Grundyeva igra

Iva Tutiš

1. Opis Grundyeve igre

Na početku igre, postoji jedna hrpa objekata. Prvi igrač dijeli hrpu na dvije hrpe nejednakih veličina. Igra završava kad ostanu samo hrpe s manje ili jednako dva objekta, jer je njih nemoguće podijeliti na dvije nejednake hrpe. U igri pobjeđuje onaj igrač koji je zadnji odigrao dozvoljenu podjelu hrpi.
Proučite pravila Grundyjeve igre i implementirajte računalnog protivnika algoritmom MiniMax s alpha-beta podrezivanjem.
Analizirati odigravanja za gomile veličine do 100.

2. Optimalna strategija

2.1. Igra Nim

Grundyeva igra je varijanta igre Nim:

„U drevnoj Kini igrala se jedna lijepa i jednostavna igra poznata pod nazivom NIM. Igru igraju dva igrača. Na stolu stoji k hrpa od n novčića. Igrači naizmjence uzimaju proizvoljno karata s jedne od k hrpa, te onaj koji uzme zadnji novčić sa stola gubi igru.“

To je igra u kojoj svaki igrač ima *potpunu informaciju* o potezu protivnika, te slučajnost (šansa) ne igra nikakvu ulogu.

Svaki igrač ima konačan broj poteza pred sobom, te oba igrača, kada bi se našla u istoj „situaciji“ (tj raspored hrpi bi bio identičan) bi imala jednak skup mogućih poteza, tj igra je *imparcijalna.*

Matematička teorija iza ovakvih (takozvanih *Nim-games*, tj igara koje se svode na klasičnu igru Nim) igara je interesantna, te u svrhu kratkog pregleda iste ćemo uvesti par definicija:

- „Nim-suma“ stanja igre je kumulativna XOR vrijednost broja novčića u svakoj hrpi u tom trenutku igre
- „Mex“ skupa prirodnih brojeva je najmanji nenegativni prirodni broj koji se ne nalazi u tom skupu.

$$\begin{aligned}
&\operatorname{mex}(\emptyset) = 0 \\
&\operatorname{mex}(\{1,2,3\}) = 0 \\
&\operatorname{mex}(\{0,2,4,6,\ldots\}) = 1 \\
&\operatorname{mex}(\{0,1,4,7,12\}) = 2 \\
&\operatorname{mex}(\{0,1,2,3,\ldots\}) = \omega \\
&\operatorname{mex}(\{0,1,2,3,\ldots,\omega\}) = \omega + 1
\end{aligned}$$

- Grundyev broj („nimber“) igre je prirodan broj koji je jednak nuli za igru u kojoj prvi igrač gubi na prvom potezu, a inače je jednak Mex vrijednosti skupa svih Grundyevih brojeva koji odgovaraju svim mogućim idućim pozicijama.

Iz toga slijedi krunski teorem koji se može primjeniti na sve Nim-igre:

Sprague-Grundyev Teorem:

„Pretpostavimo da postoji jedna složena igra G koja se sastoji od N podigara, te dva igrača A i B. Pretpostavimo da A i B igraju optimalno (to jest ne čine pogreške).

Tada, igrač koji igra prvi će sigurno pobijediti ako kumulativna XOR vrijednost Grundyevih brojeva N podigara je različita od nule. Inače (ako je ta vrijednost jednaka nuli), drugi igrač će sigurno pobijediti.“

Iz ovog teorema se izvodi da se „razbijanjem“ igara sličnih igri Nim na podigre i računanjem Grundyevih brojeva istih podigara, lako može predvidjeti pobjednik optimalne igre.

2.2. Rješavanje algoritmom MiniMax s alpha-beta podrezivanjem

Pseudokod za algoritam MiniMax, koji maksimizira dobitak za prvog igrača (MAX), tj minimizira gubitak za drugog igrača (MIN):

```
3. function minimax(node, depth, maximizingPlayer) is
4.     if depth = 0 or node is a terminal node then
5.         return the heuristic value of node
6.     if maximizingPlayer then
7.         value := −∞
8.         for each child of node do
9.             value := max(value, minimax(child, depth − 1, FALSE))
10.          return value
11.     else (* minimizing player *)
12.         value := +∞
```

```
13.          for each child of node do
14.              value := min(value, minimax(child, depth - 1, TRUE))
15.          return value
```

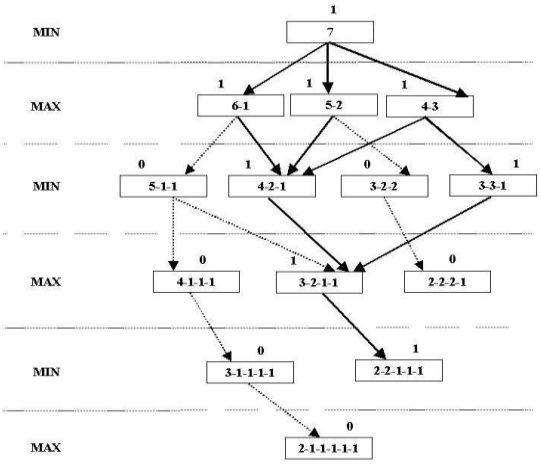
MiniMax algoritam se nadalje poboljšava s takozvanim Alpha-Beta podrezivanjem *(Alpha-Beta Pruning),*čiji je pseudokod dan dolje:

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth - 1, α, β,
FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut-off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth - 1, α, β,
TRUE))
            β := min(β, value)
            if α ≥ β then
                break (* α cut-off *)
        return value
```

U našem zadatku će ovakav algoritam za konstrukciju svog idućeg poteza koristiti implementirani računalni protivnik.

Nažalost, čak i uz alpha-beta podrezivanje je vremenska složenost algoritma $O(b^{m/2})$, gdje je b kardinalitet skupa mogućih poteza, a m dubina stabla igre koje algoritam konstruira.

Primjer stabla Grundyeve igre se nalazi dolje:



3.Implementacija optimalne igre

Implementacija je izvršena u C++-u, sa
Čvorove stabla igre (tj „stanja“ igre) smo implementirali strukturom *node,* koja sarži razne korisne funkcije.

```
 8   //čvor u drvu - sazdrži listu, evaluation value, alpha, beta
 9   struct node
10   {
11       list<int> lista;
12       int nodedepth;
13       int turn; //TKO VUČE POTEZ U TOM TRENU, 1 je kompjuter, 0 je čovjek
14       int evaluation; // EVALUATION JE 1 AKO KOMP POBJEĐUJE, -1 AKO ČOVJEK POBJEĐUJE, 0 AKO SE IGRI
15       int alpha;
16       int beta;
17       node* parent;
18       list<node> children;
19
20       //default konstuktor
21       node(){
31
32       //konstruktor s listom
33       node(list<int> L){
49
50       //kopira listu u node
51       void input_list(list<int> L){
59
60
61       void copy(node zakopirati){
76
77
78       bool gameover(node temp){
86
87       void expand(){
118
119      bool gameover(){
126
127  };
```

Od čega je od posebnog interesa funkcija *expand()*, koja konstruira sve moguće nove poteze koje igrač može povući iz trenutnog čvora, te ih sprema u listu čvorova *children* koja se nalazi u trenutnom čvoru.

```
87  void expand(){
88      list<int>::iterator li;
89      //IDEM PO HRPAMA
90      for(li=lista.begin(); li!=lista.end(); li++){
91        int element=*li;
92        int i;
93        for (i=1; i<element; i++){
94            //NE SMIJEM DIJELITI NA JEDNAKE HRPE
95            int ostatak=element-i;
96            if (ostatak!=i){
97                //RADIM NOVI CHILDLIST
98                list<int> tmplist;
99                list<int>::iterator ti;
100               for(ti=lista.begin(); ti!=lista.end(); ti++){
101                   if(ti!=li){int tmpelement=*ti; tmplist.push_back(tmpelement);}else{tmplist.push_back(i);tmplist.push_back(ostatak);}
102                   };
103               //RADIM NOVI CHILDNODE I APPENDAM MU MAMU
104               node tmpnode(tmplist);
105               tmpnode.nodedepth=nodedepth+1;
106               if (turn==1){tmpnode.turn=0;}else{tmpnode.turn=0;}
107               if (this->gameover(tmpnode)==1){
108                   if (tmpnode.turn==0){tmpnode.evaluation=-1;}else{tmpnode.evaluation=1;};
109               };
110               tmpnode.parent=this;
111               //APPENDAM CHILDNODE NA MAMU
112               children.push_back(tmpnode);
113                ;}
114           };
115       };
116
117 };
118
```

Sam MiniMax algoritam s alpha-beta podrezivanjem je implementran van svih klasa (struktura) kodom:

```
int alphabeta(node *presentstate){
    //AKO JE PRESENTSTATE LIST
    if(presentstate->gameover()==1){
        if(presentstate->turn==0){
            //MINPLAYER LOST -> MAXPLAYER WON
            presentstate->evaluation=1;
            return presentstate->evaluation;
        }else{
            //MAXPLAYER LOST -> MINPLAYER WON
            presentstate->evaluation=-1;
            return presentstate->evaluation;
        };
    };
    //AKO PRESENTSTATE NIJE LIST, GLEDAM MU DJECU I BACAM REKURZIJU NA NJIH
    presentstate->expand();
    list<node>::iterator li;
    for (li=presentstate->children.begin(); li!=presentstate->children.end(); li++){
        node childnode(*li);
        if(presentstate->turn==1){
            //MAXPLAYER
            presentstate->alpha=max(presentstate->alpha, alphabeta(&childnode));
            if(presentstate->beta<=presentstate->alpha){break;};
            return presentstate->alpha;
        }else{
            //MINPLAYER
            presentstate->beta=min(presentstate->beta, alphabeta(&childnode));
            if(presentstate->beta<=presentstate->alpha){break;};
            return presentstate->beta;
        };
    };
    return -100; //DO OVDJE FUNKCIJA NE BI SMJELA DOCI, ERROR CONTROL
};
```

3.Implementacija igre između čovjeka i kompjutera

Sama igra počinje time da program priupita igrača za početni broj objekata (u intervalu [1, 100]), te ljudski igrač povlači prvi potez.

```
Upisite pocetni broj nocica na hrpi (izmedju 1 i 100): 1
Vi ste pobjedili

Process returned 0 (0x0)   execution time : 2.980 s
Press any key to continue.
```

Implementirano sa

```
int main()
{
    cout << "Upišite početni broj nočića na hrpi (između 1 i 100): ";
    int n;
    cin >> n;
    if (n<=0 || n>100)throw "Unesen broj veći od 100 ili manji od 1!";

    Game Grundy(n);
    Grundy.run();

    return 0;
}
```

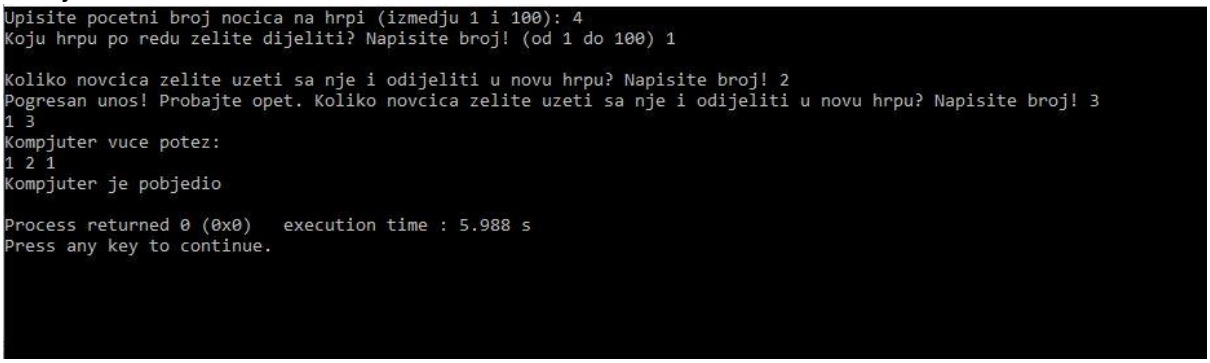
Gdje klasa *Game* zapravo je ona koja kontrolira interakciju čovjeka i i računalnog protivnika, te se sastoji od funkcija

- *Game(int n)* je konstruktor klase *Game* koji konstruira igru sa jednom početnom hrpom od n objekata
- *Run()* je funkcija koja zove funkciju *gameover()* i ispisuje pobjednika ukoliko je igra gotova, zove funkciju pripadajućeg igrača ovisno o tome tko je na potezu i zove funkciju *printheaps()*
- *Gameover()* je funkcija koja vraća *True* ako je igra gotova, a *False* ako nije
- *Printheaps()* je funkcija koja ispisuje trenutno stanje igre
- *PersonPlayer()* je funkcija koja kontrolira unos ljudskog igrača te implementira njegov potez

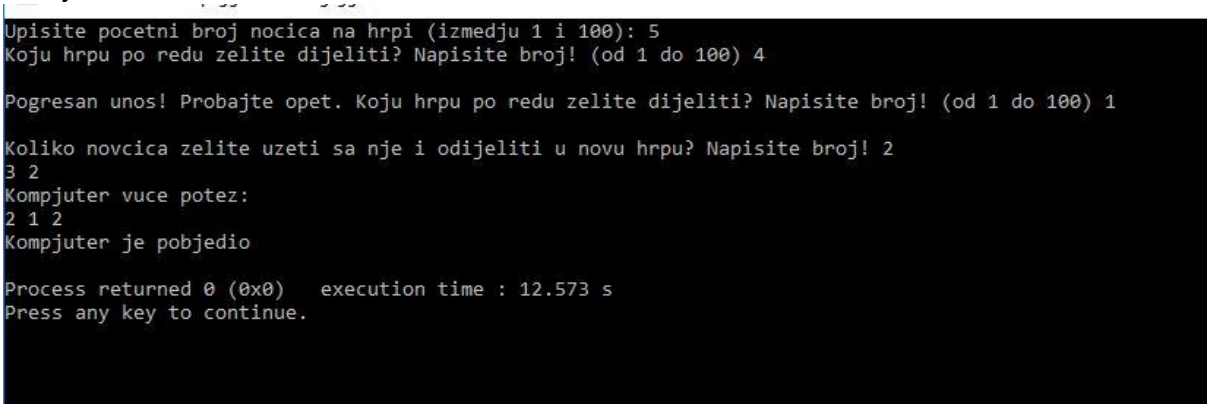
- *ComputerPlayer() je* funkcija koja stvara čvor trenutnog stanja, te pomoću funkcija *alphabeta()* konstruira optimalan potez te ga provodi

5.Primjeri testova/igranja

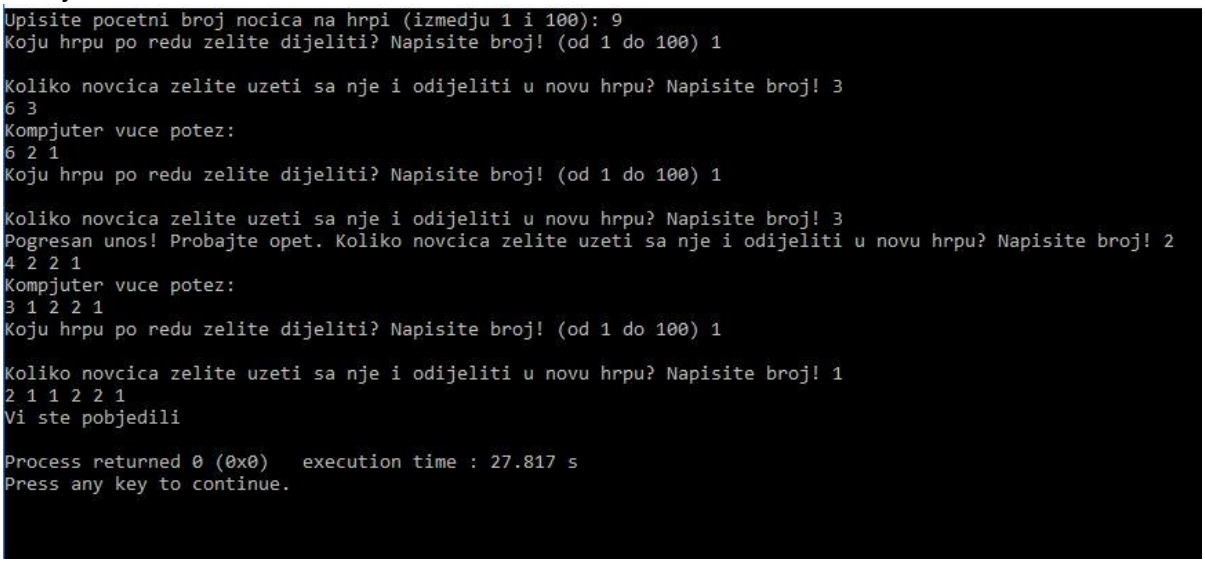
Primjer 1:



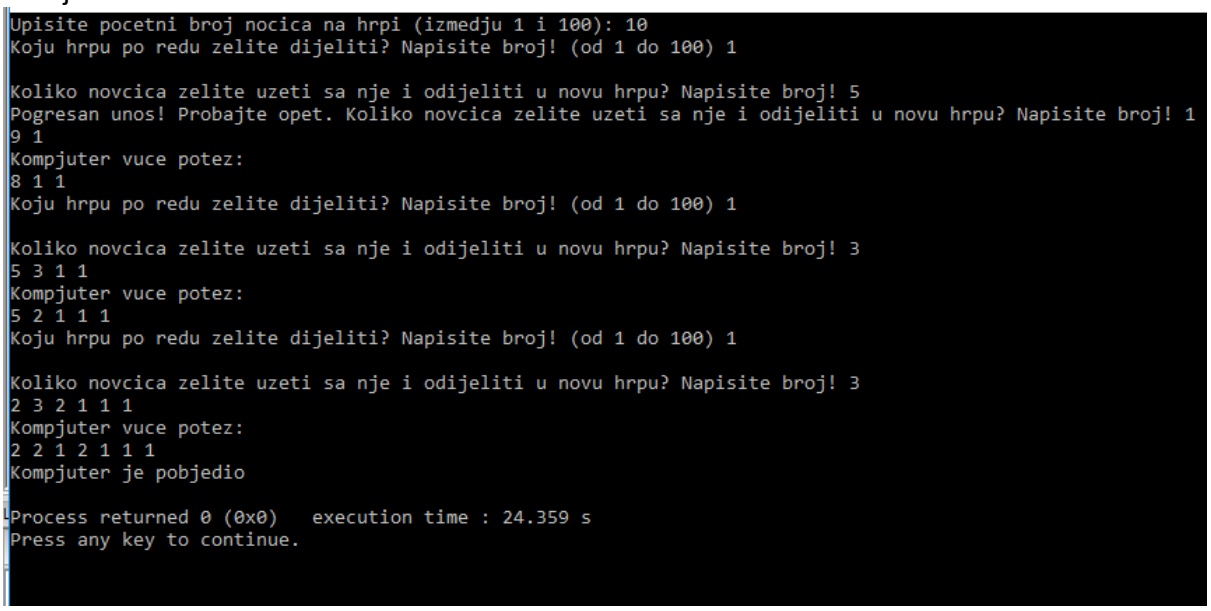
Primjer 2:



Primjer 3:



Primjer 5:



6.Izvori

- http://e.math.hr/old/igre/index.html
- https://www.geeksforgeeks.org/combinatorial-game-theory-set-2-game-nim/
- https://cs.stackexchange.com/questions/6363/nim-game-tree-minimax
- http://degiorgi.math.hr/~singer/ui/