelos sobrantes de esa eliminación, el proceso se repite alternando los padres para un nuevo hijo.

El cuarto algoritmo fue también propuesto por Sywerda y es una variación del anterior, es llamado Order Based Crossover y en este se intercambian los pasos del algoritmo anterior, primero se toman n puntos del padre 1, se proceden a eliminar estos alelos del padre 2, los alelos restantes son copiados al hijo que al final es rellenado con la secuencia de valores resultante de tomar n puntos del primer padre, y viceversa con el segundo hijo.

El ultimo algoritmo es el de Cycle Crossover, propuesto por Oliver, Smith y Holland,, tienen similitudes con el algoritmo de position based crossover en donde se toman valores de un padre y se rellena al hijo con los restantes del otro, este algoritmo consiste en encontrar un ciclo de elementos tales que seleccione n alelos diferentes de cada padre hasta que ya no sea posible agregar uno nuevo, copiar estos elementos al hijo, eliminar los elementos de uno de los padres y rellenar con los alelos restantes, es más complicado que todos los anteriores y este asegura una recombinación casi total de los alelos de cada descendiente.[1]

Objetivo:

Desarrollar un programa en lenguaje c que haga un experimento de cruza con los diferentes algoritmos expuestos en esta práctica.

Solución:

Se implementarán los algoritmos en lenguaje c, tomando en cuenta los pasos descritos en los apuntes del Dr. Coello para una población de 6 individuos de 8 cromosomas (caso particular, pueden ser n individuos de m cromosomas) .

Implementación:

Codigo en lenguaje C:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
void generaIndividuos(unsigned char**);
int checar(unsigned char*, int);
void orderCrossOver(unsigned char**);
void partiallyMapped(unsigned char**);
void positionBased(unsigned char**);
void orderBased(unsigned char**);
void cycleCrossover(unsigned char**);
void rellenar(unsigned char*);
void imprimir(unsigned char*);
int buscar(unsigned char*, int);
int main()
{
	unsigned char**individuos=calloc(6,sizeof(unsigned char*));
	int opc=0;
	generaIndividuos(individuos);
	printf("Selecciona una opcion de cruza\n1.-Order Crossover\n2.-Partially Mapped Crossover\n3.-Position Based Crossover\n4.-Order Based Crossover\n5.-Cycle Crossover\n");
	scanf("%d",&opc);
	switch(opc)
	{
		case 1:
			orderCrossOver(individuos);
```

```
                break;
                case 2:
                        partiallyMapped(individuos);
                break;
                case 3:
                        positionBased(individuos);
                break;
                case 4:
                        orderBased(individuos);
                break;
                case 5:

                break;
                default:
                        printf("Selecciona una opcíon valida\n");
                break;
        }
        free(individuos);
        return 0;
}
void generaIndividuos(unsigned char** individuos)
{
        long ltime;
        ltime=time(NULL);
        int a,stime;
        stime=(unsigned)ltime/2;
        srand(stime);
        unsigned char estaEnIndividuo=0,cromosoma;
        for(int i=0;i<6;i++)
```

```
	{
		individuos[i]=calloc(10,sizeof(unsigned char));
		for(unsigned char j=0;j<10;j++)
		{
			while(estaEnIndividuo!=1)
			{
				cromosoma=1+rand()%(10);
				estaEnIndividuo=checar(individuos[i],cromosoma);
			}
			individuos[i][j]=(unsigned char)cromosoma;
			estaEnIndividuo=0;
		}
	}
}
void orderCrossOver(unsigned char** individuos)
{
	long ltime;
	ltime=time(NULL);
	int stime,valor1,valor2,agregar=0;
	stime=(unsigned)ltime/2;
	srand(stime);
	unsigned char* hijo,*aux;
	for(unsigned char i=0;i<6;i++)
	{
		hijo=calloc(10,sizeof(unsigned char));
		valor1=rand()%10;
		valor2=rand()%(valor1-1);
		for(unsigned char j=10-valor2;j>=10-valor1;j--)
		{
```

```
        hijo[j]=individuos[i][j];
}

if(i%2==0)
{
        aux=individuos[i+1];
        printf("Padre %d: ",i+1);
        imprimir(individuos[i]);
        printf("Padre %d: ",i+2);
        imprimir(individuos[i+1]);
}else{
        aux=individuos[i-1];
}
printf("Subcadena del padre %d: ",i+1);
imprimir(hijo);
for(unsigned char k=0;k<10;k++)
{
        if(checar(hijo,aux[k])==1)
        {
                while(hijo[agregar]!=0)
                {
                        agregar++;
                }
                hijo[agregar]=aux[k];
                agregar++;
        }
}

printf("Hijo %d: ",i%2+1);
```

```
                imprimir(hijo);
                agregar=0;
        }
}
void partiallyMapped(unsigned char** individuos)
{
        long ltime;
        ltime=time(NULL);
        int stime,valor1,valor2,agregar1=0,agregar2=0;
        stime=(unsigned)ltime/2;
        srand(stime);
        unsigned char* hijo1,*hijo2,*aux1,*aux2;
        for(unsigned char i=0;i<6;i=i+2)
        {
                hijo1=calloc(10,sizeof(unsigned char));
                hijo2=calloc(10,sizeof(unsigned char));
                aux1=calloc(10,sizeof(unsigned char));
                aux2=calloc(10,sizeof(unsigned char));
                valor1=2+rand()%10;
                valor2=rand()%(valor1-1);
                for(unsigned char j=valor2;j<valor1;j++)
                {
                                hijo1[j]=individuos[i+1][j];
                                hijo2[j]=individuos[i][j];
                }
                for(unsigned char j=0;j<valor2;j++)
                {
                        if(checar(hijo1,individuos[i][j])==1)
                        {
```

```
                aux1[j]=individuos[i][j];
        }
        if(checar(hijo2,individuos[i+1][j])==1)
        {
                aux2[j]=individuos[i+1][j];
        }

}
for(unsigned char j=valor1;j<10;j++)
{
        if(checar(hijo1,individuos[i][j])==1)
        {
                aux1[j]=individuos[i][j];
        }
        if(checar(hijo2,individuos[i+1][j])==1)
        {
                aux2[j]=individuos[i+1][j];
        }
}
printf("Padre %d: ",i+1);
imprimir(individuos[i]);
printf("Padre %d: ",i+2);
imprimir(individuos[i+1]);
printf("Puntos de Cruza: %d,%d \n",valor2,valor1);

for(unsigned char k=0;k<10;k++)
{
        if(aux1[k]!=0)
        {
```

```
                while(hijo1[agregar1]!=0)
                {
                        agregar1++;
                }
                hijo1[agregar1]=aux1[k];
                agregar1++;
        }
        if(aux2[k]!=0)
        {
                while(hijo2[agregar2]!=0)
                {
                        agregar2++;
                }
                hijo2[agregar2]=aux2[k];
                agregar2++;
        }
    }
    rellenar(hijo1);
    rellenar(hijo2);
    printf("Hijo %d: ",i+1);
    imprimir(hijo1);
    printf("Hijo %d: ",i+2);
    imprimir(hijo2);
    agregar1=0;
    agregar2=0;
  }


}
```

```
void positionBased(unsigned char** individuos)
{
	long ltime;
	ltime=time(NULL);
	int stime,valor1,valor2,agregar1=0,agregar2=0;
	stime=(unsigned)ltime/2;
	srand(stime);
	unsigned char* posiciones,nPos,ale=0,*hijo,agregar=0,*aux;
	for(unsigned char i=0;i<6;i++)
	{
		nPos=(unsigned char)(1+rand()%9);
		posiciones=calloc(nPos,sizeof(unsigned char));
		hijo=calloc(10,sizeof(unsigned char));
		aux=calloc(10,sizeof(unsigned char));
		if(i%2==0)
		{
			printf("Padre %d: ",i+1);
			imprimir(individuos[i]);
			printf("Padre %d: ",i+2);
			imprimir(individuos[i+1]);
			for(int m=0;m<10;m++)
			{
				aux[m]=individuos[i+1][m];
			}

		}else{
			for(int m=0;m<10;m++)
			{
				aux[m]=individuos[i-1][m];
```

```
        }
}
printf("Numero de Puntos escogidos: %d\n",nPos);
printf("Secuencia escogida del padre %d: ",i+1);
for(unsigned char j=0;j<nPos;j++)
{
        while(checar(posiciones,ale)!=1)
        {
                ale=rand()%10;
        }
        posiciones[j]=ale;
        hijo[ale]=individuos[i][ale];
        printf("%d",hijo[ale]);
        for(unsigned char n=0;n<10;n++)
        {
                if(aux[n]==individuos[i][ale])
                {
                        aux[n]=0;
                }
        }
        ale=0;
}
printf("\n");
for(unsigned char k=0;k<10;k++)
{
        if(hijo[k]==0)
        {
                while(aux[agregar]==0)
                {
```

```
                                agregar++;
                        }
                        hijo[k]=aux[agregar];
                        agregar++;
                    }
                }

                printf("Hijo %d: ",i%2 + 1);
                imprimir(hijo);
                agregar=0;
                free(posiciones);
        }
}
void orderBased(unsigned char** individuos)
{
        long ltime;
        ltime=time(NULL);
        int stime,valor1,valor2,agregar1=0,agregar2=0;
        stime=(unsigned)ltime/2;
        srand(stime);
        unsigned char* posiciones,nPos,ale=0,*hijo,agregar=0,*aux;
        for(unsigned char i=0;i<6;i++)
        {
                nPos=(unsigned char)(1+rand()%9);
                posiciones=calloc(nPos,sizeof(unsigned char));
                hijo=calloc(10,sizeof(unsigned char));
                aux=calloc(10,sizeof(unsigned char));
                if(i%2==0)
                {
```

```
		printf("Padre %d: ",i+1);
		imprimir(individuos[i]);
		printf("Padre %d: ",i+2);
		imprimir(individuos[i+1]);
		for(int m=0;m<10;m++)
		{
			aux[m]=individuos[i+1][m];
		}

	}else{
		for(int m=0;m<10;m++)
		{
			aux[m]=individuos[i-1][m];
		}
	}
	printf("Numero de Puntos escogidos: %d\n",nPos);
	printf("Secuencia escogida del padre %d: ",i+1);
	for(unsigned char j=0;j<nPos;j++)
	{
		while(checar(posiciones,ale)!=1)
		{
			ale=rand()%10;
		}
		posiciones[j]=ale;
		printf("%d",individuos[i][ale]);
		for(unsigned char n=0;n<10;n++)
		{
			if(aux[n]!=individuos[i][ale])
			{
```

```
                                hijo[n]=aux[n];
                        }else{
                                hijo[n]=0;
                                aux[n]=0;
                        }
                }
                ale=0;
        }
        printf("\n");
        for(unsigned char k=0;k<=nPos;k++)
        {
                while(hijo[agregar]!=0)
                {
                        agregar++;
                }
                hijo[agregar]=individuos[i][posiciones[k]];
                agregar++;
        }
        printf("Hijo %d: ",i%2 + 1);
        imprimir(hijo);
        agregar=0;
        free(posiciones);
    }
}
void cycleCrossover(unsigned char** individuos)
{
    long ltime;
    ltime=time(NULL);
    int stime,posicion,agregar=0,nuevo;
```

```
	stime=(unsigned)ltime/2;
	srand(stime);
	unsigned char* elementos;
	for(unsigned char i=0;i<6;i+=2)
	{
		elementos=calloc(10,sizeof(unsigned char));
		posicion=rand()%10;
		elementos[agregar]=individuos[i][posicion];
		agregar++;
		elementos[agregar]=individuos[i+1][posicion];
		nuevo=buscar(individuos[i],individuos[i+1][posicion]);
		if(checar(elementos,individuos[i][nuevo])!=0)
		{
			elementos[agregar]=individuos[i][nuevo];
			agregar++;
		}
		nuevo=buscar(individuos[i+1],individuos[i][posicion]);
		if(checar(elementos,individuos[i+1][nuevo])!=0)
		{
			elementos[agregar]=individuos[i][nuevo];
			agregar++;
		}
	}

}
int checar(unsigned char* individuo,int cromosoma)
{
	for(int i=0;i<10;i++)
	{
```

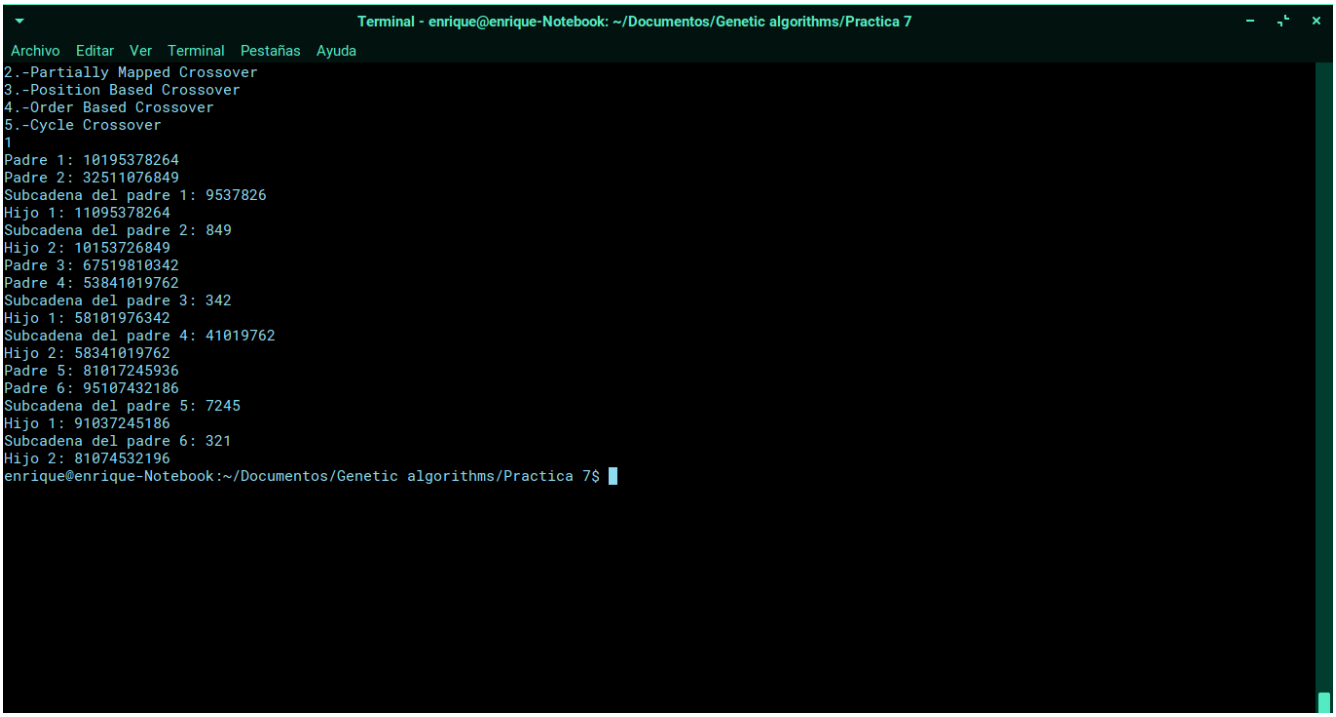
```
            if(individuo[i]==cromosoma)
                return 0;
        }
        return 1;
}
void imprimir(unsigned char* ind)
{
        for(int n=0;n<10;n++)
        {
            //if(ind[n]!=0)
                printf("%d",ind[n]);
        }
        printf("\n");
}
void rellenar(unsigned char* hijo)
{
        unsigned char posicion=0;
        for(unsigned char i=1;i<11;i++)
        {
            if(checar(hijo,i)==1)
            {
                while(hijo[posicion]!=0)
                {
                    posicion++;
                }
                hijo[posicion]=i;
            }

        }
```

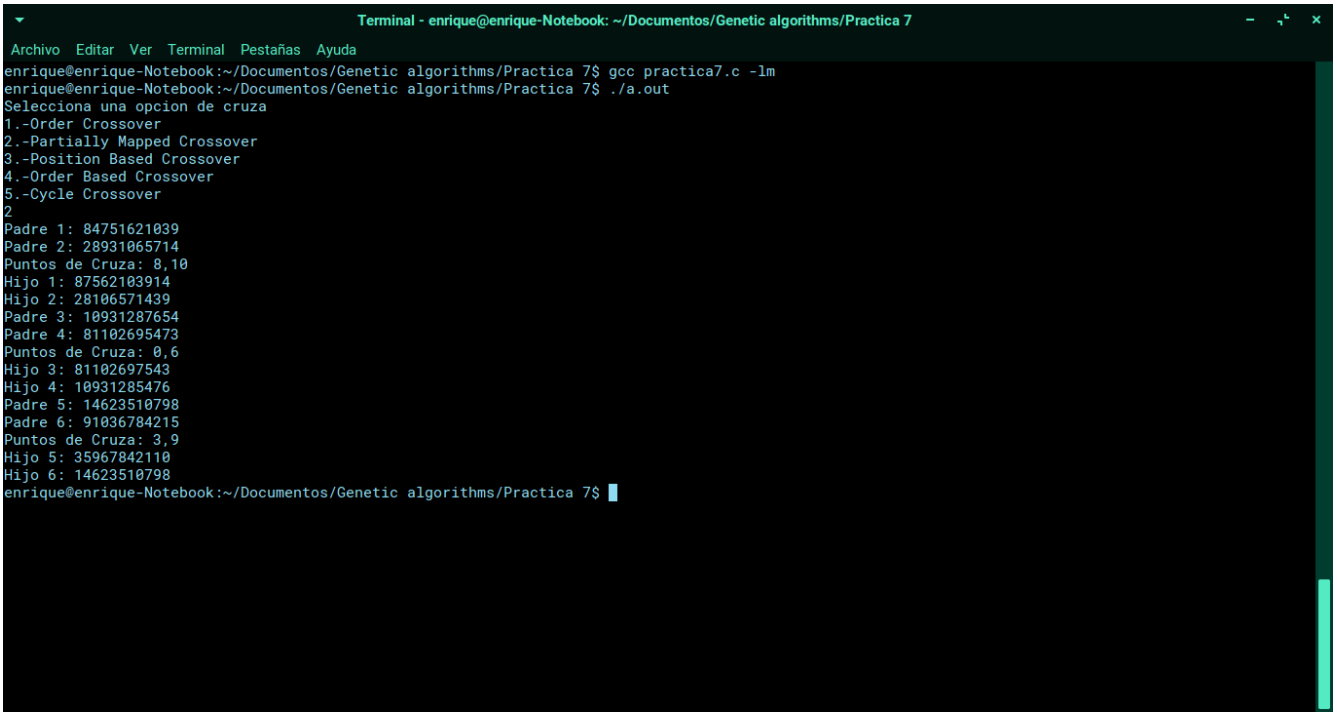
```
}
int buscar(unsigned char* individuo, int valor)
{
        return 0;
}
```

Ejecucion del programa:

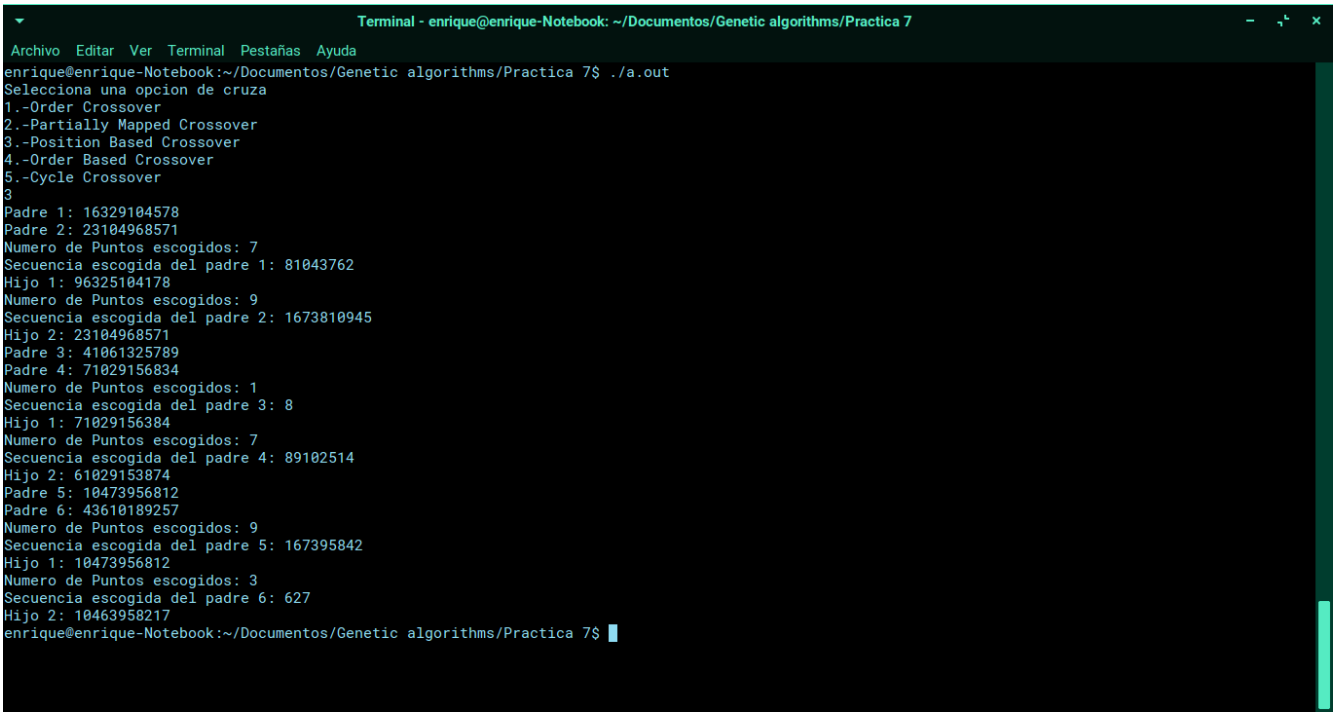
Experimento con Order Crossover:



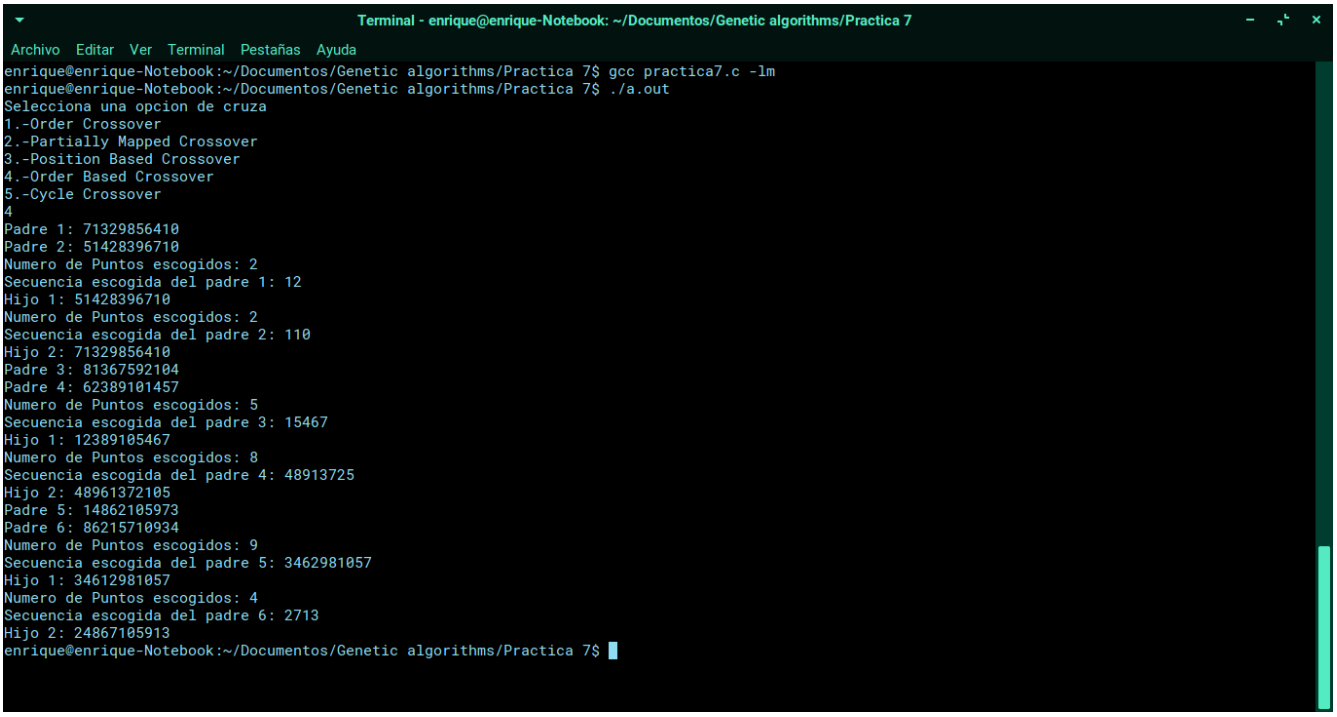
Experimento con Partially Mapped Crossover:



Experimento con Position Based Crossover:



Experinento con Order Based Crossover:



Conclusiones:

Dado que la representación entera de los individuos tiene un conjunto de alelos más grande que la representación binaria era necesario el crear nuevos algoritmos que ofrecieran una crza entre individuos que diera resultados óptimos, la mayoría de los algoritmos aquí mencionados son una daptación de los de la práctica pasada, lo que demuestra que los resultados obtenidos con esos algoritmos son óptimos y pueden ser usados en esta nueva representación, si bien el algoritmo de Cycle Crossover es el más complejo de estos, no quiere decir que sea el mejor, nosw asegura una completa mezcla de los alelos de cada padre en su descendencia, pero sería necesario probar sus cualidades para saber si en una población aleatoria este nos dé los resultados más óptimos.

Referencias:

[1]Introducción al Cómputo Evolutivo, Coello