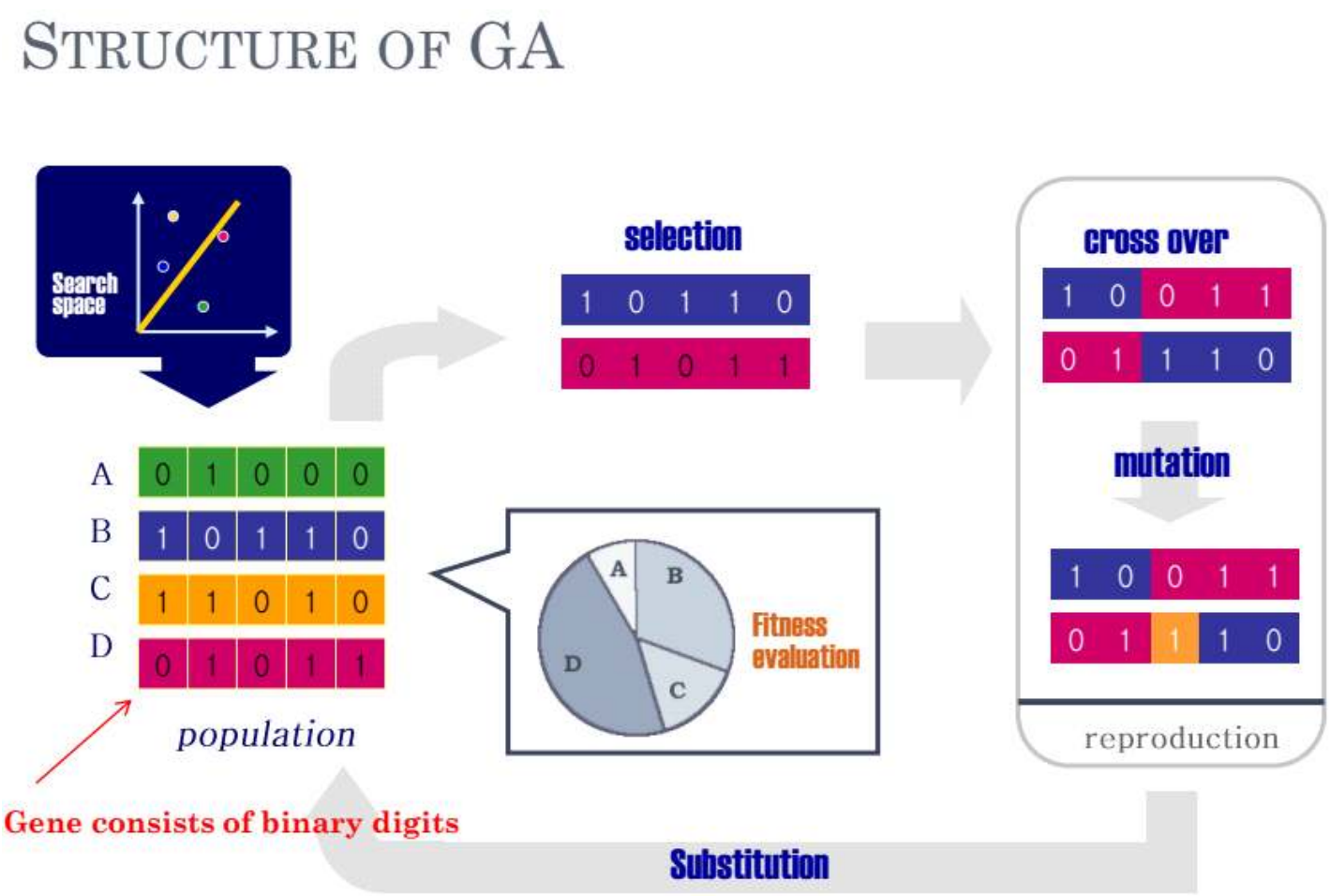
Genetic Algorithm-리포트

이름 : 박건호

학과 : 컴퓨터공학과

학번 : 201635964



목차

① 코드 흐름 설명

Step 0 : 변수 설정

```
#define SIZE 8 //찾을 비밀번호 길이
#define CHROMOSOME 10 //염색체 수
#define POINT 4 //교배할 위치

struct gen{
    int num[SIZE]; //DNA
    int score; // 적합도
};

struct gen gene[CHROMOSOME]; //유전자
struct gen next_gene[CHROMOSOME]; //다음 세대 유전자

int pin_number[SIZE]; //비밀번호
int fit[10]; // 숫자 체크
double selectp[CHROMOSOME]; // 퍼센트
int tmp = 0;
int cnt = 0;
int check = -1;
int sum = 0;
int rdm;
int select1, select2;

double total = (10 + 2) * SIZE; // 결과 출력에서 Accurate를 %값으로 나타내기 위한 변수
```

Step 1 : FitnessFunc(적합도 함수)

```
int FitnessFunc() { //적합도 함수
    tmp = 0;
    for (int i = 0; i < CHROMOSOME; i++) { //위치, 숫자 맞으면 10점
        for (int j = 0; j < SIZE; j++) {
            if (gene[i].num[j] == pin_number[j]) {
                gene[i].score = gene[i].score  + 10;
            }
        }
    }

    for (int i = 0; i < CHROMOSOME; i++) { // 숫자가 포함되어있는 만큼 2점
        for (int j = 0; j < SIZE; j++) {
            if (fit[gene[i].num[j]] == 1) {
                gene[i].score = gene[i].score + 2;
            }
        }
    }

    for (int i = 0; i < CHROMOSOME; i++) {
        if (gene[i].score == (10 + 2) * SIZE) { //전부 일치 할 경우
            return i; // 일치하는 경우 index값 반환
        }
    }
    return -1;
}
```

Step 2 : SelcetFunc(선택 함수)

```
void SelectFunc() //유전자 2개 선택
{
    double sum = 0;

    for (int i = 0; i < CHROMOSOME; i++) {
        sum = sum + gene[i].score;
    }

    for (int i = 0; i < CHROMOSOME; i++) {
        selectp[i] = ((gene[i].score / sum) * 100); //100점 만점으로 환산
    }

    rdm = rand() % 100; // 선택할 유전자 랜덤하게 결정
    sum = 0;

    for (int i = 0; i < CHROMOSOME; i++) {
        sum = sum + selectp[i];
        if (rdm < sum) {
            select1 = i;
            break;
        }
    }


    while (true) { //첫번째와 두번째가 값이 다를 때 까지 반복
        rdm = rand() % 100;
        sum = 0;
        for (int i = 0; i < CHROMOSOME; i++) {
            sum = sum + selectp[i];
            if (rdm < sum) {
                select2 = i;
                break;
            }
        }
        if (select1 == select2) {
            continue;
        }
        else {
            break;
        }
    }
}
```

Step 3 : CrossFunc(교배 함수)

```
void CrossFunc() { //유전자 교배
    for (int i = 0; i < CHROMOSOME; i += 2) {
        SelectFunc();
        for (int j = 0; j < POINT; j++) { //지정한 POINT지점에서 교차
            next_gene[i].num[j] = gene[select1].num[j];
            next_gene[i+1].num[j] = gene[select2].num[j];
        }
        for (int j = POINT; j < SIZE; j++) {
            next_gene[i].num[j] = gene[select2].num[j];
            next_gene[i+1].num[j] = gene[select1].num[j];
        }
    }

    for (int i = 0; i < CHROMOSOME; i++) { //돌연변이
        for (int j = 0; j < SIZE; j++) {
            rdm = rand() % 100;
            if (rdm == 1) {
                next_gene[i].num[j] = rand() % 10;
            }
        }
    }

    for (int i = 0; i < CHROMOSOME; i++) { //교배시킨 유전자 대입
        for (int j = 0; j < SIZE; j++) {
            gene[i].num[j] = next_gene[i].num[j];
        }
        gene[i].score = 0;
    }
}
```

SelcetFunc() 함수를 이용하여 i 번째 유전자와 i+1 번째 유전자 교차 시킨 후 1%의 확률로 돌연변이를 발생시켜 기존 유전자를 갱신한다.

Step 4 : 세대 반복

```
while (true) {
    cout << endl;
    cout << "-----" << cnt << "세대" << "-----" << endl;

    check = FitnessFunc();
    if (check > -1) {
        break;
    }

    for (int i = 0; i < CHROMOSOME; i++) {
        for (int j = 0; j < SIZE; j++) {
            cout << gene[i].num[j];
        }
        cout << " ---> " << (gene[i].score / total) * 100 << "% 일치" << endl;
    }


    CrossFunc();
    cnt++;
    cout << endl;
}
```

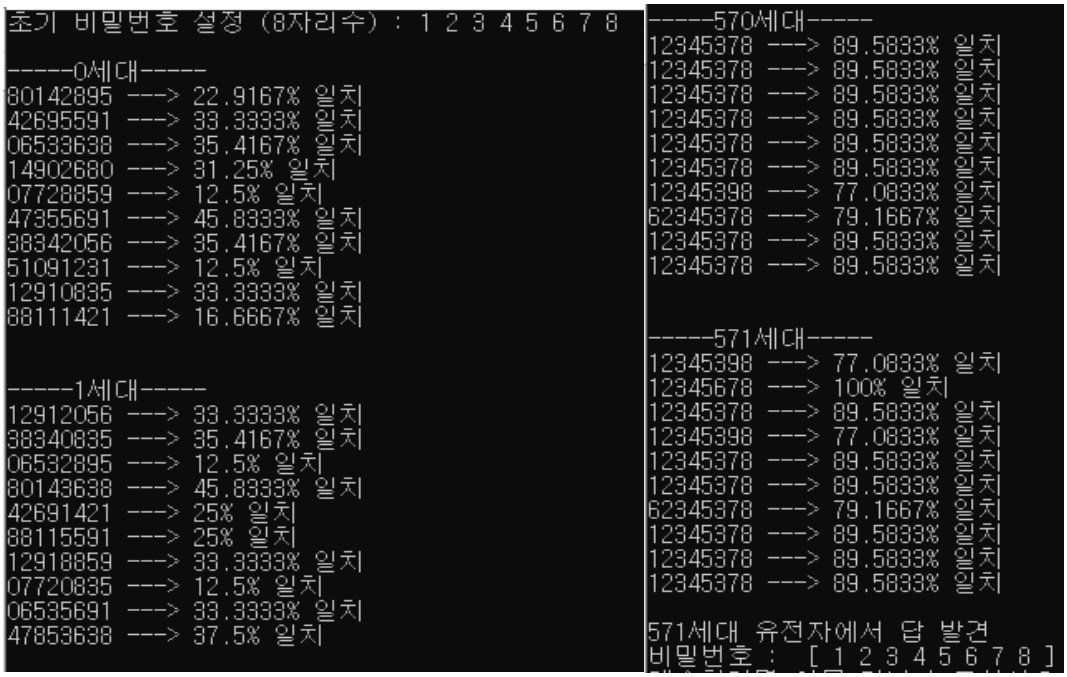
일치하는 유전자의 index값이 반환 될 때 까지 반복문을 실행한다.

Step 5 : 결과 출력

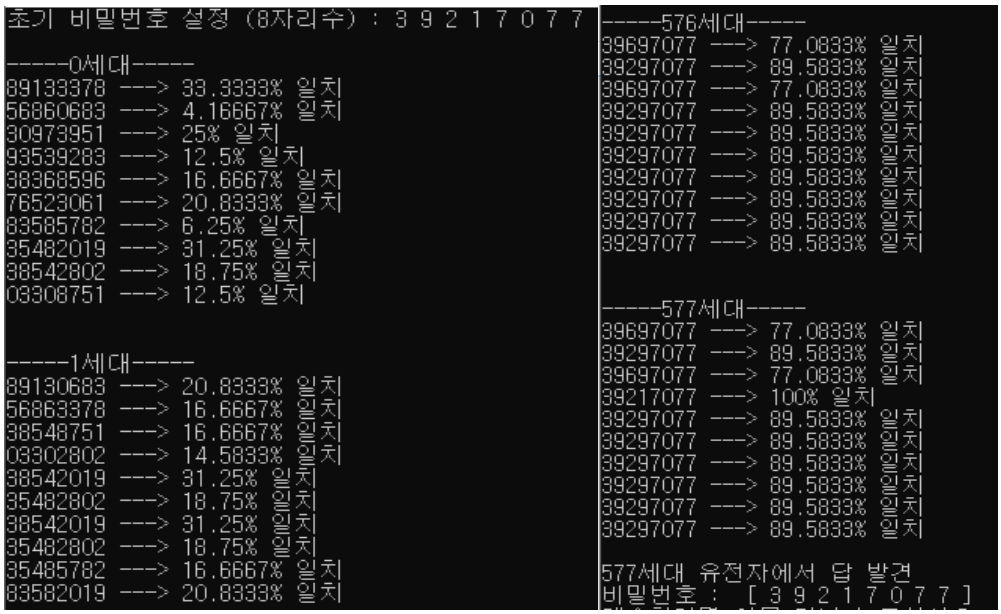
```
//결과 세대 출력
for (int i = 0; i < CHROMOSOME; i++) {
    for (int j = 0; j < SIZE; j++) {
        cout << gene[i].num[j];
    }
    cout << " ---> " << (gene[i].score / total) * 100 << "% 일치" << endl;
}

//결과 출력
cout << endl;
cout << cnt << "세대 유전자에서 답 발견" << endl;

cout << "비밀번호 : ";
cout << " [ ";
for (int i = 0; i < SIZE; i++) {
    cout << gene[check].num[i] << " ";
}
cout << "] ";
cout << endl;
```

각 세대의 유전자들을 출력하고 결과값과 얼마나 일치하는지 %단위로 출력하며 몇세대 만에 나왔는지 출력

② 실행 결과

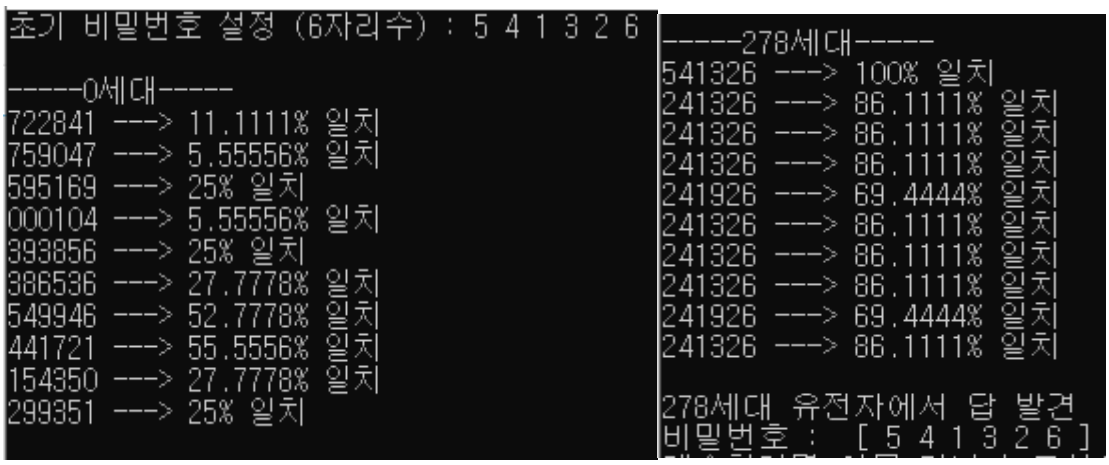


위는 12345678 로 비밀번호를 설정한 후 실행 결과이다.



위는 아무렇게 입력한 숫자로 설정한 후 실행 결과이다.

대체적으로 1000세대 미만 세대 안에서 결과값이 나오는 것을 알 수 있다. 설정된 비밀번호가 중복된 값이 적을수록 결과가 평균적으로 더 빨리 나왔다. 당연한 얘기이지만 비밀번호 자릿수가 적을수록 더 적은 세대 안에서 결과값을 찾을 수 있었다.



③ 전체 코드

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <string>
#include <cstdio>
#include <time.h>

using namespace std;

#define SIZE 8 //찾을 비밀번호 길이
#define CHROMOSOME 10 //염색체 수
#define POINT 4 //교배할 위치

struct gen{
	int num[SIZE];
	int score;
};

struct gen gene[CHROMOSOME];
struct gen next_gene[CHROMOSOME];

int pin_number[SIZE]; //비밀번호
int fit[10]; // 숫자 체크
double selectp[CHROMOSOME]; // 퍼센트
int tmp = 0;
int cnt = 0;
int check = -1;
int sum = 0;
int rdm;
int select1, select2;

double total = (10 + 2) * SIZE;


int FitnessFunc() { //적합도 함수
	tmp = 0;
	for (int i = 0; i < CHROMOSOME; i++) { //위치, 숫자 맞으면 10점
		for (int j = 0; j < SIZE; j++) {
			if (gene[i].num[j] == pin_number[j]) {
				gene[i].score = gene[i].score  + 10;
```

```
                        }
                }
        }

        for (int i = 0; i < CHROMOSOME; i++) { // 숫자가 포함되어있는 만큼 2점
                for (int j = 0; j < SIZE; j++) {
                        if (fit[gene[i].num[j]] == 1) {
                                gene[i].score = gene[i].score + 2;
                        }
                }
        }

        for (int i = 0; i < CHROMOSOME; i++) {
                if (gene[i].score == (10 + 2) * SIZE) { //전부 일치 할 경우
                        return i;
                }
        }
        return -1;
}

void SelectFunc() //유전자 2개 선택
{
        double sum = 0;

        for (int i = 0; i < CHROMOSOME; i++) {
                sum = sum + gene[i].score;
        }

        for (int i = 0; i < CHROMOSOME; i++) {
                selectp[i] = ((gene[i].score / sum) * 100); //100점 만점으로 환산
        }

        rdm = rand() % 100;
        sum = 0;

        for (int i = 0; i < CHROMOSOME; i++) {
                sum = sum + selectp[i];
                if (rdm < sum) {
                        select1 = i;
                        break;
                }
        }
```

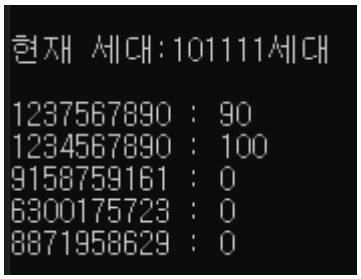
```
	while (true) { //첫번째와 두번째가 값이 다를 때 까지 반복
		rdm = rand() % 100;
		sum = 0;
		for (int i = 0; i < CHROMOSOME; i++) {
			sum = sum + selectp[i];
			if (rdm < sum) {
				select2 = i;
				break;
			}
		}
		if (select1 == select2) {
			continue;
		}
		else {
			break;
		}
	}
}


void CrossFunc() { //유전자 교배
	for (int i = 0; i < CHROMOSOME; i += 2) {
		SelectFunc();
		for (int j = 0; j < POINT; j++) {
			next_gene[i].num[j] = gene[select1].num[j];
			next_gene[i+1].num[j] = gene[select2].num[j];
		}
		for (int j = POINT; j < SIZE; j++) {
			next_gene[i].num[j] = gene[select2].num[j];
			next_gene[i+1].num[j] = gene[select1].num[j];
		}
	}

	for (int i = 0; i < CHROMOSOME; i++) { //돌연변이
		for (int j = 0; j < SIZE; j++) {
			rdm = rand() % 100;
			if (rdm == 1) {
				next_gene[i].num[j] = rand() % 10;
			}
		}
	}
```

```
	for (int i = 0; i < CHROMOSOME; i++) { //교배시킨 유전자 대입
		for (int j = 0; j < SIZE; j++) {
			gene[i].num[j] = next_gene[i].num[j];
		}
		gene[i].score = 0;
	}
}



int main() {
	srand(time(NULL));

	cout << "초기 비밀번호 설정 (" << SIZE << "자리수) : ";

	for (int i = 0; i < SIZE; i++) {
		cin >> pin_number[i];
		fit[pin_number[i]] = 1;
	}

	//초기 염색체 생성
	for (int i = 0; i < CHROMOSOME; i++) {
		for (int j = 0; j < SIZE; j++) {
			gene[i].num[j] = rand() % 10;
		}
		gene[i].score = 0;
	}

	while (true) {
		cout << endl;
		cout << "-----" << cnt << "세대" << "-----" << endl;

		check = FitnessFunc();
		if (check > -1) {
			break;
		}

		for (int i = 0; i < CHROMOSOME; i++) {
			for (int j = 0; j < SIZE; j++) {
				cout << gene[i].num[j];
			}
			cout << " ---> " << (gene[i].score / total) * 100 << "% 일치" << endl;
```

```
                }

                CrossFunc();
                cnt++;
                cout << endl;
        }

        //결과 세대 출력
        for (int i = 0; i < CHROMOSOME; i++) {
                for (int j = 0; j < SIZE; j++) {
                        cout << gene[i].num[j];
                }
                cout << " ---> " << (gene[i].score / total) * 100 << "% 일치" << endl;
        }

        //결과 출력
        cout << endl;
        cout << cnt << "세대 유전자에서 답 발견" << endl;

        cout << "비밀번호 : ";
        cout << " [ ";
        for (int i = 0; i < SIZE; i++) {
                cout << gene[check].num[i] << " ";
        }
        cout << "] ";
        cout << endl;

        return 0;
}
```

④ 소감문

유전자 알고리즘은 전체적인 알고리즘 순서로만 보았을 때 지금까지 나왔던 과제들에 비해 가장 간단해 보이는 알고리즘이였다. 하지만 코딩하면서 단계가 단순한 것이고 내용 자체는 그리 단순하지만은 않다는 것을 알게되었다. 유전자의 Fiteness를 어떻게 계산할 것인지, 어떻게 교배시킬 것인지 어떤 방식으로 엘리트 유전자들을 자손에게 물려줄 것인지 생각하며 코딩했어야 했기 때문에 처음에는 0과 1만으로 유전자들을 구성하여 구현하였었다. 구현 후에 유전자들의 수와 유전자가 가지고 있는 데이터들의 크기를 늘려나가며 Fitness함수를 수정했다. Step2에서 선택한 2개의 유전자를 Step3에서 교배시킨 후 2개의 유전자를 만들고 다른 나머지 유전자들을 랜덤하게 갱신 하였더니 결과값이 만족스럽지 못하게 나왔다. 최소 10만세대 이상 반복이 돌아야 결과값이 나왔었다.



그래서 엘리트 유전자들의 영향을 더욱 크게 주고자 1,2번째 유전자, 3,4번째 유전자.... 이런식으로 짝을지어 Selcet함수 실행 후 교배시켜 모든 유전자들을 교배시켜 구현하였더니 더욱 좋은 성능을 보여주었다.

모든 문제가 그렇겠지만 항상 풀고나면 생각만큼 어렵지 않은 문제인 것 같다. 문제를 해결하면서 알게된 지식들 덕이겠지만 끝내고 나면 후련함과 더불어 허탈감도 조금씩 있는 것 같다. 다층 퍼셉트론에 비해 구현부분에서는 크게 어렵지 않았지만 인공지능을 진짜 내가 배우고 있다는 생각이 들게 해주는 경험인 것 같다.