Semestralni rad iz Diskretne matematike

Realizacija Tjuringove mašine

Mateja Milošević, 2018/0177

December 23, 2019

Contents

1 Uvod

Zadatak ovog rada je implementacija Tjuringove Mašine u programskom jeziku C++.

Radi opštosti programa glavna ideja algoritma se sastoji u prevođenju koda pisanog za Tjurinogovu mašinu u graf, gde čvorovi predstavljaju stanja Tjuringove mašine, a grane predstavljaju predstavljaju sledeća stanja, promene trake i pomeraje glave trake.

Algoritam je realizovan objektno-orijentisanom paradigmom i podeljen je u 3 različite klase:

1. *TuringGraph* klasa koja sadži graf koji predstavlja program za Tjuringovu mašinu
2. *TuringMachine* klasa koja predstavlja samu Tjuringovu mašinu, u njoj se sadrži graf koji predstavlja program, i niz karaktera koji predstavlja traku Tjuringove mašine i celobrojnu vrednost koja predstavlja poziciju glave mašine
3. *TuringParser* klasa koja predstavlja prevodioca čija je namena prevođenje koriničkih .txt fajlova u kojima je zapisan program za Tjuringovu mašinu u objekte klase *TuringGraph*, kao i prevođenje .txt fajlova u kojima je zapisano stanje trake i glave Tjuringove mašine

U kodu je dodatno i definisano par klasa, čija je namena isključivo lakša obrada grešaka koje se mogu desiti tokom izvršavanja (klase *Finished* i *InvalidNameException*)

2 *TuringGraph* klasa

U ovoj klasi je opisan rad Tjuringovog programa grafovskim pristupom. Objekat klase sa sastoji od vektora čvorova, a svaki čvor se sastoji od svog identifikatora (q0, q1...) i vektora grana, dok se svaka sastoji od indeksa sledećeg čvora, karaktera koji predstavlja promenu na traci, i celobrojnu vrednost koja predstavlja promenu na traci (u slučaju regularnog programa biće isključivo -1 ili +1).

Objekti klase takođe sadrže i čvorove koji predstavljaju q+ i q- stanja, kao i pokazivač na trenutni čvor, tj trenutno stanje u kom se nalazi Tjuringova mašina.

Klasa je takođe opremljena metodama opšte namene za rad za grafove, poput dodavanja i brisanja čvorova i grana, i provera da li su dva čvora povezana.

Pored takvih metoda, definisane su dodatne metode specfično vezane za rad sa Tjuringovom mašinom, tj. za rad sa dodatnim pokazivačem na trenutno stanje, kao i obedzbeđivanje adekvatne strukture grafa:

- *NextState* metoda koja na osnovu karaktera koji je pročitan sa trake Tjuringove mašine, i trenutnog stanja, prelazi na sledeće stanje, a stanje trake i poziciju glave menja.
- *IsValid* metoda koja proverava da li je graf adekvatno napravljen, tj da li važi da svaki čvor grafa ima jednak broj grana.
- *Reset* metoda koja resetuje program tako što pokazivač na trenutno stanje postavi da pokazuje na stanje 0.
- Operatorska metoda za ispisivanje programa koji se realizuje na osnovu grafa.

Deklaracija i potpis metoda se mogu videti iz .h fajla, dok je implemenetacija sadžana u .cpp fajlu.

Kod 1. *TuringGraph.h*

```
#ifndef _TURING_GRAPH_GUARD_
#define _TURING_GRAPH_GUARD_
#include <vector>
#include <iostream>

class Finished
{
public:
     Finished(bool valid) : valid(valid) { }

     bool GetValid() { return valid; }

private:
     bool valid;
};

class TuringGraph
{
public:
     TuringGraph() = default;
     TuringGraph(const TuringGraph&) = delete;
     TuringGraph(TuringGraph&&) = delete;

     TuringGraph& operator=(const TuringGraph&) = delete;
     TuringGraph& operator=(TuringGraph&&) = delete;

     void AddNode(int q);

     void RemoveNode(int q);

     void AddEdge(int srcNode, char input, char destNode, char change, int move);

     void RemoveEdge(int srcNode, int destNode);

     bool AreConnected(int srcNode, int destNode) const;

     void NextState(char input, char& change, int& move);
```

```
    bool IsValid() const;

    void Destroy();

    void Reset();

    ~TuringGraph();

    friend std::ostream& operator<<(std::ostream& os, const TuringGraph& tg);

private:
    struct Edge
    {
        Edge() { *this = { -1, 'e', 0 }; }
        Edge(int index, char change, int move) : index(index), change(change), move(move) { }

        int index;
        char change;
        int move;
    };

    struct Node
    {
        Node(int q) : q(q) { edges.resize(3); }

        int q;
        std::vector<Edge> edges;

    };

    int convert(char c) const;
    char convert(int i) const;
    static int numberOfChildren(const Node& node);
    void destroy();

    Node posExit{ -1 };
    Node negExit{ -2 };
    std::vector<Node*> nodes;
    Node* curr = nullptr;
};
#endif
```

Kod konkretne implementacije metoda bitno je napomenuti par implementacionih detalja:

- U okviru jednog čvora, pozicija grane kojom treba ići u zavisnosti od pročitane vrednosti sa trake je označena indeksom u vektoru grana (npr. $b \to 0$, $0 \to 1$, $1 \to 2$).
- Kako se q+ i q- čuvaju kao posebni čvorovi (van vektora čvorova) oni imaju posebne indekse i to -1 i -2 respektivno.
- Prilikom dodavanja čvora mora se voditi računa da ne dođe do dodavanja istog čvora više puta
- Kod metode *NextState* vrednosti promene na traci i promene glave se prosleđuju preko reference, i na taj način se menjaju memorijske lokacije na kojima se nalazi ta vrednost i omogućava da klasa graph radi nezavisno od ostalih klasa.
- Kod metode *NextState* ukoliko grana iz datog čvora vodi u čvor sa indeksima -1 ili -2, to znači da je program stigao do q+ ili q- stanja, respektivno, i prijavljivanje da je program završio sa radom je ralizvom mehanizmom bacanja izuzetka *Finished* sa argumentom *true* ili *false* u zavisnosti od stanja sa kojim je završen rad programa. Na taj način se omogućava laka obrada završavanja programa i ispisivanja sa kojim stanjem je program završio sa radom.

Kod 2. *TuringGraph.cpp*

```
#include "TuringGraph.h"
```

```cpp
#include <vector>
#include <iostream>
#include <iomanip>

void TuringGraph::AddNode(int q)
{
     for (unsigned i = 0; i < nodes.size(); i++)
          if (q == nodes[i]->q)
               return;

     Node* temp = new Node(q);
     if (curr == nullptr)
          curr = temp;

     nodes.push_back(temp);
}

void TuringGraph::RemoveNode(int q)
{
     if (q > nodes.size() || q < 0)
          throw std::exception("Node does not exsist!");

     for (unsigned i = 0; i < nodes.size(); i++)
          RemoveEdge(i, q);

     delete nodes[q];
     nodes.erase(nodes.begin() + q);
}

void TuringGraph::AddEdge(int srcNode, char input, char destNode, char change, int move)
{
     if (move > 1 || move < -1)
          throw std::exception("Unvalid move of Turing machine!");

     if (destNode == '-')
          destNode = -2;
     else if (destNode == '+')
          destNode = -1;
     else
          destNode -= '0';

     int index = convert(input);

     nodes[srcNode]->edges[index] = Edge(destNode, change, move);
}

void TuringGraph::RemoveEdge(int srcNode, int destNode)
{
     for (unsigned i = 0; i < nodes[srcNode]->edges.size(); i++)
          if (nodes[srcNode]->edges[i].index == destNode)
               nodes[srcNode]->edges.erase(nodes[srcNode]->edges.begin() + i);
}

bool TuringGraph::AreConnected(int srcNode, int destNode) const
{
     for (unsigned i = 0; i < nodes[srcNode]->edges.size(); i++)
          if (nodes[srcNode]->edges[i].index == destNode)
               return true;
     return false;
}

void TuringGraph::NextState(char input, char& change, int& move)
{
     Edge& temp = curr->edges[convert(input)];
     move += temp.move;
     change = temp.change;

     if (temp.index == -1)
          throw Finished(true);
     if (temp.index == -2)
          throw Finished(false);
```

```
     curr = nodes[temp.index];
}

bool TuringGraph::IsValid() const
{
     int children = numberOfChildren(*nodes[0]);

     for (unsigned i = 1; i < nodes.size(); i++)
     {
          int currChildern = numberOfChildren(*nodes[i]);

          if (currChildern != children)
               return false;
     }

     return true;
}

void TuringGraph::Destroy()
{
     destroy();
}

void TuringGraph::Reset()
{
     curr = nodes[0];
}

TuringGraph::~TuringGraph()
{
     destroy();
}

int TuringGraph::numberOfChildren(const Node& node)
{
     int children = 0;
     for (unsigned i = 0; i < node.edges.size(); i++)
          if (node.edges[i].change != 'e')
               children++;

     return children;
}

int TuringGraph::convert(char c) const
{
     switch (c)
     {
     case 'b':
          return 0;
     case '0':
          return 1;
     case '1':
          return 2;
     case '+':
          return -1;
     case '-':
          return -2;
     default:
          throw std::exception("This should not happen...");
     }
}

char TuringGraph::convert(int i) const
{
     switch (i)
     {
     case 0:
          return 'b';
     case 1:
          return '0';
     case 2:
          return '1';
     default:
```

```
            throw std::exception("This should not happen...");
        }
}

void TuringGraph::destroy()
{
        while (!nodes.empty())
        {
            delete nodes.back();
            nodes.pop_back();
        }

        curr = nullptr;
}

std::ostream& operator<<(std::ostream& os, const TuringGraph& tg)
{
        for (unsigned i = 0; i < tg.nodes.size(); i++)
            for (unsigned j = 0; j < tg.nodes[i]->edges.size(); j++)
            {
                char temp;
                if (tg.nodes[i]->edges[j].index == -2)
                    temp = '-';
                else if (tg.nodes[i]->edges[j].index == -1)
                    temp = '+';
                else
                    temp = tg.nodes[i]->edges[j].index + '0';

                os << "f(q" << i << ',' << tg.convert((int)j) << ")=(q"
                    << temp << ',' << tg.nodes[i]->edges[j].change
                    << ',' << std::setw(2) << std::setfill('+')
                    << tg.nodes[i]->edges[j].move << ')' << std::endl;
            }

        return os;
}
```

3 *TuringMachine* klasa

U ovoj klasi je opisan konkretan rad Tjuringove mašine, tako što se povezuju u jednu celinu program (predstavljen preko objekta klase *TuringGraph*), traka (predstavljena preko vektora karaktera) i glava (celobrojna nenegativna vrednost) Tjuringove mašine.

Za rad sa ovom klasom su definisane metode:

- *Move* metoda, koja pomera traku tako što metodi *NextState* prosledi stanje trake na poziciji glave, memorijsku lokaciju tog stanja, i trenutnu poziciju glave, i na taj način se izvršava jedan korak programa.
- *ExecuteProgram* metoda, koja izvršava program tako što poziva metodu *Move* sve dok se baci izuzetak *Finished*, i tada se resetuje program i kao povratna vrednost se vraća *bool* vrednost koja predstavlja da li je program izašao sa stanjem q- ili q+.
- Operatorska metoda za ispis, koja ispisuje trenutno stanje trake, na kome je obeležena i glava Tjuringove mašine (element nad kojim se nalazi glava je okružen sa | |).

Deklaracija i potpis metoda se mogu videti iz .h fajla, dok je implemenetacija sadžana u .cpp fajlu.

Kod 3. *TuringMachine.h*

```
#ifndef _TURING_MACHINE_GUARD_
#define _TURING_MACHINE_GUARD_
#include "TuringGraph.h"

#include <vector>
#include <iostream>

class TuringMachine
{
public:
     TuringMachine(TuringGraph& graph, std::vector<char> tape, int position = 2);

     bool ExecuteProgram(std::ostream& os = std::cout);

     void Move(std::ostream& os = std::cout);

     friend std::ostream& operator<<(std::ostream& os, const TuringMachine& tm);

private:
     TuringGraph& graph;

     std::vector<char> tape;
     int pos = 0;
};
#endif
```

Kod 4. *TuringMachine.cpp*

```
#include "TuringMachine.h"

#include <iostream>
#include <vector>

TuringMachine::TuringMachine(TuringGraph& graph, std::vector<char> tape, int position) : graph(graph), tape(tape), pos(position)
{
}

bool TuringMachine::ExecuteProgram(std::ostream& os)
{
     int i = 0;
     while (true)
     {
          try
          {
               os << "Step " << i++ << ": ";
               Move(os);
```

```
                system("pause");
            }
            catch (Finished & f)
            {
                os << "End state: " << *this << std::endl;
                graph.Reset();
                return f.GetValid();
            }
        }
}

void TuringMachine::Move(std::ostream& os)
{
    os << *this << std::endl;

    graph.NextState(tape[pos], tape[pos], pos);
}

std::ostream& operator<<(std::ostream& os, const TuringMachine& tm)
{
    os << std::endl;
    for (unsigned i = 0; i < tm.tape.size() * 3; i++)
        os << '-';
    os << std::endl;

    for (unsigned i = 0; i < tm.tape.size(); i++)
        if (i == tm.pos)
            os << '|' << tm.tape[i] << '|';
        else
            os << ' ' << tm.tape[i] << ' ';

    os << std::endl;
    for (unsigned i = 0; i < tm.tape.size() * 3; i++)
        os << '-';

    return os;
}
```

4 *TuringParser* klasa

Kako je potrebno program učitati iz tekstualnog fajla i prebaciti korisnički napisan kod u odgovarajuću grafovsku strukturu, definisane su metode klase *TuringParser*:

- *Compile* metoda, koja na osnovu imena fajla zadatom u argumentu, otvara fajl i iz njega čita program napisan za Tjuringovu mašinu, na osnovu oznaka iz korisnički napisanog tekstualnog fajla se dodaju čvorovi i grane u graf i na taj način se vrši prevođenje datog programa. Metoda ima mogućnosti prijavljivanja različitih vrsta grešaka koje korisnik može napraviti.
- *GetInput* metoda, koja na osnovu imena fajla zadatom u argumentu, otvara fajl i iz njega čita prvu liniju koja treba da sadrži početno stanje trake Tjuringove mašine. Metoda ima mogućnosti prijavljivanja različitih vrsta grešaka koje korisnik može napraviti.
- *GetPosition* metoda, koja na osnovu imena fajla zadatom u argumentu, otvara fajl i iz njega čita drugu liniju koja sadrži početnu poziciju glave Tjuringove mašine. Metoda ima mogućnosti prijavljivanja različitih vrsta grešaka koje korisnik može napraviti.

Deklaracija i potpis metoda se mogu videti iz .h fajla, dok je implemenetacija sadžana u .cpp fajlu.

Kod 5. *TuringParser.h*

```
#ifndef  _TURING_PARSER_GUARD_
#define _TURING_PARSER_GUARD

#include "TuringGraph.h"

#include <string>
#include <vector>

class InvalidNameException
{
     friend std::ostream& operator<<(std::ostream& os, const InvalidNameException ex)
     {
          return os << "File with that name does not exsist!";
     }
};

class TuringParser
{
public:
     TuringGraph& Compile(const std::string& programFile);

     std::vector<char>& GetInput(const std::string& inputFile);

     int GetPosition(const std::string& inputFile);

private:
     TuringGraph program;
};
#endif
```

Kod konkretne implementacije metoda bitno je napomenuti par implementacionih detalja:

- U metodi *Compile* za čitanje iz datoteke korišćena je funkcija *getline*, radi čitanja .txt fajla liniju po liniju, i C-ovska funkcija *sscanf* iz razloga što su sve komande istog formata, i na osnovu *format string*-a se u jednoj liniji koda izvlače potrebne informacije iz fajla (stanje, ulaz, sledeće stanje, promena na traci, i promena pozicije glave). Takođe, povratne vrednost funkcije i broj učitanih parametara i u slučaju da nije učitano svih 5 parametara, ili u slučaju da graf nije adekvatno povezan metoda *Compile* će prijaviti grešku.
- U metodi *Compile* za čitanje iz datoteke korišćena je funkcija *getline*, radi čitanja prve linije .txt fajla, nakon toga se prolazi kroz celu tu liniju i čita se karakter po karakter i smešta se u vektor karaktera,

ukoliko se desi da se pročita neki karakter koji je različit od dozvoljenih metoda će prijaviti grešku. Takođe, greška se dešava ukoliko korisnik u svom fajlu ne navede početno stanje trake, kao i ukoliko se traka ne završava sa praznim simbolom **b** (kako je traka u teoriji beskonačna, u realnom okruženju se mora navesti njen kraj).

Kod 6. *TuringParser.cpp*

```
#include "TuringParser.h"
#include "TuringGraph.h"

#include <string>
#include <vector>
#include <fstream>
#include <sstream>
#include <cstdio>

TuringGraph& TuringParser::Compile(const std::string& programName)
{
    program.Destroy();

    std::ifstream programInput(programName);
    if (!programInput.is_open())
        throw InvalidNameException();

    std::string line;
    while (std::getline(programInput, line))
    {
        int q, move;
        char input, qNext, change;
        int result = sscanf(line.c_str(), "f(q %d, %c) = (q %c, %c, %d)", &q, &input, &qNext, &change, &move);
        if (result != 5)
            throw std::exception("Compile error: Invalid program format!");

        program.AddNode(q);
        program.AddEdge(q, input, qNext, change, move);
    }

    if (!program.IsValid())
        throw std::exception("Compile error: Not all inputs specified!");

    programInput.close();
    return program;
}

std::vector<char>& TuringParser::GetInput(const std::string& inputName)
{
    std::ifstream input(inputName);
    if (!input.is_open())
        throw InvalidNameException();

    std::string buffer;
    std::getline(input, buffer);

    std::vector<char> temp;
    std::istringstream iss(buffer);
    char c;
    while (iss >> c)
    {
        if (c != '0' && c != '1' && c != 'b')
            throw std::exception("Invalid input: Tape has invalid symbols!");
        temp.push_back(c);
    }

    if (temp.size() == 0)
        throw std::exception("Invalid input: Tape cant be empty!");
    if (temp.back() != 'b')
        throw std::exception("Invalid input: Tape must end with b character!");

    input.close();
    return *(new std::vector<char>(temp));
}

```

```
int TuringParser::GetPosition(const std::string& inputFile)
{
     std::ifstream input(inputFile);
     if (!input.is_open())
          throw InvalidNameException();

     std::string buffer;
     std::getline(input, buffer);

     int position = -1;
     input >> position;

     if (position == -1)
          throw std::exception("Invalid input: Initial position of head is not specified!");

     return position;
}
```

5 *main* funkcija

Program započinje svoje izvršavanje ulaskom u *main* funkciju, u njoj je implementiran korisnički interfejs, tj. meni. Stavke menija se biraju unošenjem odgovarajućih brojeva na standardni ulaz, a te stavke su:

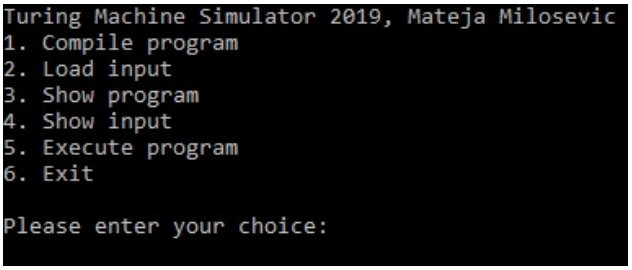

Figure 1: Izgled menija programa

1. *Compile program* stavka, izborom ove stavke se od korisnika zahteva da unese ime svog .txt fajla u kome mu se nalazi program, ukoliko korisnik pogrešno unese ime fajla, zahtevaće se ponovni unos imena fajla, a ukoliko je korisnik pogrešno napisao fajl, program će se prekinuti i prijaviće se greška.

2. *Load input* stavka, izborom ove stavke se od korisnika zahteva da unese ime svog .txt fajla u kome mu se nalaze ulazn podaci (stanje trake i pozicija glave), ukoliko korisnik pogrešno unese ime fajla, zahtevaće se ponovni unos imena fajla, a ukoliko je korisnik pogrešno napisao fajl, program će se prekinuti i prijaviće se greška.

3. *Show program* stavka, izborom ove stavke će se, ukoliko postoji definisan, ispisati trenutno učitan program.

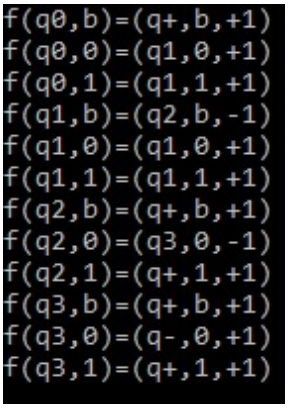

Figure 2: Izgled ispisanog programa za Tjuringovu mašinu

4. *Show input* stavka, izborom ove stavke će se, ukoliko postoji definisano, ispisati trenutno učitano stanje trake.

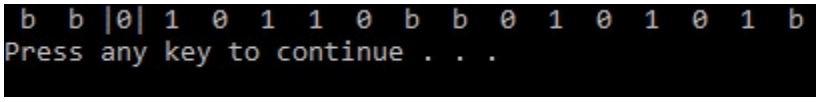

Figure 3: Izgled ispisanog stanja trake Tjuringove mašine

5. *Execute program* stavka, izborom ove stavke će se, ukoliko je definisano stanje trake i program, kreirati objekat klase *TuringMachine* na osnovu napravljenog grafa, stanja trake i pozicije glave, i pokrenuće se metoda *ExecuteProgram* koja će ispisivati šta se dešava sa trakom korak po korak, sve dok se ne završi program, nakon čega će se ispisati sa kojim stanjem je završen program.

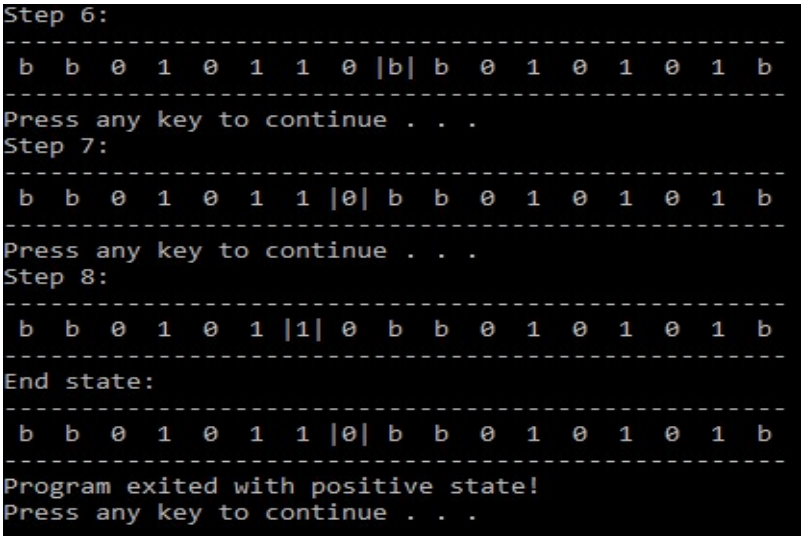

Figure 4: Izgled dela ispisanog programa za Tjuringovu mašinu

6. *Eixt* stavka, izborom ove stavke će se oslobodi svi zauzeti resursi i izaći će se iz programa.

Kod 7. *main.cpp*

```
#include <iostream>
#include "TuringGraph.h"
#include "TuringMachine.h"
#include "TuringParser.h"

using namespace std;

int main()
{
    try
    {
        TuringParser tp; TuringGraph* tg = nullptr; vector<char>* tape = nullptr; int position = -1; bool finished = false;
        while (!finished)
        {
            system("cls");
            cout << "Turing Machine Simulator 2019, Mateja Milosevic" << endl
                << "1. Compile program" << endl
                << "2. Load input" << endl
                << "3. Show program" << endl
                << "4. Show input" << endl
                << "5. Execute program" << endl
                << "6. Exit" << endl << endl;

            cout << "Please enter your choice: ";
            int choice;
            cin >> choice;
            system("cls");

            switch (choice)
            {
            case 1:
            {
                string fileName;
                bool loaded = false;
```

```
35                    while (!loaded)
                      {
                           cout << "Enter program file name: ";
                           cin >> fileName;

40                         loaded = true;
                           try
                           {
                                tg = &tp.Compile(fileName);
                           }
45                         catch (InvalidNameException & e)
                           {
                                loaded = false;
                                cout << e << endl;
                           }
50                    }
                      cout << "Program successfully loaded!" << endl;
                      break;
                 }
                 case 2:
55               {
                      string inputName;
                      delete tape;

                      bool loaded = false;
60                    while (!loaded)
                      {
                           cout << "Enter input file name: ";
                           cin >> inputName;
                           loaded = true;
65                         try
                           {
                                tape = &tp.GetInput(inputName);
                                position = tp.GetPosition(inputName);
                           }
70                         catch (InvalidNameException & e)
                           {
                                loaded = false;
                                cout << e << endl;
                           }
75                    }

                      cout << "Input successfully loaded!" << endl;
                      break;
                 }
80               case 3:
                      if (tg == nullptr)
                      {
                           cout << "Program not loaded yet!" << endl;
                           system("pause");
85                         continue;
                      }

                      cout << *tg << endl;
                      break;
90               case 4:
                      if (tape == nullptr || position == -1)
                      {
                           cout << "Input not loaded yet!" << endl;
                           system("pause");
95                         continue;
                      }

                      for (unsigned i = 0; i < tape->size(); i++)
                           if (i == position)
100                             cout << '|' << (*tape)[i] << '|';
                           else
                                cout << ' ' << (*tape)[i] << ' ';
                      cout << endl;
                      break;
105              case 5:
                 {
                      if (tg == nullptr)
```

```
                    {
                        cout << "Program not loaded yet" << endl;
                        system("pause");
                        continue;
                    }
                    if (tape == nullptr || position == -1)
                    {
                        cout << "Input not loaded yet!" << endl;
                        system("pause");
                        continue;
                    }

                    TuringMachine tm(*tg, *tape, position);
                    bool succ = tm.ExecuteProgram();
                    if (succ)
                        cout << "Program exited with positive state!" << endl;
                    else
                        cout << "Program exited with negative state!" << endl;
                    break;
                }
                case 6:
                    delete tape;
                    cout << "Goodbye!" << endl;
                    finished = true;
                    break;
                default:
                    cout << "Invalid choice: Please enter a number from the menu!" << endl;
                    break;
                }

                system("pause");
            }
        }
        catch (exception & e)
        {
            cout << e.what() << endl;
            system("pause");
        }

        return 0;
}
```

6 Uputstvo za korišćenje programa

Program se koristi tako što otvori odgovarajući .exe fajl, gde se može videti prethodno objašnjen meni, koji se koristi na već opisan način.

Korisnik mora napisati 2 svoja .txt fajlova koji predstavljaju program i početno stanje trake Tjuringove mašine i to u formatu:

- Programski .txt fajl je niz instrukcija posebnog formata. Format jedne instrukcije u progrmskom .txt fajlu izgleda ovako:

$$f(q_n, i) = (q_{next}, c, m)$$

, gde je:

  - q_n stanje u kojem se trenutno nalaz Tjurinogva mašina
  - i argument pročitan sa trake Tjuringove mašine
  - q_{next} stanje u koje treba Tjuringova mašina da pređe
  - c promena koja treba da se izvrši na trenutnoj poziciji glava na traci Tjuringove mašine
  - m pomeraj koji glava Tjuringove mašine treba da napravi

```
1   f(q0,0)=(q1,0,+1)
2   f(q0,b)=(q-,b,+1)
3   f(q0,1)=(q1,1,+1)
4   f(q1,b)=(q2,b,-1)
5   f(q1,1)=(q1,1,+1)
6   f(q1,0)=(q1,0,+1)
7   f(q2,0)=(q3,0,-1)
8   f(q2,b)=(q-,b,+1)
9   f(q2,1)=(q-,1,+1)
10  f(q3,b)=(q-,b,+1)
11  f(q3,0)=(q+,0,+1)
12  f(q3,1)=(q-,1,+1)
```

Figure 5: Izgled programskog .txt fajla

- Ulazni .txt fajl se sastoji iz dve linije, u prvoj liniji se nalazi niz karaktera koji predstavlja početno stanje trake Tjuringove mašine, dok se u drugoj liniji nalazi početna pozicija glave Tjuringove mašine

```
1   bb010110bb010101b
2   2
```

Figure 6: Izgled ulaznog .txt fajla

Nakon definisanja ova 2 .txt fajlova, korisnik ih može učitati preko menija u svoj program i pokrenuti napisan program, ukoliko su oba fajla napisana u navedenom formatu.

Bitno je napomenuti da su fajlovi nezavisni jedan od drugog, tj. može se jedan isti program primenjivati na različitim ulazima, a važi i obrnuto.