Structuri de date - Seria CC

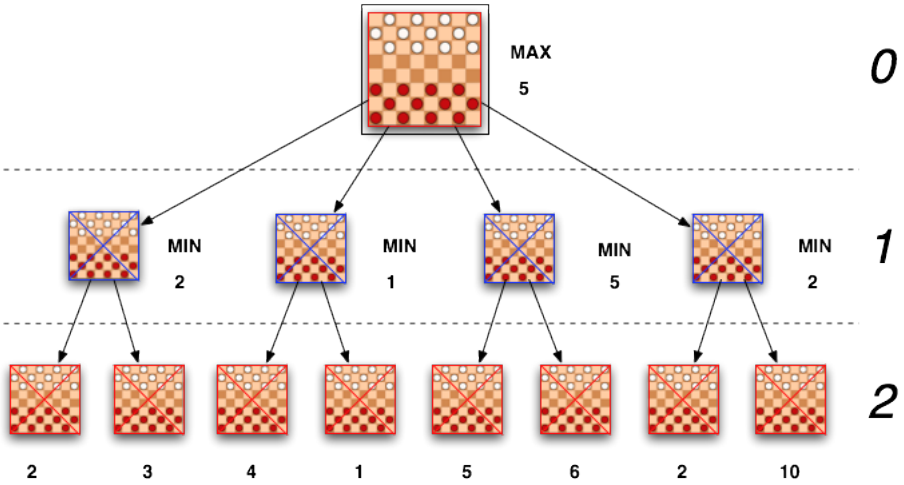
Game Tree

Tema 2

Deadline: 23.04.2017

Mihai Nan

mihai.nan.cti@gmail.com



Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2016 - 2017

1 Obiective

În urma realizării acestei teme, studentul va fi capabil:

- să implementeze şi să utilizeze arbori în rezolvarea unor probleme;
- să înţeleagă noțiunea de arbore de joc;
- să implementeze algoritmul **Minimax** pentru determinarea unei acțiuni;
- să transpună o problemă din viaţa reală într-o problemă care uzitează arbori de joc.

2 Descriere

Inteligența Artificială poate fi definită ca simularea inteligenței umane procesată de mașini, în special, de sisteme de calculatoare. Acest domeniu a fost , în general, caracterizat de cercetări complexe în laboratoare, iar, în ultima perioadă, a devenit parte a tehnologiei în aplicațiile comerciale.

Începuturile inteligenței artificiale pot fi văzute imediat după al Doilea Război Mondial, în primele programe care rezolvau puzzle-uri sau care jucau anumite jocuri. Au existat două motive pentru care jocurile au fost printre primele domenii de aplicare a inteligenței artificiale: pentru că performanța programului este ușor de măsurat, apoi, deoarece regulile sunt, în general, simple și puține la număr, deci pot fi ușor descrise și folosite.

2.1 Descrierea formală a unui joc

Faptul că în cazul jocurilor mai apare și un adversar face ca problema de alegere a unei acțiuni să fie mai complicată decât o problema de căutare clasică, deoarece adversarul este cel care aduce o incertitudine, jucătorul curent neștiind decizia următoare a adversarului. Astfel, faptul că mutările adversarului sunt imprevizibile ne face să specificăm o mutare pentru fiecare posibil răspuns al adversarului.

În continuare, vom considera cazul general al unui joc de două persoane, pe care le vom numi `Max` și `Min`, cu informație perfectă.

Pornind de la aceste considerente, un joc poate fi definit formal ca o problemă de căutare cu următoarele componente:

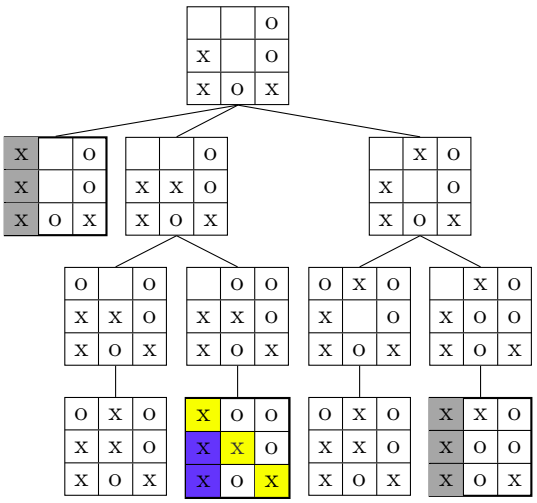
- **starea inițială** - include pozițiile de pe tablă și cine este cel care urmează să efectueze mutarea;
- **mulțimea acțiunilor posibile** - mutările admise pe care le poate face un jucător la o anumită rundă;
- **stare terminală** - stările în care jocul se încheie (cu victoria unui jucător sau cu remiză);
- **funcție de utilitate** - care întoarce o valoare numerică pentru rezultatul jocului.

Problema care se pune pentru rezolvarea problemei de căutare este găsirea unei strategii care să îl ducă pe `Max` la o stare terminală în care el este câștigătorul, indiferent de ce mutări face `Min`.

2.2 Arborele de joc

Arborele de joc este o modalitate de reprezentare a mutărilor posibile ce se pot realiza pentru un anumit joc. Rădăcina arborelui este starea inițială a jocului, iar nodurile arborelui reprezintă stări intermediare în joc. Arcele sunt mutările efectuate în joc și, astfel, putem spune că trecerea dintr-o stare intermediară (nod în arbore) în altă stare intermediară se face prin intermediul unei mutări (arc în arbore). Frunzele arborelui sunt stări finale, configurații din care jucătorul aflat la mutare nu mai poate muta. Cu alte cuvinte, frunzele corespund pozițiilor de victorie, pierdere sau remiză.

În continuare, se va prezenta un exemplu pentru celebrul joc `X` și `O`, având ca rădăcină o configurație diferită de cea inițială, reprezentând, astfel, un subarbore al întregului arbore de joc.



Observație

Un **joc de sumă zero** este un joc la care există un învingător şi un învins, eventual jocul se termină cu o remiză. Pentru a înţelege, într-un mod simplu, ce este un joc de sumă zero putem folosi următorul punctaj: învingătorul primeşte **+1** punct, învinsul **-1** punct, iar în caz de remiză fiecare primeşte **0** puncte.

Vom numi joc cu informaţie perfectă acel joc în care toţi jucătorii ştiu la orice moment **t** ce decizii s-au luat în etapa anterioară (la momentul **t-1**). De asemenea, în jocurile analizate nu vor exista elemente de șansă, cum ar fi aruncarea unui zar.

2.3 Arborele ȘI/SAU

Multe probleme se pot descompune într-o serie de subprobleme astfel încât rezolvarea tuturor acestor subprobleme, sau a unora din ele, să ducă la rezolvarea problemei inițiale. Descompunerea unei probleme complexe, în mod recursiv, în subprobleme mai simple poate fi reprezentată printr-un arbore. Această descompunere se numește *reducerea problemei* și este folosită în demonstrarea automată, integrarea simbolică și, în general, în probleme ce țin de inteligența artificială. Într-un arbore de acest tip, vom permite unui nod neterminal oarecare N două alternative: nodul este de tip **ȘI** dacă reprezintă o problemă care este rezolvată doar dacă **toate** subproblemele reprezentate de copiii nodului sunt rezolvate, respectiv este de tip **SAU** dacă reprezintă o problema care este rezolvată în cazul în care cel puțin o subproblemă, reprezentată de nodurile copil, este rezolvată. Un astfel de arbore se mai numește arbore de tip **ȘI/SAU**.

Având în vedere că am spus anterior că un joc poate fi văzut ca o problemă de căutare, reprezentarea spațiului de căutare se poate realiza cu ajutorul arborilor **ȘI/SAU** pentru jocurile cu 2 utilizatori, astfel:

- nodul **SAU** - selectarea unei mutări pentru jucătorul curent;
- nodul **ȘI** - considerarea tuturor mutărilor posibile ale adversarului.

Pași urmați în rezolvarea unui joc folosind abori **ȘI/SAU** sunt următorii:

1. Descompunerea jocului în sub-jocuri și inserarea nodurilor pe mai multe nivele, nodurile de pe un nivel corespunzând mutărilor posibile ale unui jucător;

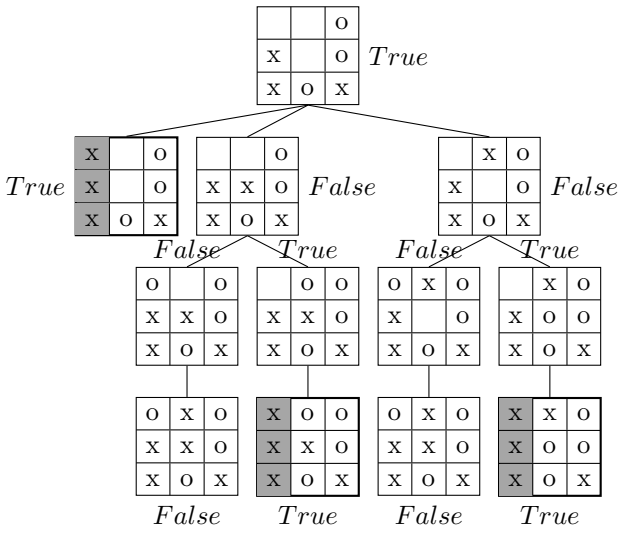
2. Găsirea unei strategii perfecte de câștig pentru fiecare sub-joc (etichetarea nodurilor cu **T** - *True*; sau **F** - *False*, în funcție de posibilitatea jucătorului pentru care dorim să determinăm mutarea de câștig pentru acel sub-joc);

   a) frunzele se etichetează cu **T** sau **F**, în funcție de configurația jocului;

   b) pentru nodurile interne de tip **SAU**, cu **T** dacă există cel puțin un nod fiu ce a fost etichetat cu **T**;

   c) pentru nodurile interne de tip **ȘI**, cu **T** dacă toate nodurile fiu au fost etichetate cu **T**.

3. Combinarea sub-strategiilor pentru obținerea strategiei finale.



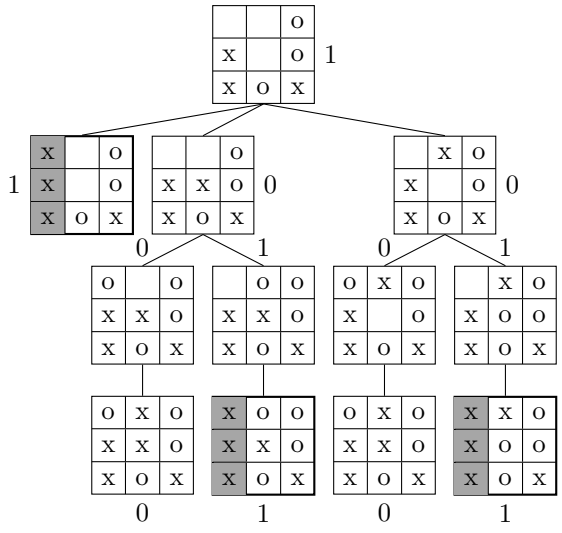
2.4 Algoritmul Minimax

În cazul jocurilor cu doi jucători, în care oponenți își modifică pe rând poziția de joc sau starea, algoritmul **Minimax** este cel mai uzitat, împreună cu variantele sale îmbunătățite. În linii mari, acest algoritm folosește o funcție care decide cât de bună este o poziție, prin atribuirea unor scoruri. Algoritmul se bazează pe existența a doi jucători cu strategii diferite: jucătorul ***Max*** este cel care va încerca în permanență să-și maximizeze câștigul, în timp ce jucătorul ***Min*** dorește să minimizeze câștigul jucătorului ***Max*** la fiecare mutare. Având în vedere că, în cazul jocurilor ce au un factor de ramificare mare, arborele de căutare ar conține foarte multe noduri, algoritmul ajungând să fie aproape imposibil de aplicat, datorită timpului mare necesar analizării tuturor pozițiilor disponibile, în vederea selectării celei mai potrivite, s-a încercat optimizarea acestuia. Astfel, au apărut diverse variante echivalente optimizate cum ar fi **Negascout**, **Negamax**, **Alpha-Beta**.

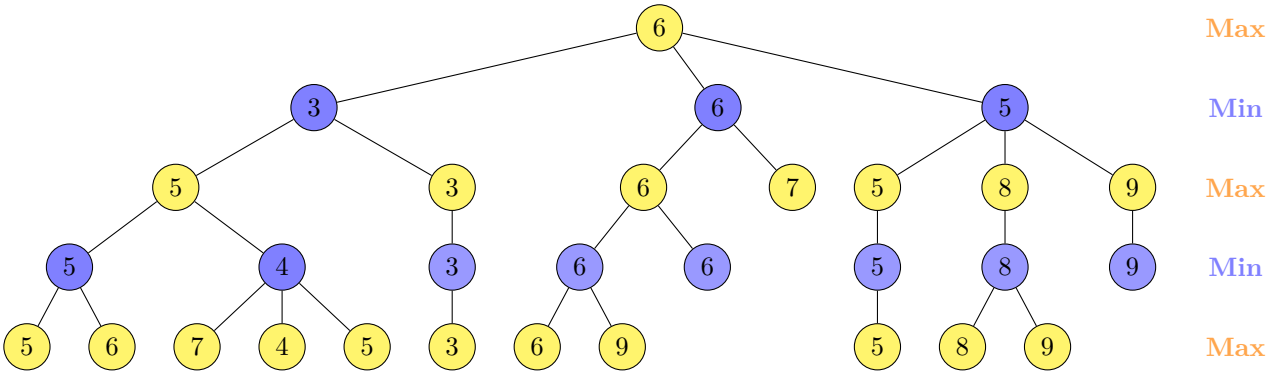
Algoritmul **Minimax** realizează o căutare în adâncime în arborele de joc, unde nodurile reprezintă stări ale jocului, iar arcele definesc acțiuni posibile ce se pot realiza dintr-o stare de joc. Astfel, configurația inițială a jocului este rădăcina arborelui de joc, iar frunzele reprezintă stări finale pentru joc, pentru care se poate aplica o funcție de utilitate în vederea determinării unei valori, care este propagată spre nivelurile superioare ale arborelui.

Pentru o înțelegere mai bună a acestui algoritm, vom relua exemplul anterior, utilizând o funcție de utilitate care atribuie valoarea 1 unei stări finale dacă jucătorul X a câștigat, -1 dacă jucătorul 0 a câștigat și 0 pentru remiză.

În continuare, se va prezenta un exemplu pentru celebrul joc X și 0, având ca și rădăcină o configurație diferită de cea inițială, reprezentând, astfel, un subarbore al întregului arbore de joc. În acest caz, funcția de utilitate atribuie valoarea 1 unei stări finale dacă jucătorul X a câștigat, -1 dacă jucătorul 0 a câștigat și 0 pentru remiză.



Pentru acest algoritm, arborele de căutare constă în alternarea nivelelor pe care un jucător încearcă să-și maximizeze câștigul cu nivelele pe care adversarul încearcă să îi minimizeze câștigul. Primul jucător va încerca să-și maximizeze câștigul, astfel, jucătorul `Max` este cel care mută primul, iar al doilea jucător va încerca să minimizeze câștigul primului jucător, jucătorul `Min` reprezentând adversarul.



În concluzie, algoritmul **Minimax** determină o strategie optimă pentru `MAX`, iar acesta constă în următorii pași:

1. Generează tot arborele de joc, până la stările terminale.
2. Aplică funcția de utilitate pentru fiecare stare terminală pentru a îi determina valoarea.
3. Folosește utilitatea stărilor terminale pentru a determina utilitatea stărilor de la un nivel superior din arborele de căutare.
4. Continuă evaluarea utilităților nodurilor pe niveluri mergând până la rădăcină.
5. Când se ajunge la rădăcină, `Max` alege nodul de pe nivelul inferior cu valoarea cea mai mare.

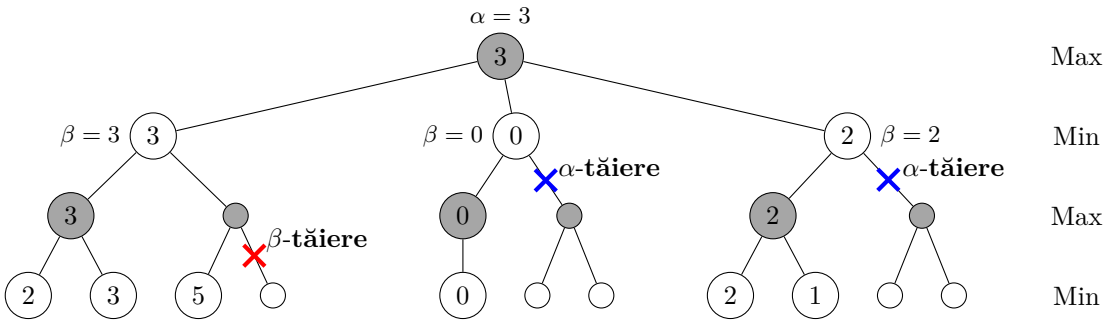
2.5 Alpha - Beta Pruning

Alpha-Beta pruning reprezintă o îmbunătățire semnificativă a Minimax-ului, deoarece această tehnică elimină întregi subarbori corespunzători unei anumite mutări dacă mutarea respectivă este mai slabă, din punct de vedere calitativ, decât cea mai bună mutare curentă. Această tehnică de optimizare folosește două valori: `alpha` și `beta` care se actualizează în timpul execuției algoritmului; care constituie o fereastră folosită pentru filtrarea mutărilor posibile.

- α - mai este numit și plafonul de minim, fiind cel care stabilește faptul că o mutare nu poate avea o valoare mai mică decât acest prag;
- β - numit și plafonul de maxim, se folosește pentru a verifica dacă o mutare este prea bună pentru a putea fi luată în calcul.

Ținând cont de principiul algoritmului Minimax, se poate concluziona că plafonul minim al unui jucător reprezintă plafonul maxim al celuilalt și invers. Această fereastră, definita de cele două plafoane, este inițializată cu intervalul $[-\infty, +\infty]$ și se tot micșorează pe parcursul rulării algoritmului, determinând tăierea unui număr din ce în ce mai mare de mutări din arborele de joc. Aceste tăieri sunt de două tipuri:

- **α-tăieri** - În cazul în care există, pentru un nod **Min**, o acțiune ce are asociată o valoare $v \leq \alpha$, atunci putem renunța la expandarea subarborelui său, deoarece **Max** poate atinge deja un câștig mai mare, α, dintr-un subarbore precedent.
- **β-tăieri** - În cazul în care există, pentru un nod de tip **Max**, o acțiune ce are asociată o valoare $v \geq \beta$, atunci putem renunța la expandarea subarborelui său, deoarece **Min** a limitat deja câștigul lui **Max** la β.



Tăierea de tip β s-a realizat, deoarece am determinat pentru un nod de pe nivelul **Max** ce conține o acțiune mai bună (**5**) decât ce aveam până în acest moment (**3**), dar pe care nu o putem realiza, deoarece nivelul **Min** nu va permite acest lucru.

Cele două tăieri de tip α au fost realizate, deoarece am determinat pentru un nod **Min** acțiuni (**0** și **2**) ce au asociate valori mai mici decât α (**3**).

3 Cerințe

Această temă dorește să vă introducă în vasta lume a jocurilor, propunându-vă spre implementare cele mai simple tehnici folosite, în inteligența artificială, pentru jocurile cu informație perfectă ce au doi jucători.

3.1 Cerința 1

Pentru această cerință va trebui să construiți arborele de joc pentru unul din cele mai simple și cunoscute jocuri de sumă zero, X și 0.

Fișierul de intrare va conține descrierea stării de joc corespunzătoare rădăcinii arborelui, iar în fișierul de ieșire veți afișa arborele rezultat. Pentru fiecare nod al arborelui, va trebui să afișați, în fișierul de ieșire, starea de joc corespunzătoare nodului, o stare fiind reprezentată de tabla efectivă de joc.

Fișierul de intrare va conține pe prima linie X sau 0, reprezentând indicele jucătorului care urmează să efectueze mutarea, iar pe următoarele linii o să fie tabla de joc de la care veți porni construcția arborelui. În fișierul de ieșire, afișarea se va face indentând corespunzător, folosind tab-uri, fiecare nivel din arborele rezultat. Cu alte cuvinte, pentru rădăcina arborelui, considerat primul nivel, nu se va folosi tab, pentru al doilea nivel se va folosi un tab si tot așa.

Pentru o înțelegere mai bună, puteți analiza următorul exemplu.

3.1.1 Exemplu

```
0
- 0 X
0 X X
0 X -
```

```
- 0 X
0 X X
0 X -

        0 0 X
        0 X X
        0 X -

        - 0 X
        0 X X
        0 X 0

                X 0 X
                0 X X
                0 X 0
```

Observație

Atunci când veți insera nodurile în abore veți cont de ordinea acțiunilor. Dacă pentru o stare de joc avem la dispoziție două acțiuni $a_1 = (0, 1)$ și $a_2 = (i, j)$, o vom alege prima pe a_1 dacă $i \cdot 3 + j > 0 \cdot 3 + 1$, altfel începând construcția subarborilor cu a_2. Acțiunea (x, y) este echivalentă cu plasarea simbolului pe linia x, coloana y.

Atenţie!

Pentru a primi punctaj, trebuie să respectați formatul de afișare impus. Pentru a indenta corespunzător output-ul, folosind tab-uri, veți uzita $\backslash t$, iar pentru fiecare nod se vor afișa toți subarborii săi, respectând indentarea.

3.2 Cerința 2

În cadrul acestei cerințe, veți determina arborele **ȘI/SAU** pentru o anumită stare de joc, aparținând, de asemenea, jocului X și 0.

Fișierul de intrare pentru această cerință respectă formatul propus în cadrul cerinței anterioare, dar, de această dată, fișierul de ieșire va conține arborele rezultat prin evaluarea nodurilor **ȘI/SAU**.

3.2.1 Exemplu

```
X
- - 0
X - 0
X 0 X
```

```
T
    T
    F
        F
            F
        T
            T
    F
        F
            F
        T
            T
```

3.3 Cerința 3

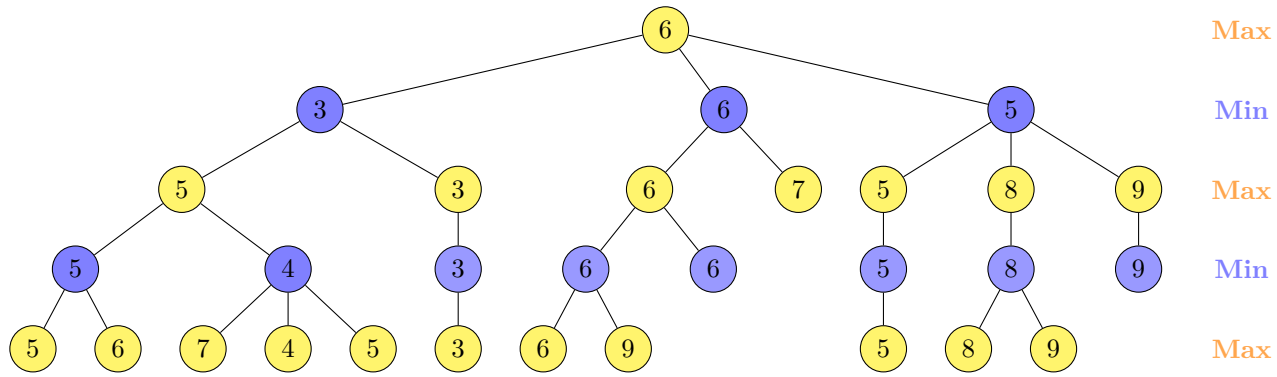
De această dată, va trebui să aplicați algoritmul **Minimax** pentru un arbore de joc pentru care cunoașteți valorile nodurilor frunză. Astfel, în fișierul de intrare veți avea descris un arbore care conține valori efective doar în frunze, iar voi va trebui să determinați valorile pentru fiecare nod din arbore, aplicând algoritmul **Minimax** și să afișați, în fișierul de ieșire, arborele rezultat.

Prima linie a fișierului de intrare va conține înalțimea arborelui, a doua linie va conține numărul de noduri copil ce au ca părinte direct rădăcina arborelui, pe linia următoare se va indica pentru fiecare nod de pe nivelul doi câți copii are și tot așa. Pentru a face diferența între nodurile interne și frunze, numărul de copii pentru nodurile interne o să fie plasat între paranteze rotunde, în timp ce pentru frunze se va plasa între paranteze drepte utilitatea stării.

Toate valorile oferite pentru utilitatea vor fi numere întregi.

3.3.1 Exemplu

Pentru următorul arbore de joc, vom prezenta fișierul de intrare și cel de ieșire.



Fișierul de intrare

```
5
(3)
(2) (2) (3)
(2) (1) (2) [7] (1) (1) (1)
(2) (3) (1) (2) [6] (1) (2) [9]
[5] [6] [7] [4] [5] [3] [6] [9] [5] [8] [9]
```

Fișierul de ieșire

```
6
        3
                5
                        5
                                5
                                6
                        4
                                7
                                4
                                5
                3
                        3
                                3
        6
                6
                        6
                                6
                                9
                        6
                7
        5
                5
                        5
                                5
                8
                        8
                                8
                                9
                9
                        9
```

3.4 Bonus

După cum se poate observa, pentru jocurile care au un factor de ramificare mare, un număr mare de acțiuni ce se pot aplica, arborele de joc rezultat este unul destul de complex. Primind un arbore de joc, descris folosind fomatul de la cerința anterioară, va trebui să realizați tăierile corespunzătoare, aplicând algoritmul **Alpha - Beta Pruning**.

În fișierul de ieșire veți afișa arborele rezultat în urma aplicării algoritmului **Alpha - Beta Pruning**, păstrând convenția definită anterior.

4 Restricţii şi precizări

Temele trebuie să fie încărcate pe vmchecker. **NU** se acceptă teme trimise pe e-mail sau altfel decât prin intermediul vmchecker-ului.

O rezolvare constă într-o arhivă de tip **zip** care conţine toate fişierele sursă alaturi de un **Makefile**, ce va fi folosit pentru compilare, şi un fisier README, în care se vor preciza detaliile implementării.

Makefile-ul trebuie să aibă obligatoriu regulile pentru **build** şi **clean**. Regula **build** trebuie să aibă ca efect compilarea surselor şi crearea binarului **minimax**.

Programul vostru va primi, ca argumente în linia de comandă, numele fişierului de intrare şi a celui de ieşire, dar şi o opţiune în felul următor:

./minimax [-c1 | -c2 | -c3 | -b] [fişier_intrare] [fişier_ieşire] unde:

- **-c1** indică faptul că programul va rezolva cerinţa 1;
- **-c2** indică faptul că programul va rezolva cerinţa 2;
- **-c3** indică faptul că programul va rezolva cerinţa 3;
- **-b** indică faptul că programul va rezolva bonusul;
- **fişier_intrare** reprezintă numele fişierului de intrare;
- **fişier_ieşire** reprezintă numele fişierului de ieşire, în care se va scrie, in funcţie de comanda primită, rezultatul execuţiei programului.

5 Punctaj

Cerinta	Punctaj
Cerinţa 1	40 puncte
Cerinţa 2	20 puncte
Cerinţa 3	30 puncte
Codying style, README, warning-uri	10 puncte
Bonus	20 puncte

Atenţie!

Orice rezolvare care nu conţine structurile de date specificate nu este punctată.
Temele vor fi punctate doar pentru testele care sunt trecute pe vmchecker.
Nu lăsaţi warning-urile nerezolvate, deoarece veţi fi depunctaţi.
Dealocaţi toată memoria alocată pentru reţinerea informaţiilor, deoarece se vor depuncta pierderile de memorie.
Tema este individuală! Toate soluţiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.