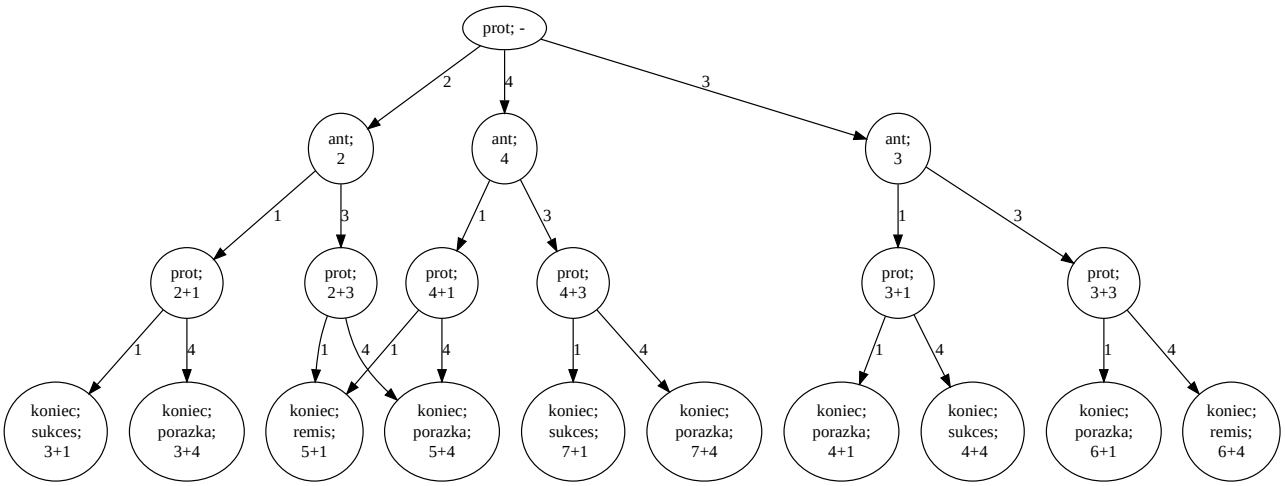
Graf gry + algorytm MinMax

Graf gry służy do przedstawienia wszystkich możliwości (stanów) gry oraz zależności między tymi stanami za pomocą grafu skierowanego. Oto główne założenia:

1. Gra jest dwu-osobowa, naprzemiennie wybierają akcję protagonista (nasz gracz) i antagonista (wróg) dopóki nie dojdzie do zakończenia gry.
2. Zakłada się, że gra w końcu się skończy, istnieje wiele możliwych zakończeń.
3. Obaj gracza dokładnie znają konsekwencje swoich wyborów, a wybór każdej akcji daje znane rezultaty (brak czynnika losowego, gra jest całkowicie przewidywalna).
4. Pierwszy ruch może dokonać protagonista albo antagonista.
5. Pojedynczy węzeł w grafie opisuje całkowicie stan gry, a krawędź opisuje akcję jaką może wybrać protagonista lub antagonista.
6. Aktualny stan gry jest całkowicie znany przez obu graczy.
7. Węzły mogą należeć do jednego z 3 typów:
    - Kolej na ruch protagonisty, węzły wychodzące opisują akcje dokonywane przez protagonistę.
    - Kolej na ruch antagonisty, węzły wychodzące opisują akcje dokonywane przez antagonistę.
    - Koniec gry, zawiera informację o wyniku gry.
8. W zależności od stanu obaj gracze mogą mieć dostępne różne akcje.
9. W celu znacznego uproszczenia implementacji łatwiej nie łączyć kilku identycznych stanów w jeden węzeł, czyli najłatwiej jest pozostać przy drzewie i nie upraszczać grafu.
10. W grafie nie ma cykli.

Przykład 1. grafu gry

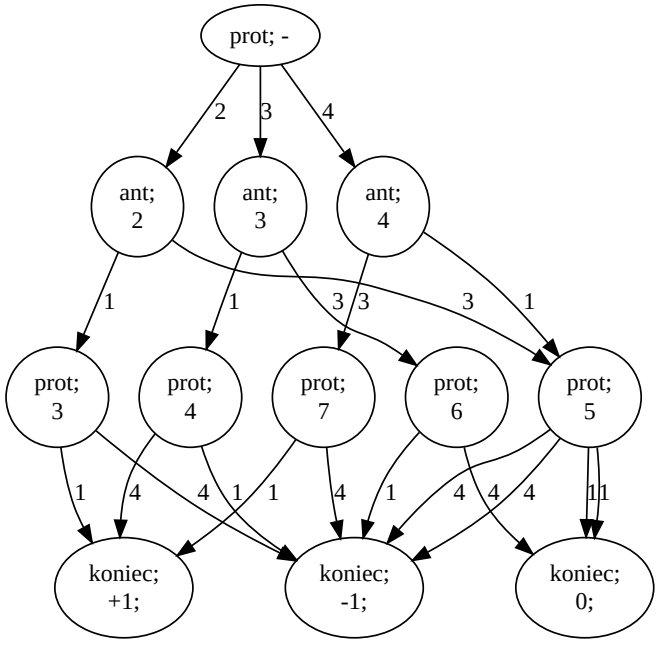
Grę trwa 3 ruchy. W pierwszym ruchu protagonista wybiera liczbę 2, 3 albo 4; w drugim antagonista wybiera liczbę 1 albo 3; w trzecim ruchu protagonista wybiera liczbę 1 albo 4. Grę wygra protagonista jeśli po zsumowaniu wynik pędzie podzielny bez reszty przez 4, a remis jeśli wynik będzie parzysty, ale nie podzielny przez 4.



wykres wygenerowany został przez program GraphVis (dostępny również jako aplikacja online) za pomocą skryptu widocznego poniżej.

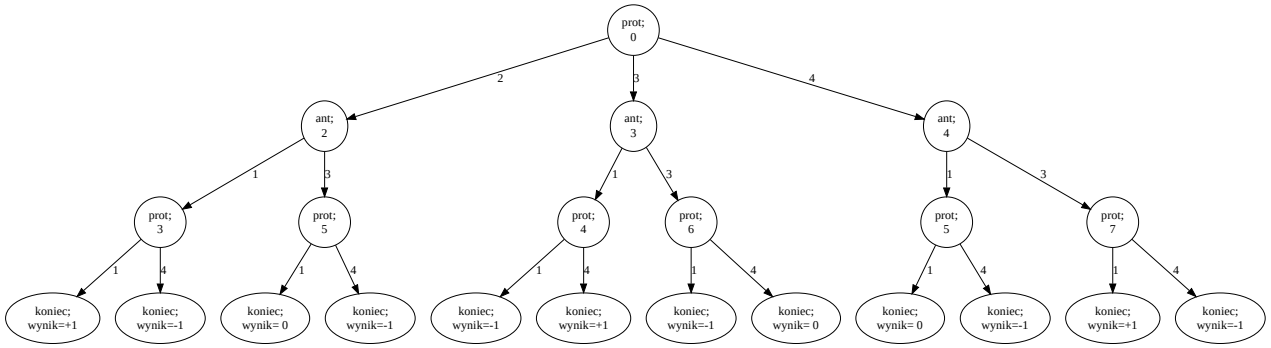
```
digraph G {

"prot; -" -> "ant;\n 2" [label = "2"];
"prot; -" -> "ant;\n 3" [label = "3"];
"prot; -" -> "ant;\n 4" [label = "4"];

"ant;\n 2" -> "prot;\n 2+1" [label = "1"];
"ant;\n 2" -> "prot;\n 2+3" [label = "3"];

"ant;\n 3" -> "prot;\n 3+1" [label = "1"];
"ant;\n 3" -> "prot;\n 3+3" [label = "3"];

"ant;\n 4" -> "prot;\n 4+1" [label = "1"];
"ant;\n 4" -> "prot;\n 4+3" [label = "3"];

"prot;\n 2+1" -> "koniec;\n sukces;\n 3+1" [label = "1"];
"prot;\n 2+1" -> "koniec;\n porazka;\n 3+4" [label = "4"];

"prot;\n 2+3" -> "koniec;\n remis;\n 5+1" [label = "1"];
"prot;\n 2+3" -> "koniec;\n porazka;\n 5+4" [label = "4"];

"prot;\n 3+1" -> "koniec;\n porazka;\n 4+1" [label = "1"];
"prot;\n 3+1" -> "koniec;\n sukces;\n  4+4" [label = "4"];

"prot;\n 3+3" -> "koniec;\n porazka;\n 6+1" [label = "1"];
"prot;\n 3+3" -> "koniec;\n remis;\n 6+4" [label = "4"];

"prot;\n 4+1" -> "koniec;\n remis;\n 5+1" [label = "1"];
"prot;\n 4+1" -> "koniec;\n porazka;\n 5+4" [label = "4"];

"prot;\n 4+3" -> "koniec;\n sukces;\n  7+1" [label = "1"];
"prot;\n 4+3" -> "koniec;\n porazka;\n 7+4" [label = "4"];
}
```

Ten sam graf można przedstawić w innej formie na przykład tak, aby można go było przetwarzać algorytmem MinMax. W tym celu etykietki służące do poinformowania o rezultacie końcowym gry należy zamienić liczbami tak, aby im większa liczba tym lepszy bym wynik rozgrywki. Nie ma jedynie słusznego rozwiązania jakie konkretne liczby mają się tu znaleźć. Przykład zmiany zapisu tego samego przebiegu gry:




```
digraph G {

"prot; -" -> "ant;\n 2" [label = "2"];
"prot; -" -> "ant;\n 3" [label = "3"];
"prot; -" -> "ant;\n 4" [label = "4"];

"ant;\n 2" -> "prot;\n 3" [label = "1"];
"ant;\n 2" -> "prot;\n 5" [label = "3"];

"ant;\n 3" -> "prot;\n 4" [label = "1"];
"ant;\n 3" -> "prot;\n 6" [label = "3"];

"ant;\n 4" -> "prot;\n 5" [label = "1"];
"ant;\n 4" -> "prot;\n 7" [label = "3"];

"prot;\n 3" -> "koniec;\n +1;\n" [label = "1"];
"prot;\n 3" -> "koniec;\n -1;\n" [label = "4"];

"prot;\n 5" -> "koniec;\n 0;\n" [label = "1"];
"prot;\n 5" -> "koniec;\n -1;\n" [label = "4"];

"prot;\n 4" -> "koniec;\n -1;\n" [label = "1"];
"prot;\n 4" -> "koniec;\n +1;\n" [label = "4"];

"prot;\n 6" -> "koniec;\n -1;\n" [label = "1"];
"prot;\n 6" -> "koniec;\n 0;\n" [label = "4"];

"prot;\n 5" -> "koniec;\n 0;\n" [label = "1"];
"prot;\n 5" -> "koniec;\n -1;\n" [label = "4"];

"prot;\n 7" -> "koniec;\n +1;\n" [label = "1"];
"prot;\n 7" -> "koniec;\n -1;\n" [label = "4"];
}
```

W celu ułatwienia implementacji można zapisać ten graf również w postaci drzewa:



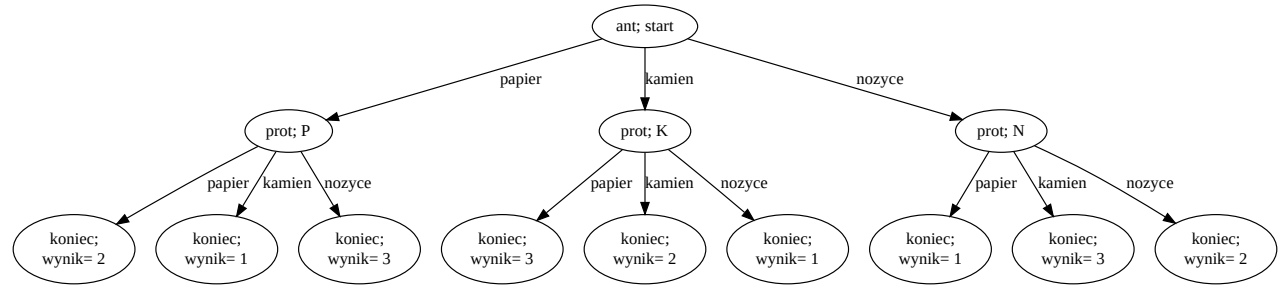

```
digraph G {

"prot;\n 0" -> "ant;\n 2" [label = "2"];
"prot;\n 0" -> "ant;\n 3" [label = "3"];
"prot;\n 0" -> "ant;\n 4" [label = "4"];

"ant;\n 2" -> "prot;\n 3+1" [label = "1"];
"ant;\n 2" -> "prot;\n 3+3" [label = "3"];

"ant;\n 3" -> "prot;\n 4+1" [label = "1"];
"ant;\n 3" -> "prot;\n 4+3" [label = "3"];

"ant;\n 4" -> "prot;\n 5+1" [label = "1"];
"ant;\n 4" -> "prot;\n 5+3" [label = "3"];

"prot;\n 3+1" [label = "prot;\n 3"];
"prot;\n 3+3" [label = "prot;\n 3"];

"prot;\n 4+1" [label = "prot;\n 4"];
"prot;\n 4+3" [label = "prot;\n 4"];

"prot;\n 5+1" [label = "prot;\n 5"];
"prot;\n 5+3" [label = "prot;\n 5"];

"prot;\n 3+1" -> "koniec;\n wynik=+1;\n 3+1+1" [label = "1"];
"prot;\n 3+1" -> "koniec;\n wynik=-1;\n 3+1+4" [label = "4"];

"prot;\n 3+3" -> "koniec;\n wynik= 0;\n 3+3+1" [label = "1"];
"prot;\n 3+3" -> "koniec;\n wynik=-1;\n 3+3+4" [label = "4"];

"prot;\n 4+1" -> "koniec;\n wynik=-1;\n 4+1+1" [label = "1"];
"prot;\n 4+1" -> "koniec;\n wynik=+1;\n 4+1+4" [label = "4"];

"prot;\n 4+3" -> "koniec;\n wynik=-1;\n 4+3+1" [label = "1"];
"prot;\n 4+3" -> "koniec;\n wynik= 0;\n 4+3+4" [label = "4"];

"prot;\n 5+1" -> "koniec;\n wynik= 0;\n 5+1+1" [label = "1"];
"prot;\n 5+1" -> "koniec;\n wynik=-1;\n 5+1+4" [label = "4"];

"prot;\n 5+3" -> "koniec;\n wynik=+1;\n 5+3+1" [label = "1"];
"prot;\n 5+3" -> "koniec;\n wynik=-1;\n 5+3+4" [label = "4"];

"koniec;\n wynik=+1;\n 3+1+1" [label = "koniec;\n wynik=+1"];
"koniec;\n wynik=-1;\n 3+1+4" [label = "koniec;\n wynik=-1"];

"koniec;\n wynik= 0;\n 3+3+1" [label = "koniec;\n wynik= 0"];
"koniec;\n wynik=-1;\n 3+3+4" [label = "koniec;\n wynik=-1"];

"koniec;\n wynik=-1;\n 4+1+1" [label = "koniec;\n wynik=-1"];
"koniec;\n wynik=+1;\n 4+1+4" [label = "koniec;\n wynik=+1"];

"koniec;\n wynik=-1;\n 4+3+1" [label = "koniec;\n wynik=-1"];
"koniec;\n wynik= 0;\n 4+3+4" [label = "koniec;\n wynik= 0"];

"koniec;\n wynik= 0;\n 5+1+1" [label = "koniec;\n wynik= 0"];
"koniec;\n wynik=-1;\n 5+1+4" [label = "koniec;\n wynik=-1"];

"koniec;\n wynik=+1;\n 5+3+1" [label = "koniec;\n wynik=+1"];
"koniec;\n wynik=-1;\n 5+3+4" [label = "koniec;\n wynik=-1"];

}
```

Przykład 2. grafu gry, papier-kamień-nożyce

Kolejny dotyczy gry podobnej do zabawy „papier-kamień-nożyce”. Modyfikacja polega na tym, że najpierw antagonista wybiera jedną z trzech opcji (papier, kamień, nożyce), a następnie wybiera protagonista jedną z tych samych opcji. Zwycięstwo następuje gdy:

- protagonista wybrał papier, a antagonista kamień,
- protagonista wybrał nożyce, a antagonista papier,
- protagonista wybrał kamień, a antagonista nożyce.

Porażka występuje w odwrotnych sytuacjach, natomiast remis, jeśli obaj graczy wybrali to samo. Poniżej widać graf tej gry, z etykietkami końcowymi zastąpionymi liczbami zgodnie ze schematem: remis = 2, porażka = 1, zwycięstwo = 3.




```
digraph G {
"ant; start" -> "prot; P" [label = "papier"]
"ant; start" -> "prot; K" [label = "kamien"]
"ant; start" -> "prot; N" [label = "nozyce"]

"prot; P" -> "PP" [label = "papier"]
"prot; P" -> "PK" [label = "kamien"]
"prot; P" -> "PN" [label = "nozyce"]

"prot; K" -> "KP" [label = "papier"]
"prot; K" -> "KK" [label = "kamien"]
"prot; K" -> "KN" [label = "nozyce"]

"prot; N" -> "NP" [label = "papier"]
"prot; N" -> "NK" [label = "kamien"]
"prot; N" -> "NN" [label = "nozyce"]

"PP" [label = "koniec;\n wynik= 2"]
"PK" [label = "koniec;\n wynik= 1"]
"PN" [label = "koniec;\n wynik= 3"]

"KP" [label = "koniec;\n wynik= 3"]
"KK" [label = "koniec;\n wynik= 2"]
"KN" [label = "koniec;\n wynik= 1"]

"NP" [label = "koniec;\n wynik= 1"]
"NK" [label = "koniec;\n wynik= 3"]
"NN" [label = "koniec;\n wynik= 2"]

}
```

Algorytm MinMax

Algorytm ten służy do podpowiadaniu protagoniście i antagoniście jaką akcję należy wybrać. Podczas wyboru akcji dla protagonisty zakłada się, że antagonista wybierze najlepszą dla siebie akcję i vice-versa. Aby algorytm działał stany końcowe grafu gry (opisujące zakończenie i wynik rozgrywki) muszą mieć liczbowy opis wyniku rozgrywki. Nie ma jednego słusznego sposobu numerowania wyników, należy jedynie nadać takie liczby rzeczywiste, aby im wynik końcowy rozgrywki był lepszy, tym wyższa liczba go opisywała. Ponieważ algorytm ten podpowie jakie akcje powinien wybrać antagonista i protagonista, dlatego za pomocą tego algorytmu można przewidzieć końcowy wynik rozgrywki, jeśli obaj gracze będą grali najlepiej jak to możliwe.

Algorytm ten działa w sposób rekurencyjny i jest uruchamiany początkowo dla korzenia grafu gry (wskazującego na pierwszy ruch). Dla tego węzła algorytm wykona:

- Jeśli aktualny węzeł to „koniec gry” (węzeł „koniec”) wtedy zwróć wynik równy opisowi tego węzła.
- Jeśli aktualny węzeł dotyczy ruchu protagonisty (węzeł „prot”), wtedy wybierz tę akcję, która daje maksymalny (najlepszy dla protagonisty) wynik, a następnie sam zwróć ten maksymalny wynik. Inaczej mówiąc dla takiego węzła należy znaleźć taki węzeł potomny, który daje maksymalny wynik i samemu go zwrócić. W przypadku remisu należy wybrać jedną z akcji dających maksymalny wynik.
- Jeśli aktualny węzeł dotyczy ruchu protagonisty (węzeł „ant”), wtedy wybierz tę akcję, która daje minimalny (najlepszy dla antagonisty) wynik, a następnie sam zwróć ten minimalny wynik. Inaczej mówiąc dla takiego węzła należy znaleźć taki węzeł potomny, który daje minimalny wynik i samemu go zwrócić. W przypadku remisu należy wybrać jedną z akcji dających minimalny wynik.

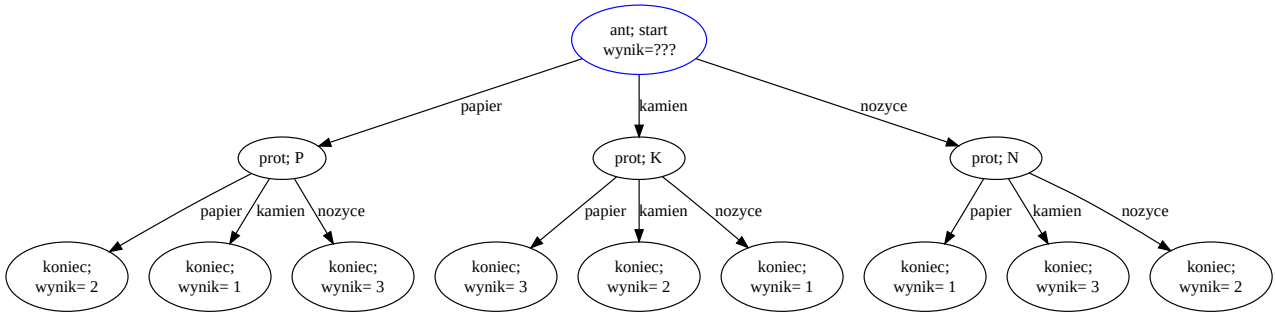
Inaczej można to napisać w postaci matematycznej:

$$MinMax(s)=\begin{cases} wynik(s) & jeśli\ s\ jest\ końcowe \\ max_a(MinMax(a(s))) & jeśli\ s\ dotyczy\ ruchu\ protagonisty \\ min_a(MinMax(a(s))) & jeśli\ s\ dotyczy\ ruchu\ antagonisty \end{cases},$$

gdzie *s* to aktualny węzeł w grafie gry, *a(s)* to węzeł uzyskany po użyciu akcji *a* dla aktualnego węzła *s*.

Przykład 2. grafu gry, działanie algorytmu MinMax

Najpierw algorytm uruchamiany jest dla korzenia grafu gry



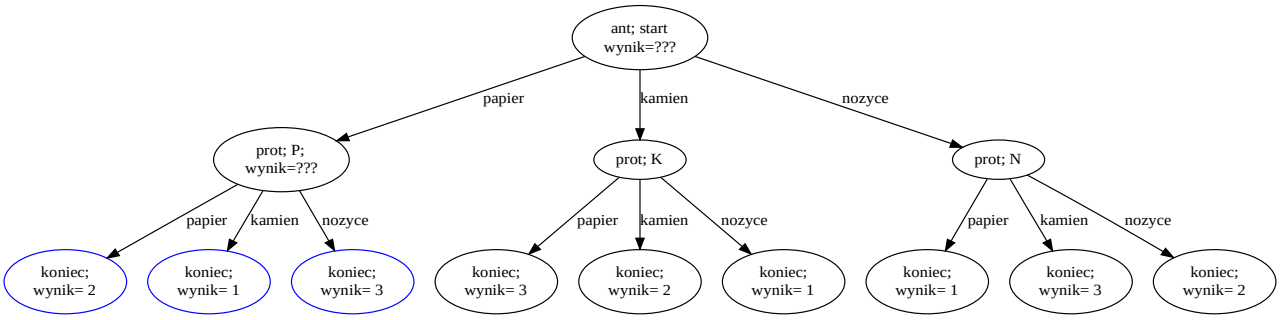

```
digraph G {
"ant; start" -> "prot; P" [label = "papier"]
"ant; start" -> "prot; K" [label = "kamien"]
"ant; start" -> "prot; N" [label = "nozyce"]

"prot; P" -> "PP" [label = "papier"]
"prot; P" -> "PK" [label = "kamien"]
"prot; P" -> "PN" [label = "nozyce"]

"prot; K" -> "KP" [label = "papier"]
"prot; K" -> "KK" [label = "kamien"]
"prot; K" -> "KN" [label = "nozyce"]

"prot; N" -> "NP" [label = "papier"]
"prot; N" -> "NK" [label = "kamien"]
"prot; N" -> "NN" [label = "nozyce"]

"ant; start" [label = "ant; startu wynik=???" color = "blue"]
"prot; P" [label = "prot; P"],
"prot; K" [label = "prot; K"]
"prot; N" [label = "prot; N"]

"PP" [label = "koniec;\n wynik= 2"]
```

"PK" [label = "koniec;\n wynik= 1"]
"PN" [label = "koniec;\n wynik= 3"]

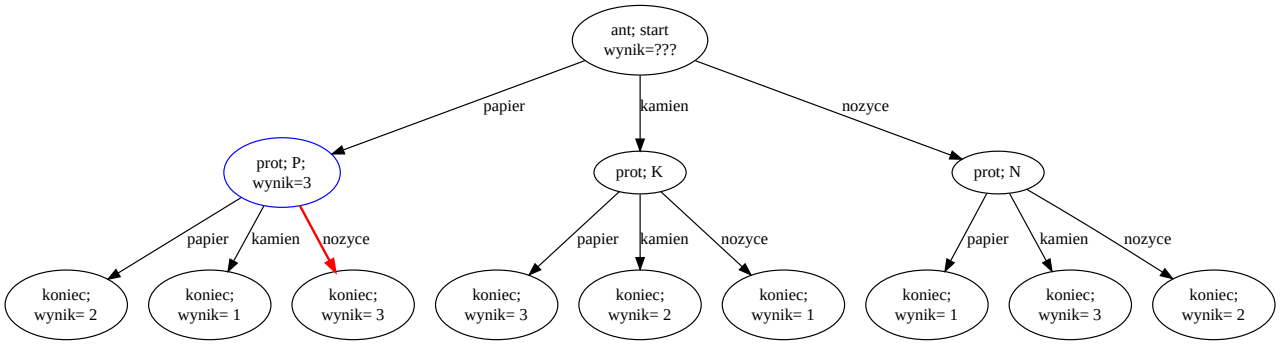
"KP" [label = "koniec;\n wynik= 3"]
"KK" [label = "koniec;\n wynik= 2"]
"KN" [label = "koniec;\n wynik= 1"]

"NP" [label = "koniec;\n wynik= 1"]
"NK" [label = "koniec;\n wynik= 3"]
"NN" [label = "koniec;\n wynik= 2"]
}

Następnie algorytm rekursywnie odpalany jest dla wszystkich węzłów potomnych, aż dochodzi do pierwszych węzłów końcowych, które natychmiast zwracają konkretny wynik.



Na podstawie wyników węzłów potomnych można dla węzła „prot; P” ustalić wynik, jak również wybrać konkretną akcję (maksimum z 2; 1; 3).



digraph G {
"ant; start" -> "prot; P" [label = "papier"]
"ant; start" -> "prot; K" [label = "kamien"]
"ant; start" -> "prot; N" [label = "nozyce"]

"prot; P" -> "PP" [label = "papier"]
"prot; P" -> "PK" [label = "kamien"]
"prot; P" -> "PN" [label = "nozyce" style = bold color="red"]

"prot; K" -> "KP" [label = "papier"]
"prot; K" -> "KK" [label = "kamien"]
"prot; K" -> "KN" [label = "nozyce"]

"prot; N" -> "NP" [label = "papier"]
"prot; N" -> "NK" [label = "kamien"]
"prot; N" -> "NN" [label = "nozyce"]

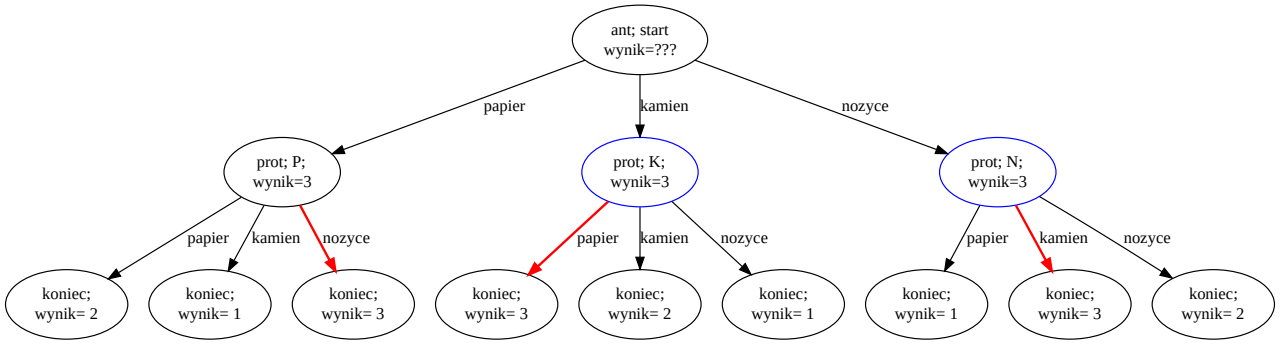
"ant; start" [label = "ant; start\n wynik=???"]
"prot; P" [label = "prot; P\n wynik=3" color = "blue"]
"prot; K" [label = "prot; K"]
"prot; N" [label = "prot; N"]

"PP" [label = "koniec;\n wynik= 2"]
"PK" [label = "koniec;\n wynik= 1"]
"PN" [label = "koniec;\n wynik= 3"]

"KP" [label = "koniec;\n wynik= 3"]
"KK" [label = "koniec;\n wynik= 2"]
"KN" [label = "koniec;\n wynik= 1"]

"NP" [label = "koniec;\n wynik= 1"]
"NK" [label = "koniec;\n wynik= 3"]
"NN" [label = "koniec;\n wynik= 2"]
}

Analogicznie można wyliczyć wyniki dla węzłów „prot; K” i „prot; N”.



digraph G {
"ant; start" -> "prot; P" [label = "papier"]
"ant; start" -> "prot; K" [label = "kamien"]
"ant; start" -> "prot; N" [label = "nozyce"]

"prot; P" -> "PP" [label = "papier"]
"prot; P" -> "PK" [label = "kamien"]
"prot; P" -> "PN" [label = "nozyce" style = bold color="red"]

"prot; K" -> "KP" [label = "papier" style = bold color="red"]
"prot; K" -> "KK" [label = "kamien"]
"prot; K" -> "KN" [label = "nozyce"]

"prot; N" -> "NP" [label = "papier"]
"prot; N" -> "NK" [label = "kamien" style = bold color="red"]
"prot; N" -> "NN" [label = "nozyce"]

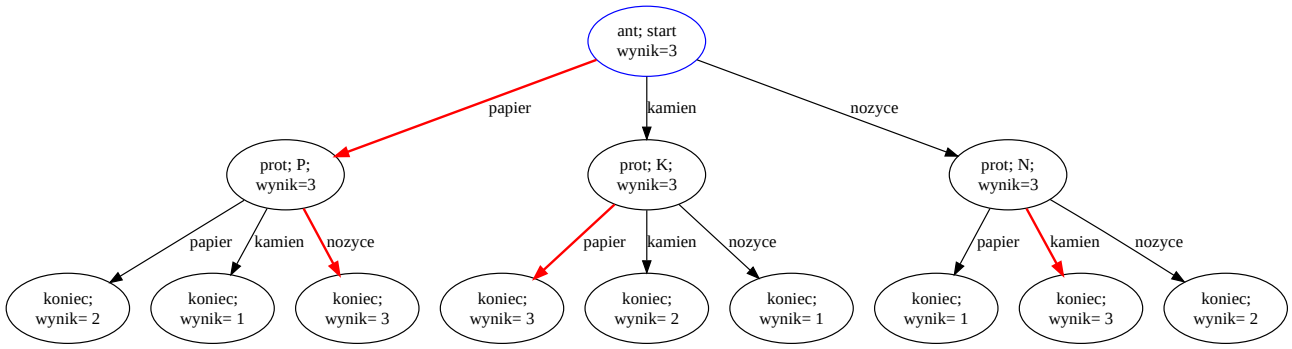
"ant; start" [label = "ant; start\n wynik=???"]
"prot; P" [label = "prot; P\n wynik=3"]
"prot; K" [label = "prot; K\n wynik=3" color = "blue"]
"prot; N" [label = "prot; N\n wynik=3" color = "blue"]

"PP" [label = "koniec;\n wynik= 2"]
"PK" [label = "koniec;\n wynik= 1"]
"PN" [label = "koniec;\n wynik= 3"]

"KP" [label = "koniec;\n wynik= 3"]
"KK" [label = "koniec;\n wynik= 2"]
"KN" [label = "koniec;\n wynik= 1"]

"NP" [label = "koniec;\n wynik= 1"]
"NK" [label = "koniec;\n wynik= 3"]
"NN" [label = "koniec;\n wynik= 2"]
}

Na końcu można wyliczyć wynik dla węzła „ant; start” i tym samym wynik dla całego grafu gry.

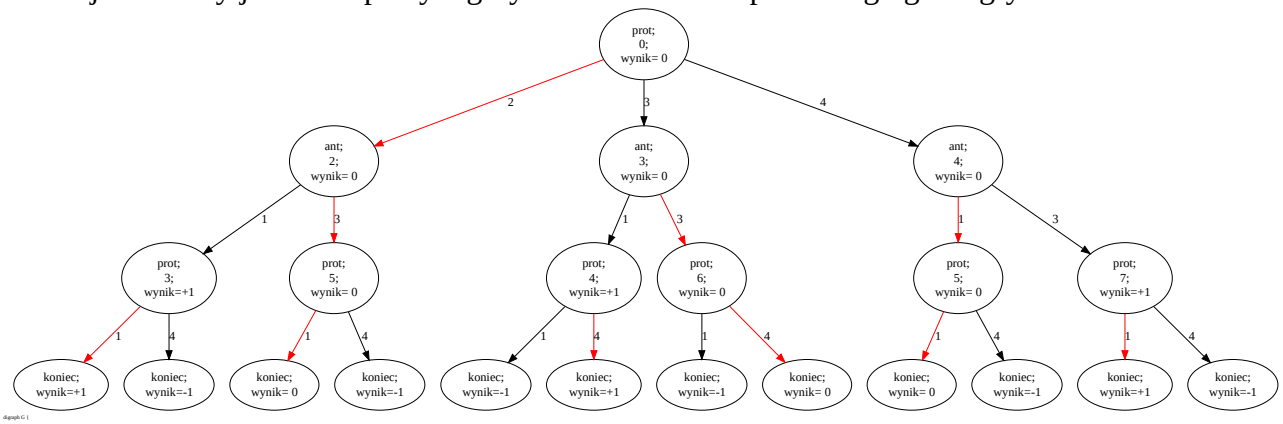



```
digraph G {
"ant; start" -> "prot; P" [label = "papier" style = bold color="red"]
"ant; start" -> "prot; K" [label = "kamien" ]
"ant; start" -> "prot; N" [label = "nozyce" ]

"prot; P" -> "PP" [label = "papier" ]
"prot; P" -> "PK" [label = "kamien" ]
"prot; P" -> "PN" [label = "nozyce" style = bold color="red"]

"prot; K" -> "KP" [label = "papier" style = bold color="red"]
"prot; K" -> "KK" [label = "kamien"]
"prot; K" -> "KN" [label = "nozyce"]

"prot; N" -> "NP" [label = "papier"]
"prot; N" -> "NK" [label = "kamien" style = bold color="red"]
"prot; N" -> "NN" [label = "nozyce"]

"ant; start" [label = "ant; start\n wynik=3" color = "blue"]
"prot; P" [label = "prot; P\n wynik=3" ]
"prot; K" [label = "prot; K\n wynik=3" ]
"prot; N" [label = "prot; N\n wynik=3" ]

"PP" [label = "koniec;\n wynik= 2" ]
"PK" [label = "koniec;\n wynik= 1" ]
"PN" [label = "koniec;\n wynik= 3" ]

"KP" [label = "koniec;\n wynik= 3" ]
"KK" [label = "koniec;\n wynik= 2" ]
"KN" [label = "koniec;\n wynik= 1" ]

"NP" [label = "koniec;\n wynik= 1" ]
"NK" [label = "koniec;\n wynik= 3" ]
"NN" [label = "koniec;\n wynik= 2" ]
}
```


Wynik działania algorytmu MinMax mówi, że w tej grze protagonista bez względu na wybory antagonisty zawsze może wygrać, jeśli będzie dokonywał właściwych wyborów.

Przykład 1. grafu gry, działanie algorytmu MinMax

Poniżej widoczny jest efekt pracy algorytmu MinMax dla pierwszego grafu gry.




```
digraph G {
"prot;\n 0\n wynik= 0" -> "ant;\n 2\n wynik= 0" [label = "2" color="red"];
"prot;\n 0\n wynik= 0" -> "ant;\n 3\n wynik= 0" [label = "3"];
"prot;\n 0\n wynik= 0" -> "ant;\n 4\n wynik= 0" [label = "4"];

"ant;\n 2\n wynik= 0" -> "prot;\n 3\n wynik= +1" [label = "1"];
"ant;\n 2\n wynik= 0" -> "prot;\n 5\n wynik= 0" [label = "3" color="red"];

"ant;\n 3\n wynik= 0" -> "prot;\n 3\n wynik= +1" [label = "1"];
"ant;\n 3\n wynik= 0" -> "prot;\n 5\n wynik= 0" [label = "3" color="red"];

"ant;\n 4\n wynik= 0" -> "prot;\n 4\n wynik= +1" [label = "1" color="red"];
"ant;\n 4\n wynik= 0" -> "prot;\n 6\n wynik= 0" [label = "3"];

"prot;\n 3\n wynik= +1" [label = "prot;\n 3\n wynik= +1"];
"prot;\n 5\n wynik= 0" [label = "prot;\n 5\n wynik= 0"];

"prot;\n 4\n wynik= +1" [label = "prot;\n 4\n wynik= +1"];
"prot;\n 6\n wynik= 0" [label = "prot;\n 6\n wynik= 0"];

"prot;\n 3\n wynik= +1" -> "koniec;\n wynik= +1\n 2\n 1" [label = "1" color="red"];
"prot;\n 3\n wynik= +1" -> "koniec;\n wynik= -1\n 2\n 4" [label = "4"];

"prot;\n 5\n wynik= 0" -> "koniec;\n wynik= 0\n 2\n 3" [label = "3" color="red"];
"prot;\n 5\n wynik= 0" -> "koniec;\n wynik= -1\n 2\n 4" [label = "4"];

"prot;\n 4\n wynik= +1" -> "koniec;\n wynik= -1\n 3\n 1" [label = "1"];
"prot;\n 4\n wynik= +1" -> "koniec;\n wynik= +1\n 3\n 4" [label = "4" color="red"];

"prot;\n 6\n wynik= 0" -> "koniec;\n wynik= -1\n 3\n 1" [label = "1"];
"prot;\n 6\n wynik= 0" -> "koniec;\n wynik= 0\n 3\n 4" [label = "4" color="red"];

"prot;\n 4\n wynik= +1" -> "koniec;\n wynik= 0\n 4\n 1" [label = "1" color="red"];
"prot;\n 4\n wynik= +1" -> "koniec;\n wynik= +1\n 4\n 4" [label = "4"];

"prot;\n 6\n wynik= 0" -> "koniec;\n wynik= +1\n 4\n 3" [label = "1" color="red"];
"prot;\n 6\n wynik= 0" -> "koniec;\n wynik= 0\n 4\n 4" [label = "4"];

"koniec;\n wynik= +1\n 2\n 1" [label = "koniec;\n wynik= +1"];
"koniec;\n wynik= -1\n 2\n 4" [label = "koniec;\n wynik= -1"];

"koniec;\n wynik= 0\n 2\n 3" [label = "koniec;\n wynik= 0"];
"koniec;\n wynik= -1\n 2\n 4" [label = "koniec;\n wynik= -1"];

"koniec;\n wynik= -1\n 3\n 1" [label = "koniec;\n wynik= -1"];
"koniec;\n wynik= +1\n 3\n 4" [label = "koniec;\n wynik= +1"];

"koniec;\n wynik= -1\n 3\n 1" [label = "koniec;\n wynik= -1"];
"koniec;\n wynik= 0\n 3\n 4" [label = "koniec;\n wynik= 0"];

"koniec;\n wynik= 0\n 4\n 1" [label = "koniec;\n wynik= 0"];
"koniec;\n wynik= -1\n 4\n 4" [label = "koniec;\n wynik= -1"];

"koniec;\n wynik= +1\n 4\n 1" [label = "koniec;\n wynik= +1"];
"koniec;\n wynik= -1\n 4\n 4" [label = "koniec;\n wynik= -1"];
}
```


Po dokonaniu oceny wyników działania algorytmu MinMax dla tego grafu widać, iż zakładając wybór najlepszych akcji przez obu graczy, wynik działania tej gry to zawsze remis.

Zadanie do wykonania 1.

Należy stworzyć graf dla gry podobnej do „oczka”. Pierwszy ruch ma protagonista. Zaczynamy z pustym stołem, na którym nie ma żetonów. Gracze w swojej turze mogą dołożyć na stół żeton o wartości 4, 5 albo 6. Przegrywa ten gracz, jeśli po dołożeniu żetonu suma wartości żetonów na stole

jest większa niż 21. W przypadku, gdy po dołożeniu żetonu suma wartości żetonów na stole wynosi dokładnie 21, wtedy następuje remis.

Dla ambitnych. Proszę stworzyć graf w postaci graficznej na przykład przez napisanie skryptu dla programu GraphVis (przykłady są widoczne powyżej).

Zadanie do wykonania 2.

Należy dla stworzonego wcześniej grafu gry zaimplementować algorytm MinMax, a następnie wskazać wynik końcowy, oraz kolejne akcje wykonywane przez obu graczy. W celu uproszczenie sprawdzania algorytmu, proszę założyć, że w przypadku wystąpienia remisu należy użyć żetonu o niższej wartości.