



Katedra za računarstvo

Elektronski fakultet, Univerzitet u Nišu

Veštačka inteligencija

Algoritmi traženja u Python-u (III deo)

Osnovno o Min-Max algoritmu

- ▶ Dva igrača (Min i Max) koji naizmenično povlače poteze
- ▶ Bez slučajnih elemenata
- ▶ Samo jedan je pobednik
- ▶ Traženje je definisano:
 - ▶ Inicijalnim stanjem
 - ▶ Operatorima
 - ▶ Krajnjim stanjem (pobeda, poraz, ili nerešeno)
 - ▶ Funkcijom korisnosti (utility function). Obično ima vrednosti: +1, -1, 0.



Karakteristike Min-Max algoritma

- ▶ Generiše kompletno stablo traženja
- ▶ Na svakom nivou jedan od igrača povlači potez
- ▶ Analiza celokupnog stabla daje optimalne poteze (vode ka pobedi)
- ▶ Za većinu igara nije izvodivo analizirati celo stablo, jer može biti preveliko. Rešenje je:
 - ▶ Prekidanje traženja – traženje ograničeno po dubini
 - ▶ Uvođenje funkcije za procenu vrednosti stanja na osnovu njegovih osobina
 - ▶ Primer funkcije za procenu – linearna težinska funkcija:
 - ▶ $w_1*f_1 + w_2*f_2 + \dots + w_n*f_n$



Min-Max – Algoritam

- ▶ Ako je X list stabla traženja:
 - ▶ $P(X) = 1 \Rightarrow$ Pera je pobednik
 - ▶ $P(X) = 0 \Rightarrow$ Nerešeno je
 - ▶ $P(X) = -1 \Rightarrow$ Mara je pobednik
- ▶ Ako X nije list stabla traženja:
 - ▶ $V(X) = \max \{V(c) \mid c \text{ je potomak } X\}$ ako Pera igra sledeći
 - ▶ $V(X) = \min \{V(c) \mid c \text{ je potomak } X\}$ ako Mara igra sledeća
- ▶ Min-Max algoritam odigrava najbolje poteze za oba igrača.



Reprezentacija stanja

- ▶ Da bi izbegli korišćenje string literala za predstavljanje stanja, koristićemo klasu, čiji objekti mogu da se koriste na identičan način:

```
class Symbol(object):  
    def __init__(self, name):  
        self.name = name  
    def __repr__(self):  
        return self.name
```

```
(A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z) =  
(Symbol(x) for x in "ABCDEFGHIJKLMNOPQRSTUVWXYZ")
```

```
(a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z) =  
(A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z)
```



Min-Max – Funkcije

- ▶ Glavna funkcija
 - ▶ *minimax* (stanje, dubina, moj_potez)
- ▶ Pomoćne funkcije
 - ▶ *nova_stanja* (stanje)
 - ▶ *proceni_stanje* (stanje)
 - ▶ *max_stanje* (lsv)
 - ▶ *min_stanje* (lsv)



Min-Max – Funkcija *nova_stanja*

- ▶ Funkcija *nova_stanja* određuje u koja se sve stanja može preći iz zadatog stanja (funkcija promene stanja)
- ▶ U ovom primeru funkcija promene stanja je statička, ali kod realnih problema ona dinamički određuje sledeće stanje na osnovu prethodnog i ograničenja koja postoje pri odigravanju poteza da bi on bio regularan.

```
def nova_stanja(stanje):  
    stablo = {  
        A: [B, C, D], B: [E, F], C: [G, H], D: [I, J],  
        E: [K, L], F: [M, N], G: [O], H: [P, Q],  
        I: [R, S], J: [T, U]  
    }  
  
    return stablo[stanje] if stanje in stablo else None
```



Min-Max – Funkcija *proceni_stanje*

- ▶ Funkcija *proceni_stanje* koliko konkretno stanje vodi određenog igrača ka pobjedi.
- ▶ U ovom primeru funkcija procene stanja je statička, ali kod realnih problema ona dinamički dodeljuje vrednost nekom stanju prema odabranim kriterijumima.

```
def proceni_stanje(stanje):  
    procena = {  
        K: 2, L: 3, M: 5, N: 9, O: 0, P: 7,  
        Q: 4, R: 2, S: 1, T: 5, U: 6  
    }  
  
    return procena[stanje] if stanje in procena else 0
```



Min-Max – Funkcija *max_stanje*

- ▶ Funkcija *max_stanje* određuje stanje koje ima najveću vrednost funkcije procene za zadatu listu stanja (*lsv*).
- ▶ Stanje je lista oblika (*stanje vrednost*), gde je *stanje* čvor u grafu traženja, a *vrednost* je rezultat koji daje funkcije procene za dati čvor.

```
def max_stanje(lsv):  
    return max(lsv, key=lambda x: x[1])
```

- ▶ `key=lambda x: x[1]`, kod `min` i `max` funkcija omogućava poređenje složenih tipova (tuple u ovom slučaju). Element koji će da se poredi je na 1. poziciji, a rezultat koji će da bude vraćen je celokupni tuple koji je najveći



Min-Max – Funkcije *min_stanje*

- ▶ Funkcija *min_stanje* određuje stanje koje ima najmanju vrednost funkcije procene za zadatu listu stanja (*lsv*)

```
def min_stanje(lsv):  
    return min(lsv, key=lambda x: x[1])
```

- ▶ Kod ove dve funkcije, ključ se koristi da se iz tuple podatka izvuče druga vrednost
- ▶ Primer lsv promenjive: [(A, 1), (C, 10), (M, 5)]
 - ▶ Povratna vrednost je tuple, koji se sastoji od čvora i heuristike



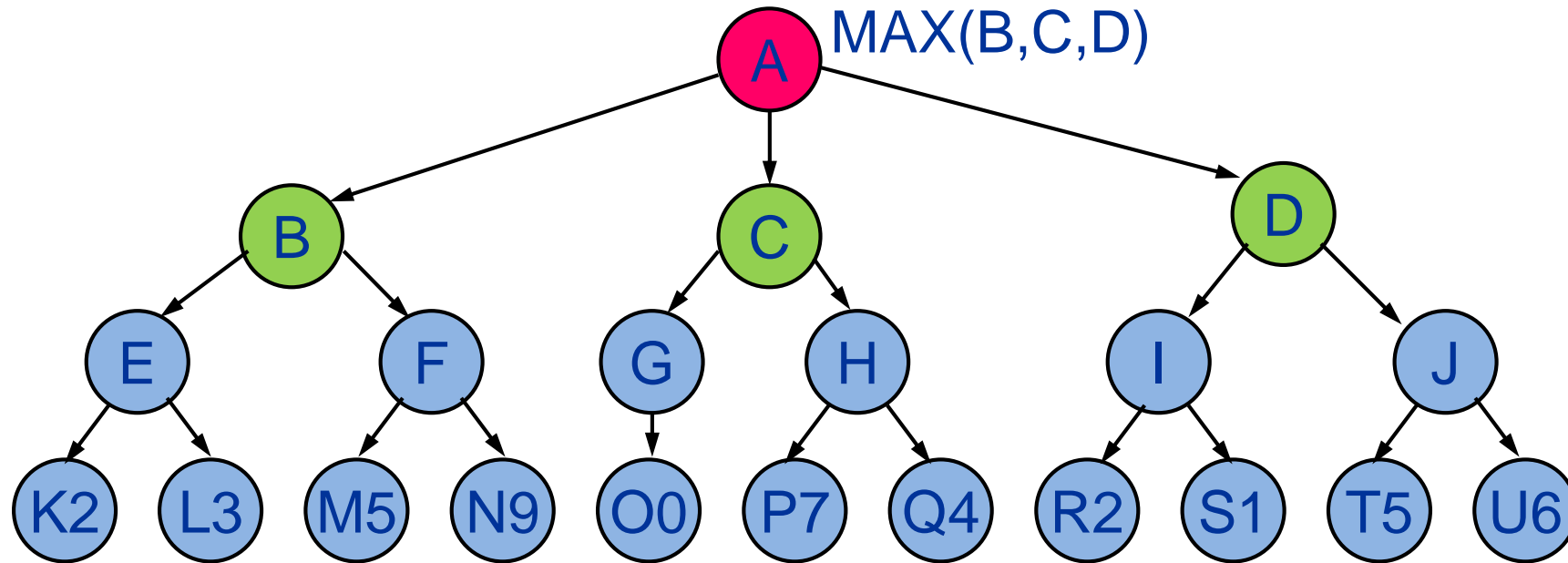
Min-Max – Funkcija *minimax*

- ▶ Glavna funkcija ***minimax*** koja određuje vrednost (funkciju procene) nekog stanja na osnovu vrednosti (funkciju procene) potomaka i igrača koji je na potezu:
 - ▶ (Računar – **True**, Igrač – **False**)

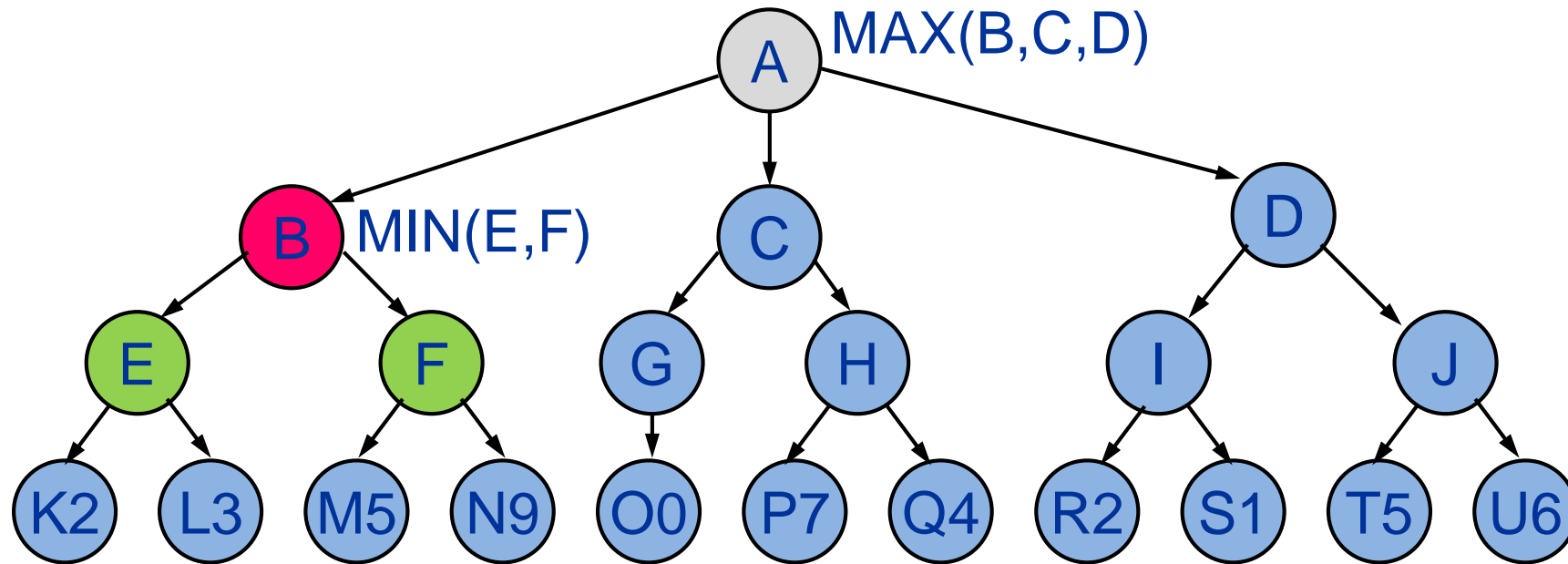
```
def minimax(stanje, dubina, moj_potez):  
    lp = nova_stanja(stanje)  
    fja = max_stanje if moj_potez else min_stanje  
  
    if dubina == 0 or lp is None:  
        return (stanje, proceni_stanje(stanje))  
  
    return fja([minimax(x, dubina - 1, not moj_potez) for x in lp])
```



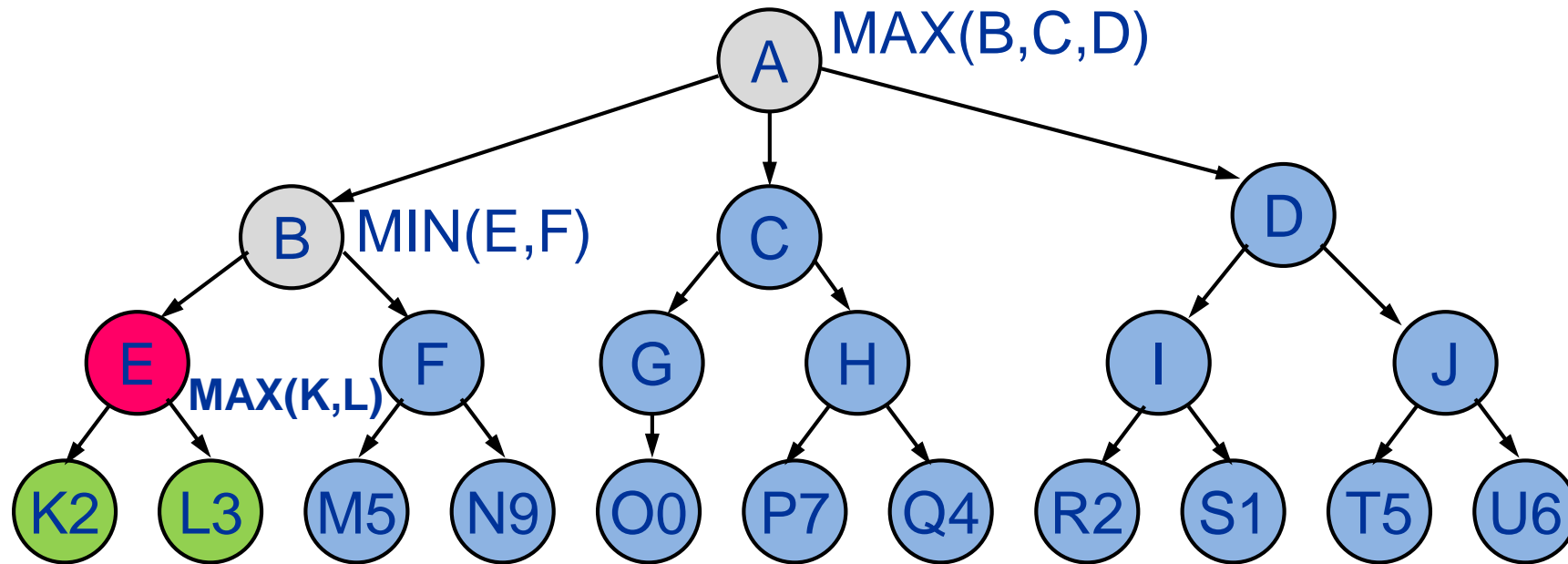
Min-Max algoritam (ilustracija)



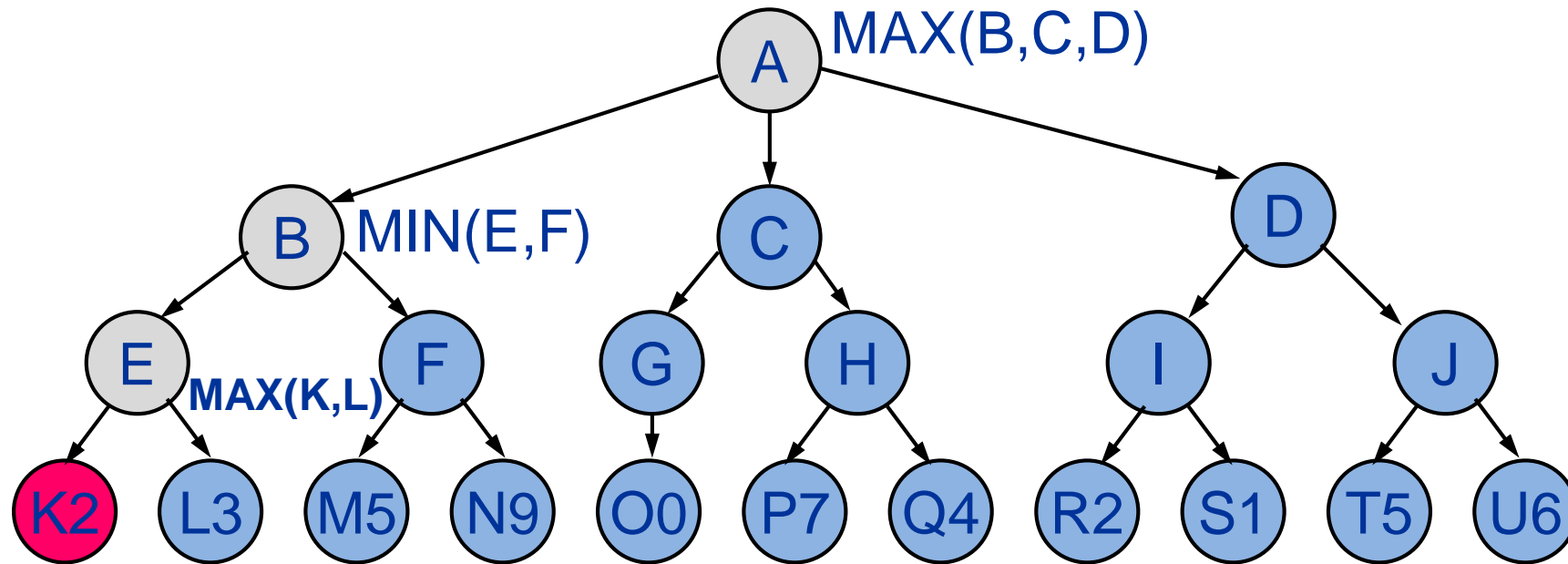
Min-Max algoritam (ilustracija)



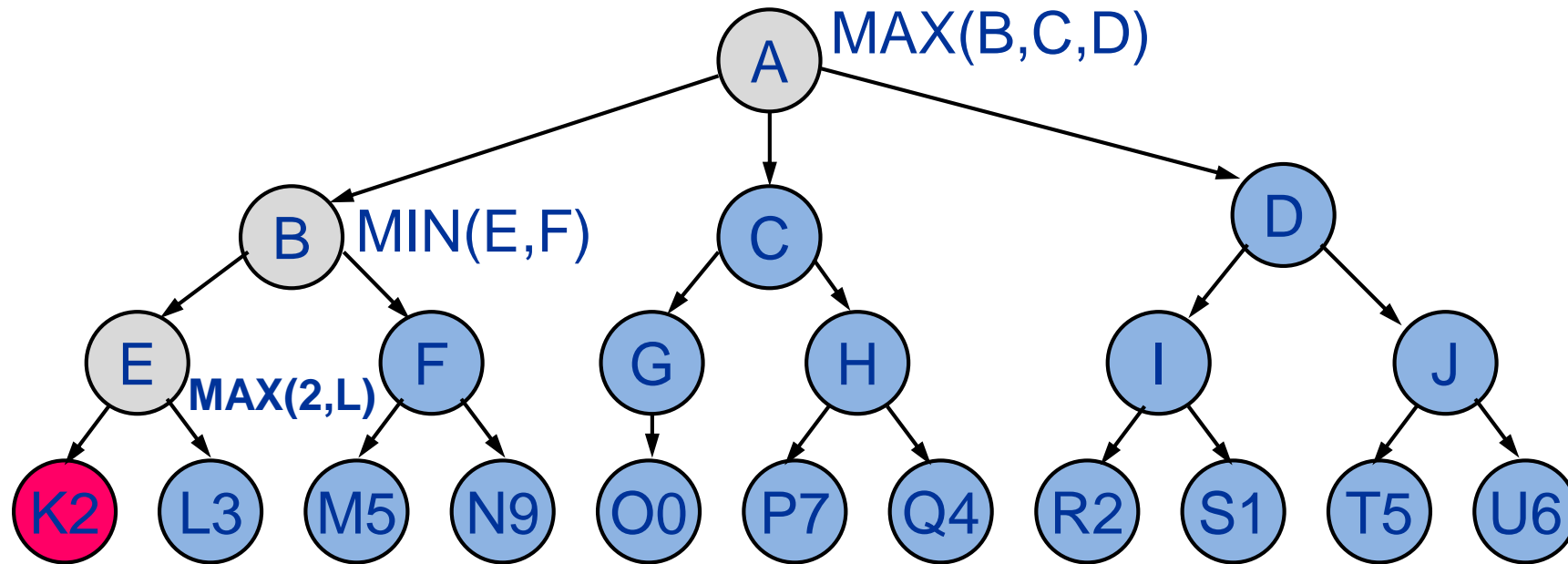
Min-Max algoritam (ilustracija)



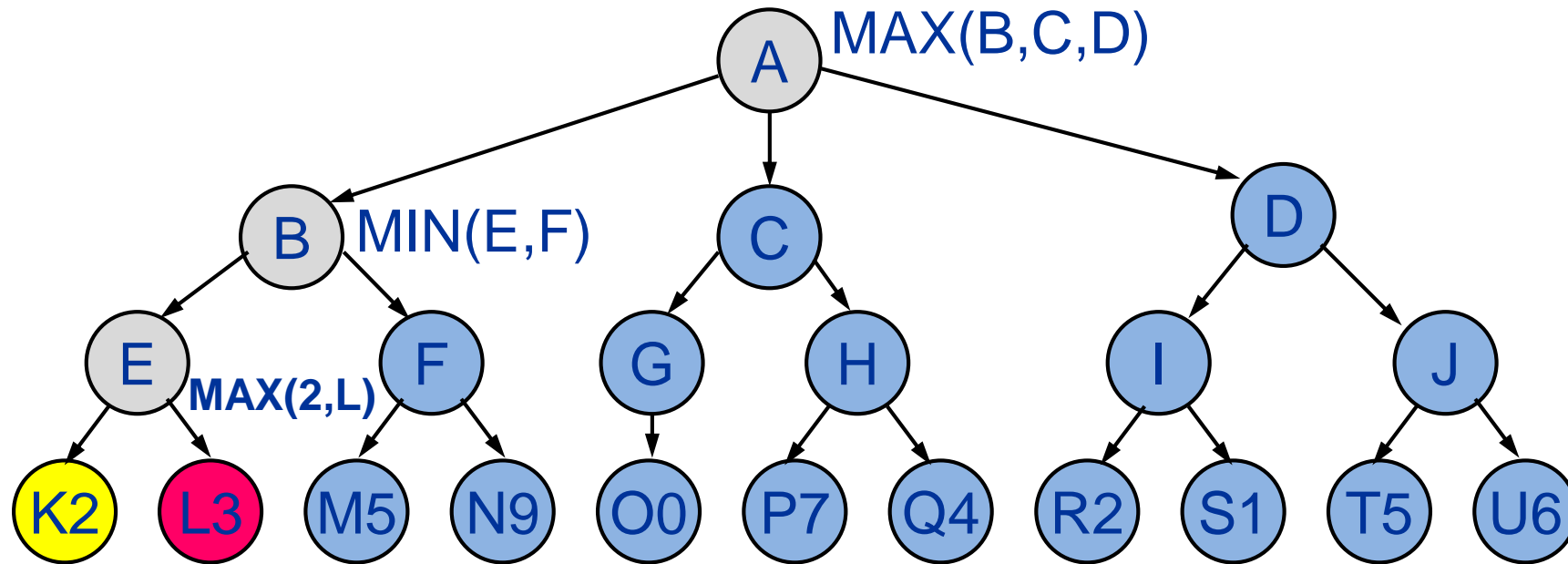
Min-Max algoritam (ilustracija)



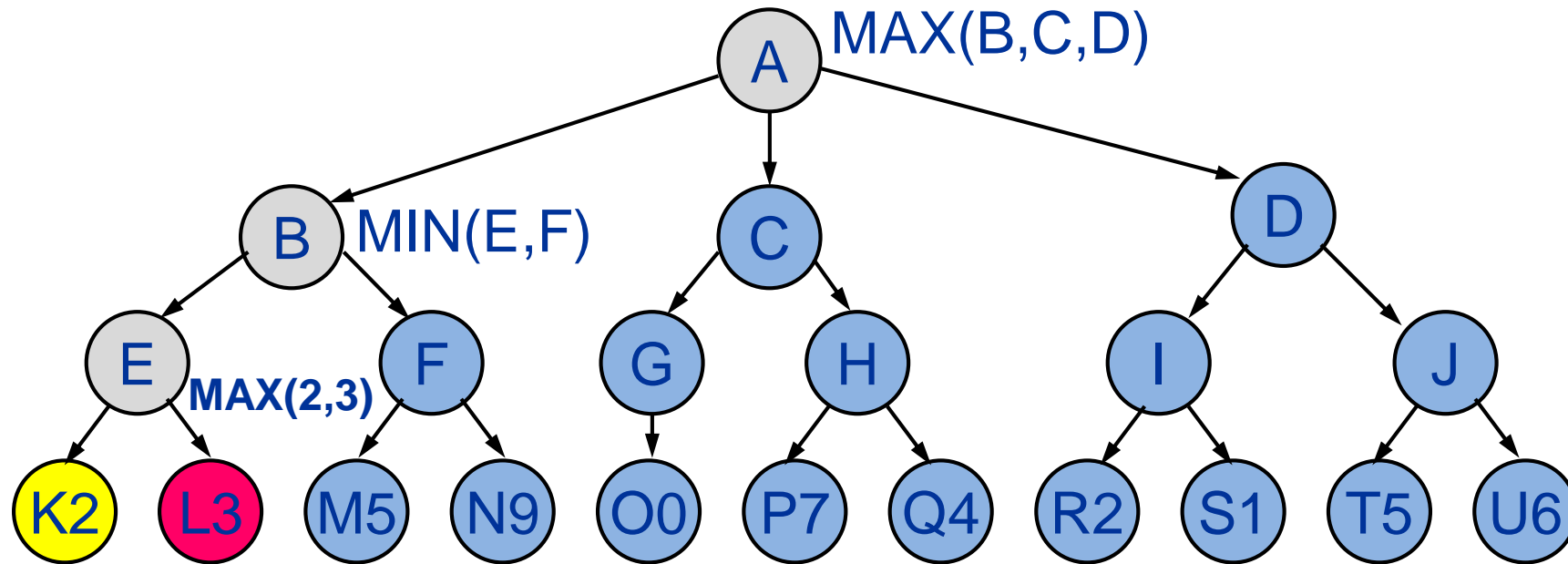
Min-Max algoritam (ilustracija)



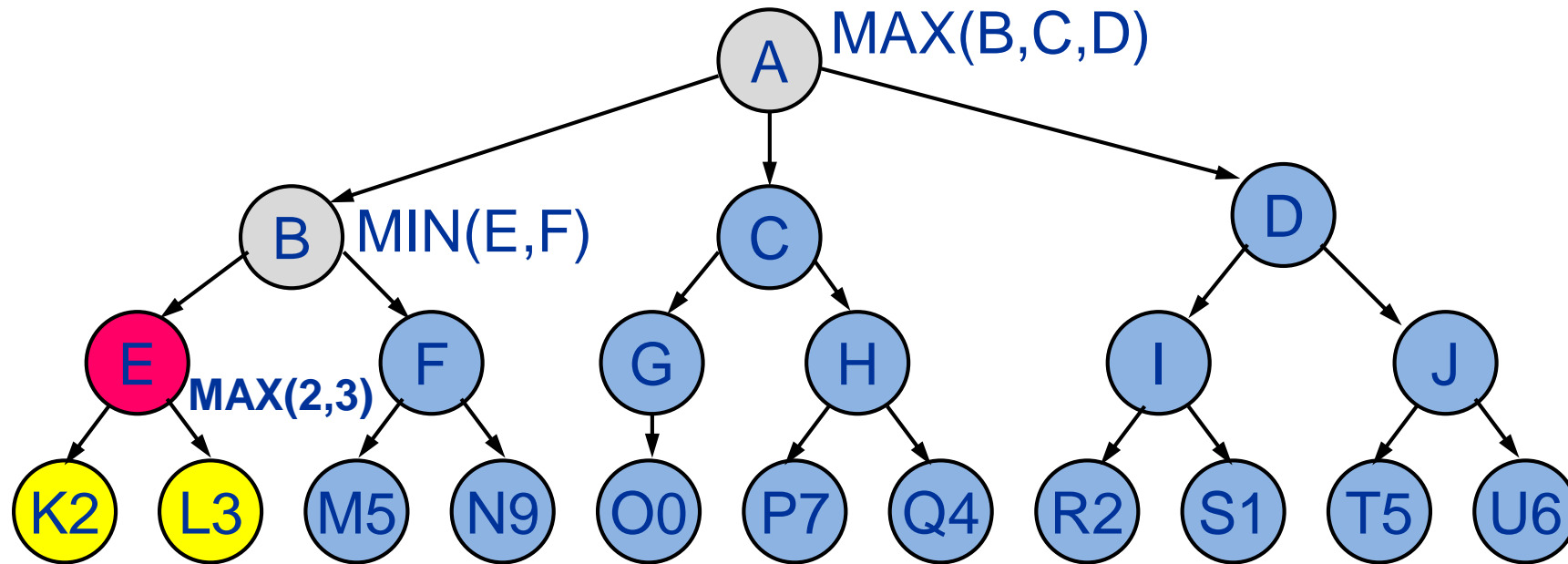
Min-Max algoritam (ilustracija)



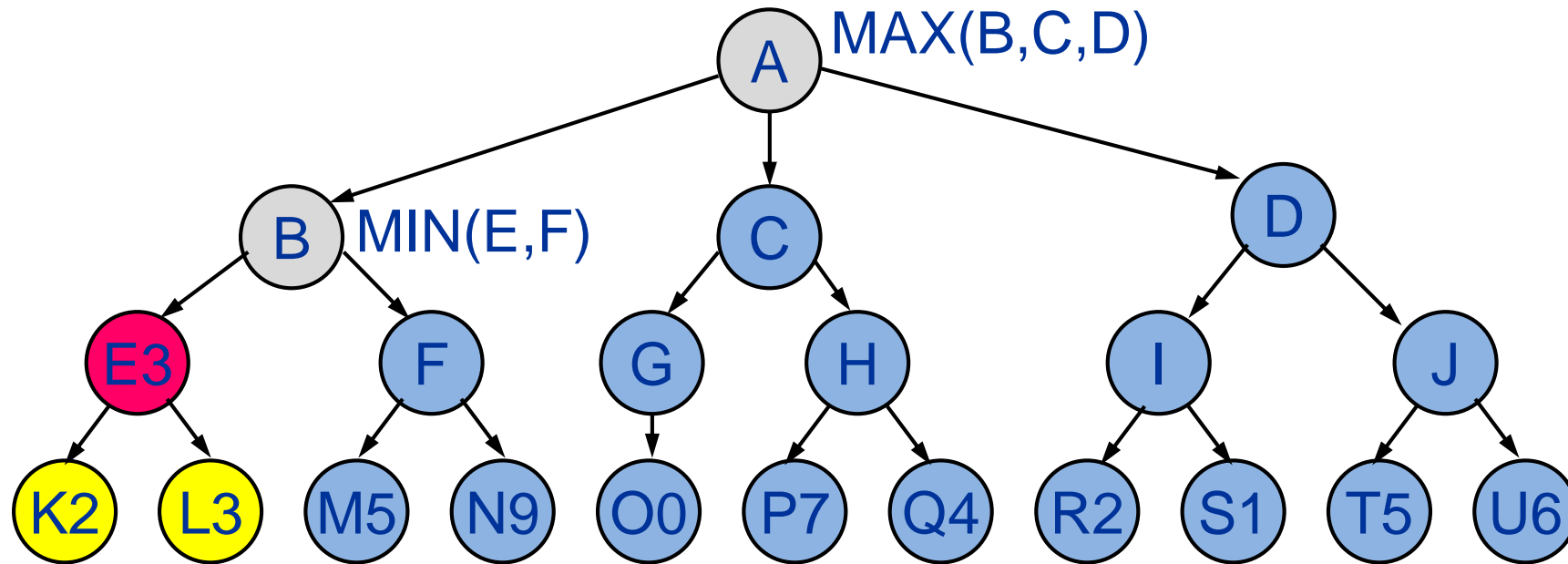
Min-Max algoritam (ilustracija)



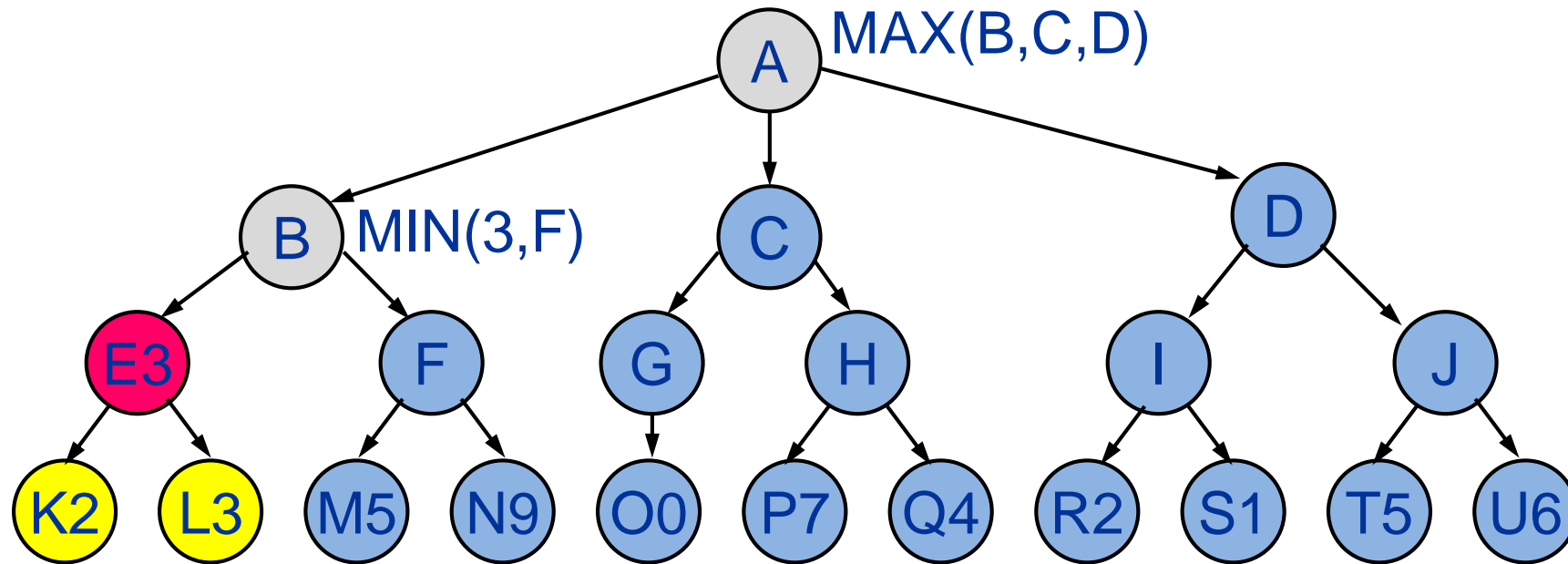
Min-Max algoritam (ilustracija)



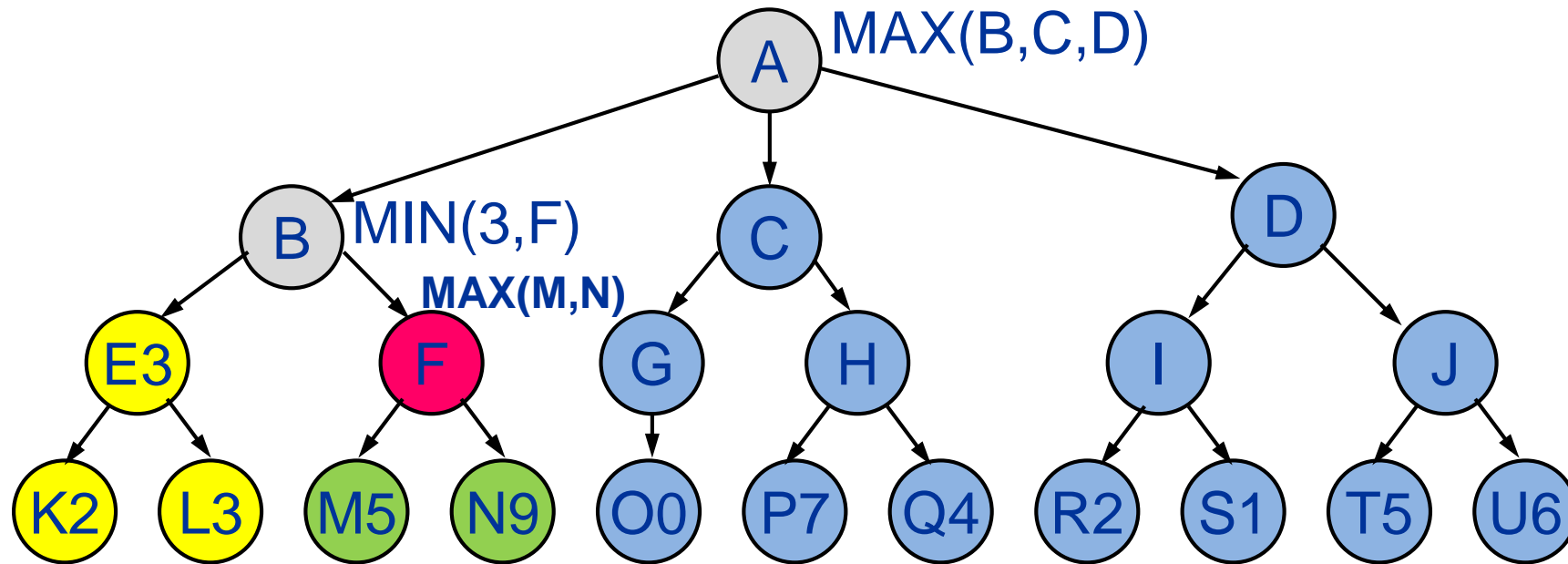
Min-Max algoritam (ilustracija)



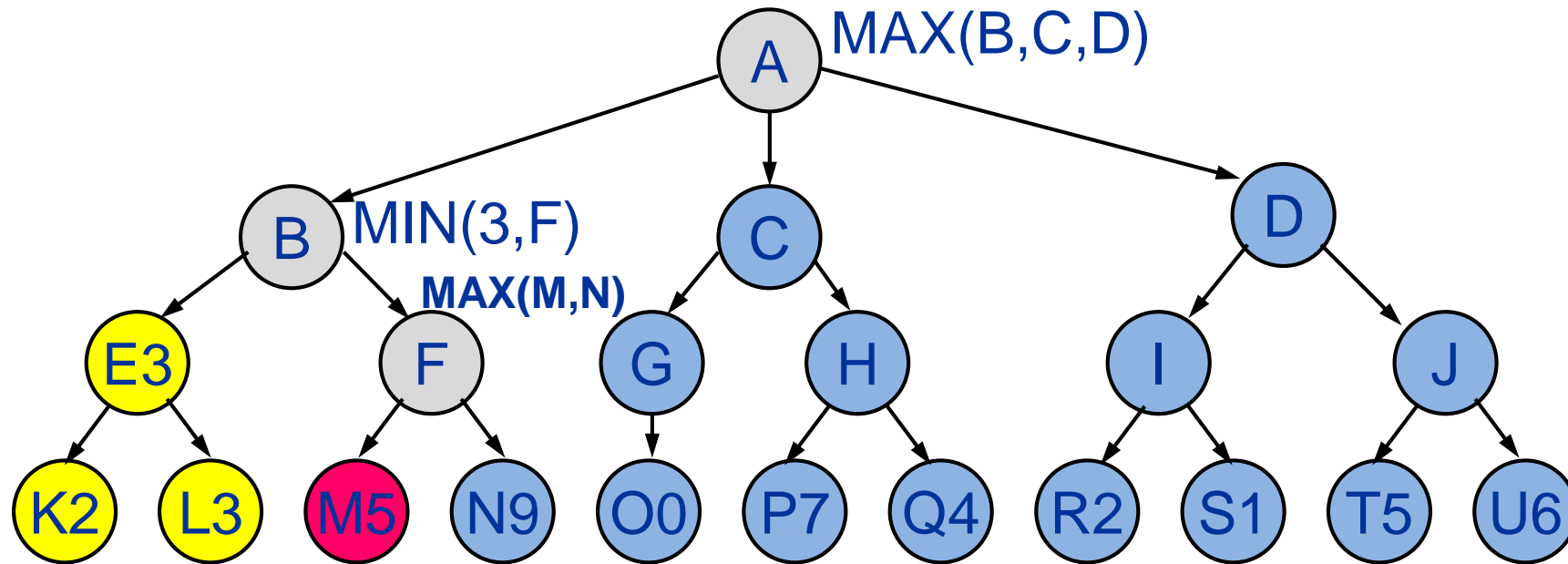
Min-Max algoritam (ilustracija)



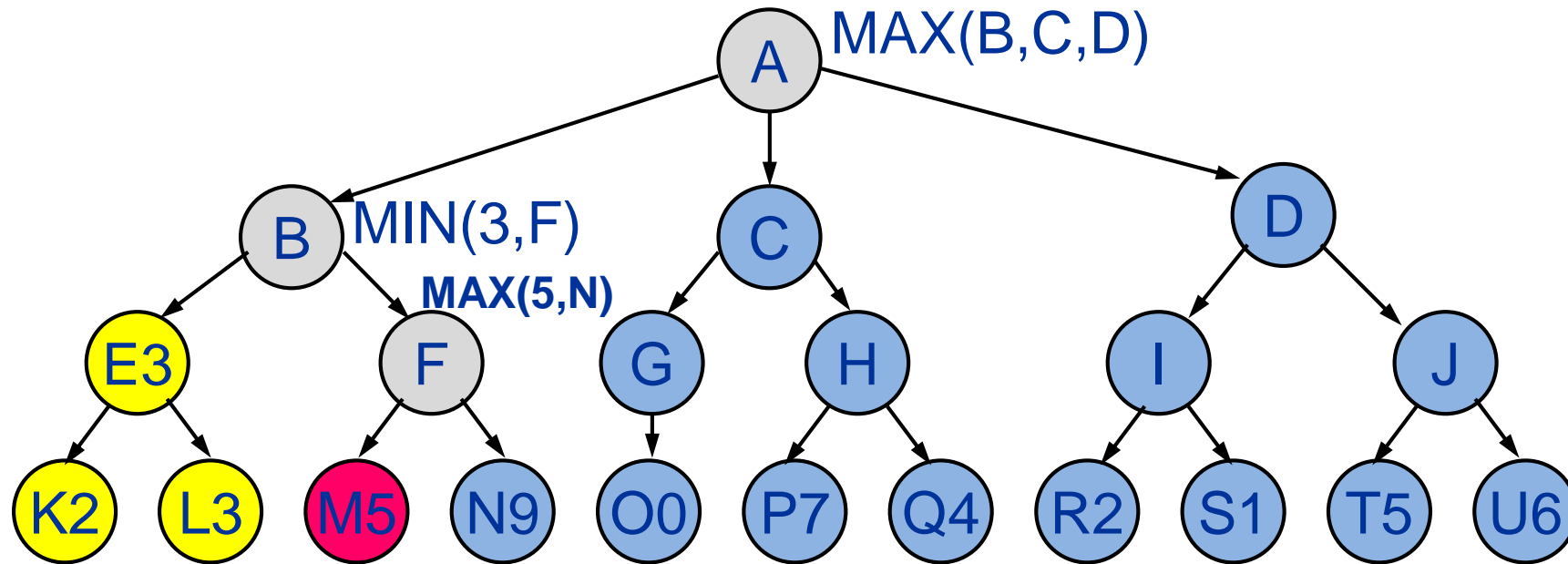
Min-Max algoritam (ilustracija)



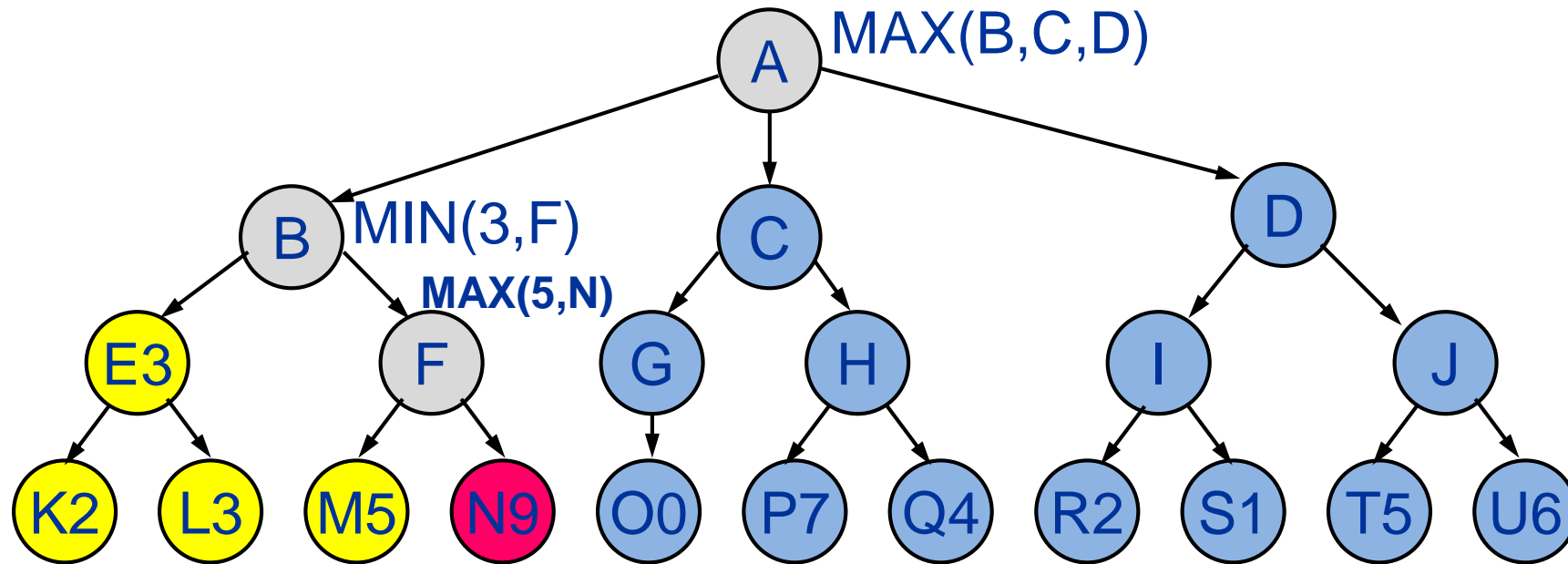
Min-Max algoritam (ilustracija)



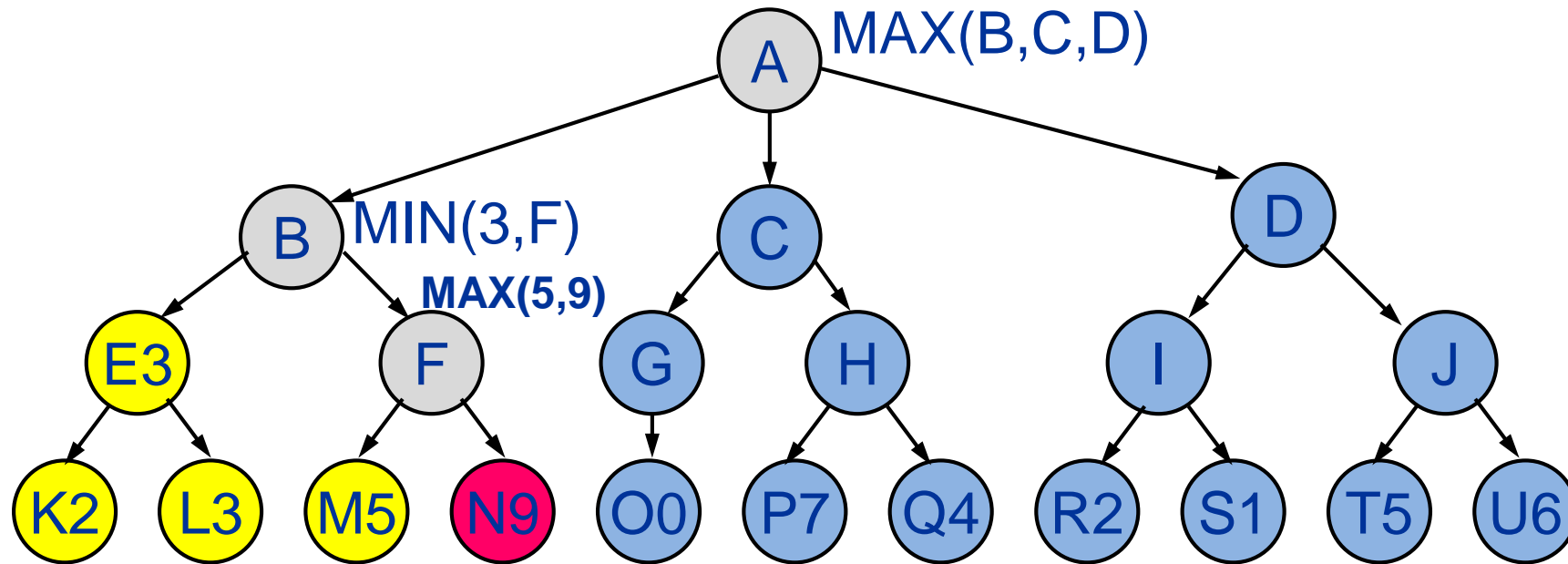
Min-Max algoritam (ilustracija)



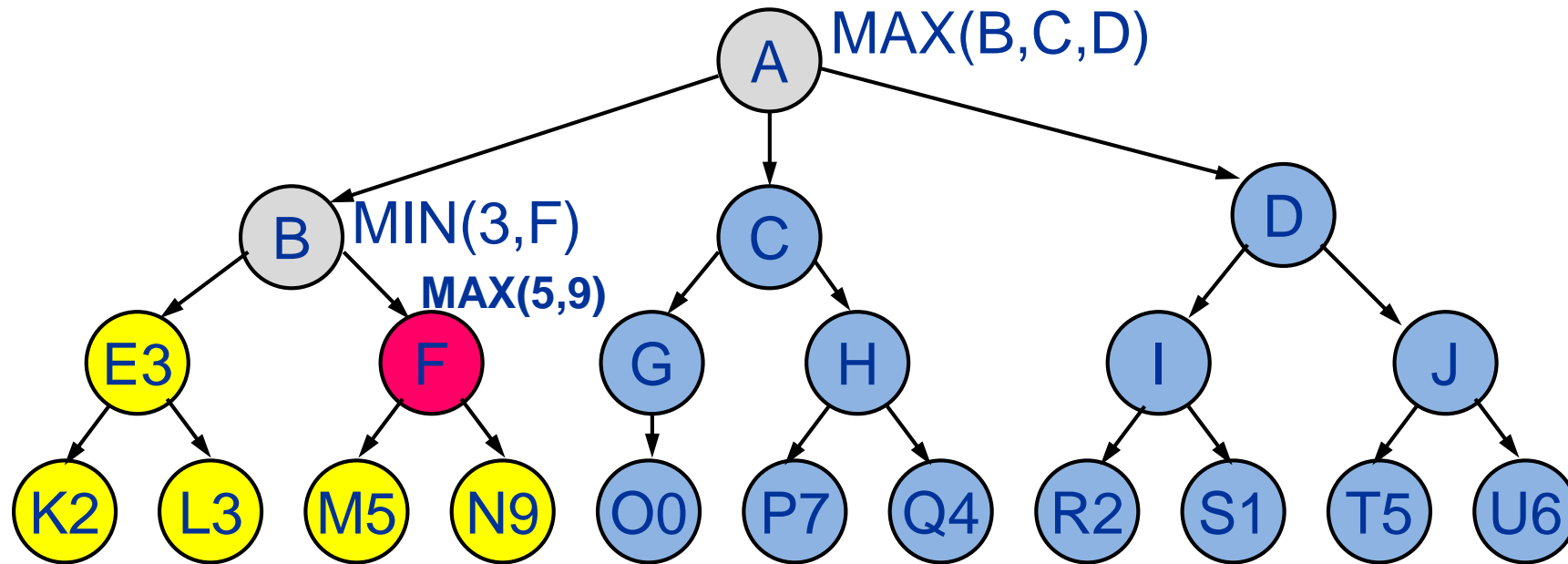
Min-Max algoritam (ilustracija)



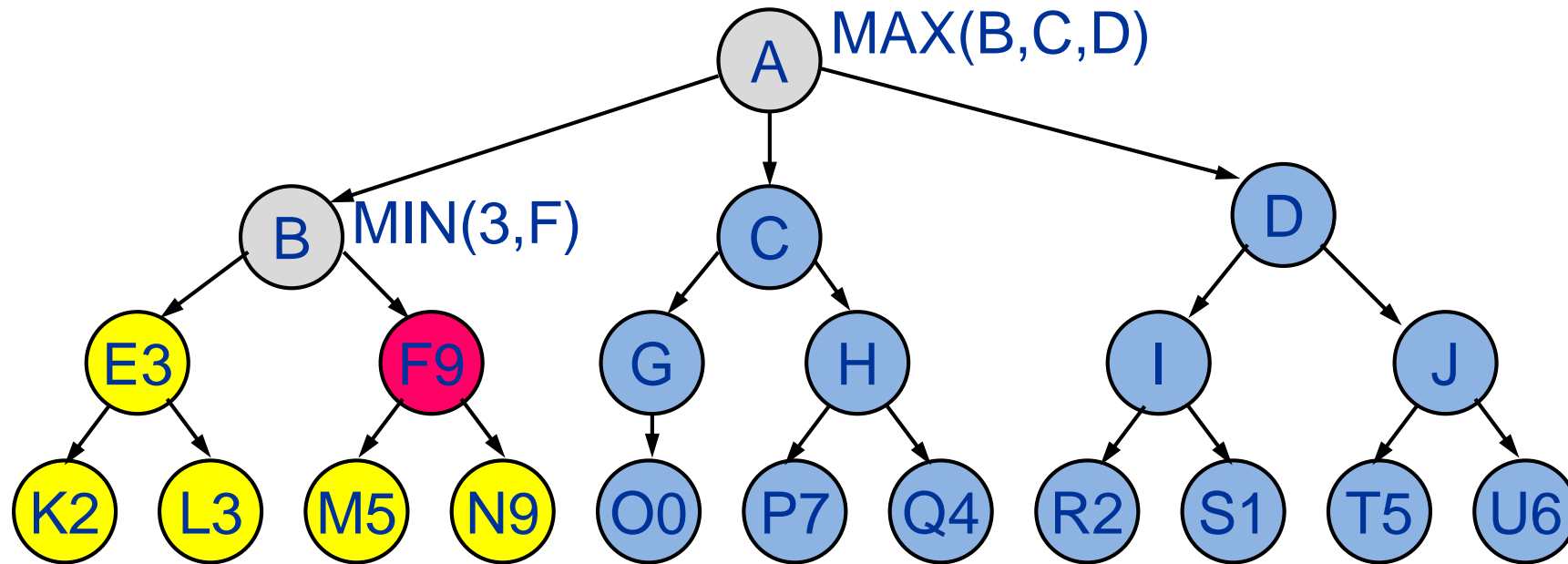
Min-Max algoritam (ilustracija)



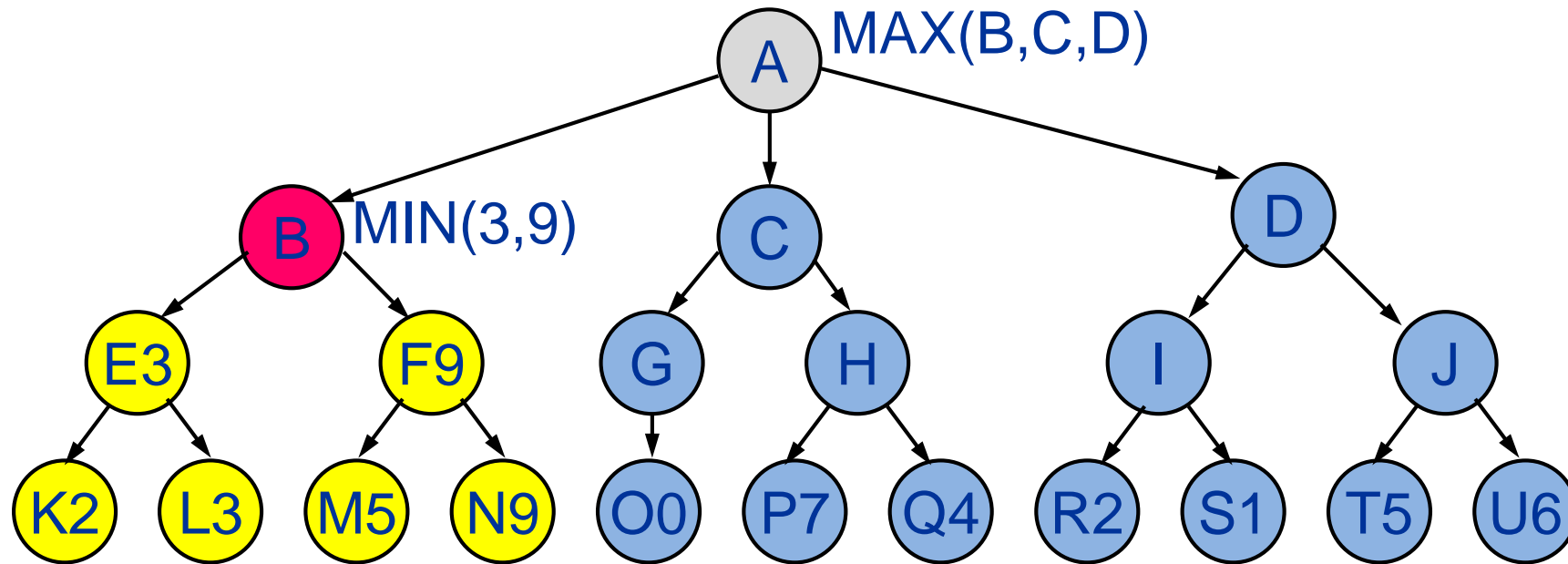
Min-Max algoritam (ilustracija)



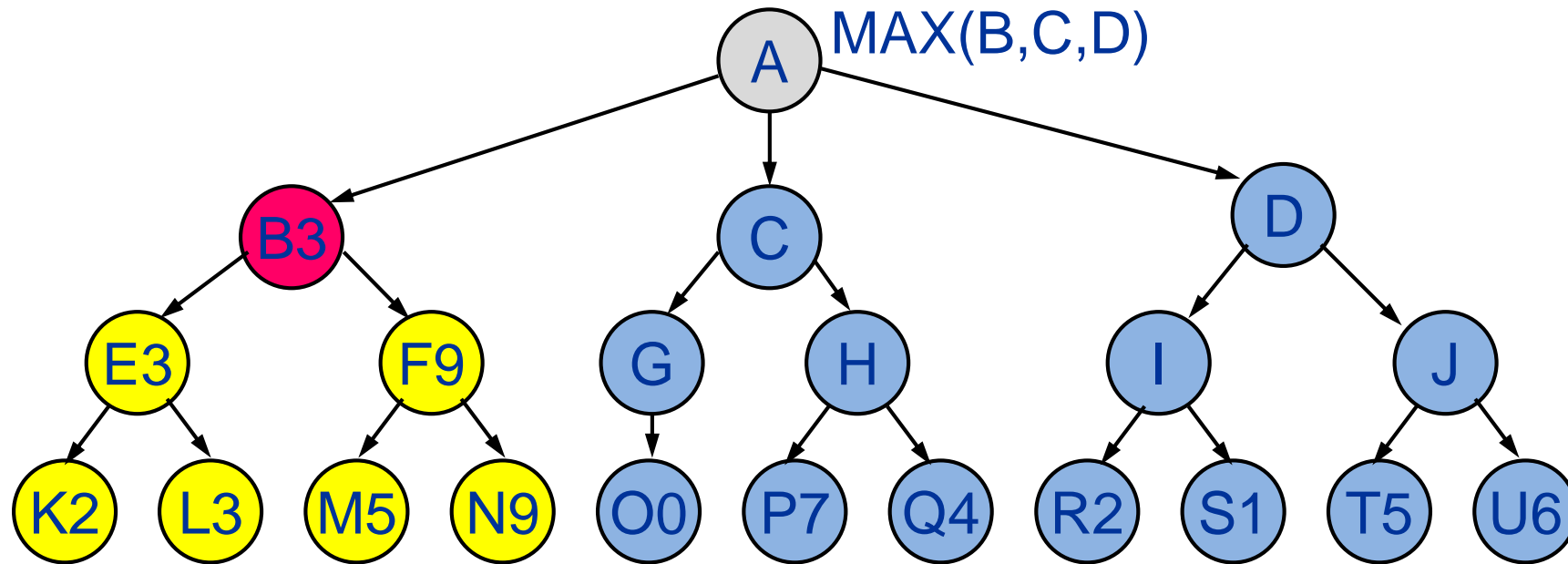
Min-Max algoritam (ilustracija)



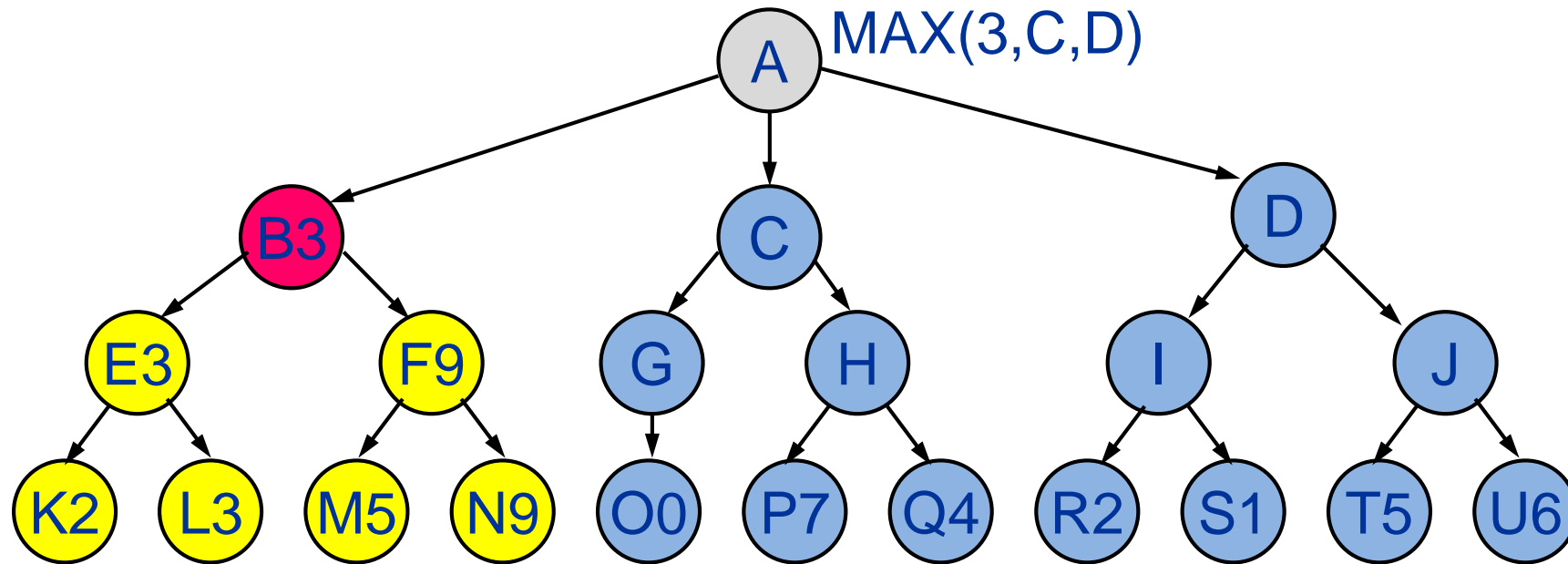
Min-Max algoritam (ilustracija)



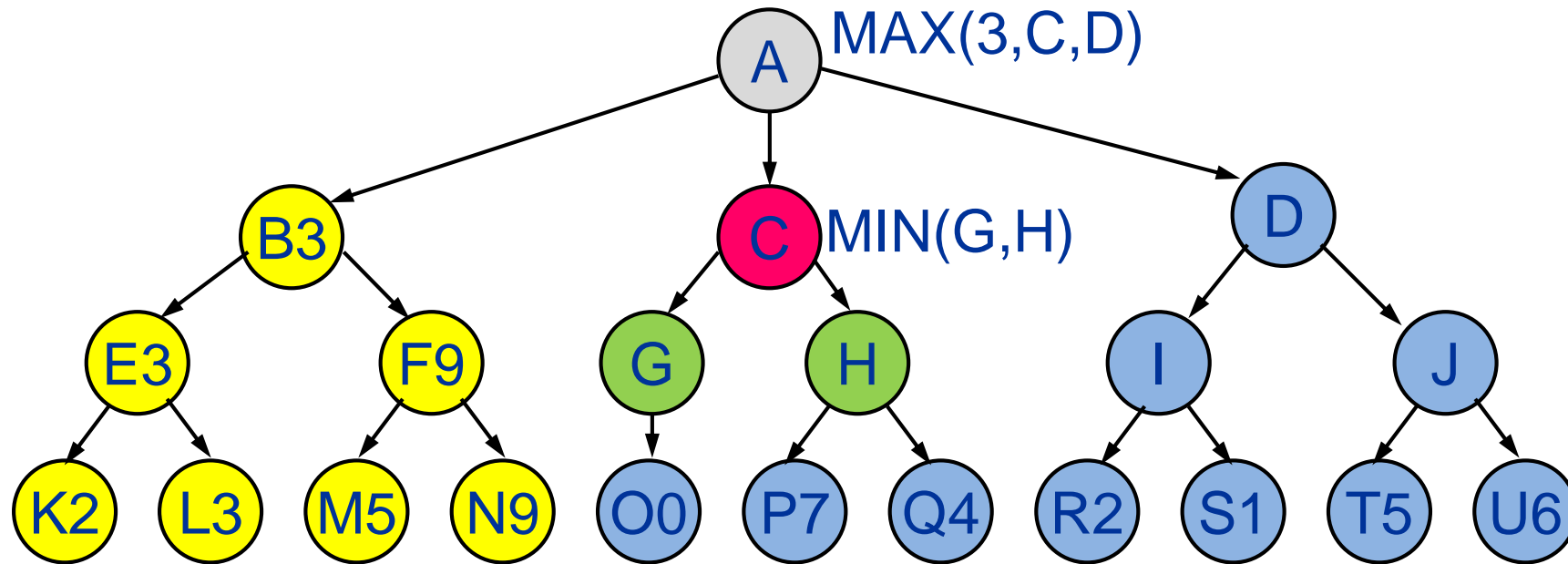
Min-Max algoritam (ilustracija)



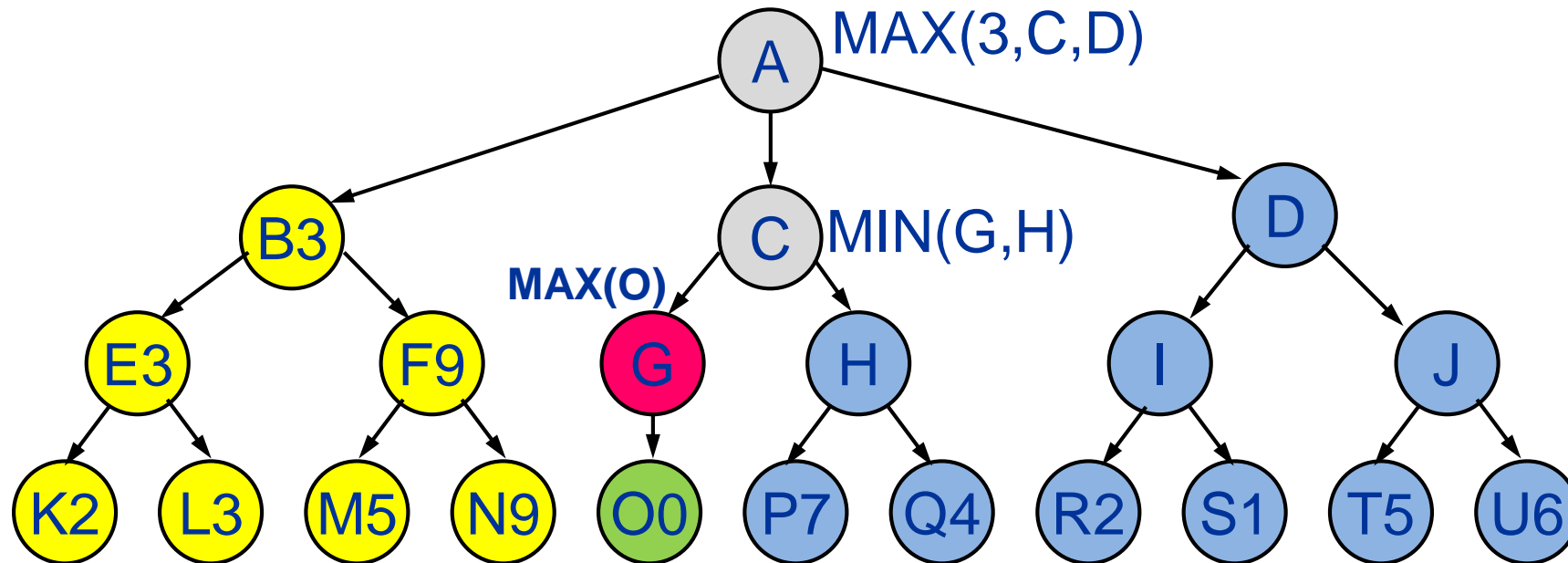
Min-Max algoritam (ilustracija)



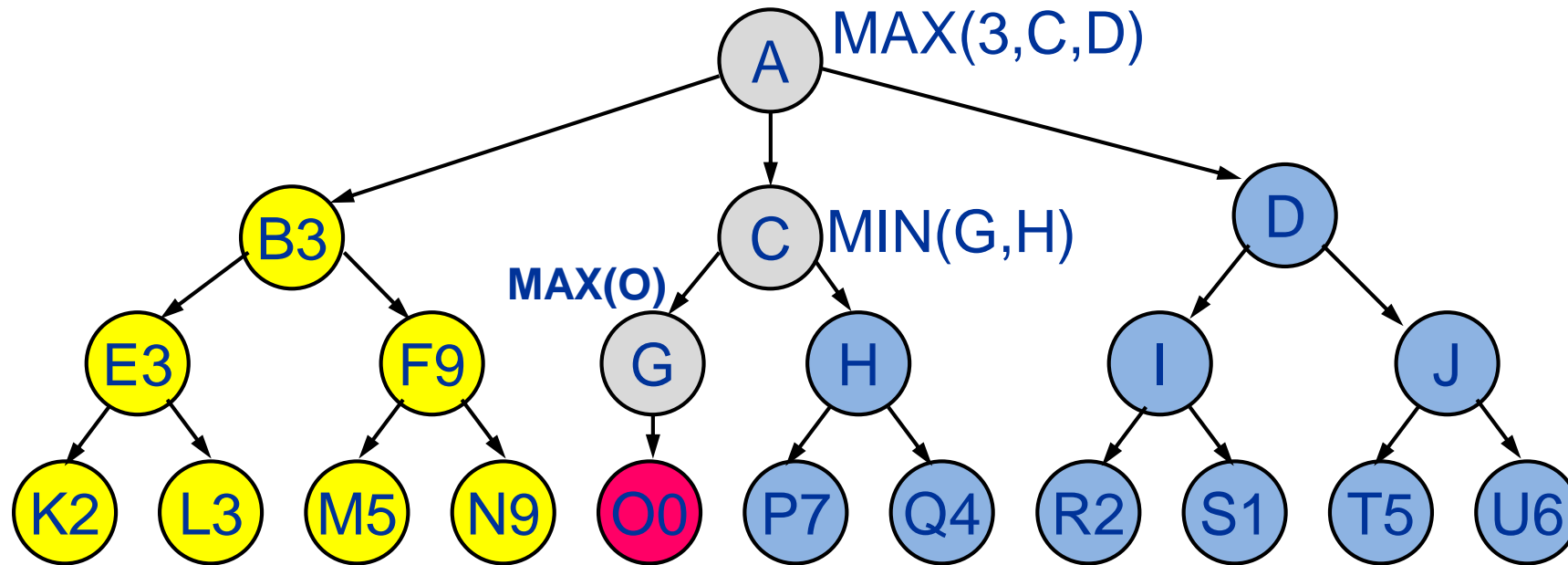
Min-Max algoritam (ilustracija)



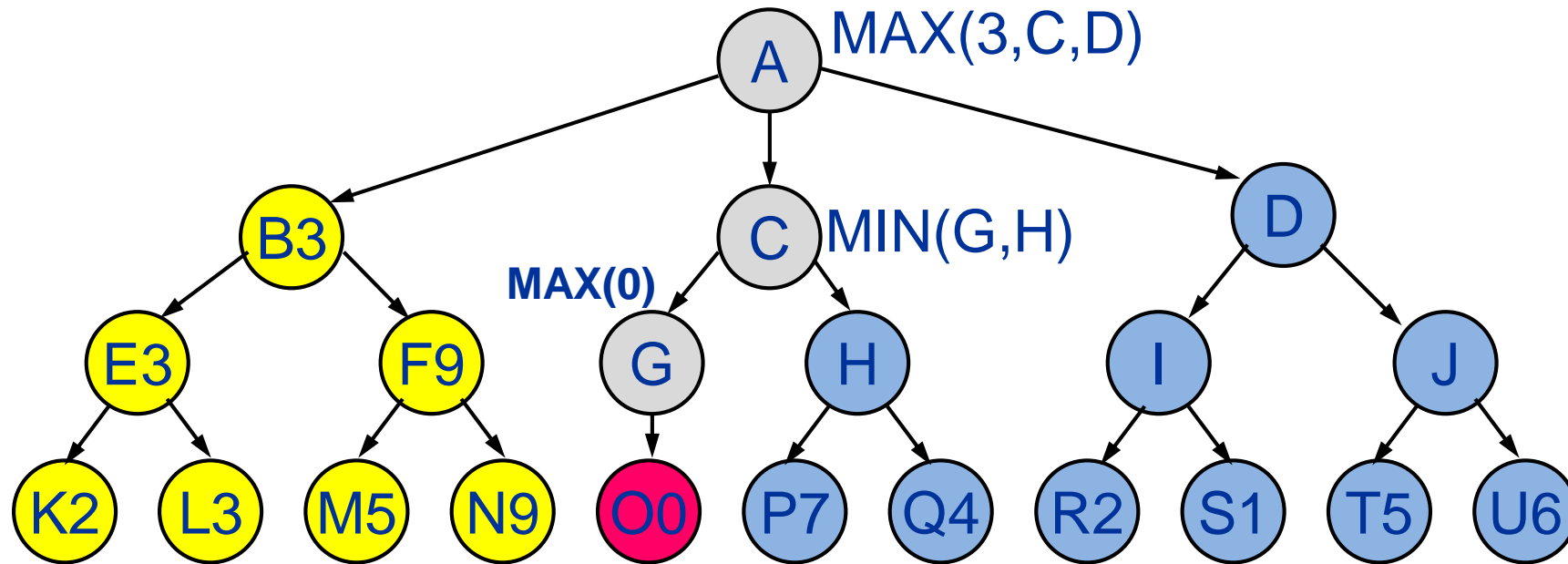
Min-Max algoritam (ilustracija)



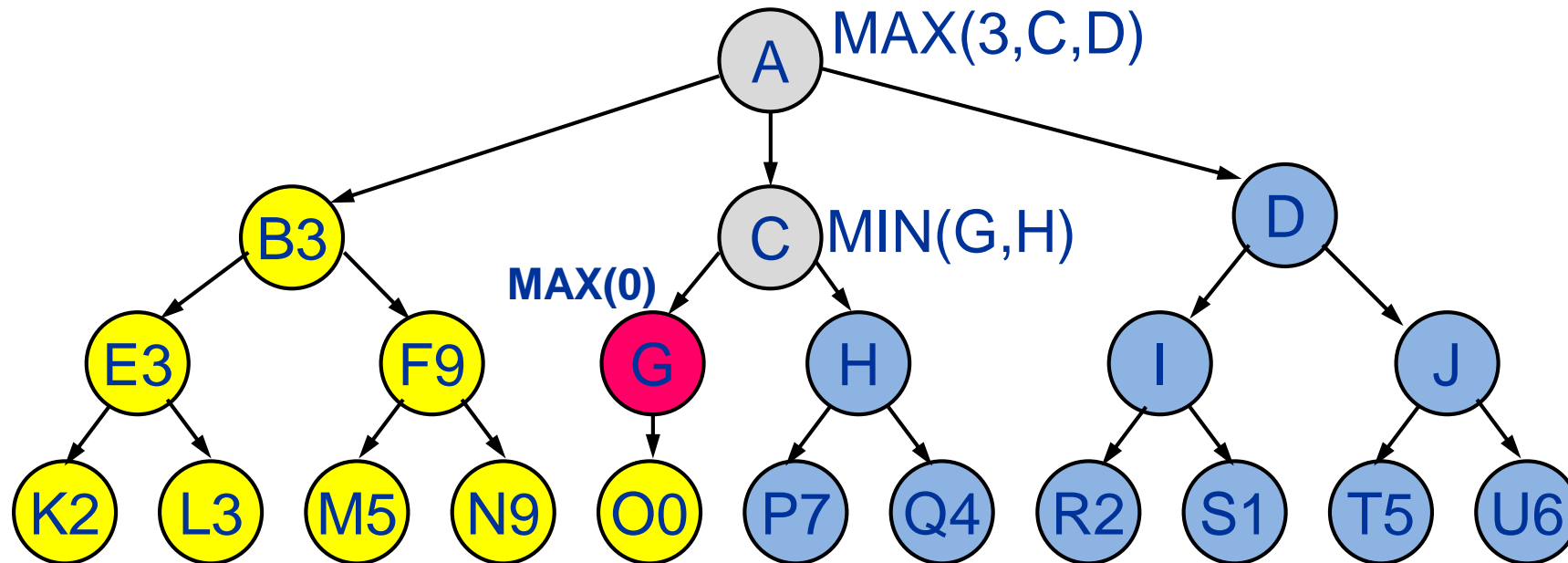
Min-Max algoritam (ilustracija)



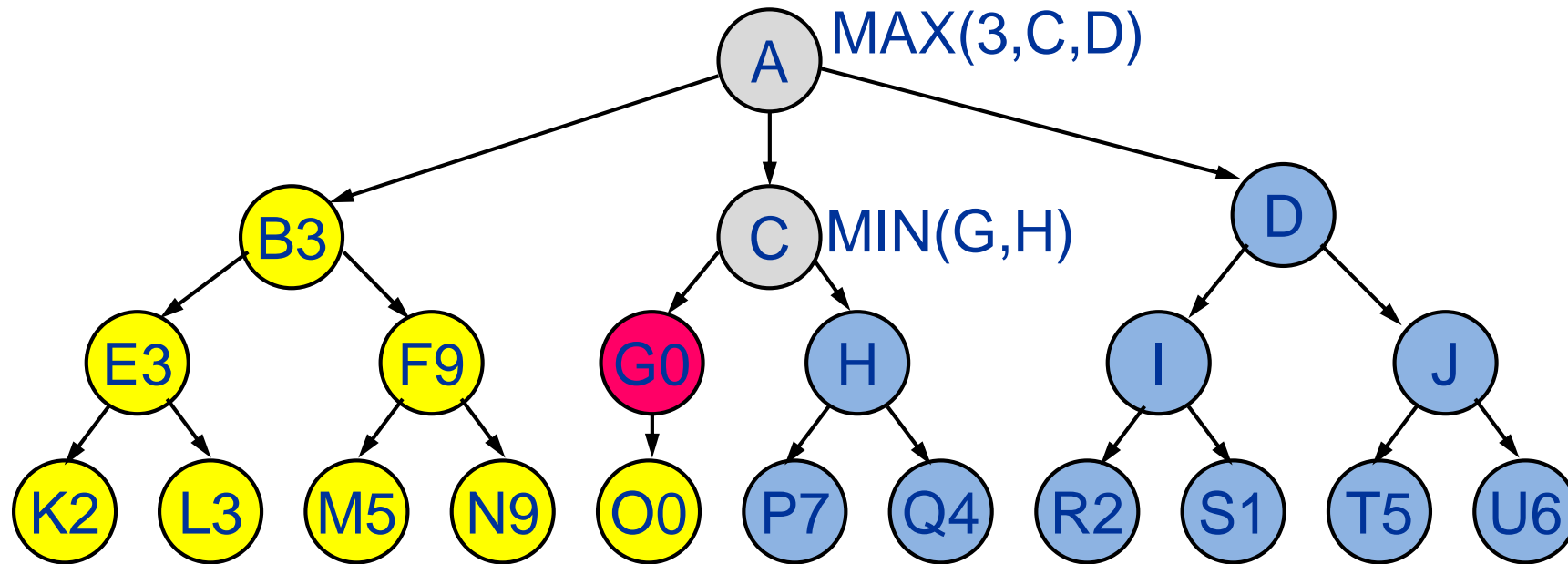
Min-Max algoritam (ilustracija)



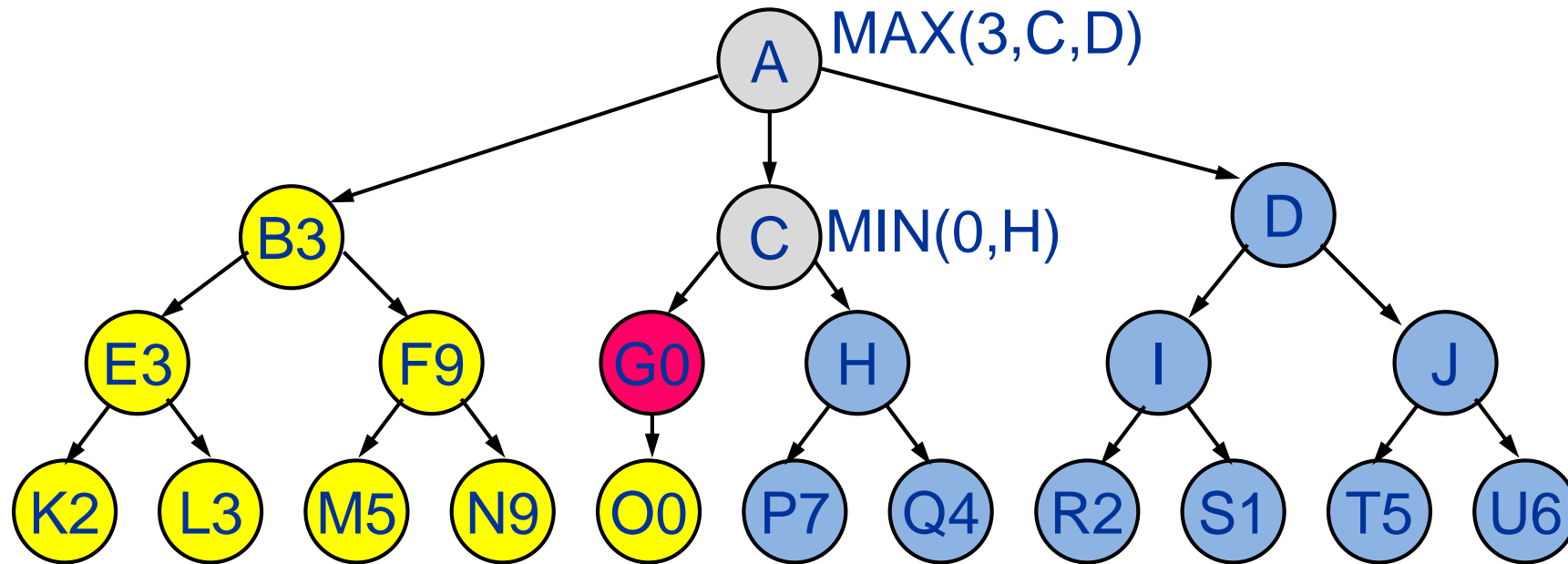
Min-Max algoritam (ilustracija)



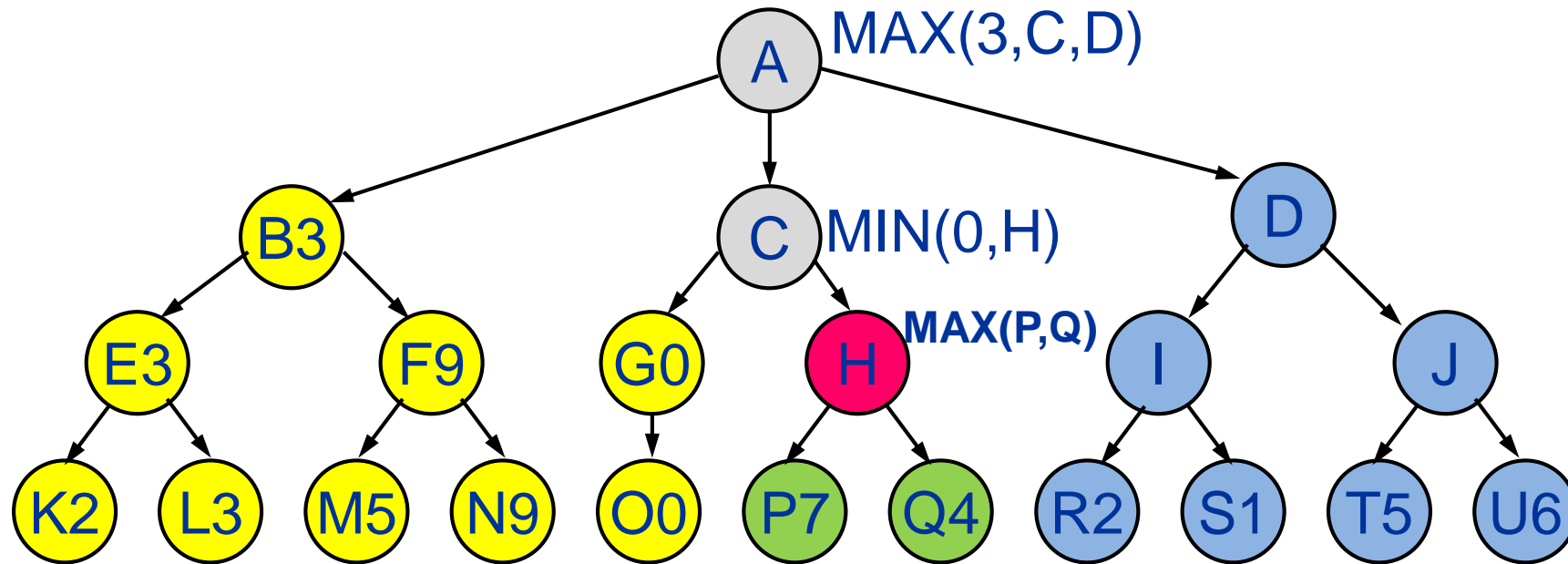
Min-Max algoritam (ilustracija)



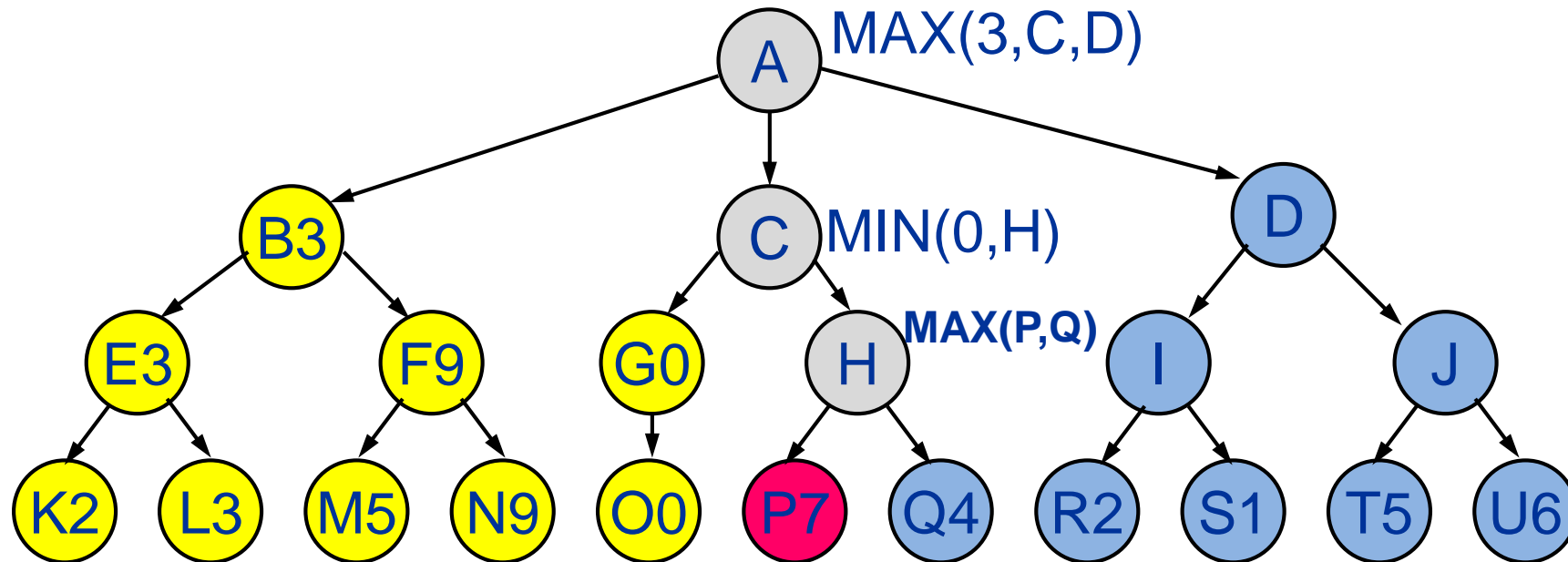
Min-Max algoritam (ilustracija)



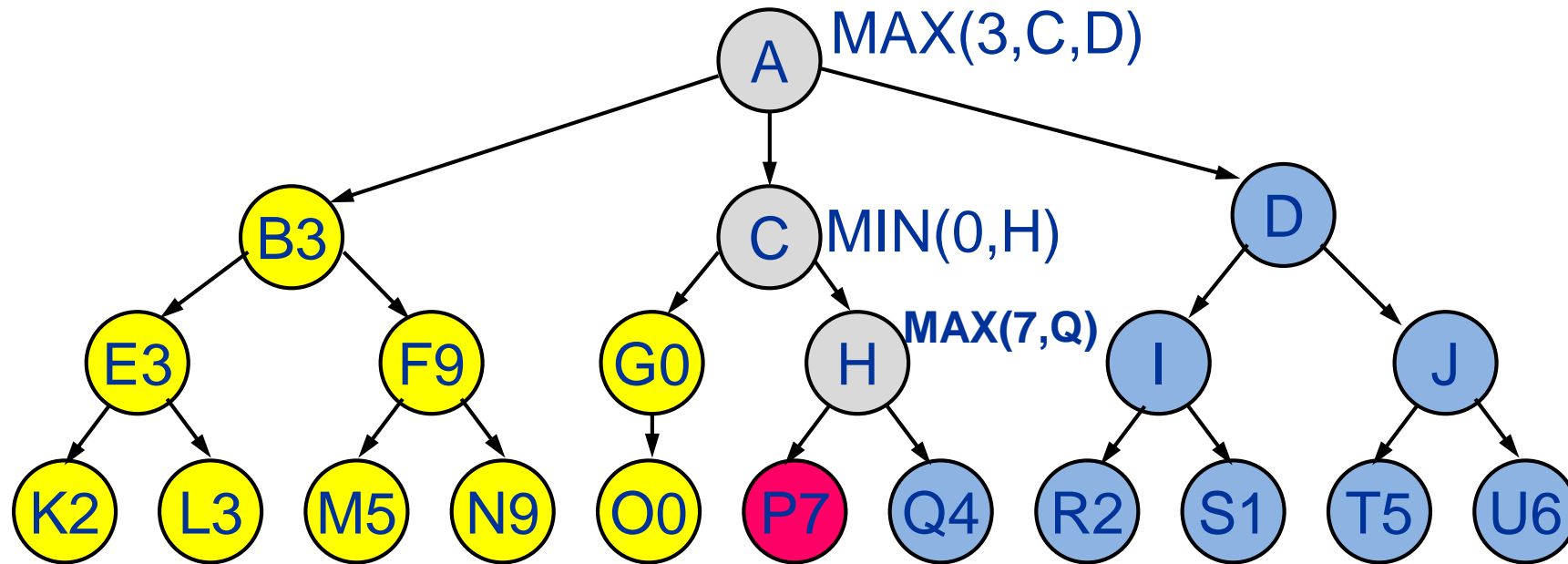
Min-Max algoritam (ilustracija)



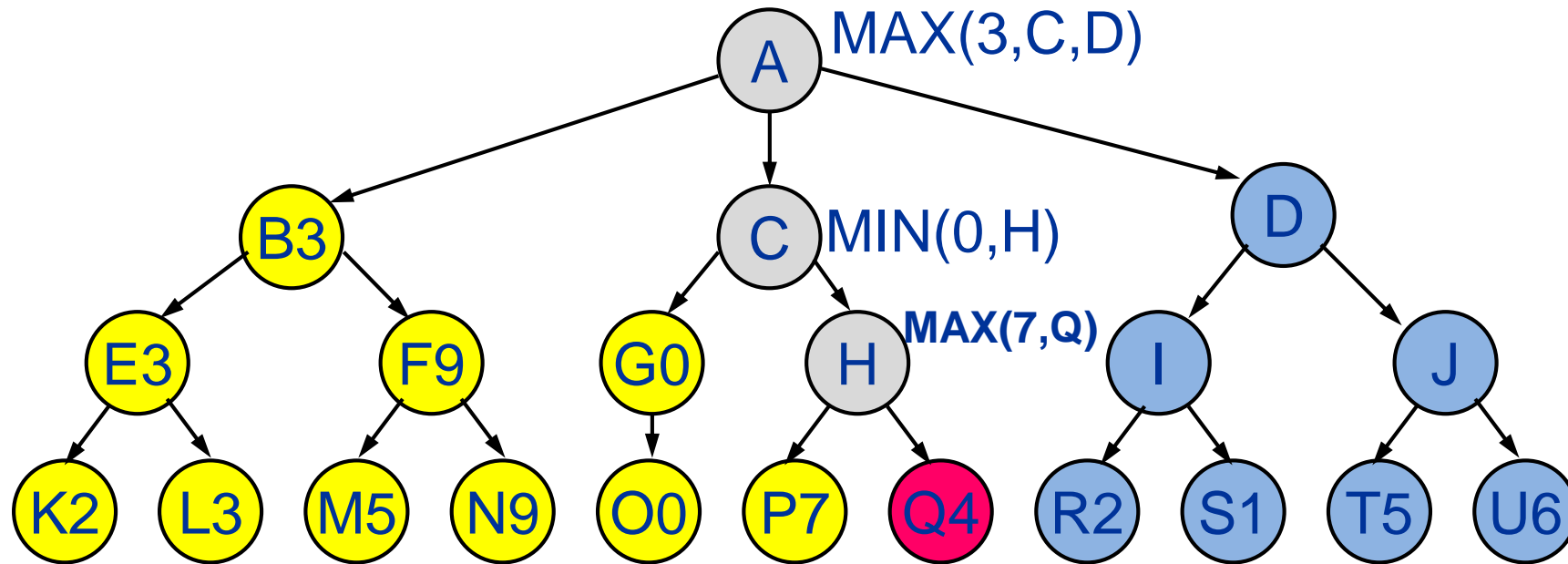
Min-Max algoritam (ilustracija)



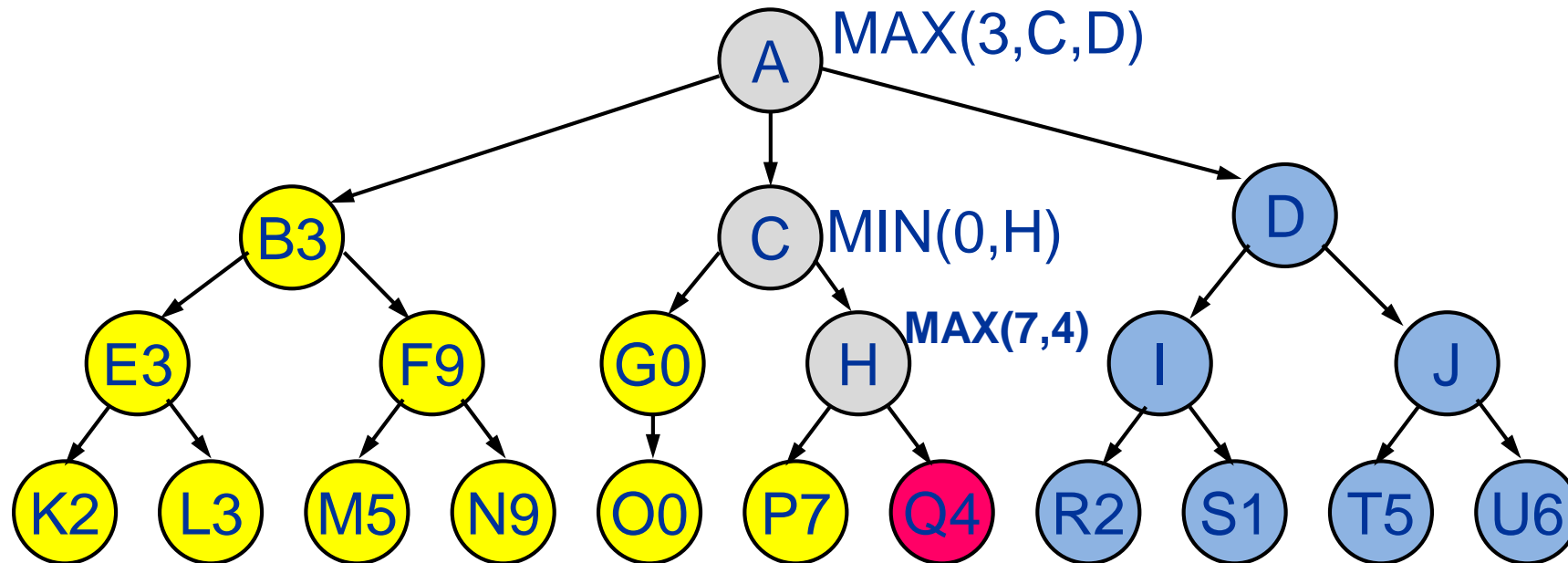
Min-Max algoritam (ilustracija)



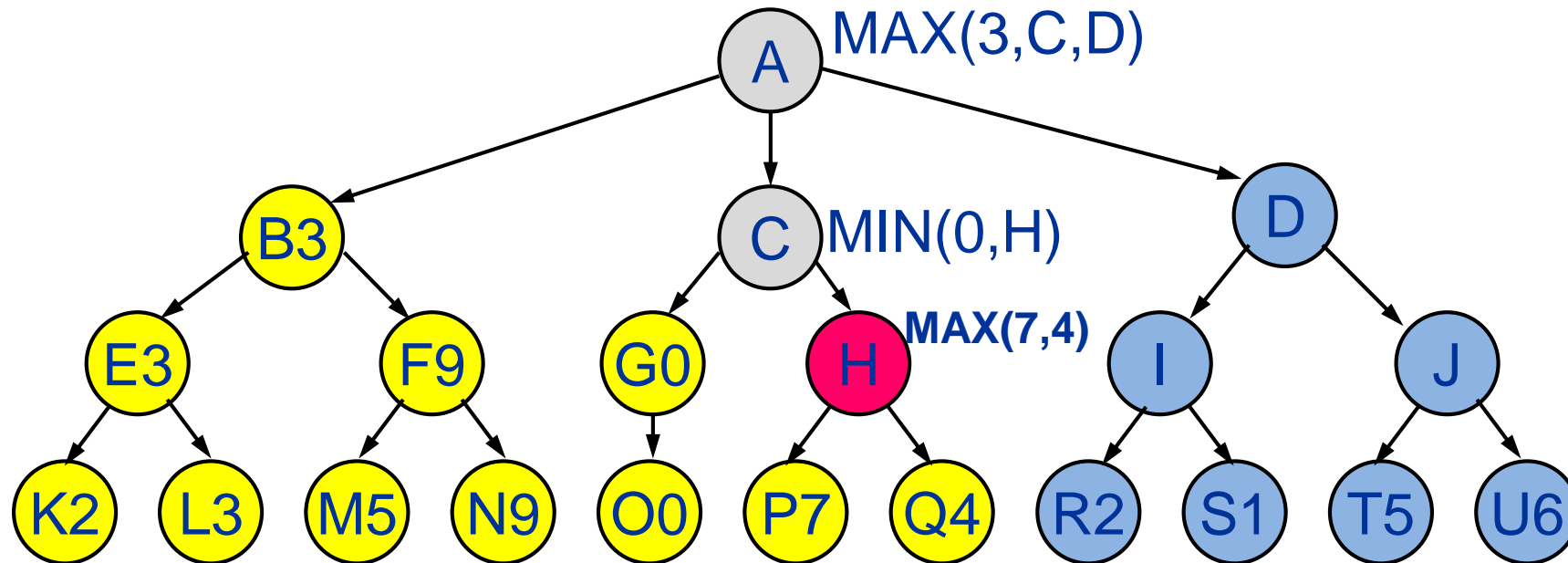
Min-Max algoritam (ilustracija)



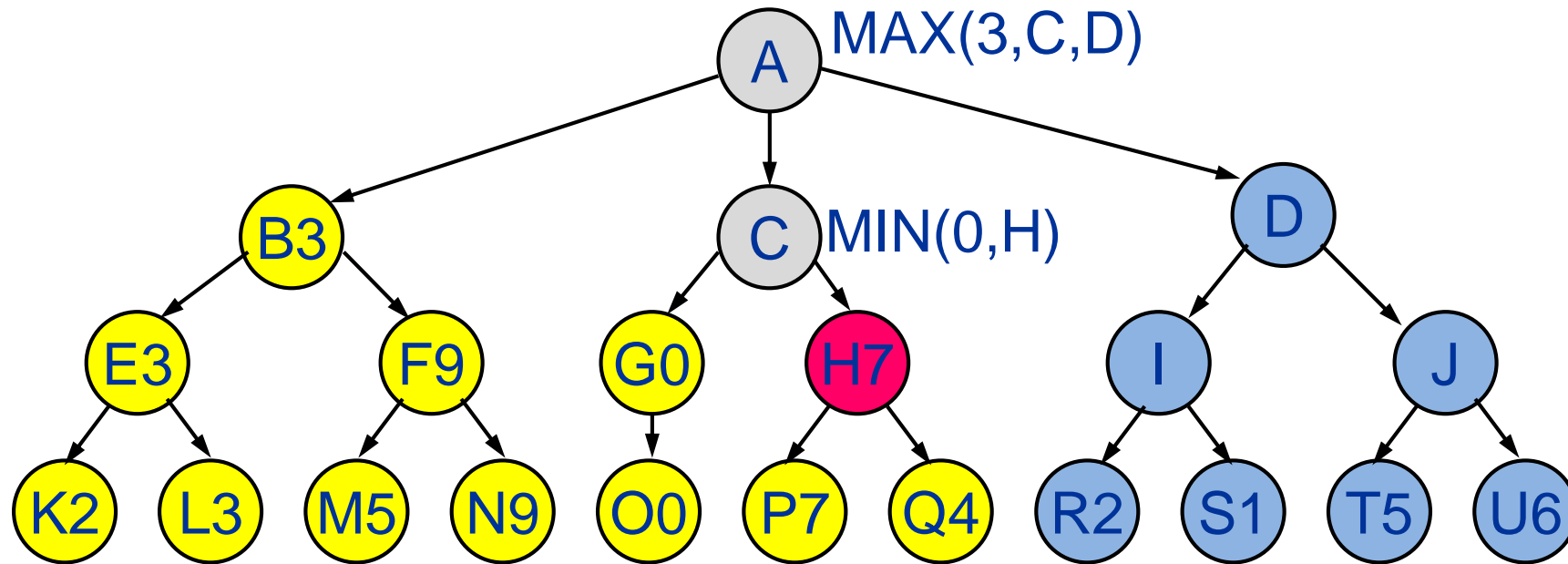
Min-Max algoritam (ilustracija)



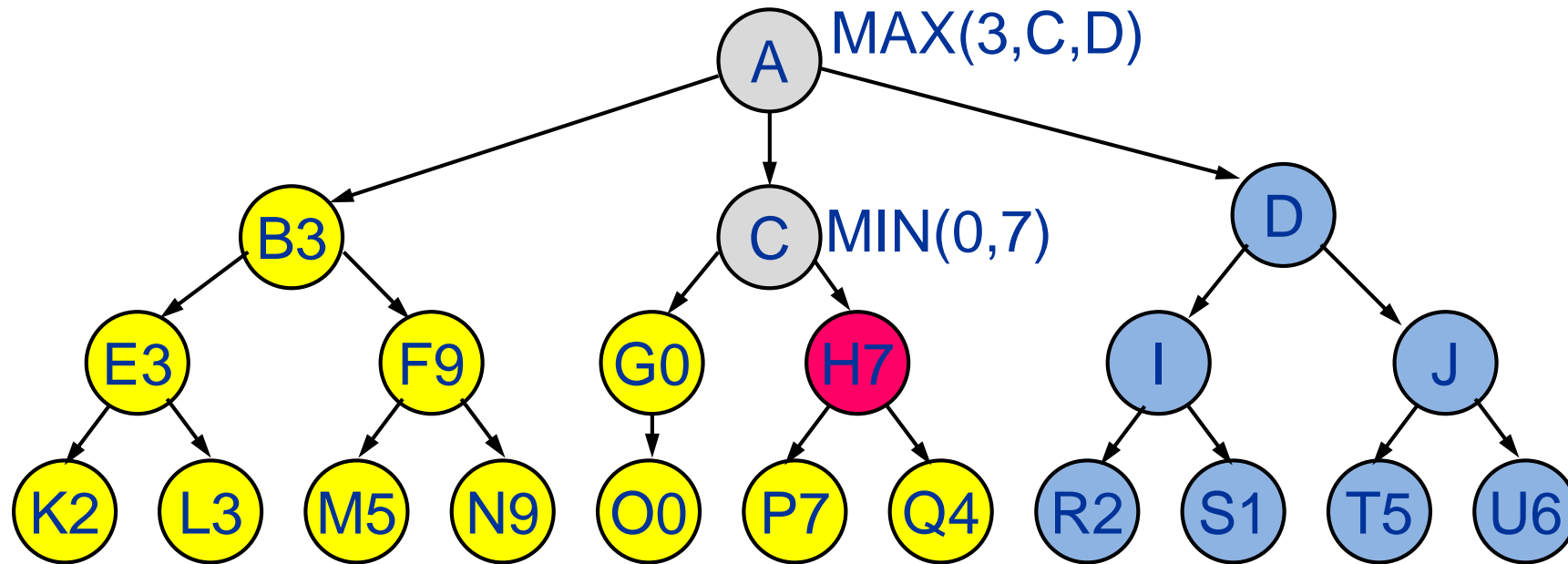
Min-Max algoritam (ilustracija)



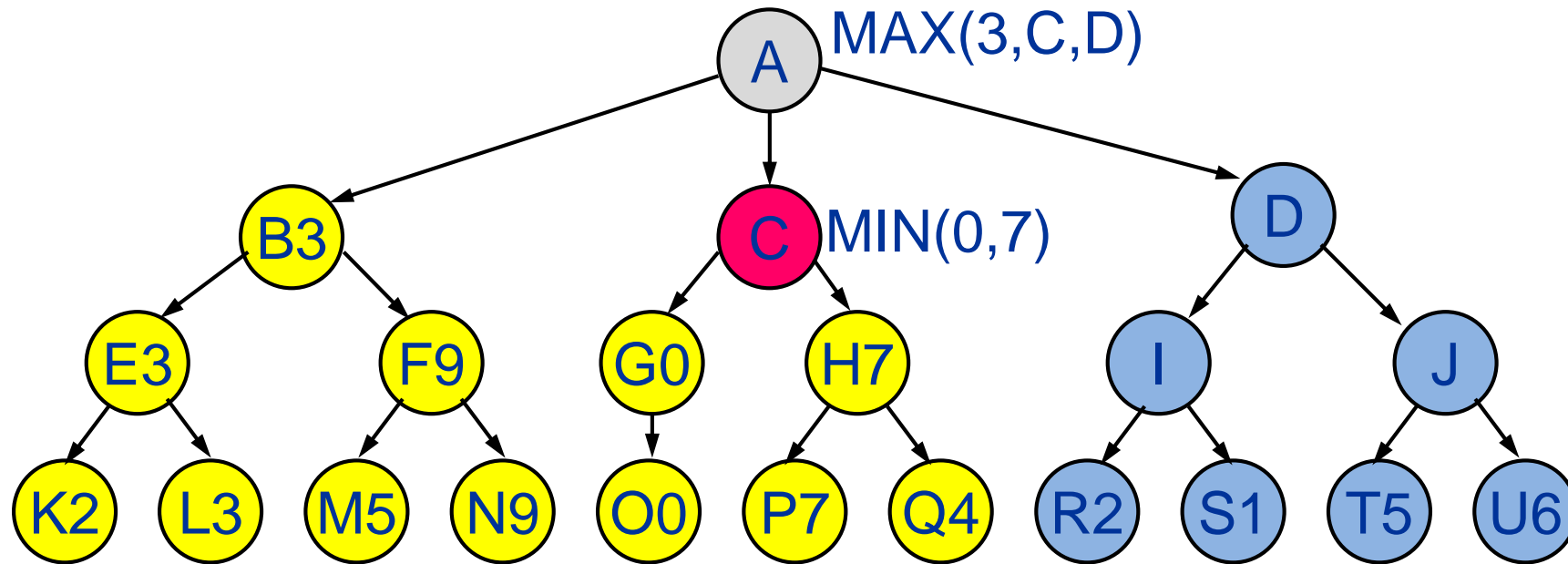
Min-Max algoritam (ilustracija)



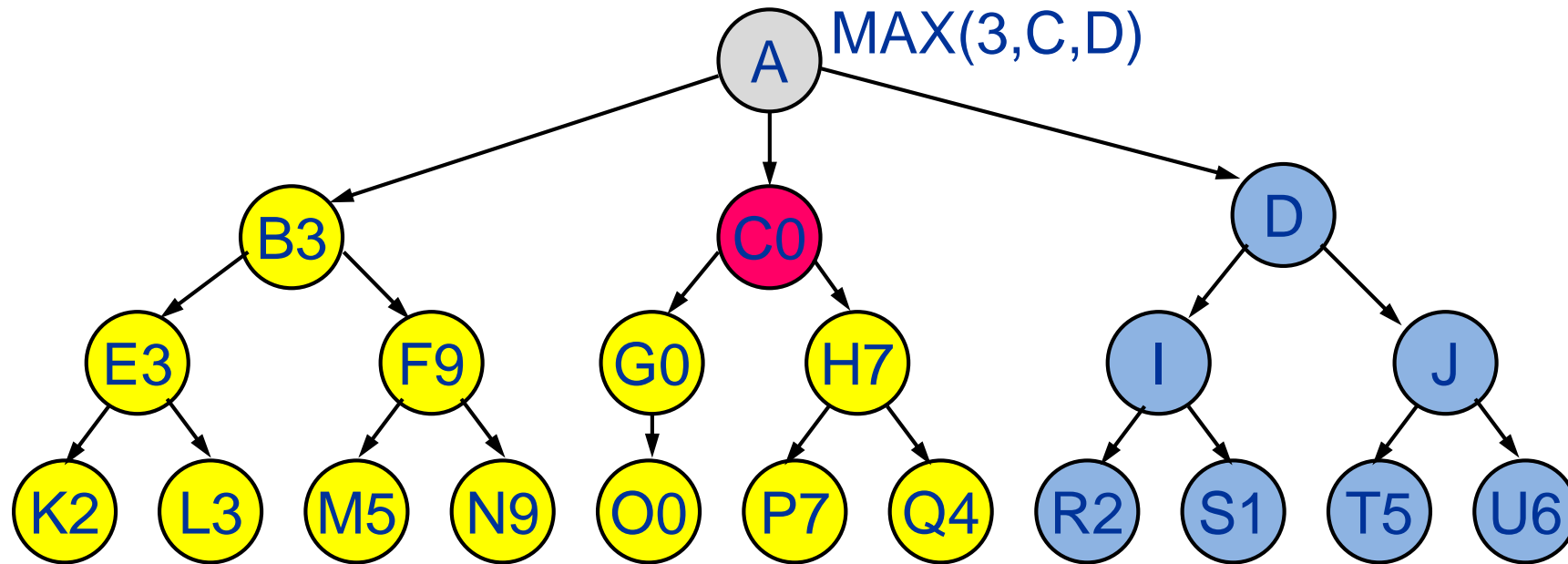
Min-Max algoritam (ilustracija)



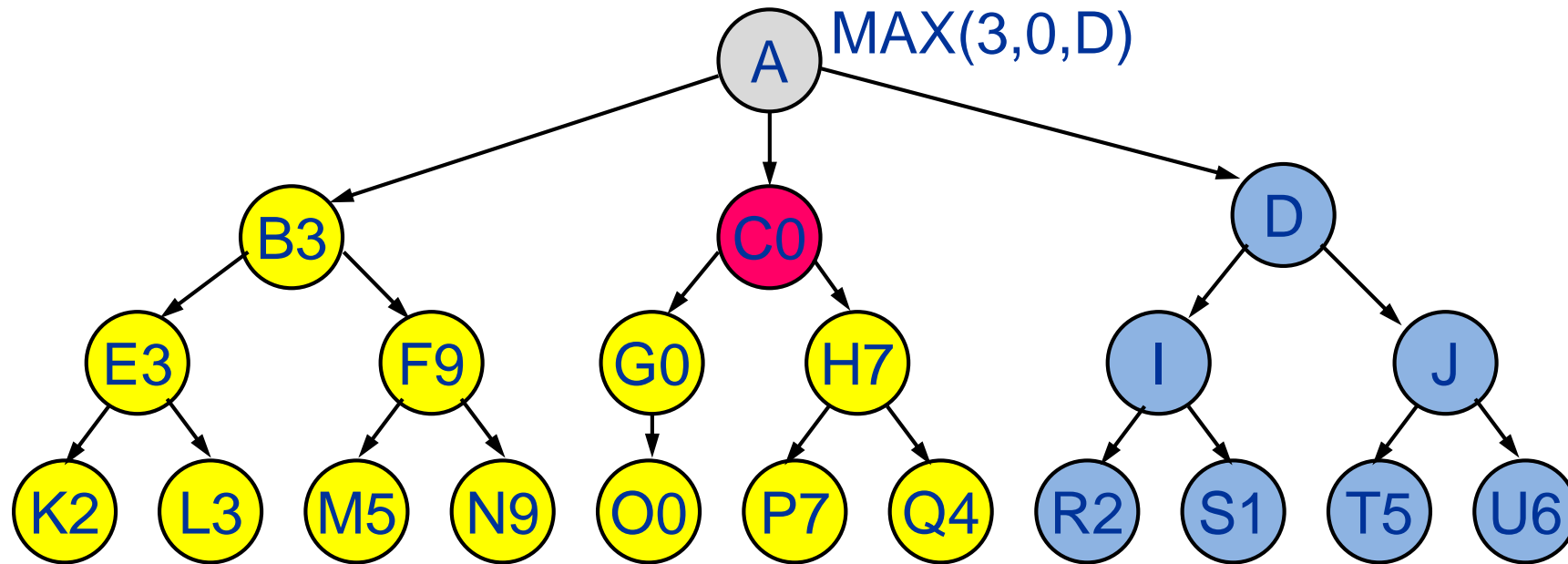
Min-Max algoritam (ilustracija)



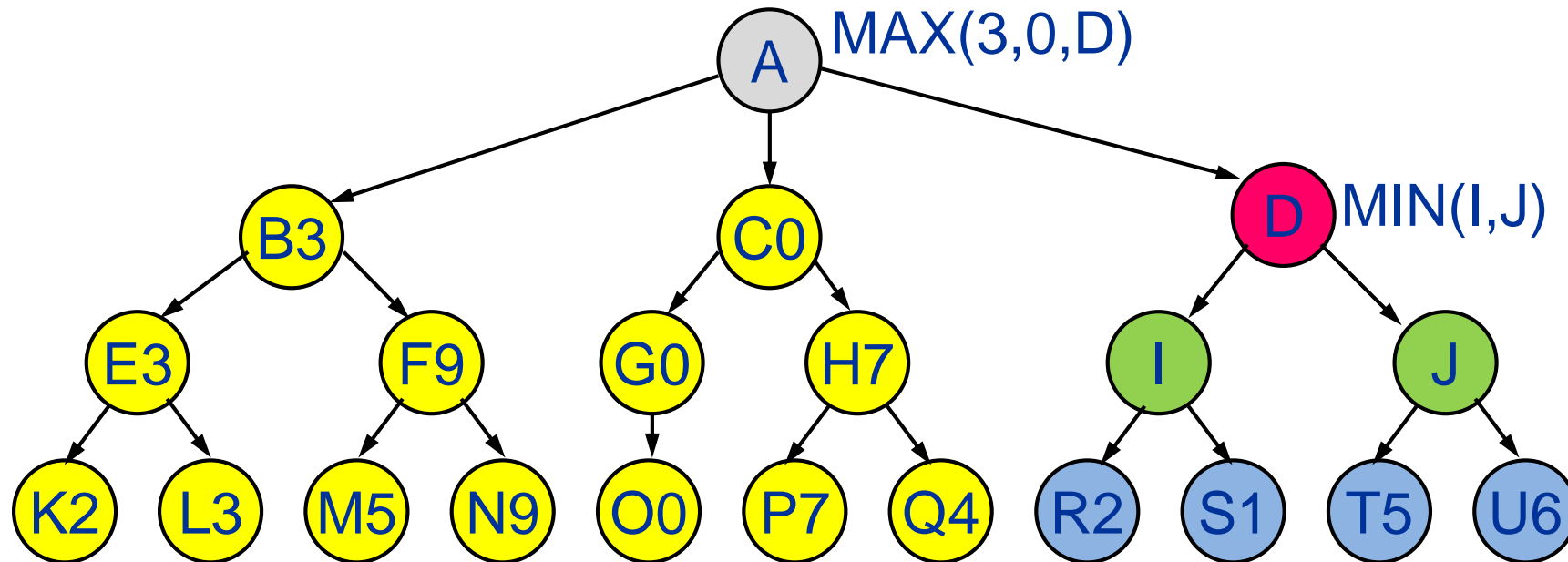
Min-Max algoritam (ilustracija)



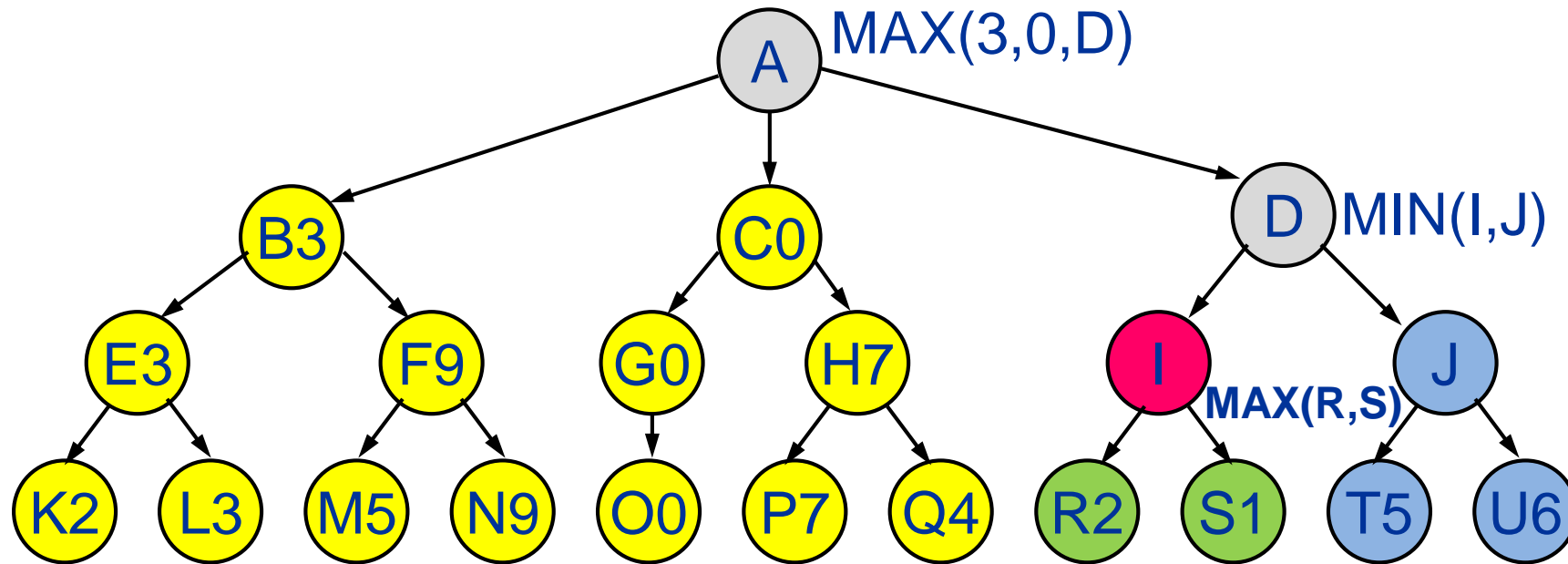
Min-Max algoritam (ilustracija)



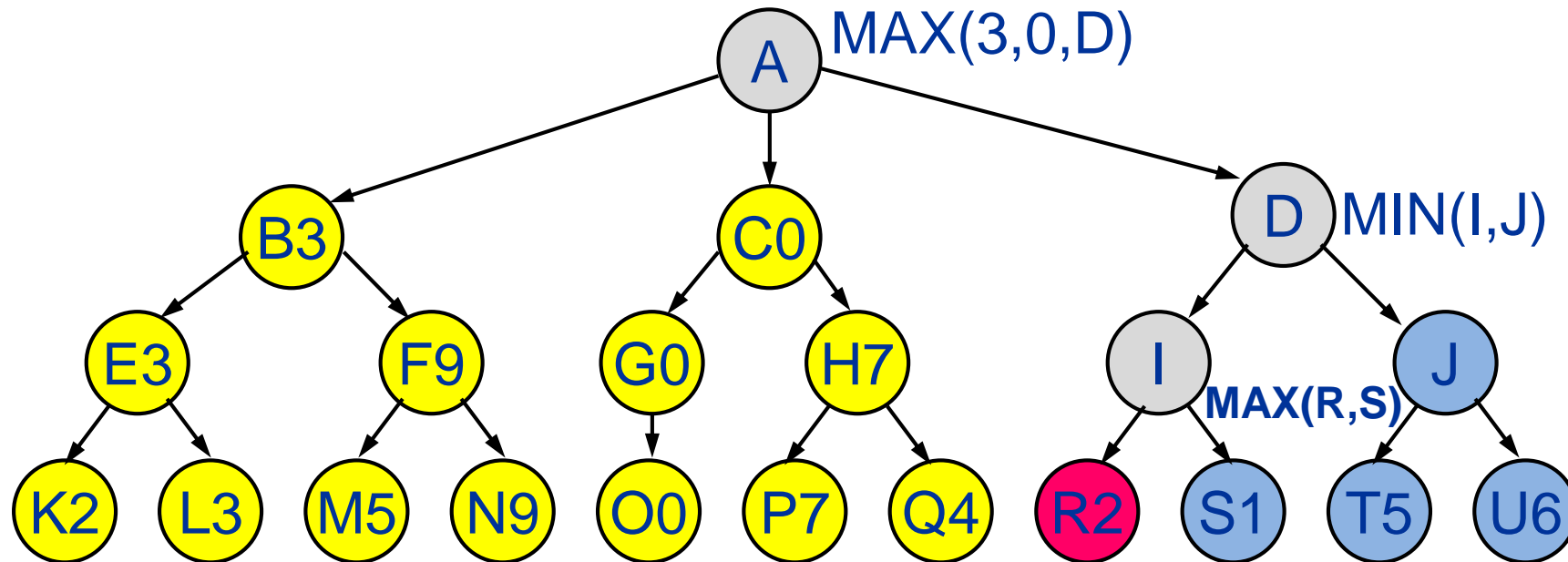
Min-Max algoritam (ilustracija)



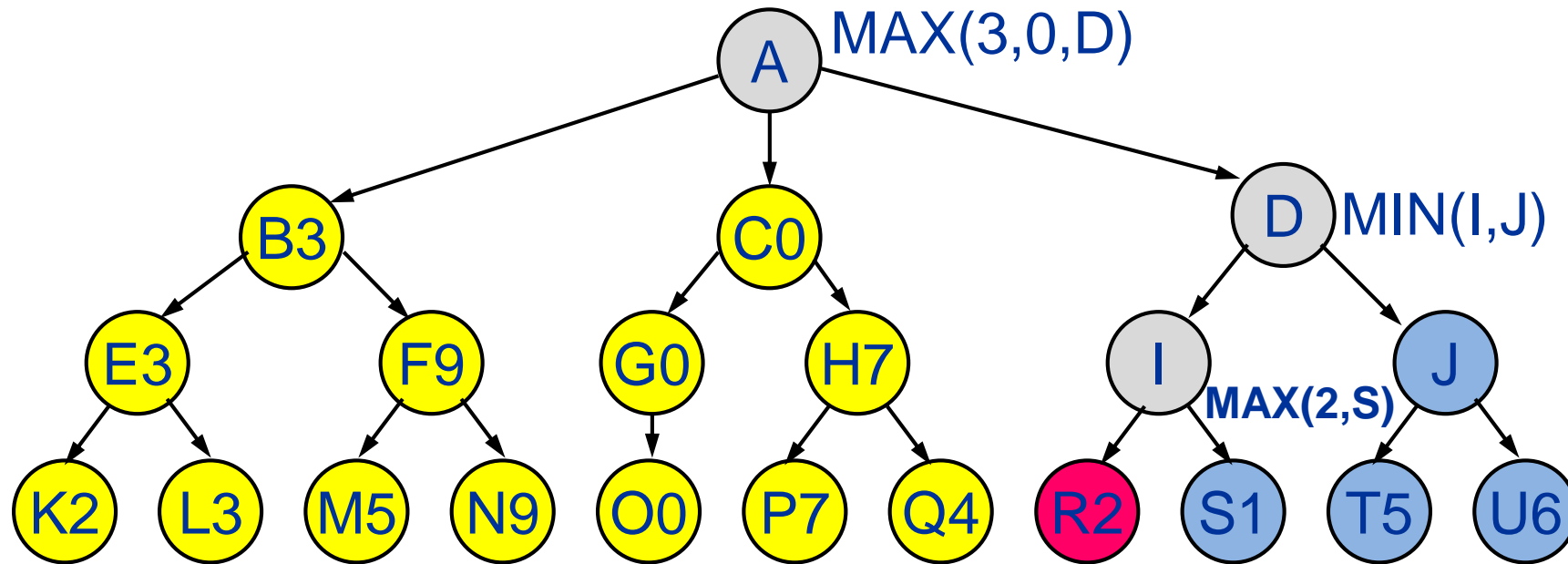
Min-Max algoritam (ilustracija)



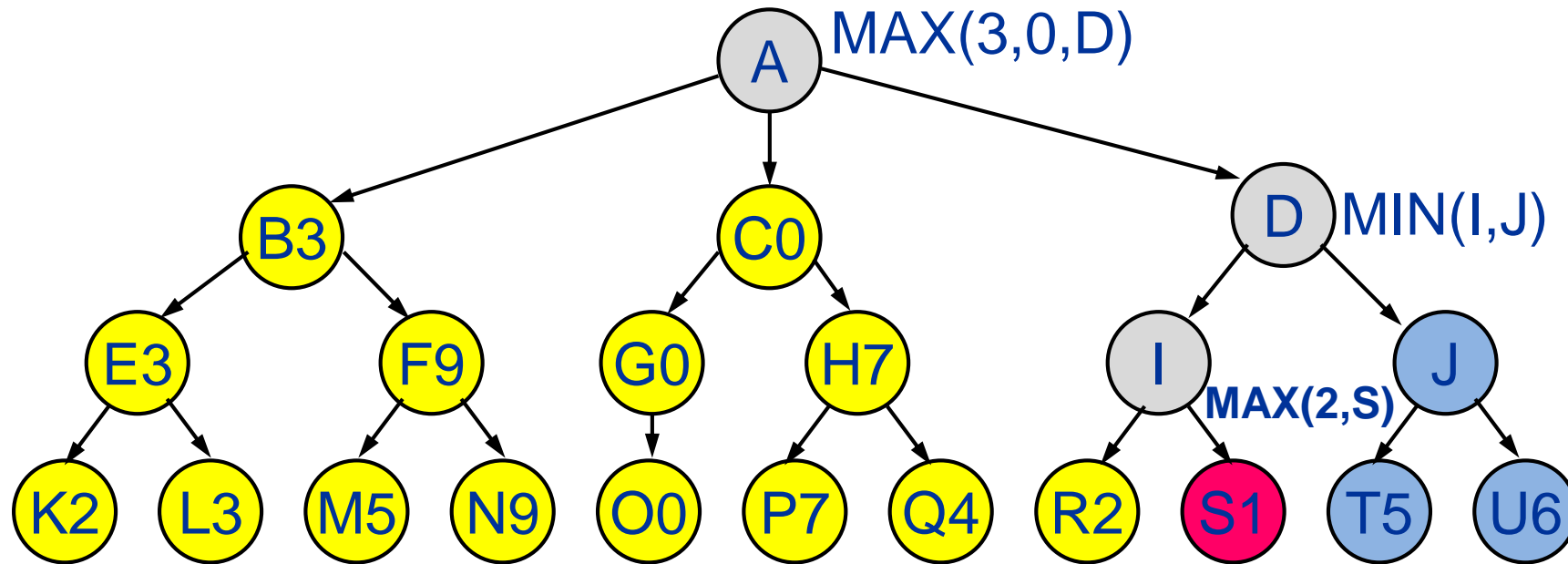
Min-Max algoritam (ilustracija)



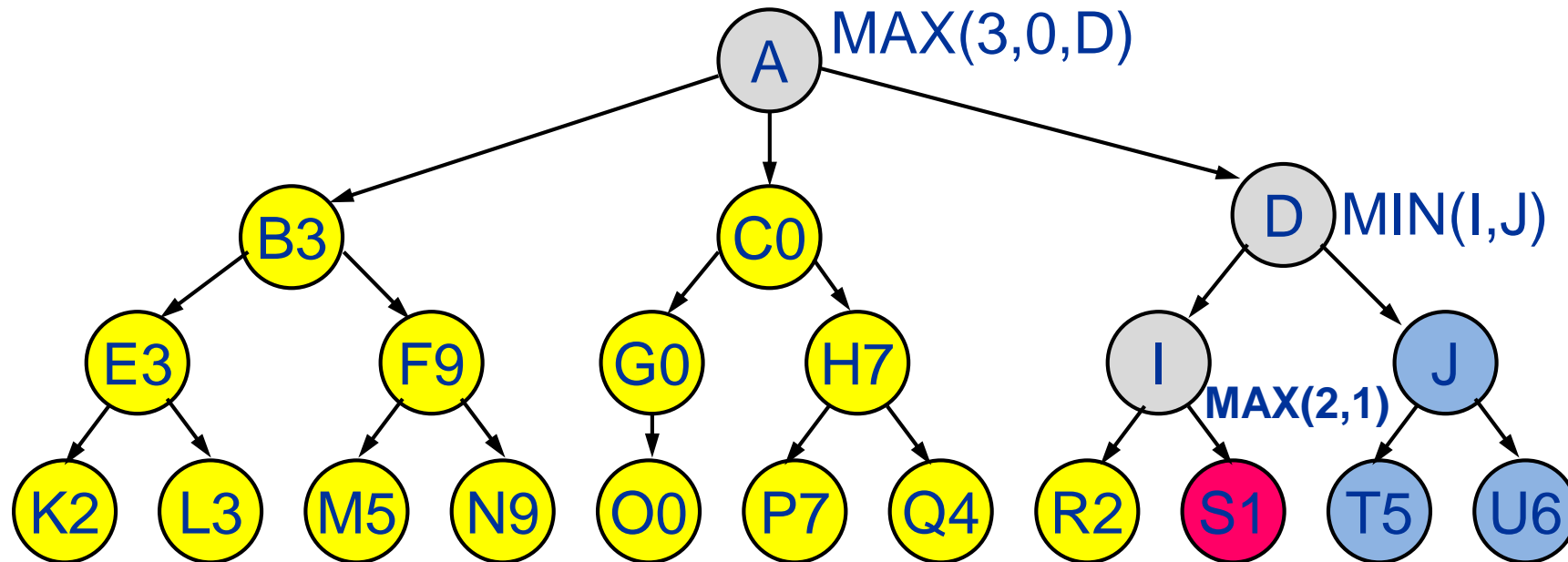
Min-Max algoritam (ilustracija)



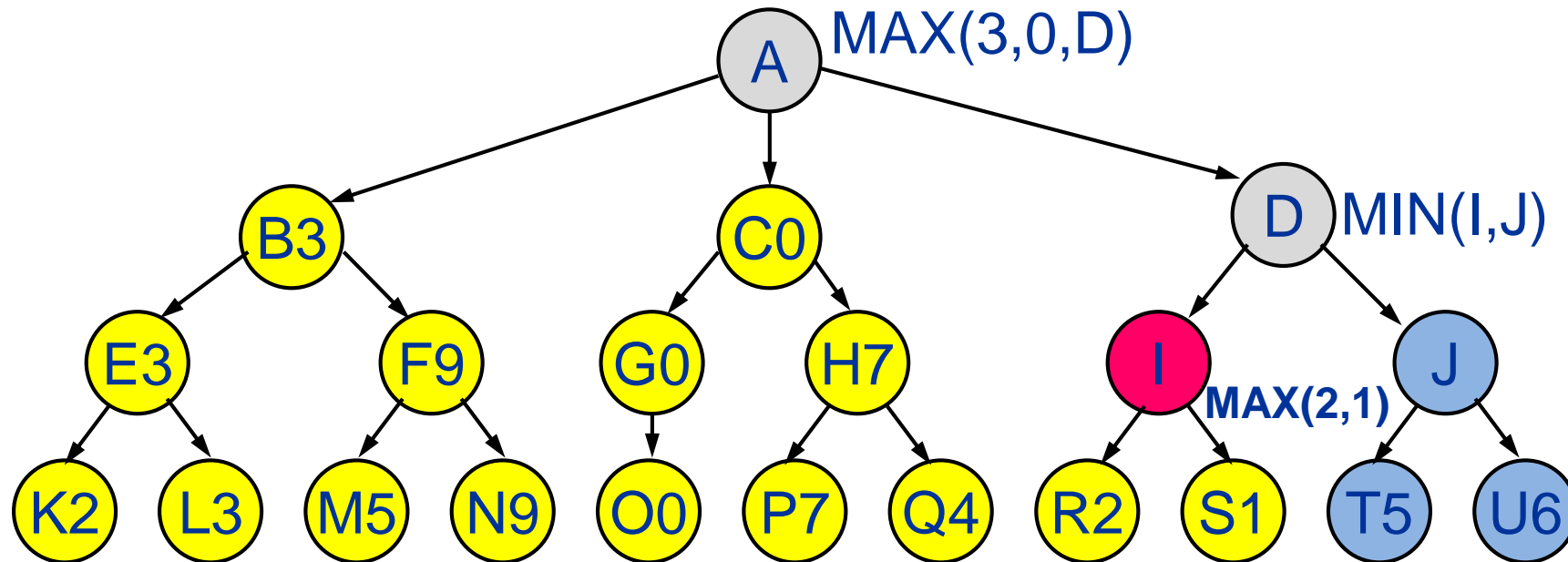
Min-Max algoritam (ilustracija)



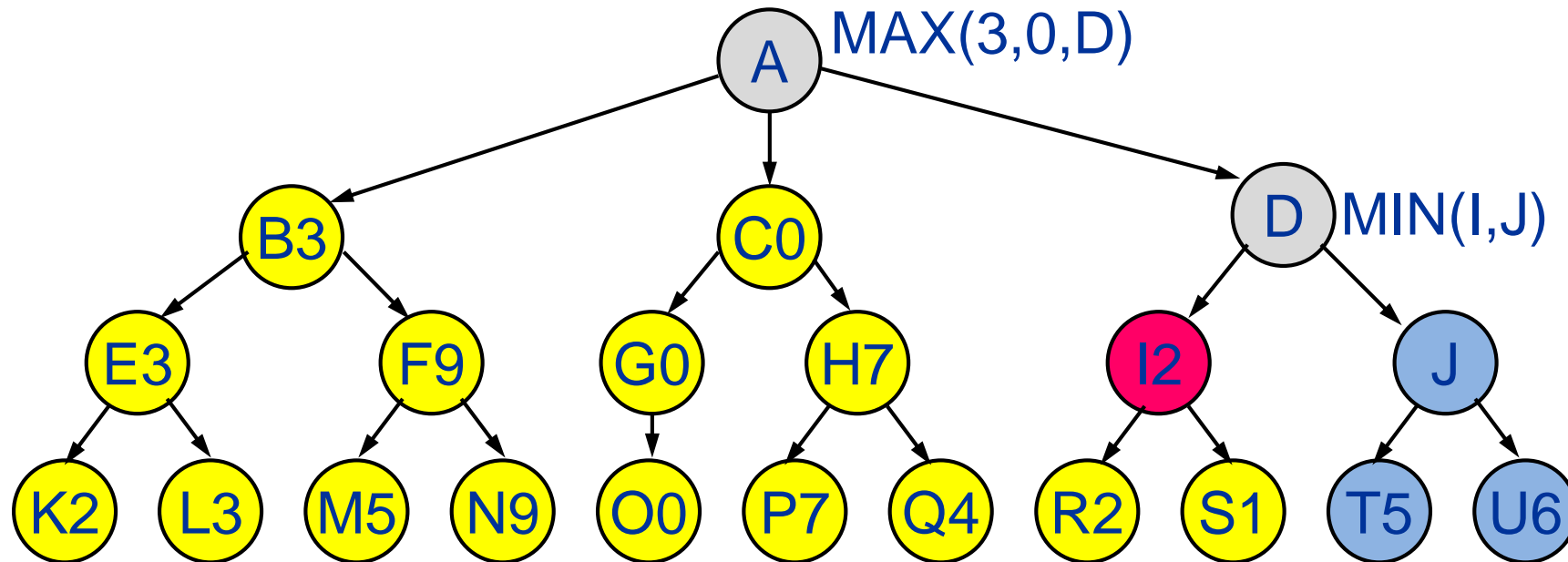
Min-Max algoritam (ilustracija)



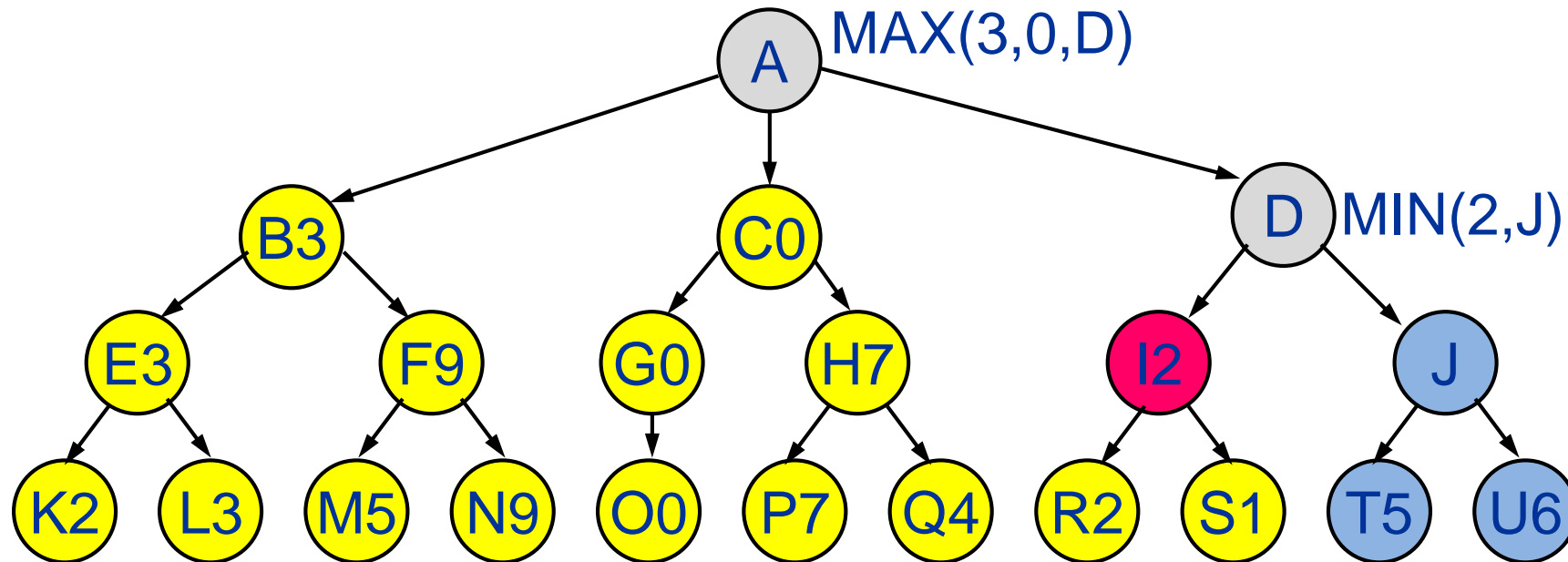
Min-Max algoritam (ilustracija)



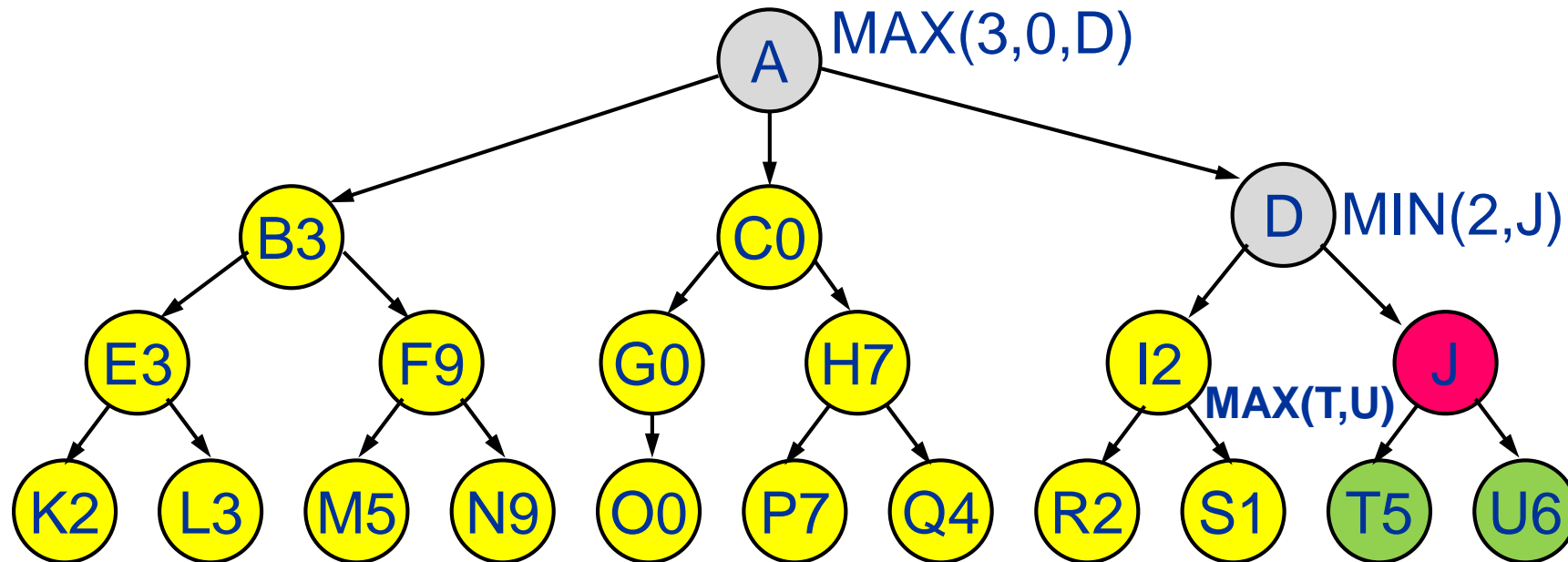
Min-Max algoritam (ilustracija)



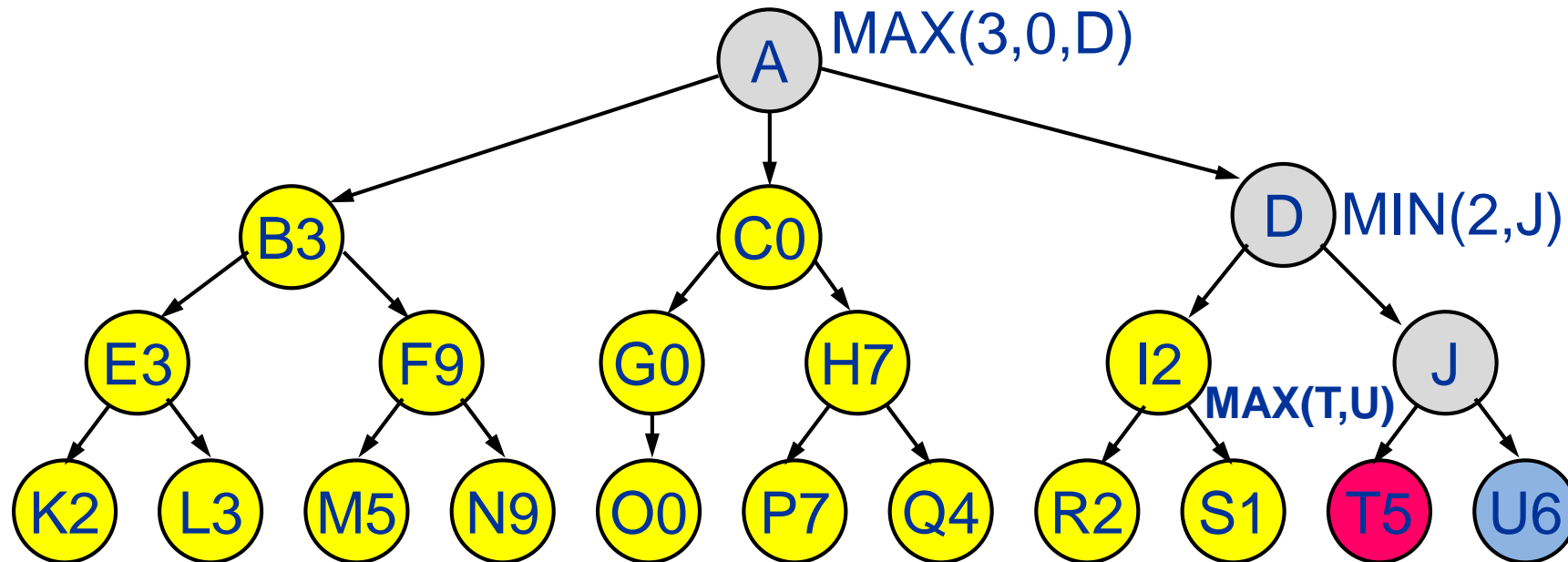
Min-Max algoritam (ilustracija)



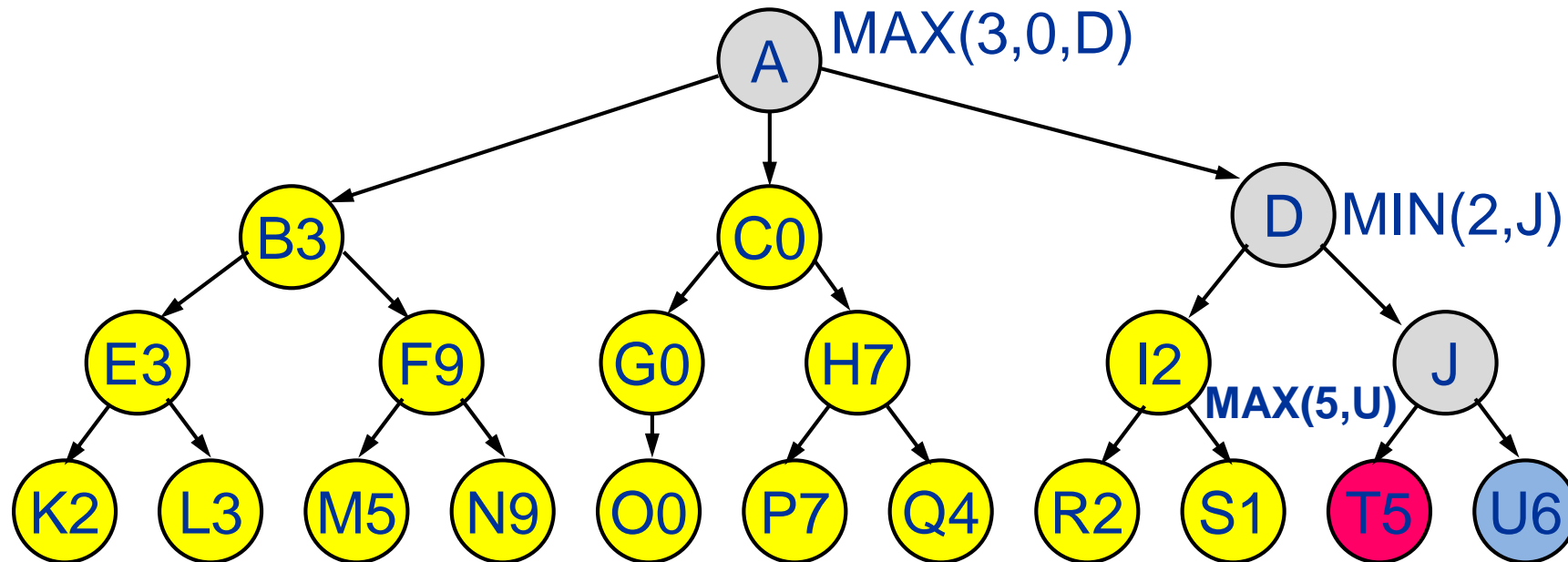
Min-Max algoritam (ilustracija)



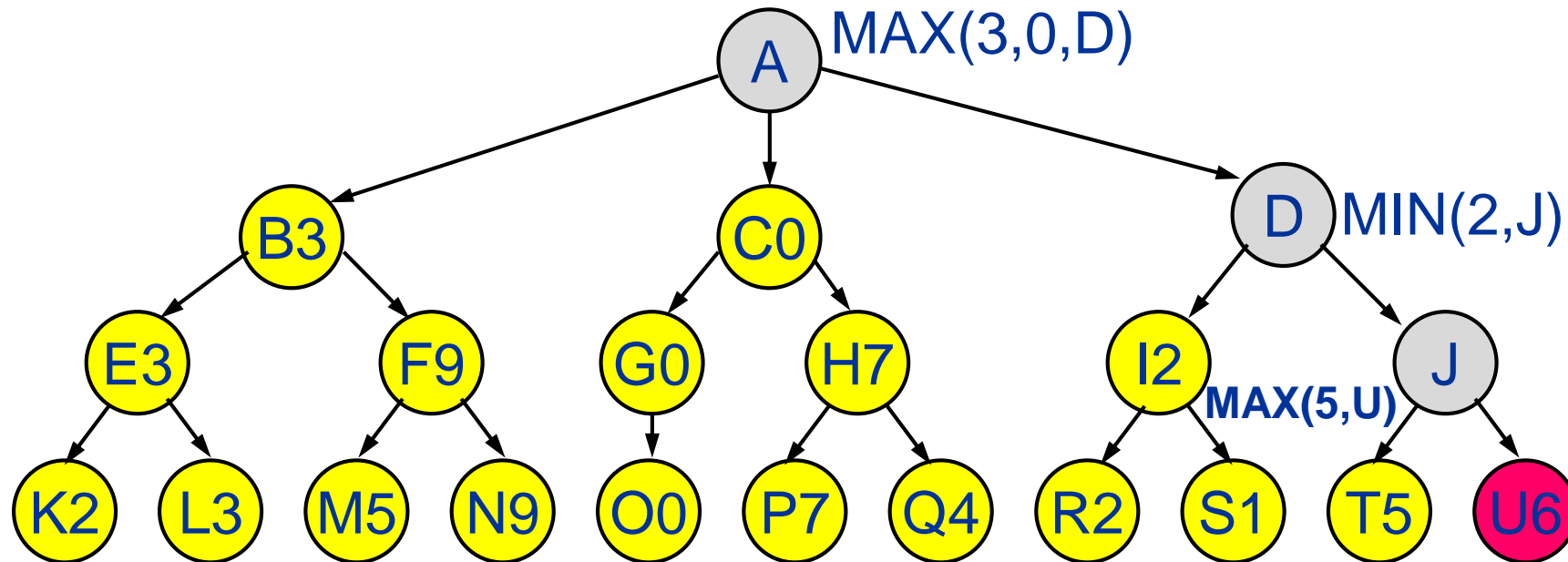
Min-Max algoritam (ilustracija)



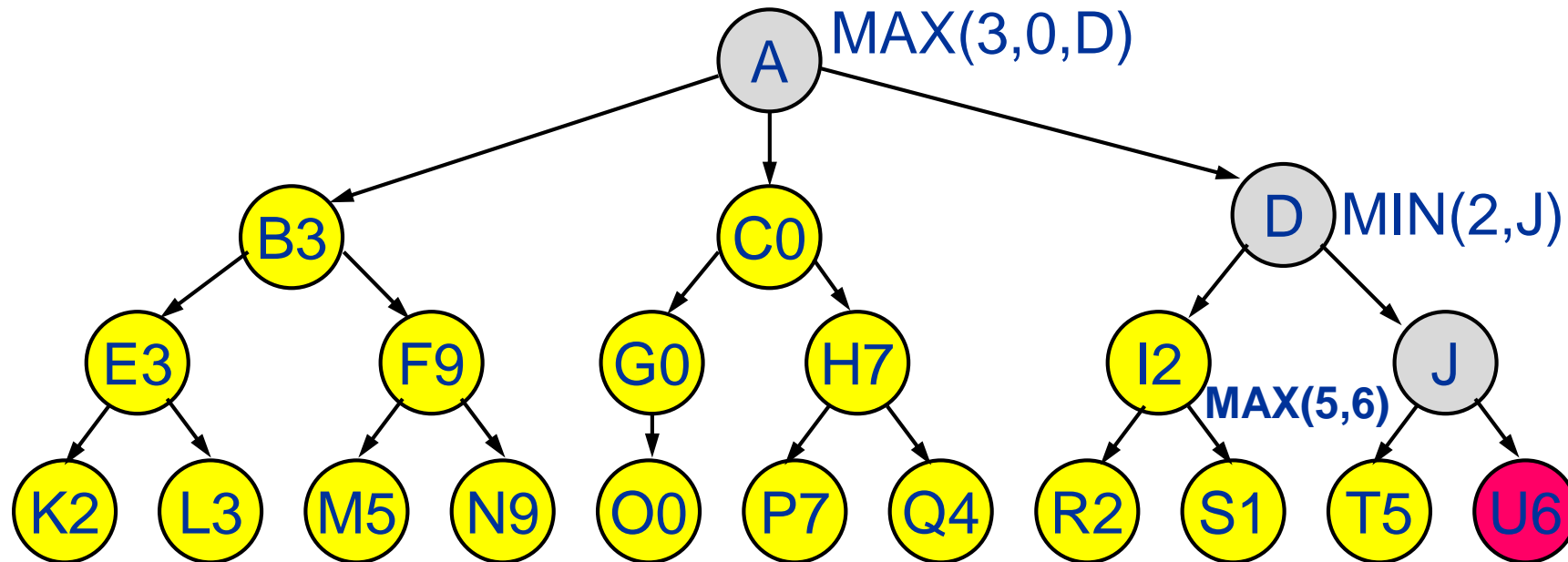
Min-Max algoritam (ilustracija)



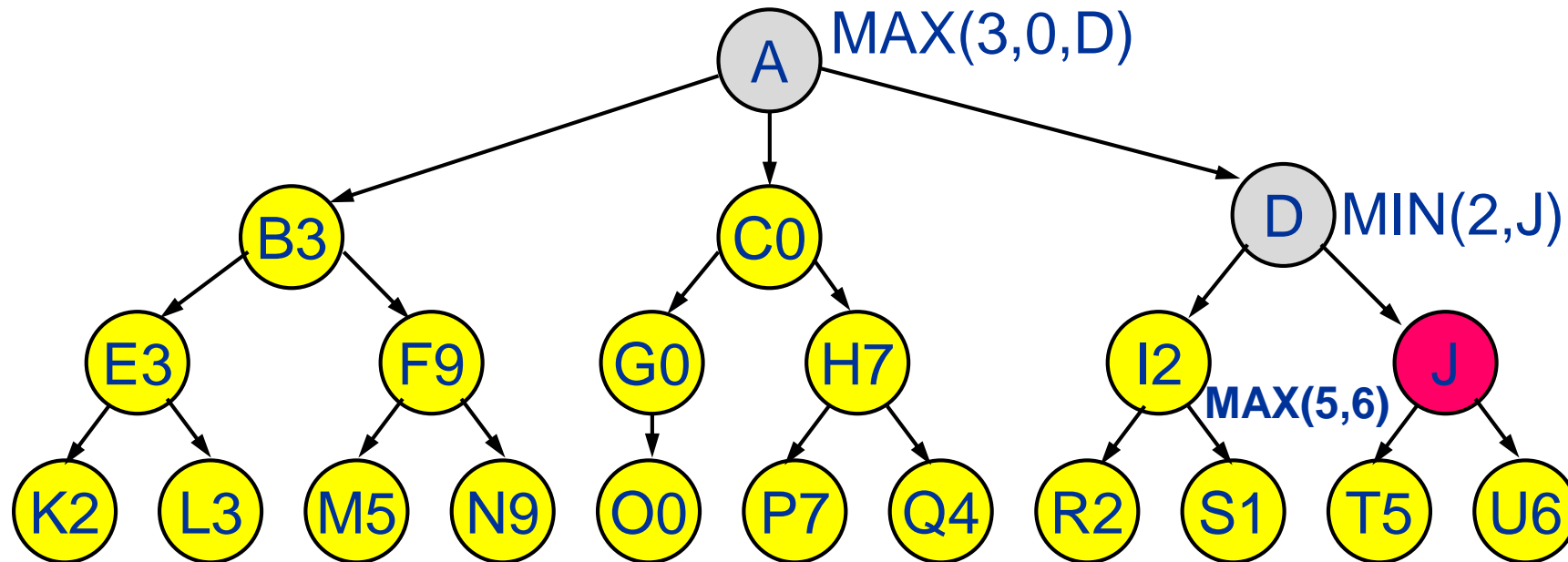
Min-Max algoritam (ilustracija)



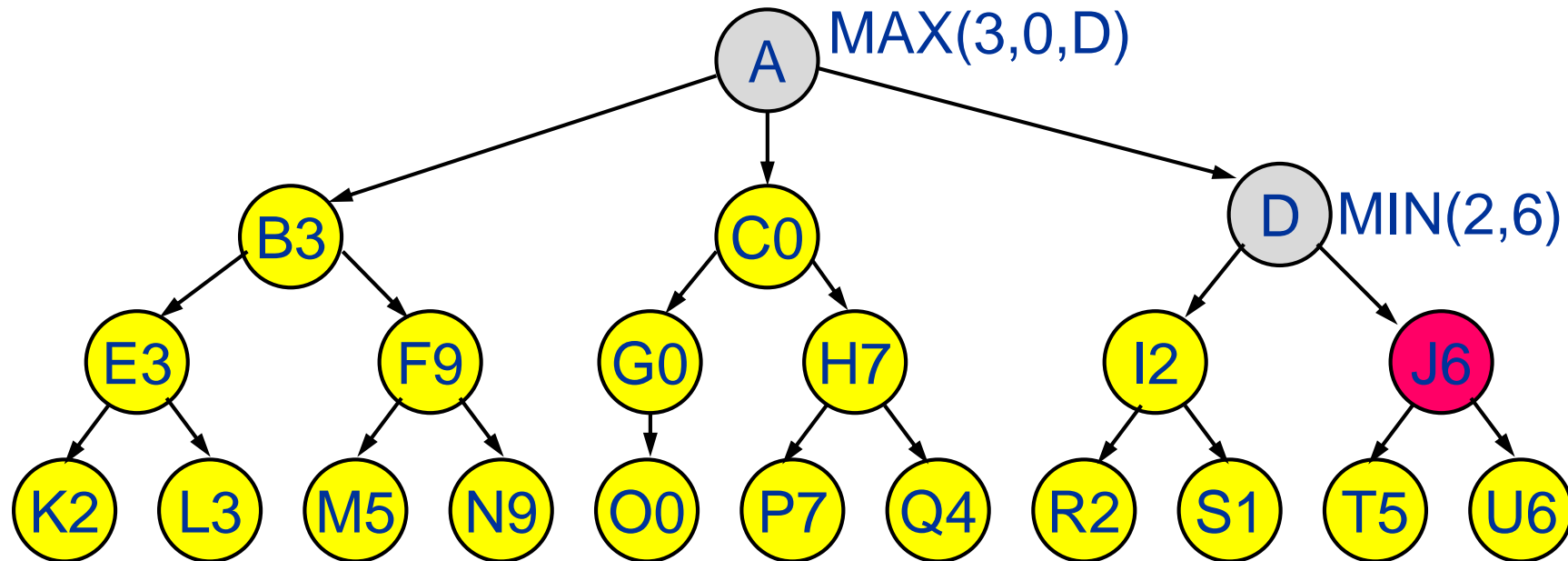
Min-Max algoritam (ilustracija)



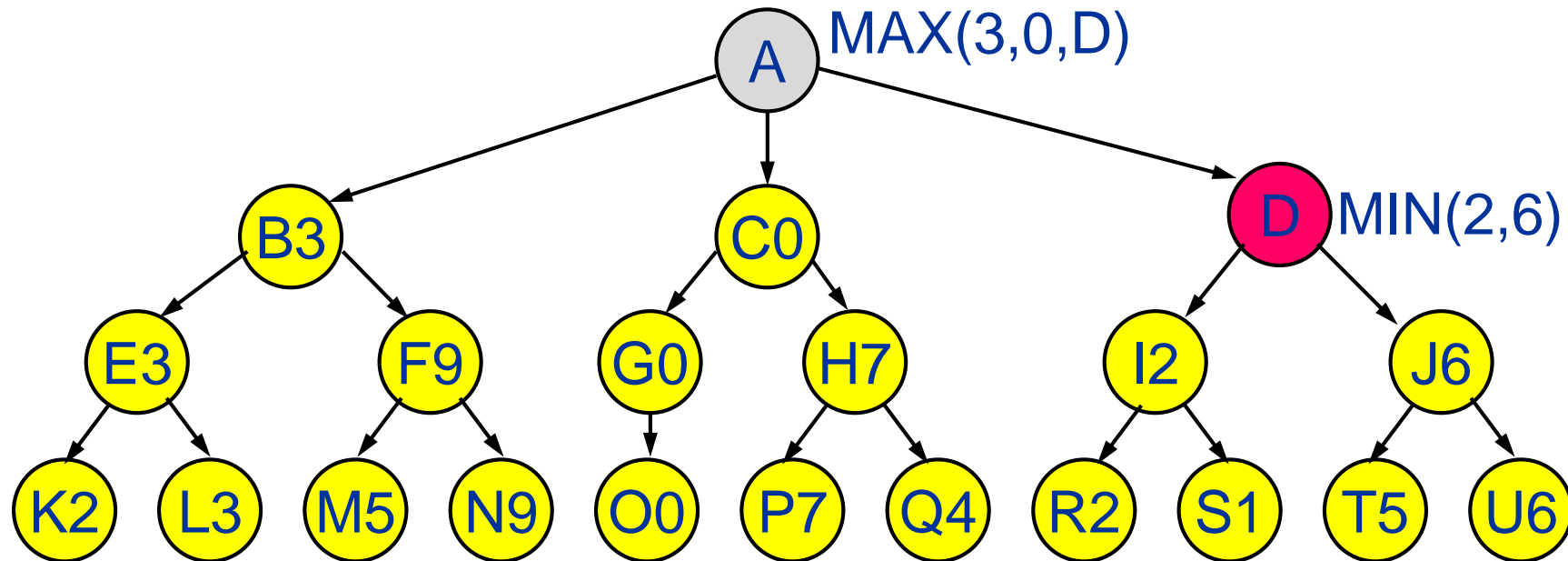
Min-Max algoritam (ilustracija)



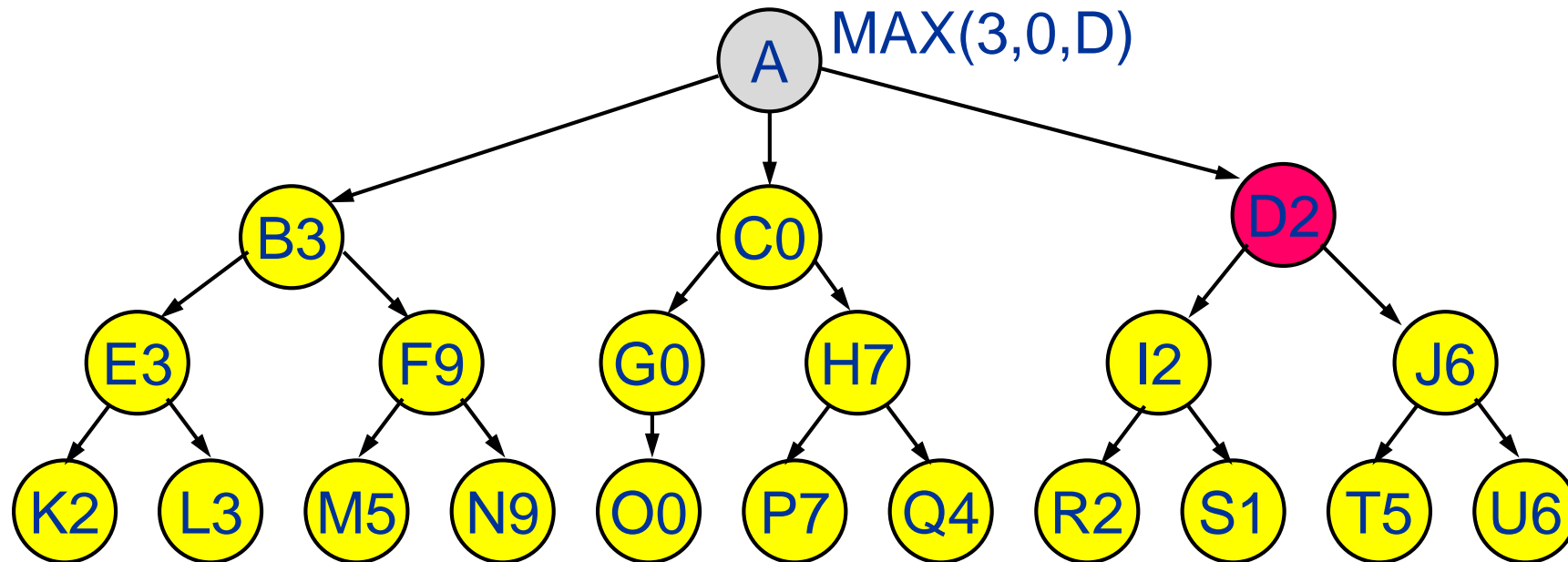
Min-Max algoritam (ilustracija)



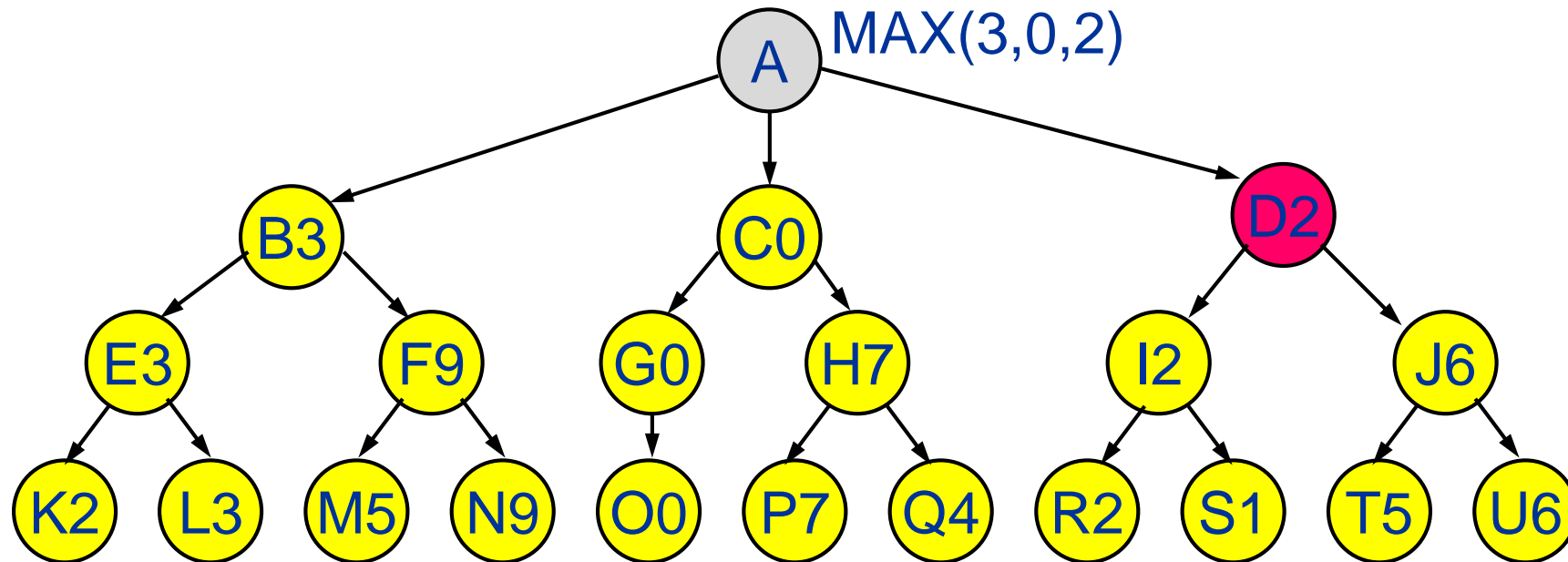
Min-Max algoritam (ilustracija)



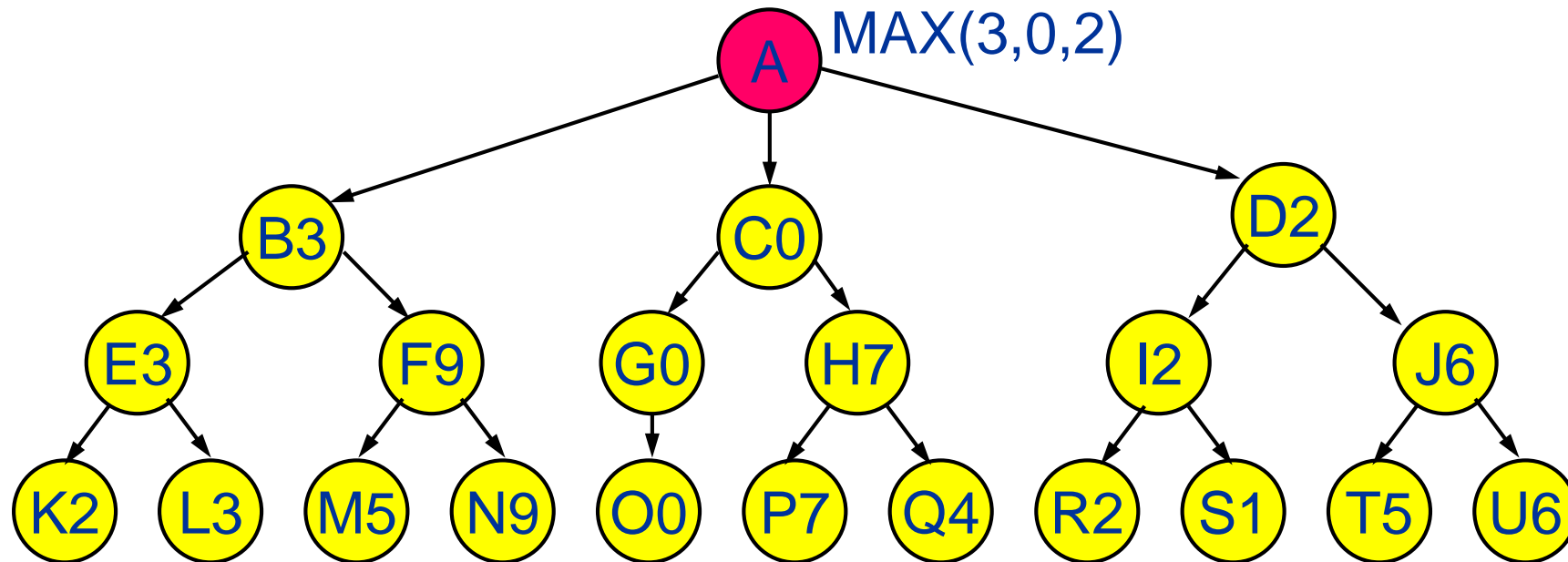
Min-Max algoritam (ilustracija)



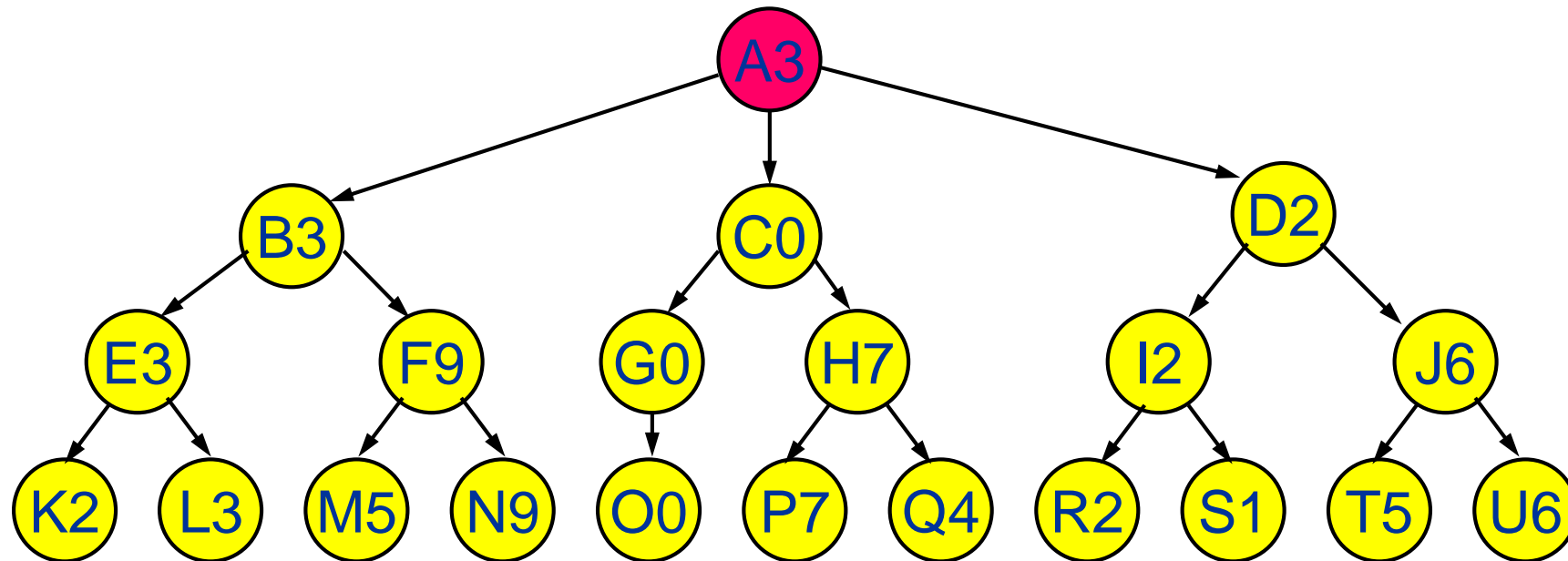
Min-Max algoritam (ilustracija)



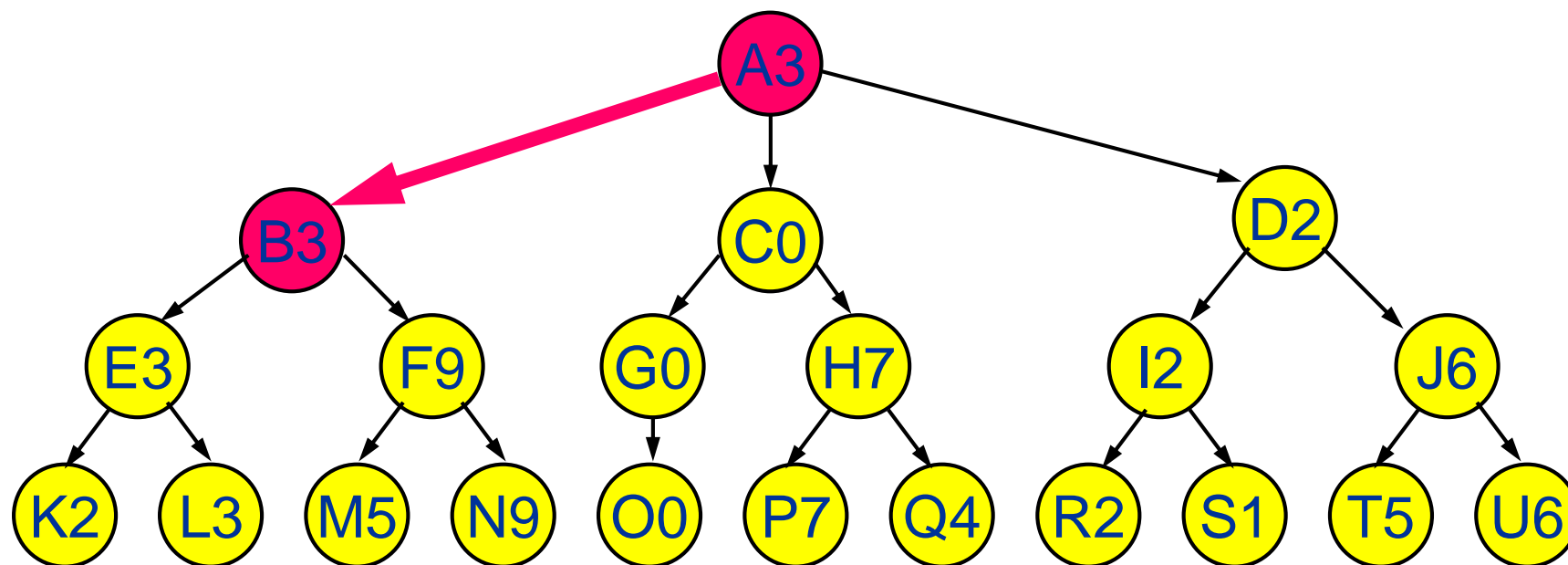
Min-Max algoritam (ilustracija)



Min-Max algoritam (ilustracija)



Min-Max algoritam (ilustracija)



Najbolje je odigrati potez koji igru prevodi iz stanja A u stanje B.



α - β odsecanje – Koncepti

- ▶ Cilj algoritma: smanjiti stablo traženja
- ▶ Odseca grane koje ne obećavaju
- ▶ Osnovna ideja: Pamti se vrednost najboljeg poteza do sada
 - ▶ α – najbolja vrednost za igrača Max
 - ▶ β – najbolja vrednost za igrača Min
- ▶ Kada Max ispituje moguće akcije, ako je bilo koja od njih veća od β (sto je gore po Min), onda može da se prestane sa traženjem (podrazumeva se da Min neće odigrati potez koji nije dobar)



α - β odsecanje – Funkcije

- ▶ Polazne vrednosti: $\alpha = -\infty$, $\beta = +\infty$
- ▶ **max-value** (stanje, graf, α , β , dubina, maxdub)
 - ▶ Ako je dubina == maxdub onda vrati **proceni**(stanje);
 - ▶ Za svako $S \in \text{lebenici}(\text{stanje})$
 - ▶ $\alpha = \max(\alpha, \text{min-value}(S, \text{graf}, \alpha, \beta, \text{dubina}-1, \text{maxdub}))$
 - ▶ Ako je $\alpha \geq \beta$ onda vrati β ;
 - ▶ Vrati α ;
- ▶ **min-value** (stanje, graf, α , β , dubina, maxdub)
 - ▶ Ako je dubina == maxdub onda vrati **proceni**(stanje);
 - ▶ Za svako $S \in \text{lebenici}(\text{stanje})$
 - ▶ $\beta := \min(\beta, \text{max-value}(S, \text{graf}, \alpha, \beta, \text{dubina}-1, \text{maxdub}))$
 - ▶ Ako je $\beta \leq \alpha$ onda vrati α ;
 - ▶ Vrati β ;



α - β odsecanje – Funkcije - Python (max_value)

```
def max_value(stanje, dubina, alpha, beta):
    if dubina == 0:
        return (stanje, proceni_stanje(stanje))
    else:
        for s in nova_stanja(stanje):
            alpha = max(alpha,
                        min_value(s, dubina - 1, alpha, beta),
                        key=lambda x: x[1])

            if alpha[1] >= beta[1]:
                return beta
        return alpha
```



α - β odsecanje – Funkcije - Python (min_value)

```
def min_value(stanje, dubina, alpha, beta):
    if dubina == 0:
        return (stanje, proceni_stanje(stanje))
    else:
        for s in nova_stanja(stanje):
            beta = min(beta,
                        max_value(s, dubina - 1, alpha, beta),
                        key=lambda x: x[1])

            if beta[1] <= alpha[1]:
                return alpha
        return beta
```



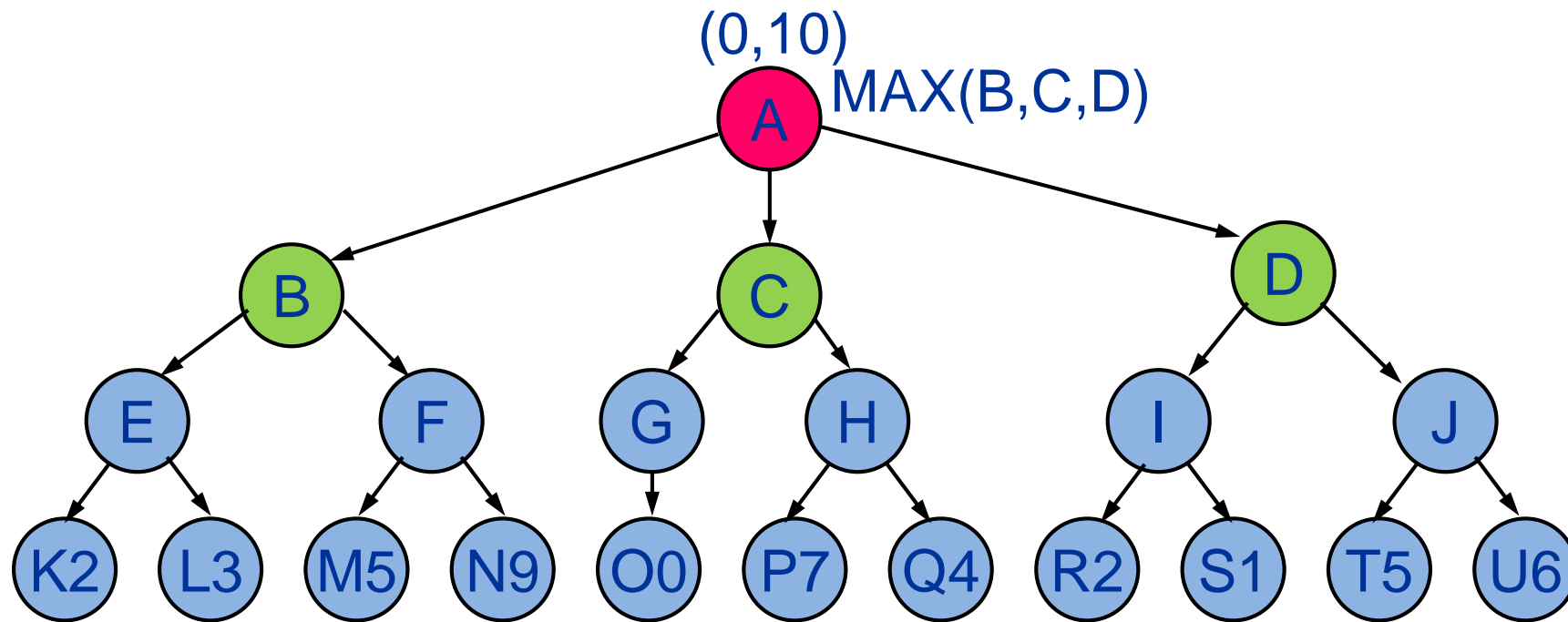
α - β odsecanje – Funkcije - Python (minimax)

```
def minimax(stanje, dubina, moj_potez, alpha = (A, 0), beta = (A, 10)):  
    if moj_potez:  
        return max_value(stanje, dubina, alpha, beta)  
    else:  
        return min_value(stanje, dubina, alpha, beta)
```

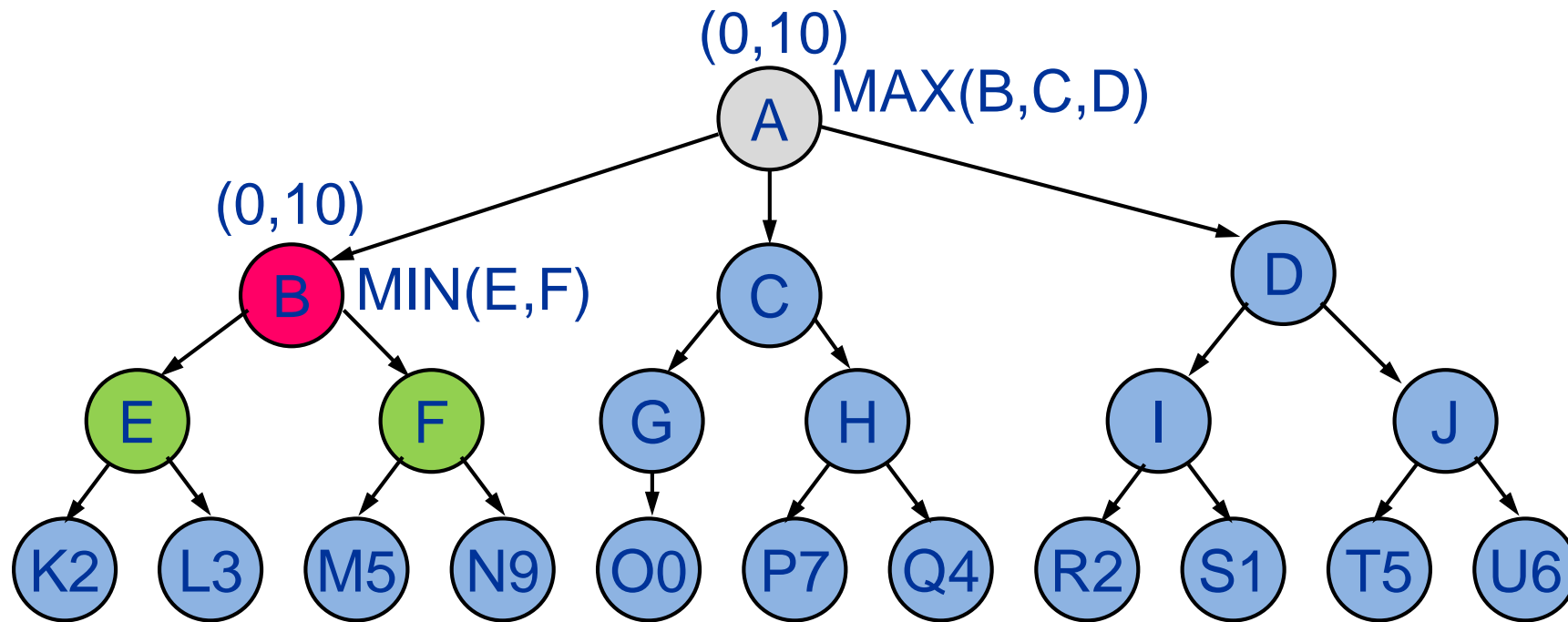
- ▶ minimax funkcija u ovom slučaju služi samo kao „proxy“ funkcija
- ▶ U zavisnosti informacije ko je na potezu, ona samo poziva odgovarajuću funkciju
- ▶ Moguće je pozivati i funkciju `max_value` direktno, pošto će ona uvek vratiti optimalan potez onome ko je na potezu, ali ipak postoje situacije kada želimo da proverimo koji je najgori potez po nas, koji bi protivnik mogao da odigra
- ▶ α , β vrednosti su u ovom primeru tuple podaci, zato što nam je potrebna i informacija o tome koji čvor nosi vrednost koju je algoritam vratio



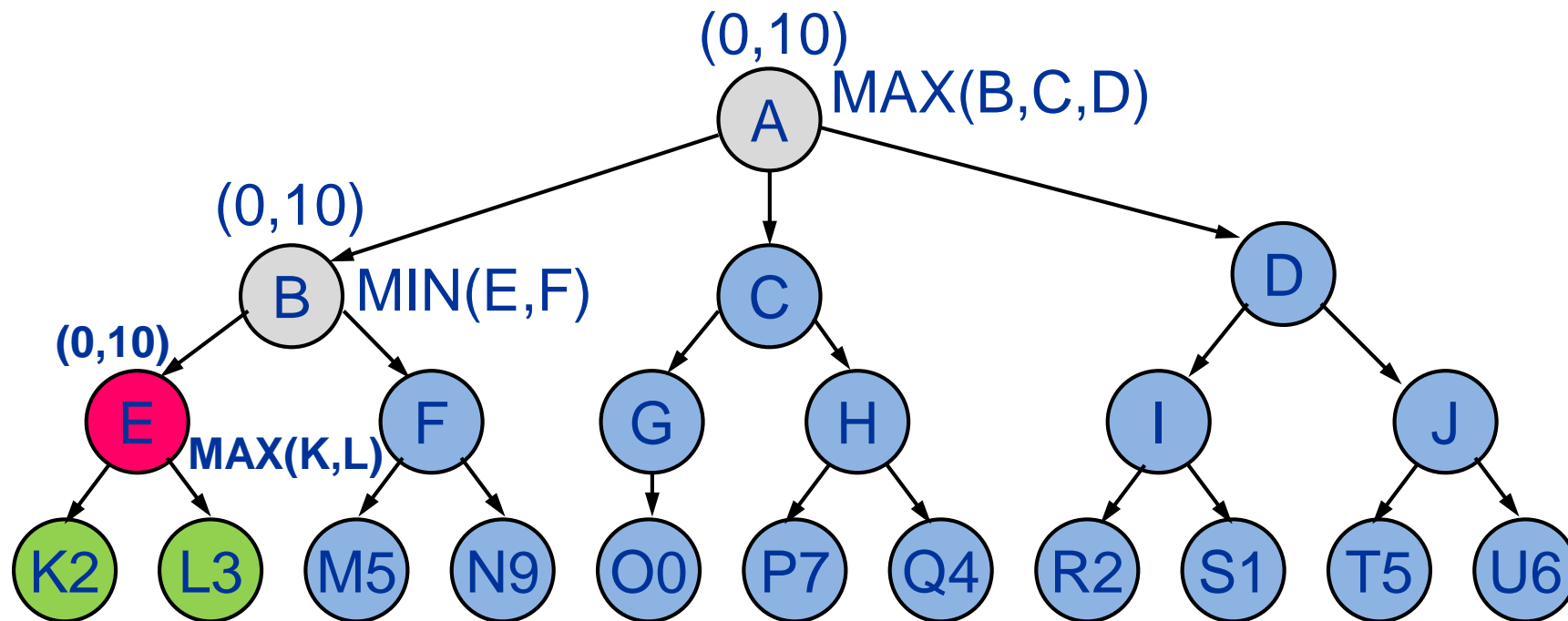
α - β odsecanje (ilustracija)



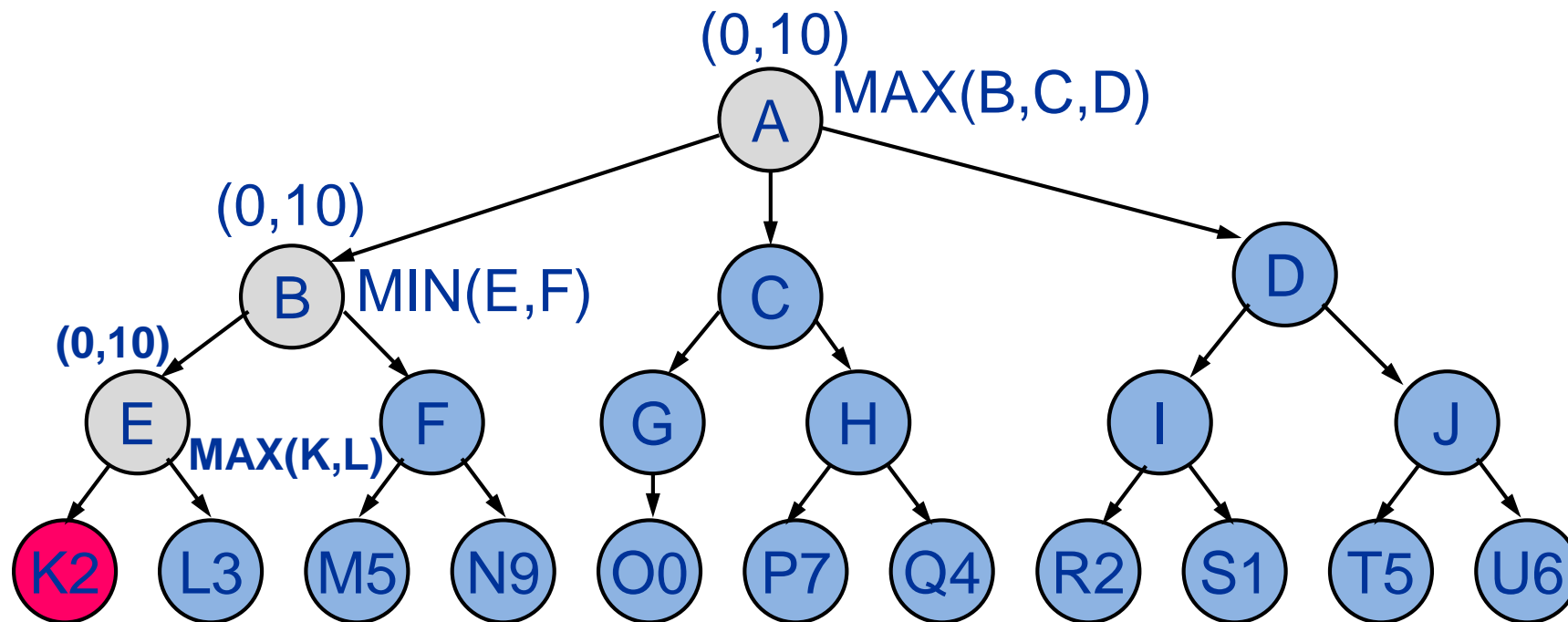
α - β odsecanje (ilustracija)



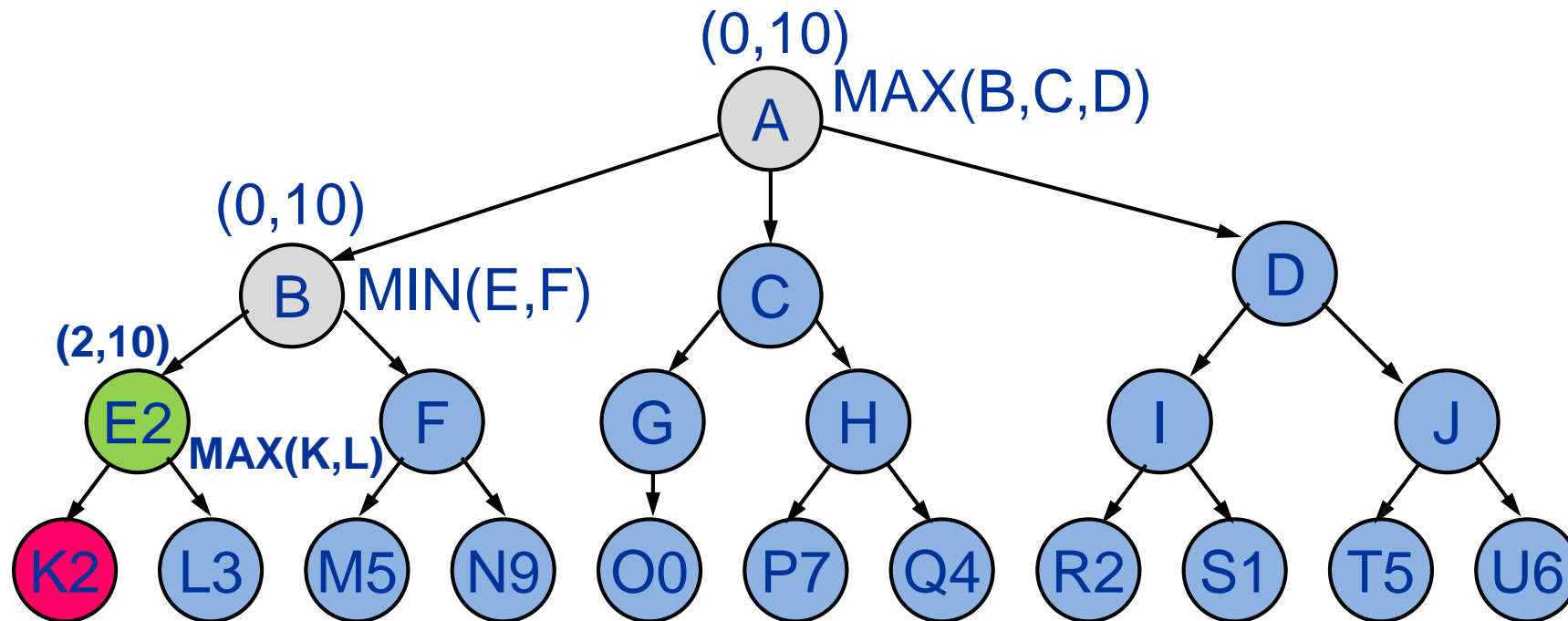
α - β odsecanje (ilustracija)



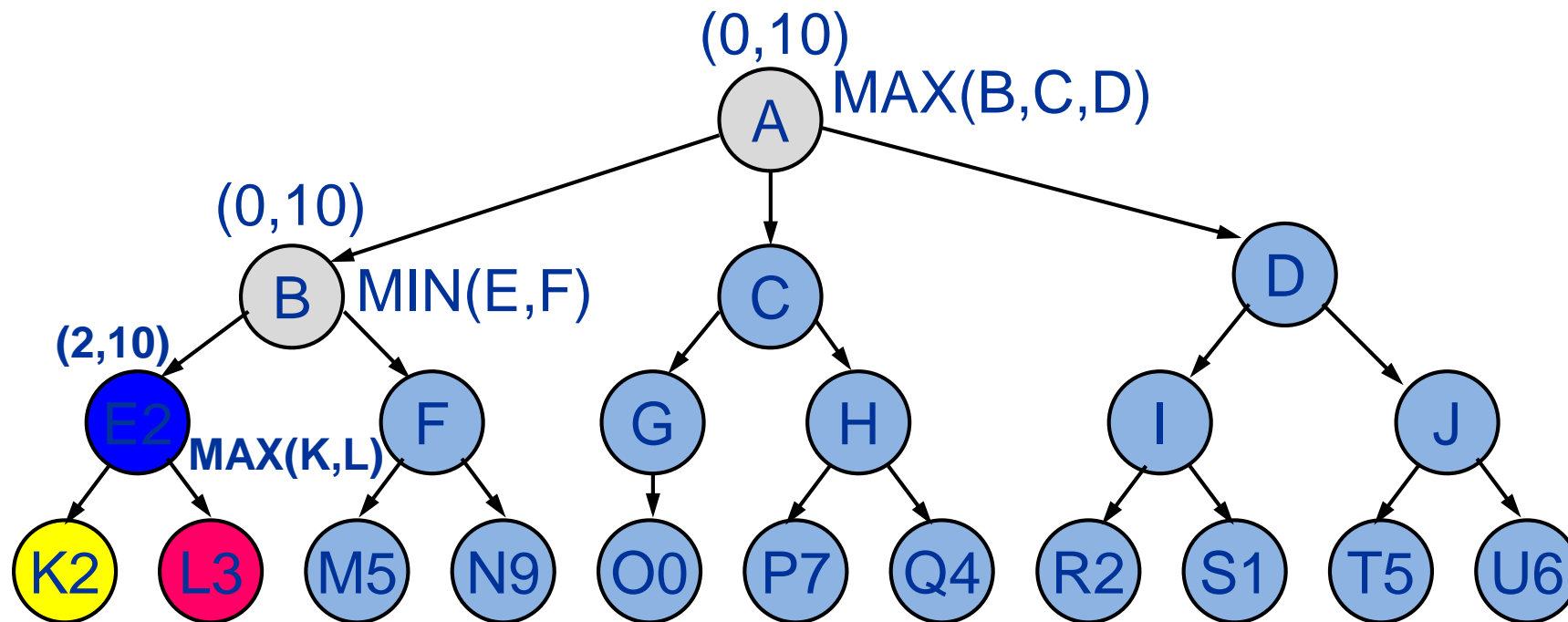
α - β odsecanje (ilustracija)



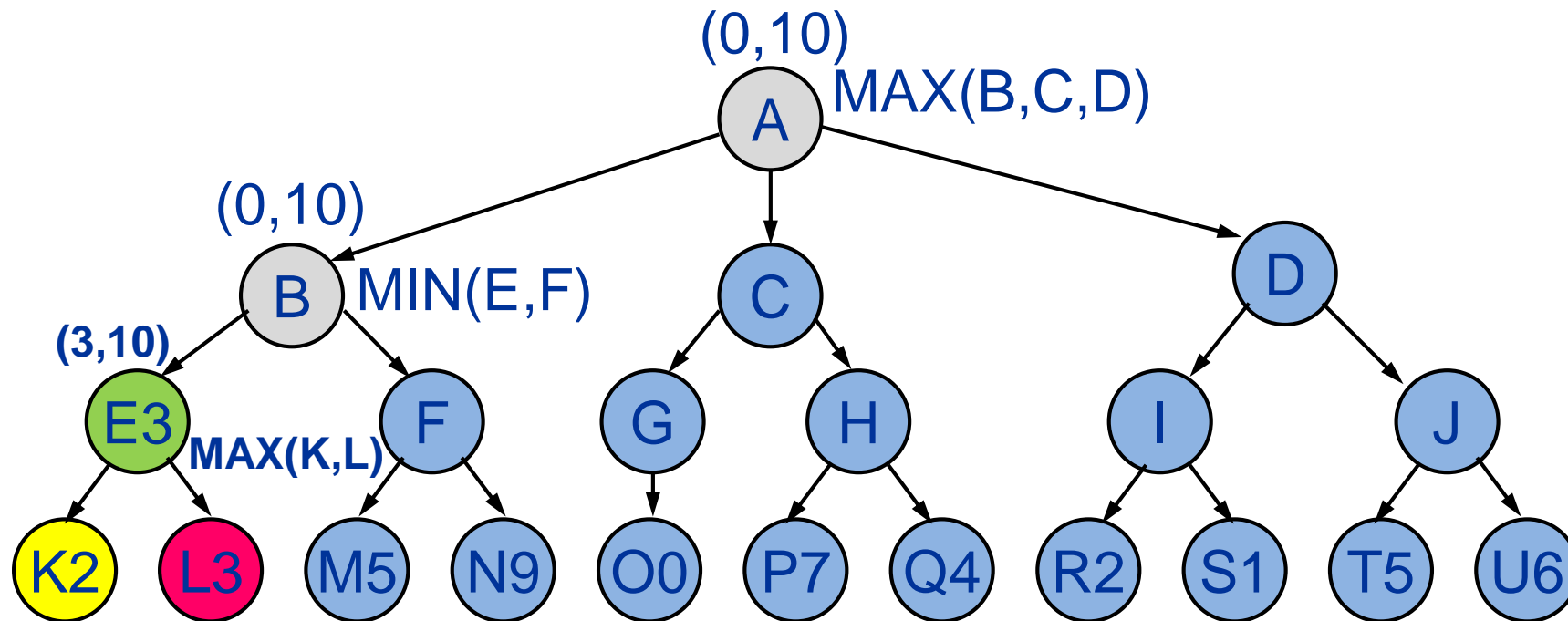
α - β odsecanje (ilustracija)



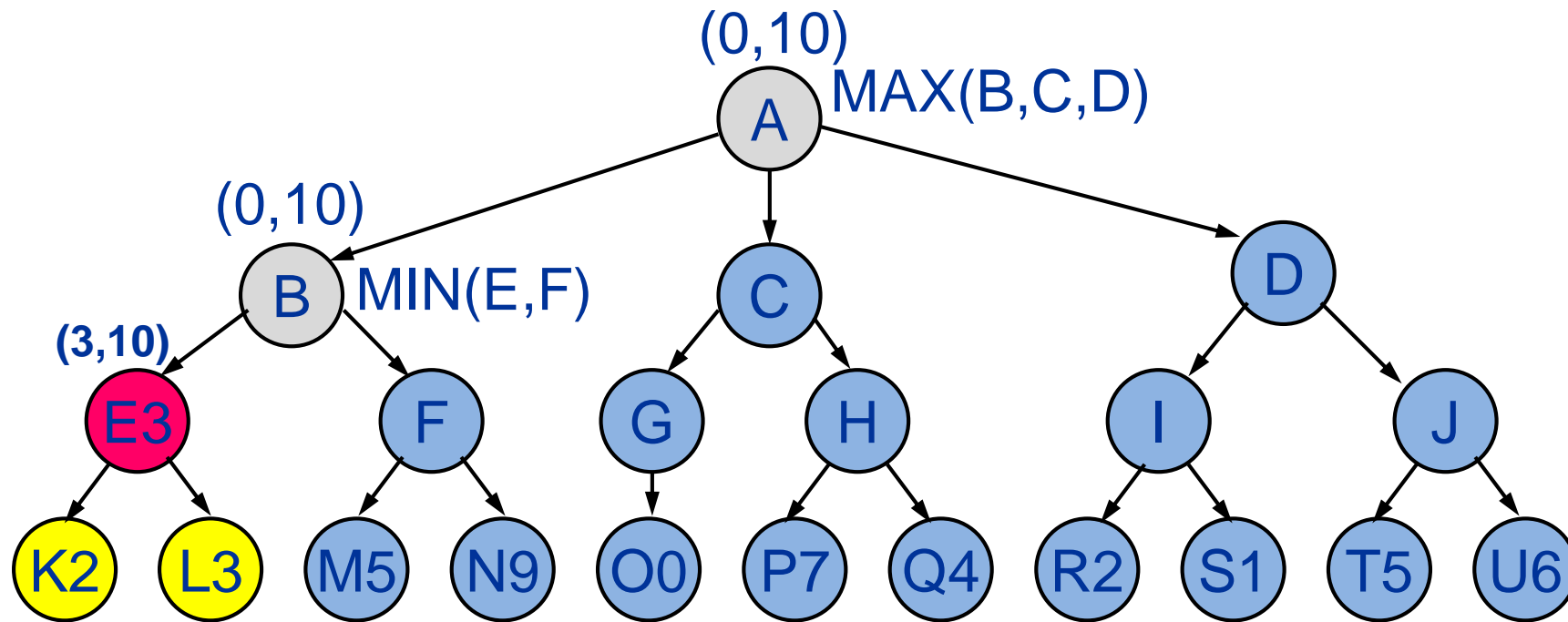
α - β odsecanje (ilustracija)



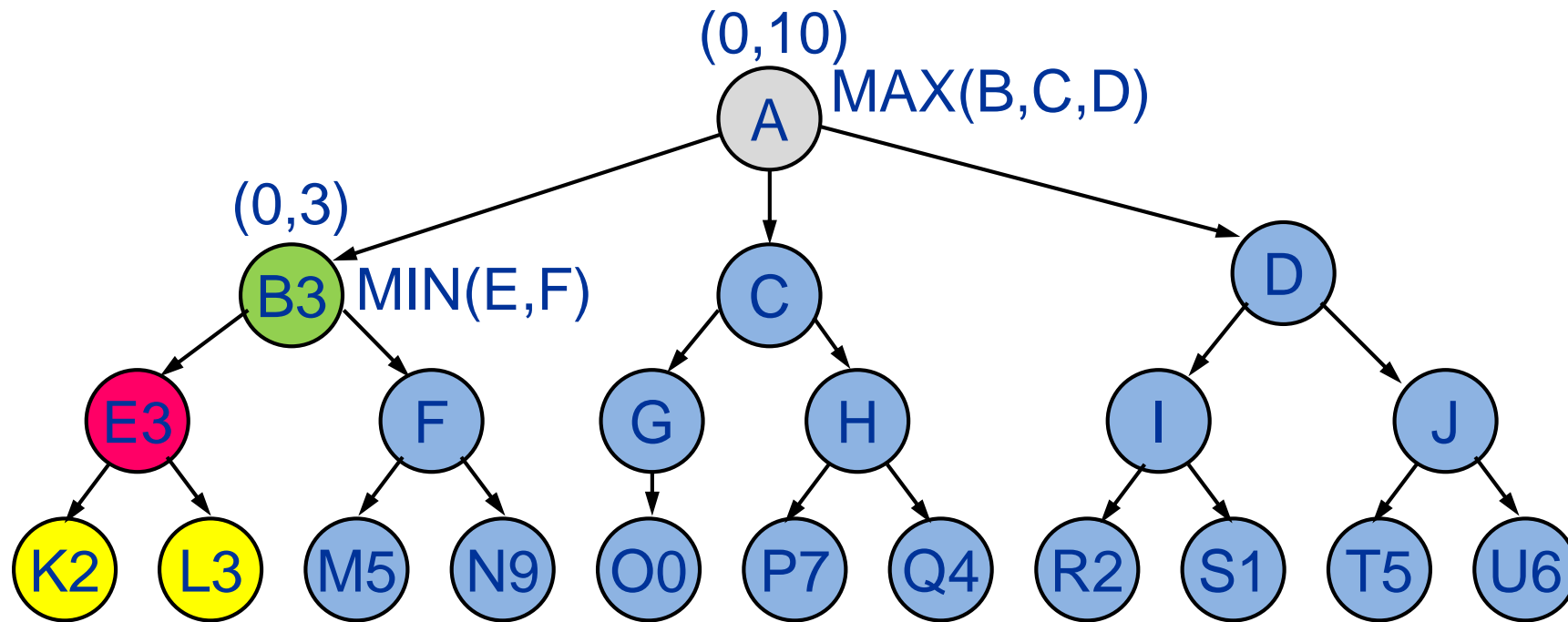
α - β odsecanje (ilustracija)



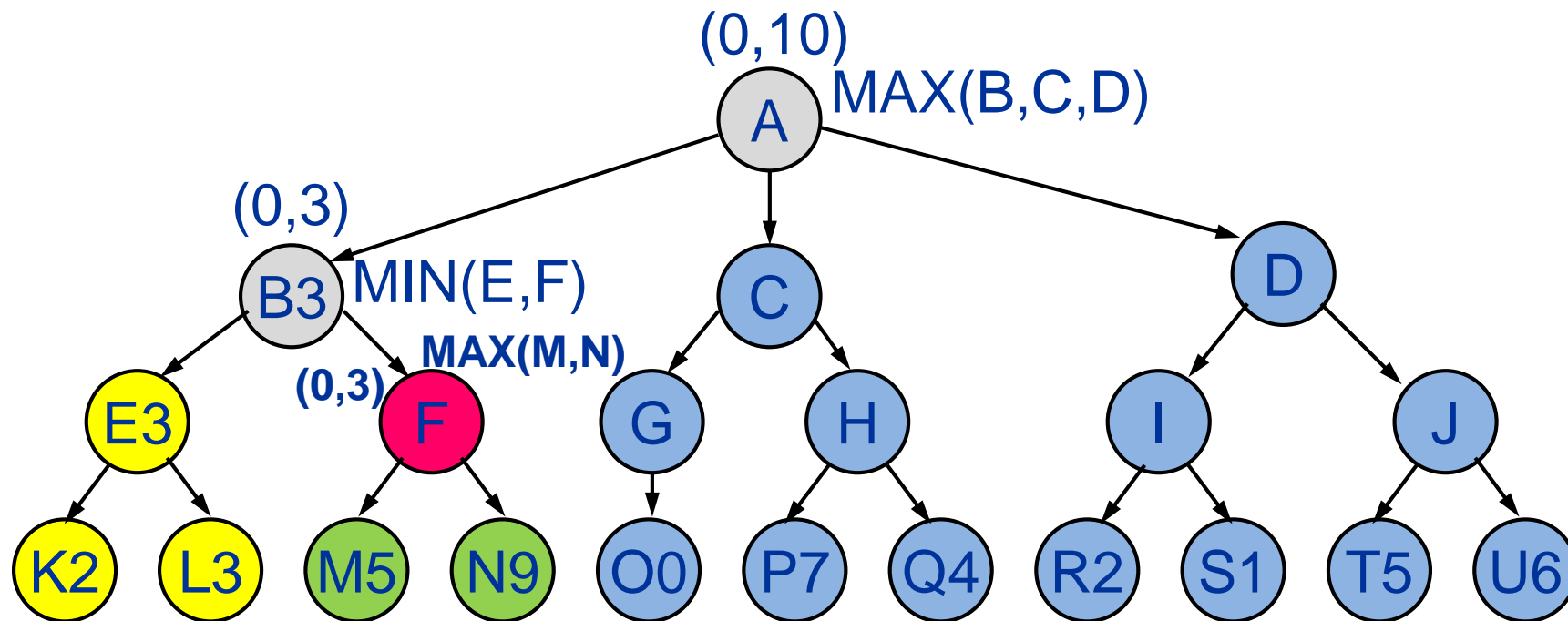
α - β odsecanje (ilustracija)



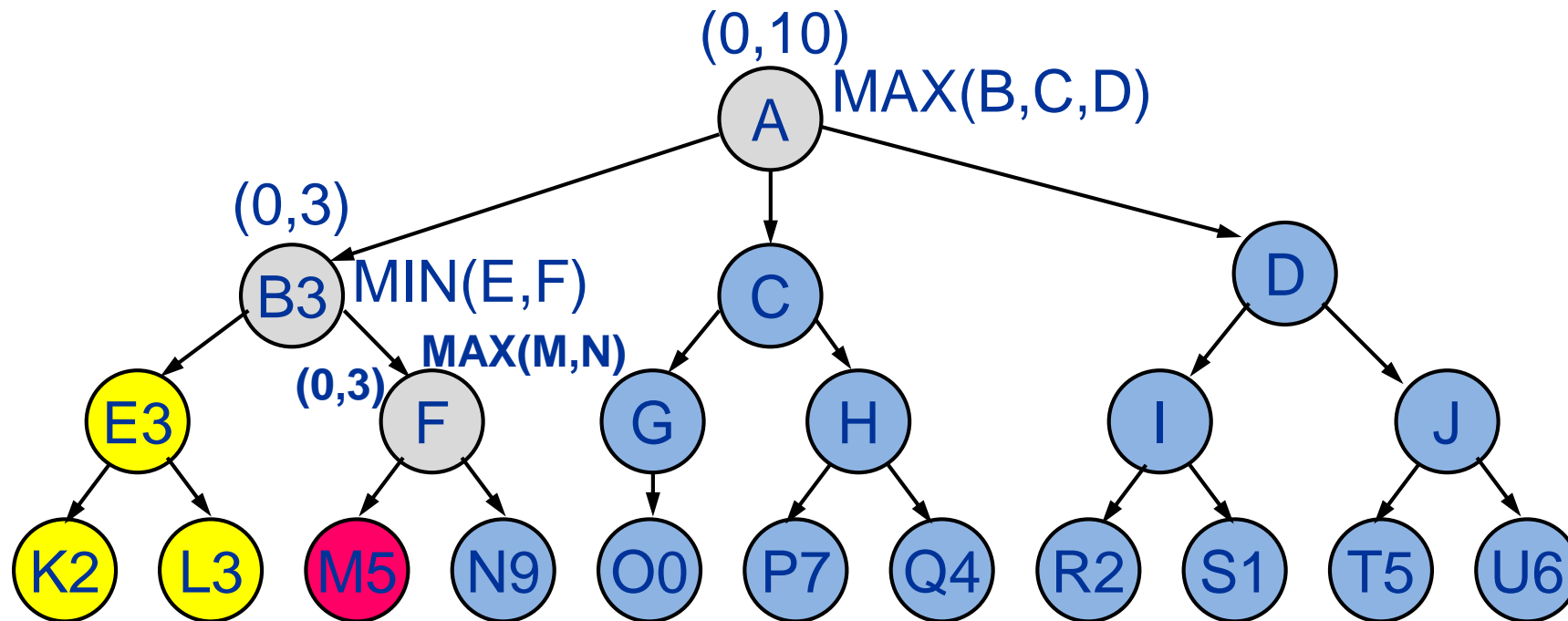
α - β odsecanje (ilustracija)



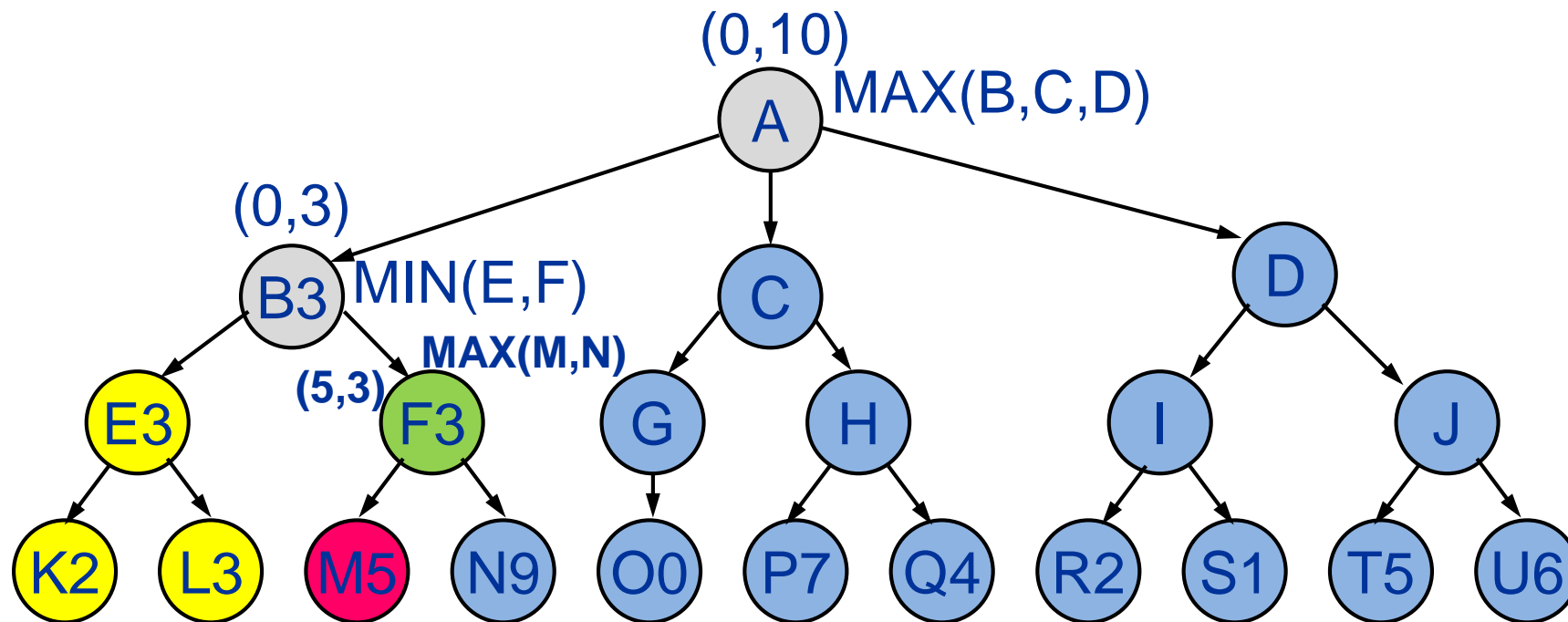
α - β odsecanje (ilustracija)



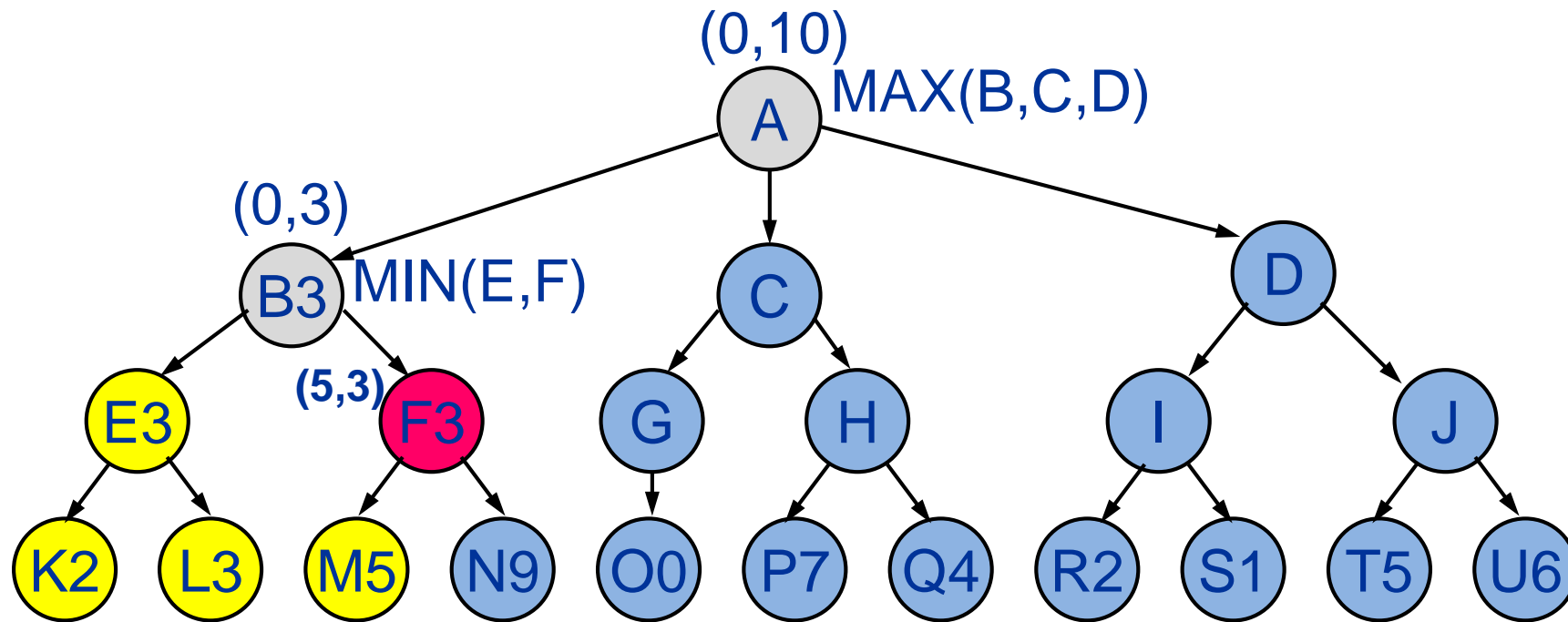
α - β odsecanje (ilustracija)



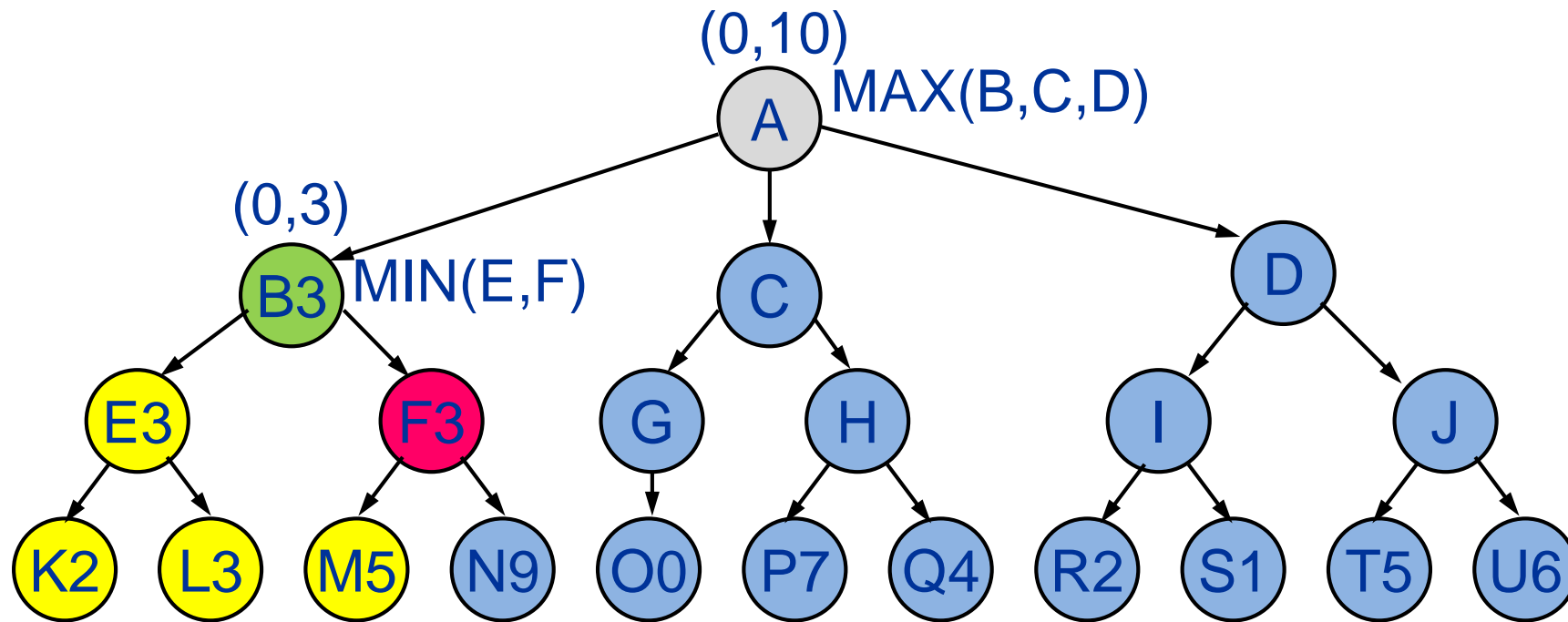
α - β odsecanje (ilustracija)



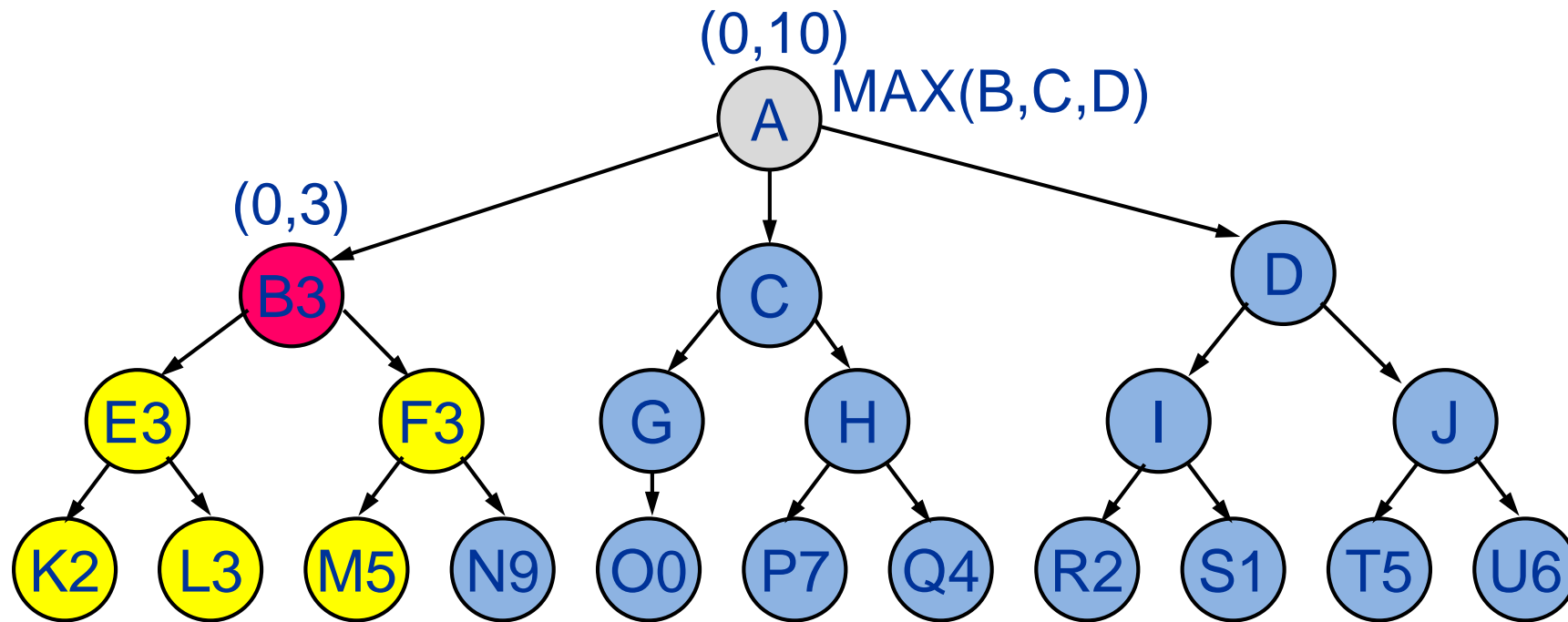
α - β odsecanje (ilustracija)



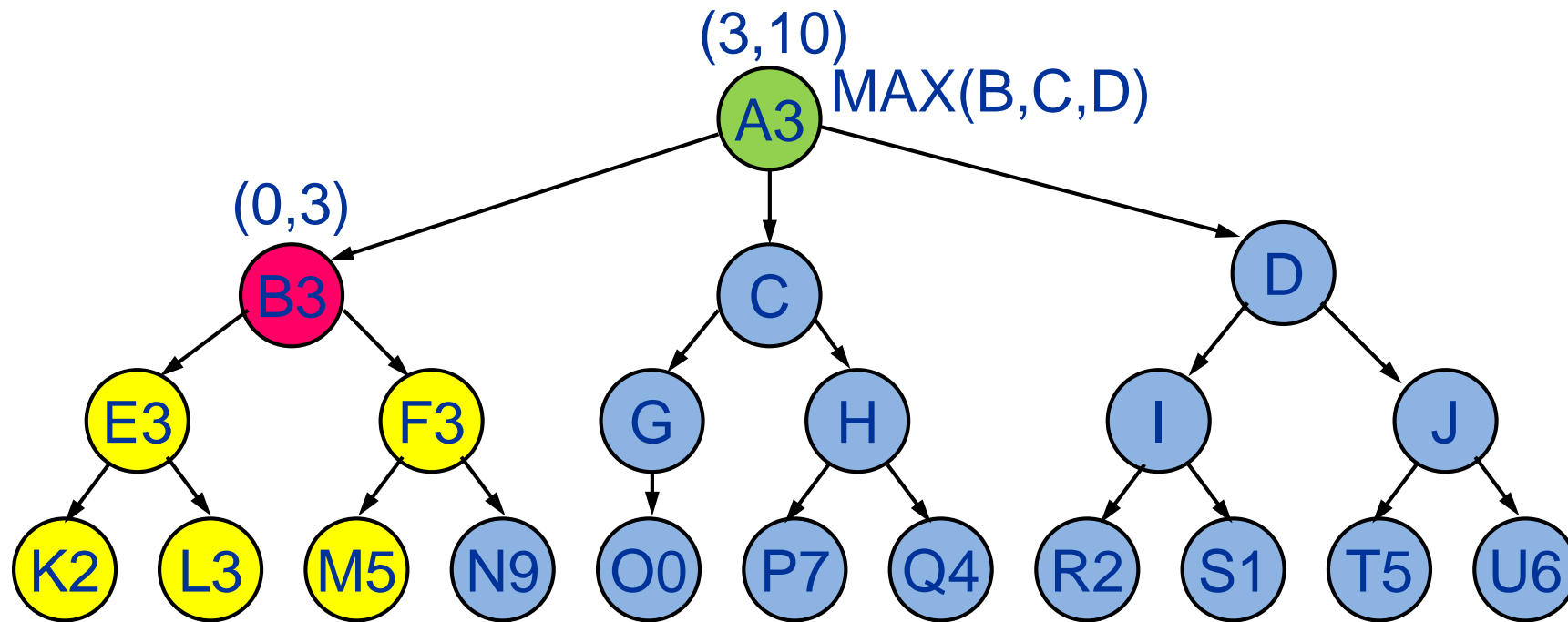
α - β odsecanje (ilustracija)



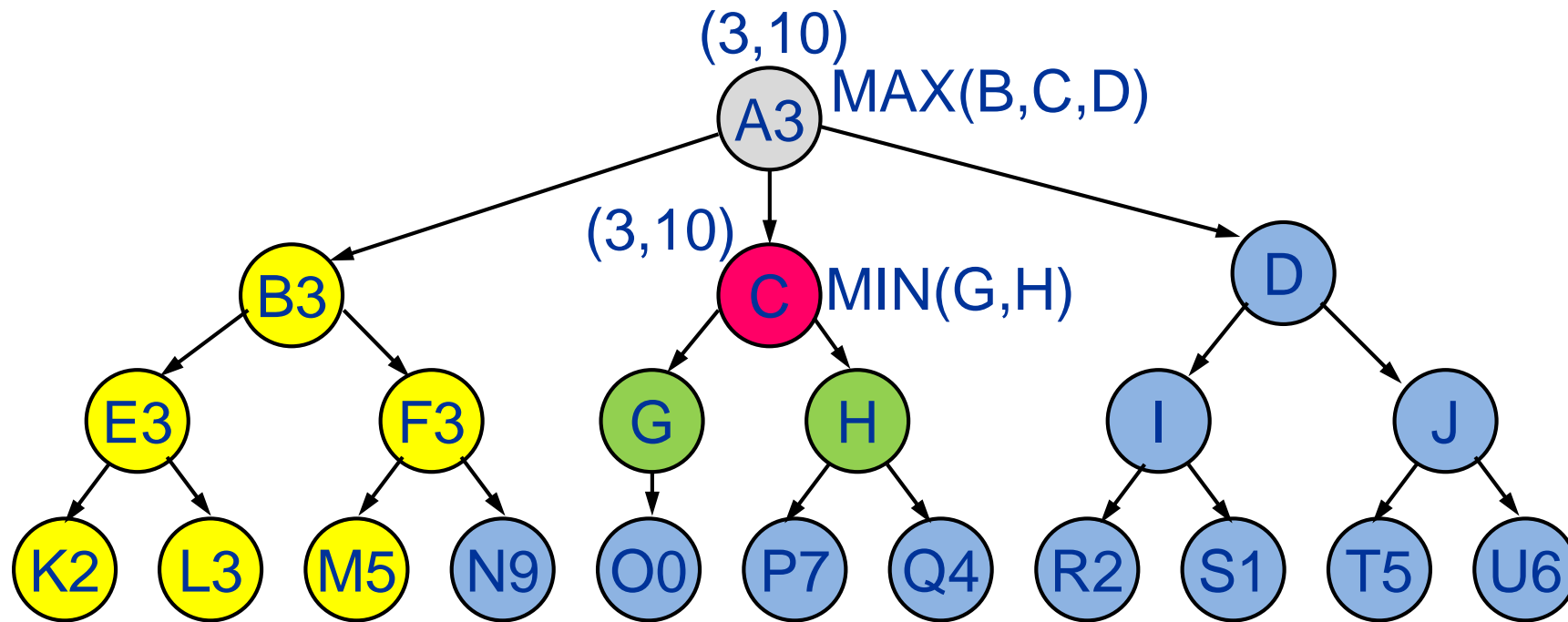
α - β odsecanje (ilustracija)



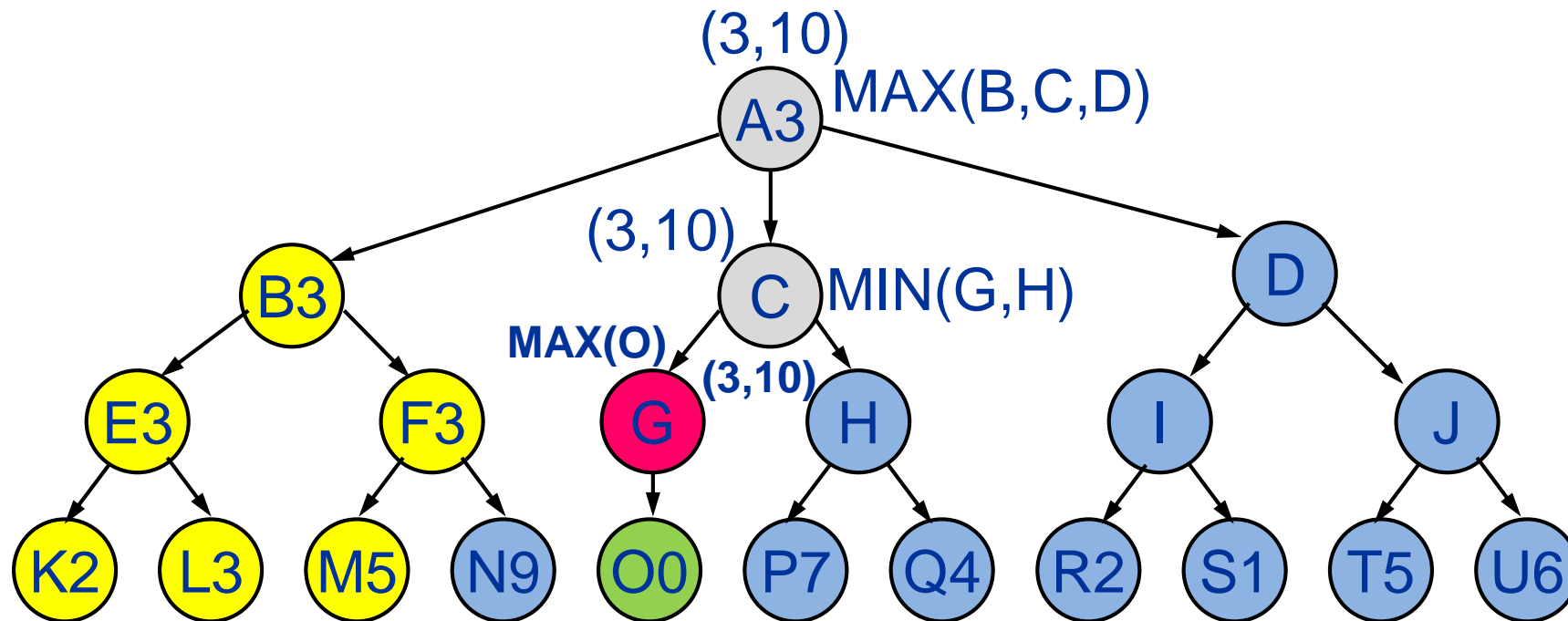
α - β odsecanje (ilustracija)



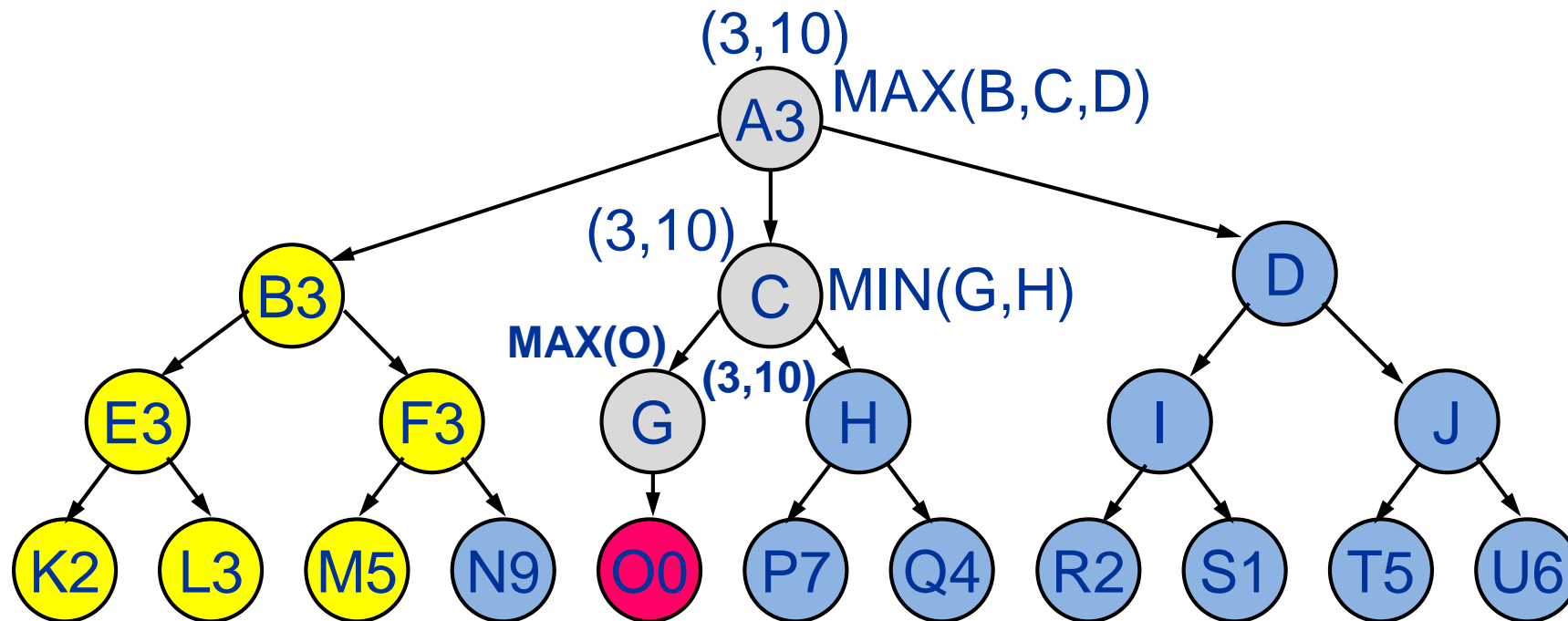
α - β odsecanje (ilustracija)



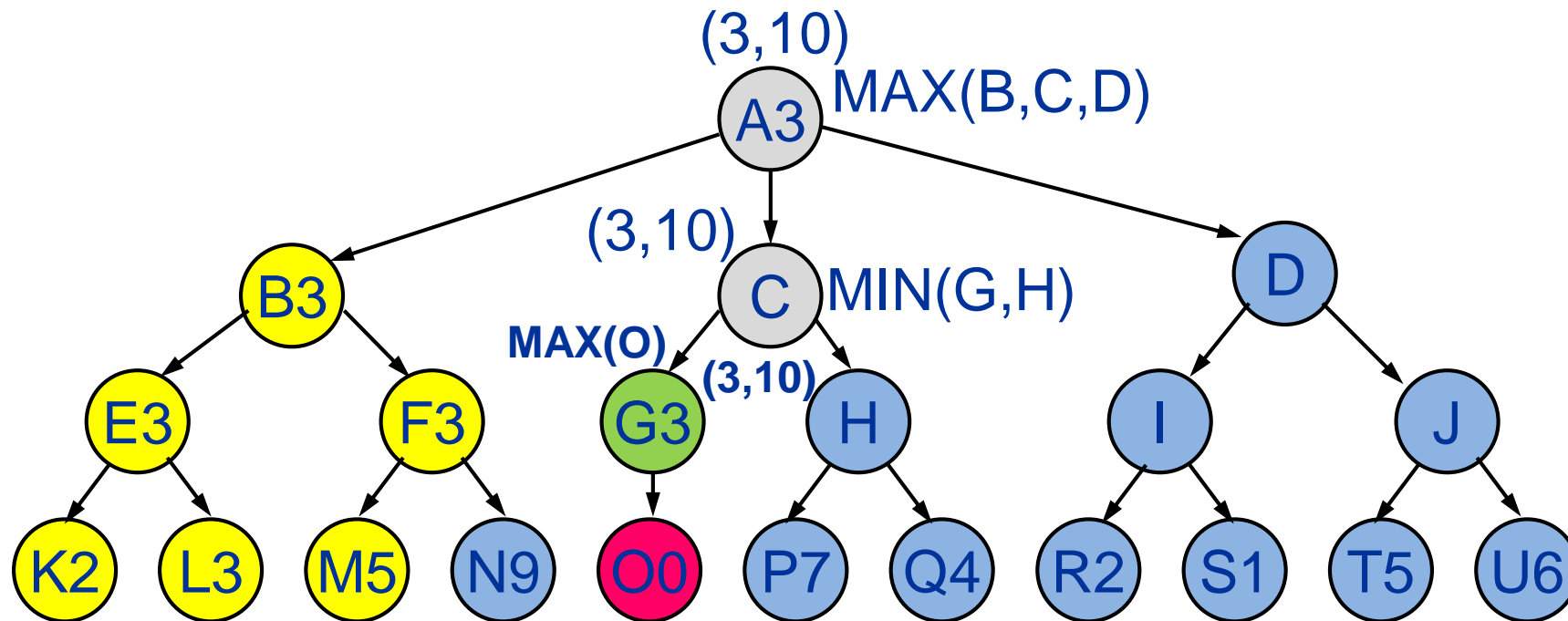
α - β odsecanje (ilustracija)



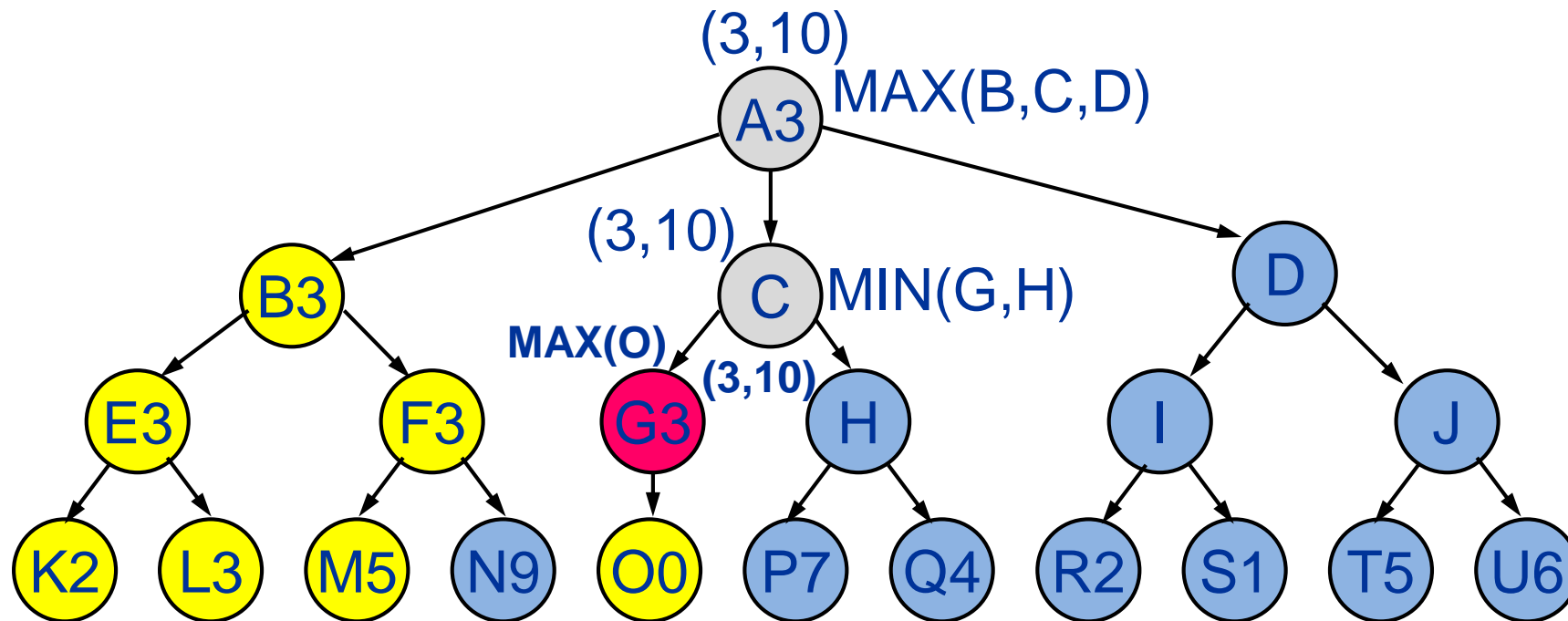
α - β odsecanje (ilustracija)



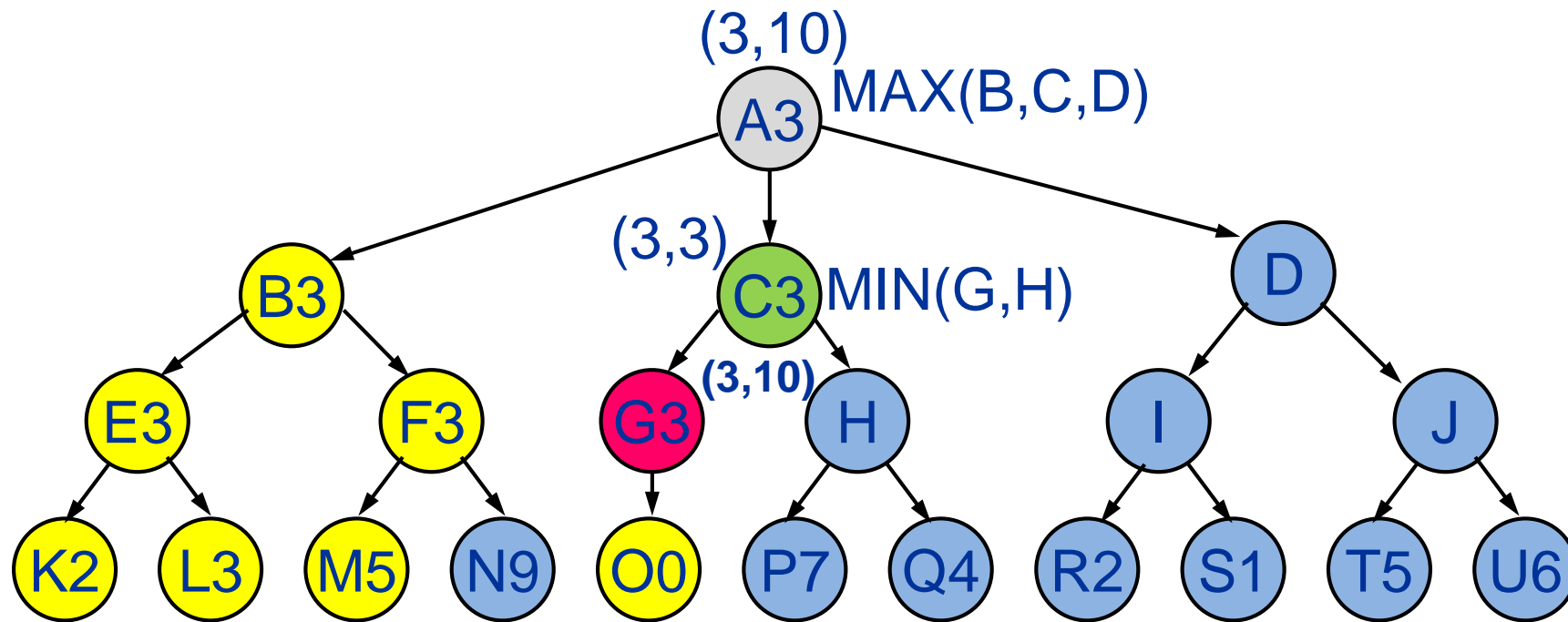
α - β odsecanje (ilustracija)



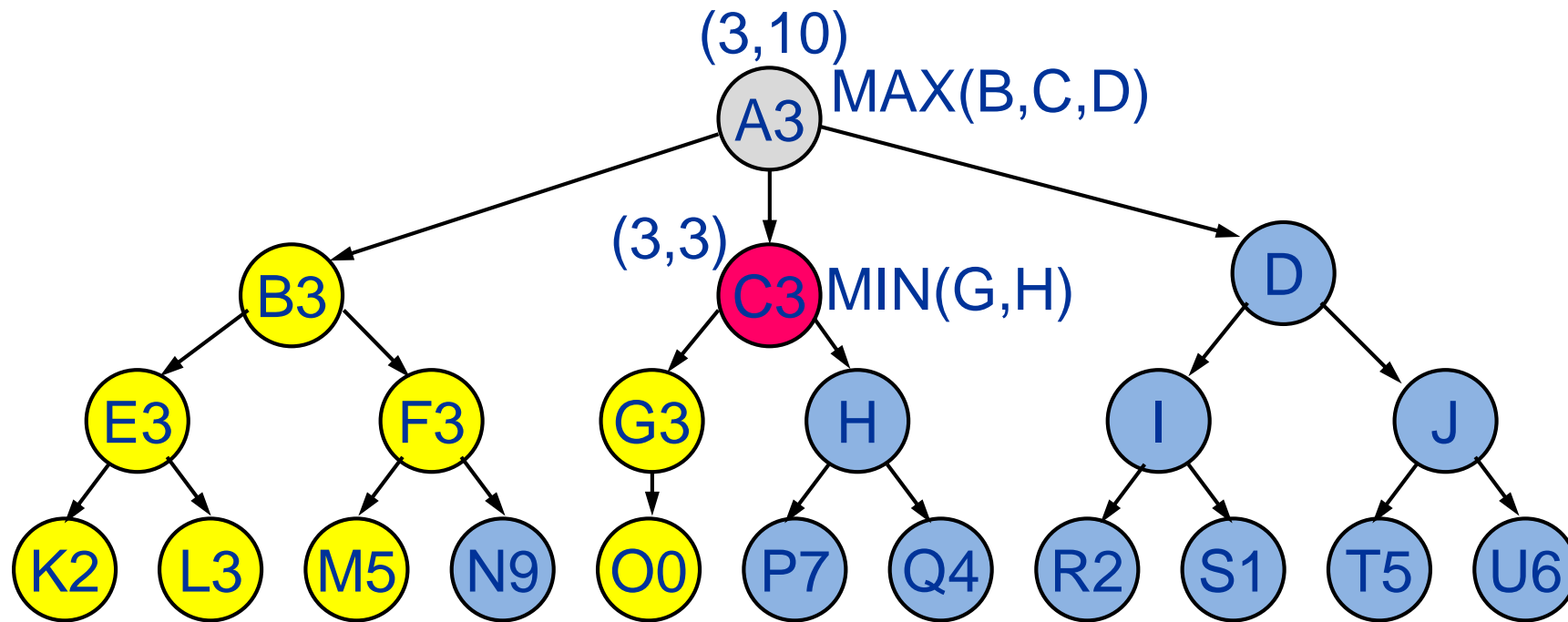
α - β odsecanje (ilustracija)



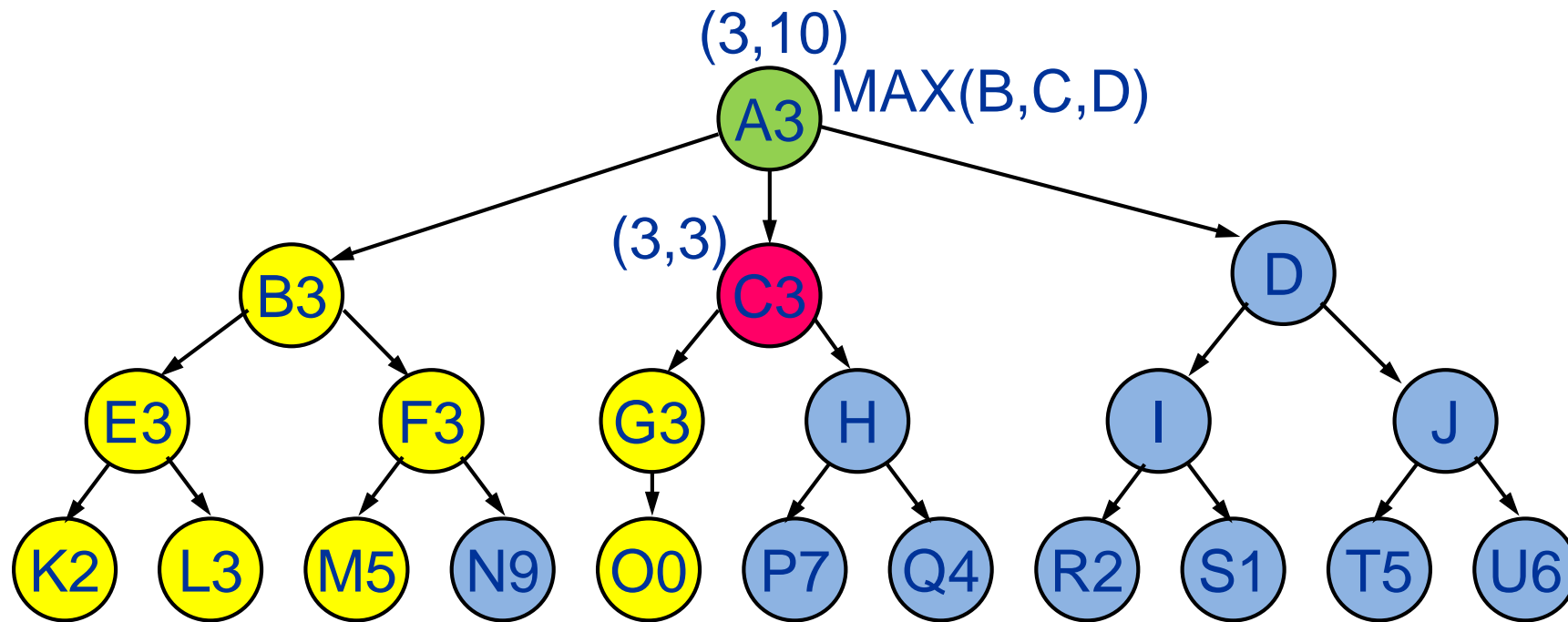
α - β odsecanje (ilustracija)



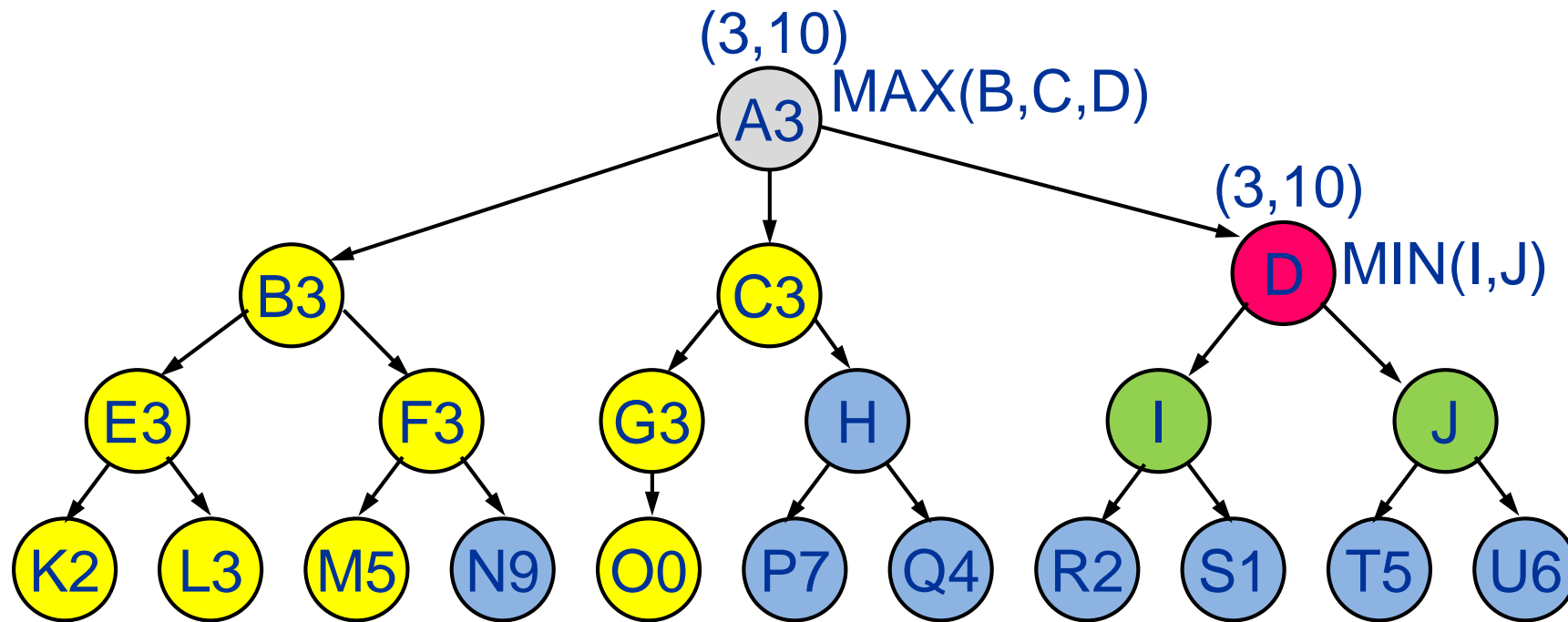
α - β odsecanje (ilustracija)



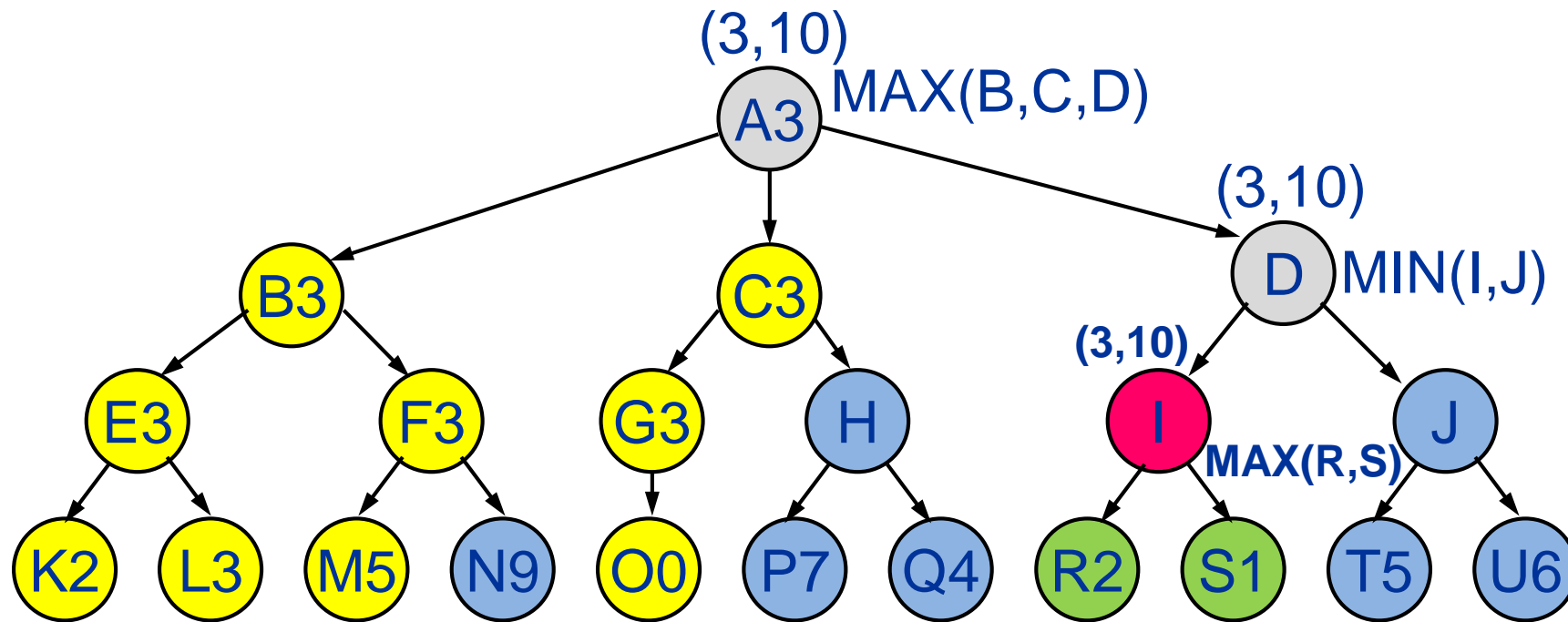
α - β odsecanje (ilustracija)



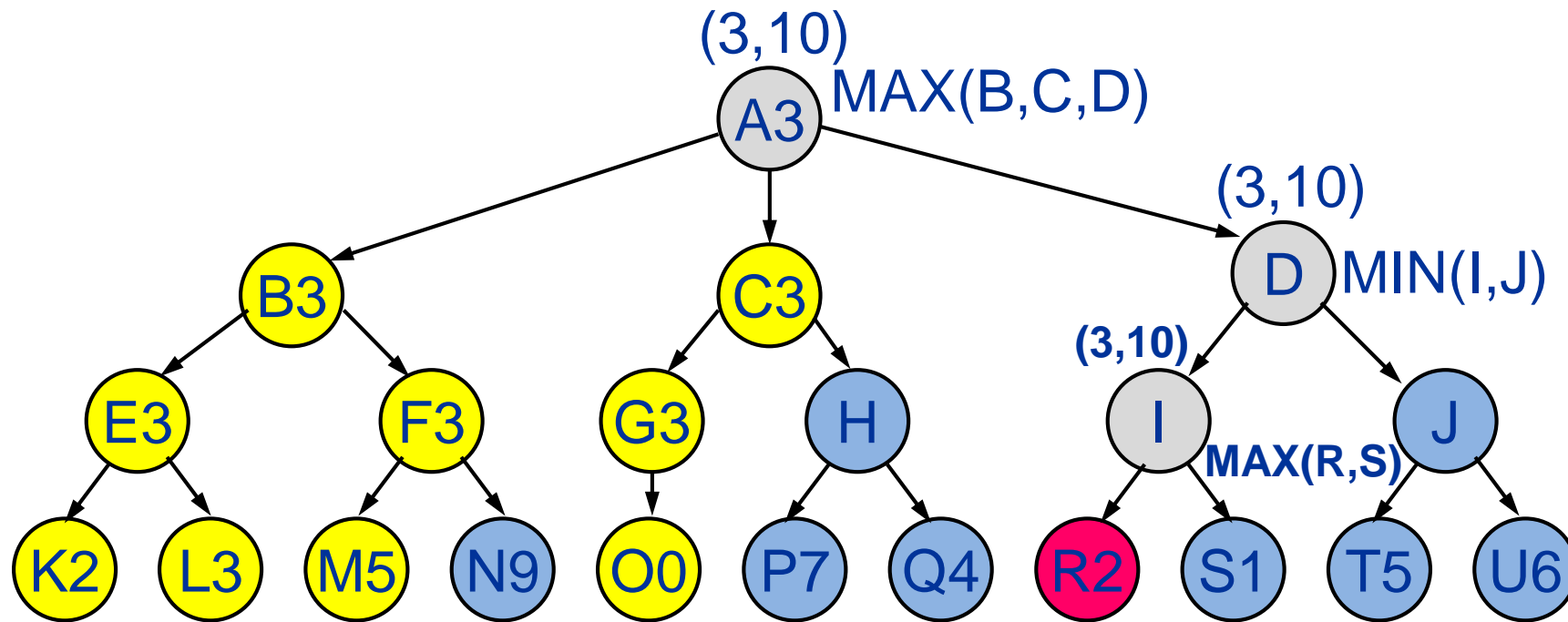
α - β odsecanje (ilustracija)



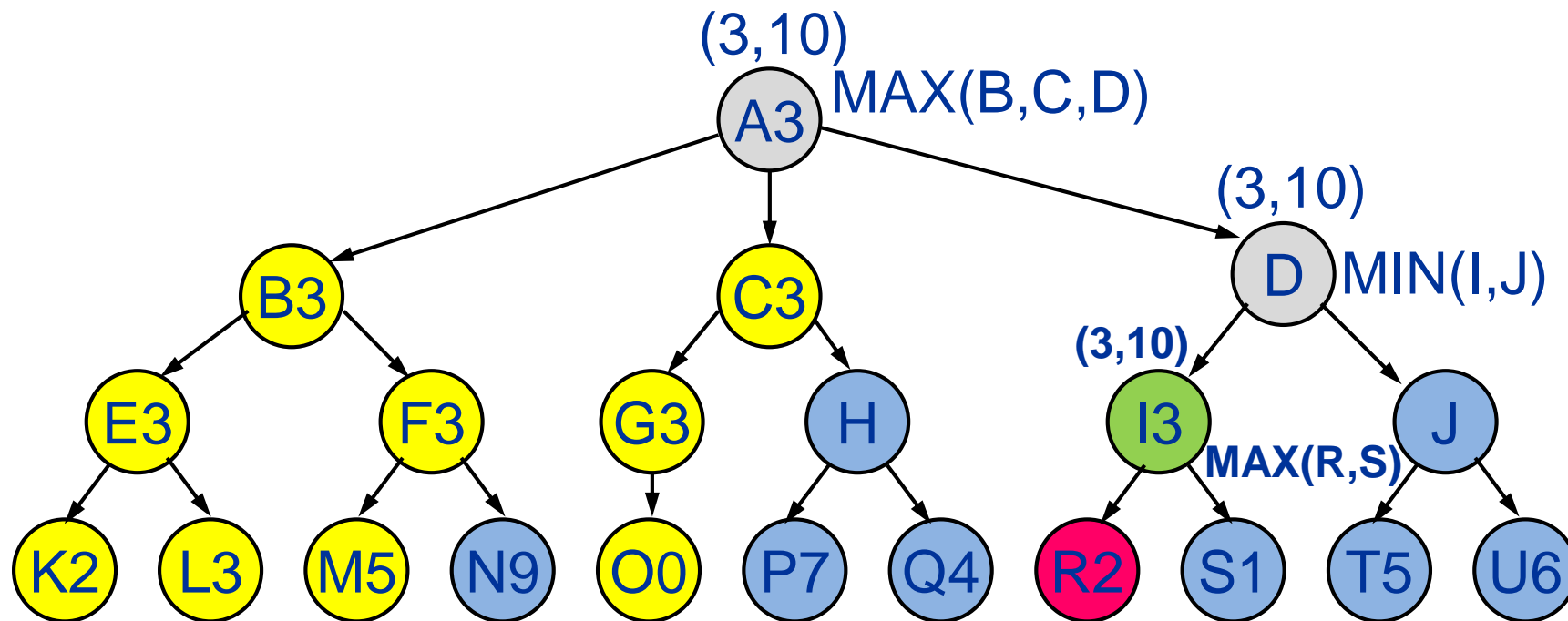
α - β odsecanje (ilustracija)



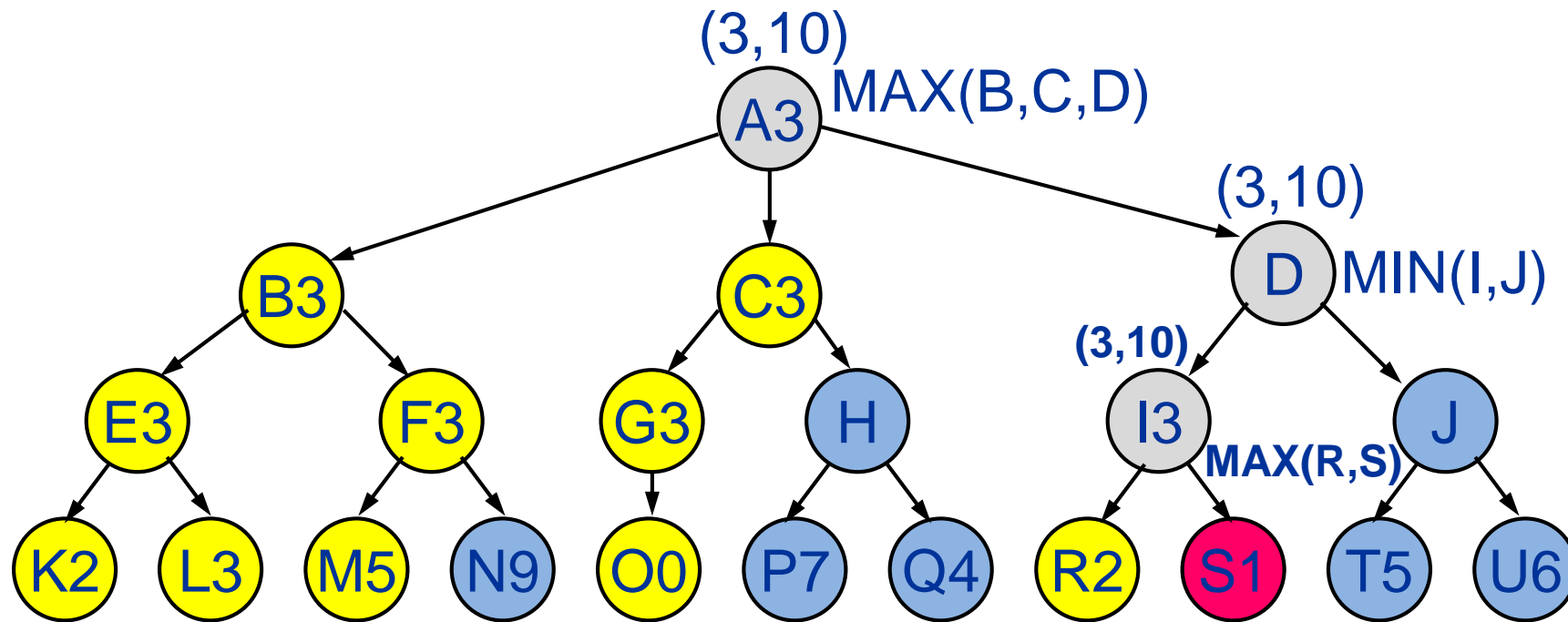
α - β odsecanje (ilustracija)



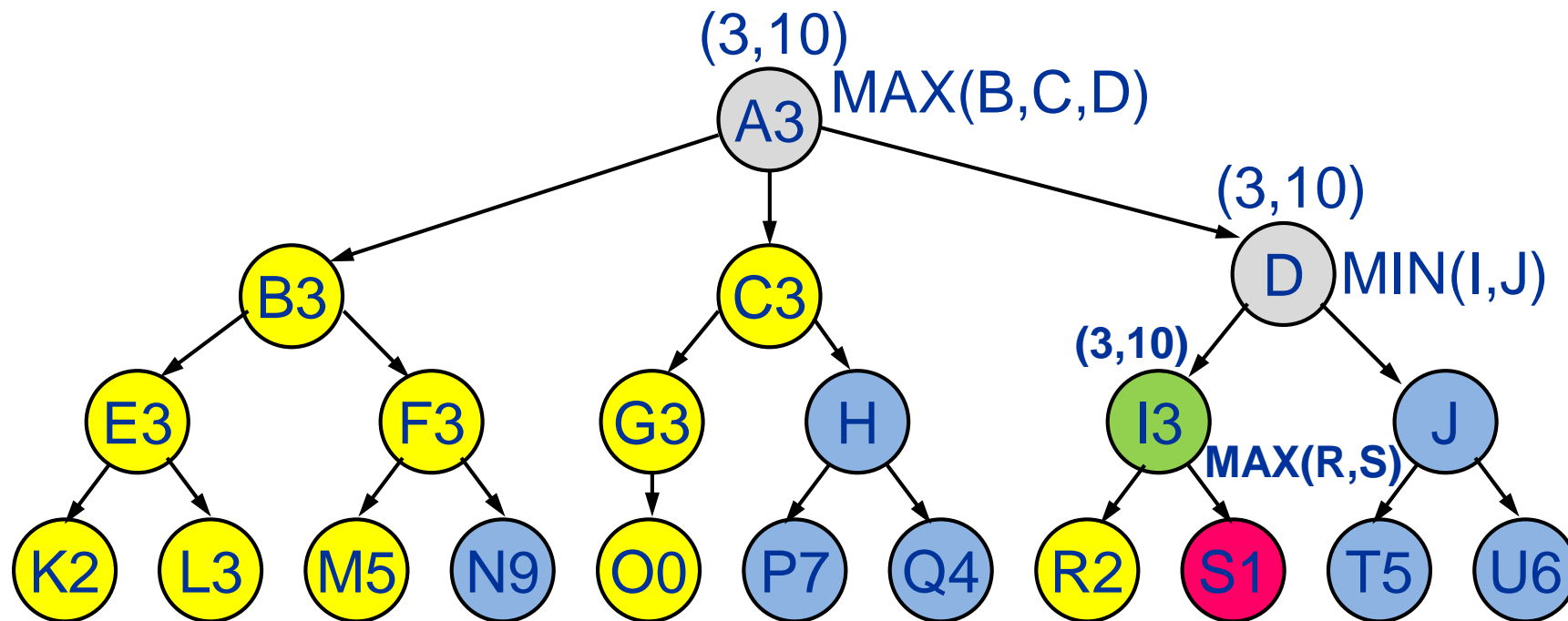
α - β odsecanje (ilustracija)



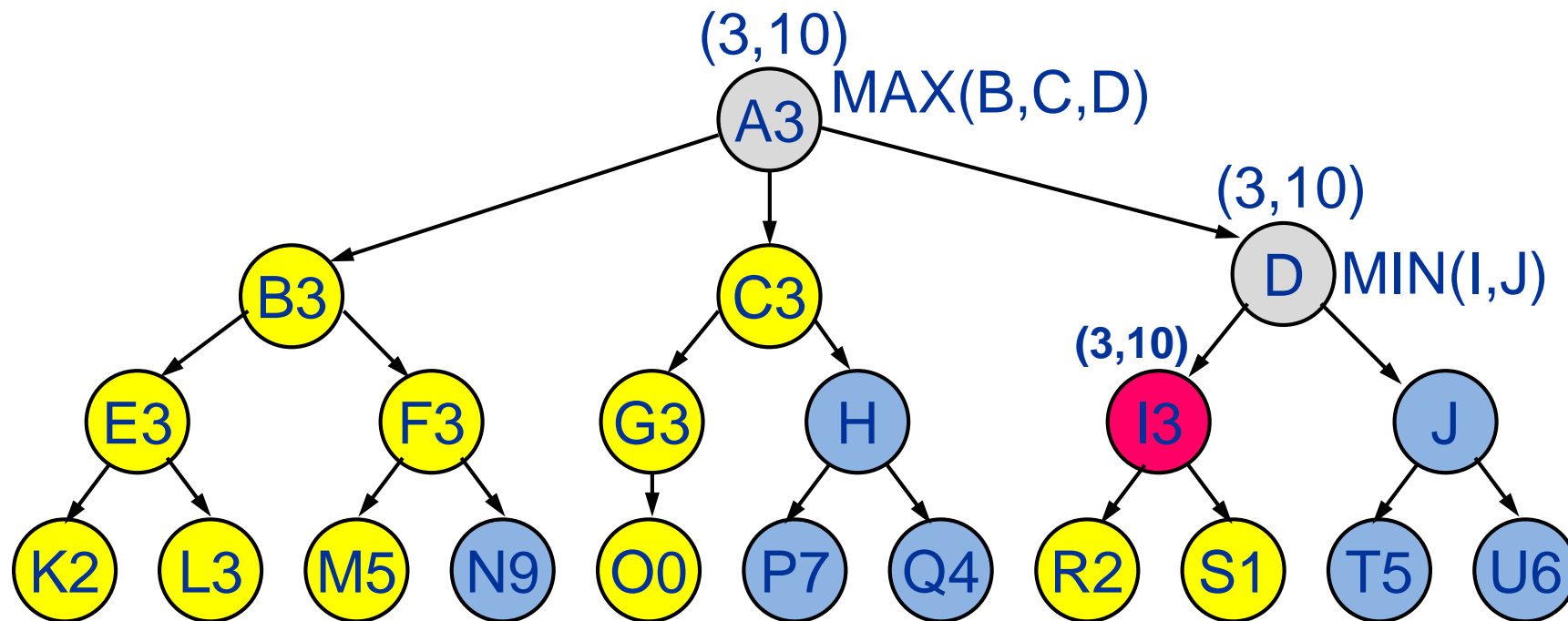
α - β odsecanje (ilustracija)



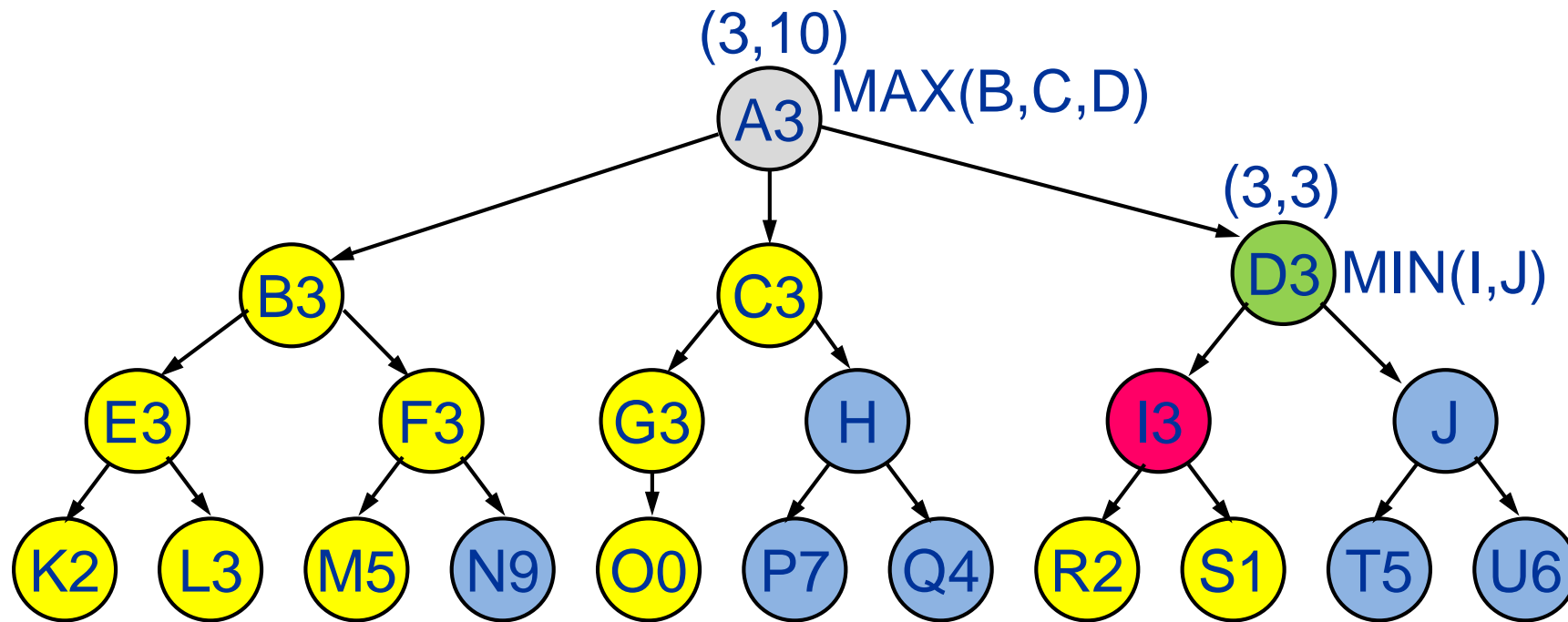
α - β odsecanje (ilustracija)



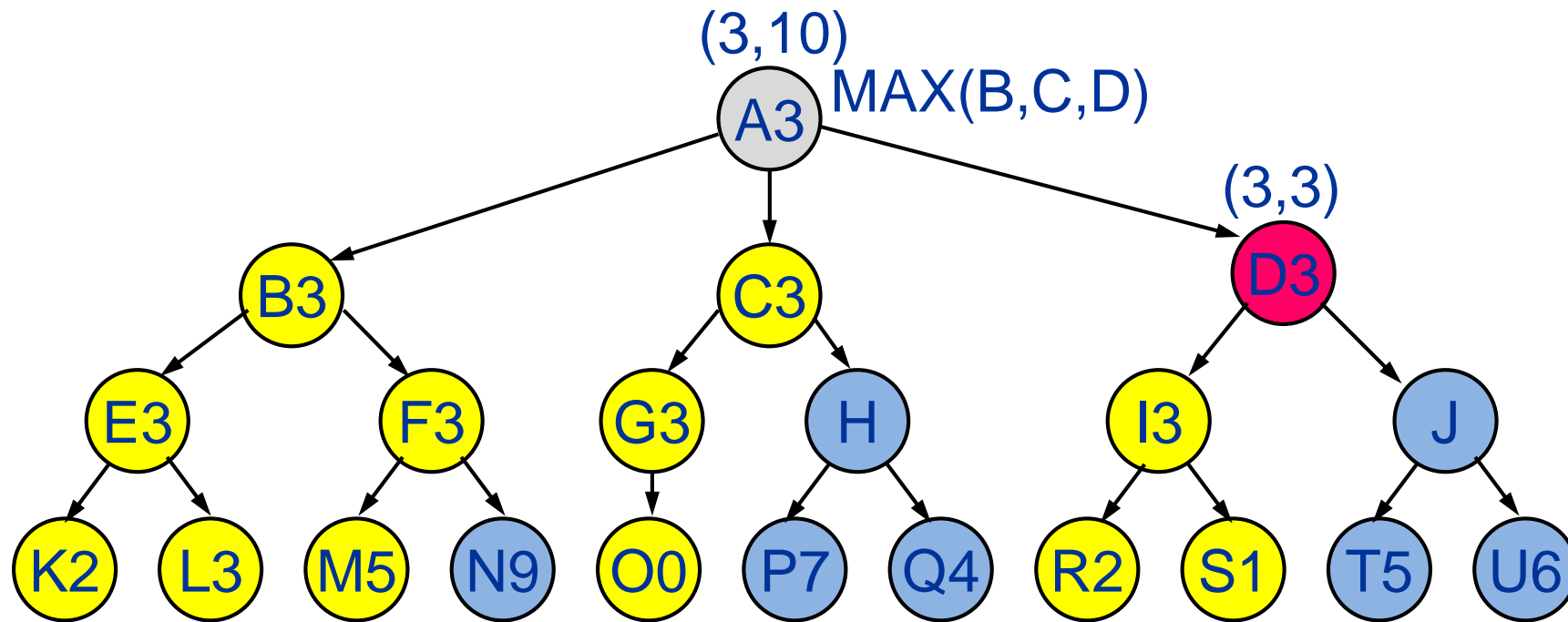
α - β odsecanje (ilustracija)



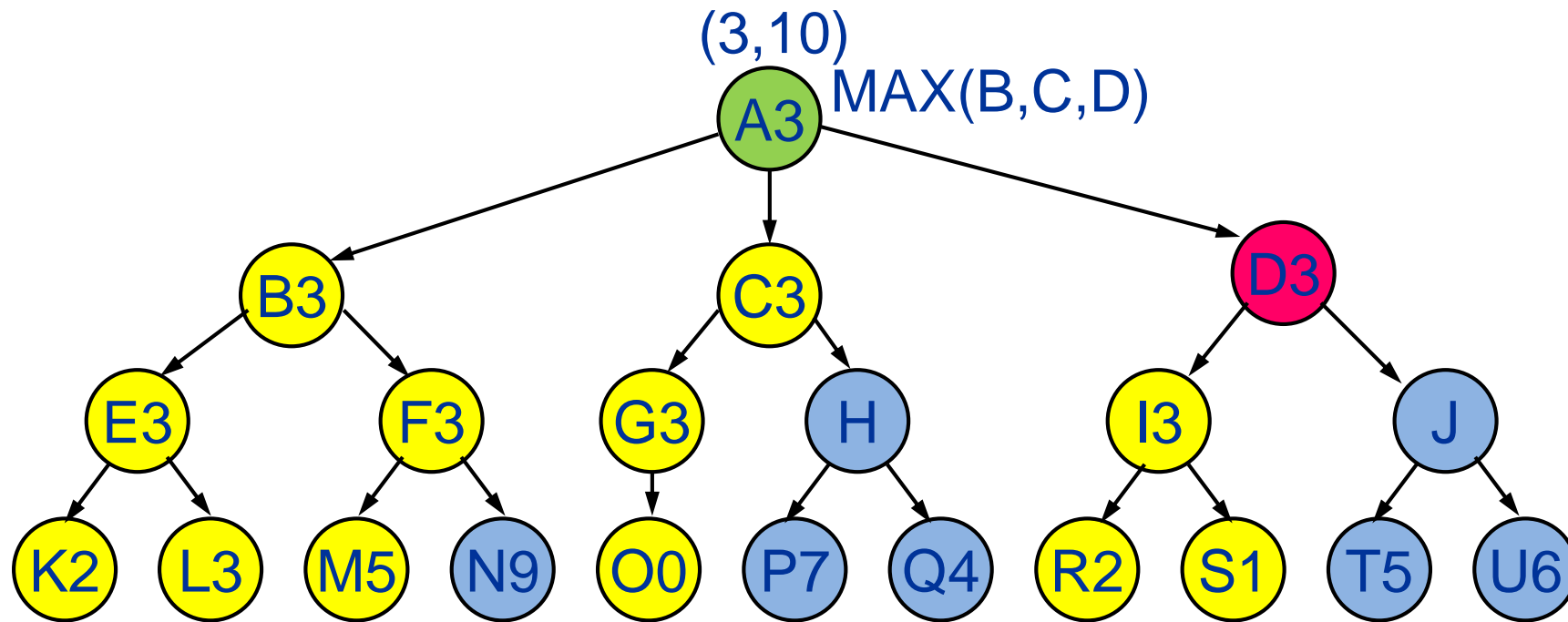
α - β odsecanje (ilustracija)



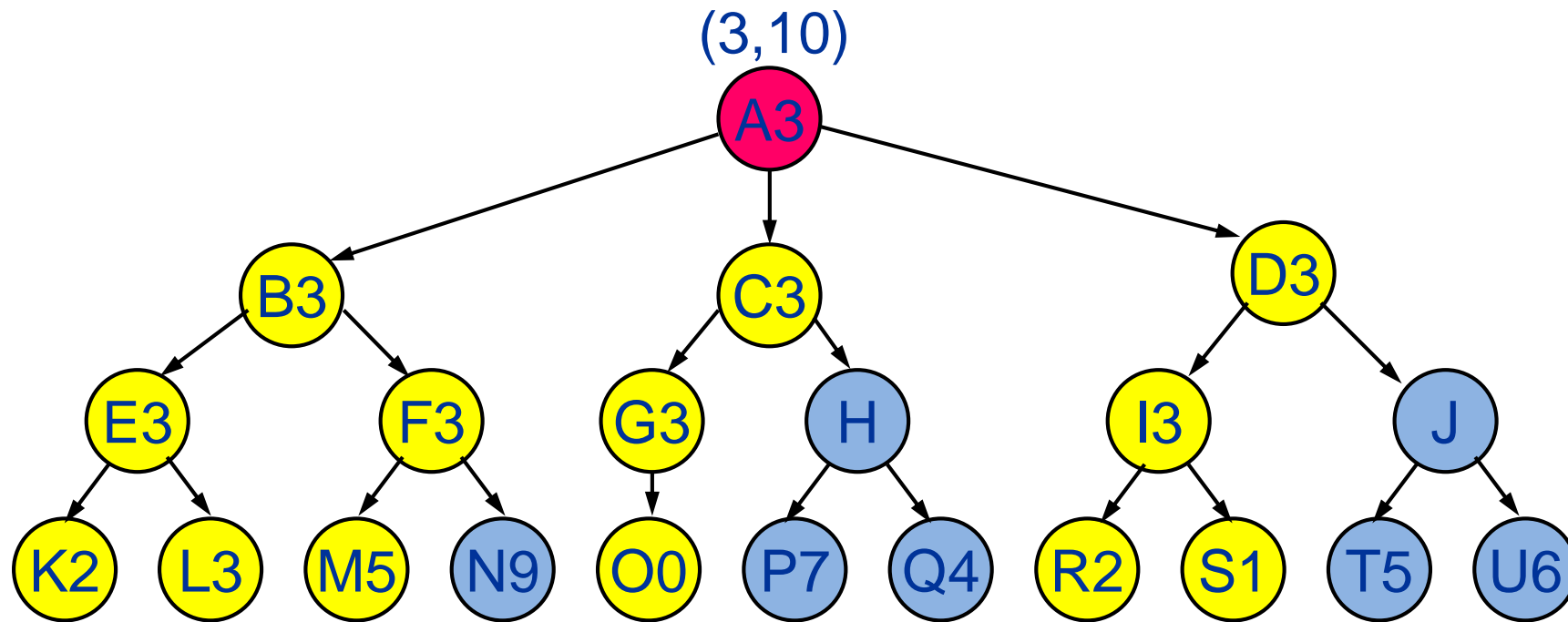
α - β odsecanje (ilustracija)



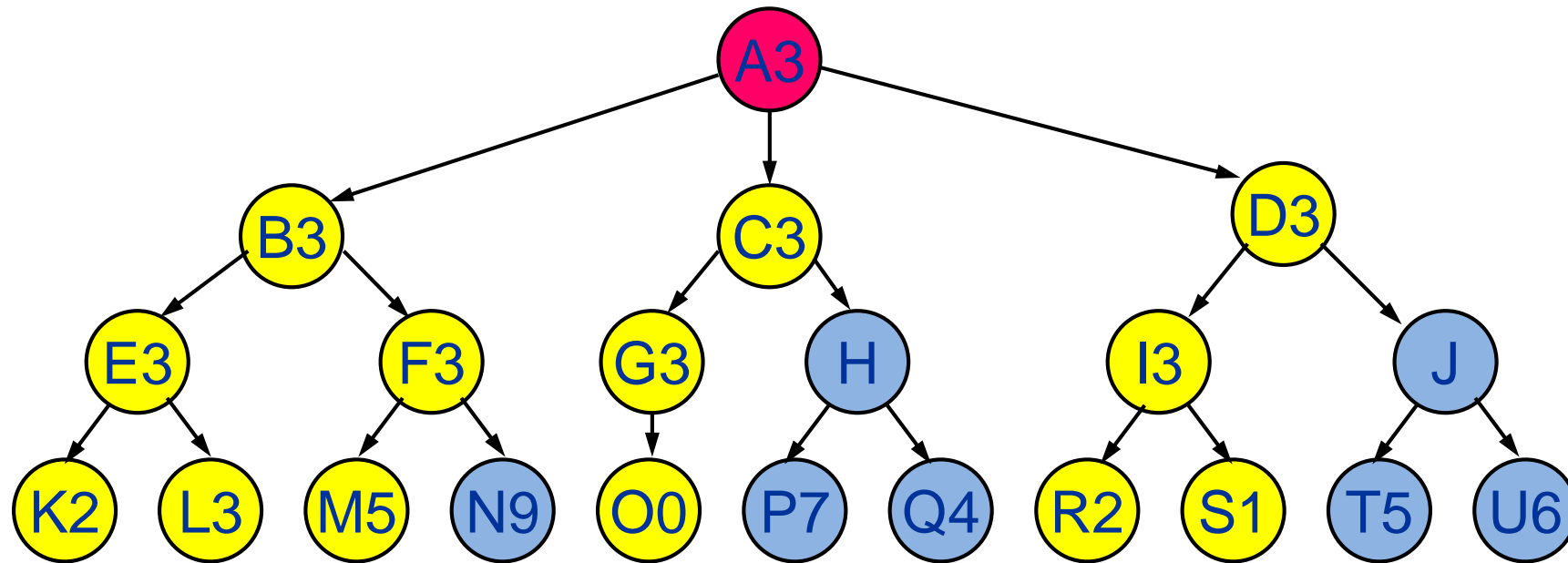
α - β odsecanje (ilustracija)



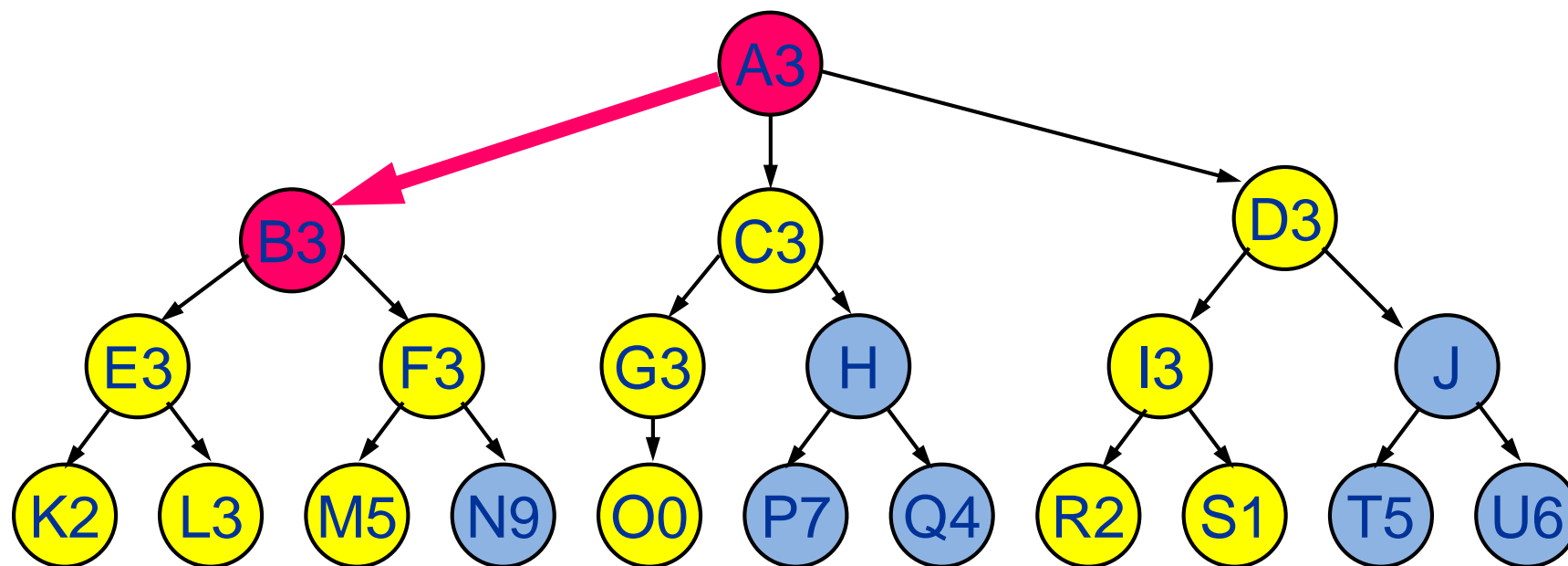
α - β odsecanje (ilustracija)



α - β odsecanje (ilustracija)



α - β odsecanje (ilustracija)



Najbolje je odigrati potez koji igru prevodi iz stanja A u stanje B.

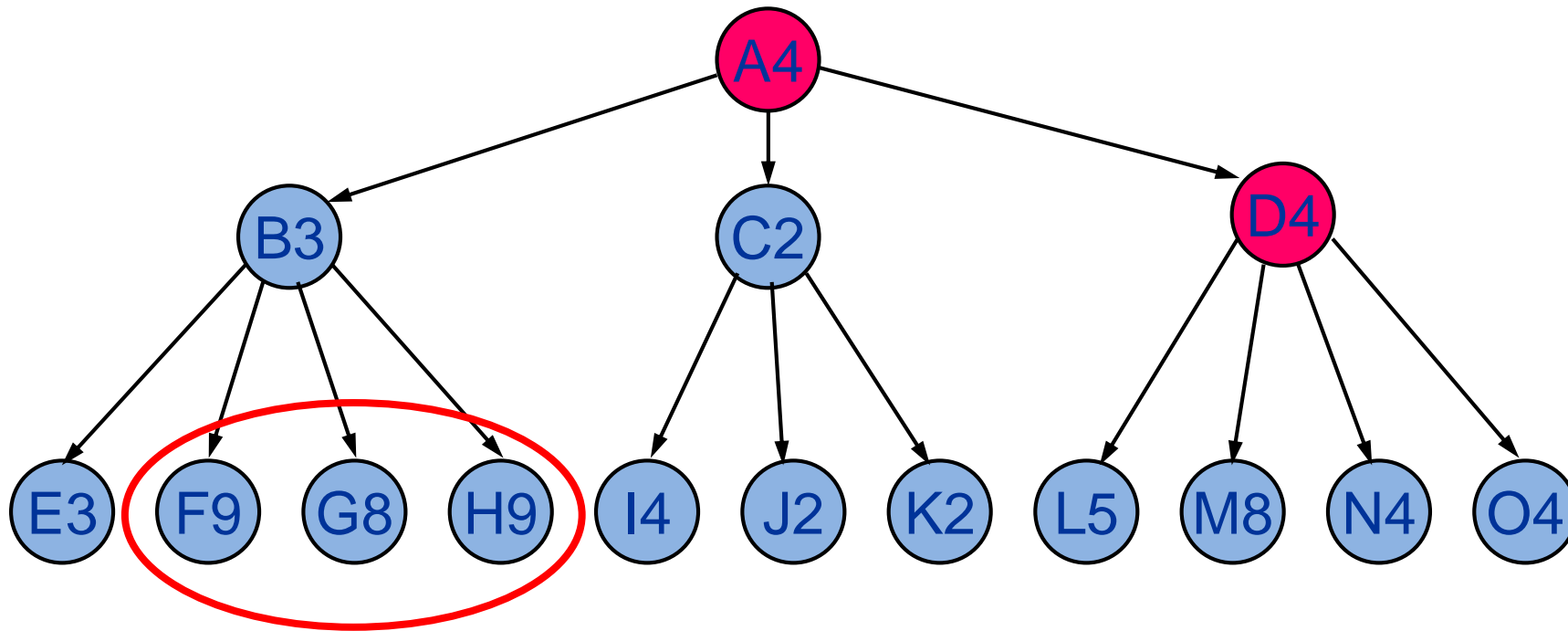


α - β odsecanje – Zaključak

- ▶ Standardni Min-Max prolazi kroz celo stablo, tj. kroz 21 čvor.
- ▶ Min-Max sa α - β odsecanjem ne prolazi kroz celo stablo, već kroz 14 čvorova.
- ▶ Jedna trećina čvorova (7) se ne obilazi, tj. efikasnost je veća za 33%.
- ▶ Ako se uzme da je **b** prosečan broj stanja sledbenika čvorova u grafu i **d** broj nivoa za koje se obavlja Min-Max, složenost algoritama je:
- ▶ **$O(b^d)$** – standardni Min-Max
- ▶ **$O(b^{d/2})$** – Min-Max sa α - β odsecanjem
- ▶ α - β odsecanje omogućuje obradu stabla duplo veće visine za isto vreme ili isti utrošak radne memorije.



Nedostatak Min-Max algoritma



Primer rešavanja problema – Min-Max

Iks-Oks

Iks-Oks

- ▶ Dva igrača (X i O) koji naizmenično postavljaju svoj simbol na tablu 3x3
- ▶ Pobednik je igrač koji prvi spoji 3 polja istog znaka horizontalno, vertikalno ili dijagonalno
- ▶ Ukoliko se ne povežu tri simbola, a ispuni tabla onda nema pobednika



Koraci rešavanje problema

- ▶ Definirati stanja problema
- ▶ Definirati početno stanje
- ▶ Definirati ciljno stanje
- ▶ Definirati prelaze iz jednog stanja u drugo
- ▶ Definirati ograničenja pri prelazu iz jednog stanja u drugo



Predstavljanje stanja

O		X
	X	
O		

X		O
	X	X
O	O	

Predstavljanje stanja:

1. `[[None, None, None], [None, None, None], [None, None, None]]`
2. `[O, None, X, None, X, None, O, None, None]`
3. `[(X, None, O), (None, X, X), (O, O, None)]`

Bilo koji od ovih načina je pogodan



Ciljna stanja

X	O	X
X	O	O
X	X	O

O	X	X
X	X	O
O	X	O

X	X	O
O	X	X
O	O	X

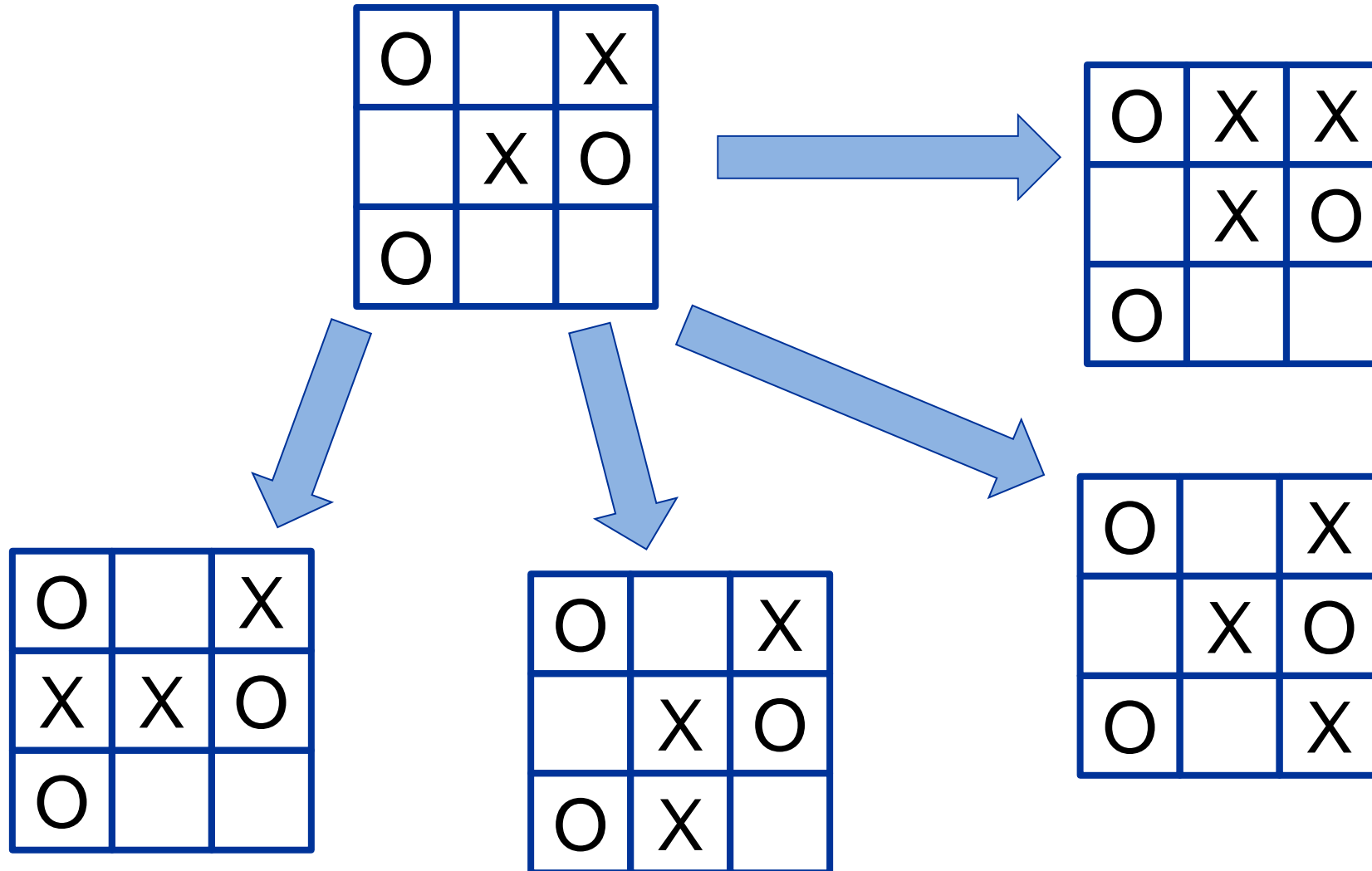
	X	X
X	X	O
O	O	O

O	X	X
O	O	O
X		X

X	X	O
	O	X
O	X	O



Prelaz iz stanja



Ograničenja

- ▶ Ne može se odigrati potez na polje koje nije prazno
- ▶ Ne može se odigrati potez van table
 - ▶ (0, 4)
 - ▶ (5, 2)

```
def validno(pos: tuple):  
    return (pos[0] < 3 and pos[0] >= 0  
            and pos[1] < 3 and pos[1] >= 0)
```



Ispitivanje polja na tabli

```
def ista(pos_x: tuple, pos_y: tuple, stanje):  
    if validno(pos_x) and validno(pos_y):  
        if (stanje[pos_x[0]][pos_x[1]] ==  
            stanje[pos_y[0]][pos_y[1]]):  
            return True  
    return False
```

```
def puno(pos, stanje):  
    if validno(pos):  
        if stanje[pos[0]][pos[1]] is not None:  
            return True  
    return False
```



Ispitivanje polja na tabli

```
def ima(pos, igrac, stanje):  
    if validno(pos):  
        if stanje[pos[0]][pos[1]] is igrac:  
            return True  
    return False  
  
# Sign funkcija koja će nam biti potrebna za procenu stanja  
def sign(val):  
    if val < 0:  
        return -1  
    elif val > 0:  
        return 1  
    else:  
        return 0
```



Određivanje mogućih poteza

```
def slobodna_polja(stanje):  
    for i in range(0, 3):  
        for j in range(0, 3):  
            if not puno((i, j), stanje):  
                yield (i, j)
```

```
def nova_stanja(stanje):  
    return list(slobodna_polja(stanje))
```



Procena po horizontali

```
def oceni_hor(stanje):  
    count = 0  
    for i in range(0, 3):  
        temp = 0  
        for j in range(0, 3):  
            if ima((i, j), X, stanje):  
                temp += 1  
            if ima((i, j), 0, stanje):  
                temp -= 1  
        count += temp  
    return count
```



Procena po vertikali

```
def oceni_ver(stanje):  
    count = 0  
    for i in range(0, 3):  
        temp = 0  
        for j in range(0, 3):  
            if ima((j, i), X, stanje):  
                temp += 1  
            if ima((j, i), 0, stanje):  
                temp -= 1  
        count += temp  
    return count
```



Procena po dijagonali

```
def oceni_diag(stanje):  
    count = 0  
    for i in range(0, 3):  
        if ima((i, i), X, stanje):  
            count += 1  
        if ima((i, i), 0, stanje):  
            count -= 1  
        if ima((i, 2 - i), X, stanje):  
            count += 1  
        if ima((i, 2 - i), 0, stanje):  
            count -= 1  
    return count
```



Ispitivanje kraja igre

```
def kraj(stanje):
    temp_diag_g = 0
    temp_diag_s = 0
    for i in range(0, 3):
        temp_hor = 0
        temp_ver = 0
        for j in range(0, 3):
            if ima((i, j), X, stanje):
                temp_hor += 1
            if ima((i, j), 0, stanje):
                temp_hor -= 1
            if ima((j, i), X, stanje):
                temp_ver += 1
            if ima((j, i), 0, stanje):
                temp_ver -= 1
```

```
        if ima((i, i), X, stanje):
            temp_diag_g += 1
        if ima((i, i), 0, stanje):
            temp_diag_g -= 1
        if ima((i, 2 - i), X, stanje):
            temp_diag_s += 1
        if ima((i, 2 - i), 0, stanje):
            temp_diag_s -= 1

        if abs(temp_hor) == 3:
            return sign(temp_hor) * 10
        if abs(temp_ver) == 3:
            return sign(temp_ver) * 10

    if abs(temp_diag_g) == 3:
        return sign(temp_diag_g) * 10
    if abs(temp_diag_s) == 3:
        return sign(temp_diag_s) * 10

    return 0
```



Ispitivanje kraja igre

```
def full(stanje):  
    count = 0  
  
    for i in range(0, 3):  
        for j in range(0, 3):  
            if puno((i, j), stanje):  
                count += 1  
  
    return True if count == 9 else False
```



Ocena heuristike

```
def oceni(stanje):  
    return (oceni_hor(stanje) +  
            oceni_ver(stanje) +  
            oceni_diag(stanje))  
  
def max_stanje(lsv):  
    return max(lsv, key=lambda x: x[1])  
  
def min_stanje(lsv):  
    return min(lsv, key=lambda x: x[1])
```



MinMax algoritam

```
def minimax(stanje, dubina, moj_potez, potez = None):
    if abs(kraj(stanje)) == 10:
        return (potez, kraj(stanje))
    if full(stanje):
        return (potez, 0)

    igrac = X if moj_potez else 0
    fja = max_stanje if moj_potez else min_stanje
    lp = nova_stanja(stanje)

    if dubina == 0:
        return (potez, oceni(stanje))





    if lp is None or len(lp) == 0:
        return (potez, oceni(stanje))

    return fja([minimax(igranj(x, igrac, stanje), dubina - 1, not moj_potez, x if potez is None else potez) for x in lp])
```



Stampanje

```
def print_table(stanje: list[list]):  
    board = [print_repr(x) for y in stanje for x in y]
```

```
    print("""  
|| {} || {} || {} ||  
  
|| {} || {} || {} ||  
  
|| {} || {} || {} ||  
  
""").format(*board))
```

```
def print_repr(symbol):  
    if symbol == X:  
        return X  
    elif symbol == 0:  
        return 0  
    else:  
        return " "
```



Odigravanje poteza

```
def igranj(pos, igrac, stanje):  
    if validno(pos):  
        if not puno(pos, stanje):  
            return [[igrac  
                    if pos[0] == j and pos[1] == i  
                    else stanje[j][i] for i in range(0, 3)]  
                    for j in range(0, 3)]  
    return stanje
```



Pokretanje igre

```
def igra():
    tabla = [[None, None, None], [None, None, None], [None, None, None]]
    igrac = X if input("Uneti igrača (X ili O): ") == "X" else O
    moj = True if igrac == X else False
    print_table(tabla)

    while (kraj(tabla) == 0 and not full(tabla)):
        rez = minimax(tabla, 9, moj)
        naj = rez[0] if type(rez) is tuple else (0, 0)
        tabla = igraj(naj, igrac, tabla)
        print_table(tabla)
        igrac = O if igrac is X else X
        moj = not moj
    pobednik = (X if kraj(tabla) == 10 else
                (O if kraj(tabla) == -10 else
                 "Nerešeno"))
    print(f"Pobednik je: {pobednik}")
```

Pokretanje igre:

igra()

