TUGAS 2 - DESAIN DAN ANALISIS ALGORITMA
S1 INFORMATIKA UNS
SEMESTER GANJIL 2021/2022

Nama: Michael Raditya Krisnadhi
NIM: M0520047
Kelas: B

1. IMPLEMENTASI STRING HASHING MENGGUNAKAN STRATEGI DIVIDE AND CONQUER

Deskripsi

Dalam ilmu kriptografi, fungsi hash adalah fungsi yang memetakan sebuah input dengan sembarang ukuran ke dalam output (hash) yang berukuran tetap. Misalkan f adalah fungsi hash dengan masukan S dan keluaran H sedemikian hingga $f(S) = H$, maka $f^{-1}(H)$ tidak terdefinisi (tidak bisa ditemukan inversnya). Dengan demikian, input asal S dari suatu hash H tidak bisa dicari. Fungsi hash memiliki banyak kegunaan, di antaranya untuk membandingkan file, mencocokkan password, menjaga integritas data, dan lain sebagainya.

Misalkan terdapat fungsi hash $f(S)$ yang terdefinisi sebagai

$$f(S) = \sum(aX_i + b) \bmod m$$

yang dalam hal ini $X_i = i\ S_i$ serta i adalah letak karakter dalam string (indeks berbasis satu sehingga $X_i \neq 0$ untuk $S_i \neq 0$, serta terdapat pengalian dengan letak karakter juga supaya nilai hash juga dipengaruhi oleh letak karakter dalam string), dan a, b, dan m adalah sembarang konstanta. Dengan demikian, fungsi hash tersebut memetakan string S ke dalam bilangan bulat berukuran $\log_2(m)$-bit.

Buatlah algoritma yang merupakan representasi dari $f(S)$ menggunakan strategi Divide and Conquer! Bandingkan juga dengan penyelesaian secara iteratif dengan memerhatikan kompleksitas waktunya!

Penyelesaian

Akan diselesaikan untuk kasus $a = 525046832$, $b = 4039040925$, dan $m = 2^{32}$ sehingga hash H berukuran 32-bit.

Terdapat 3 proses dalam algoritma ini, yaitu:

- **Divide:** bagi string masukan S menjadi dua bagian (substring) sampai panjangnya hanya satu karakter
- **Conquer:** apabila substring sudah berukuran satu karakter, hitung hash untuk substring tersebut menggunakan rumus $(aX_1 + b) \bmod m$
- **Combine:** jumlahkan hash kedua substring bagian (sampai dengan hash seluruh substring dijumlahkan) dalam modulo m

Berikut adalah algoritmanya jika dinyatakan dalam pseudocode:

```
function hash(s: string, left: int, right: int): int
 # Conquer
 if left >= right then
  return (A * (right * s[right]) + B) mod M
 end

 # Divide
 middle := (left + right) / 2
 lhash := hash(left, middle)
 rhash := hash(middle + 1, right)

 # Combine
 return (lhash + rhash) mod M
end
```

Sangat dianjurkan untuk melakukan proses perhitungan dalam integer berukuran lebih dari 32-bit supaya menghindari integer overflow yang berpotensi terjadi pada saat operasi perkalian dan penjumlahan bilangan bulat yang besar.

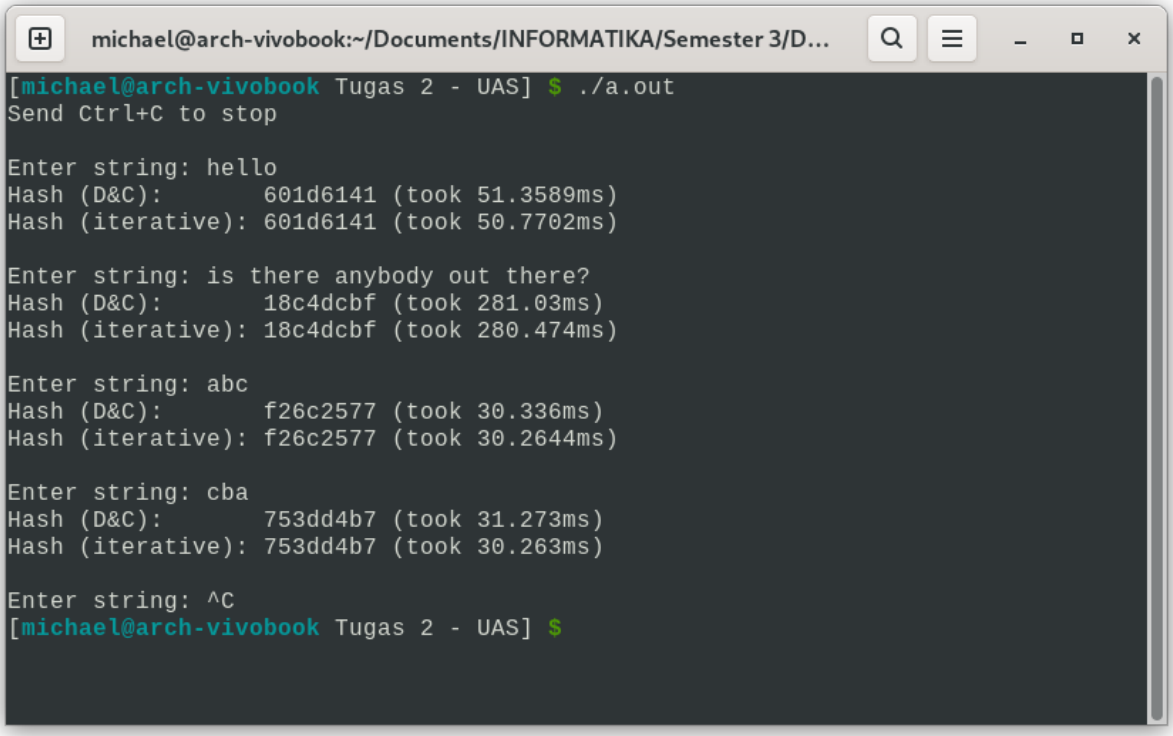
Dengan demikian, nilai dari $f(S)$ dapat dikomputasi dengan pemanggilan `hash(S, 1, len(S))` dengan $len(S)$ adalah panjang string S.

Source Code

```
// Bahasa: C++

#include <iomanip>
#include <iostream>
#include <string>
#include <thread>
#include <chrono>

using namespace std;

#define A   525046832ULL
#define B   4039040925ULL
#define M   0x800000000ULL

unsigned long long __str_hash_dnc(const string &str, size_t left, size_t
right) {
    if (left >= right) {
        // assume (a X[i] + b) mod m costs 10ms
        this_thread::sleep_for(10ms);
        return (A * str[right] * (right + 1) + B) % M;
    }

    size_t middle = (left + right) / 2;

    unsigned long long lhash = (unsigned long long)__str_hash_dnc(str, left,
middle);
    unsigned long long rhash = (unsigned long long)__str_hash_dnc(str,
```

```
middle + 1, right);

    return (lhash + rhash) % M;
}

unsigned int str_hash_dnc(const string &str) {
    return (unsigned int)(__str_hash_dnc(str, 0, str.size() - 1) &
0xFFFFFFFF);
}

unsigned int str_hash_iterative(const string &str) {
    unsigned long long hash = 0;

    for (size_t i = 0; i < str.size(); i++) {
        // assume (a X[i] + b) mod m costs 10ms
        this_thread::sleep_for(10ms);
        hash = (hash + (A * str[i] * (i + 1) + B) % M) % M;
    }

    return (unsigned int)(hash & 0xFFFFFFFF);
}

int main() {
    cout << "Send Ctrl+C to stop" << endl << endl;
    while (true) {
        string str;
        cout << "Enter string: ";
        getline(cin, str);

        unsigned int dnc_hash;
        auto dnc_start = chrono::high_resolution_clock::now();
        dnc_hash = str_hash_dnc(str);
        auto dnc_stop = chrono::high_resolution_clock::now();
        chrono::duration<double> dnc_time = dnc_stop - dnc_start;
        cout << "Hash (D&C):       " << hex << setw(8) << setfill('0') <<
dnc_hash << " (took " << (dnc_time.count() * 1000) << "ms)" << endl;

        unsigned int iterative_hash;
        auto iterative_start = chrono::high_resolution_clock::now();
        iterative_hash = str_hash_iterative(str);
        auto iterative_stop = chrono::high_resolution_clock::now();
        chrono::duration<double> iterative_time = iterative_stop -
iterative_start;
        cout << "Hash (iterative): " << hex << setw(8) << setfill('0') <<
iterative_hash << " (took " << (iterative_time.count() * 1000) << "ms)" <<
endl;

        cout << endl;
    }

    return 0;
}
```

Hasil Program



Kesimpulan

Setelah program dijalankan, terlihat strategi Divide and Conquer tidak mempercepat proses komputasi apabila dibandingkan dengan hasil secara iteratif. Misalkan terdapat string *abcdefgh* yang akan dicari hashnya, berikut adalah skemanya:

<table>
<tr><td colspan="8">abcdefgh</td></tr>
<tr><td colspan="4">abcd</td><td colspan="4">efgh</td></tr>
<tr><td colspan="2">ab</td><td colspan="2">cd</td><td colspan="2">ef</td><td colspan="2">gh</td></tr>
<tr><td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td><td>g</td><td>h</td></tr>
<tr><td>f(a)</td><td>f(b)</td><td>f(c)</td><td>f(d)</td><td>f(e)</td><td>f(f)</td><td>f(g)</td><td>f(h)</td></tr>
<tr><td colspan="2">f(ab)</td><td colspan="2">f(cd)</td><td colspan="2">f(ef)</td><td colspan="2">f(gh)</td></tr>
<tr><td colspan="4">f(abcd)</td><td colspan="4">f(efgh)</td></tr>
<tr><td colspan="8">f(abcdefgh)</td></tr>
</table>

Apabila setiap proses memilki kompleksitas waktu $O(1)$, maka $f(abcdefgh)$ memiliki kompleksitas sebesar $1 + 2 + 4 + ... + 2^d$ di mana d adalah kedalaman operasi rekursi Divide and Conquer untuk menuju ke base case. Dapat diketahui juga bahwa $len(S) = N = 2^d$. Dengan demikian, kompleksitas $f(S)$ dalam hal ini adalah $O(N)$ untuk strategi Divide and Conquer, yang mana sama dengan versi iteratifnya yaitu $O(N)$ juga (karena melakukan proses looping untuk seluruh karakter dalam string). Bisa dipastikan juga bahwa hampir tidak ada perbedaan

waktu dalam hasil output program karena kedua strategi memiliki kompleksitas waktu yang sama.

Dengan demikian, kesimpulan yang dapat diambil adalah strategi Divide and Conquer tidak selalu mempercepat proses komputasi jika dibandingkan solusi interatif biasa (atau solusi naif).

2. MENCARI WAKTU TERCEPAT DAN BAHAN BAKAR MINIMUM PESAWAT RUANG ANGKASA UNTUK MENJELAJAH DARI SUATU SISTEM KE SISTEM LAIN MENGGUNAKAN ALGORITMA SHORTEST PATH

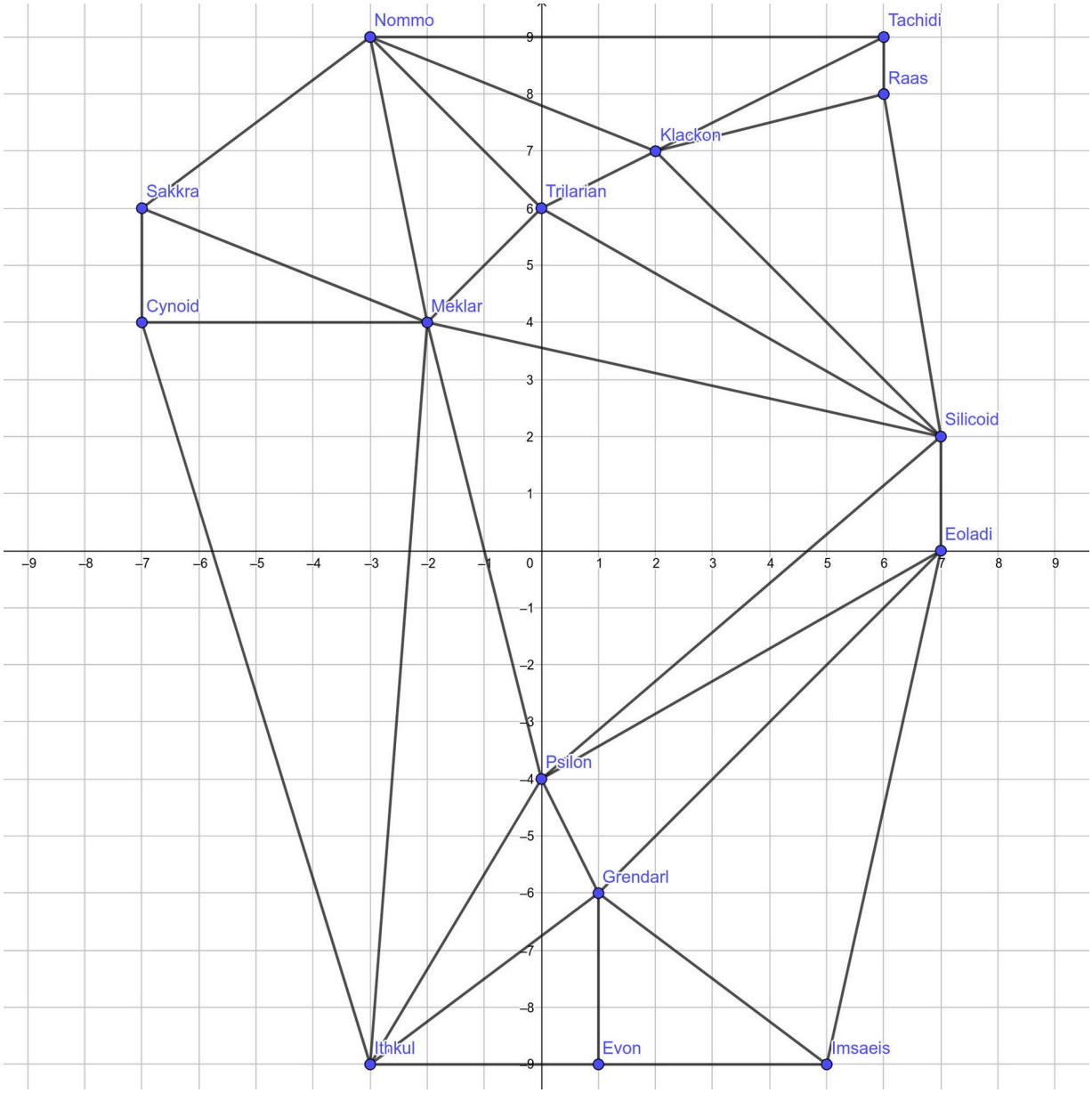
Deskripsi

Terdapat suatu galaksi *G* yang berisi sekumpulan sistem tata surya *S*. Koordinat dari sistem tata surya *S* dapat dinyatakan dalam bentuk kartesius 2-dimensi dalam satuan parsec. Suatu sistem tata surya S_i dapat memiliki rute ke sistem tata surya lain S_j yang dinyatakan dalam garis lurus. Sistem S_i dapat dikatakan memiliki rute ke sistem S_j apabila rute tersebut pernah dilintasi oleh pesawat berjenis scout dan jaraknya (dalam parsec) terukur sekaligus.

Diketahui seluruh sistem dalam galaksi *Orion* beserta koordinatnya:

#	Nama Sistem	Koordinat
1.	Evon	(1, -9)
2.	Psilon	(0, -4)
3.	Meklar	(-2, 4)
4.	Cynoid	(-7, 4)
5.	Sakkra	(-7, 6)
6.	Raas	(6, 8)
7.	Grendarl	(1, -6)
8.	Trilarian	(0, 6)
9.	Nommo	(-3, 9)
10.	Imsaeis	(5, -9)
11.	Eoladi	(7, 0)
12.	Silicoid	(7, 2)
13.	Klackon	(2, 7)
14.	Tachidi	(6, 9)
15.	Ithkul	(-3, -9)

Serta berikut adalah rute-rute yang dikenal:



Untuk sampai dari sistem S_i ke sistem S_j membutuhkan waktu dalam satuan tahun dan bahan bakar hidrogen cair dalam satuan barel.

Carilah rute terpendek sedemikian hingga menghasilkan waktu tercepat dan tentukan juga jumlah barel minimum bahan bakar yang diperlukan oleh pesawat ruang angkasa untuk menjelajah dari sistem P ke sistem Q apabila disediakan kecepatan rata-rata pesawat ruang angkasa sebesar V parsec/tahun dan penggunaan bahan bakar sebanyak F barel/parsec!

Penyelesaian

Untuk menemukan rute terpendek menggunakan algoritma Shortest Path dari Dijkstra yang bekerja secara greedy, dengan pseudocode adalah sebagai berikut:

```
function shortest_path(G: graph, src: vertex, dst: vertex): (real, vertex ->
vertex)
 distances := vertex -> real
 parents := vertex -> vertex
 unvisited := G.V
 for v in G.V do
  distances[v] := INFINITY
  parents[v] := nil
 end

 visit := src
 distances[src] := 0
 while unvisited is not empty do
  unvisited := unvisited - {visit}

  # Hitung total jarak ke vertex tetangga
  for neighbour in (visit.neighbours intersect unvisited) do
   nbdist := G.distance(visit, neighbour)
   if distances[visit] + nbdist < distances[neighbour] then
    distances[neighbour] := distances[visit] + nbdist
    parents[neighbour] := visit
   end
  end

  # Pilih vertex dengan jarak terpendek yang belum dikunjungi
  visit := v in unvisited | min(distances[v])
 end

 return (distances[dst], parents)
end
```

Source Code

```
// Bahasa: C++

#include <cmath>
#include <iostream>
#include <map>
#include <set>
#include <stack>

using namespace std;

typedef string system_t;

struct coord_t {
    double x;
    double y;
};

struct systemdesc_t {
    system_t name;
    coord_t coord;
};
```

```
map<system_t, size_t> indices;
map<size_t, system_t> names;

map<system_t, systemdesc_t> systems;
double adjmatrix[15][15];

double system_distance(system_t a, system_t b) {
    coord_t ca = systems[a].coord;
    coord_t cb = systems[b].coord;
    return sqrt(pow(ca.x - cb.x, 2) + pow(ca.y - cb.y, 2));
}

void add_route(system_t a, system_t b) {
    if ((systems.count(a) == 1) && (systems.count(b) == 1)) {
        adjmatrix[indices[a]][indices[b]] = system_distance(a, b);
    } else {
        throw "system does not exist";
    }
}

double shortest_path(system_t src, system_t dst, map<system_t, system_t>
&parents) {
    if ((systems.count(src) == 0) || (systems.count(dst) == 0)) {
        throw "system does not exist";
    }

    map<system_t, double> distances;
    set<system_t> unvisited;
    for (
        map<system_t, size_t>::iterator it = indices.begin();
        it != indices.end();
        ++it
    ) {
        unvisited.insert(it->first);
        distances[it->first] = INFINITY;
        parents[it->first] = "";
    }

    system_t visit = src;
    distances[src] = 0;
    while (!unvisited.empty()) {
        unvisited.erase(visit);

        for (size_t i = 0; i < 15; i++) {
            if ((adjmatrix[indices[visit]][i] != NAN) &&
(unvisited.count(names[i]) == 1)) {
                double nbdist = adjmatrix[indices[visit]][i];
                if (distances[visit] + nbdist < distances[names[i]]) {
                    distances[names[i]] = distances[visit] + nbdist;
                    parents[names[i]] = visit;
                }
            }
        }

        system_t next = "";
        for (
            set<system_t>::iterator it = unvisited.begin();
            it != unvisited.end();
            ++it
```

```
        ) {
            if (next == "") {
                next = *it;
            } else {
                if (distances[*it] < distances[next]) {
                    next = *it;
                }
            }
        }
        visit = next;
    }

    return distances[dst];
}

void print_route(const map<system_t, system_t> &parents, system_t dst) {
    stack<system_t> stk;

    system_t cur = dst;
    while (cur != "") {
        stk.push(cur);
        cur = parents.at(cur);
    }

    cout << "ROUTE";
    while (!stk.empty()) {
        cout << " -> " << stk.top();
        stk.pop();
    }
    cout << endl;
}

int main(void) {
    double velocity;
    cout << "Enter ship velocity (parsecs/year) : ";
    cin >> velocity;

    double fuel_consumption;
    cout << "Enter ship fuel consumption (barrels/parsec) : ";
    cin >> fuel_consumption;

    systemdesc_t systems_arr[15] = {
        {"Evon", {1, -9}},
        {"Psilon", {0, -4}},
        {"Meklar", {-2, 4}},
        {"Cynoid", {-7, 4}},
        {"Sakkra", {-7, 6}},
        {"Raas", {6, 8}},
        {"Grendarl", {1, -6}},
        {"Trilarian", {0, 6}},
        {"Nommo", {-3, 9}},
        {"Imsaeis", {5, -9}},
        {"Eoladi", {7, 0}},
        {"Silicoid", {7, 2}},
        {"Klackon", {2, 7}},
        {"Tachidi", {6, 9}},
        {"Ithkul", {-3, -9}}
    };
```

```
for (size_t i = 0; i < 15; i++) {
    indices[systems_arr[i].name] = i;
    names[i] = systems_arr[i].name;

    systems[systems_arr[i].name] = systems_arr[i];

    for (size_t j = 0; j < 15; j++) {
        if (i == j) {
            adjmatrix[i][j] = 0;
        } else {
            adjmatrix[i][j] = NAN;
        }
    }
}

// Evon routes
add_route("Evon", "Ithkul");
add_route("Evon", "Grendarl");
add_route("Evon", "Imsaeis");

// Psilon routes
add_route("Psilon", "Ithkul");
add_route("Psilon", "Meklar");
add_route("Psilon", "Grendarl");
add_route("Psilon", "Eoladi");
add_route("Psilon", "Silicoid");

// Meklar routes
add_route("Meklar", "Cynoid");
add_route("Meklar", "Sakkra");
add_route("Meklar", "Ithkul");
add_route("Meklar", "Nommo");
add_route("Meklar", "Psilon");
add_route("Meklar", "Trilarian");
add_route("Meklar", "Silicoid");

// Cynoid routes
add_route("Cynoid", "Sakkra");
add_route("Cynoid", "Ithkul");
add_route("Cynoid", "Meklar");

// Sakkra routes
add_route("Sakkra", "Cynoid");
add_route("Sakkra", "Nommo");
add_route("Sakkra", "Meklar");

// Raas routes
add_route("Raas", "Klackon");
add_route("Raas", "Tachidi");
add_route("Raas", "Silicoid");

// Grendarl routes
add_route("Grendarl", "Ithkul");
add_route("Grendarl", "Psilon");
add_route("Grendarl", "Evon");
add_route("Grendarl", "Imsaeis");
add_route("Grendarl", "Eoladi");

// Trilarian routes
```

```
add_route("Trilarian", "Nommo");
add_route("Trilarian", "Meklar");
add_route("Trilarian", "Klackon");
add_route("Trilarian", "Silicoid");

// Nommo routes
add_route("Nommo", "Sakkra");
add_route("Nommo", "Meklar");
add_route("Nommo", "Trilarian");
add_route("Nommo", "Klackon");
add_route("Nommo", "Tachidi");

// Imsaeis routes
add_route("Imsaeis", "Evon");
add_route("Imsaeis", "Grendarl");
add_route("Imsaeis", "Eoladi");

// Eoladi routes
add_route("Eoladi", "Psilon");
add_route("Eoladi", "Grendarl");
add_route("Eoladi", "Imsaeis");
add_route("Eoladi", "Silicoid");

// Silicoid routes
add_route("Silicoid", "Meklar");
add_route("Silicoid", "Psilon");
add_route("Silicoid", "Trilarian");
add_route("Silicoid", "Klackon");
add_route("Silicoid", "Raas");
add_route("Silicoid", "Eoladi");

// Klackon routes
add_route("Klackon", "Nommo");
add_route("Klackon", "Trilarian");
add_route("Klackon", "Raas");
add_route("Klackon", "Tachidi");
add_route("Klackon", "Silicoid");

// Tachidi routes
add_route("Tachidi", "Nommo");
add_route("Tachidi", "Klackon");
add_route("Tachidi", "Raas");

// Ithkul routes
add_route("Ithkul", "Cynoid");
add_route("Ithkul", "Meklar");
add_route("Ithkul", "Psilon");
add_route("Ithkul", "Evon");
add_route("Ithkul", "Grendarl");

cout << "Send Ctrl+C to stop" << endl << endl;
while (true) {
    try {
        system_t src, dst;
        cout << "Source: ";
        cin >> src;
        cout << "Destination: ";
        cin >> dst;
```

```
            map<system_t, system_t> parents;
            double shortest = shortest_path(src, dst, parents);
            double years = shortest / velocity;
            double barrels = shortest * fuel_consumption;

            print_route(parents, dst);
            cout << "DISTANCE -> " << shortest << " parsecs" << endl;
            cout << "TIME -> " << years << " years" << endl;
            cout << "FUEL -> " << barrels << " barrels" << endl;
        } catch (const char *err) {
            cout << "ERROR: " << err << endl;
        }
        cout << endl;
    }

    return 0;
}
```

Hasil Program