



# PROJEK UAS ALGORITMA GENETIKA

PEMROGRAMAN LANJUTAN

Present by:  
KELOMPOK 3

Rismawati(231021002)-Anggitamaya(231021004)-Evei(231061001)

# BAHAN DISKUSI

## Pengertian Algoritma Genetika

### Bentuk Crossover

### Bentuk Mutasi

### Hasil Percobaan

# PENGERTIAN

Algoritma genetika adalah teknik optimisasi berbasis populasi yang menggunakan prinsip-prinsip seleksi dan genetika untuk menemukan solusi yang optimal atau mendekati optimal dalam ruang pencarian yang besar.

–Mitchell Melone

# TARGET

```
crossover_point2 = random.randint(crossover_point1, len(target))
crossover_point3 = random.randint(crossover_point2, len(target))

child1 = parent1[:crossover_point1] + parent2[crossover_point1:crossover_point2] + parent1[crossover_point2:crossover_point3]
child2 = parent2[:crossover_point1] + parent3[crossover_point1:crossover_point2] + parent2[crossover_point2:crossover_point3]
return child1, child2

# Fungsi untuk melakukan mutasi pada individu
def mutate(individual, mutation_rate):
    return ''.join(char if random.random() > mutation_rate else random.choice(string.ascii_uppercase + " ") for char in individual)

# Fungsi utama algoritma genetika
def genetic_algorithm():
    population = initial_population()
    best_individual = None
    for generation in range(1, generations + 1):
        fitnesses = [calculate_fitness(individual) for individual in population]
        best_fitness = max(fitnesses)
        best_individual = population[fitnesses.index(best_fitness)]
        if best_fitness == len(target):
            break
        next_population = []
        while len(next_population) < population_size:
            parent1, parent2 = random.sample(population, 2)
            child1, child2 = crossover(parent1, parent2)
            next_population.append(mutate(child1, mutation_rate))
            next_population.append(mutate(child2, mutation_rate))
        population = next_population
```

*# Target string yang ingin dicapai*

**target = "PEMROGRAMAN LANJUTAN"**

*# Parameter algoritma genetika*

**population\_size = 100**

**mutation\_rate = 0.05**

**generations = 100000**



# BENTUK CROSSOVER

## Pemisalan crossover:

[illegible]

```
best_individual = population[fitnesses.index(best_fitness)]
if best_fitness == len(target):
    break
next_population = [best_individual] # Elitisme: mempertahankan individu terbaik
while len(next_population) < population_size:
    parent1, parent2, parent3 = selection(population, fitnesses)
    child1, child2 = crossover(parent1, parent2, parent3)
    next_population.append(mutate(child1, mutation_rate))
    if len(next_population) < population_size:
        next_population.append(mutate(child2, mutation_rate))
population = next_population
```

# BENTUK CROSSOVER

```
crossover_point2 = random.randint(crossover_point1, len(target))
crossover_point3 = random.randint(crossover_point2, len(target))
```

```
child1 = parent1[:crossover_point1] + parent2[crossover_point1:crossover_point2] + parent1[crossover_point2:crossover_point3]
child2 = parent2[:crossover_point1] + parent1[crossover_point1:crossover_point2] + parent2[crossover_point2:crossover_point3]
return child1, child2
```

# Fungsi untuk melakukan mutasi pada individu

```
def mutate(individual, mutation_rate):
    return ''.join(char if random.random() > mutation_rate else random.choice(string.ascii_uppercase + " ") for char in individual)
```

## Crossover:

pemisalan:																					
array	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
cros1	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
cros2	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
cros3	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I

```
best_individual = ''
```

for

parent:																				
1	E	E	H	K	W	D	H	6	M	F	E	W	2	A	X	V	H	L	A	E
2	H	J	M	I	K	N	R	J	R	I	S	M	A	A	J	N	R	E	Q	X
3	P	6	E	T	G	I	R	I	S	M	A	W	A	T	I	J	M	A	Y	A
child1:																				
1	E	E	H	K	W	D	H	6	M	F	E	W	2	A	X	V	H	L	A	E
2	H	J	M	I	K	N	R	J	R	I	S	M	A	A	J	N	R	E	Q	X
a1	E	E	H	K	W	N	R	J	R	I	E	W	2	A	X	V	R	E	Q	X
child2:																				
2	H	J	M	I	K	N	R	J	R	I	S	M	A	A	J	N	R	E	Q	X
3	P	6	E	T	G	I	R	I	S	M	A	W	A	T	I	J	M	A	Y	A
a2	H	J	M	I	K	I	R	I	S	M	S	M	A	A	J	N	M	A	Y	A

```
if len(next_population) < population_size:
    next_population.append(mutate(child2, mutation_rate))
```

```
population = next_population
```

# BENTUK MUTASI

```
crossover_point2 = random.randint(crossover_point1, len(target))
crossover_point3 = random.randint(crossover_point2, len(target))
```

```
child1 = parent1[:crossover_point1] + parent2[crossover_point1:crossover_point2] + parent1[crossover_point2:crossover_point3]
child2 = parent2[:crossover_point1] + parent3[crossover_point1:crossover_point2] + parent2[crossover_point2:crossover_point3]
return child1, child2
```

```
# Fungsi untuk melakukan mutasi pada individu
```

```
def mutate(individual, mutation_rate):
    return ''.join(char if random.random() > mutation_rate else random.choice(string.ascii_uppercase + " ") for char in individual)
```

```
# Fungsi utama algoritma genetika
```

```
def genetic_algorithm():
    population = initialize_population()
    best_individual = None
    for generation in range(generations):
        fitnesses = [calculate_fitness(ind) for ind in population]
        best_fitness = max(fitnesses)
```

	P	E	M	R	O	G	R	A	M	A	N		L	A	N	J	U	T	A	N	
	0.7	0.9	0.09	0.5	0.5	0.1	0.2	0.2	0.3	0.8	0.6	0.9	0.4	0.03	0.6	0.8	0.9	0.3	0.7	0.5	
a1	E	E	H	K	W	N	R	J	R	I	E	W	2	A	X	V	R	E	Q	X	
A1	D	1	r	;	^	o	%	!	M	3	G	r	A	A	P	s	r	i	G	Y	
	0.5	0.7	0.6	0.07	0.7	0.3	0.03	0.02	0.9	0.2	0.3	0.5	0.4	0.01	0.8	0.3	0.8	0.2	0.7	0.03	
a2	H	J	M	I	K	I	R	I	S	M	S	M	A	A	J	N	M	A	Y	A	
A2	x	.		V	F	W	R	J	n	?	N	<	I	A	I	P	f	A	b	A	

```
    parent1, parent2, parent3 = selection(population, fitnesses)
    child1, child2 = crossover(parent1, parent2, parent3)
    next_population.append(mutate(child1, mutation_rate))
    if len(next_population) < population_size:
        next_population.append(mutate(child2, mutation_rate))
    population = next_population
```

# HASIL PERCOBAAN

PERCOBAAN	GENERASI	WAKTU
1	32	0.05
2	68	0.21
3	35	0.05
4	44	0.06
5	34	0.06
6	24	0.01
7	37	0.07
8	39	0.05
9	78	0.1
10	98	0.13
Jumlah:	489	0.79
Hasil jumlah / 10:	48.9	0.079



```
def crossover(parent1, parent2, parent3, target):
    crossover_point1 = random.randint(0, len(target)-1)
    crossover_point2 = random.randint(crossover_point1, len(target))
    crossover_point3 = random.randint(crossover_point2, len(target))

    child1 = parent1[:crossover_point1] + parent2[crossover_point1:crossover_point2] + parent1[crossover_point2:crossover_point3]
    child2 = parent2[:crossover_point1] + parent3[crossover_point1:crossover_point2] + parent2[crossover_point2:crossover_point3]
    return child1, child2

# Fungsi untuk melakukan mutasi pada individu
def mutate(individual, mutation_rate):
    return ''.join(char if random.random() > mutation_rate else random.choice(string.ascii_uppercase + " ") for char in individual)

# Fungsi utama algoritma genetika
def genetic_algorithm():
    population = initialize_population(population_size, target)
    best_individual = ''
    for generation in range(generations):
        fitnesses = [calculate_fitness(ind) for ind in population]
        best_fitness = max(fitnesses)
        best_individual = population[fitnesses.index(best_fitness)]
        if best_fitness == len(target):
            break
    next_population = [best_individual] # Elitisme: mempertahankan individu terbaik
    while len(next_population) < population_size:
        parent1, parent2, parent3 = selection(population, fitnesses)
        child1, child2 = crossover(parent1, parent2, parent3)
        next_population.append(mutate(child1, mutation_rate))
        if len(next_population) < population_size:
            next_population.append(mutate(child2, mutation_rate))
    population = next_population
```

TERIMAKASIH

PEMROGRAMAN LANJUTAN

Dosen Pengampu:

Zaitun, S.Si., M.Mat.