Hraní her

(Teorie a algoritmy hraní her)

8. 3. 2019

Hraní her pro dva a více hráčů

Počítač je při hraní jakékoli hry:

- silný v komplikovaných situacích s množstvím kombinací,
- má obrovskou znalost zahájení a koncovek,
- nevnímá hluk, stress a psychický tlak,
- tvrdý hráč proti soupeři bez fantazie, je však "překvapen" originálními tahy,
- má malou nebo žádnou znalost strategie, díky efektu horizontu neodhalí plán na mnoho kol dopředu,
- je obvykle zaměřen na materiální výhodu – neobětuje figuru či pozici, pokud se to brzy nevyplatí.

Algoritmy hraní her –

– algoritmus minimaxu a alfa-beta prořezávání

Hraní her je zcela přirozeně spojováno s představou prohledávání stavového prostoru, tedy jakési zkoušení možných tahů. Téměř všichni lidé právě tímto způsobem hrají – přemýšlejí, co by mohl soupeř udělat, pokud oni táhnou takhle.

Algoritmy hraní her –

– algoritmus minimaxu a alfa-beta prořezávání

Hraní her je zcela přirozeně spojováno s představou prohledávání stavového prostoru, tedy jakési zkoušení možných tahů. Téměř všichni lidé právě tímto způsobem hrají – přemýšlejí, co by mohl soupeř udělat, pokud oni táhnou takhle.

Základní rozdělení algoritmů:

1) algoritmy, které prohledávají celý stavový prostor do určité hloubky,
2) algoritmy, prohledávající jen "dobře vypadající" části stavového prostoru,
3) cílově orientované algoritmy

Nejčastěji se používají programy prvního typu, ale je i hodně programů třetího typu (ty jsou ale většinou dobré jen na jednu konkrétní věc - v případě šachu např. některé koncovky).

Klíčové otázky hraní her:

1. Je nalezené řešení dobré ?
   - libovolné řešení (lze použít heuristiku)
   - nejlepší (prohledat celý prostor)
2. Mohu "vrátit" tah ?
   - ano (osmička)
   - ne (šachy)
3. Je "jistý" vstup ?
   - ano (šachy)
   - ne (poker)

Klasické deskové logické hry

- Základem je PROBLEM SOLVING – snažíme se dostat z počátečního stavu do cílového (např. mat) za použití pravidel.
- Významný rozdíl je v tom, že hráči mají rozdílné úkoly. Cílem jednoho je maximalizovat ohodnocující funkci, zatímco cílem druhého je minimalizovat jí.

Kromě klasického hledání řešení se používají i jiné techniky:

- knihovna zahájení a koncovek
- rozpoznávání vzorů (šablon)

Algoritmus existence vítěze a minimaxu

- Algoritmy na prohledávání stavového prostoru se používají pro hry dvou hráčů s úplnou informací, tzn. když má každý hráč všechny informace o hře a jejím současném stavu.
- Tato informace je konečná (nazývá se pozice) a obsahuje též údaj, který hráč je na tahu.
- Dále existují pravidla hry, která určují ke každé pozici konečný počet přípustných tahů, pro hráče, který je právě na tahu.
- Krokem hry (tahem, popř. půltahem) je, když si hráč, který je na tahu, vybere jeden z přípustných tahů a provede jej. Tím vznikne nová pozice a na tahu je soupeř.
- Hra pokračuje, dokud se nedostane do závěrečné pozice, u které musí být též definováno, kdo vyhrál či prohrál, příp. že jde o remízu.

- Na závěr je ještě nutné, aby pravidla vylučovala nekonečnou hru.

Existence vítěze

Hra je znázorněna stromem, v němž

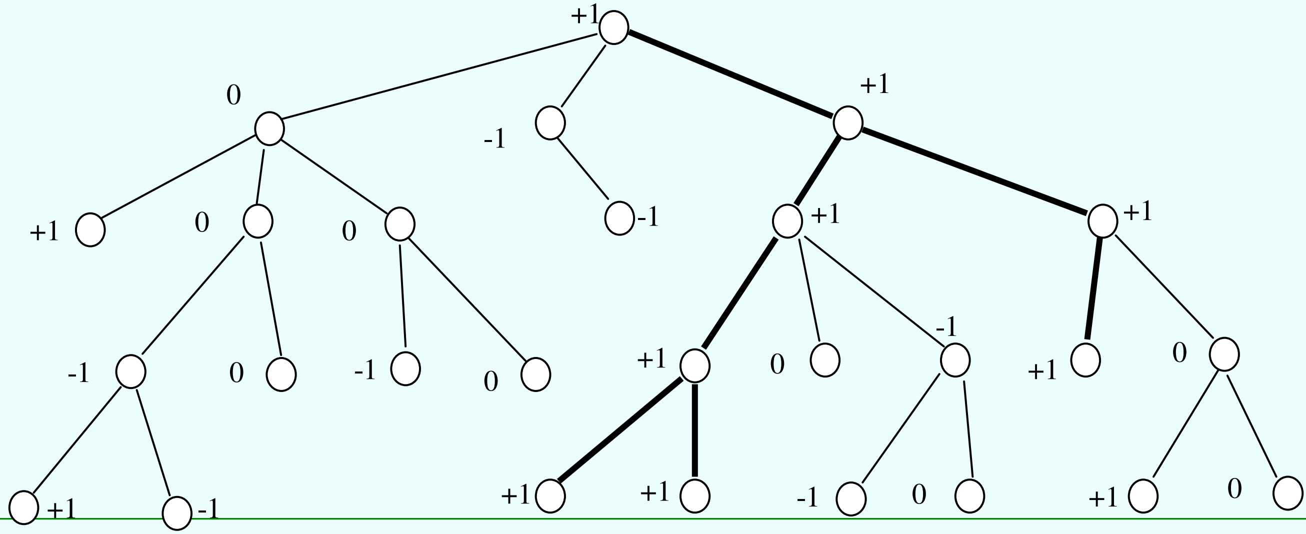
- uzly reprezentují jednotlivé pozice (stavy) hry,
- hrany reprezentují přípustné tahy,
- listy stromu odpovídají koncovým pozicím (stavům) hry.

Existence vítěze

Hra je znázorněna stromem, v němž

- uzly reprezentují jednotlivé pozice (stavy) hry,
- hrany reprezentují přípustné tahy,
- listy stromu odpovídají koncovým pozicím (stavům) hry.

Př.: Pozice ohodnocená
„+1“ je vítězná,
„–1“ je prohraná,
„ 0 “ je remízou.
Zvýrazněné tahy jednoznačně vedou k výhře.

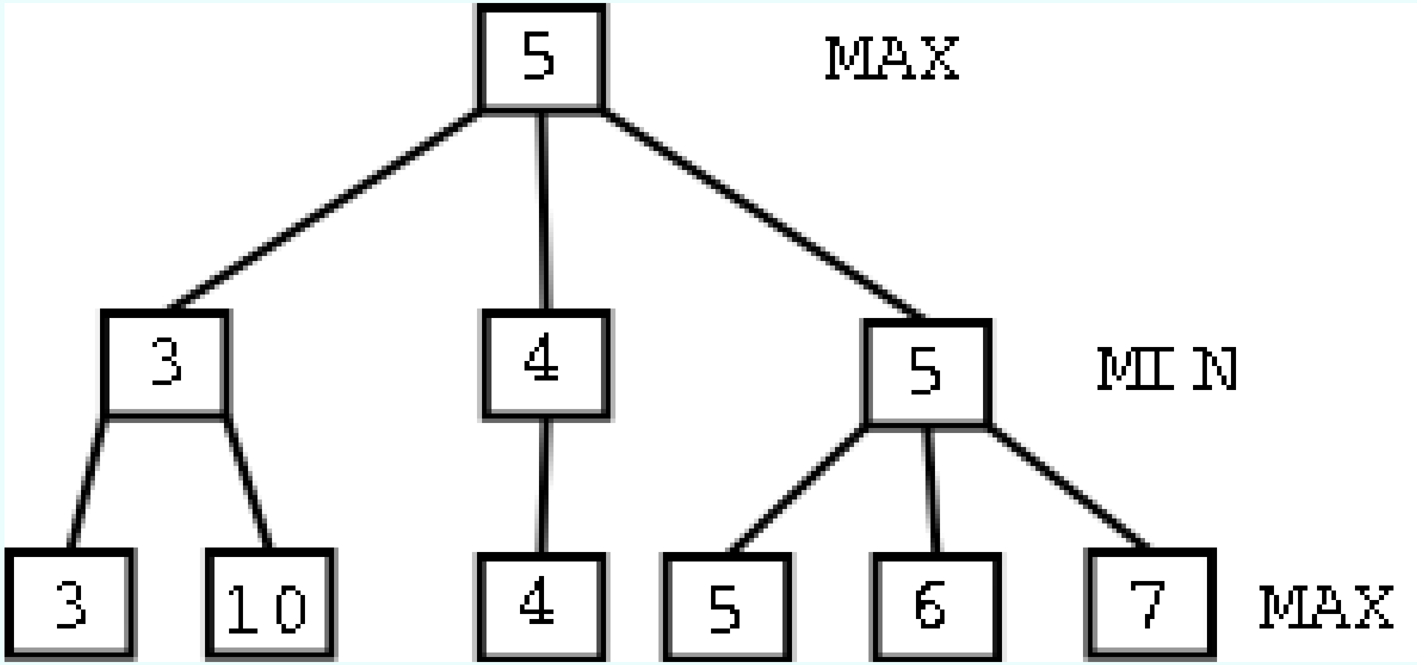


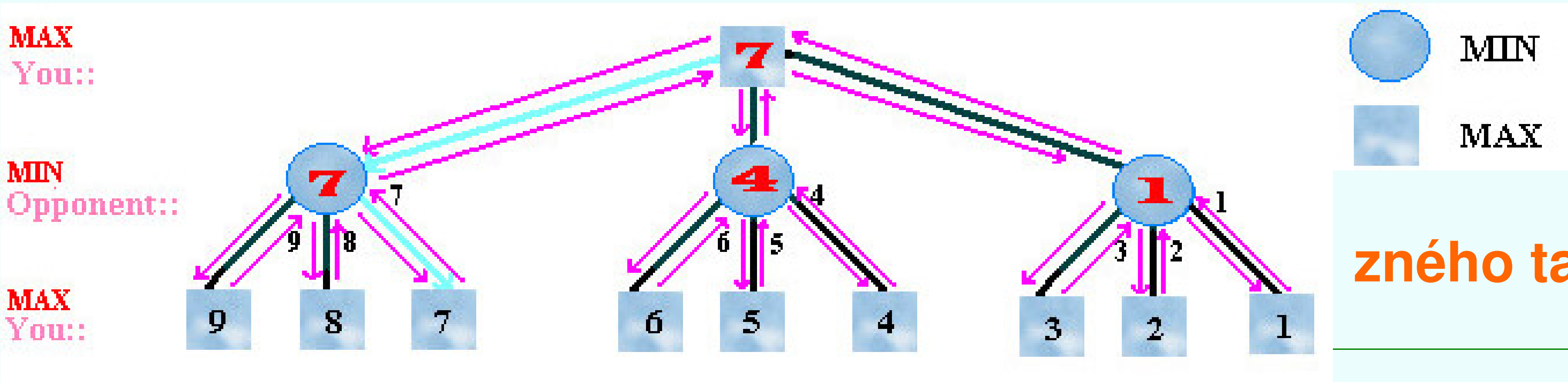
Algoritmus minimaxu

Používá ohodnocení uzlů, které zpravidla vyjadřuje pravděpodobnost, s jakou lze ve hře dosáhnout vítězství.

Proto – na úrovni hráče vybíráme maximum,

– na úrovni protihráče vybíráme minimum.





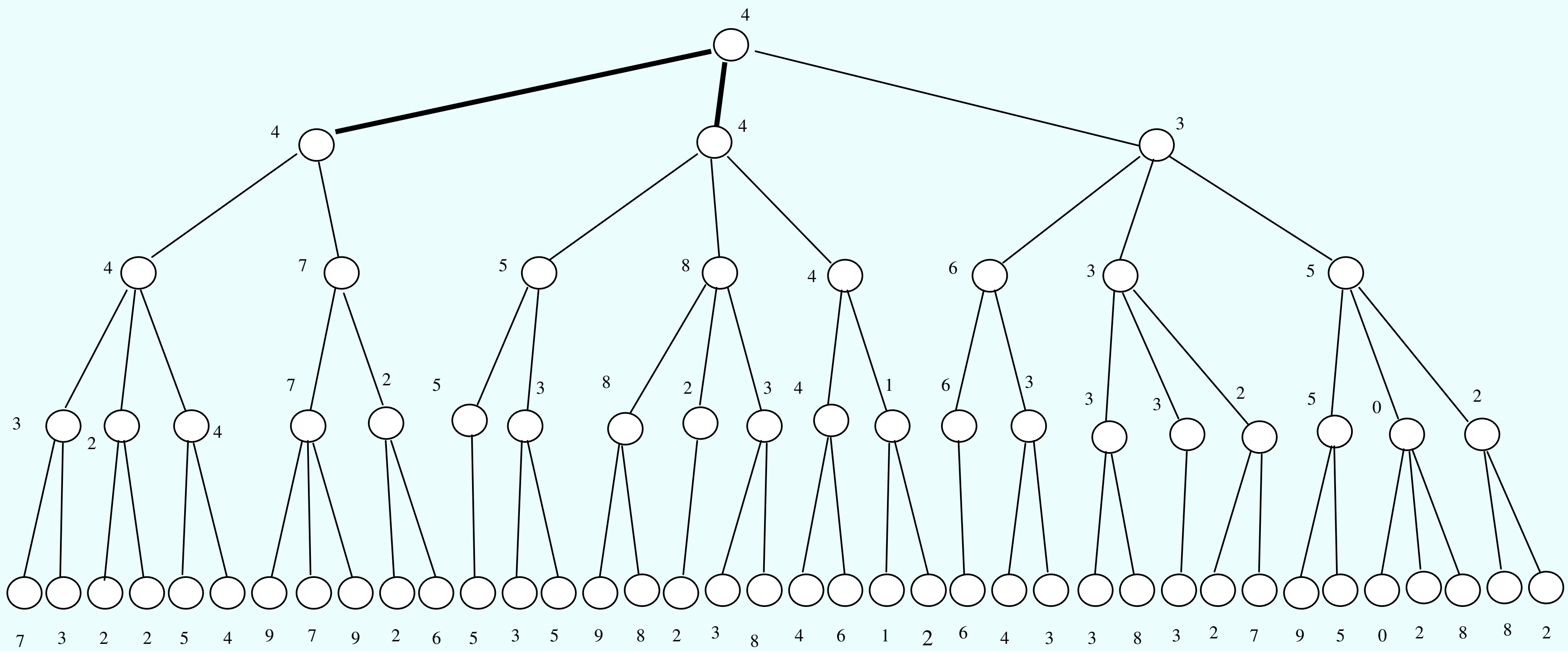
Hledání vítězného tahu

algoritmem minimaxu:

PATH (stored)

```
minmax(u) {              //   u is the node you want to score
  if  u is a leaf  return ( score of u );
  else  if u in a min node
           for all childern of u: v1,...,  vn
   return min({minmax(v1),...,minmax(vn)});
 else
     for all childern of u: v1,  ...,  vn;
   return max({minmax(v1),...,minmax(vn)});
}
```

Příklad: Ohodnocení uzlů stromu řešení při hledání řešení metodou minimaxu:



Příklad: Hledání řešení hry „piškvorky“ (ve variantě omezené na hrací pole 3 x 3 čtverečky) metodou minimaxu

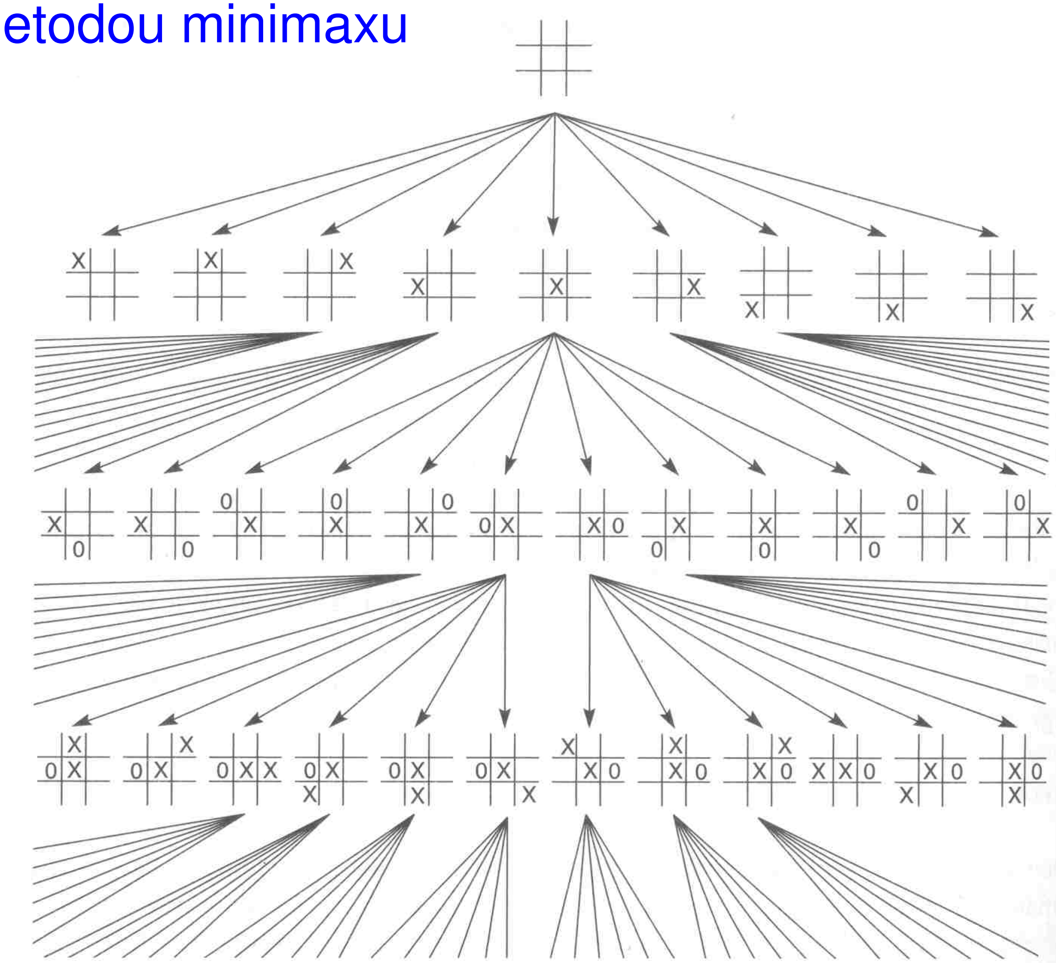


Figure II.5 Portion of the state space for tic-tac-toe.

Použijeme omezení stromu:

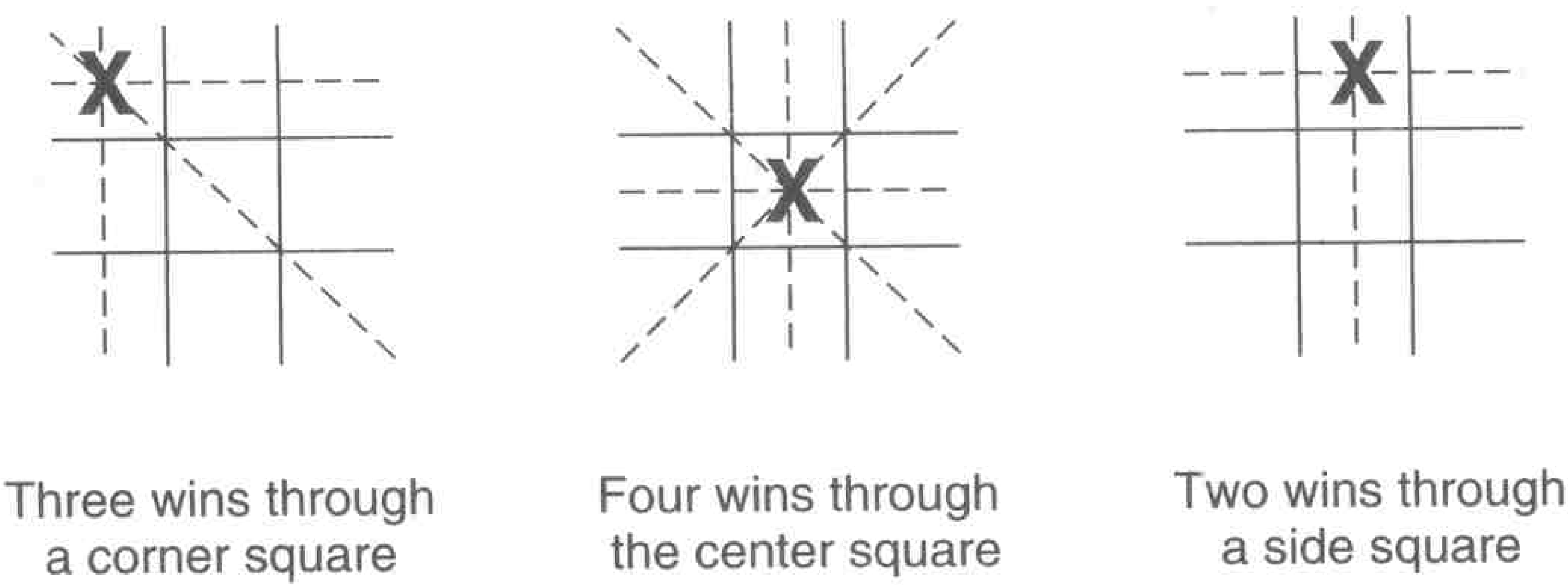


Figure 4.2 The most wins heuristic applied to the first children in tic-tac-toe.

2. Řešení úloh – hraní her

Omezený strom řešení:

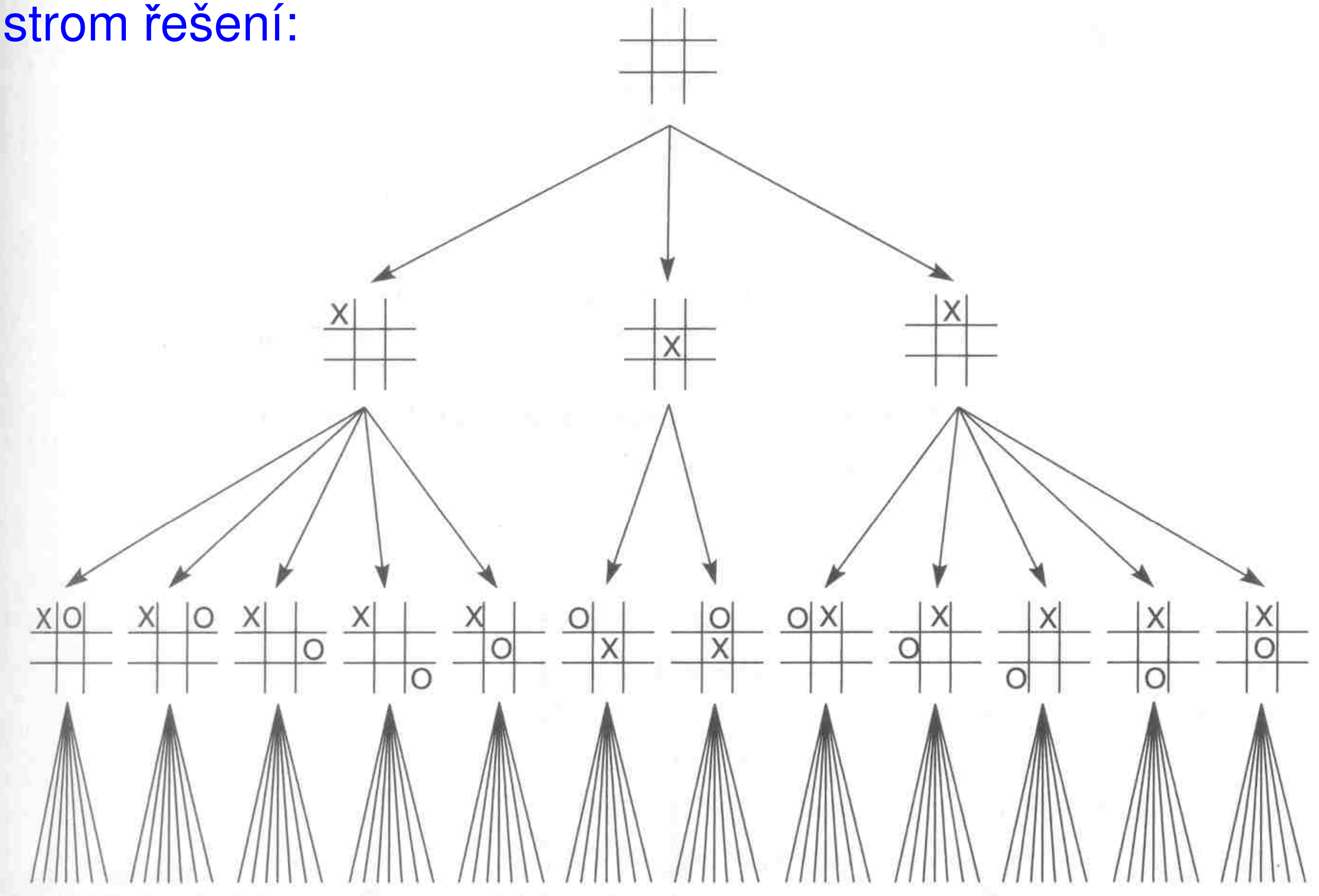


Figure 4.1 First three levels of the tic-tac-toe state space reduced by symmetry.

2. Řešení úloh – hraní her

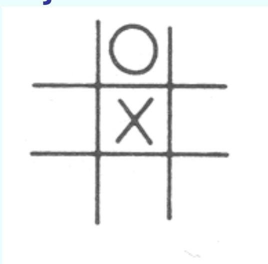
Popis úlohy a algoritmus výpočtu ohodnocující funkce:

Vložení křížku (x) do hracího pole značí tah 1. hráče a bude reprezentováno uzlem stromu, v němž budeme volit maximum (MAX).

Vložení kroužku (o) do hracího pole značí tah protihráče a bude reprezentováno uzlem stromu, v němž budeme volit minimum (MIN).

Výpočet ohodnocující funkce definujeme takto:

- nachází-li se 1. hráč (MAX-uzel) v pozici výherce, pak $f(n_i) = \infty$ (maxint);
- nachází-li se protihráč (MIN-uzel) v pozici výherce, pak $f(n_i) = -\infty$;
- není-li ani jeden z hráčů v pozici výherce, pak $f(n_i)$ vyčíslíme jako počet kompletních řádků, sloupců a diagonál (souvislých trojic křížků) hracího pole, které 1. hráč ještě může vyplnit – (minus) počet kompletních řádků, sloupců a diagonál (souvislých trojic kroužků) hracího pole, které ještě může vyplnit protihráč. Např. pro situaci:



$$f(n_i) = 6 - 4 = 2 \quad .$$

Vkládání uzlů:

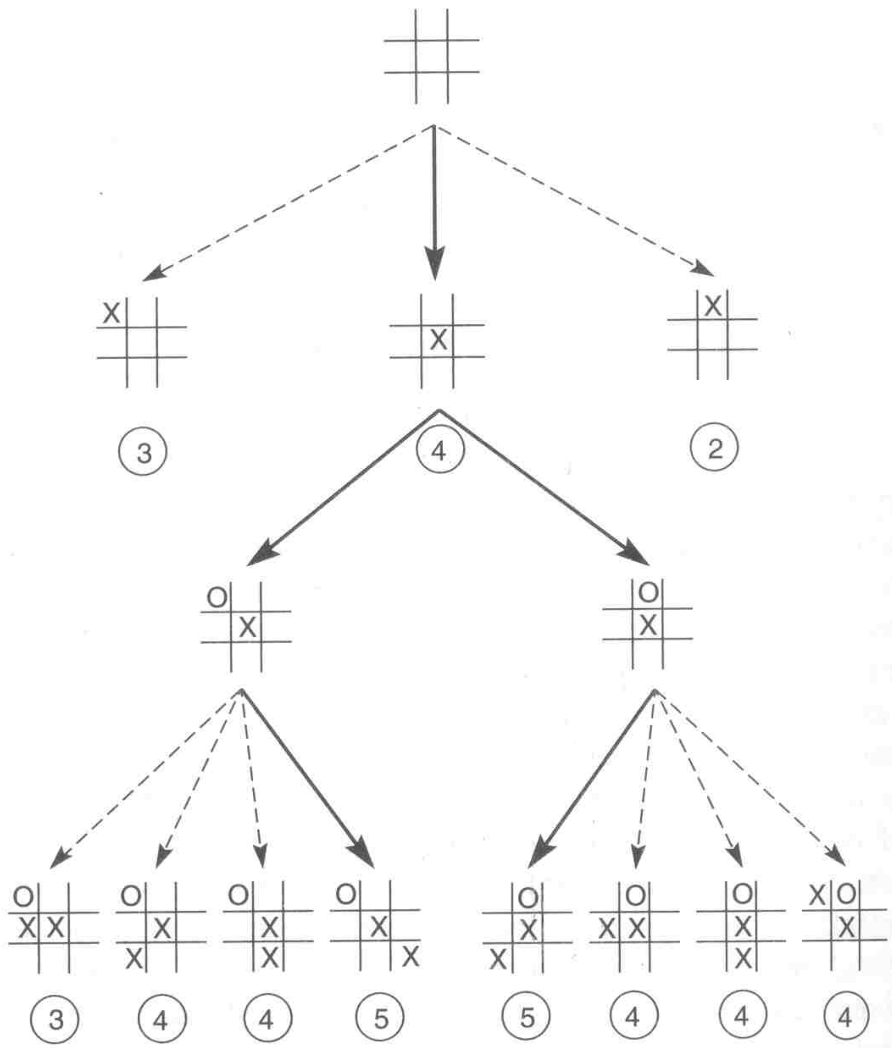


Figure 4.3 Heuristically reduced state space for tic-tac-toe.

1. tah 1. hráče + 1. tah protihráče (část stromu řešení):

2. Řešení úloh – hraní her

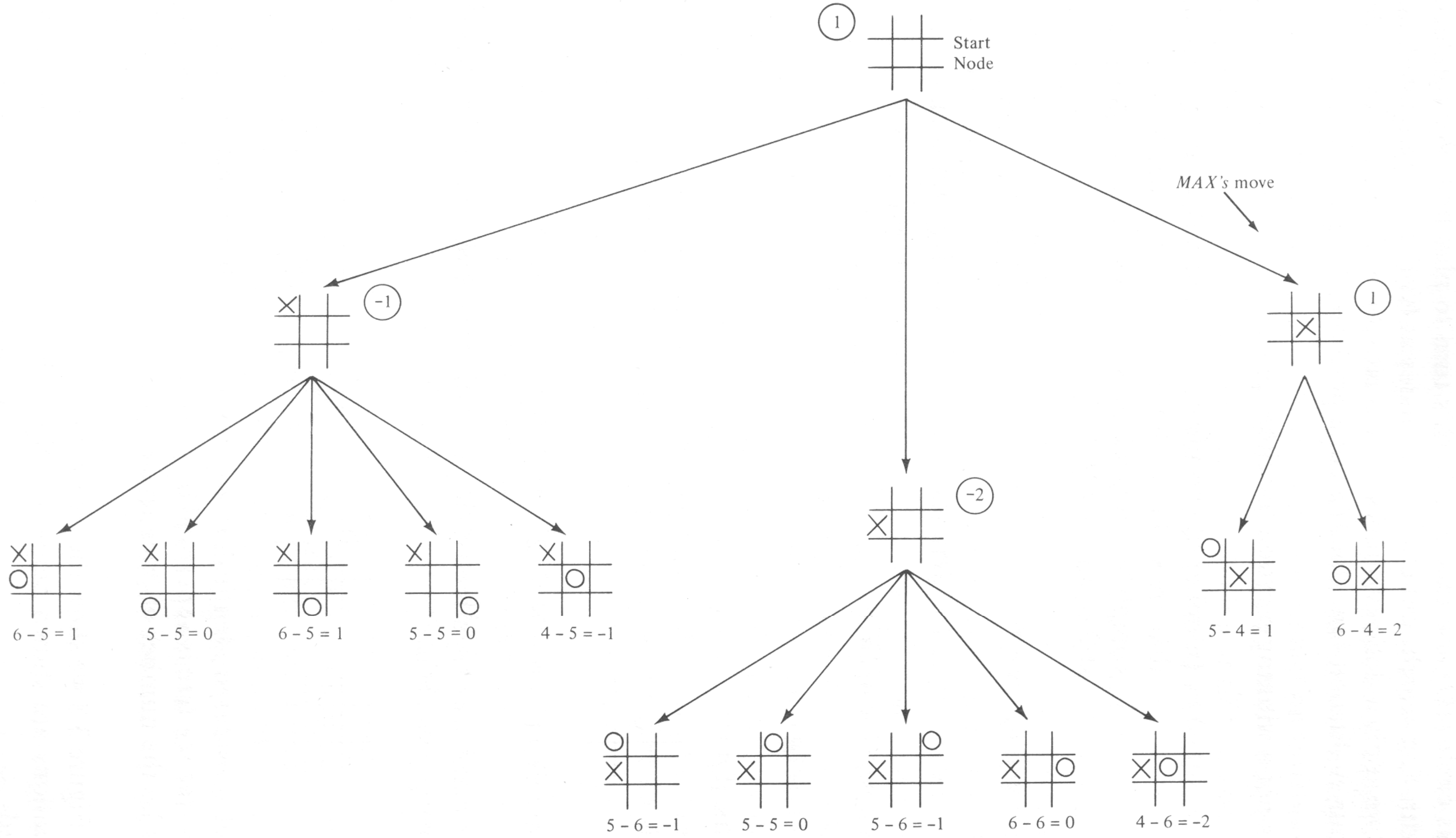


Fig. 3.8 Minimax applied to tic-tac-toe (stage 1).

2. tah obou hráčů (pouze část stromu řešení):

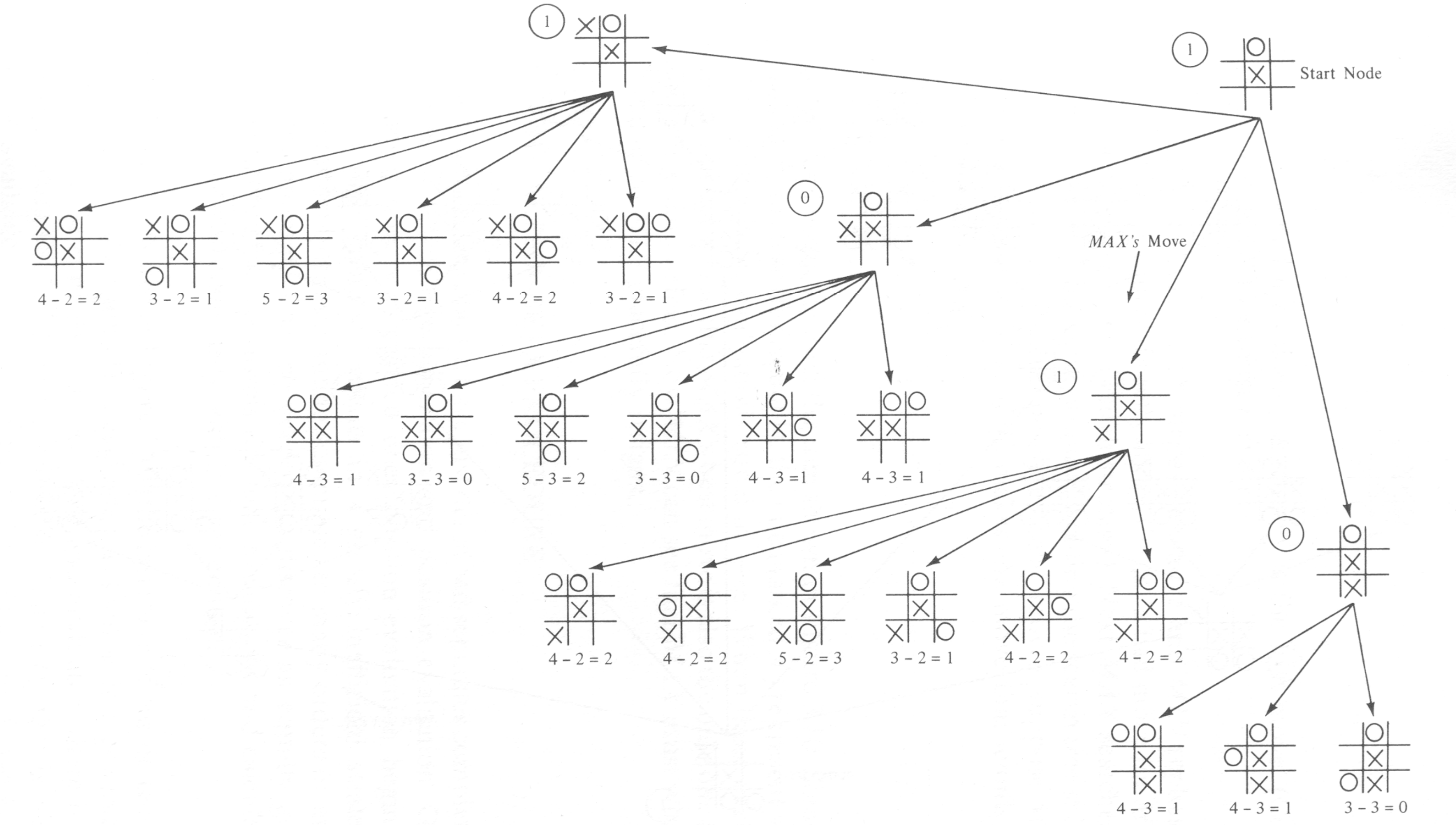


Fig. 3.9 Minimax applied to tic-tac-toe (stage 2).

3. tah obou hráčů vedoucí k vítězství:

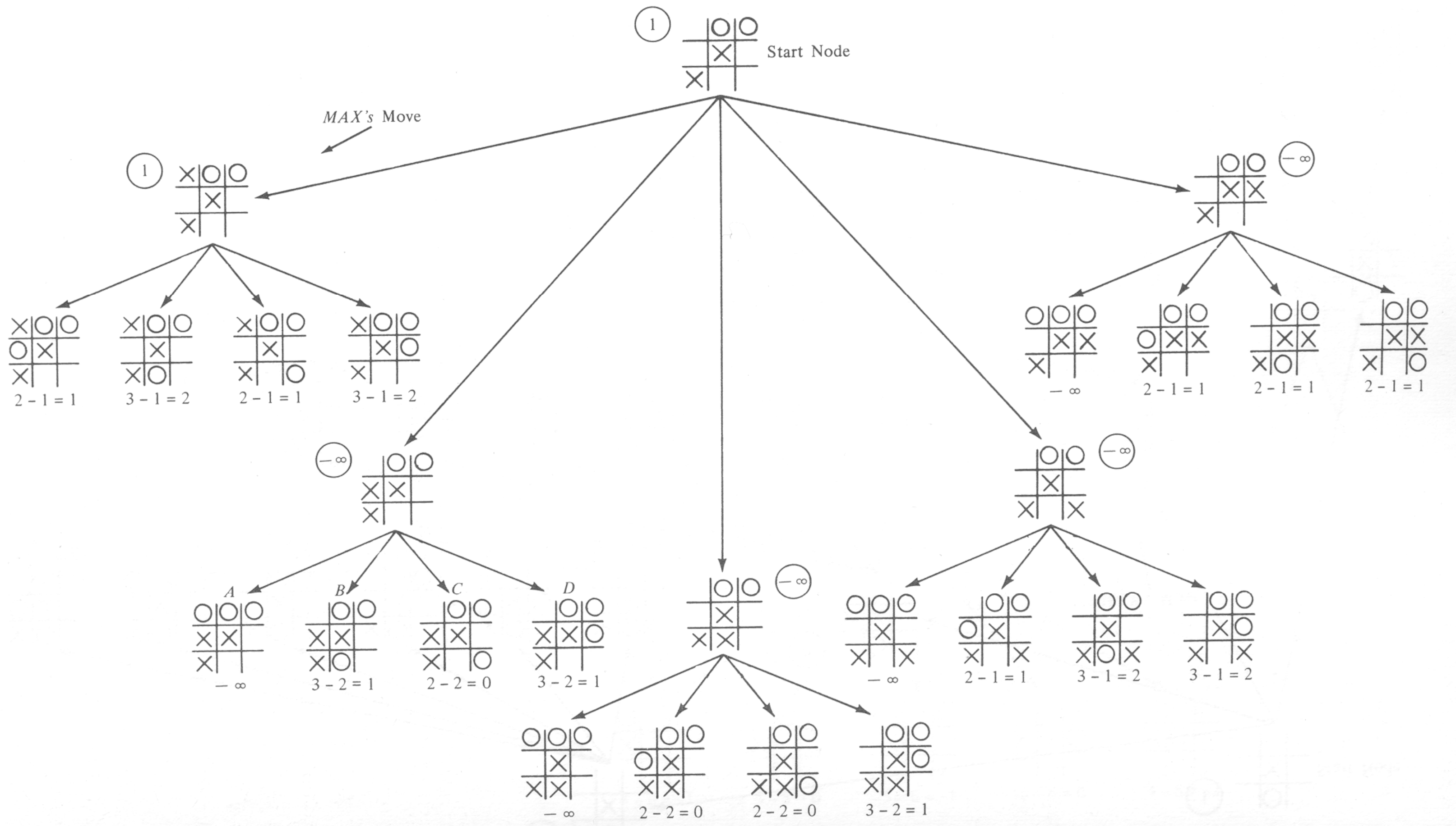
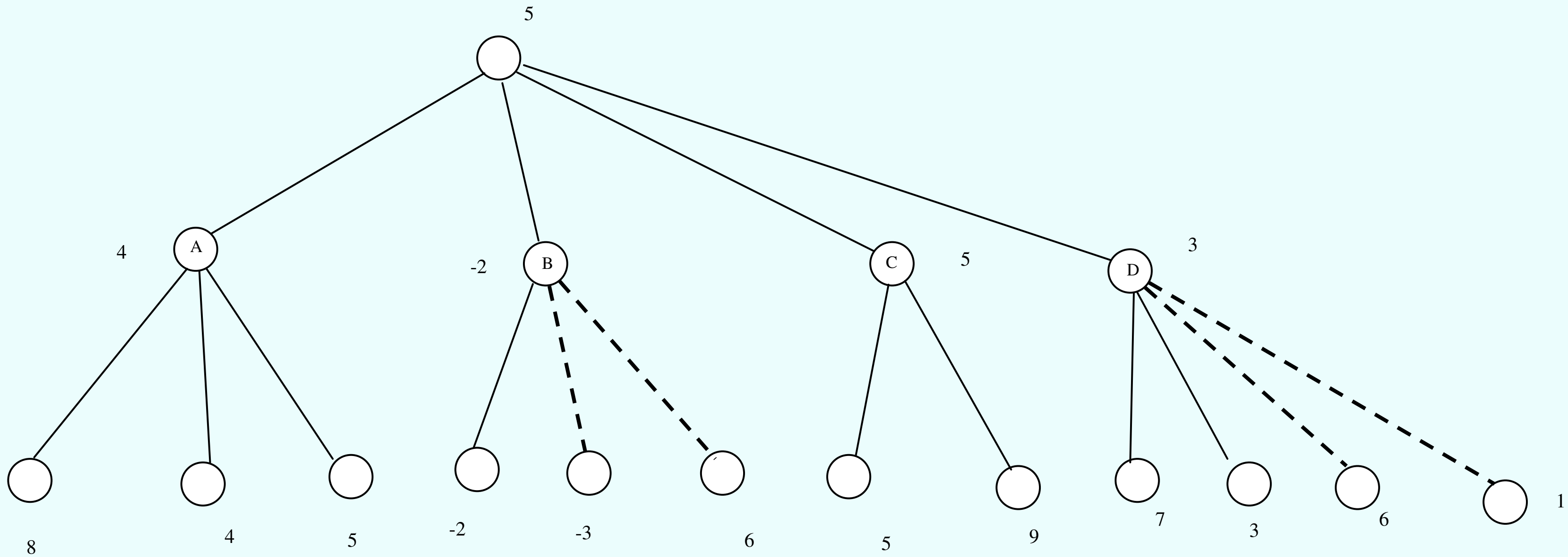


Fig. 3.10 Minimax applied to tic-tac-toe (stage 3).

Algoritmus alfa–beta prořezávání

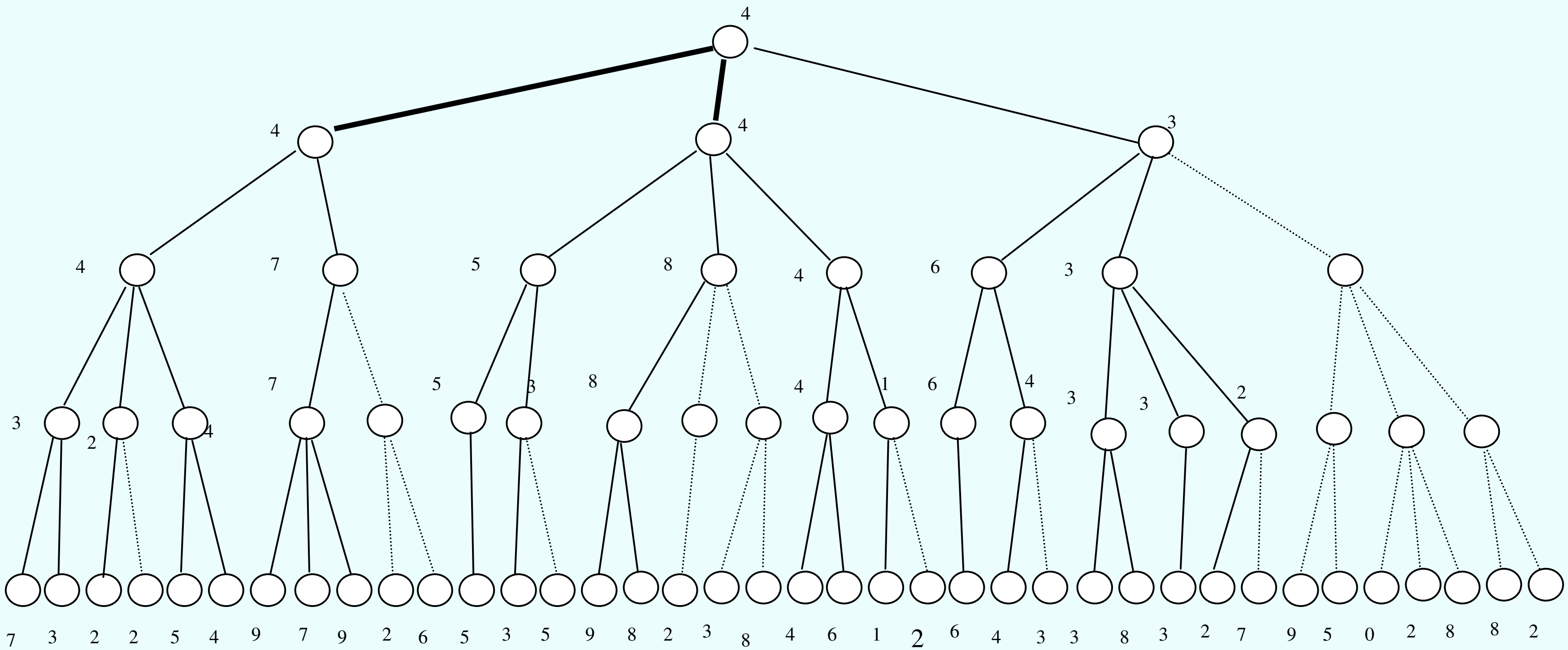
Podstata algoritmu: Když víme, že nějaká větev je špatná (tj. horší než doposud nalezená nejlepší), nepotřebujeme už vědět, jak moc je špatná. Navíc nás ani nezajímá, zdali tam nebude lepší podvětev, protože soupeř by se jí jistě uměl vyhnout.

Př.:

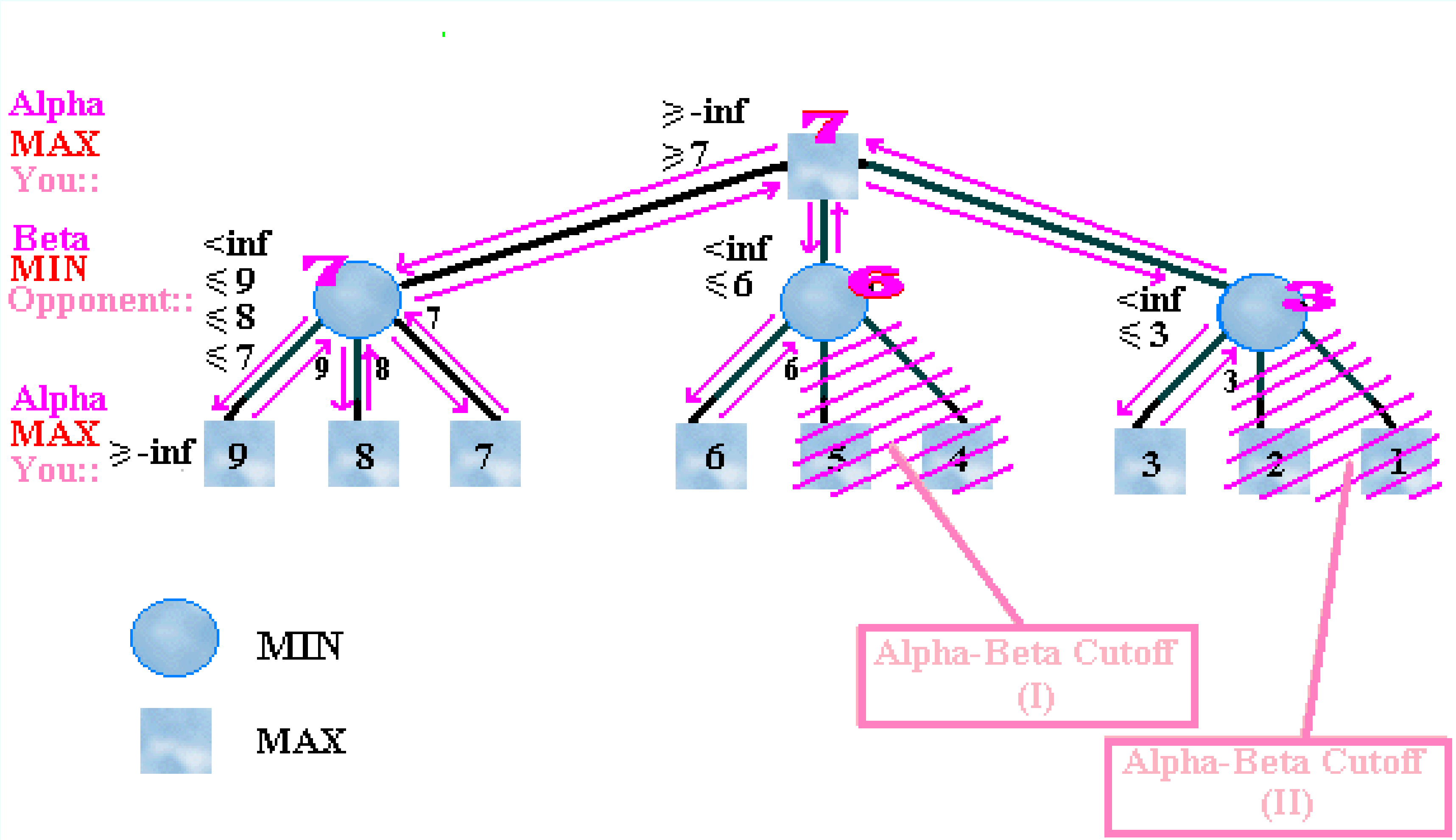


2. Řešení úloh – hraní her

Postup: Pro množinu uzlů na jedné úrovni si pamatujeme nějaké maximum (nebo minimum, pokud jde o minimalizující úroveň) a pokud zjistíme, že ohodnocení synů dalšího uzlu v řadě je menší (větší) než toto maximum (minimum), nepotřebujeme tento uzel dál rozvíjet, protože víme, že už minimaxovou hodnotu svého otce neovlivní (tečkované větve).



Hledání vítězného tahu algoritmem alfa–beta prořezávání:



Algoritmus alfa–beta prořezávání v pseudokódu:

```
function MINIMAX-AB(N, A, B) is ;;                    //  Here A is always less than B
 begin
   if N is a leaf then
     return the estimated score of this leaf
   else
     Set Alpha value of N to -infinity
           and Beta value of N to +infinity;
     if N is a Min node then
      For each successor Ni of N loop
       Let Val be MINIMAX-AB(Ni,A,Min{B,Beta of N});
       Set Beta value of N to Min{Beta value of N,Val};
     When A >= Beta value of N then
     Return Beta value of N
      endloop;
      Return Beta value of N;
   else
    For each successor Ni of N loop
     Let Val be MINIMAX-AB(Ni,Max{A,Alpha value of N},B);
     Set Alpha value of N to Max{Alpha value of N, Val};
     When Alpha value of N >= B then
     Return Alpha value of N
      endloop;
    Return Alpha value of N;
 end MINIMAX-AB;
```

Jiná varianta implementace algoritmu:

```
minimax(in game board, in best_opposing_score, out score chosen_score,
                                                      out move chosen_move)
begin
        best_score = -infinity;
        Gt_generate_moves(current_mimx_data,moves_list);
        for (i = 0 to moves_list.num_moves-1) do
            new_board = board;
            Gt_move(board, moves_list[i],the_unmove);
            minimax(board, the_score,the_move);
            Gt_unmove(board,the_unmove);
            if (the_score > best_score) then
               best_score = the_score;
               best_move = moves_list[i];
               if (best_score > best_opposing_score) then
              cut;/* the opponent will not allow you to reach this node
                         in real life so there is no since in continuing*/
          endif
            endif
         enddo
         chosen_move = best_move;
      Gt_evaluate(current_mimx_data,chosen_score,best_score);
end.
```

Př.: **Strom řešení „piškvorek“ „prořezaný“ algoritmem alfa–beta prořezávání**

2. Řešení úloh – hraní her

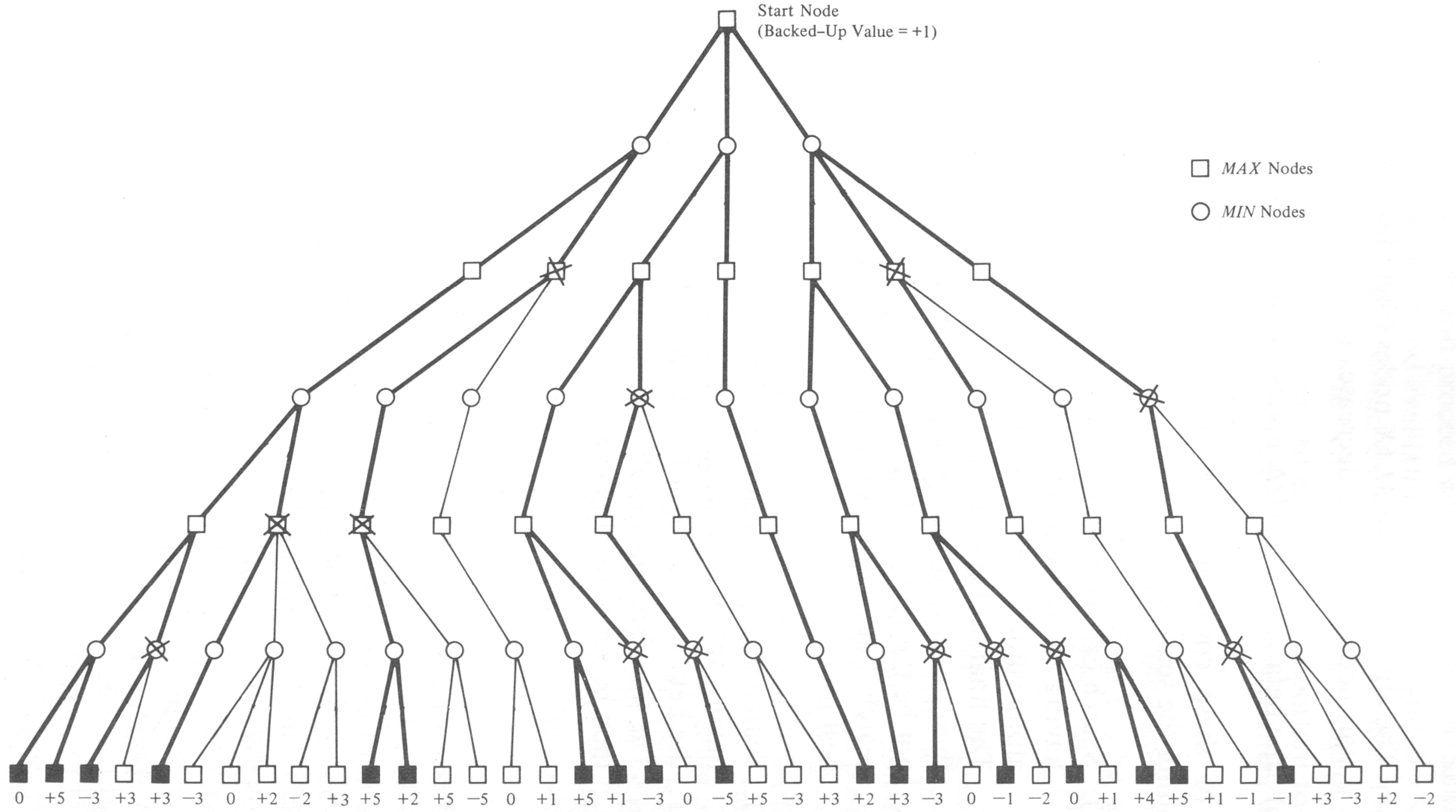
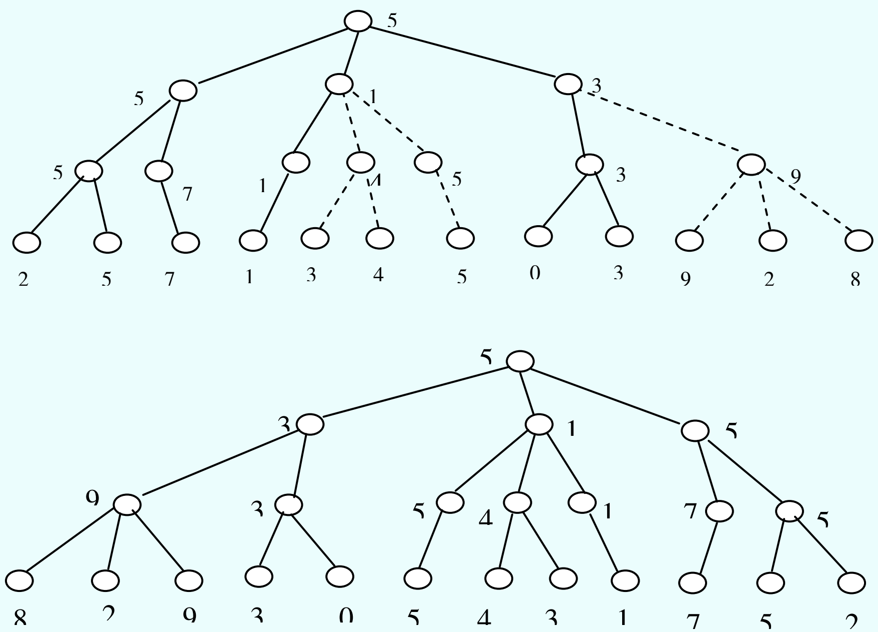


Fig. 3.12 An example illustrating the alpha-beta search procedure.

Efektivnost algoritmu alfa–beta prořezávání

Z porovnání algoritmů minimaxu a alfa–beta prořezávání vyplývá vlastnost, že algoritmus alfa–beta prořezávání vrací hodnotu, kterou by vrátil původní algoritmus, aniž by musel projít celým stromem, což je vlastně to, co jsme potřebovali získat. Z předchozích obrázků je patrna jistá efektivita, ale je toto efektivní vždy? Může se stát, že je ořezávání lepší nebo naopak horší? Na následujícím obrázku jsou ukázány dva příklady:



Oba stromy jsou ohodnoceny algoritmem alfa–beta ořezávání. Alfa–beta metoda se neuplatní, protože strom nemá dost zde pater. Aby se uplatnila, jsou potřeba alespoň čtyři úrovně (viz výše). Na prvním obrázku byl strom algoritmem ořezán poměrně hodně, na druhém vůbec. Přitom jediná odlišnost, která je mezi oběma obrázky, je pořadí, v jakém jsou v obou stromech znázorněny tahy. Důkaz, proč tomu tak je, si proveďte jako cvičení.

Úloha k samostatnému řešení: sedm mostů v Königsbergu

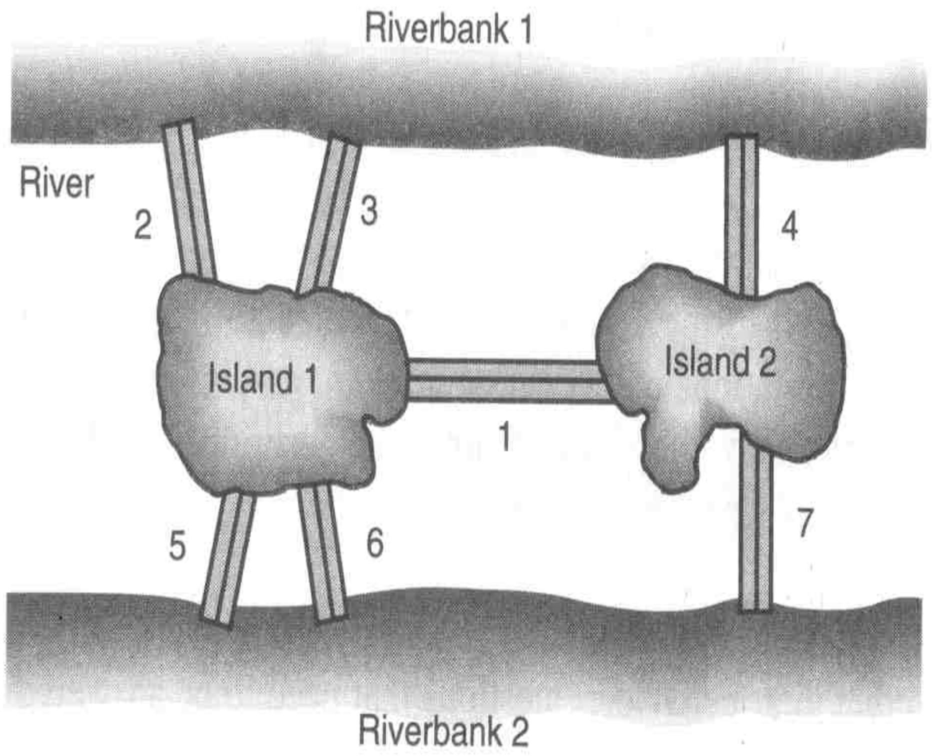


Figure 3.1 The city of Königsberg.

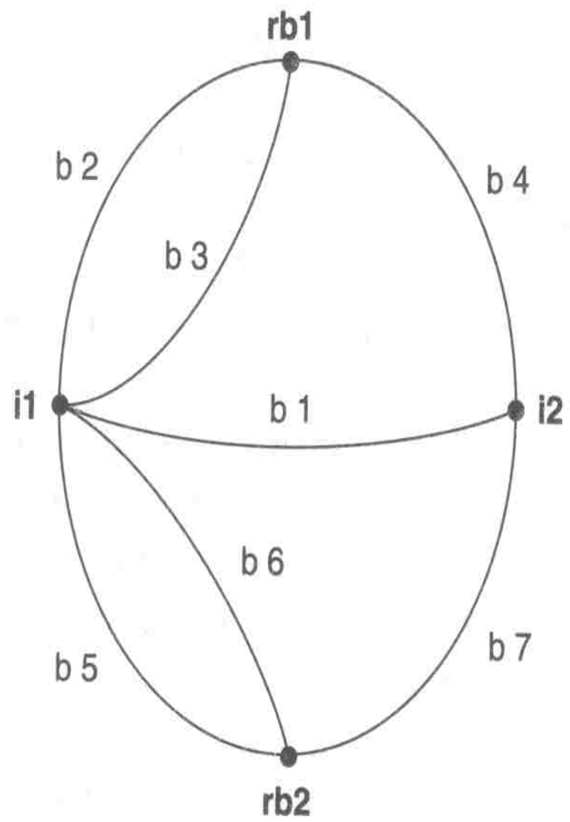


Figure 3.2 Graph of the Königsberg bridge system.