NORTH WESTERN UNIVERSITY, KHULNA



Course Title: Artificial Intelligence and Expert Systems Sessional

Course Code: CSE-3302

Lab Report

|  |  |
| --- | --- |
| **Submitted by:**  Name: Fatema Zahan Shayla  Id: 20201059010  Department of Computer Science and Engineering.  North Western University, Khulna. | **Submitted to:**  Tajul Islam  Department Head  Department of Computer Science and Engineering.  North Western University, Khulna. |

**Submission Date:25.01.2023** **Teacher’s Signature**

**1. Algorithm Name: Breadth-First Search(BFS).**

**Description:** Breadth-first search (BFS) is an algorithm that is used to graph data or searching tree or traversing structures. The full form of BFS is the Breadth-first search.

The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadthwise fashion. This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. Remember, BFS accesses these nodes one by one.

Once the algorithm visits and marks the starting node, then it moves towards the nearest unvisited nodes and analyses them. Once visited, all nodes are marked. These iterations continue until all the nodes of the graph have been successfully visited and marked.

**Code Implementation using C-Programming:**

#include<stdio.h>

#include<conio.h>

/\*

Naming Convention-->

Name: Sajib Kumar

s = s\_forMatrixArray,

a = a\_forQuee,

j = j\_forVisitedArray,

i = i\_forNumberOfVartex,

b = b\_forLoop,

k = k\_forNastedLoop,

u = f\_forFront,

m = m\_forRare

\*/

int s\_forMatrixArray[20][20], a\_forQuee[20], j\_forVisitedArray[20], i\_forNumberOfVartex, b\_forLoop, k\_forNastedLoop, u\_forFront= 0, m\_forRare = -1;

void sajib\_forBFS(int vertex)

{

for(b\_forLoop = 1; b\_forLoop <= i\_forNumberOfVartex; b\_forLoop ++)

if(s\_forMatrixArray[vertex][b\_forLoop] && !j\_forVisitedArray[b\_forLoop])

a\_forQuee[++m\_forRare] = b\_forLoop;

if(u\_forFront <= m\_forRare)

{

j\_forVisitedArray[a\_forQuee[u\_forFront]] = 1;

sajib\_forBFS(a\_forQuee[u\_forFront++]);

}

}

void main()

{

printf("\t\t\t\t Hay Welcome...!");

printf("\n\t\t\t Breadth First Search(BFS)\n\n");

char name[200] = "Sajib Bhattacharjee";

printf("\nEnter the number of Vertex: ");

scanf("%d",&i\_forNumberOfVartex);

for(b\_forLoop=1; b\_forLoop <= i\_forNumberOfVartex; b\_forLoop++)

{

a\_forQuee[b\_forLoop] = 0;

j\_forVisitedArray[b\_forLoop] = 0;

}

printf("\nEnter graph data in matrix form:\n");

for(b\_forLoop=1; b\_forLoop<=i\_forNumberOfVartex; b\_forLoop++)

{

for(k\_forNastedLoop=1; k\_forNastedLoop<=i\_forNumberOfVartex; k\_forNastedLoop++)

{

scanf("%d",&s\_forMatrixArray[b\_forLoop][k\_forNastedLoop]);

}

}

int vertex;

printf("\nEnter the starting vertex:");

scanf("%d", &vertex);

sajib\_forBFS(vertex);

printf("\nThe node which are reachable are: \n");

for(b\_forLoop=1; b\_forLoop <= i\_forNumberOfVartex; b\_forLoop++)

{

if(j\_forVisitedArray[b\_forLoop])

printf("%d\t", b\_forLoop);

else

{

printf("\nBFS is not possible. Not all nodes are reachable....\n");

printf("Please Try Again...!!!");

break;

}

}

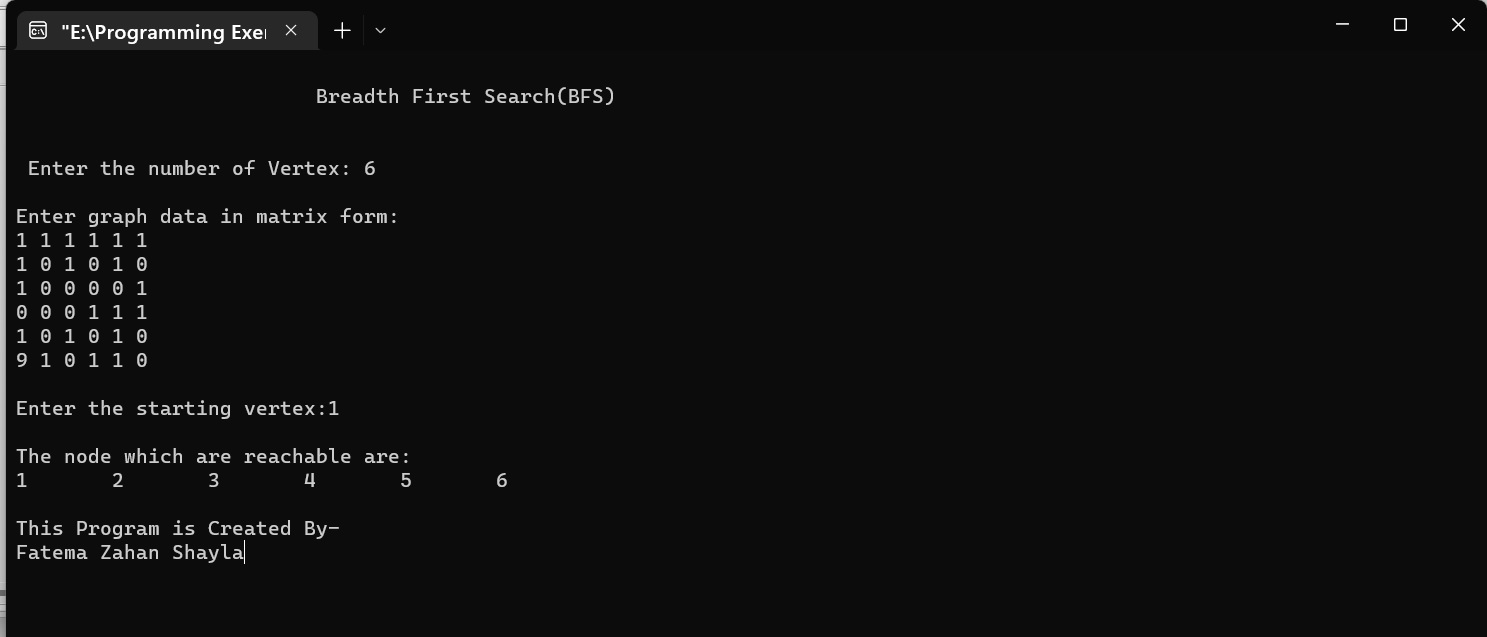
printf("\n\nThis Program is Created By- \n");

printf("%s",name);;

getch();

}

**Input & Output:**

****

**2. Algorithm Name: Depth-First Search(DFS).**

**Description:** Depth First Search (DFS) is an algorithm that is mainly used to traverse the graph data structure. The algorithm starts from an arbitrary node (root node in the case of trees) and explores as far as possible in the graph before backtracking. After backtracking it repeats the same process for all the remaining vertices which have not been visited till now.

**Code Implementation using C-Programming:**

#include<stdio.h>

#include<conio.h>

/\*

Naming Convention.

sajib kumar

s = s\_forAdjacencyMatrix,

a = a\_forQuee,

j = j\_forNumberOfVartex,

i = i\_forLoop,

b = b\_forNastedLoop,

k = k\_forCount,

\*/

int s\_forAdjacencyMatrix[20][20],a\_forQuee[20],j\_forNumberOfVartex,i\_forLoop , b\_forNastedLoop;

void sajib\_forDFS(int v)

{

a\_forQuee[v]=1;

for(i\_forLoop=1; i\_forLoop<=j\_forNumberOfVartex; i\_forLoop++)

if(s\_forAdjacencyMatrix[v][i\_forLoop] && !a\_forQuee[i\_forLoop])

{

printf("\n %d->%d",v,i\_forLoop);

sajib\_forDFS(i\_forLoop);

}

}

void main()

{

int k\_forCount=0;

//clrscr();

printf("\nEnter number of vertices:");

scanf("%d",&j\_forNumberOfVartex);

for(i\_forLoop=1; i\_forLoop<=j\_forNumberOfVartex; i\_forLoop++)

{

a\_forQuee[i\_forLoop]=0;

for(b\_forNastedLoop=1; b\_forNastedLoop<=j\_forNumberOfVartex; b\_forNastedLoop++)

s\_forAdjacencyMatrix[i\_forLoop][b\_forNastedLoop]=0;

}

printf("\nEnter the adjacency matrix:\n");

for(i\_forLoop=1; i\_forLoop<=j\_forNumberOfVartex; i\_forLoop++)

for(b\_forNastedLoop=1; b\_forNastedLoop<=j\_forNumberOfVartex; b\_forNastedLoop++)

scanf("%d",&s\_forAdjacencyMatrix[i\_forLoop][b\_forNastedLoop]);

sajib\_forDFS(1);

printf("\n");

for(i\_forLoop=1; i\_forLoop<=j\_forNumberOfVartex; i\_forLoop++)

{

if(a\_forQuee[i\_forLoop])

k\_forCount++;

}

if(k\_forCount==j\_forNumberOfVartex)

printf("\n Graph is connected");

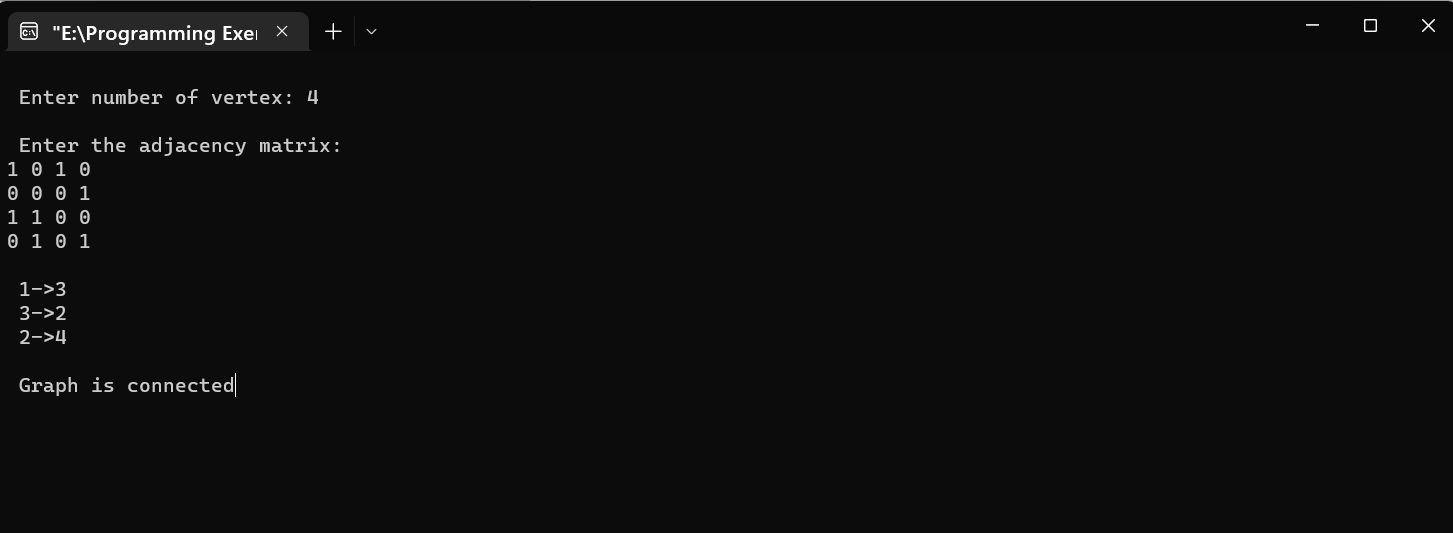
else

printf("\n Graph is not connected");

getch();

}

**Input & Output:**



**3. Algorithm Name: Uniform-Cost Search(UCS).**

**Description:** Uniform-cost search (UCS) is a search algorithm that works on search graphs whose edges do not have the same cost. In the previous examples, we did not mention or define any edge costs. In doing so, we treated every node as having the same cost. The cost of an edge can be interpreted as a value or loss that occurs when that edge is traversed. Mathematically, a cost is just a scalar value associated with some edge, and graphs with non-uniform cost edges tend to have a cost for every edge. Such a graph is known as a **weighted graph.**

Instead of exploring nodes in order of their depth from the root, like what BFS does, UCS expands nodes in order of their cost from the root. At each step, the next step n is chosen to be the one that minimizes a cost value g(n). g(n) is defined as the total cost of getting to a node n from the current position. The nodes are stored in a priority queue.

**Code Implementation using C++:**

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

const int MAX = 100;

int cost[MAX][MAX], dist[MAX], parent[MAX];

bool visited[MAX];

int n;

struct Node {

int vertex;

int distance;

bool operator<(const Node& other) const {

return distance > other.distance;

}

};

void ucs(int start) {

for (int i = 0; i < n; i++) {

dist[i] = INT\_MAX;

visited[i] = false;

}

priority\_queue<Node> q;

dist[start] = 0;

q.push({start, 0});

while (!q.empty()) {

int u = q.top().vertex;

q.pop();

if (visited[u]) {

continue;

}

visited[u] = true;

for (int v = 0; v < n; v++) {

if (cost[u][v] != INT\_MAX && dist[u] + cost[u][v] < dist[v]) {

dist[v] = dist[u] + cost[u][v];

parent[v] = u;

q.push({v, dist[v]});

}

}

}

}

int main() {

cin >> n;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

cin >> cost[i][j];

}

}

int start;

cin >> start;

ucs(start);

for (int i = 0; i < n; i++) {

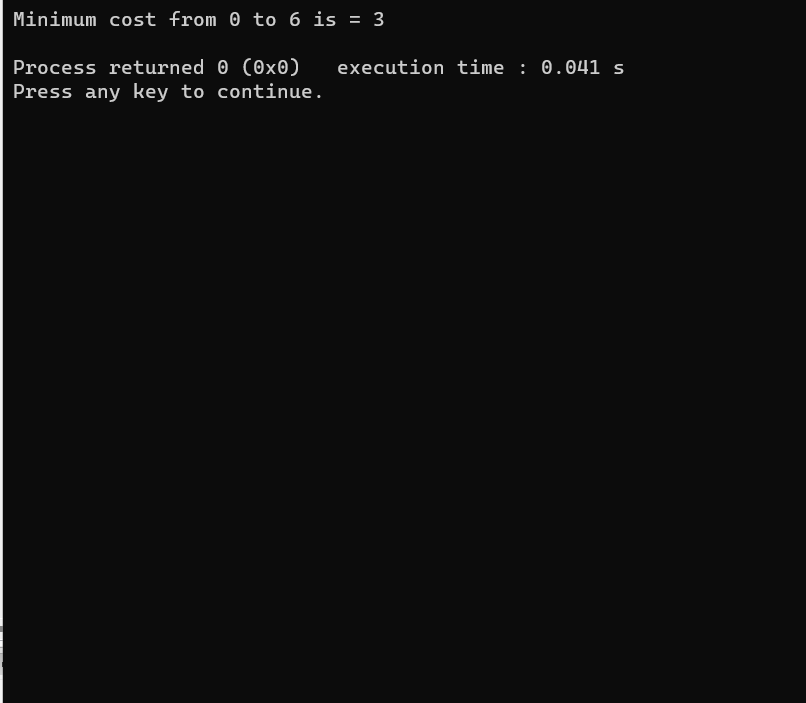
cout << i << " " << dist[i] << endl;

}

return 0;

}

**Input & Output:**



**4. Algorithm Name: Genetic Algorithms(GAs)**

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals from the current population to be parents and uses them to produce children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, nondifferentiable, stochastic, or highly nonlinear. The genetic algorithm can address problems of mixed integer programming, where some components are restricted to be integer-valued.

**Code Implementation using C++:**

#include <bits/stdc++.h>

using namespace std;

#define POPULATION\_SIZE 100

const string nam = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP"\

"QRSTUVWXYZ 1234567890, .-;:\_!\"#%&/()=?@${[]}";

const string Target\_nam= "I am Fatema";

int random\_num(int start, int end)

{

int range = (end-start)+1;

int random\_int = start+(rand()%range);

return random\_int;

}

char mutated\_nam()

{

int len = nam.size();

int r = random\_num(0, len-1);

return nam[r];

}

string create\_gnome()

{

int len = Target\_nam.size();

string gnome = "";

for(int i = 0;i<len;i++)

gnome += mutated\_nam();

return gnome;

}

class Individual

{

public:

string chromosome;

int fitness;

Individual(string chromosome);

Individual mate(Individual parent2);

int cal\_fitness();

};

Individual::Individual(string chromosome)

{

this->chromosome = chromosome;

fitness = cal\_fitness();

};

Individual Individual::mate(Individual par2)

{

string child\_chromosome = "";

int len = chromosome.size();

for(int i = 0;i<len;i++)

{

float p = random\_num(0, 100)/100;

if(p < 0.45)

child\_chromosome += chromosome[i];

else if(p < 0.90)

child\_chromosome += par2.chromosome[i];

else

child\_chromosome += mutated\_nam();

}

return Individual(child\_chromosome);

};

int Individual::cal\_fitness()

{

int len = Target\_nam.size();

int fitness = 0;

for(int i = 0;i<len;i++)

{

if(chromosome[i] != Target\_nam[i])

fitness++;

}

return fitness;

};

bool operator<(const Individual &ind1, const Individual &ind2)

{

return ind1.fitness < ind2.fitness;

}

int main()

{

srand((unsigned)(time(0)));

int generation = 0;

vector<Individual> population;

bool found = false;

for(int i = 0;i<POPULATION\_SIZE;i++)

{

string gnome = create\_gnome();

population.push\_back(Individual(gnome));

}

while(! found)

{

sort(population.begin(), population.end());

if(population[0].fitness <= 0)

{

found = true;

break;

}

vector<Individual> new\_generation;

int s = (10\*POPULATION\_SIZE)/100;

for(int i = 0;i<s;i++)

new\_generation.push\_back(population[i]);

s = (90\*POPULATION\_SIZE)/100;

for(int i = 0;i<s;i++)

{

int len = population.size();

int r = random\_num(0, 50);

Individual parent1 = population[r];

r = random\_num(0, 50);

Individual parent2 = population[r];

Individual offspring = parent1.mate(parent2);

new\_generation.push\_back(offspring);

}

population = new\_generation;

cout<< "Generation: " << generation << "\t";

cout<< "String: "<< population[0].chromosome <<"\t";

cout<< "Fitness: "<< population[0].fitness << "\n";

generation++;

}

cout<< "Generation: " << generation << "\t";

cout<< "String: "<< population[0].chromosome <<"\t";

cout<< "Fitness: "<< population[0].fitness << "\n";

}

**Output:**

Generation: 0 String: /73m!S"t&W/ Fitness: 9

Generation: 1 String: %kf9 nA9ebd Fitness: 9

Generation: 2 String: /73m!S"t&W/ Fitness: 9

Generation: 3 String: %kf9 nA9ebd Fitness: 9

Generation: 4 String: %kfn nA9ebd Fitness: 9

Generation: 5 String: %kf9 nA9ebd Fitness: 9

Generation: 6 String: %kf9 nA9ebd Fitness: 9

Generation: 7 String: %kf9 nA9ebd Fitness: 9

Generation: 8 String: %kf9 nABebd Fitness: 9

Generation: 9 String: %kf9 nA8ebd Fitness: 9

Generation: 10 String: %ka9 nA9ebd Fitness: 8

Generation: 11 String: %ka9 nA9ebd Fitness: 8

Generation: 12 String: %ka9 nA9ebd Fitness: 8

Generation: 13 String: %ka9 nA9ebd Fitness: 8

Generation: 14 String: %ka9 nA9ebd Fitness: 8

Generation: 15 String: %kfm nA9ebd Fitness: 8

Generation: 16 String: %ka9 nA9ebd Fitness: 8

Generation: 17 String: %ka9 nA9eba Fitness: 7

Generation: 18 String: %ka9 nA9eba Fitness: 7

Generation: 19 String: %ka9 nA9eba Fitness: 7

Generation: 20 String: %ka9 nA9eba Fitness: 7

Generation: 21 String: %ka9 nA9eba Fitness: 7

Generation: 22 String: %7a9 nA9eba Fitness: 7

Generation: 23 String: %ka9 nA9eba Fitness: 7

Generation: 24 String: %ka9 nA9eba Fitness: 7

Generation: 25 String: %ka9 nA9eba Fitness: 7

Generation: 26 String: %ka9 nA9eba Fitness: 7

Generation: 27 String: %ka9 nATeba Fitness: 7

Generation: 28 String: %ka9 n89eba Fitness: 7

Generation: 29 String: %kaa nA9eba Fitness: 7

Generation: 30 String: %(aa nA9eba Fitness: 7

Generation: 31 String: %ka9 6A9eba Fitness: 7

Generation: 32 String: Pka9 nA9eNa Fitness: 7

Generation: 33 String: %ka9 n89eba Fitness: 7

Generation: 34 String: %kae nA9eba Fitness: 7

Generation: 35 String: %kaF nA9eba Fitness: 7

Generation: 36 String: %kaa nA9eba Fitness: 7

Generation: 37 String: %kaF nA9eba Fitness: 7

Generation: 38 String: %kar nA9e@a Fitness: 7

Generation: 39 String: ka9 nA9eNa Fitness: 7

Generation: 40 String: %kaZ nA9eba Fitness: 7

Generation: 41 String: %ka9 nA9eba Fitness: 7

Generation: 42 String: Skar nM"eba Fitness: 7

Generation: 43 String: %kax nA9eba Fitness: 7

Generation: 44 String: %kaF nA9eba Fitness: 7

Generation: 45 String: 1kar nA9eMa Fitness: 7

Generation: 46 String: Pka9 nA9eNa Fitness: 7

Generation: 47 String: %kaF nA9eba Fitness: 7

Generation: 48 String: %kaV na9eba Fitness: 6

Generation: 49 String: %kaV na9eba Fitness: 6

Generation: 50 String: %kaV na9eba Fitness: 6

Generation: 51 String: %kaV na9eba Fitness: 6

Generation: 52 String: %kaV na9eba Fitness: 6

Generation: 53 String: %kaV na9eba Fitness: 6

Generation: 54 String: %kaV na9eba Fitness: 6

Generation: 55 String: %kaV na9eba Fitness: 6

Generation: 56 String: %kai na9eba Fitness: 6

Generation: 57 String: %kaV na9eba Fitness: 6

Generation: 58 String: %kaV na9eba Fitness: 6

Generation: 59 String: %maV na9eba Fitness: 6

Generation: 60 String: %kaV na9eba Fitness: 6

Generation: 61 String: %kaV na9eba Fitness: 6

Generation: 62 String: %maV na9eba Fitness: 6

Generation: 63 String: %kaV ha9eca Fitness: 6

Generation: 64 String: %kaV ha9eca Fitness: 6

Generation: 65 String: %kaV na9eba Fitness: 6

Generation: 66 String: %maV na9eba Fitness: 6

Generation: 67 String: %maV na9eba Fitness: 6

Generation: 68 String: %maV haAeba Fitness: 6

Generation: 69 String: %kai na9eYa Fitness: 6

Generation: 70 String: %kaV ha9eca Fitness: 6

Generation: 71 String: %naV nameba Fitness: 6

Generation: 72 String: %kaV na9eba Fitness: 6

Generation: 73 String: %kam na9eba Fitness: 5

Generation: 74 String: %kam na9eba Fitness: 5

Generation: 75 String: %kam na9eba Fitness: 5

Generation: 76 String: %kam na9eba Fitness: 5

Generation: 77 String: %kam na9eba Fitness: 5

Generation: 78 String: %kam na9eba Fitness: 5

Generation: 79 String: %kam na9eba Fitness: 5

Generation: 80 String: %kam na9eba Fitness: 5

Generation: 81 String: %Uam na9eba Fitness: 5

Generation: 82 String: %kam na9eba Fitness: 5

Generation: 83 String: %kam naCeba Fitness: 5

Generation: 84 String: %kam na9eba Fitness: 5

Generation: 85 String: %\_am nateba Fitness: 4

Generation: 86 String: %\_am nateba Fitness: 4

Generation: 87 String: %\_am nateba Fitness: 4

Generation: 88 String: %\_am nateba Fitness: 4

Generation: 89 String: %\_am nateba Fitness: 4

Generation: 90 String: %\_am }ateba Fitness: 4

Generation: 91 String: %\_am nateba Fitness: 4

Generation: 92 String: %\_am natexa Fitness: 4

Generation: 93 String: % am nateba Fitness: 3

Generation: 94 String: % am nateba Fitness: 3

Generation: 95 String: % am nateba Fitness: 3

Generation: 96 String: % am nateba Fitness: 3

Generation: 97 String: % am nateba Fitness: 3

Generation: 98 String: % am nateba Fitness: 3

Generation: 99 String: % am nateba Fitness: 3

Generation: 100 String: % am nateba Fitness: 3

Generation: 101 String: % am nateba Fitness: 3

Generation: 102 String: % am nateba Fitness: 3

Generation: 103 String: % am nateba Fitness: 3

Generation: 104 String: % am nateba Fitness: 3

Generation: 105 String: % am nateba Fitness: 3

Generation: 106 String: % am nateba Fitness: 3

Generation: 107 String: I\_am nateba Fitness: 3

Generation: 108 String: % am bateba Fitness: 3

Generation: 109 String: f am nateba Fitness: 3

Generation: 110 String: f am nateba Fitness: 3

Generation: 111 String: % am Fateba Fitness: 2

Generation: 112 String: % am Fateba Fitness: 2

Generation: 113 String: % am Fateba Fitness: 2

Generation: 114 String: % am Fateba Fitness: 2

Generation: 115 String: % am Fateba Fitness: 2

Generation: 116 String: % am Fateba Fitness: 2

Generation: 117 String: % am Fateba Fitness: 2

Generation: 118 String: % am Fateba Fitness: 2

Generation: 119 String: $ am Fateba Fitness: 2

Generation: 120 String: % am Fateba Fitness: 2

Generation: 121 String: % am Fateba Fitness: 2

Generation: 122 String: % am Fateba Fitness: 2

Generation: 123 String: % am Fateba Fitness: 2

Generation: 124 String: % am Fateba Fitness: 2

Generation: 125 String: % am Fateba Fitness: 2

Generation: 126 String: % am Fateba Fitness: 2

Generation: 127 String: % am Fateba Fitness: 2

Generation: 128 String: % am Fateba Fitness: 2

Generation: 129 String: B am Fateba Fitness: 2

Generation: 130 String: % am Fateba Fitness: 2

Generation: 131 String: % am Fateba Fitness: 2

Generation: 132 String: p am Fatema Fitness: 1

Generation: 133 String: p am Fatema Fitness: 1

Generation: 134 String: p am Fatema Fitness: 1

Generation: 135 String: p am Fatema Fitness: 1

Generation: 136 String: p am Fatema Fitness: 1

Generation: 137 String: p am Fatema Fitness: 1

Generation: 138 String: p am Fatema Fitness: 1

Generation: 139 String: p am Fatema Fitness: 1

Generation: 140 String: p am Fatema Fitness: 1

Generation: 141 String: p am Fatema Fitness: 1

Generation: 142 String: p am Fatema Fitness: 1

Generation: 143 String: p am Fatema Fitness: 1

Generation: 144 String: p am Fatema Fitness: 1

Generation: 145 String: j am Fatema Fitness: 1

Generation: 146 String: Z am Fatema Fitness: 1

Generation: 147 String: p am Fatema Fitness: 1

Generation: 148 String: p am Fatema Fitness: 1

Generation: 149 String: p am Fatema Fitness: 1

Generation: 150 String: Z am Fatema Fitness: 1

Generation: 151 String: p am Fatema Fitness: 1

Generation: 152 String: j am Fatema Fitness: 1

Generation: 153 String: p am Fatema Fitness: 1

Generation: 154 String: j am Fatema Fitness: 1

Generation: 155 String: Z am Fatema Fitness: 1

Generation: 156 String: p am Fatema Fitness: 1

Generation: 157 String: Z am Fatema Fitness: 1

Generation: 158 String: p am Fatema Fitness: 1

Generation: 159 String: j am Fatema Fitness: 1

Generation: 160 String: j am Fatema Fitness: 1

Generation: 161 String: Z am Fatema Fitness: 1

Generation: 162 String: j am Fatema Fitness: 1

Generation: 163 String: Z am Fatema Fitness: 1

Generation: 164 String: p am Fatema Fitness: 1

Generation: 165 String: j am Fatema Fitness: 1

Generation: 166 String: Z am Fatema Fitness: 1

Generation: 167 String: U am Fatema Fitness: 1

Generation: 168 String: j am Fatema Fitness: 1

Generation: 169 String: j am Fatema Fitness: 1

Generation: 170 String: j am Fatema Fitness: 1

Generation: 171 String: j am Fatema Fitness: 1

Generation: 172 String: j am Fatema Fitness: 1

Generation: 173 String: j am Fatema Fitness: 1

Generation: 174 String: p am Fatema Fitness: 1

Generation: 175 String: j am Fatema Fitness: 1

Generation: 176 String: j am Fatema Fitness: 1

Generation: 177 String: j am Fatema Fitness: 1

Generation: 178 String: p am Fatema Fitness: 1

Generation: 179 String: j am Fatema Fitness: 1

Generation: 180 String: i am Fatema Fitness: 1

Generation: 181 String: j am Fatema Fitness: 1

Generation: 182 String: p am Fatema Fitness: 1

Generation: 183 String: j am Fatema Fitness: 1

Generation: 184 String: j am Fatema Fitness: 1

Generation: 185 String: j am Fatema Fitness: 1

Generation: 186 String: j am Fatema Fitness: 1

Generation: 187 String: j am Fatema Fitness: 1

Generation: 188 String: i am Fatema Fitness: 1

Generation: 189 String: i am Fatema Fitness: 1

Generation: 190 String: i am Fatema Fitness: 1

Generation: 191 String: i am Fatema Fitness: 1

Generation: 192 String: i am Fatema Fitness: 1

Generation: 193 String: j am Fatema Fitness: 1

Generation: 194 String: j am Fatema Fitness: 1

Generation: 195 String: i am Fatema Fitness: 1

Generation: 196 String: j am Fatema Fitness: 1

Generation: 197 String: i am Fatema Fitness: 1

Generation: 198 String: I am Fatema Fitness: 0