Osnovi računarskih sistema

Milena Vujošević–Janičić

Čas 1

Vežbe 01 03 2005

1.1 Lavirint—zadatak

Zadatak 1 *Lavirint je zapisan u datoteci "lavirint.dat" na sledeći način:*
- u prvom redu zapisuju se praznim prostorom razdvojeni celi brojevi koji predstavljaju širinu i visinu lavirinta;
- zatim sledi onoliko redova kolika je visina lavirinta, a u svakom redu po onoliko znakova kolika je širina lavirinta;
- znak 'X' oznaka za zid, ' ' za put, 'O' za izlaz (ili cilj), a '@' za ulaz (ili početnu poziciju);
- mora postojati bar jedan ulaz i bar jedan izlaz.

Primer:

```
10 5
XXXXXXOXXX
X      X X
X XX XXX X
X        X
XXX@XXXXXX
```

Napisati program koji čita lavirint, proverava da li je ispravno zapisan, pronalazi najkraći put od ulaza do izlaza i prikazuje na standardonm izlazu najkraći put ucrtan u lavirint upotrebom znaka '.', kao u narednom primeru:

```
10 5
XXXXXXOXXX
X......X X
X.XXXXXX X
X.... X
XXXX@XXXXX
```

1.2 Lavirint—rešenje

Na osnovu teksta zadatka, jasno je da je potrebno napraviti klasu Lavirint i u njoj obezbediti metod za pronalaženje najkraćeg puta. Takođe, potrebno je i predefinisati operatore << i >> kako bi omogućili jednostavno učitavanje i ispisivanje lavirinta. Na osnovu ovih zaključaka moguće je napisati main funkciju:

```
main()
{
    Lavirint l;
    ifstream f("lavirint.dat");
    f >> l;
    l.PronalazenjeNajkracegPuta();
    cout << l;
    return 0;
}
```

Operatore << i >> implementiraćemo na uobičajni način, koristeći metode Citaj i Pisi.

```
istream& operator >> ( istream& istr, Lavirint& l )
{
    l.Citaj(istr);
    return istr;
}

ostream& operator << ( ostream& ostr, const Lavirint& l )
{
    l.Pisi(ostr);
    return ostr;
}
```

Sada je neophodno da razmišljamo o strukturi naše klase Lavirint i o tome šta sve ona treba da sadrži. Dakle, neophodno je naći ideju kako formirati najkraći put u lavirintu.

Prvo, jasno je da nam je potrebna nekakva matrica u kojoj ćemo čuvati ulaz iz datoteke. Recimo da to može da bude niz nizova karaktera (rešenje u programskom jeziku C) odnosno vektor čiji su elementi vektori karaktera. Neka se ta matrica zove _mapa.

Recimo da smo pronašli najkraći put, gde će se on čuvati? Taj put možemo da smestimo u matricu čiji su elementi tipa bool i za svaki element važi da je true ako pripada putu odnosno da je false ako putu ne pripada.

Na osnovu ove dve matrice možemo da uradimo štampanje rezultata, ako je neki element matrice pripada putu onda se štampa tačka, ako ne pripada onda se štampa ono što je inače u mapi.

Potrebno je čuvati i koordinate početne pozicije u lavirntu kao i podatak da li je put pronađen (ne mora svaki lavirint da ima put do izlaza).

Šta dalje?

Pretpostavimo da je nula početna pozicija i da nemamo zidova. Posmatrajmo sledeću sliku i kretanje iz svake tačke na sve četiri strane bez vraćanja.

```
        3
      3 2 3
    3 2 1 2 3
  3 2 1 0 1 2 3
    3 2 1 2 3
      3 2 3
        3
```

Da bismo obezbedili kretanje bez vraćanja potrebno je negde da čuvamo podatke o poljima na kojima smo već bili. Za to nam je potrebna i treća matrica.

Pošto su nam potrebne tri matrice, nameće se da bi bilo dobro napraviti šablon klase matrica i onda upotrebljavati taj šablon kako bismo imali udobniji rad i pristup članovima matrice.

```
template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v, T t )
        : _podaci(s)
        {
        // proveru da su s i v > 0 ostavljamo za drugu priliku
        for( unsigned i=0; i<s; i++ ){
            // _podaci[i] = vector<T>(v);
            _podaci[i].resize(v);
            for( unsigned j=0; j<v; j++ )
                _podaci[i][j] = t;
            }
        }

    unsigned Sirina() const
        { return _podaci.size(); }

    unsigned Visina() const
        { return _podaci[0].size(); }

    vector<T>& operator[]( unsigned i )
```

```
        { return _podaci[i]; }

    const vector<T>& operator[]( unsigned i ) const
        { return _podaci[i]; }

private:
    vector< vector<T> > _podaci;
};
```

Sada znamo neke privatne podatke klase Lavirint

```
private:
    Matrica<char> _mapa;
    Matrica<bool> _obradjeno;
    Matrica<bool> _put;
    unsigned xPocetka, yPocetka;
    bool pronadjenPut;
```

i možemo da napišemo metode Pisi i Citaj.

```
void Citaj( istream& istr )
    {
    unsigned s,v;
    istr >> s >> v;
    _mapa = Matrica<char>( s, v, ' ' );
    for( unsigned i=0; i<v; i++ ){
        istr >> ws;
        for( unsigned j=0; j<s; j++ )
            _mapa[j][i] = istr.get();
        }
    }

void Pisi( ostream& ostr ) const
    {
    unsigned
        s = _mapa.Sirina(),
        v = _mapa.Visina();
    ostr << s << ' ' << v << endl;
    for( unsigned i=0; i<v; i++ ){
        for( unsigned j=0; j<s; j++ )
            if( pronadjenPut && _put[j][i] )
                ostr << '.';
            else
                ostr << _mapa[j][i];
        ostr << endl;
        }
    }
```

Takođe, možemo da implementiramo konstruktor bez argumenata klase Lavirint.

```
Lavirint()
    : pronadjenPut(false)
    {}
```

Do sada smo "sve" uradili osim implemetacije algoritma pronalaženja najkraćeg puta. Držaćemo se ideje odlaska na sve četiri strane onda kada je to moguće. Kada stignemo do izlaza pronašli smo put, ali u međuvremenu moramo nekako da čuvamo kuda smo išli i odakle smo došli. Te podatke stavljaćemo u jedan niz čiji će argumenti biti koordinata tačke u koju smo došli i indeks tačke u nizu iz koje smo došli. Ta tri podatka apstrahovaćemo u klasu Pozicija koju ćemo definisati u okviru klase Lavirint.

```
class Pozicija {
    public:
        unsigned x,y;
        int prethodna;
        Pozicija( unsigned _x, unsigned _y, int _p )
            : x(_x), y(_y), prethodna(_p)
            {}
};
```

Dakle, u okviru klase Lavirint pamtimo još jedan podatak koji će da čuva putanju.

```
    vector<Pozicija> putanja;
```

U tu putanju, prvo ubacujemo poziciju početne tačke. Koja je to pozicija? Moramo je pronaći na osnovu mape.

```
void PronalazenjePocetaka()
    {
    unsigned
        s = _mapa.Sirina(),
        v = _mapa.Visina();
    for( unsigned i=0; i<v; i++ )
        for( unsigned j=0; j<s; j++ )
            if( _mapa[j][i] == '@' )
                putanja.push_back( Pozicija(j,i,-1) );
    }
```

Pre nego što neko polje ubacimo u putanju moramo da proverimo da ono nije već obrađeno, da kroz njega može da se prođe ili da li je to polje koje označava izlaz.

```
void proveraPolja( unsigned x, unsigned y, int prethodno )
    {
    if( _mapa[x][y] == 'O' ){
        pronadjenPut = true;
```

```
        }
    else if( _mapa[x][y] == ' ' && !_obradjeno[x][y] ){
        putanja.push_back( Pozicija(x,y,prethodno) );
        _obradjeno[x][y] = true;
        }
    }
```

Kako pronaći najkraći put? Treba implementirati ideju sa odlaskom na sve četiri strane tj na one koje je moguće otići. Put kojim idemo pamtimo u nizu putanja.

```
void PronalazenjeNajkracegPuta()
    {
    // pocinjemo trazenje
    putanja.clear();
    PronalazenjePocetaka();

    unsigned tekuca = 0;
    pronadjenPut = false;
    _obradjeno = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );

    // trazimo
    while( tekuca < putanja.size() && !pronadjenPut ){
        if( putanja[tekuca].y > 0 )
            proveraPolja( putanja[tekuca].x, putanja[tekuca].y-1,tekuca);
        if( putanja[tekuca].y < _mapa.Visina()-1 )
            proveraPolja( putanja[tekuca].x, putanja[tekuca].y+1,tekuca);
        if( putanja[tekuca].x > 0 )
            proveraPolja( putanja[tekuca].x-1, putanja[tekuca].y,tekuca);
        if( putanja[tekuca].x < _mapa.Sirina()-1 )
            proveraPolja( putanja[tekuca].x+1, putanja[tekuca].y,tekuca);
        tekuca++;
        }

    if( pronadjenPut )
        RestauracijaPutanje( tekuca-1 );
    }
```

Na osnovu formiranje putanje moramo da rekonstruišemo najkraći put tj put kojim smo došli do izlaza.

```
void RestauracijaPutanje( int poslednje )
    {
    _put = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );
    for( int i=poslednje; i>0; i=putanja[i].prethodna )
```

```
        _put[putanja[i].x][putanja[i].y] = true;
    }
```

Kompletno rešenje:

```
#include <fstream>
#include <iostream>
#include <vector>

using namespace std;

template <class T>
class Matrica
{
public:
    Matrica()
        {}

    Matrica( unsigned s, unsigned v, T t )
        : _podaci(s)
        {
        // proveru da su s i v > 0 ostavljamo za drugu priliku
        for( unsigned i=0; i<s; i++ ){
            // _podaci[i] = vector<T>(v);
            _podaci[i].resize(v);
            for( unsigned j=0; j<v; j++ )
                _podaci[i][j] = t;
            }
        }

    unsigned Sirina() const
        { return _podaci.size(); }

    unsigned Visina() const
        { return _podaci[0].size(); }

    vector<T>& operator[]( unsigned i )
        { return _podaci[i]; }

    const vector<T>& operator[]( unsigned i ) const
        { return _podaci[i]; }

private:
    vector< vector<T> > _podaci;
};
```

```
class Lavirint
{
public:
    class Pozicija {
        public:
            unsigned x,y;
            int prethodna;
            Pozicija( unsigned _x, unsigned _y, int _p )
                : x(_x), y(_y), prethodna(_p)
                {}
    };

    Lavirint()
        : pronadjenPut(false)
        {}

    void PronalazenjePocetaka()
        {
        unsigned
            s = _mapa.Sirina(),
            v = _mapa.Visina();
        for( unsigned i=0; i<v; i++ )
            for( unsigned j=0; j<s; j++ )
                if( _mapa[j][i] == '@' )
                    putanja.push_back( Pozicija(j,i,-1) );
        }

    void PronalazenjeNajkracegPuta()
        {
        // pocinjemo trazenje
        putanja.clear();
        PronalazenjePocetaka();

        unsigned tekuca = 0;
        pronadjenPut = false;
        _obradjeno = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );

        // trazimo
        while( tekuca < putanja.size() && !pronadjenPut ){
            if( putanja[tekuca].y > 0 )
                proveraPolja( putanja[tekuca].x, putanja[tekuca].y-1,tekuca);
            if( putanja[tekuca].y < _mapa.Visina()-1 )
                proveraPolja( putanja[tekuca].x, putanja[tekuca].y+1,tekuca);
```

```
        if( putanja[tekuca].x > 0 )
            proveraPolja( putanja[tekuca].x-1, putanja[tekuca].y,tekuca);
        if( putanja[tekuca].x < _mapa.Sirina()-1 )
            proveraPolja( putanja[tekuca].x+1, putanja[tekuca].y,tekuca);
        tekuca++;
        }

    if( pronadjenPut )
        RestauracijaPutanje( tekuca-1 );
    }

void RestauracijaPutanje( int poslednje )
    {
    _put = Matrica<bool>( _mapa.Sirina(), _mapa.Visina(), false );
    for( int i=poslednje; i>0; i=putanja[i].prethodna )
        _put[putanja[i].x][putanja[i].y] = true;
    }

void proveraPolja( unsigned x, unsigned y, int prethodno )
    {
    if( _mapa[x][y] == 'O' ){
        pronadjenPut = true;
        }
    else if( _mapa[x][y] == ' ' && !_obradjeno[x][y] ){
        putanja.push_back( Pozicija(x,y,prethodno) );
        _obradjeno[x][y] = true;
        }
    }

void Citaj( istream& istr )
    {
    unsigned s,v;
    istr >> s >> v;
    _mapa = Matrica<char>( s, v, ' ' );
    for( unsigned i=0; i<v; i++ ){
        istr >> ws;
        for( unsigned j=0; j<s; j++ )
            _mapa[j][i] = istr.get();
        }
    }

void Pisi( ostream& ostr ) const
    {
    unsigned
```

```
            s = _mapa.Sirina(),
            v = _mapa.Visina();
        ostr << s << ' ' << v << endl;
        for( unsigned i=0; i<v; i++ ){
            for( unsigned j=0; j<s; j++ )
                if( pronadjenPut && _put[j][i] )
                    ostr << '.';
                else
                    ostr << _mapa[j][i];
            ostr << endl;
            }
        }

private:
    Matrica<char> _mapa;
    Matrica<bool> _obradjeno;
    Matrica<bool> _put;
    vector<Pozicija> putanja;
    unsigned xPocetka, yPocetka;
    bool pronadjenPut;
};

istream& operator >> ( istream& istr, Lavirint& l )
{
    l.Citaj(istr);
    return istr;
}

ostream& operator << ( ostream& ostr, const Lavirint& l )
{
    l.Pisi(ostr);
    return ostr;
}

main()
{
    Lavirint l;
    ifstream f("lavirint.dat");
    f >> l;
    l.PronalazenjeNajkracegPuta();
    cout << l;
    return 0;
}
```

Sadržaj