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 izvodljivo analizirati celo stablo, zato što može da bude preveliko. Rešenje je:
  - Prekidanje traženja – traženje je ograničeno po dubini
  - Uvođenje funkcije za procenu vrednosti stanja na osnovu njegovih osobina
  - Primer funkcije za procenu – linearna težinska funkcija:
    - w1*f1 + w2*f2 + ... + wn*fn

Min-Max algoritam

- Ako je X list stabla traženja:
  - P(X) = 1 => Pera je pobednik
  - P(X) = 0 => Nerešeno je
  - P(X) = -1 => Mara je pobednik
- Ako X nije list stabla traženja:
  - V(X) = max {V(C) | C je potomak X} ako Pera igra sledeći
  - V(X) = min {V(C) | C 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 se koriste kao i ostale promenjive, a vrednost može da se prevede u string:

```
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)`

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
```

Funkcija ***proceni_stanje***

- Funkcija ***proceni_stanje*** određuje koliko konkretno stanje vodi određenog igrača ka pobedi.
- 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
```

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 funkcija 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 ili najmanji.

Funkcija ***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])
```

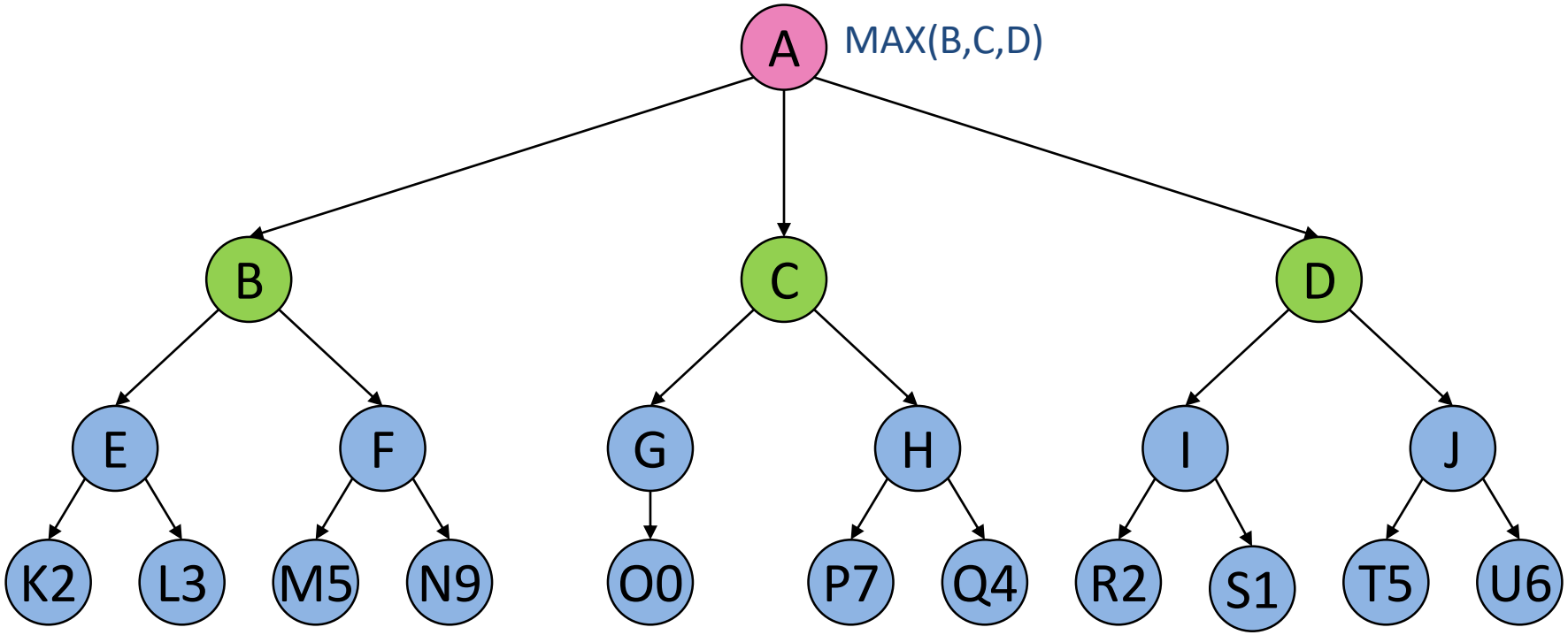
- Kod ove dve funkcije, ključ se koristi da se iz tuple podatka izvuče druga vrednost
- Primer lsv promenjive: `lsv = [(A, 1), (C, 10), (M, 5)]`
  - Povratna vrednost je tuple, koji se sastoji od čvora i heuristike

Funkcija ***minimax***

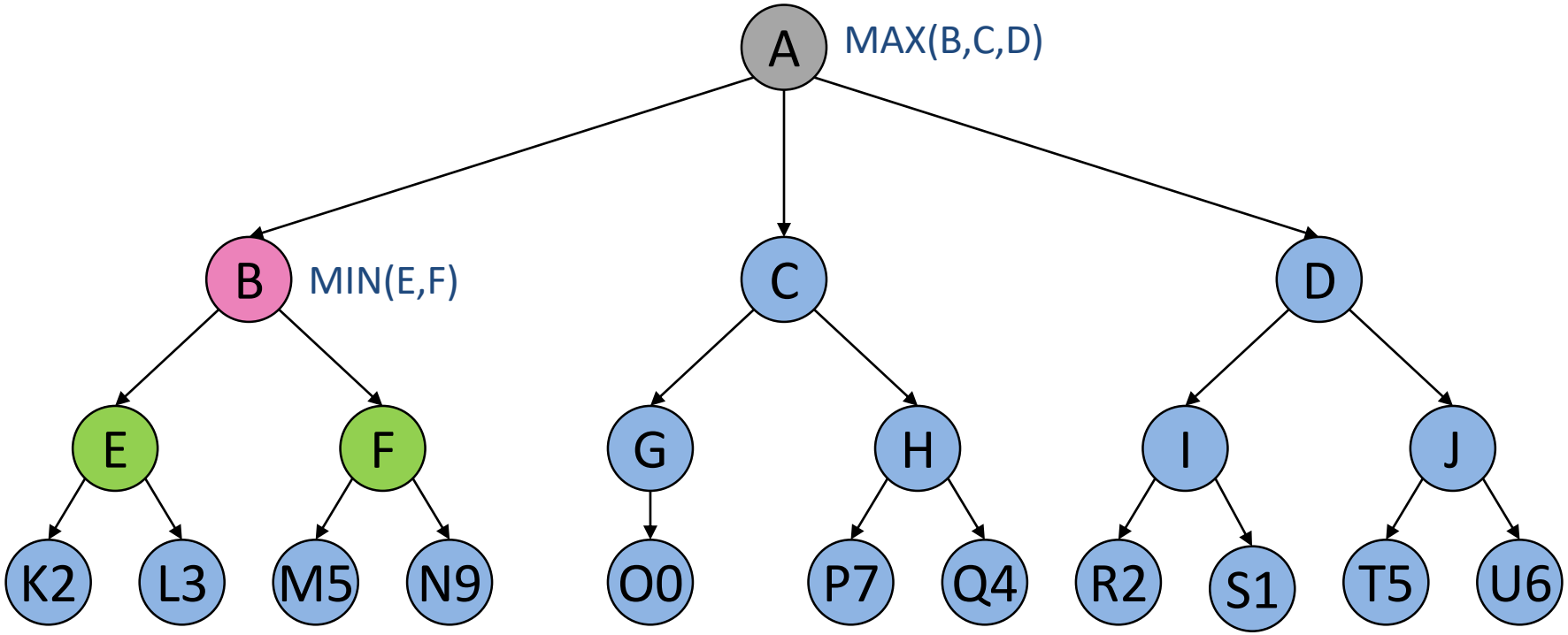
- Glavna funkcija ***minimax*** koja određuje vrednost (funkciju procene) nekog stanja na osnovu vrednosti (funkcije procene) potomaka i igrača koji je na potezu:
  - (Računar – **True**, Igrač – **False**)

```
def minimax(stanje, dubina, moj_potez):
    lista_poteza = nova_stanja(stanje)
    min_max_stanje = max_stanje if moj_potez else min_stanje

    if dubina == 0 or lista_poteza is None:
        return(stanje, proceni_stanje(stanje))

    return min_max_stanje([minimax(x, dubina - 1, not moj_potez) for x in lista_poteza])
```

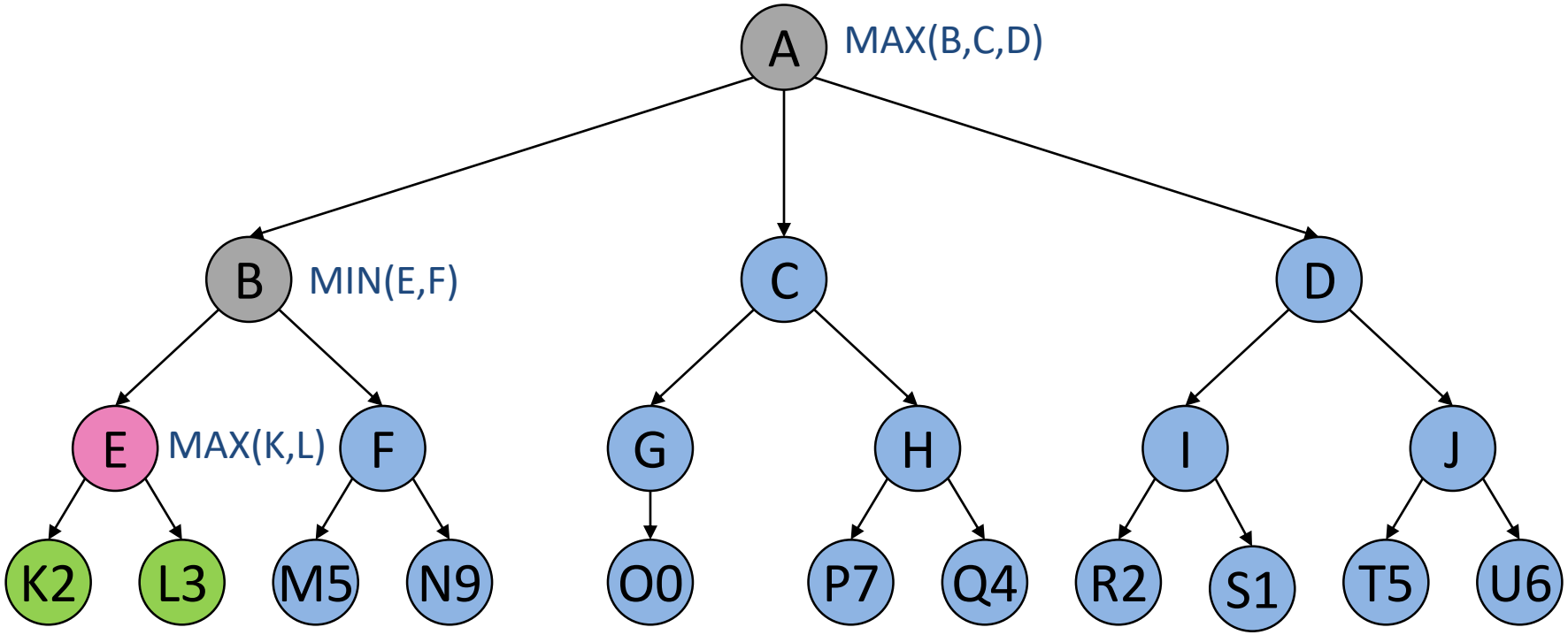
Min-Max algoritam (ilustracija)



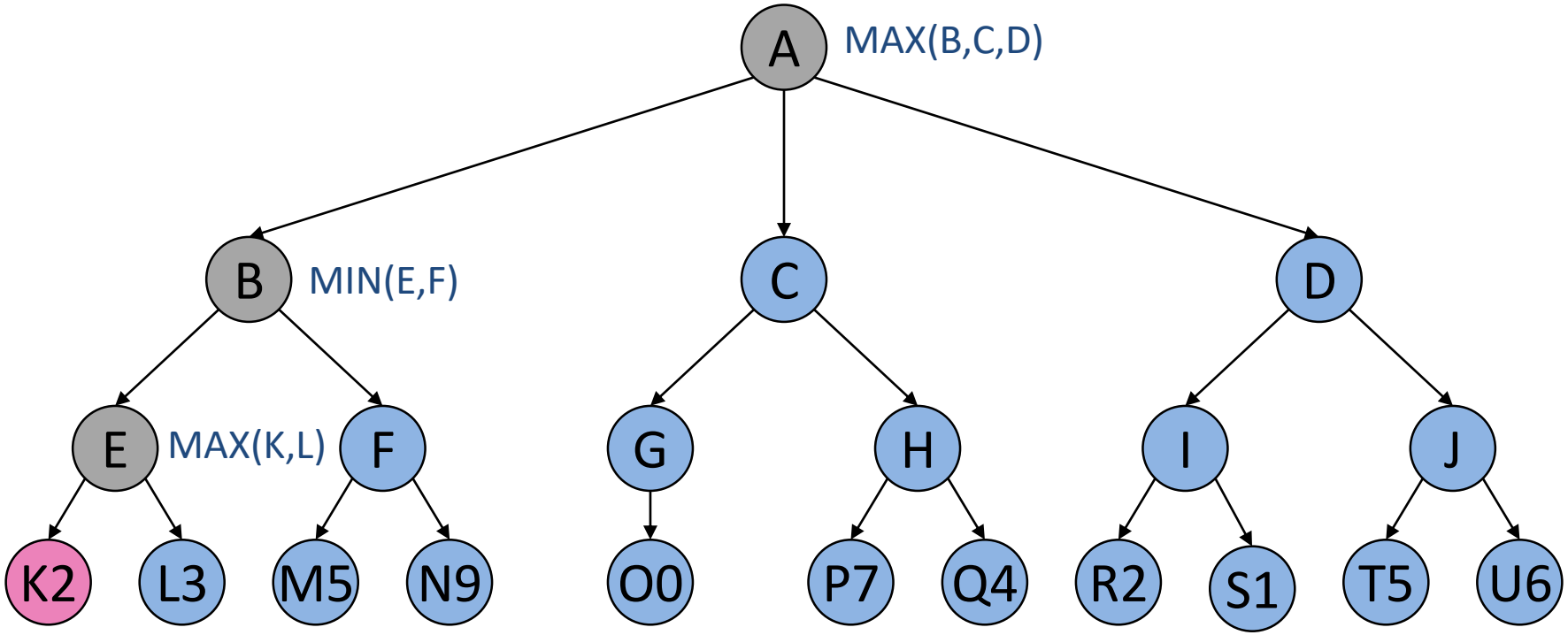
Min-Max algoritam (ilustracija)



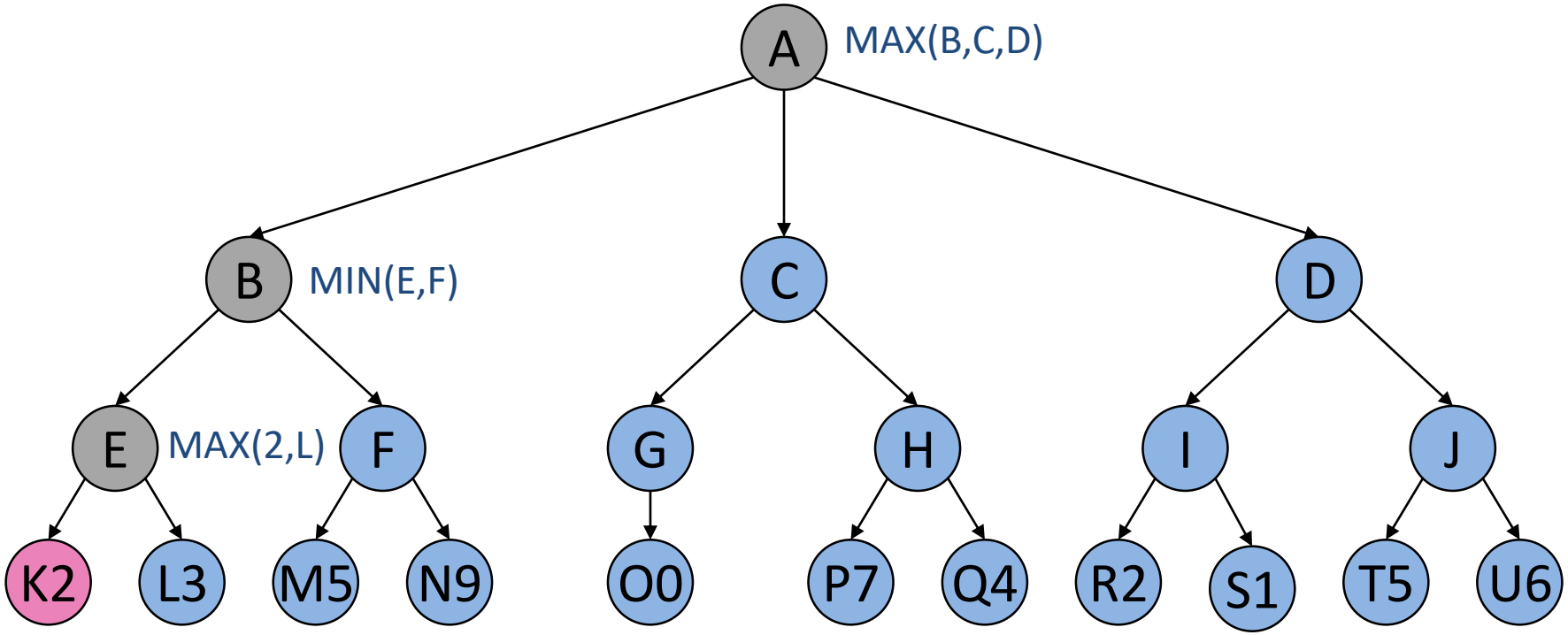
Min-Max algoritam (ilustracija)



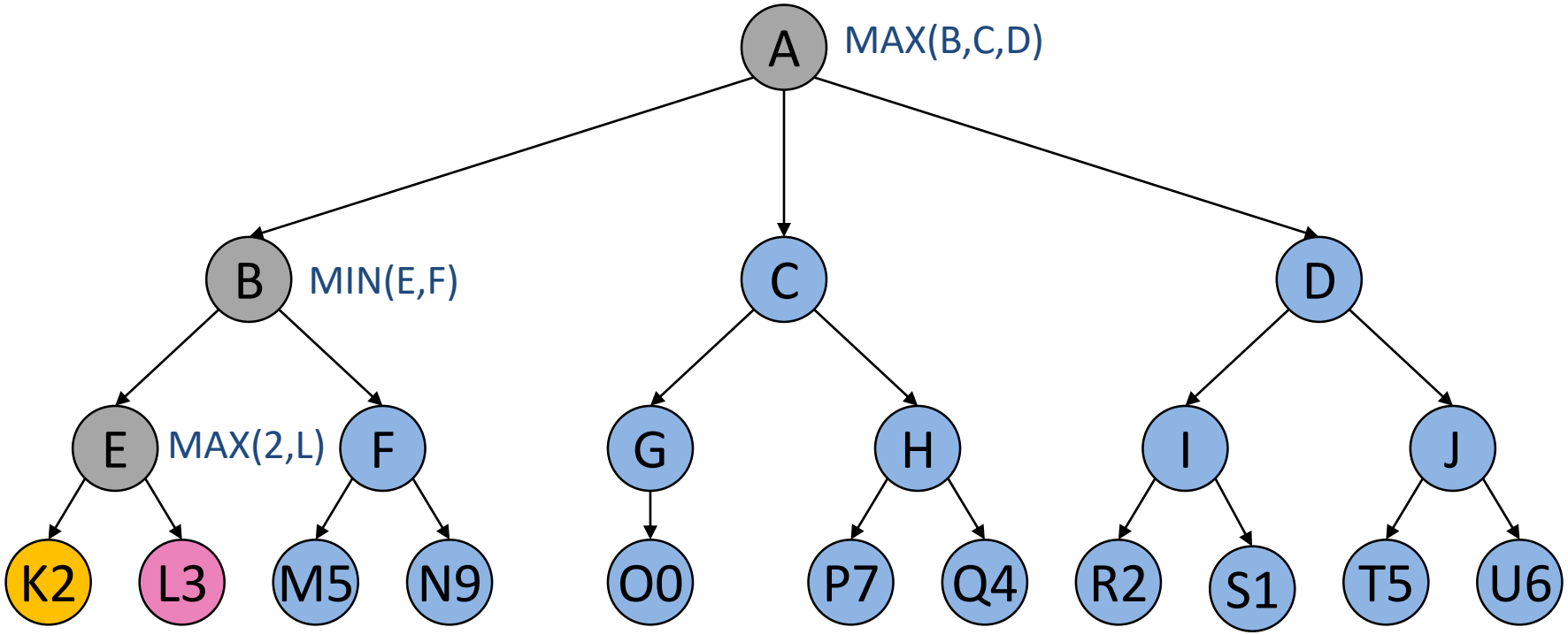
Min-Max algoritam (ilustracija)



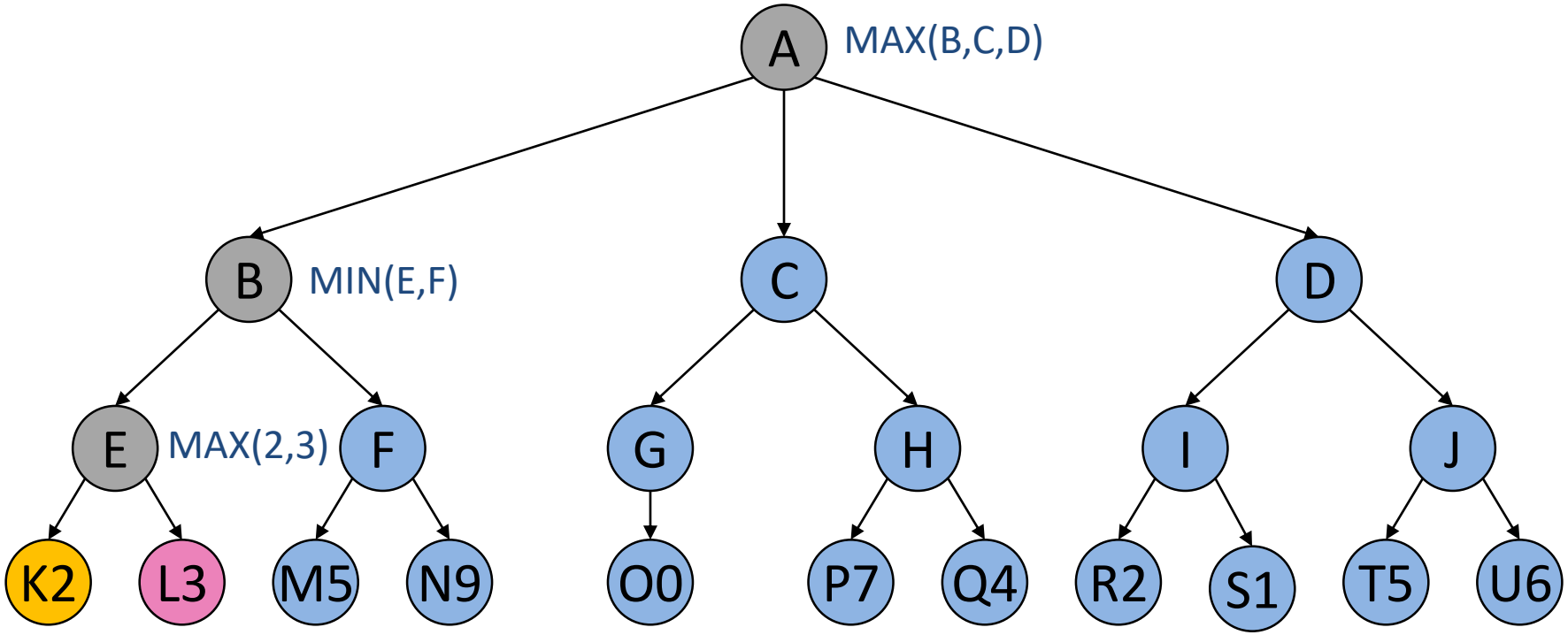
Min-Max algoritam (ilustracija)



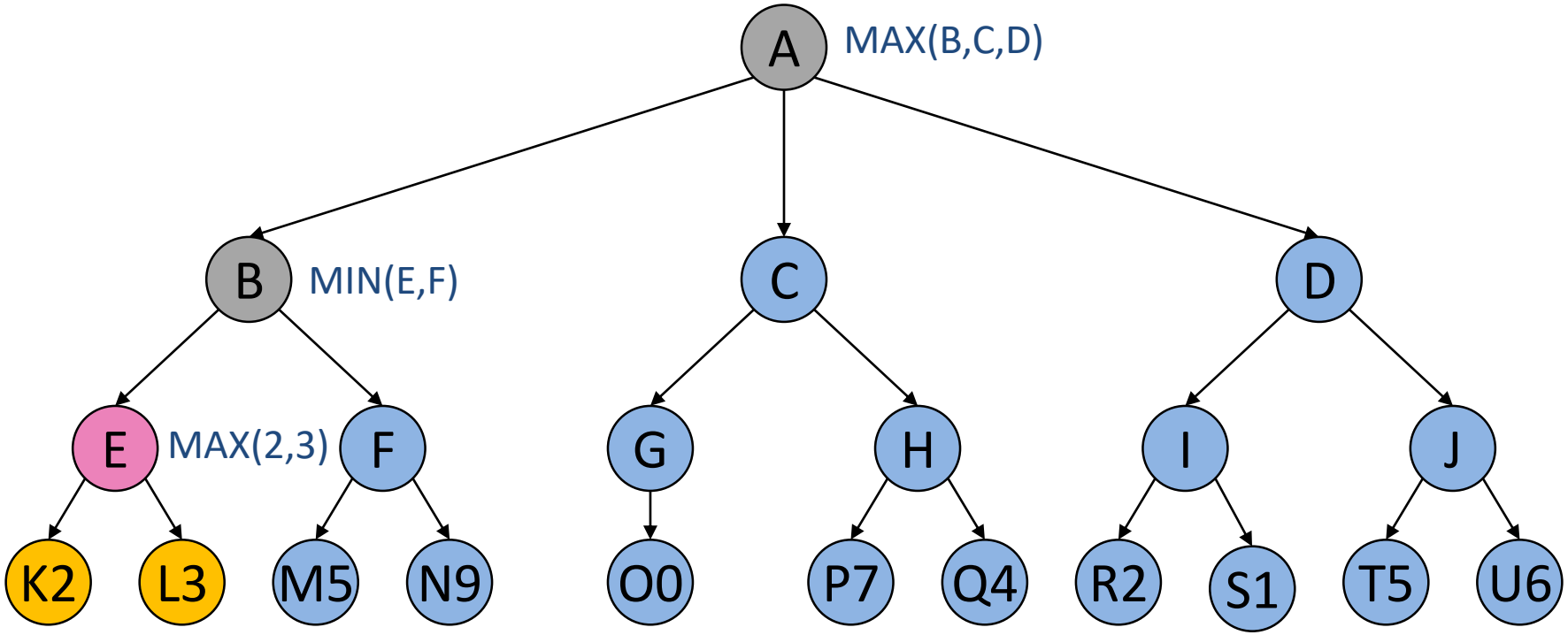
Min-Max algoritam (ilustracija)



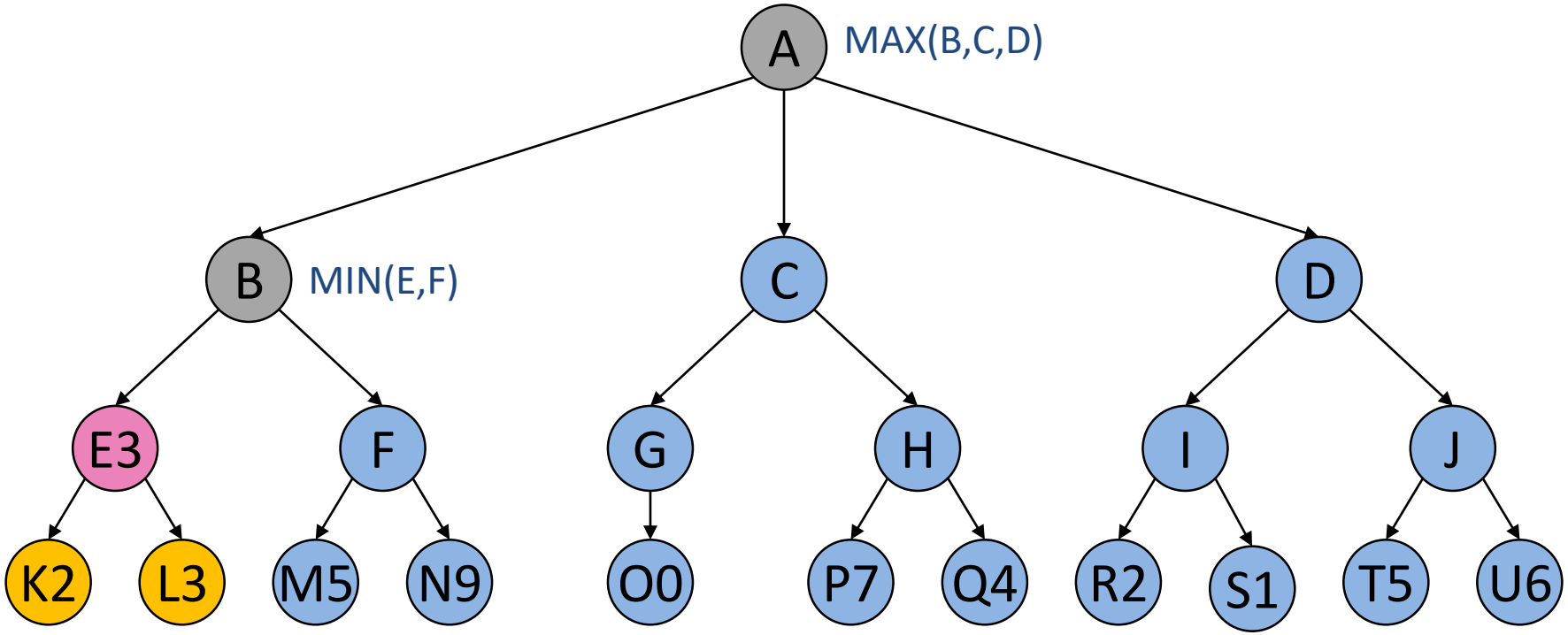
Min-Max algoritam (ilustracija)



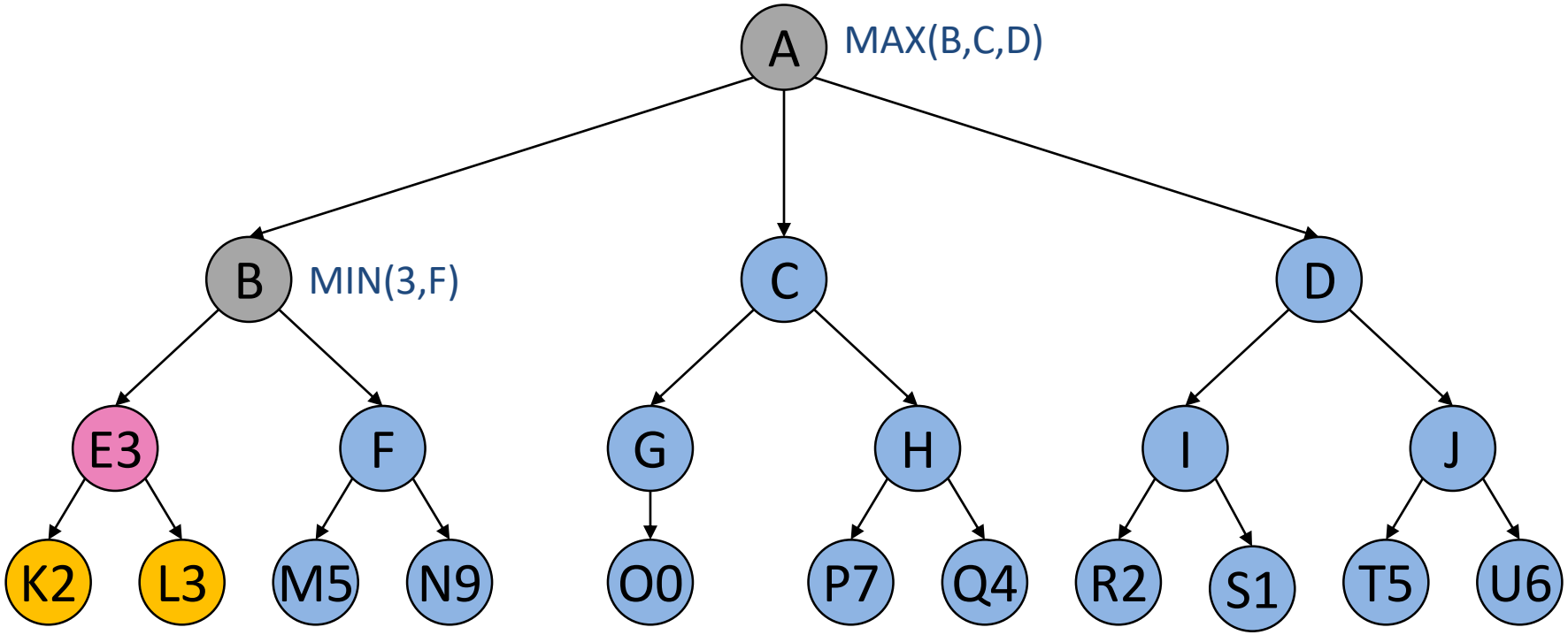
Min-Max algoritam (ilustracija)



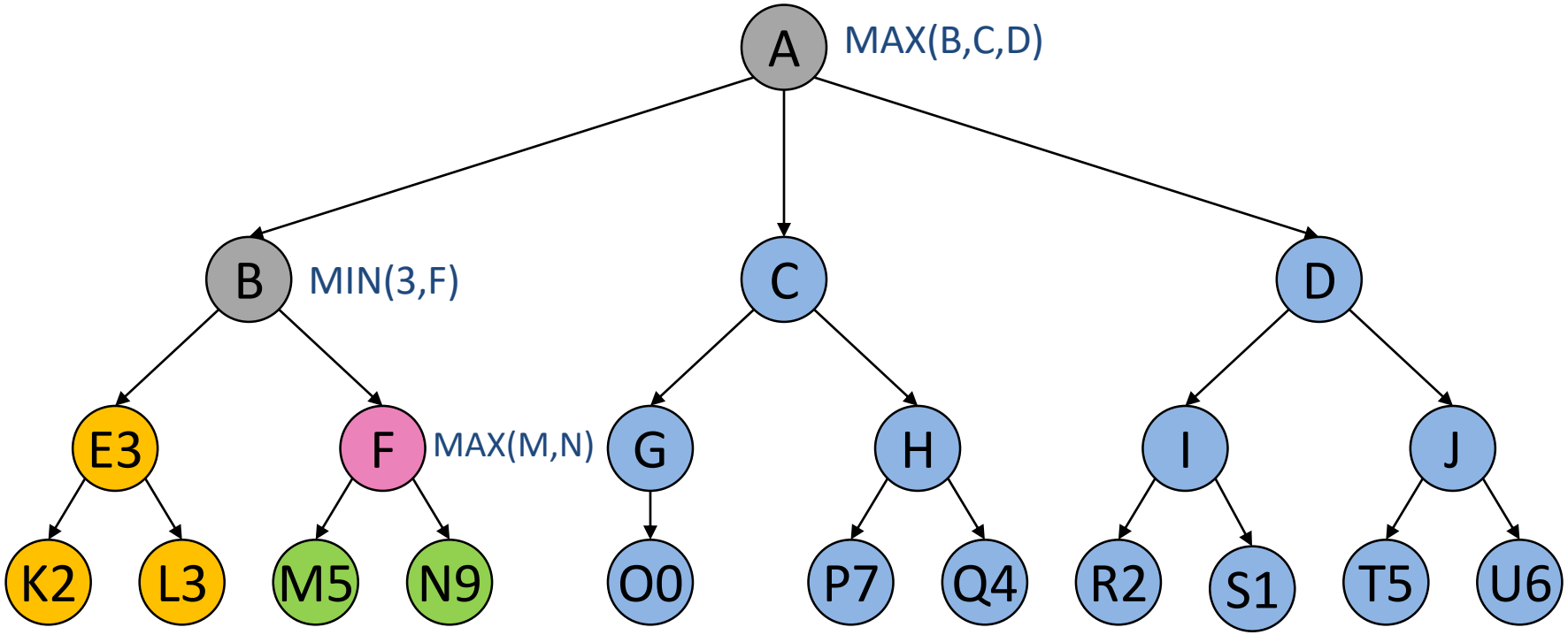
Min-Max algoritam (ilustracija)



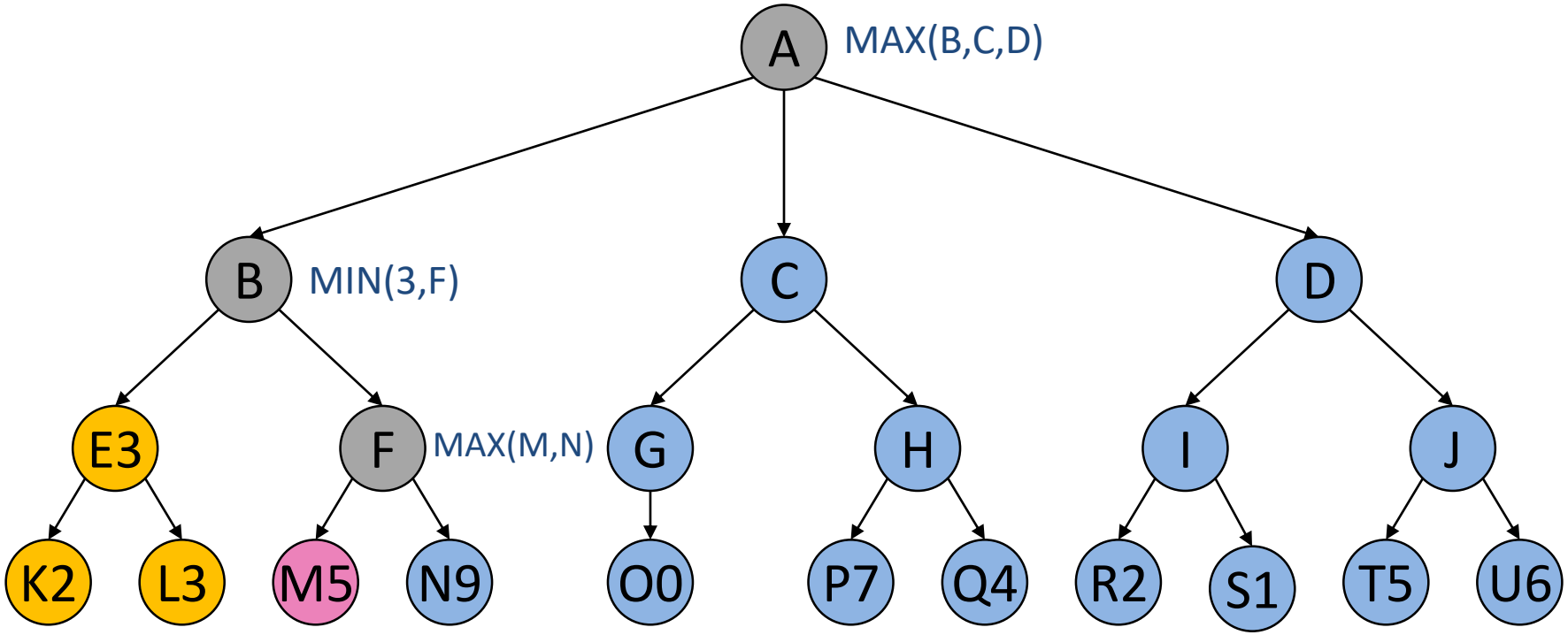
Min-Max algoritam (ilustracija)



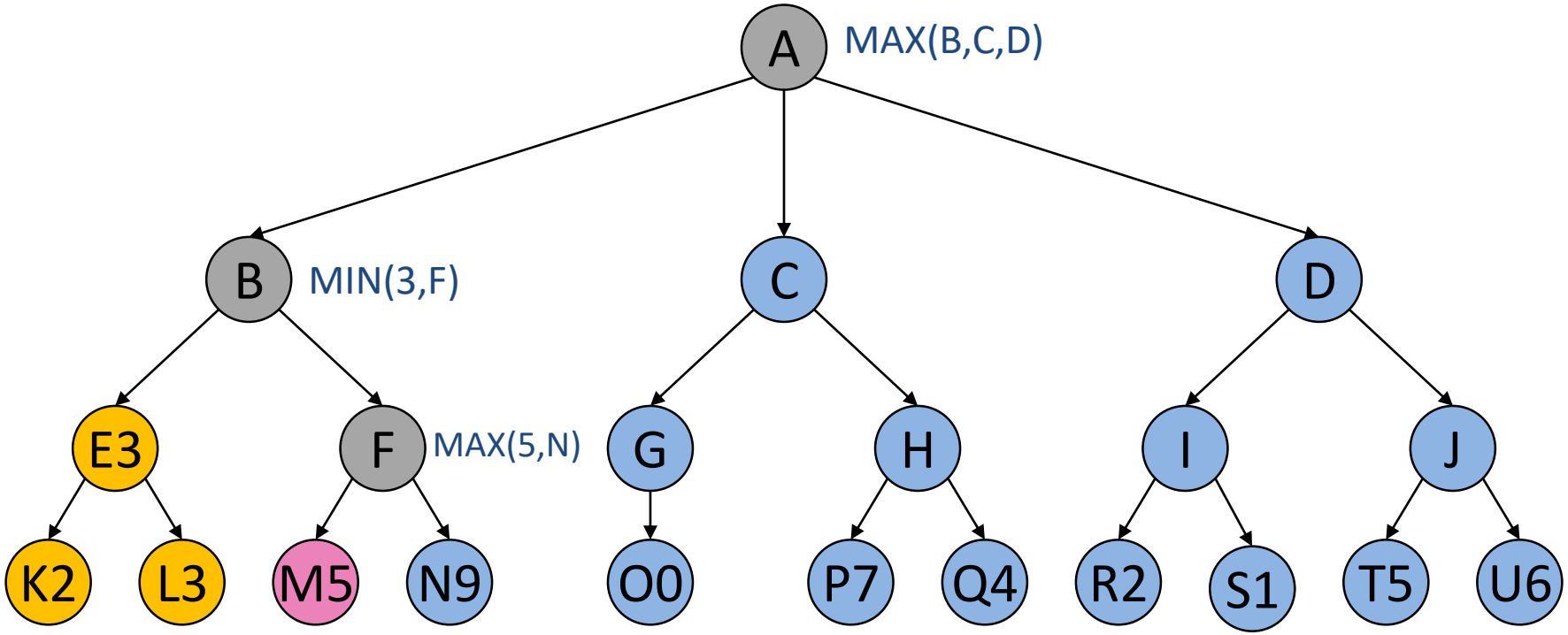
Min-Max algoritam (ilustracija)



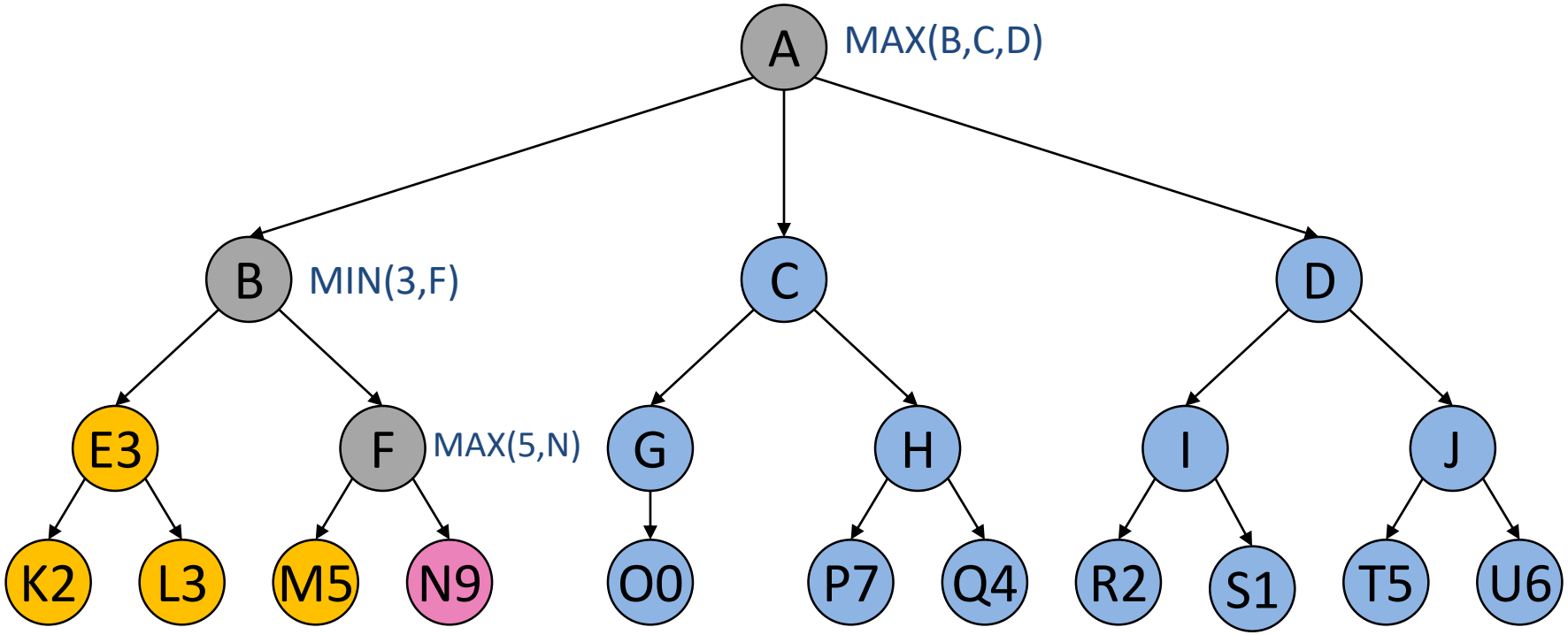
Min-Max algoritam (ilustracija)



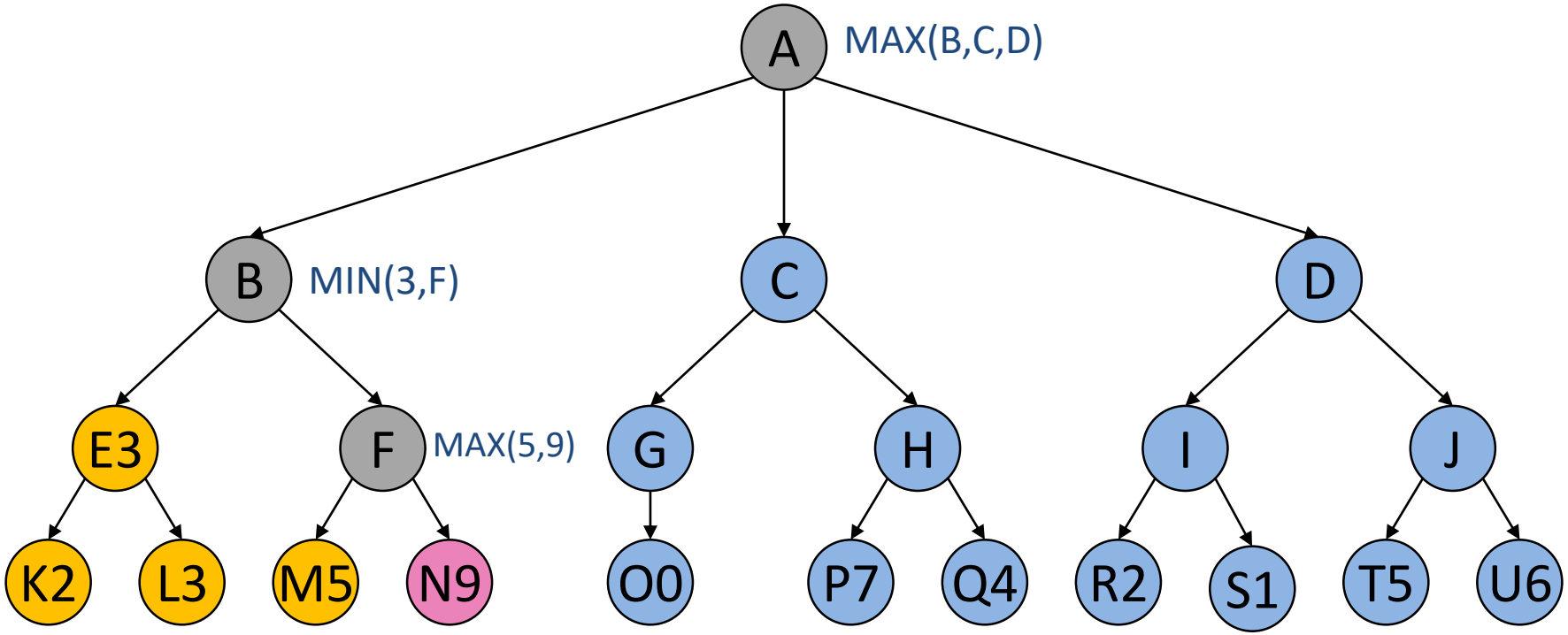
Min-Max algoritam (ilustracija)



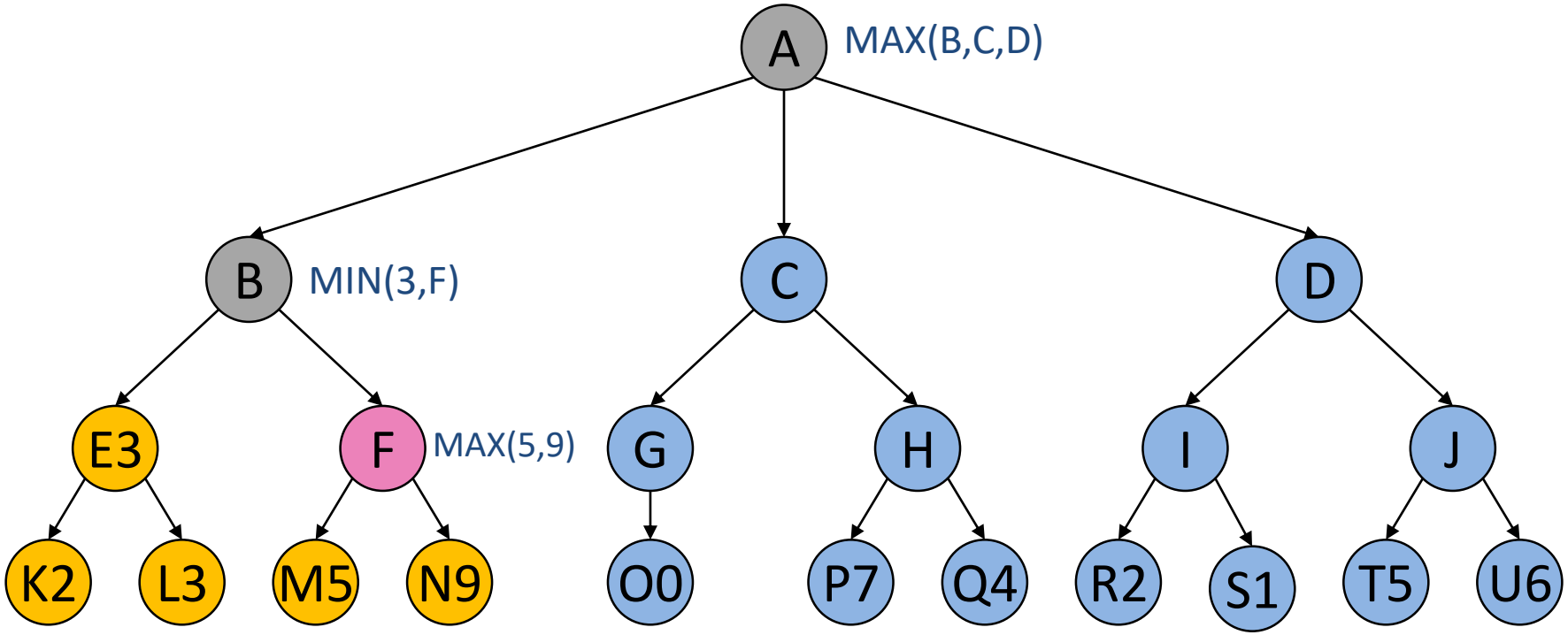
Min-Max algoritam (ilustracija)



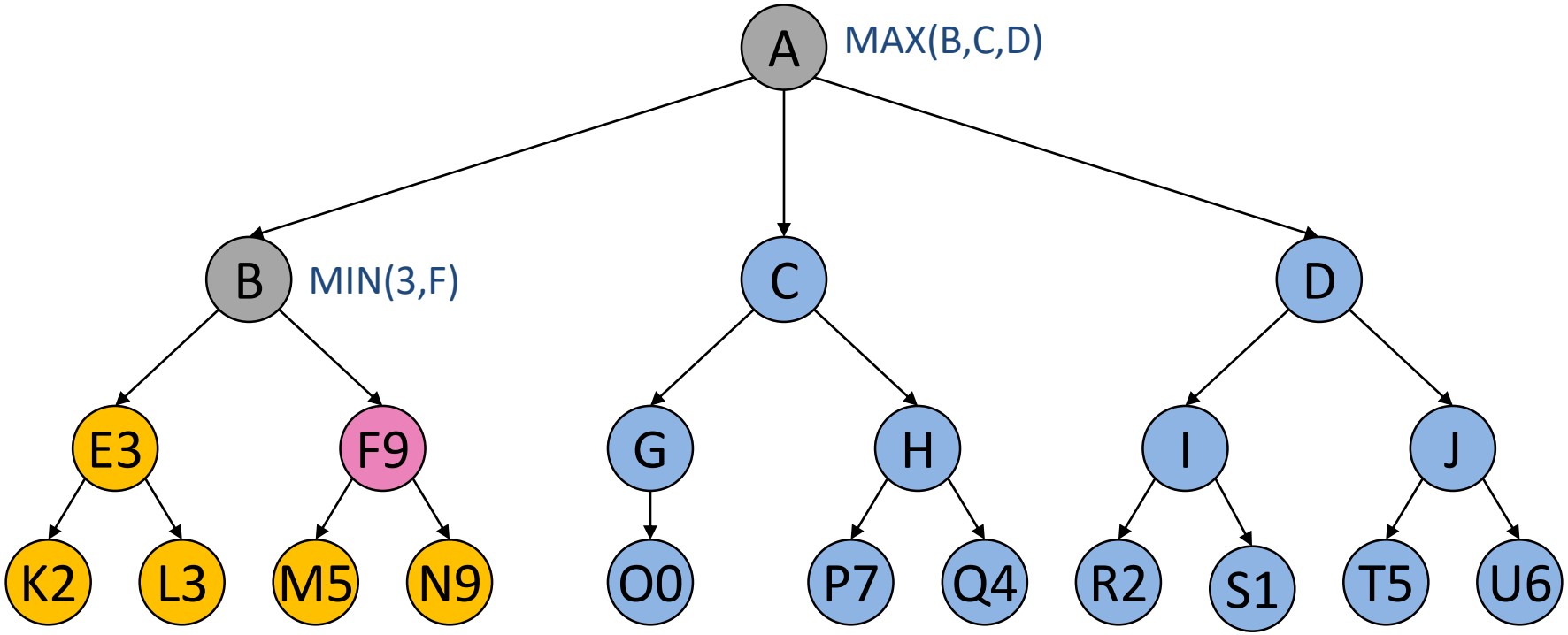
Min-Max algoritam (ilustracija)



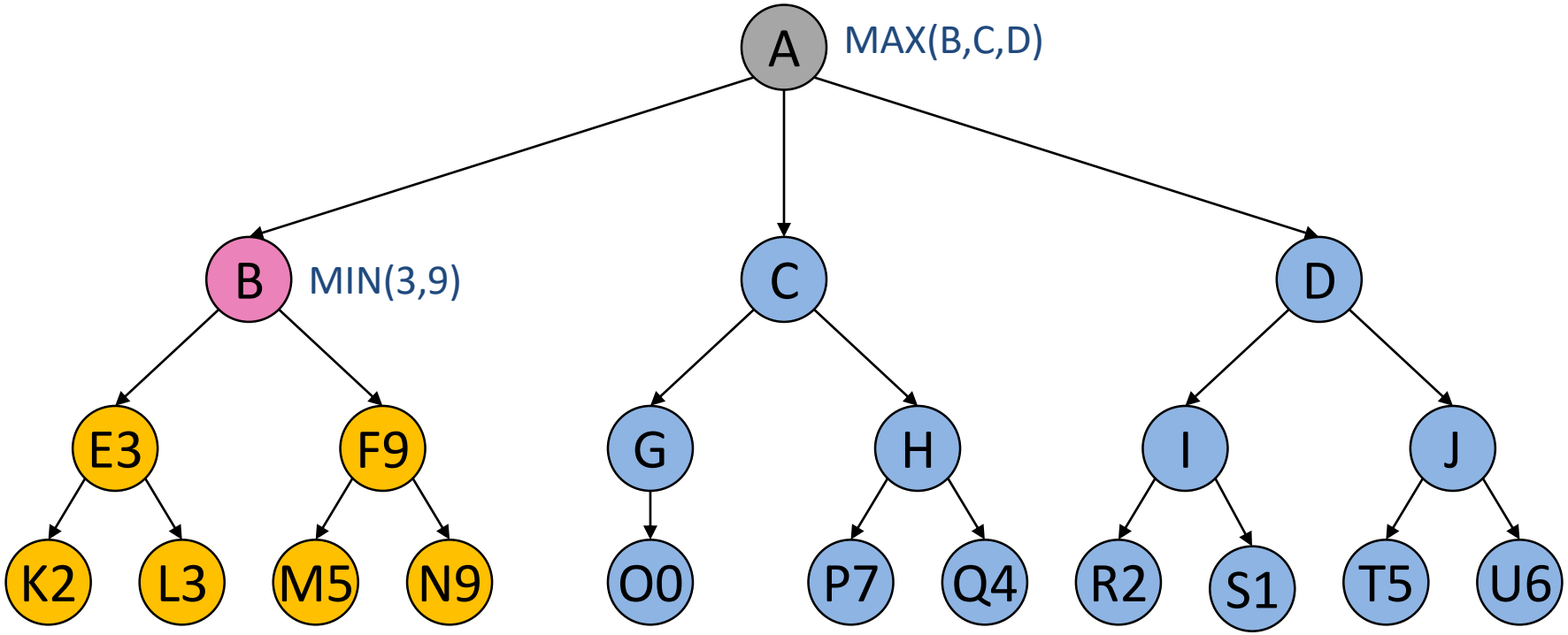
Min-Max algoritam (ilustracija)



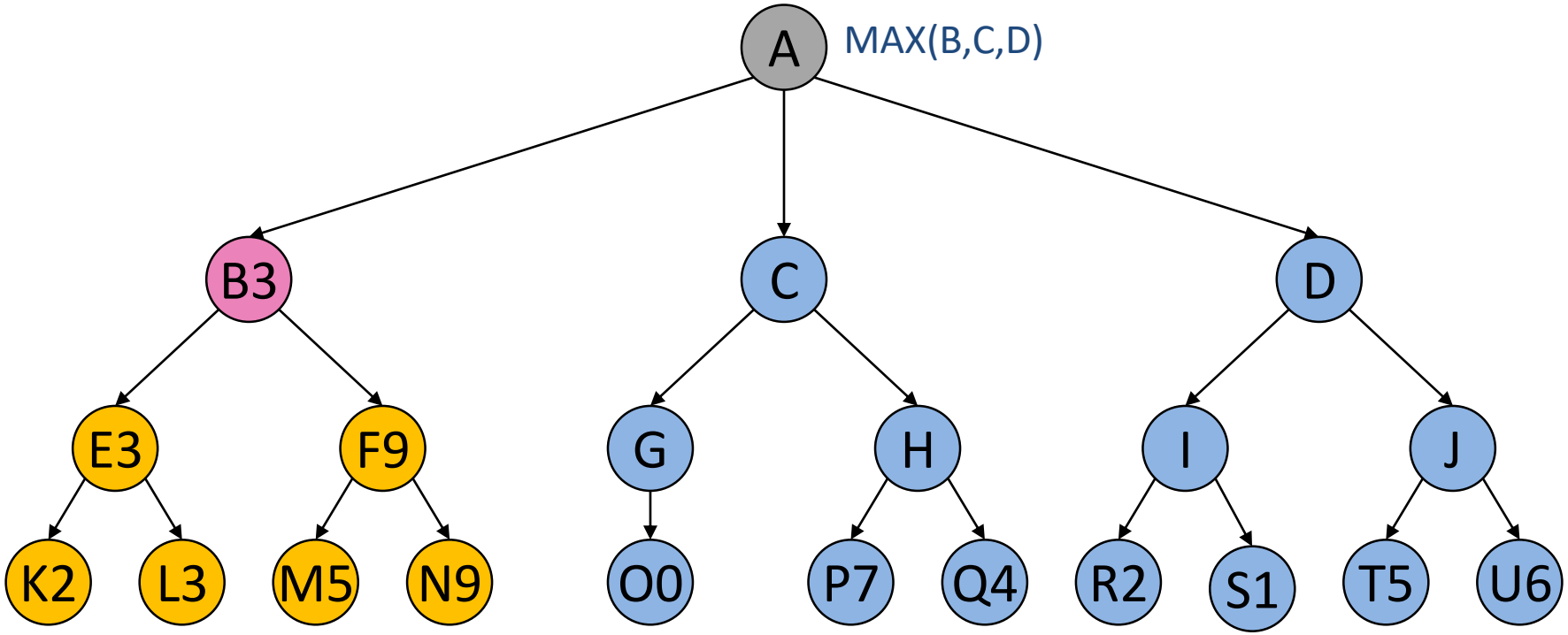
Min-Max algoritam (ilustracija)



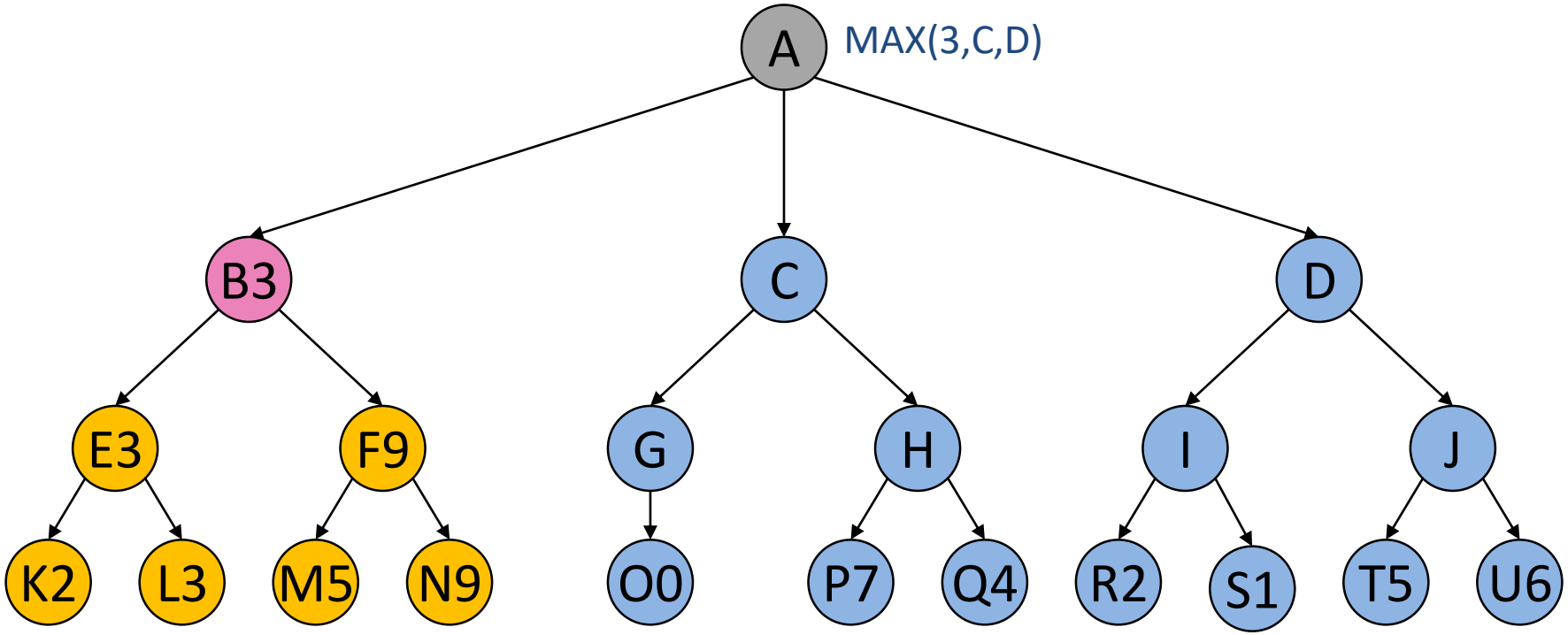
Min-Max algoritam (ilustracija)



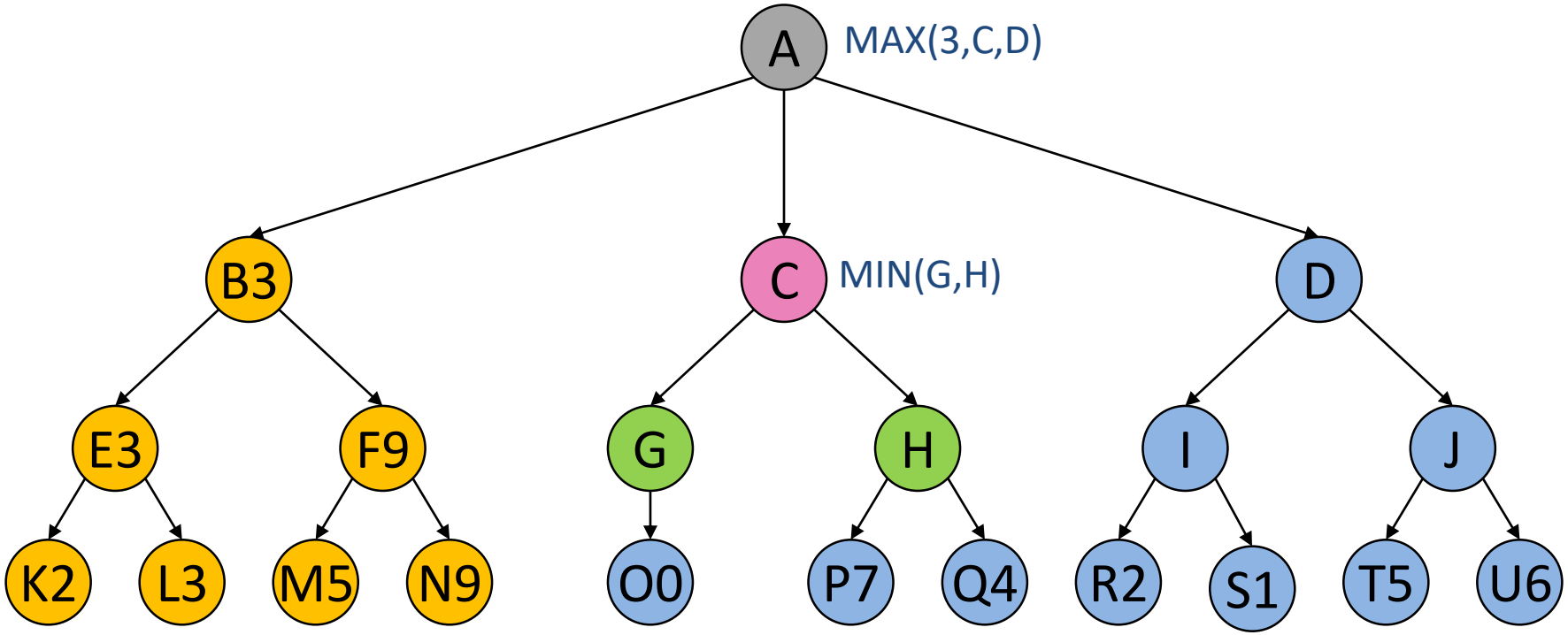
Min-Max algoritam (ilustracija)



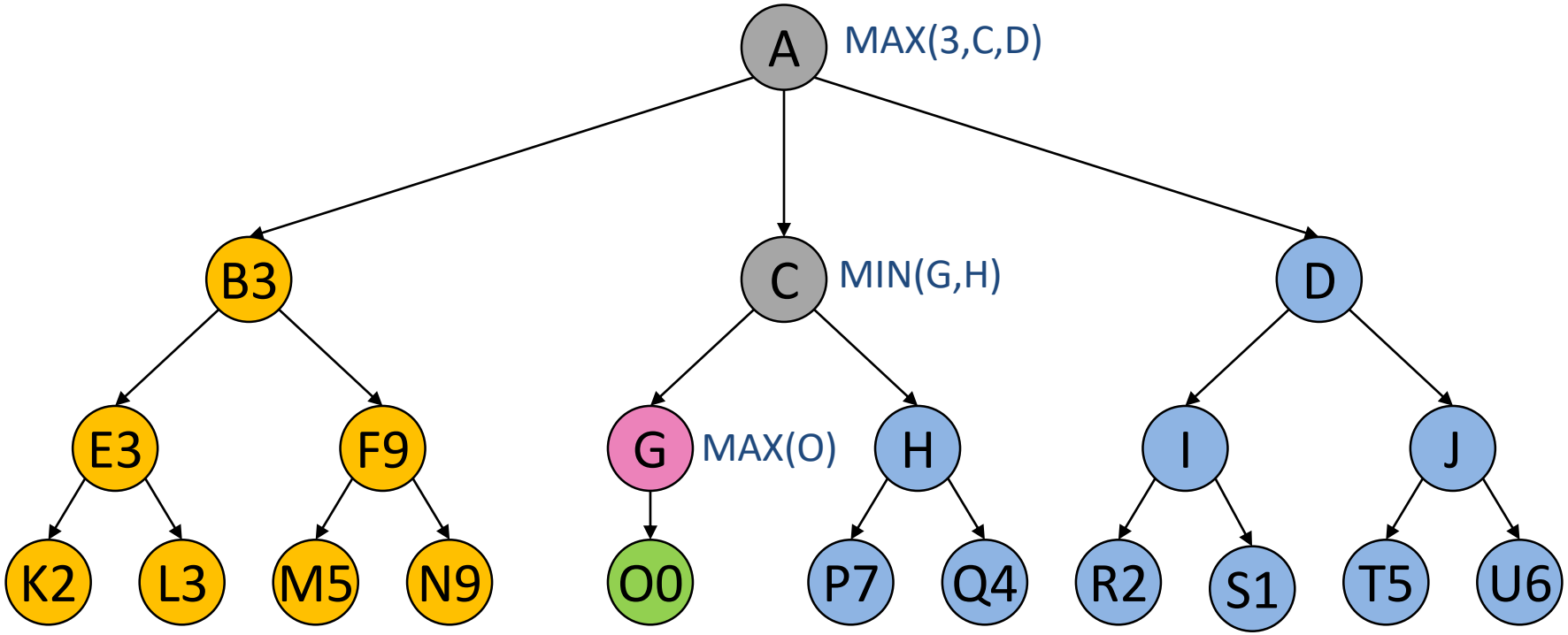
Min-Max algoritam (ilustracija)



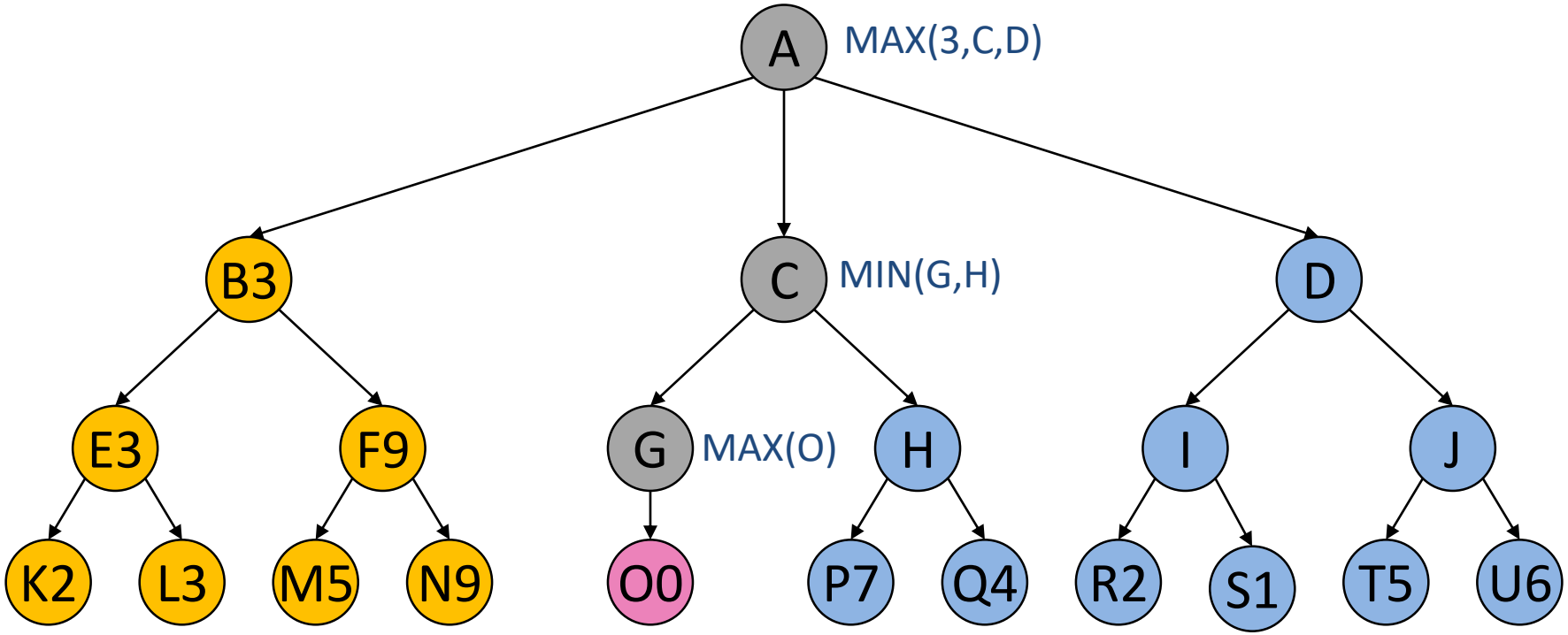
Min-Max algoritam (ilustracija)



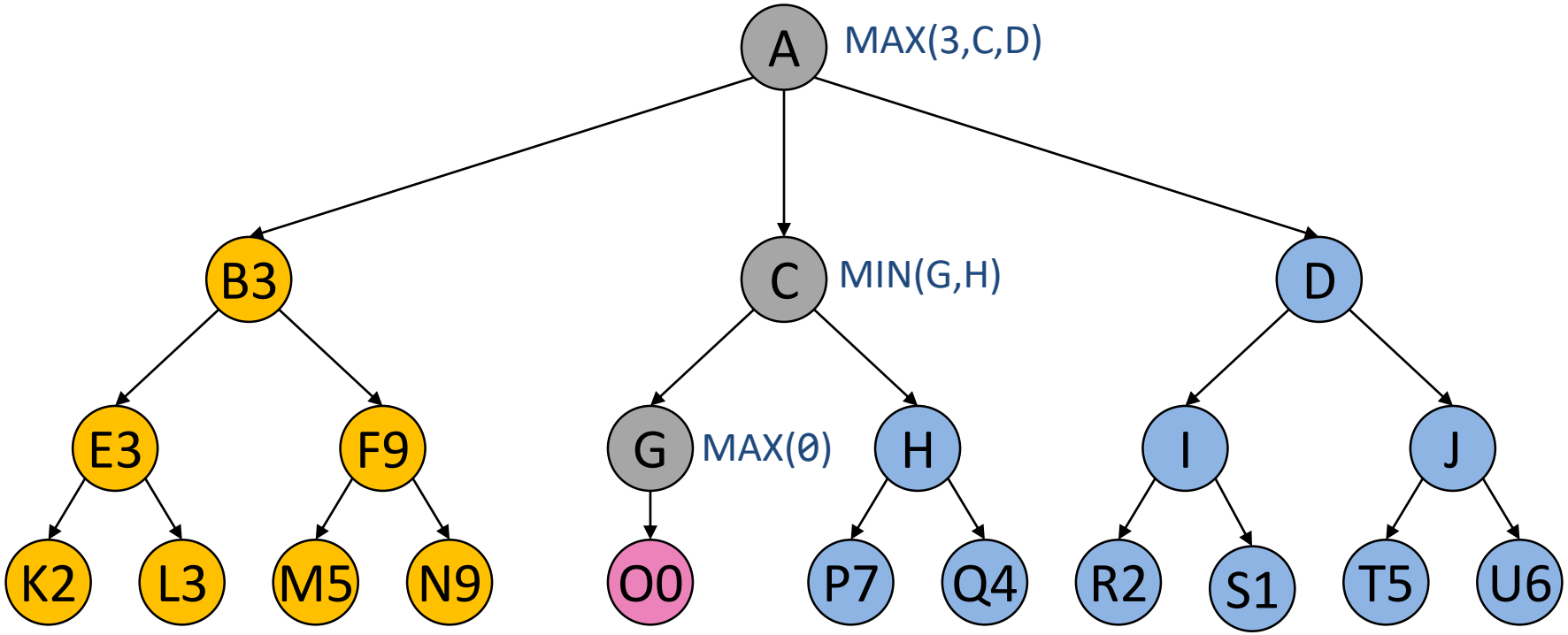
Min-Max algoritam (ilustracija)



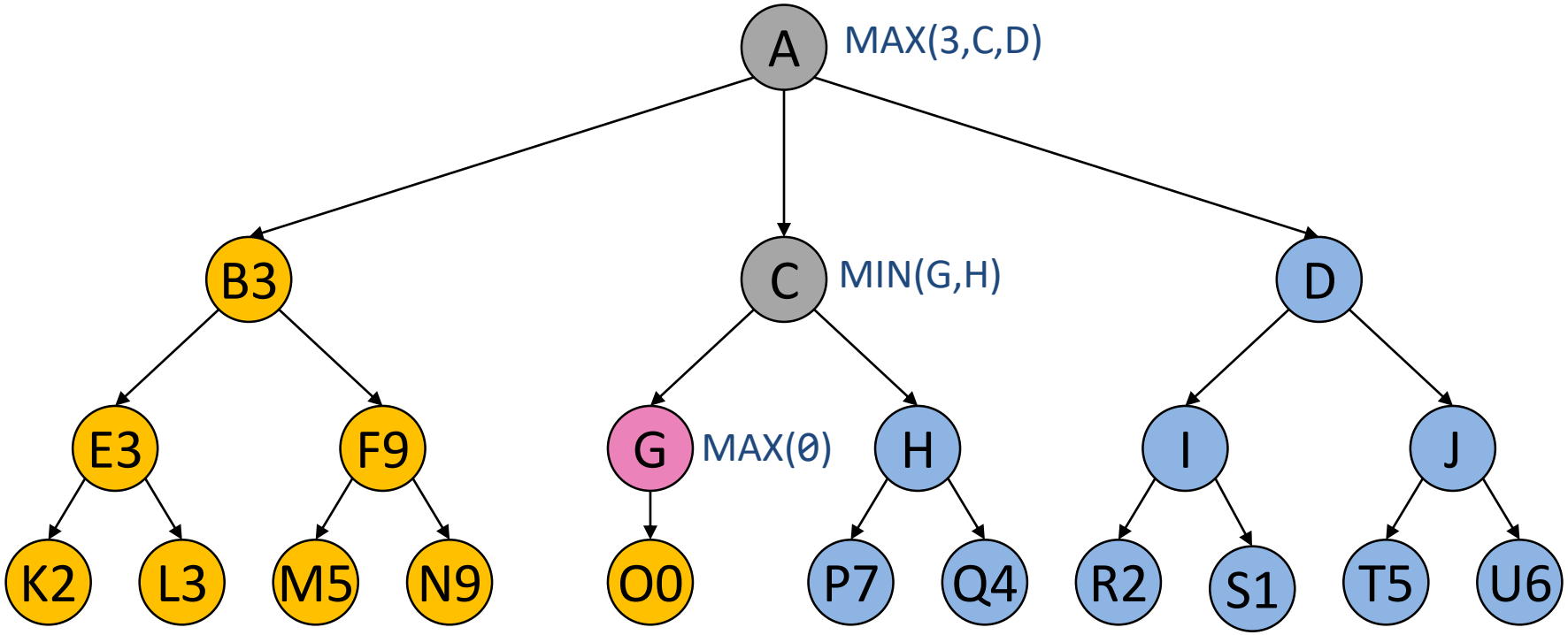
Min-Max algoritam (ilustracija)



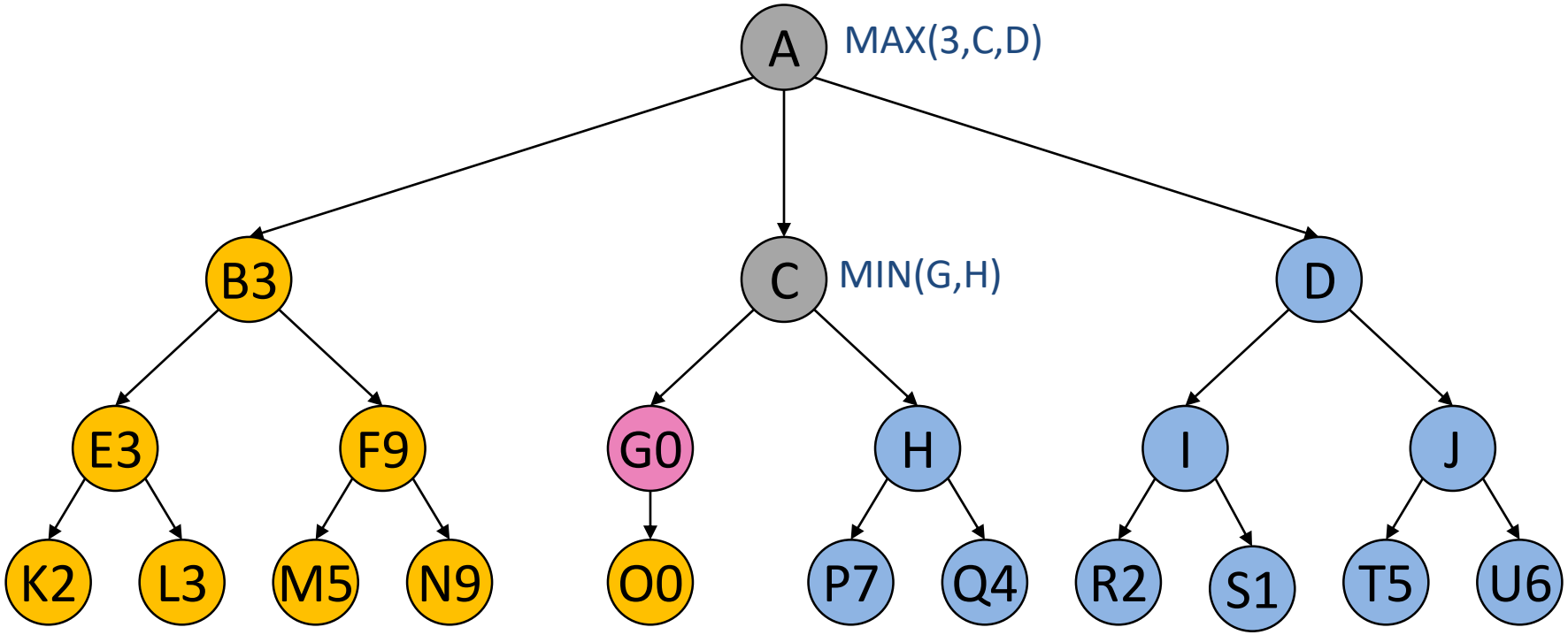
Min-Max algoritam (ilustracija)



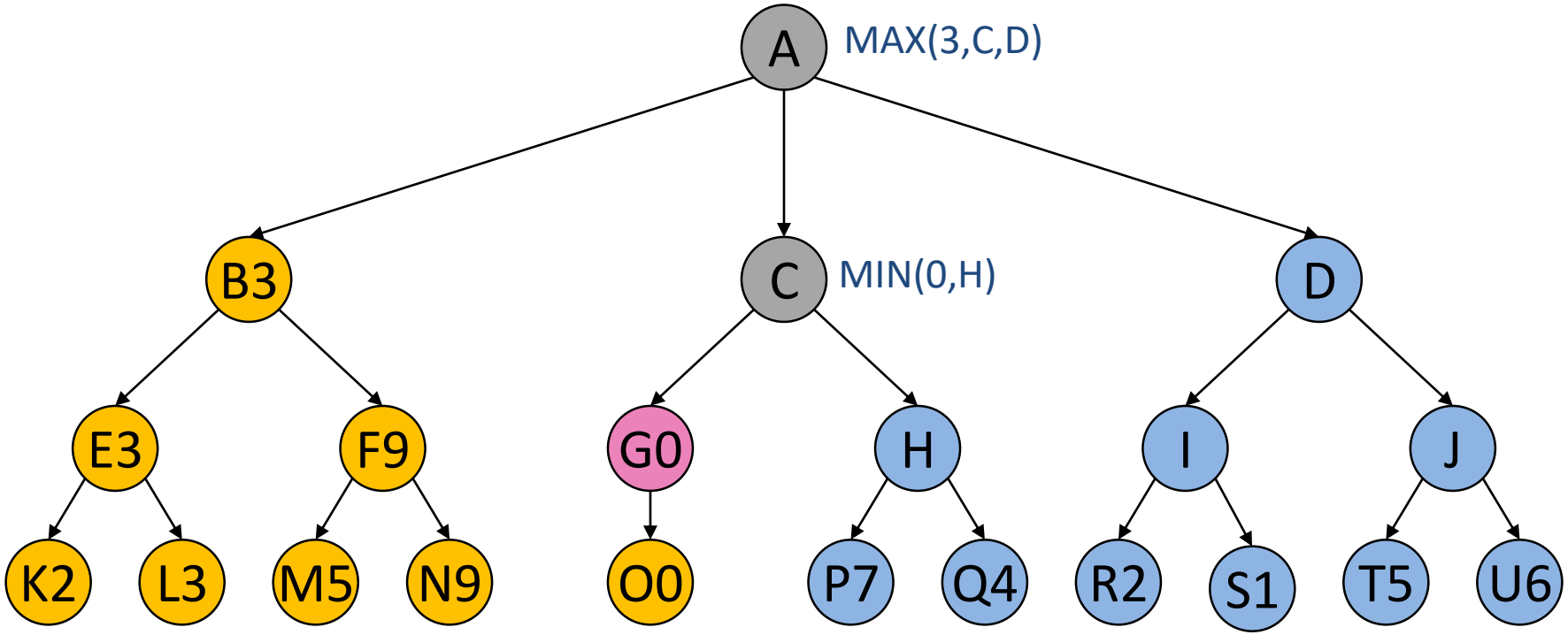
Min-Max algoritam (ilustracija)



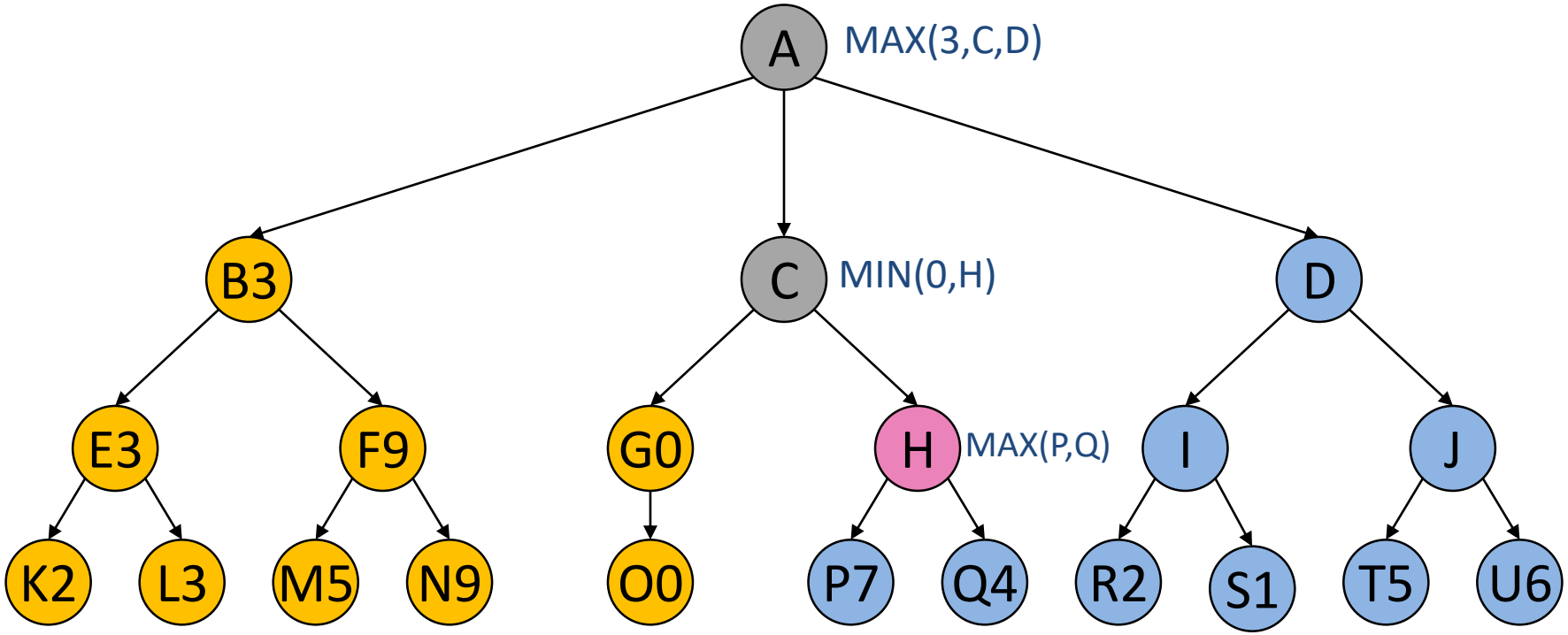
Min-Max algoritam (ilustracija)



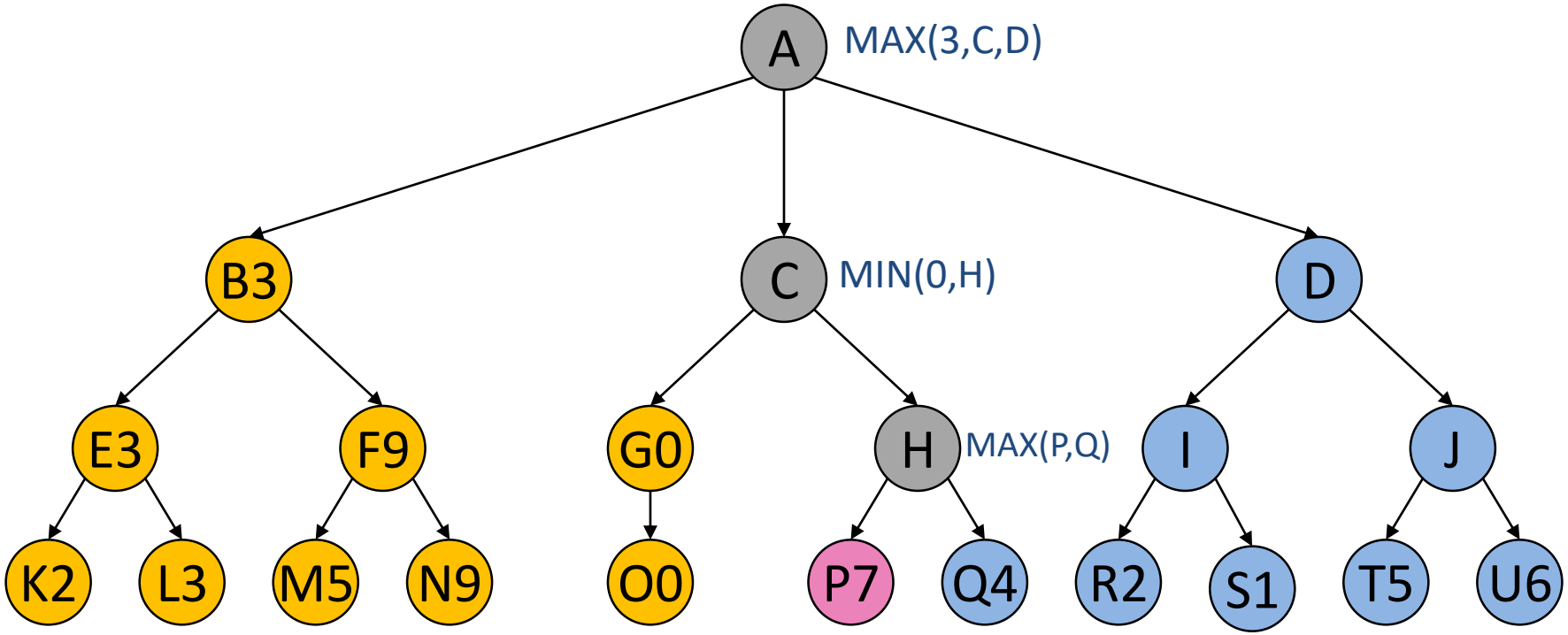
Min-Max algoritam (ilustracija)



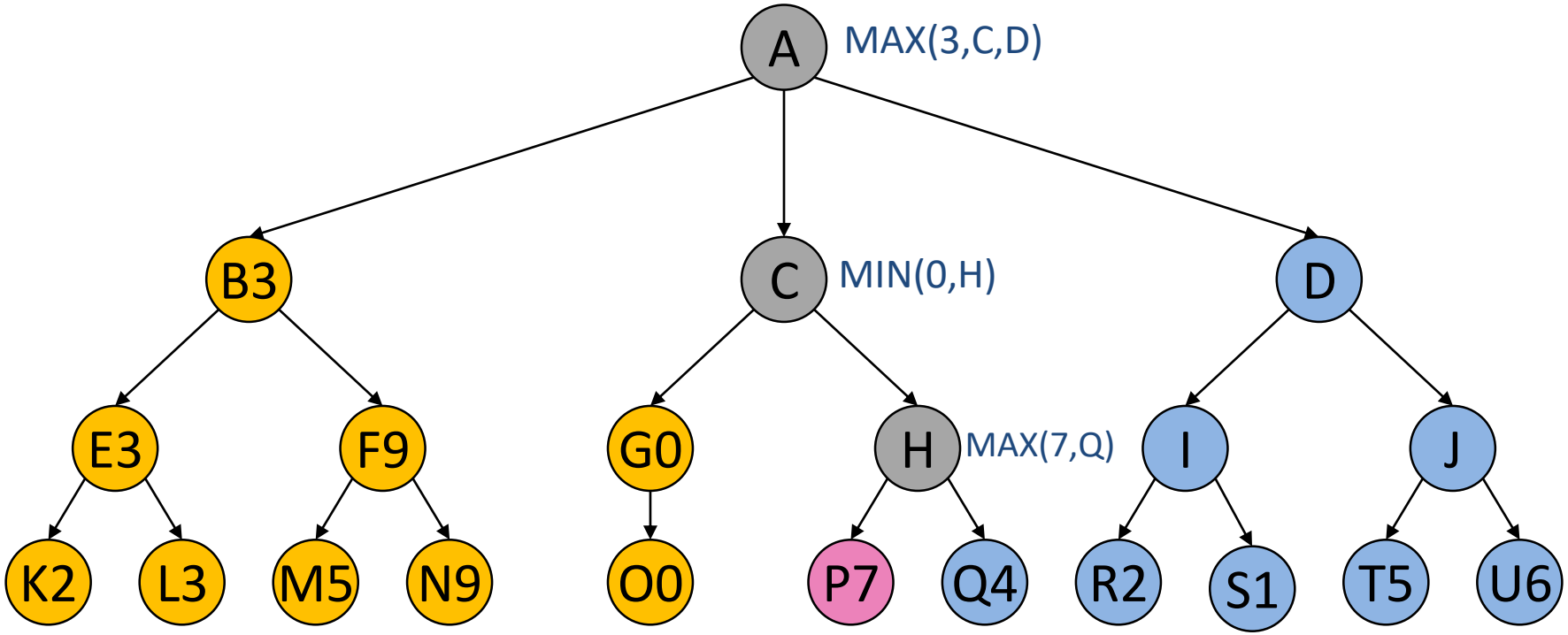
Min-Max algoritam (ilustracija)



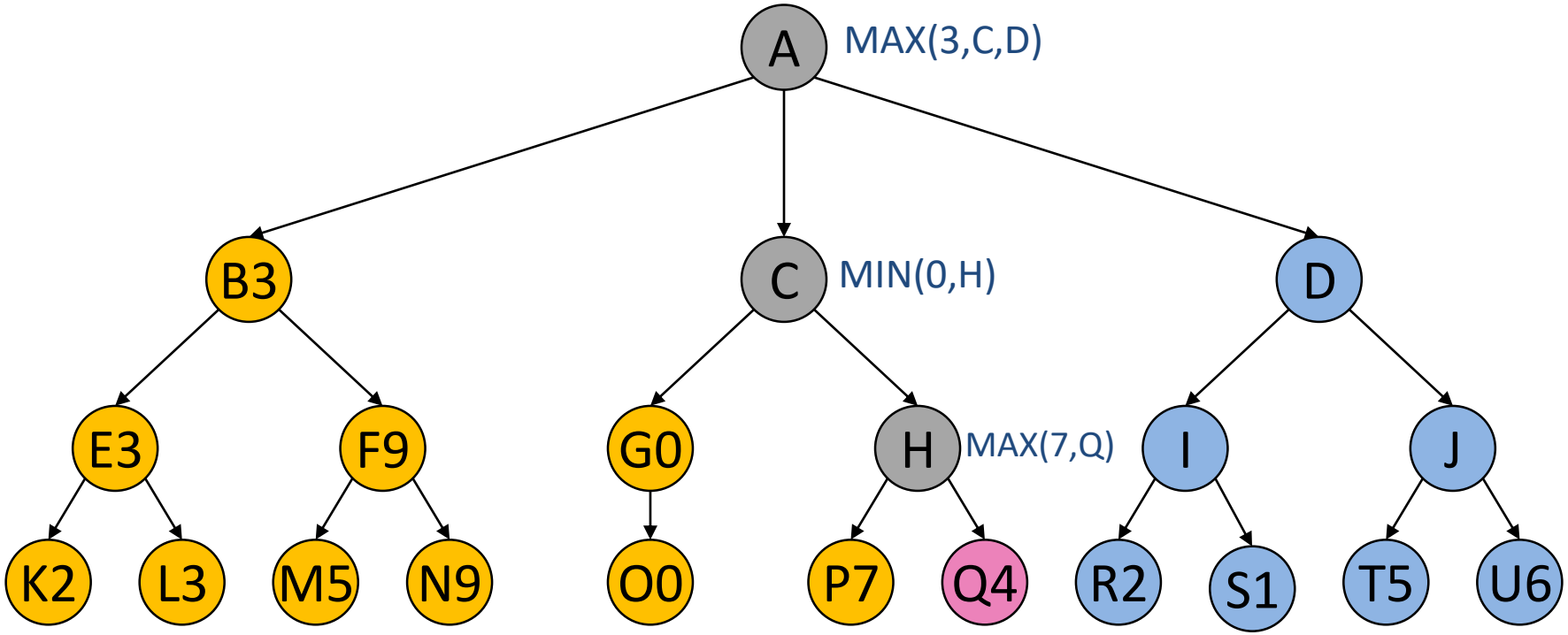
Min-Max algoritam (ilustracija)



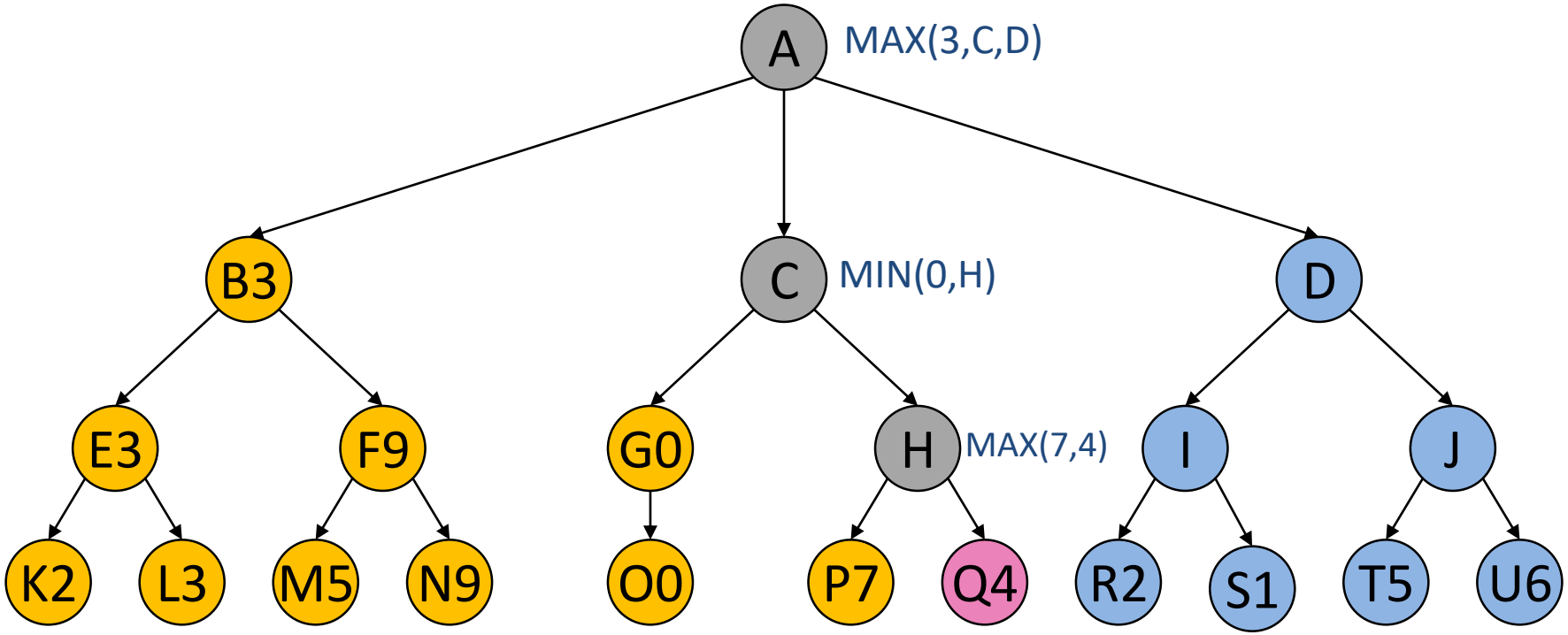
Min-Max algoritam (ilustracija)



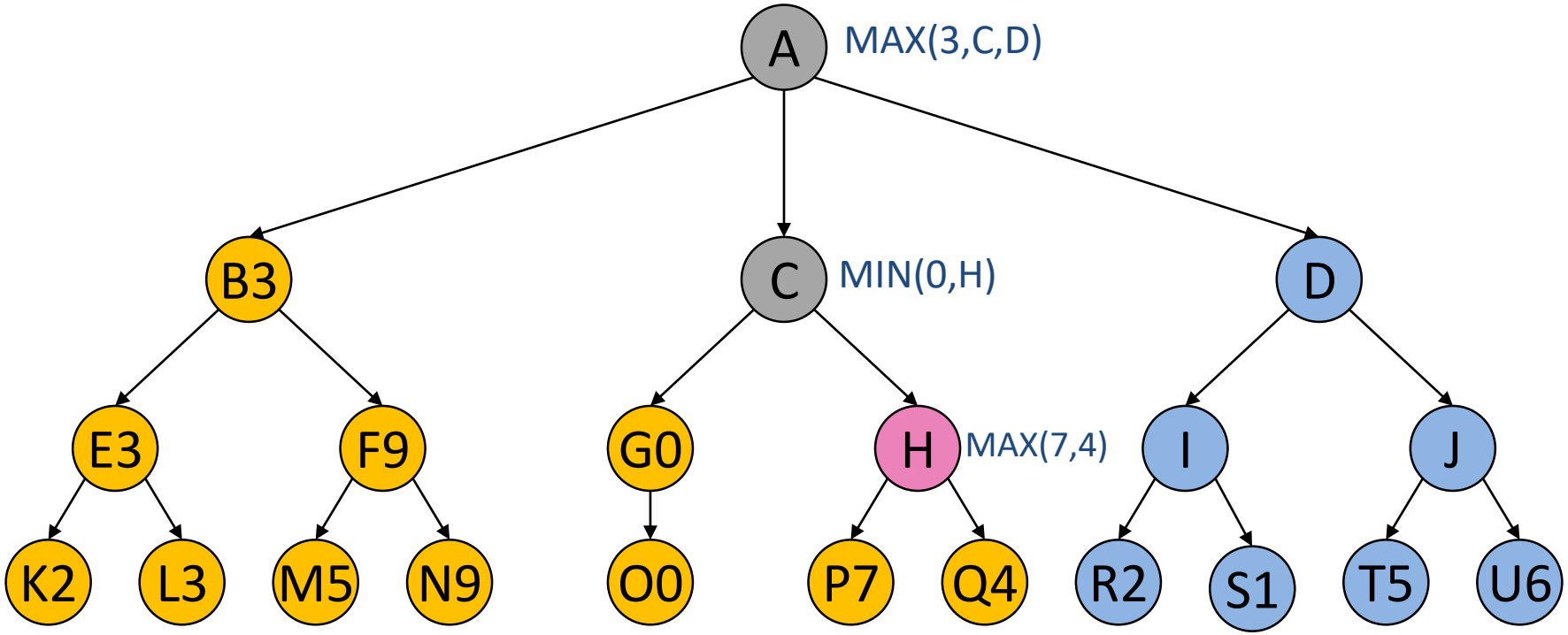
Min-Max algoritam (ilustracija)



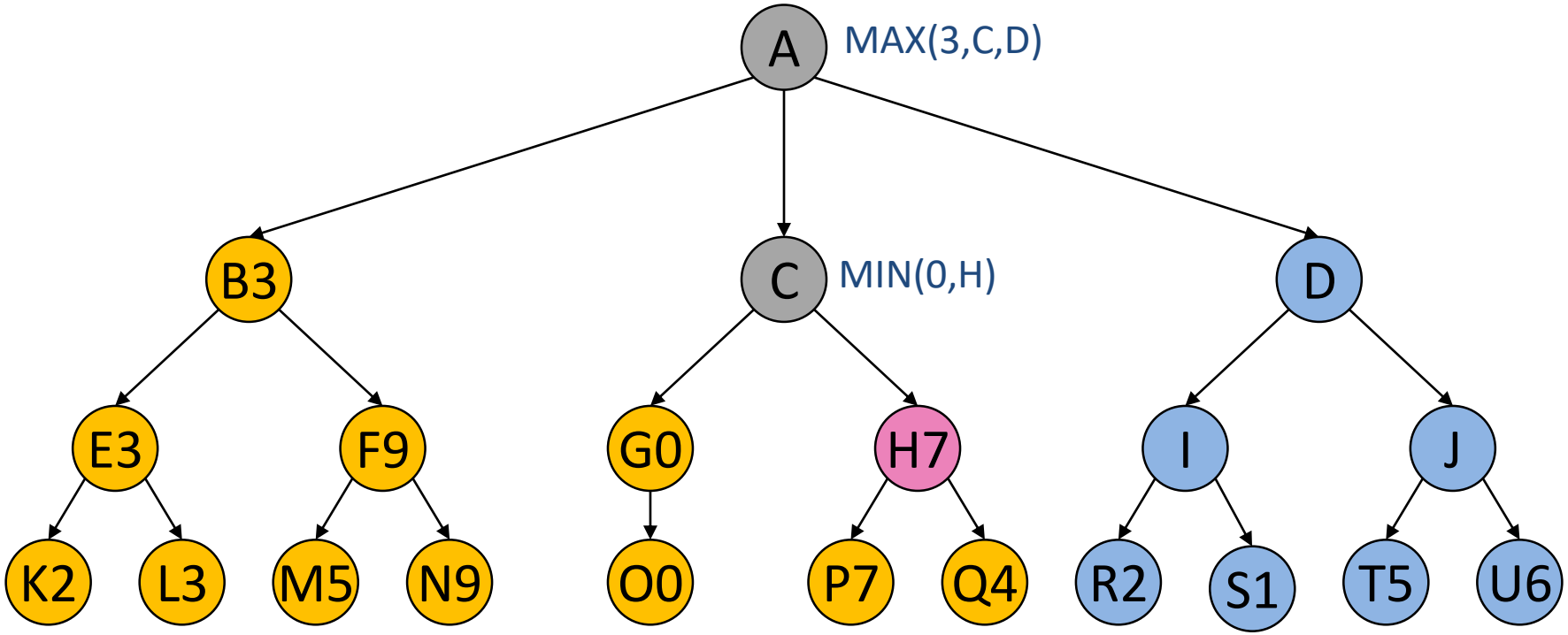
Min-Max algoritam (ilustracija)



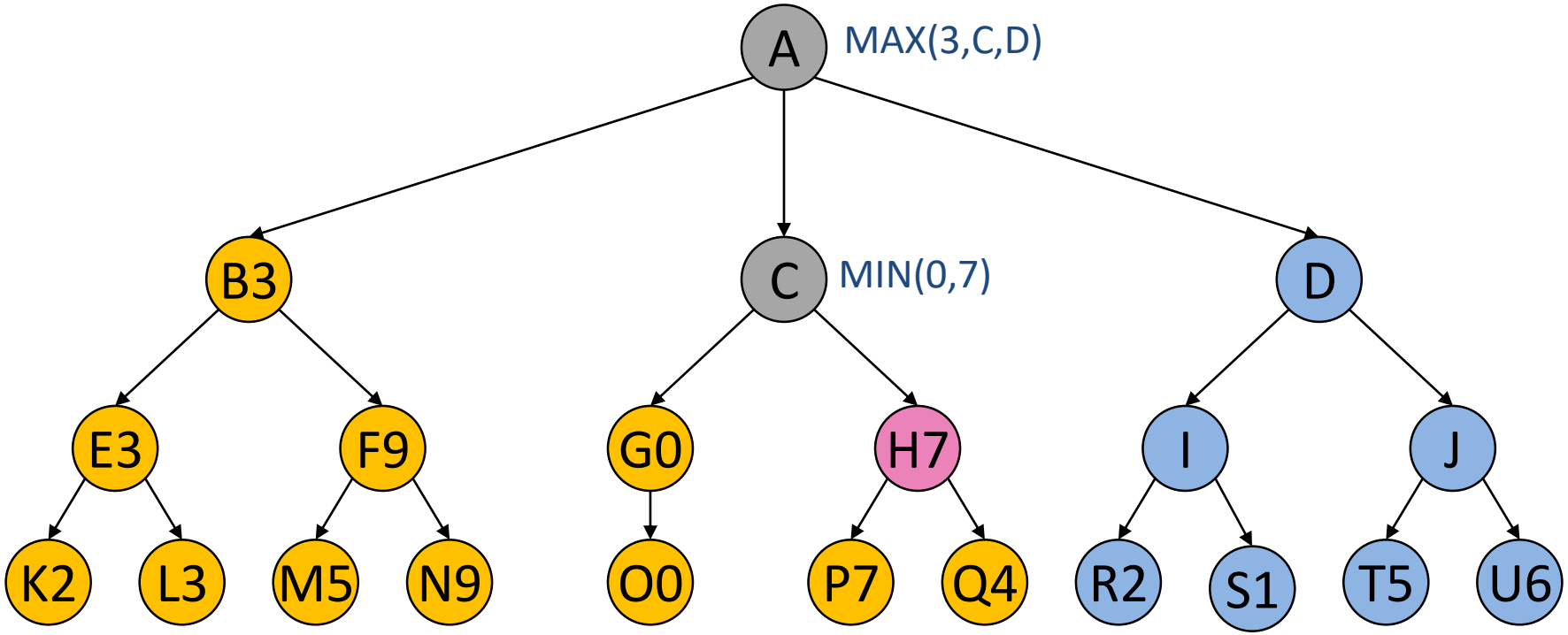
Min-Max algoritam (ilustracija)



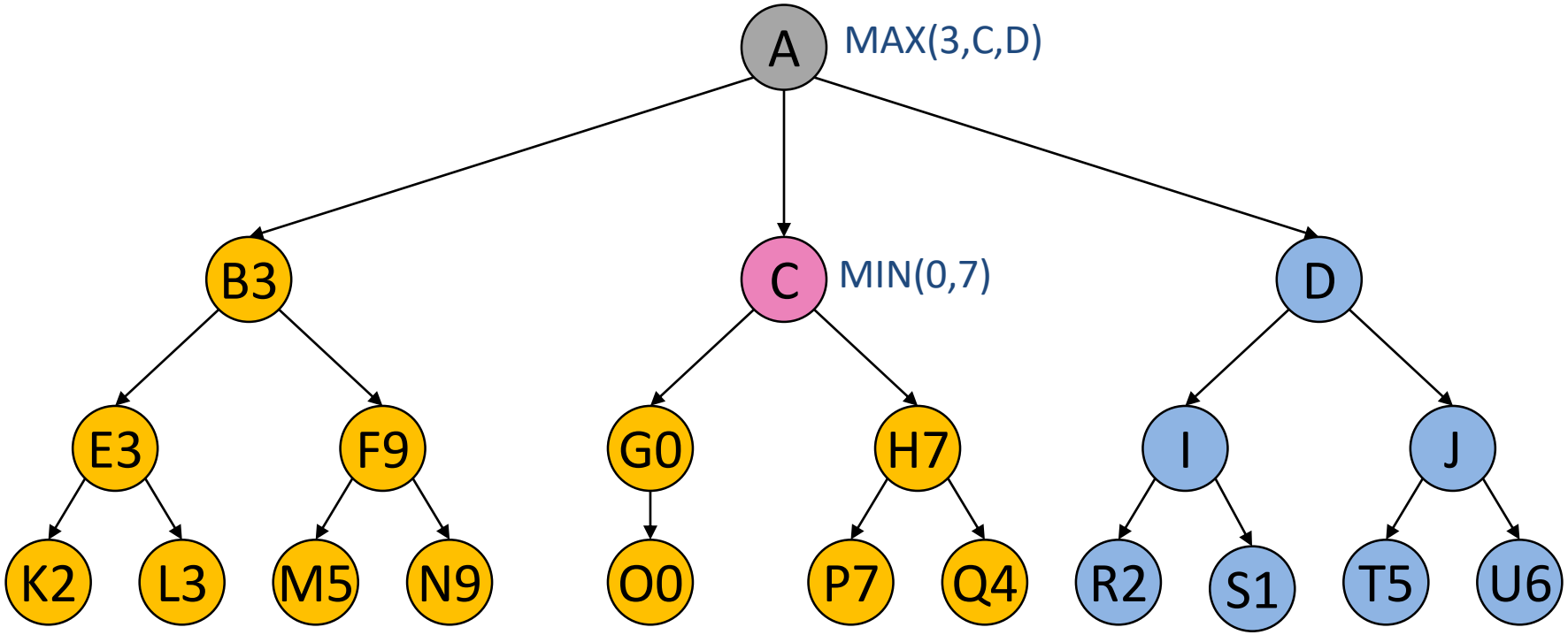
Min-Max algoritam (ilustracija)



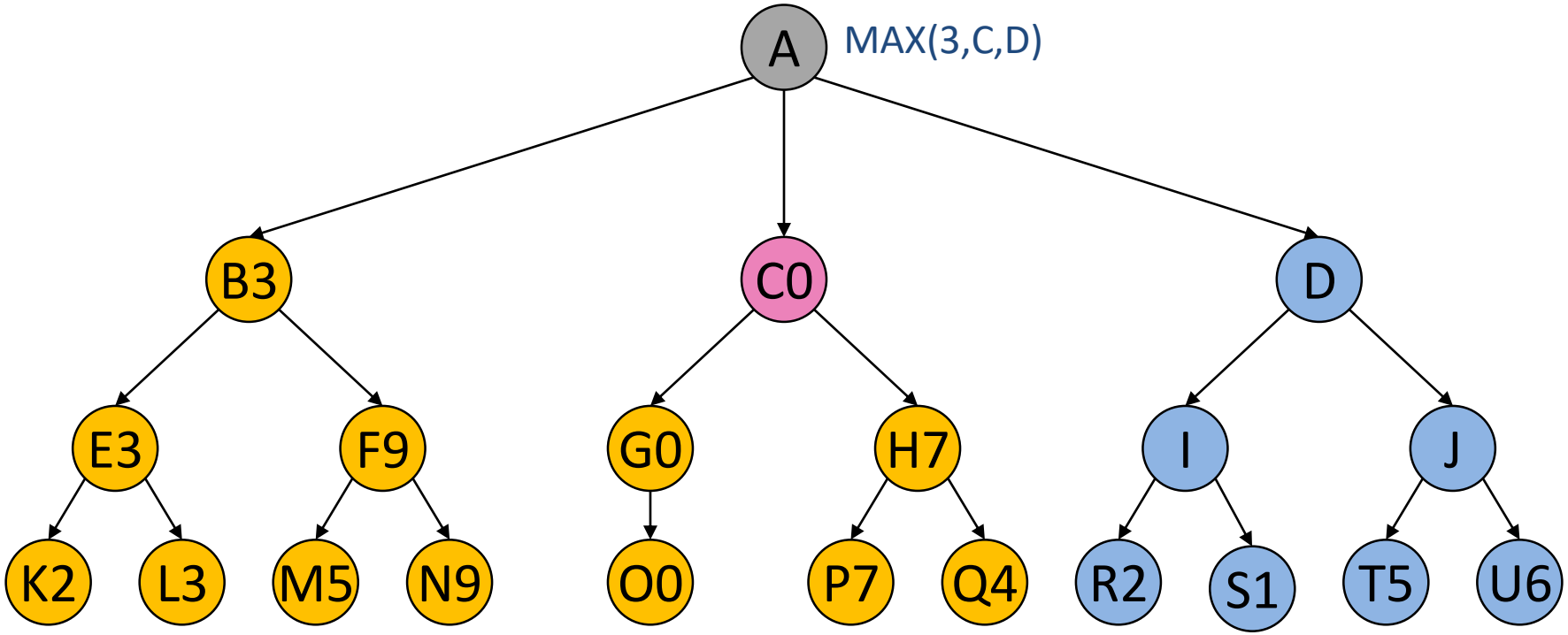
Min-Max algoritam (ilustracija)



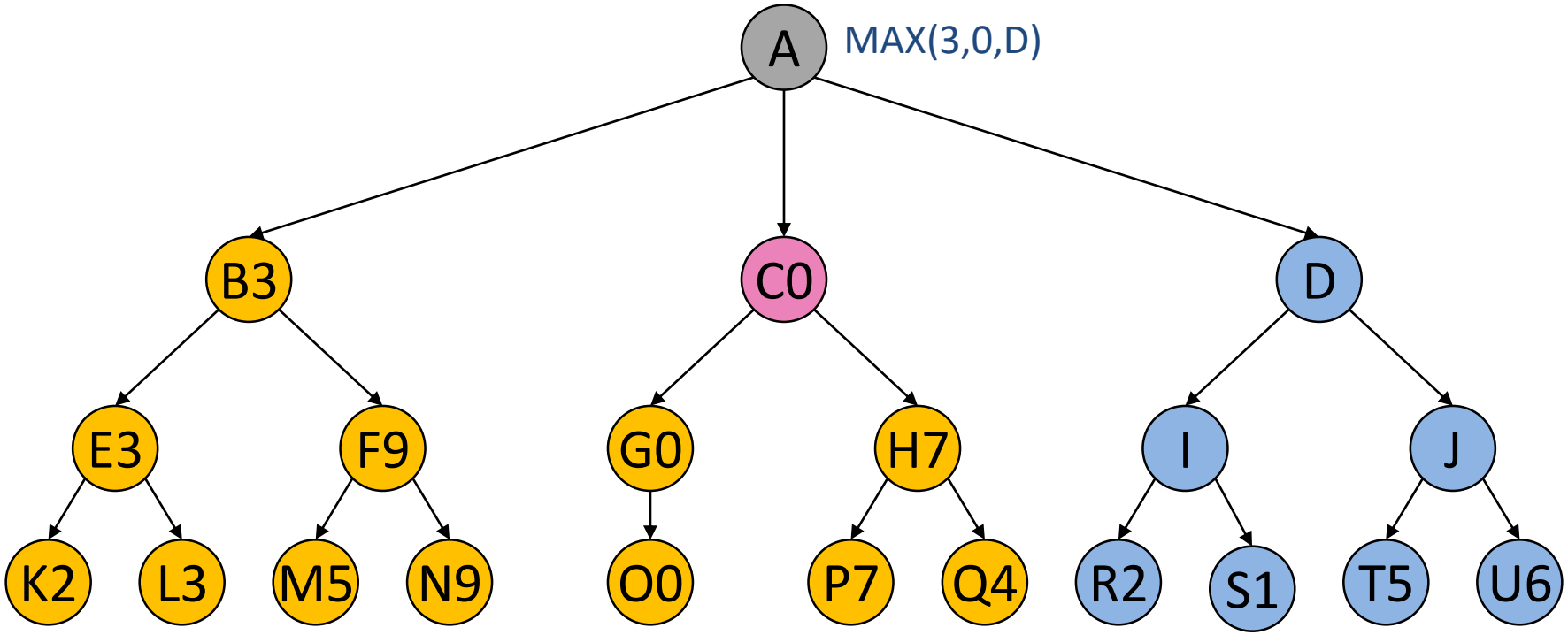
Min-Max algoritam (ilustracija)



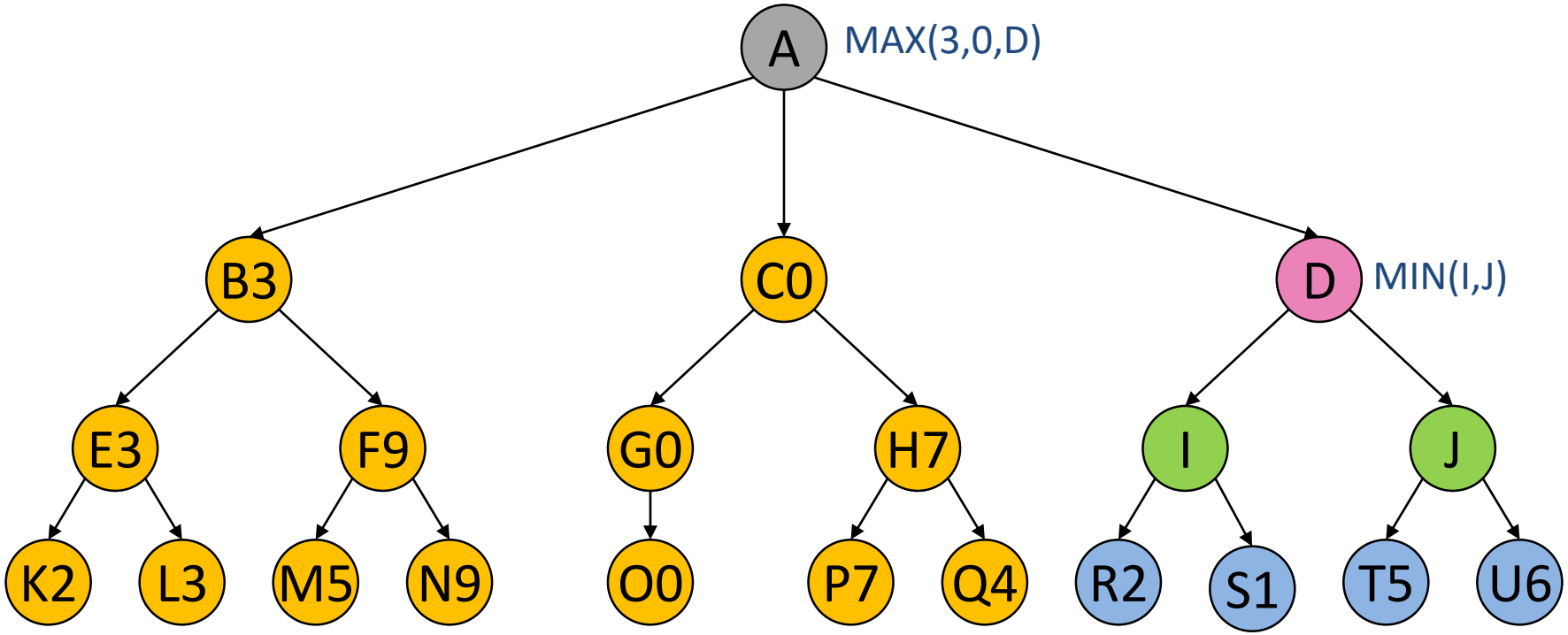
Min-Max algoritam (ilustracija)



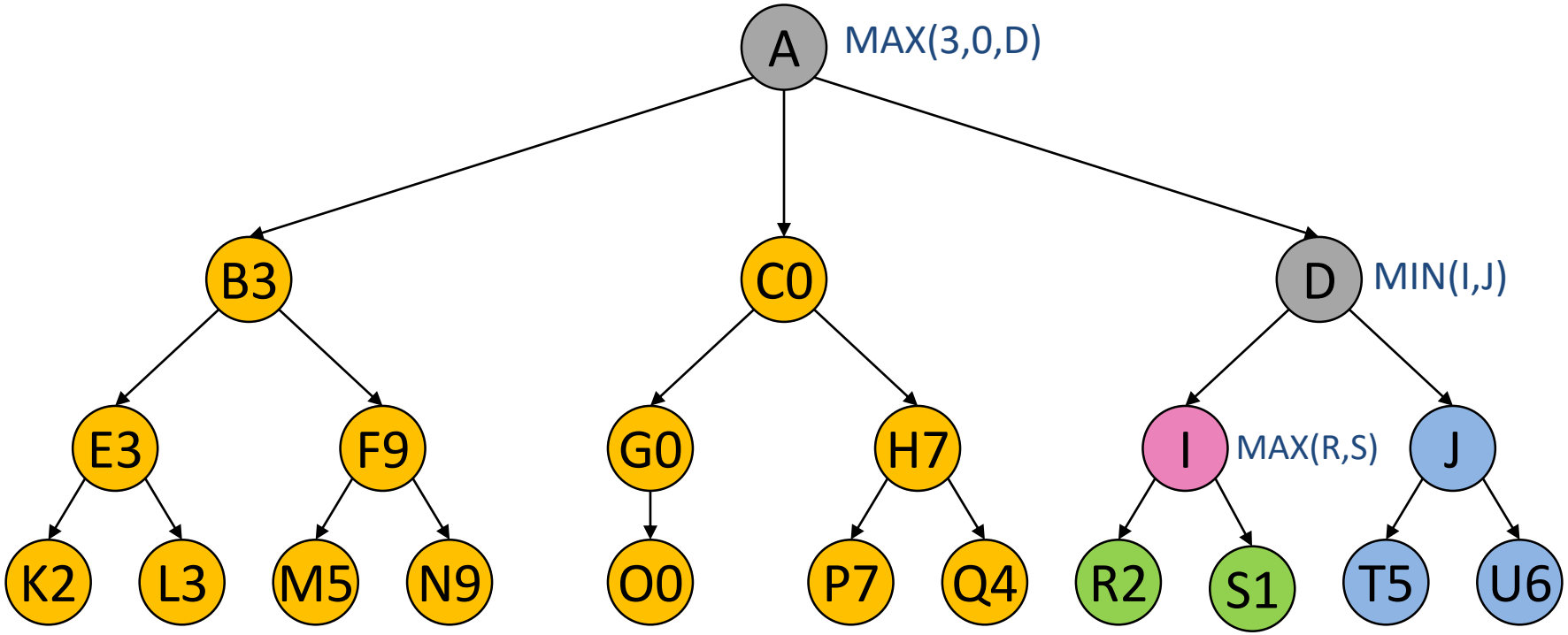
Min-Max algoritam (ilustracija)



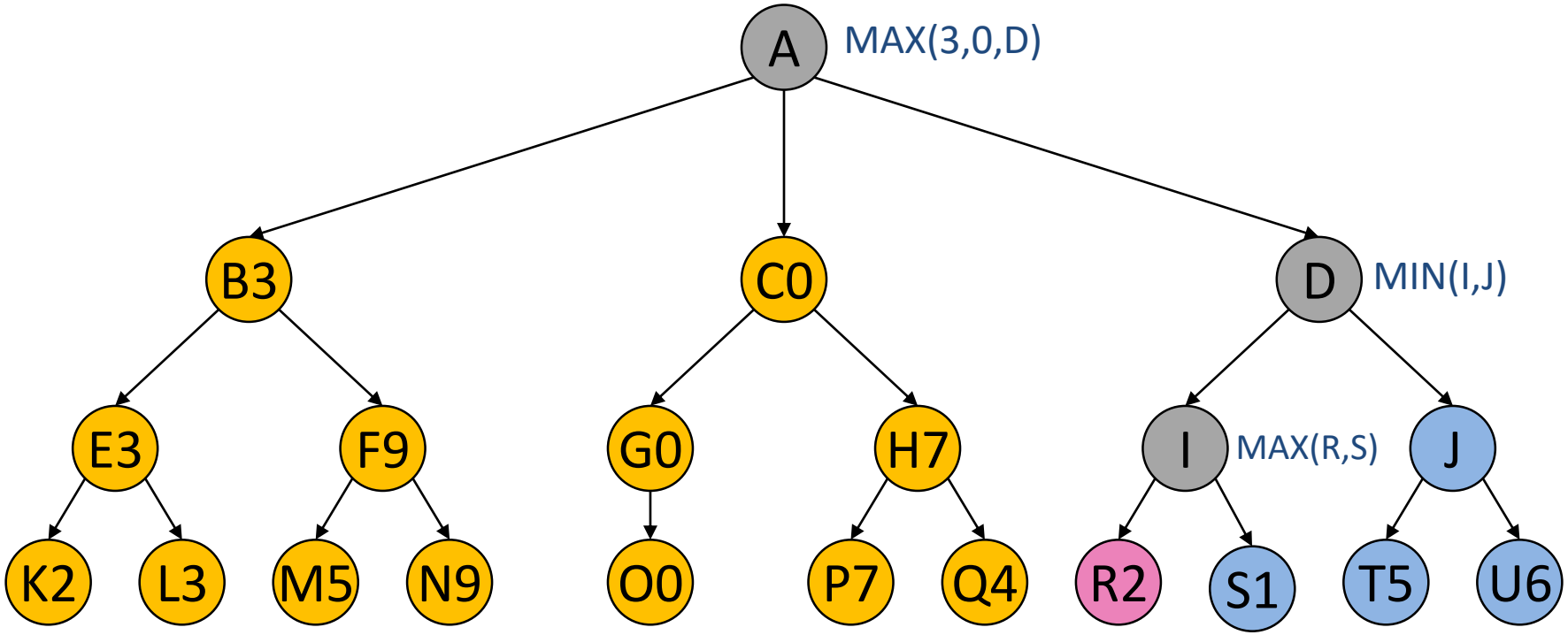
Min-Max algoritam (ilustracija)



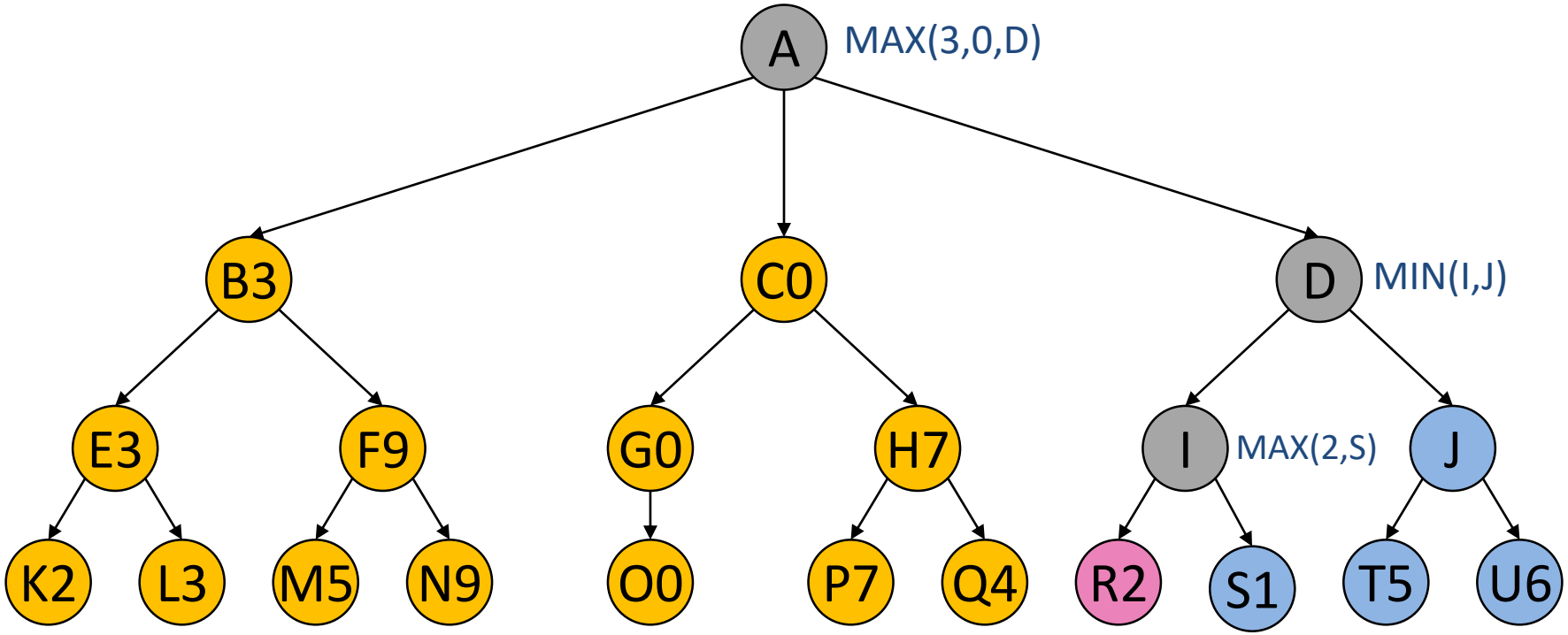
Min-Max algoritam (ilustracija)



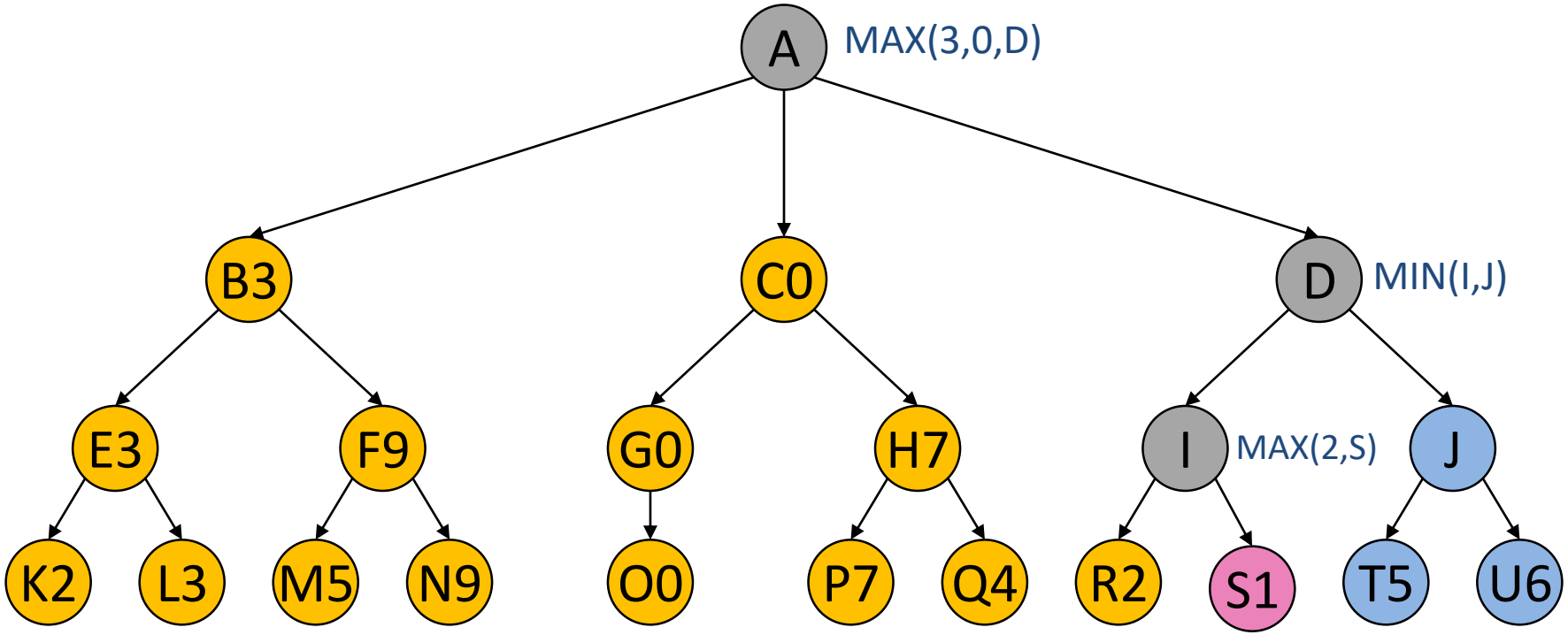
Min-Max algoritam (ilustracija)



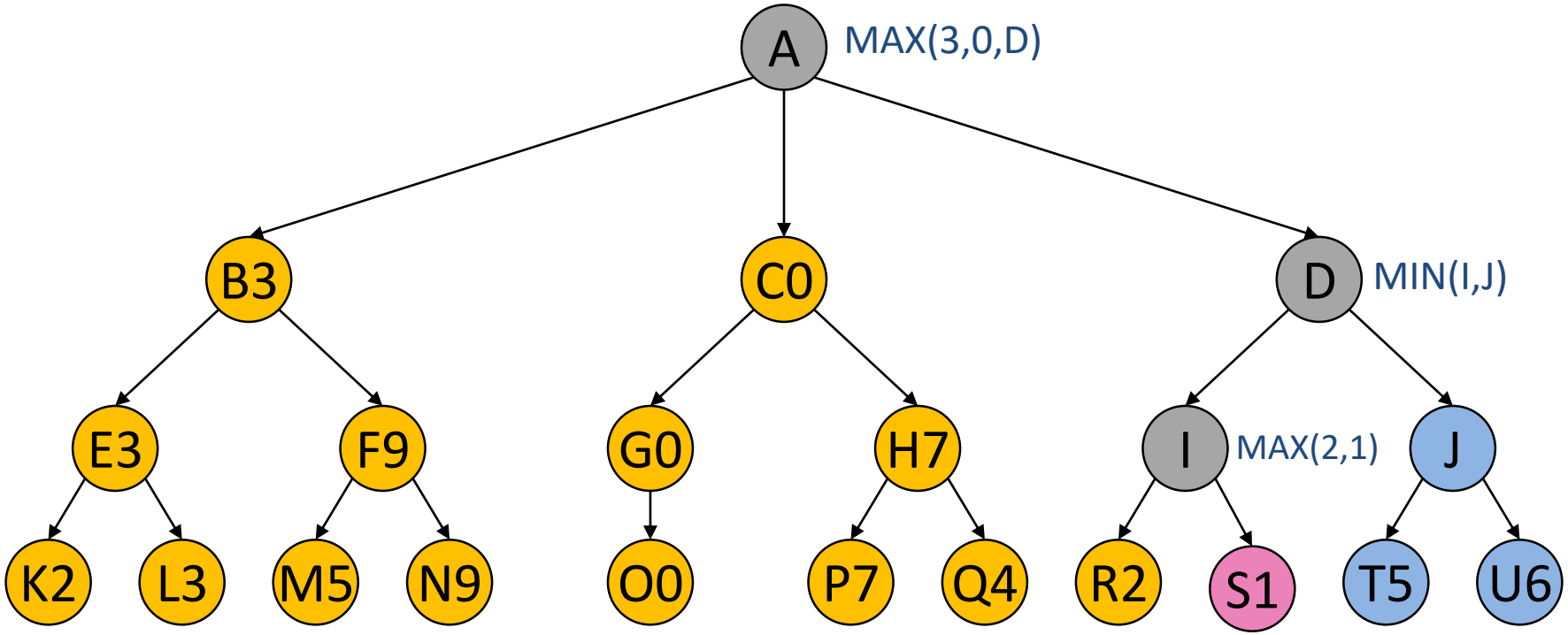
Min-Max algoritam (ilustracija)



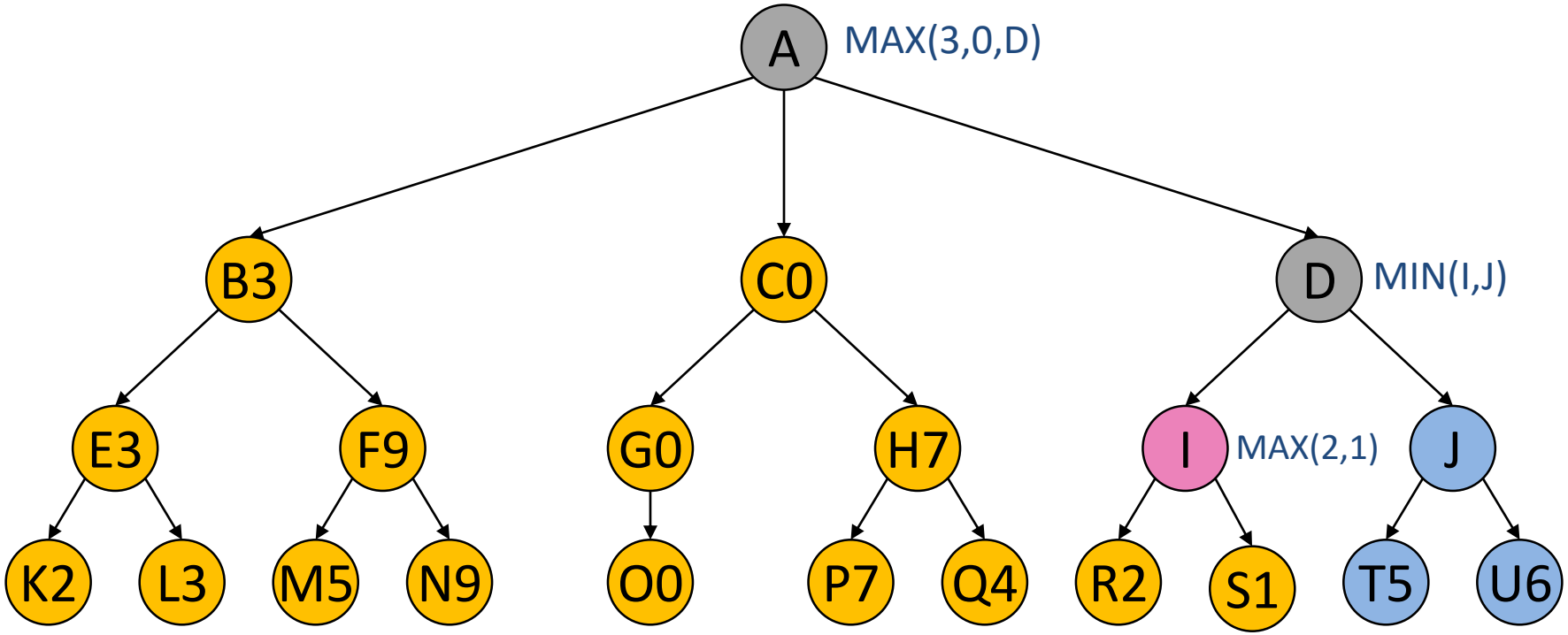
Min-Max algoritam (ilustracija)



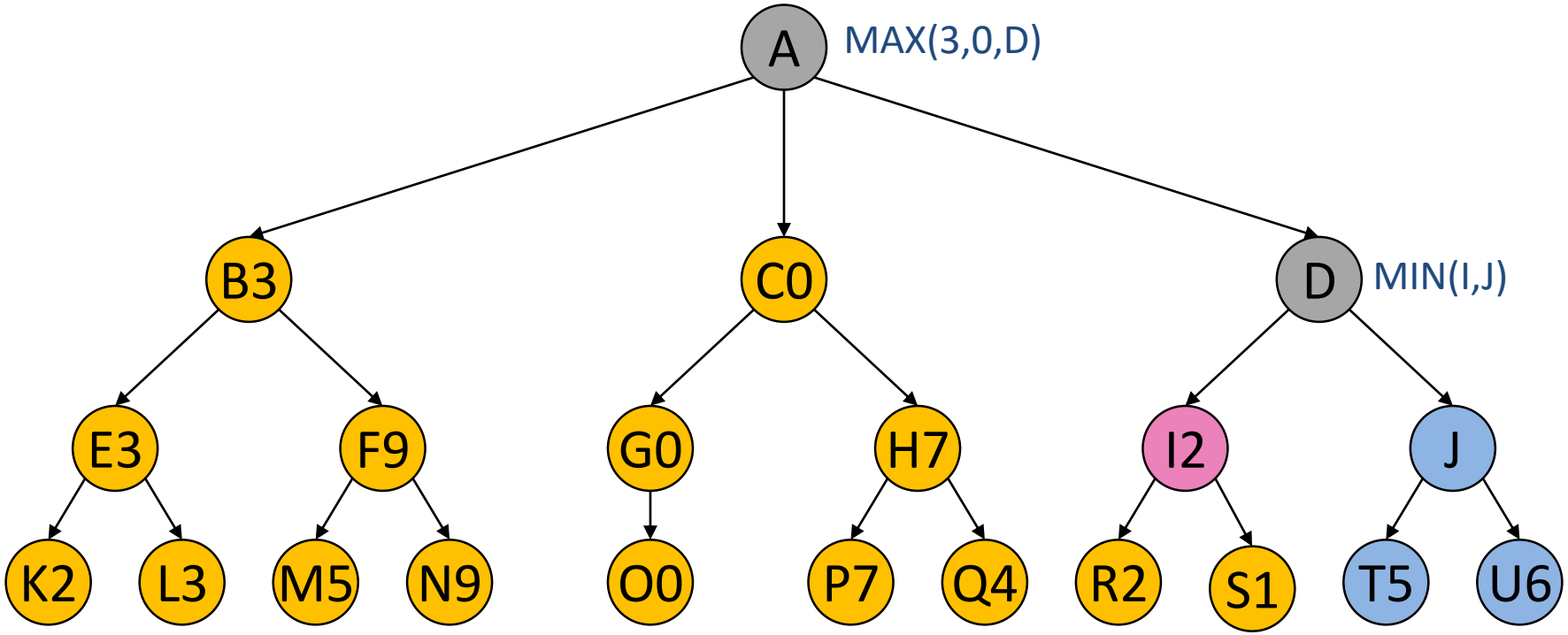
Min-Max algoritam (ilustracija)



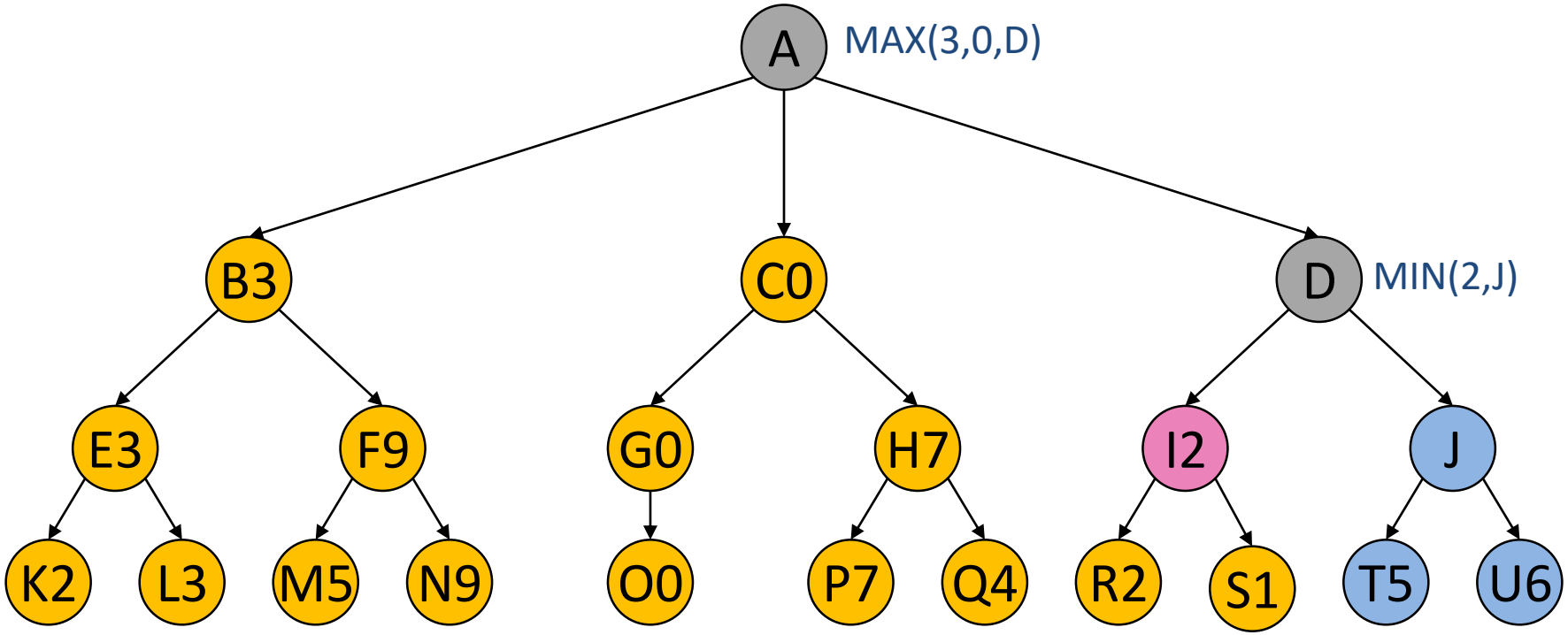
Min-Max algoritam (ilustracija)



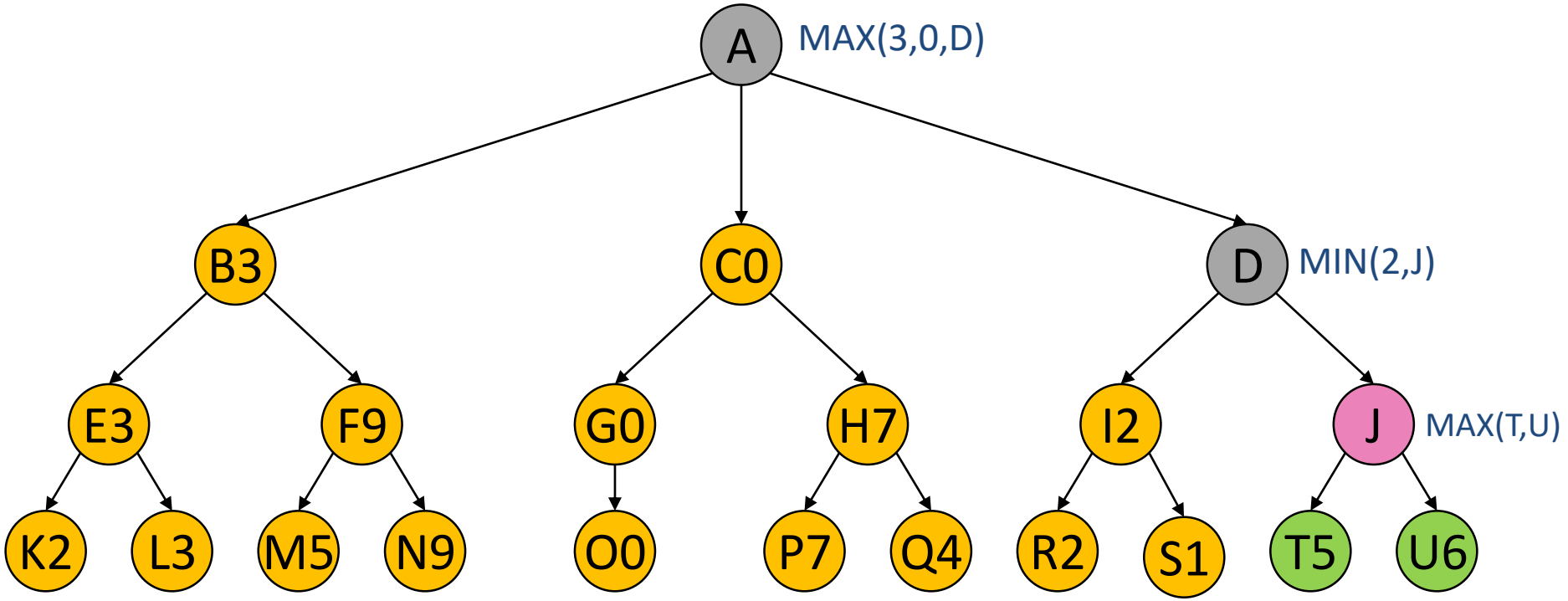
Min-Max algoritam (ilustracija)



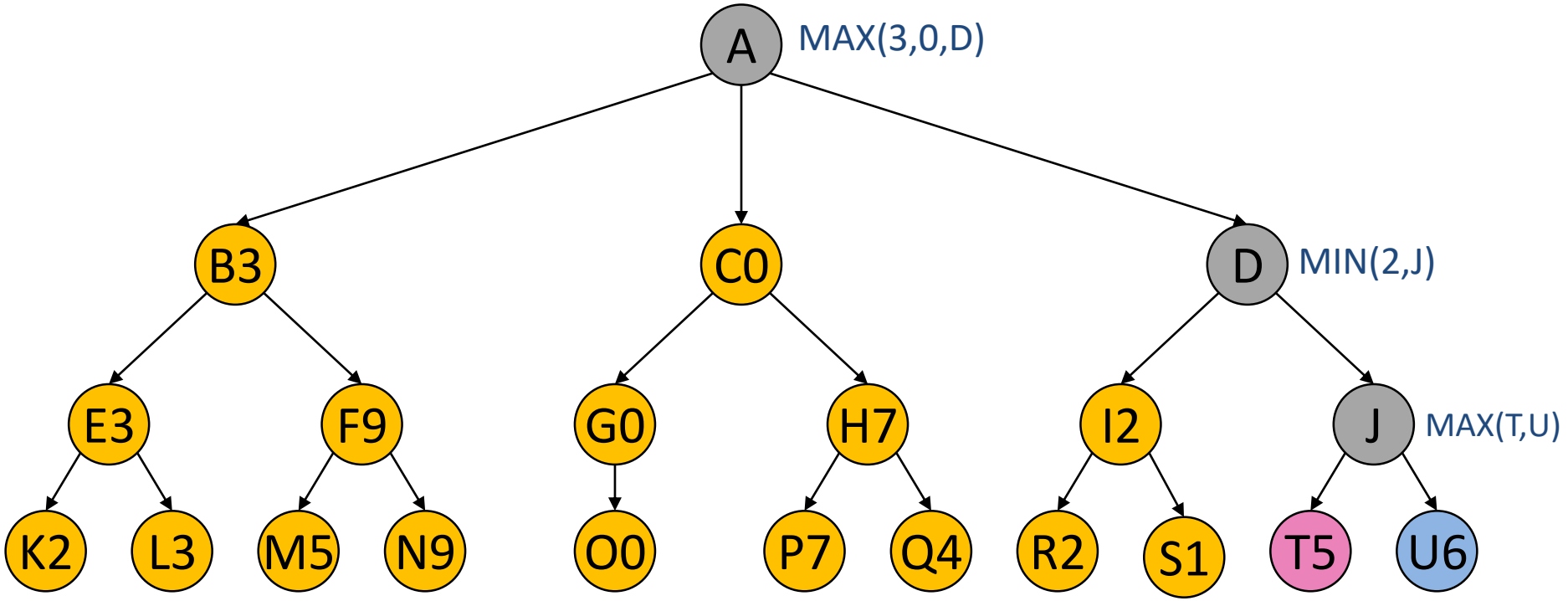
Min-Max algoritam (ilustracija)



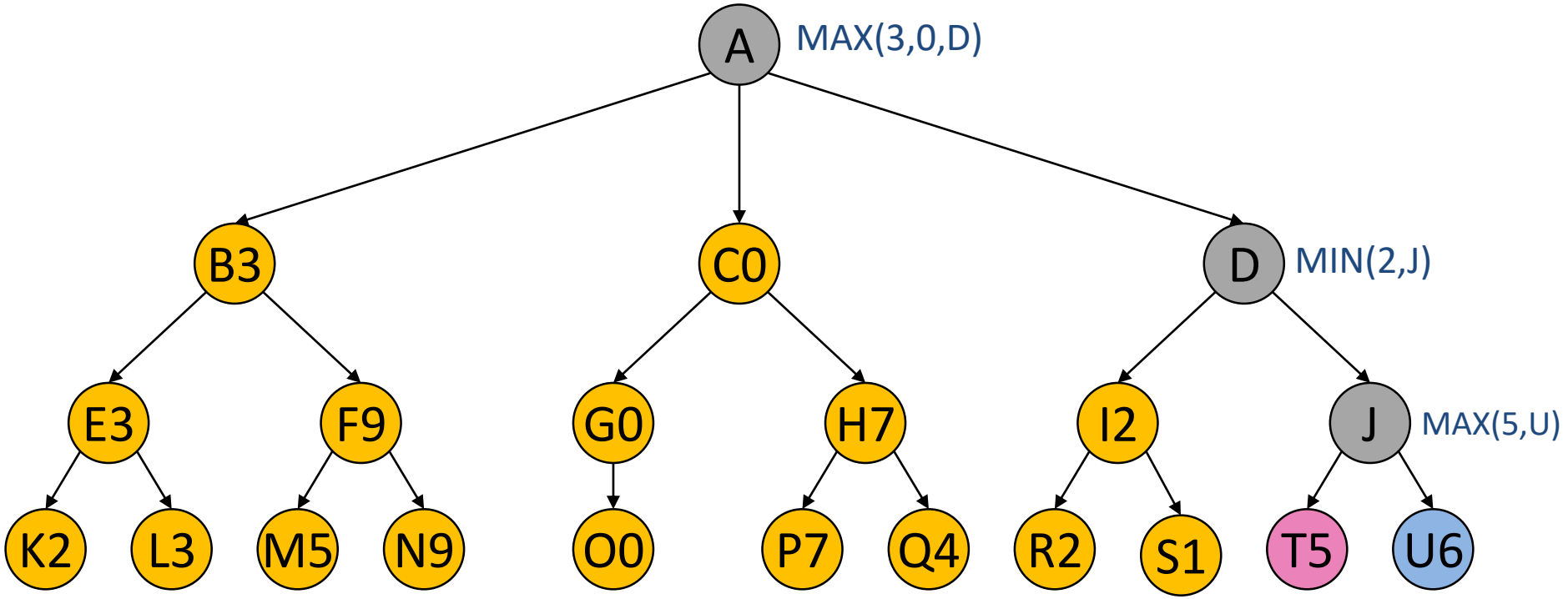
Min-Max algoritam (ilustracija)



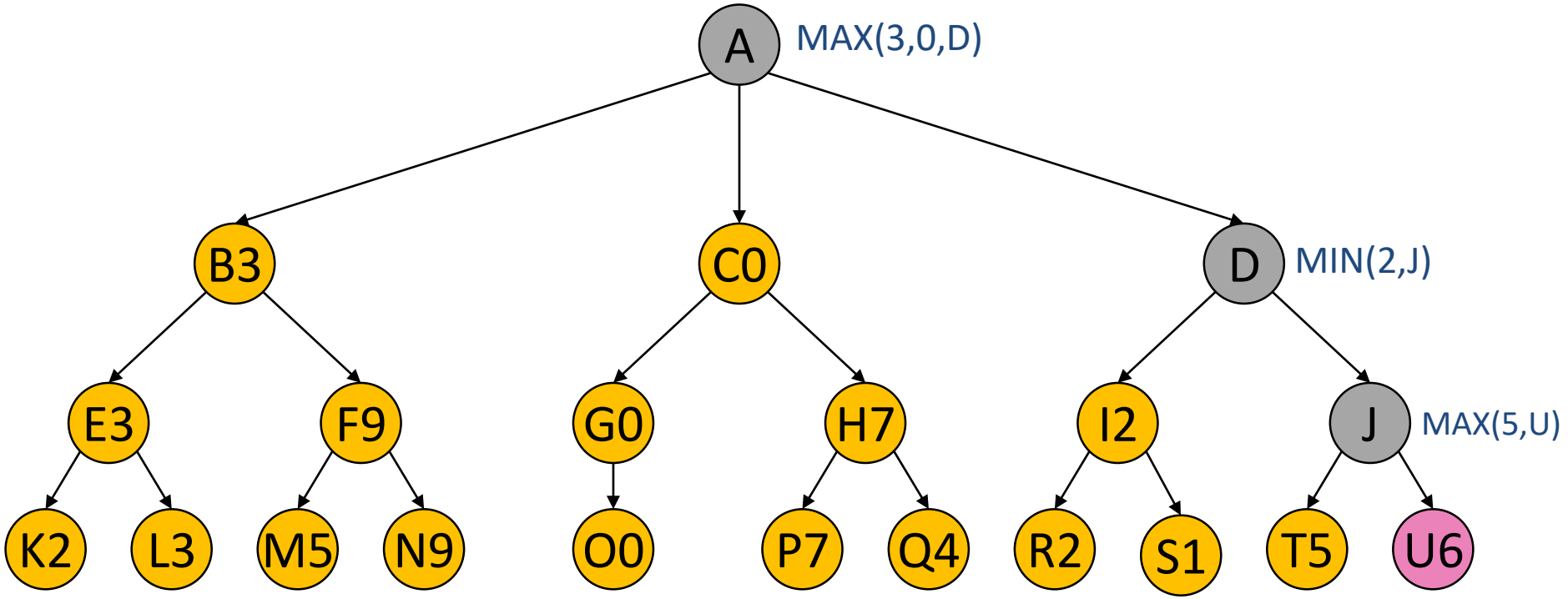
Min-Max algoritam (ilustracija)



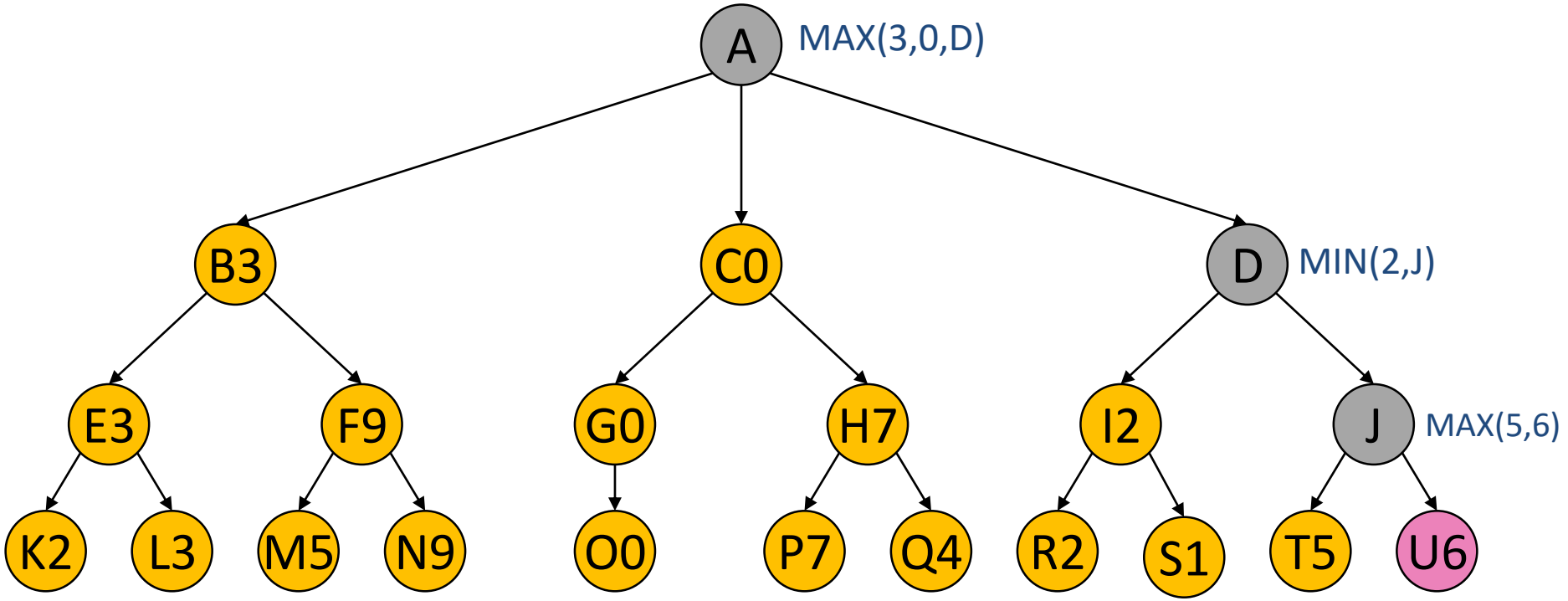
Min-Max algoritam (ilustracija)



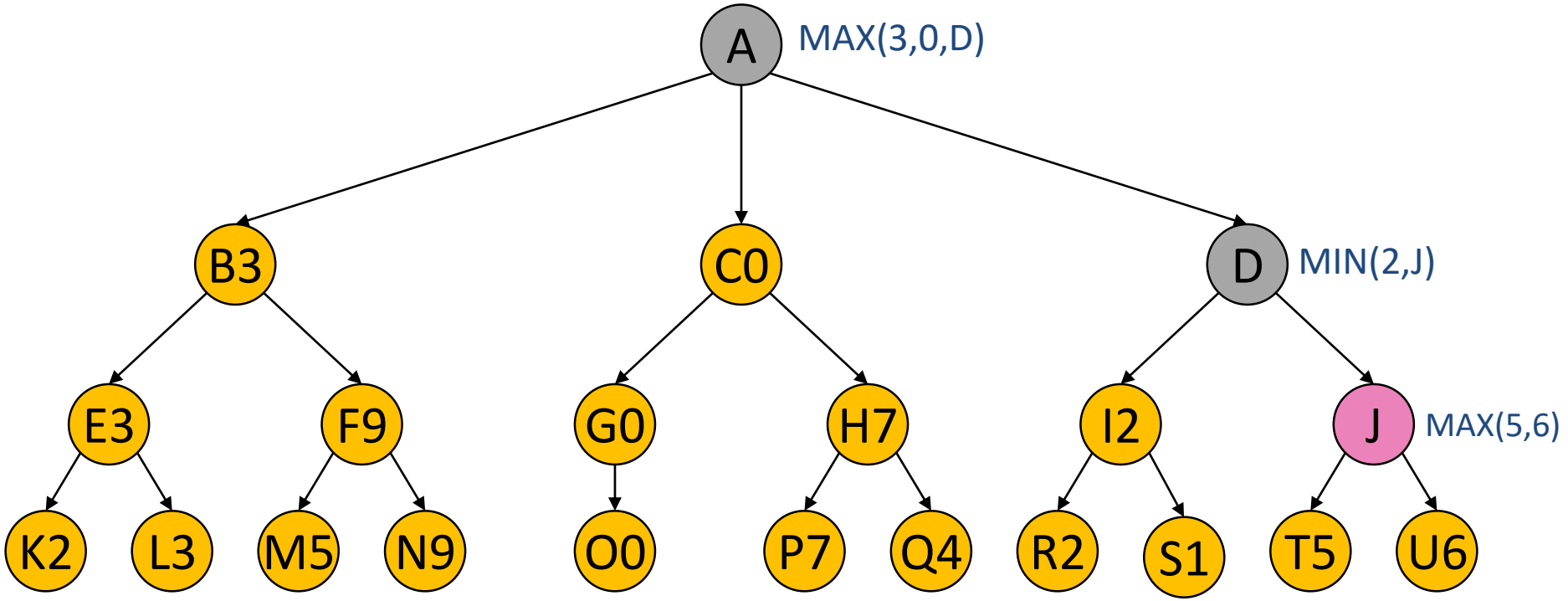
Min-Max algoritam (ilustracija)



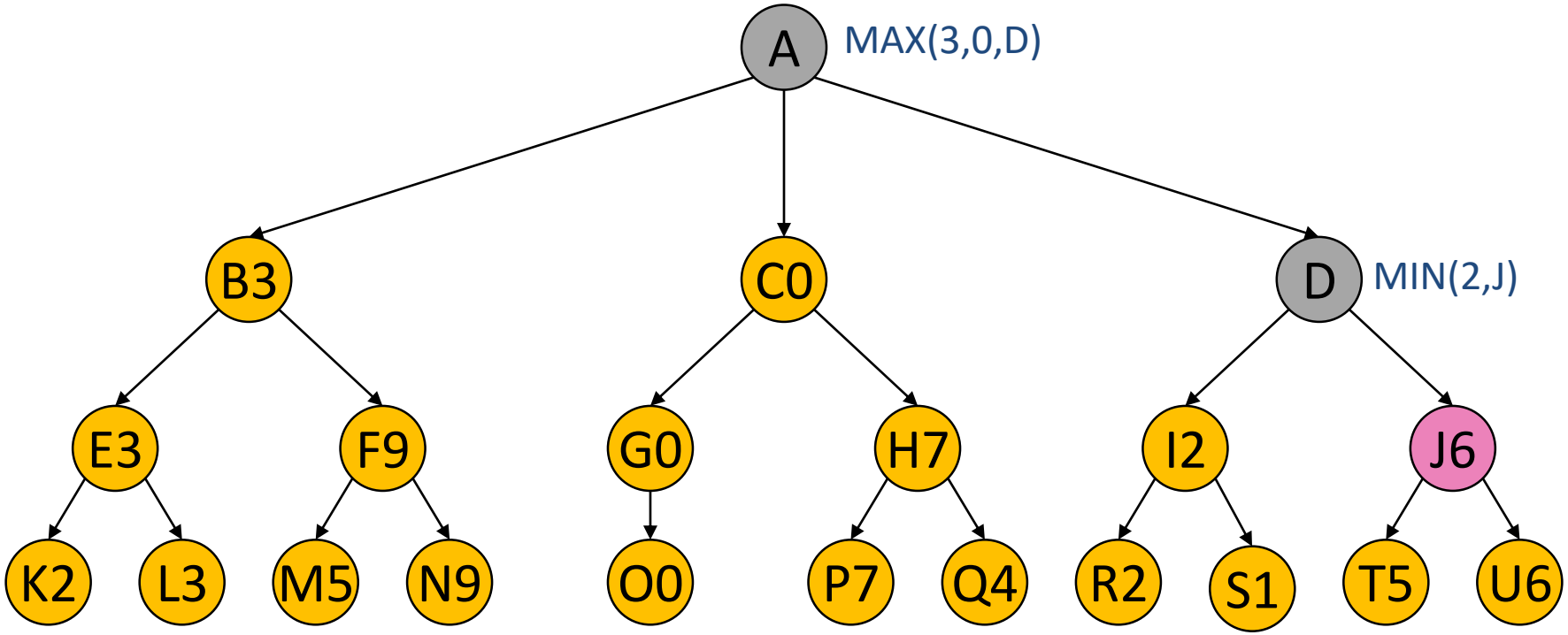
Min-Max algoritam (ilustracija)



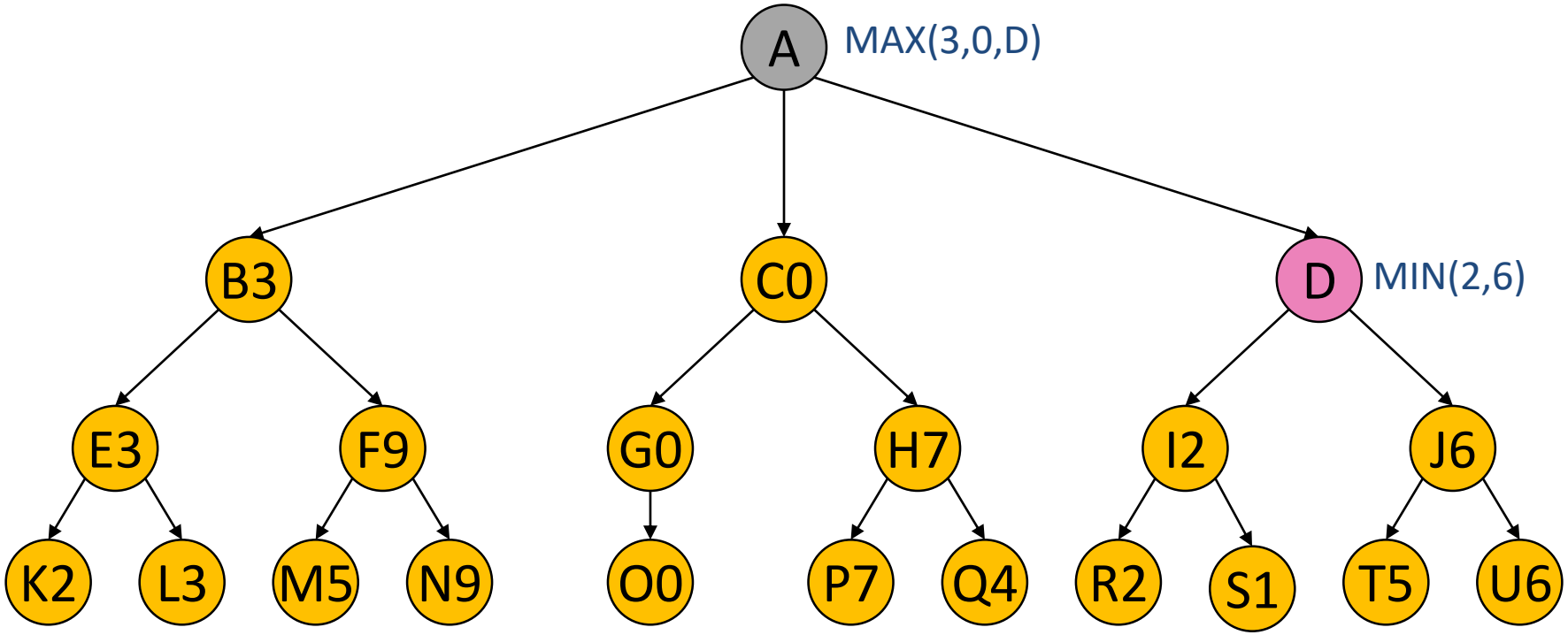
Min-Max algoritam (ilustracija)



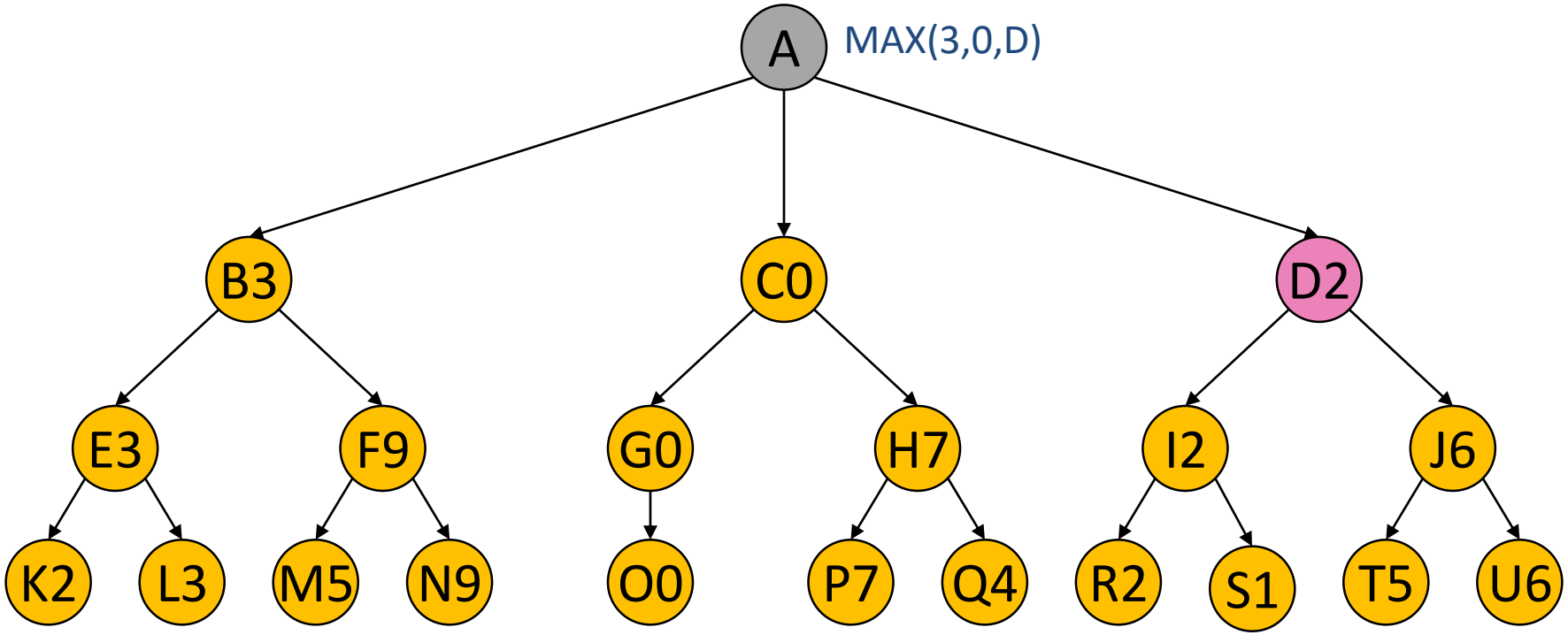
Min-Max algoritam (ilustracija)



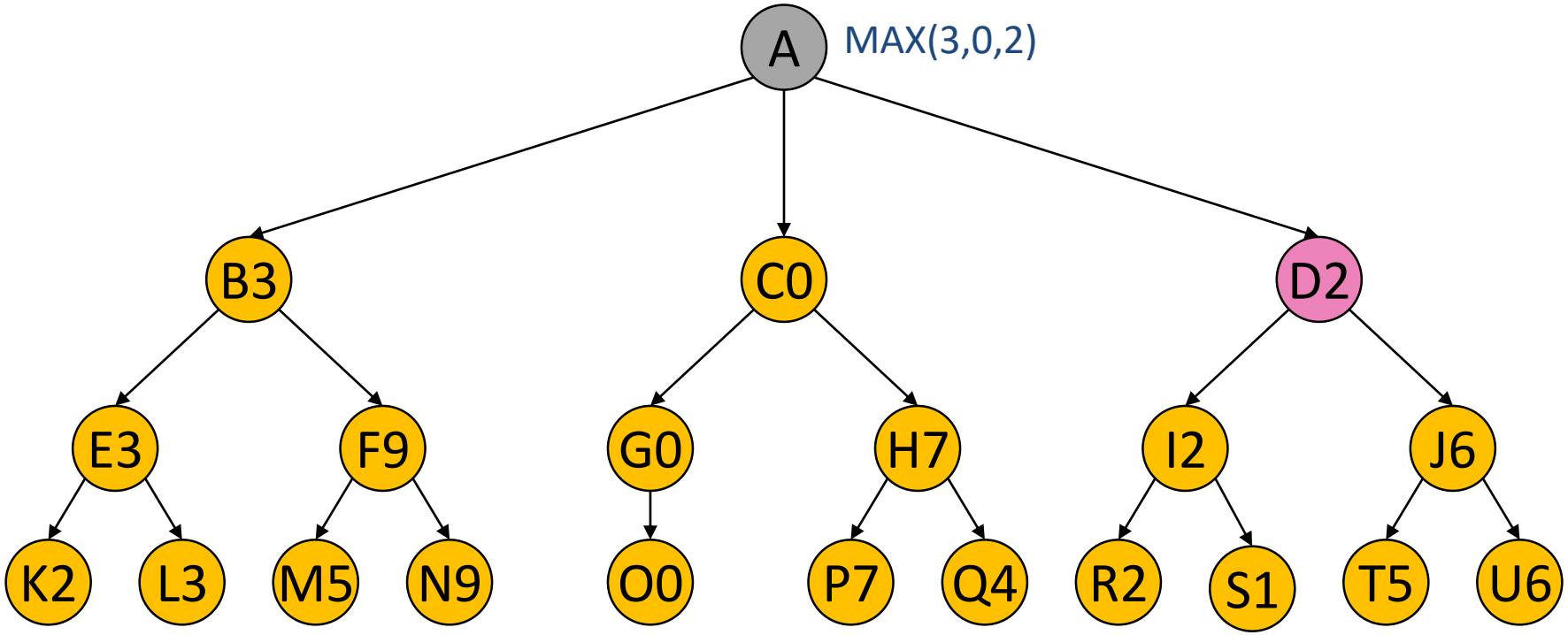
Min-Max algoritam (ilustracija)



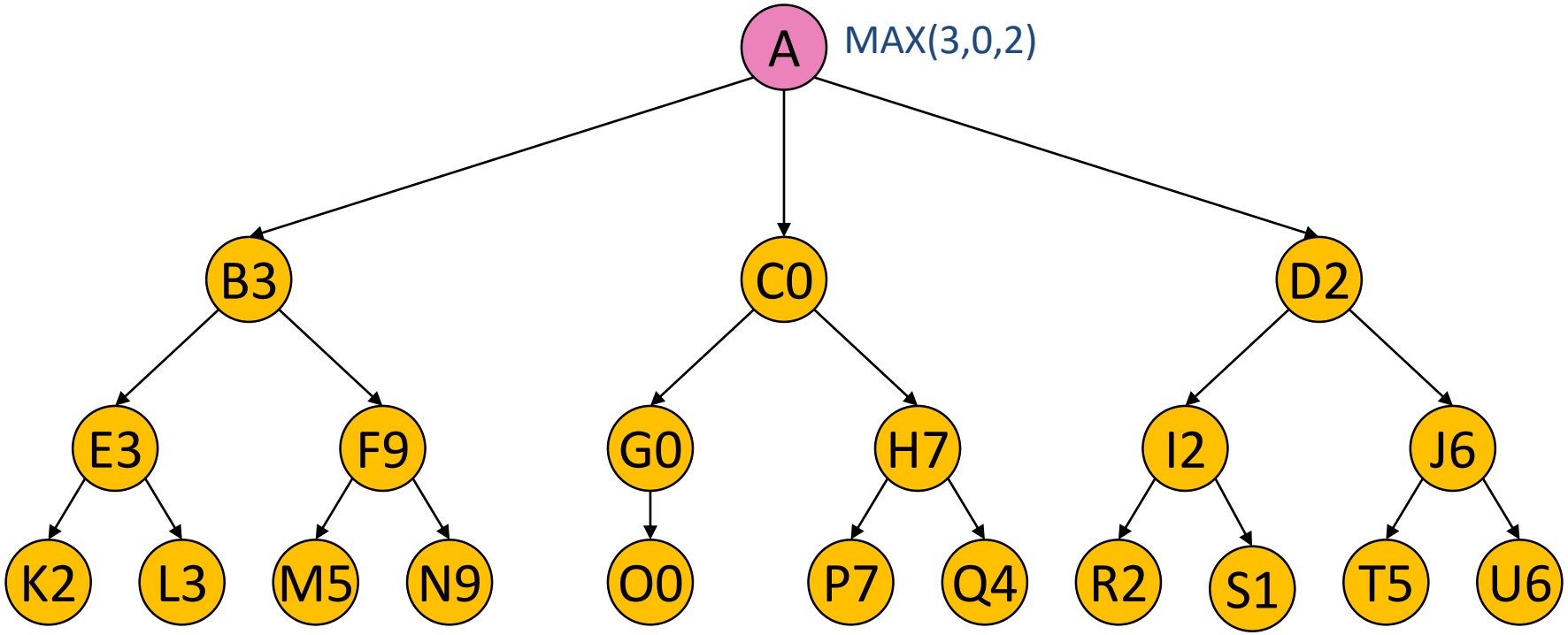
Min-Max algoritam (ilustracija)



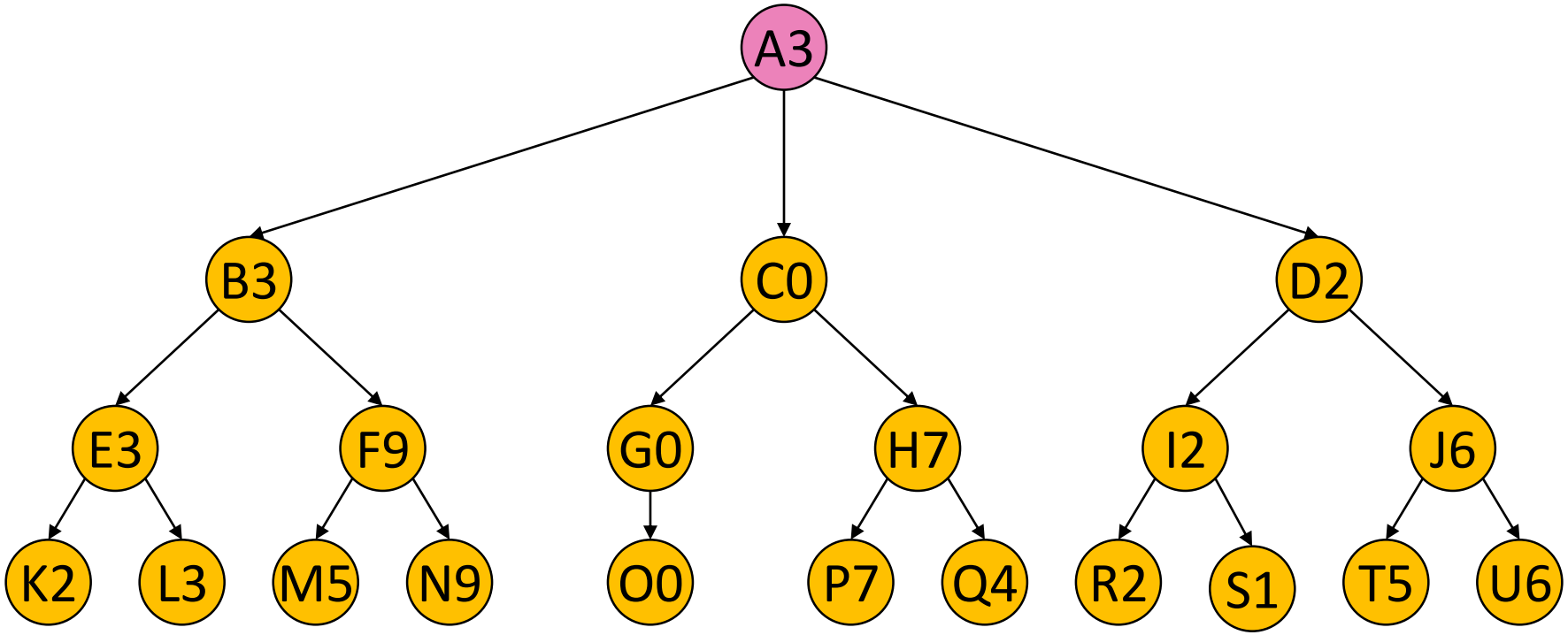
Min-Max algoritam (ilustracija)



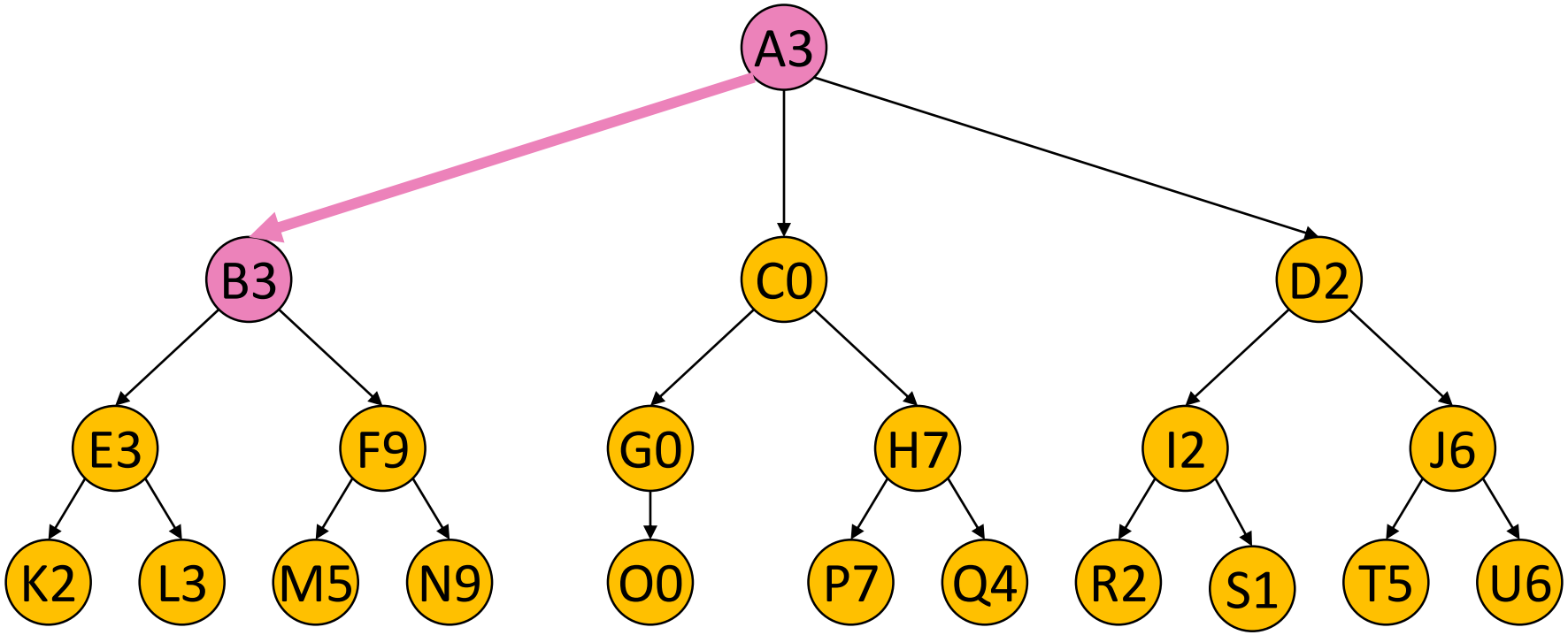
Min-Max algoritam (ilustracija)



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 (što je gore po Min), onda može da prestane sa traženjem (podrazumeva se da Min neće odigrati potez koji nije dobar za njega)

α-β odsecanje – funkcije

- Polazne vrednosti: α = -∞, β = +∞
- **max-value**(stanje, graf, α, β, dubina, maxdub)
- *Ako je* dubina == maxdub *onda vrati* **proceni**(stanje);
- *Za svako* S ∈ **sledbenici**(stanje)
- α := max(α, **min-value**(S, graf, α, β, dubina + 1, maxdub))
- *Ako je* α >= β *onda vrati* β;
- *Vrati* α;

- **min-value**(stanje, graf, α, β, dubina, maxdub)
- *Ako je* dubina == maxdub *onda vrati* **proceni**(stanje);
- *Za svako* S ∈ **sledbenici**(stanje)
- β := min(β, **max-value**(S, graf, α, β, dubina + 1, maxdub))
- *Ako je* β <= α *onda vrati* α;
- *Vrati* β

Funkcija *max_value*

```
def max_value(stanje, dubina, alpha, beta):
    lista_novih_stanja = nova_stanja(stanje)
    if dubina == 0 or lista_novih_stanja is None:
        return (stanje, proceni_stanje(stanje))
    else:
        for s in lista_novih_stanja:
            alpha = max(alpha,
                min_value(s, dubina - 1, alpha, beta),
                key = lambda x: x[1])
            if alpha[1] >= beta[1]:
                return beta
    return alpha
```

Funkcija *min_value*

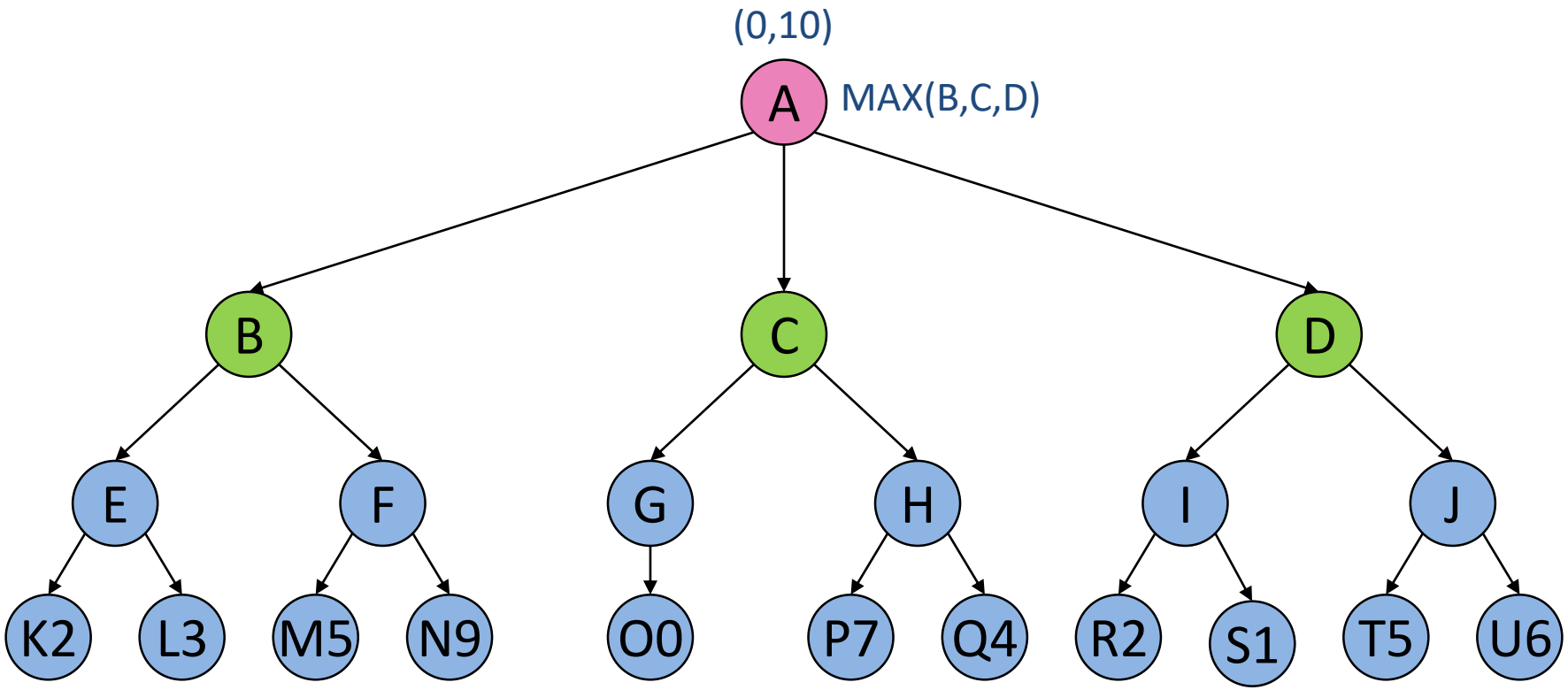
```
def min_value(stanje, dubina, alpha, beta):
    lista_novih_stanja = nova_stanja(stanje)
    if dubina == 0 or lista_novih_stanja is None:
        return (stanje, proceni_stanje(stanje))
    else:
        for s in lista_novih_stanja:
            beta = min(beta,
                max_value(s, dubina - 1, alpha, beta),
                key = lambda x: x[1])
            if beta[1] <= alpha[1]:
                return alpha
    return beta
```

Funkcija *minimax*

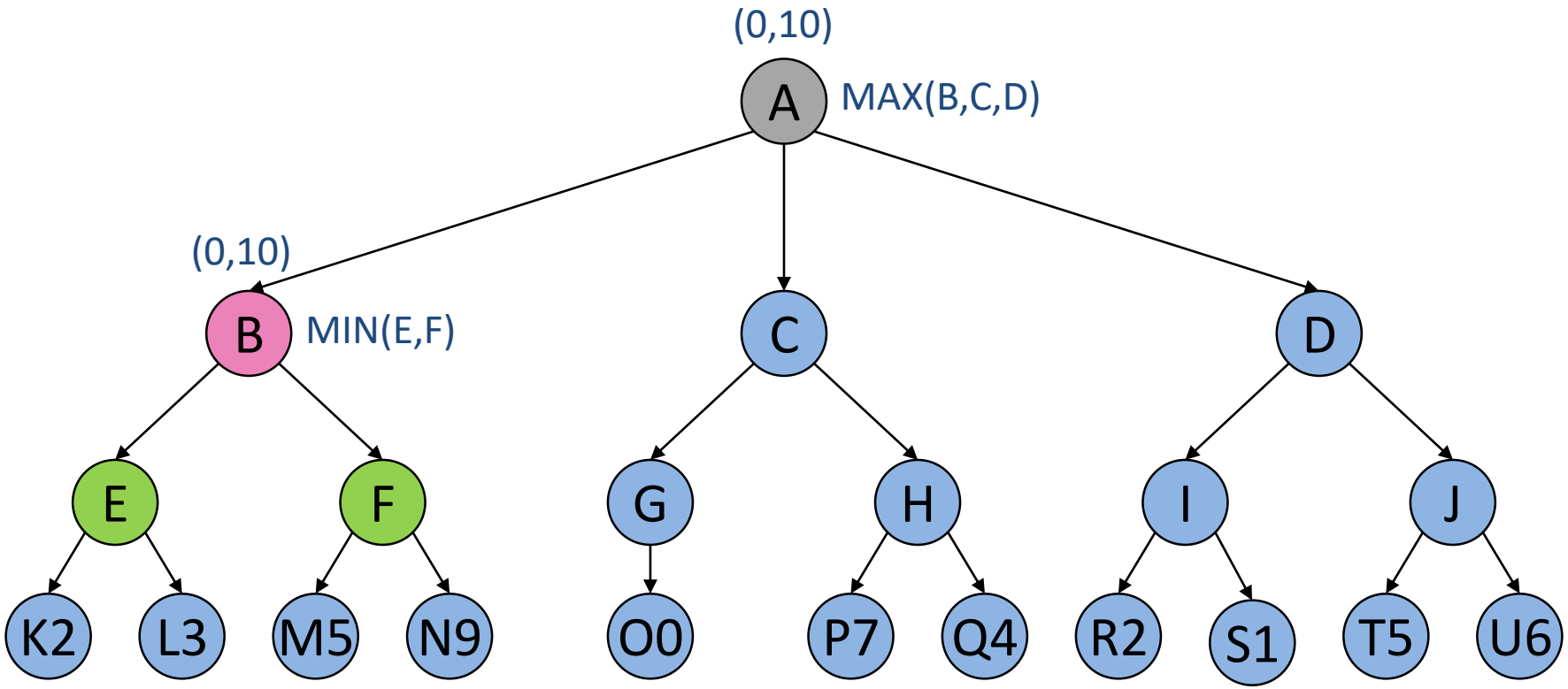
```
def minimax(stanje, dubina, moj_potez, alpha = (A, 0), beta = (A, 10)):
    if moj_potez:
        return max_value(stanje, dubina, alpha, beta)
    else:
        return min_value(stanje, dubina, alpha, beta)
```

- minimax funkcija u ovom slučaju služi samo kao „proxy“ funkcija
- U zavisnosti od informacije ko je na potezu, ona samo poziva odgovarajuću funkciju
- 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

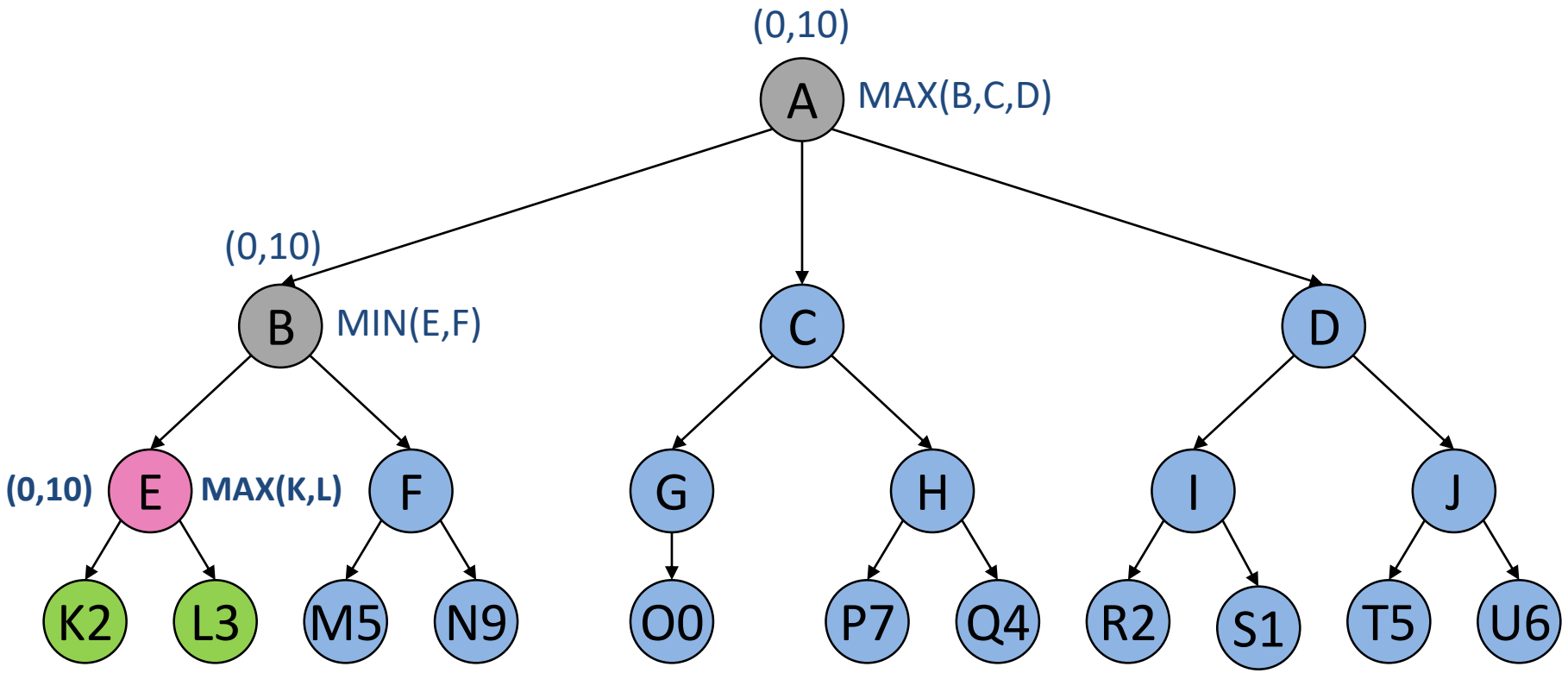
α-β odsecanje (ilustracija)



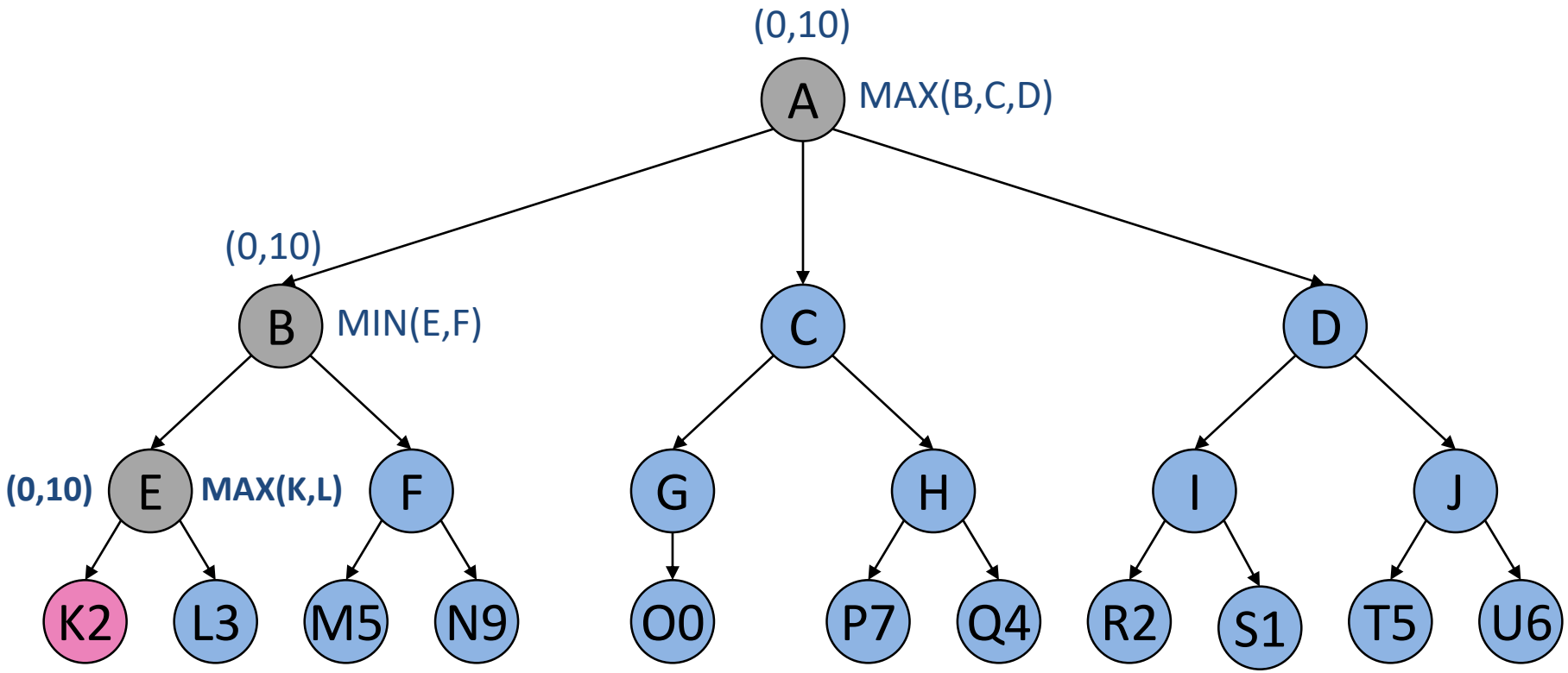
α-β odsecanje (ilustracija)



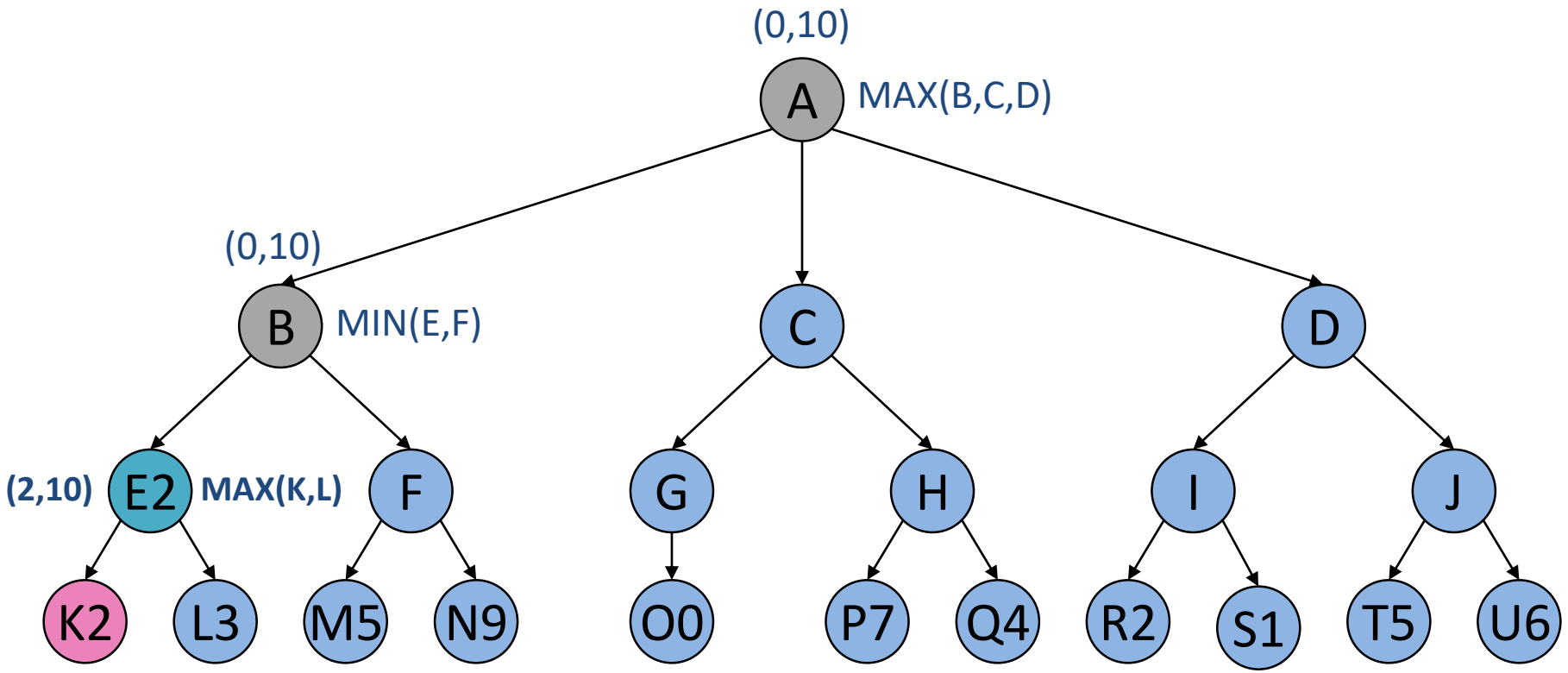
α-β odsecanje (ilustracija)



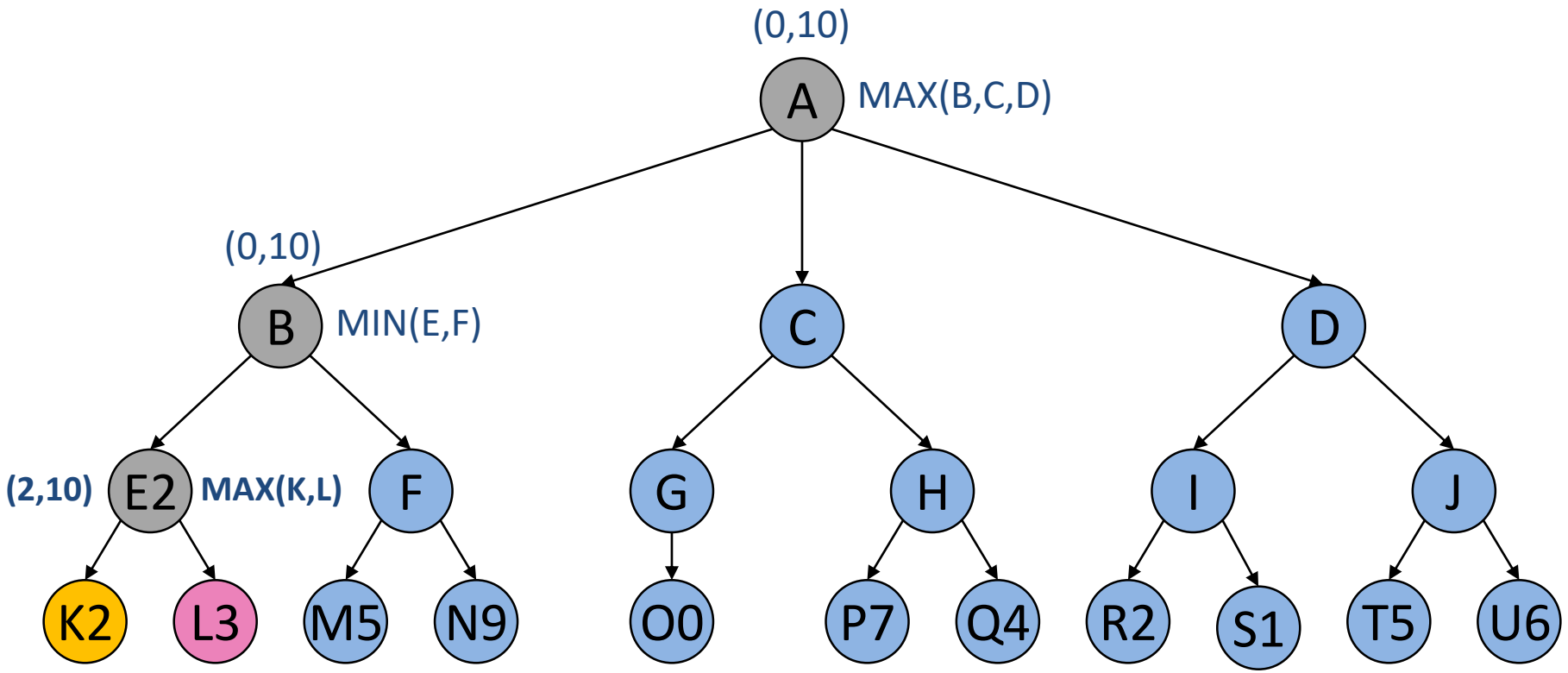
α-β odsecanje (ilustracija)



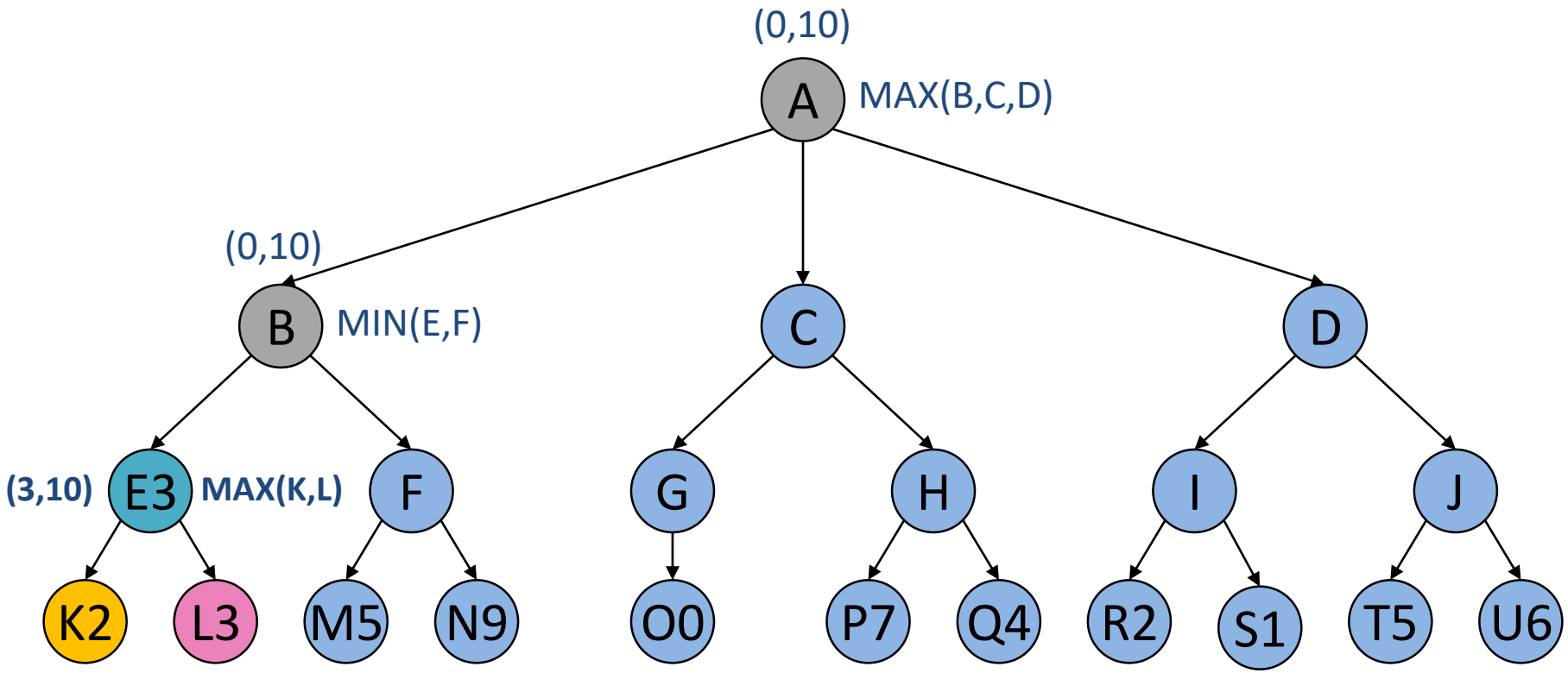
α-β odsecanje (ilustracija)



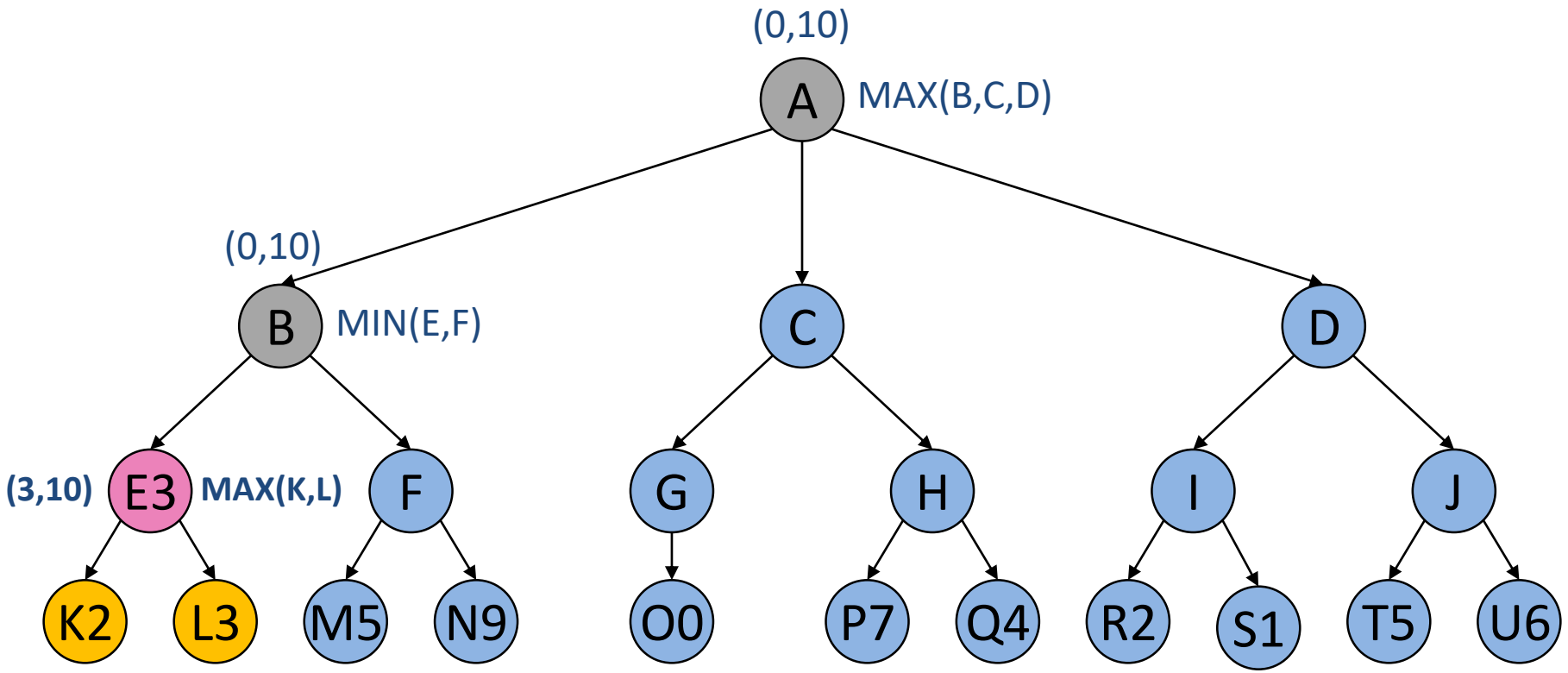
α-β odsecanje (ilustracija)



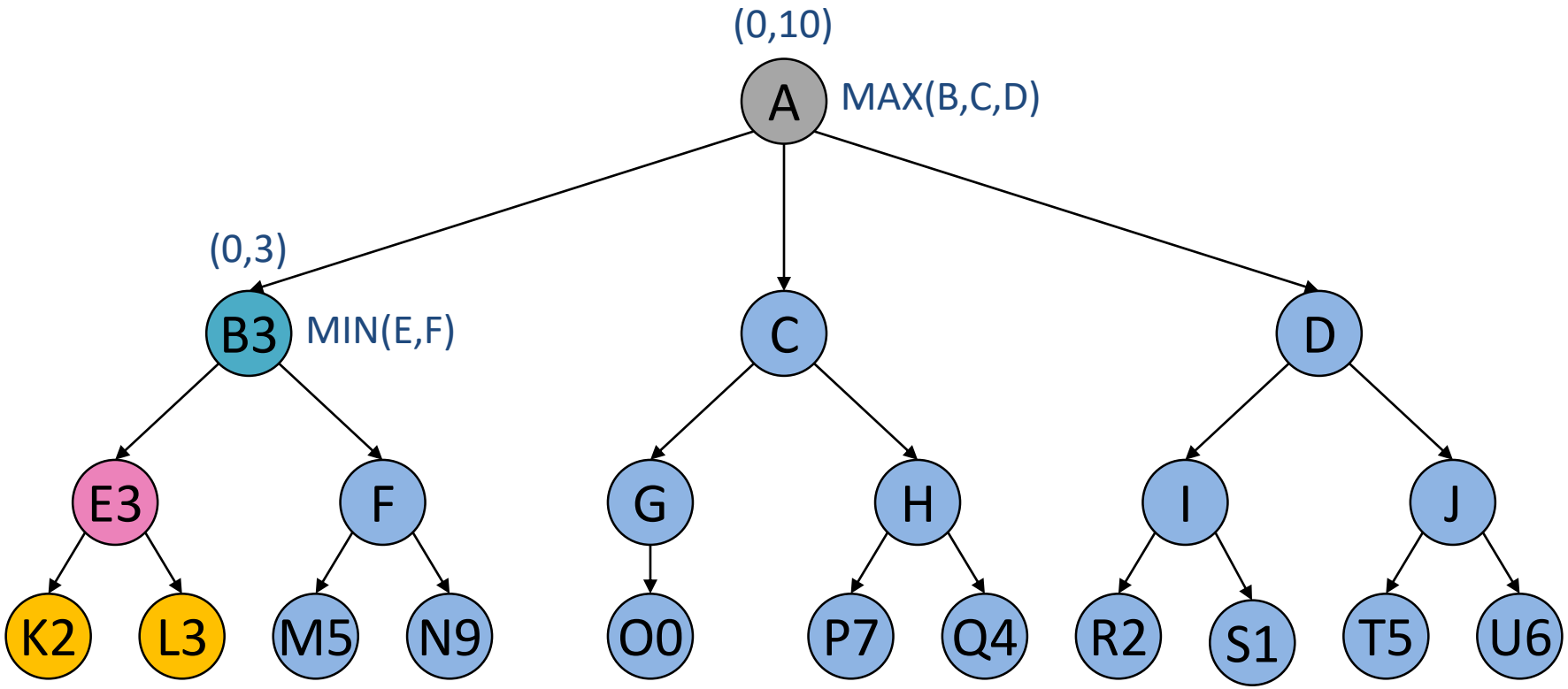
α-β odsecanje (ilustracija)



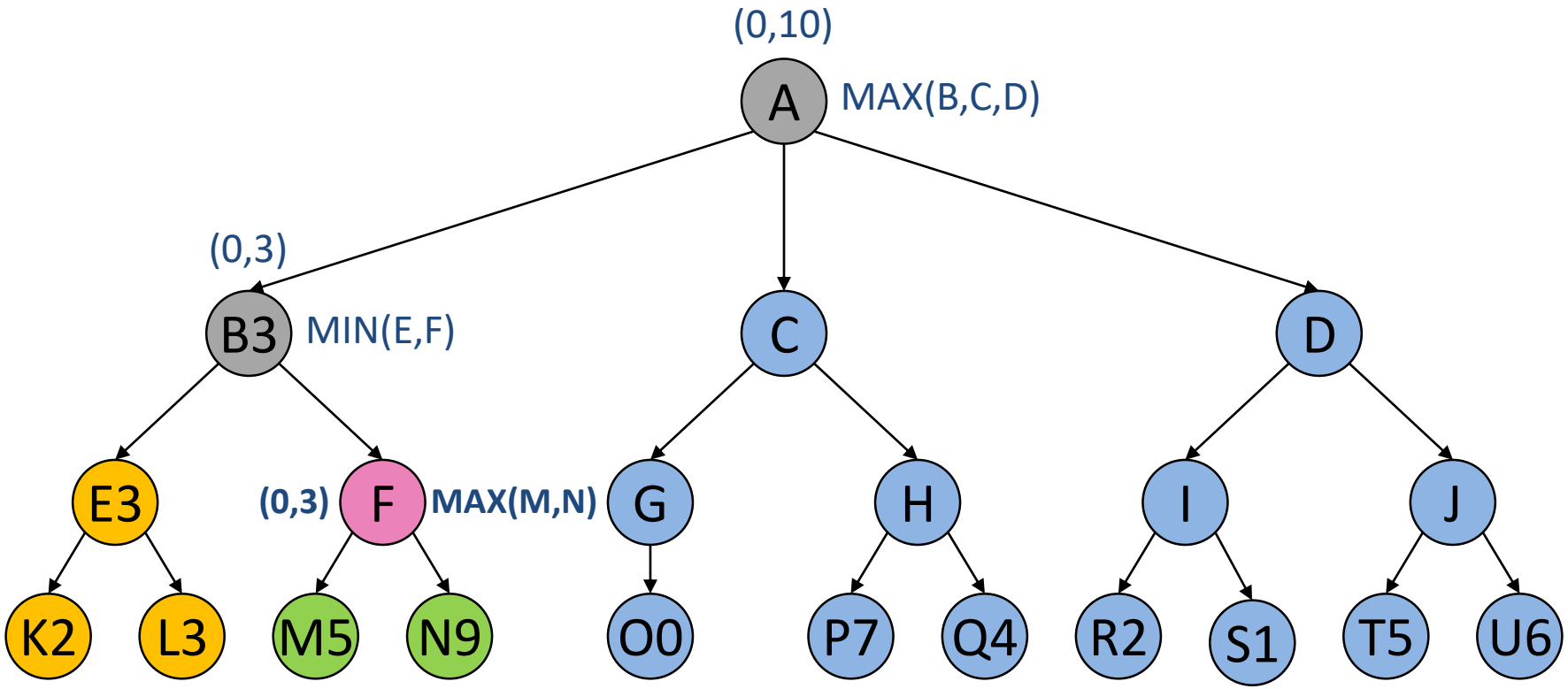
α-β odsecanje (ilustracija)



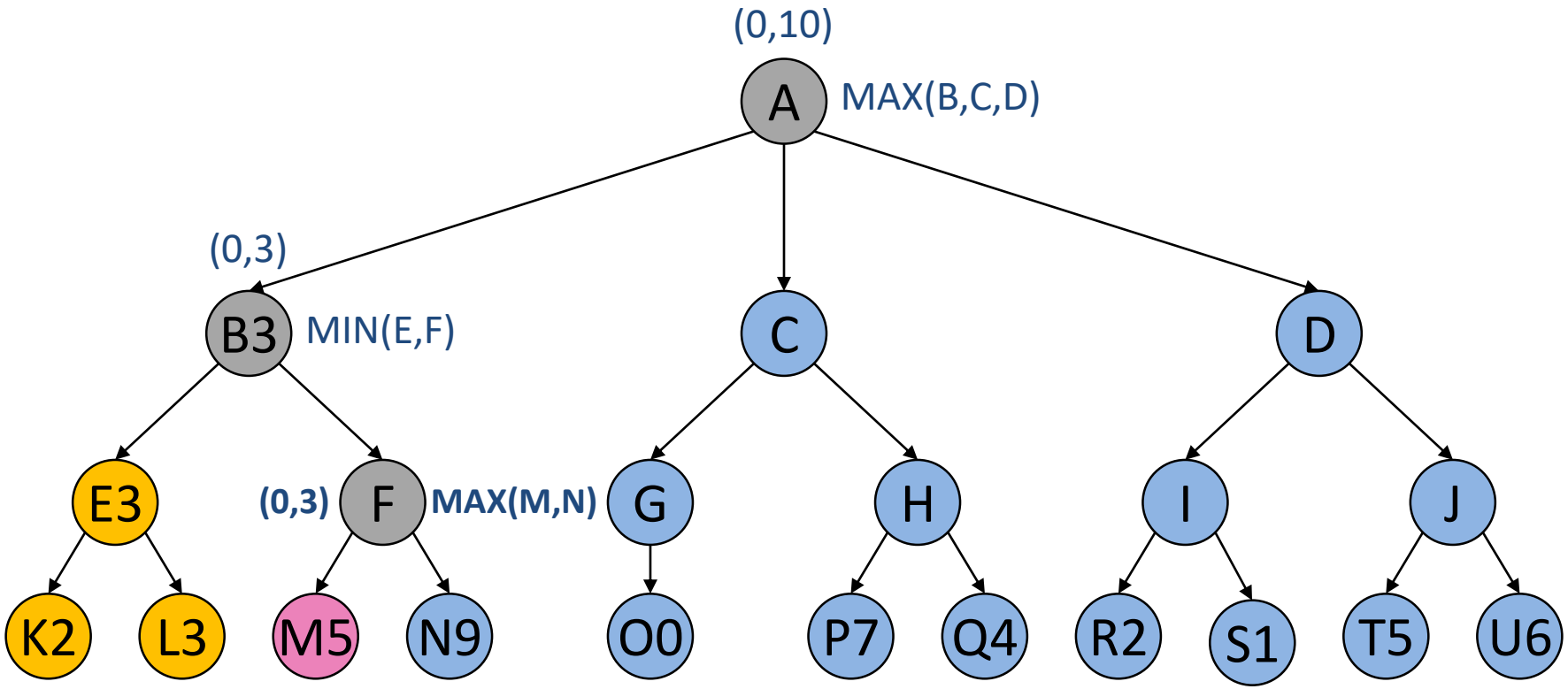
α-β odsecanje (ilustracija)



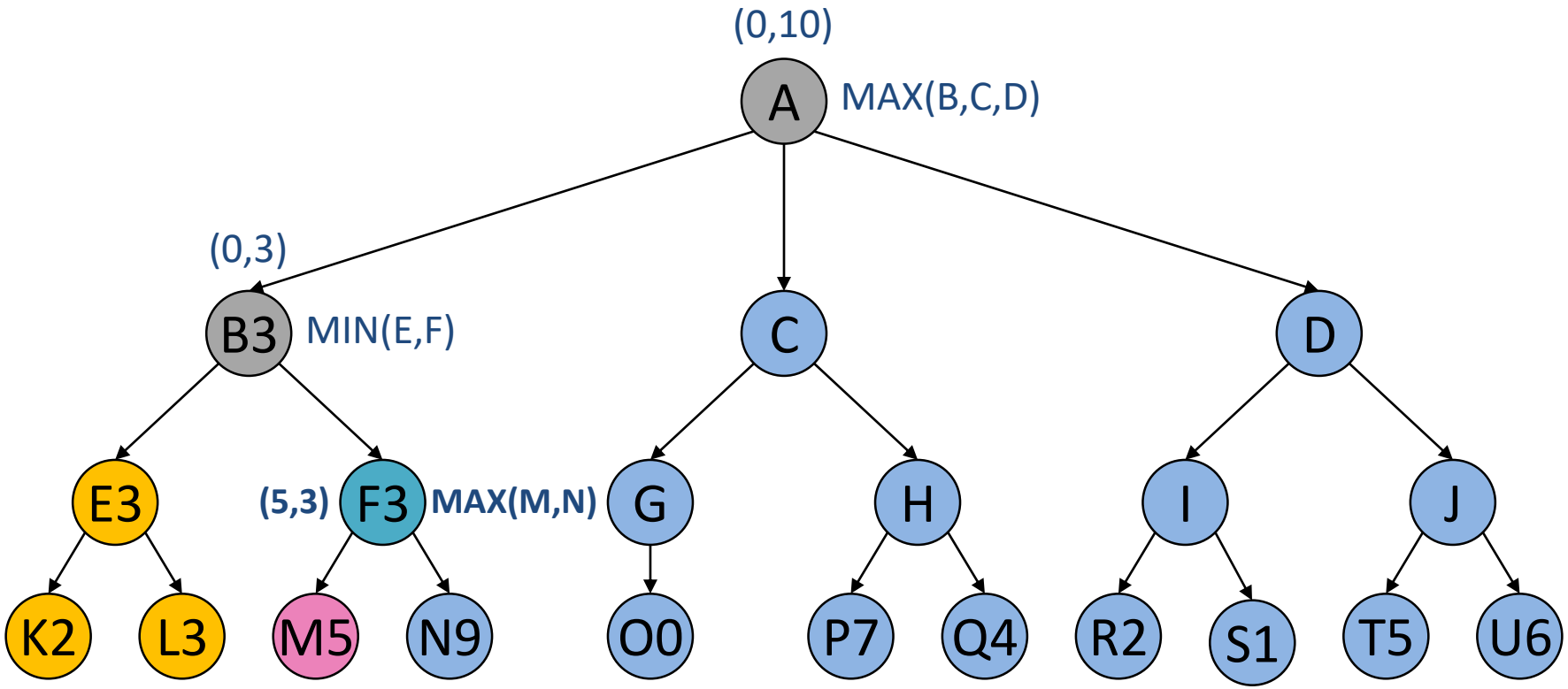
α-β odsecanje (ilustracija)



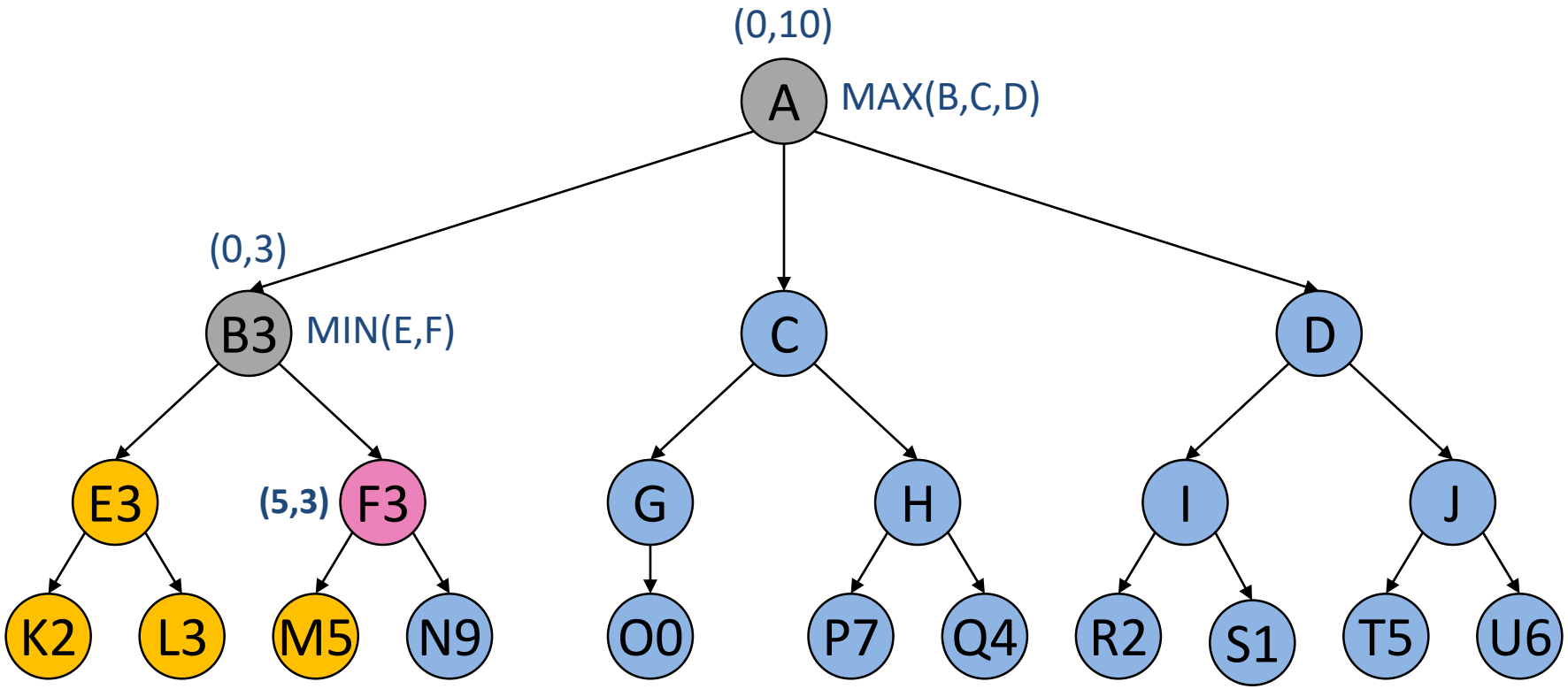
α-β odsecanje (ilustracija)



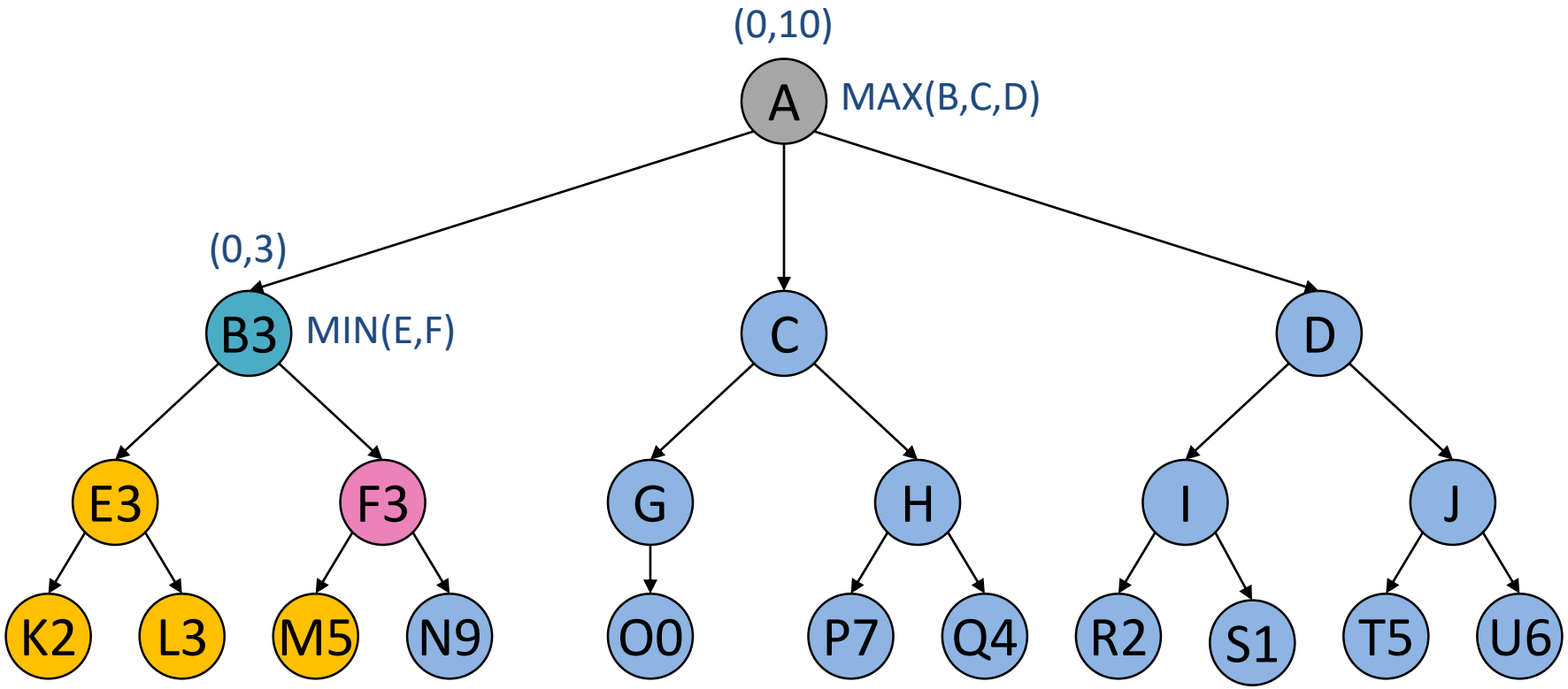
α-β odsecanje (ilustracija)



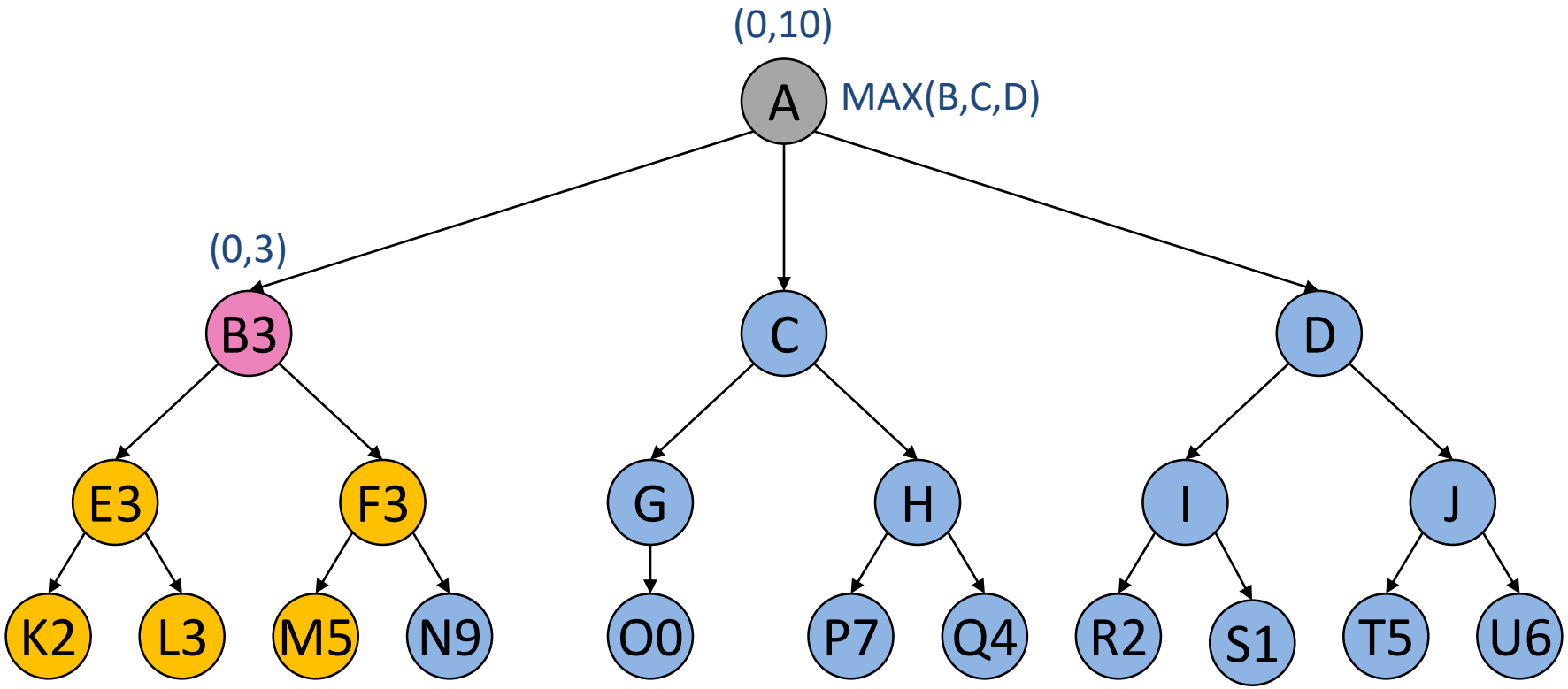
α-β odsecanje (ilustracija)



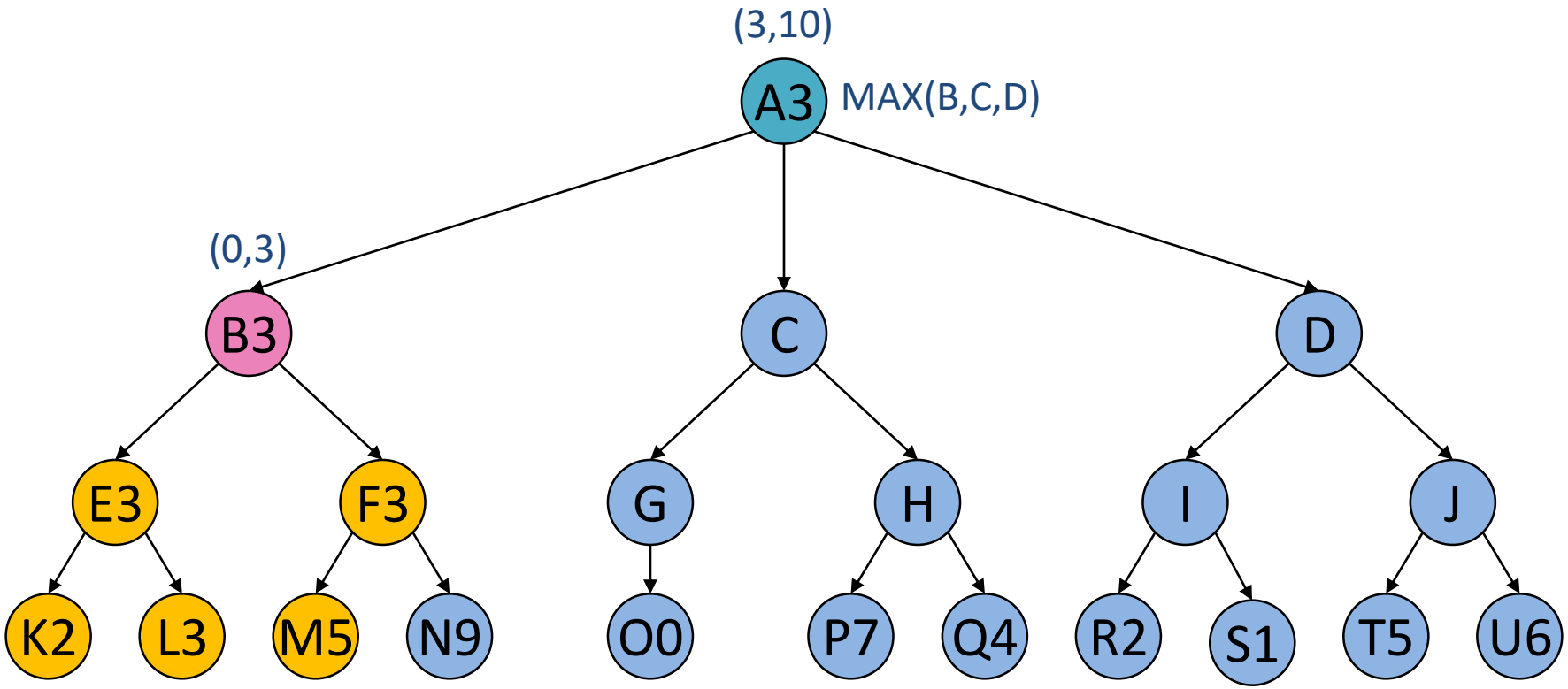
α-β odsecanje (ilustracija)



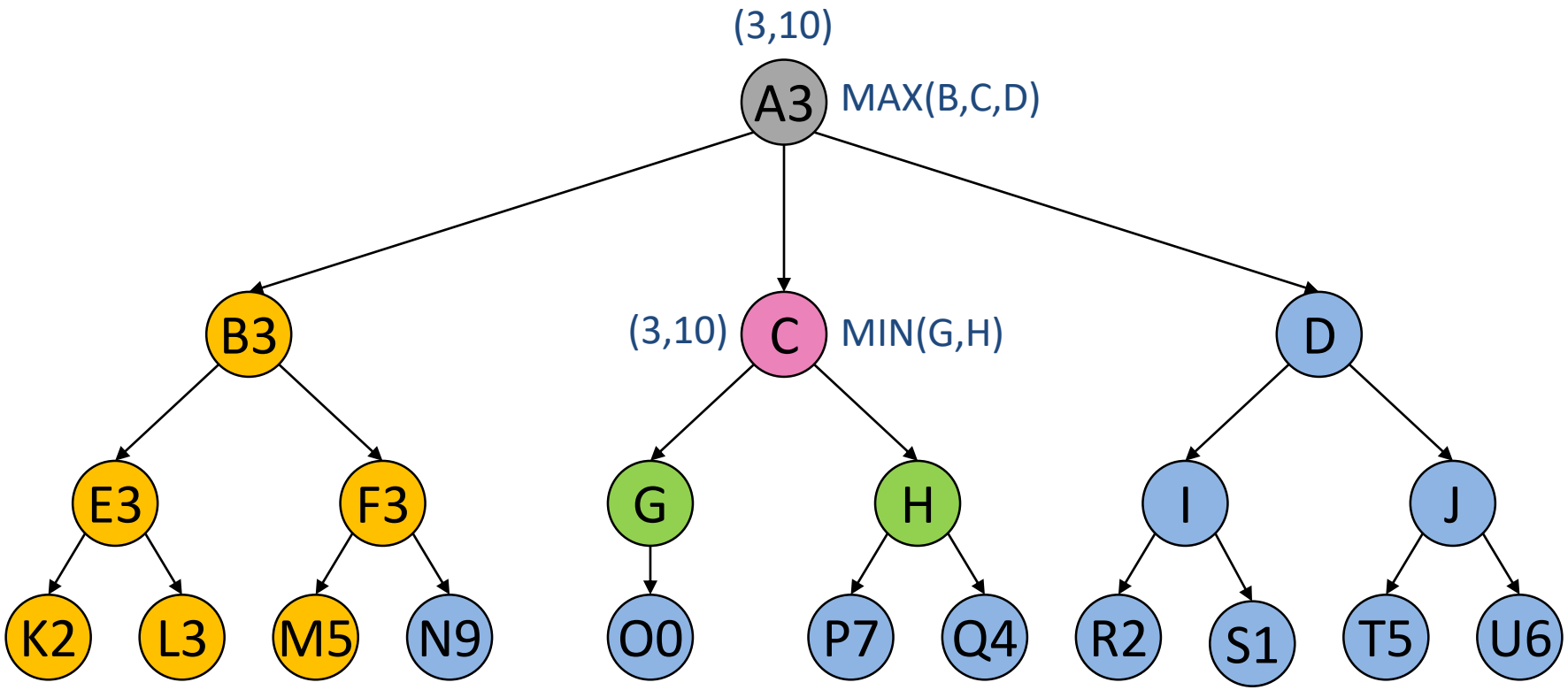
α-β odsecanje (ilustracija)



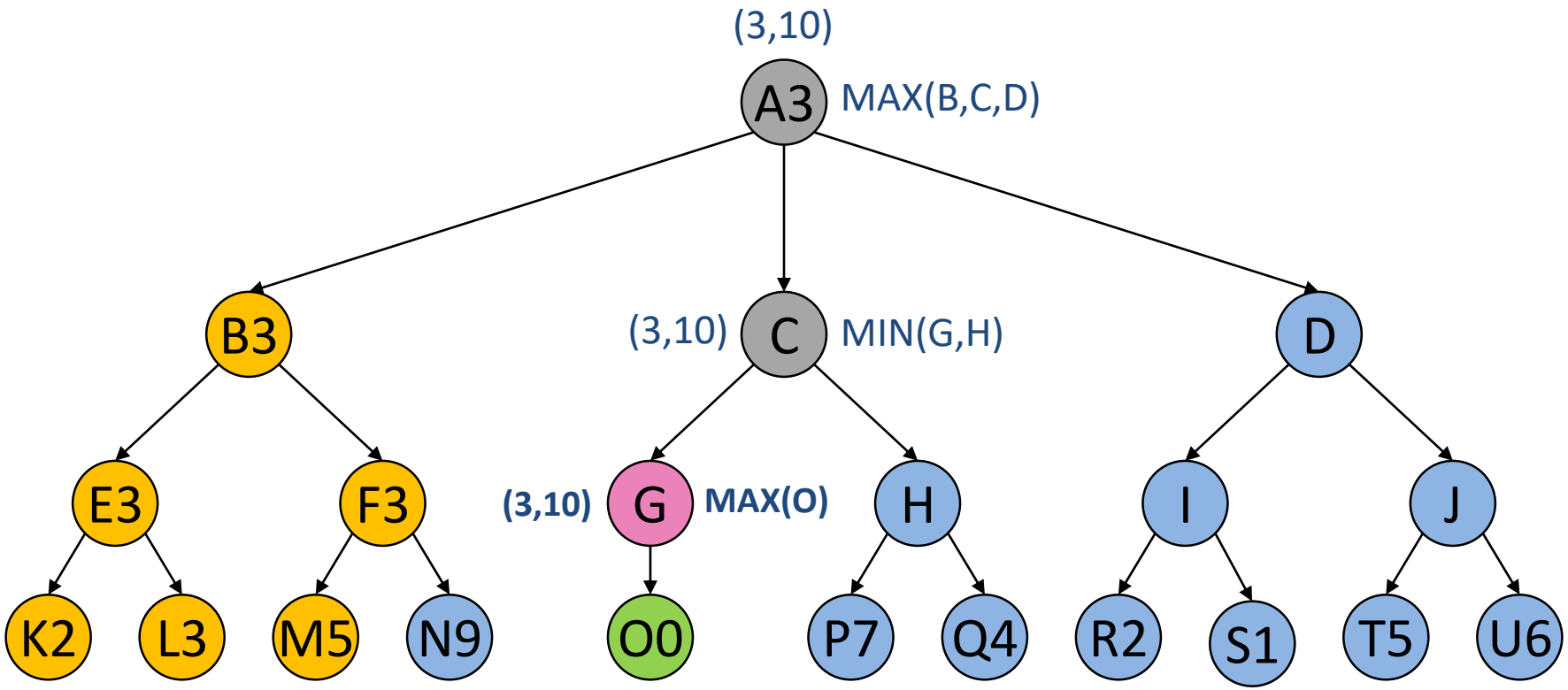
α-β odsecanje (ilustracija)



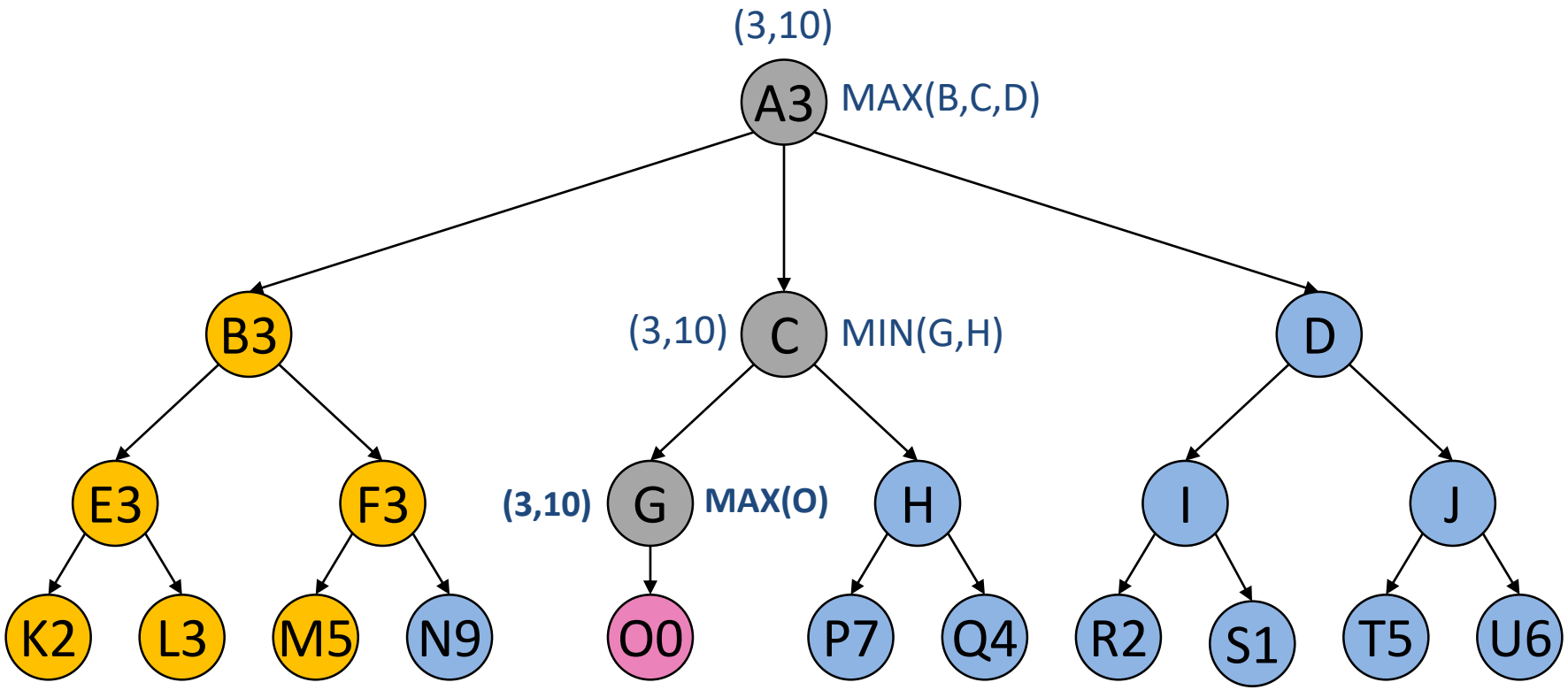
α-β odsecanje (ilustracija)



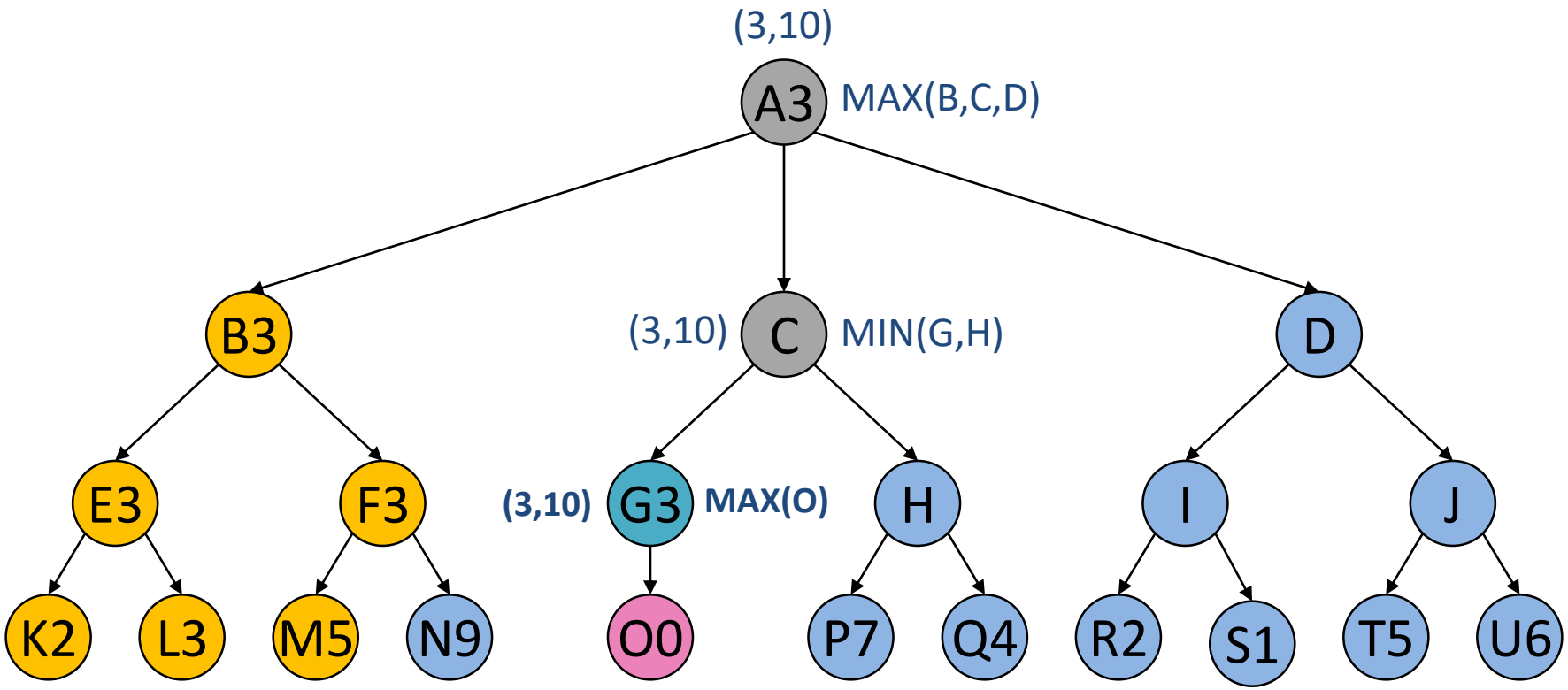
α-β odsecanje (ilustracija)



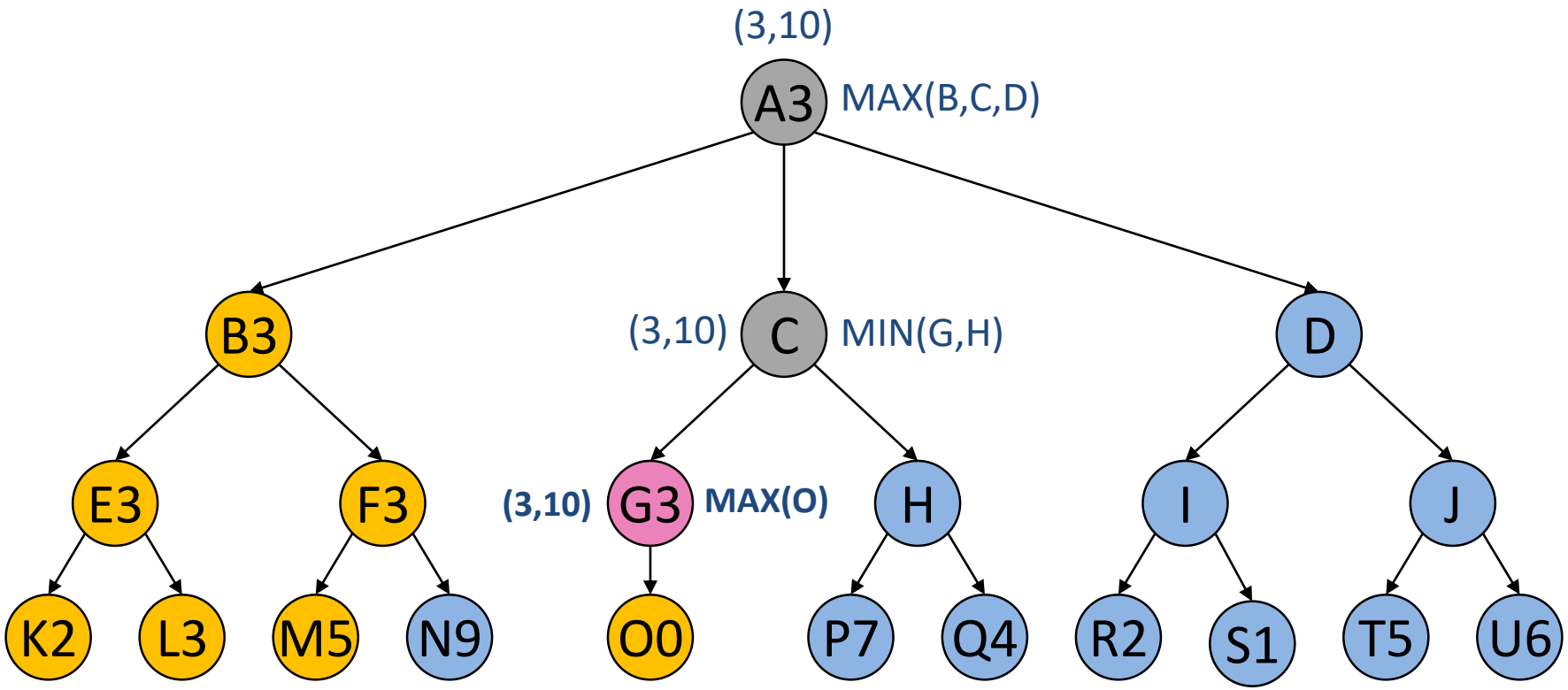
α-β odsecanje (ilustracija)



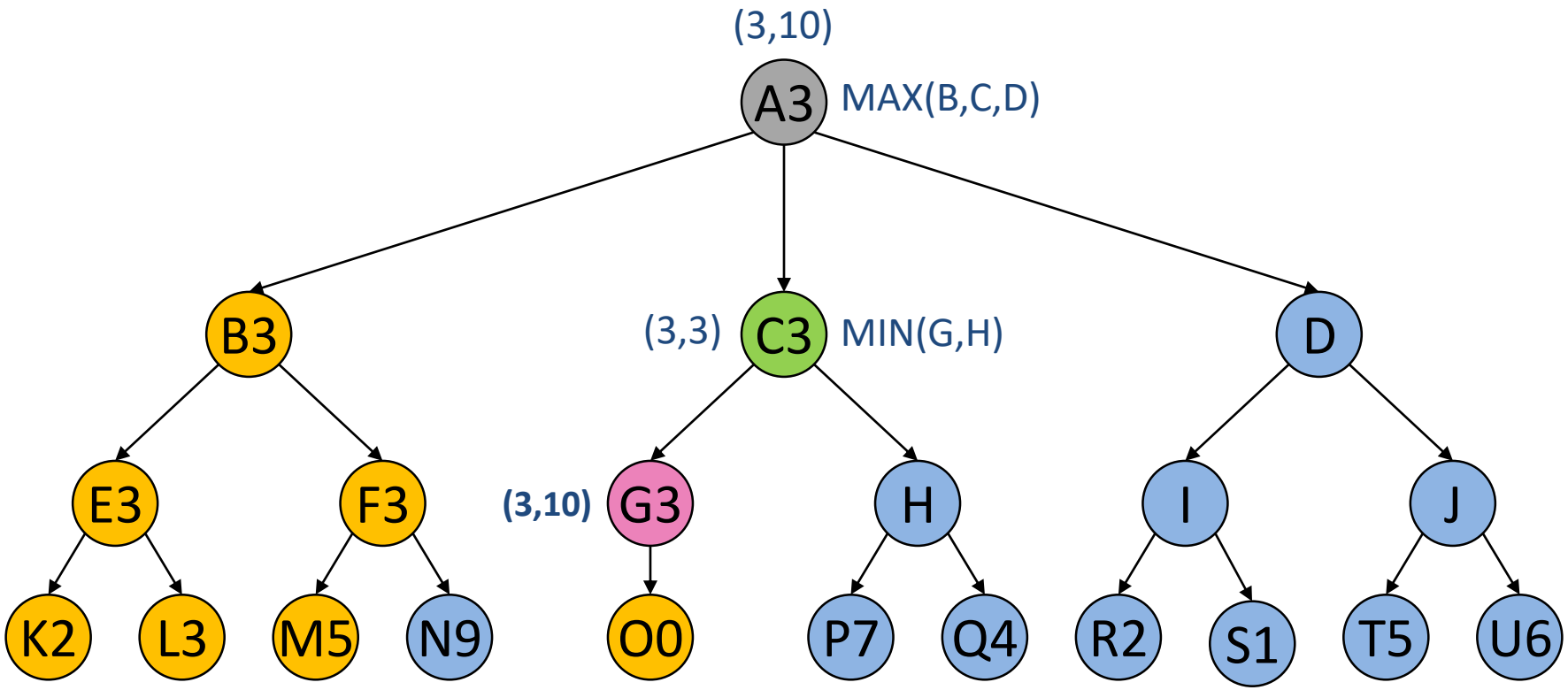
α-β odsecanje (ilustracija)



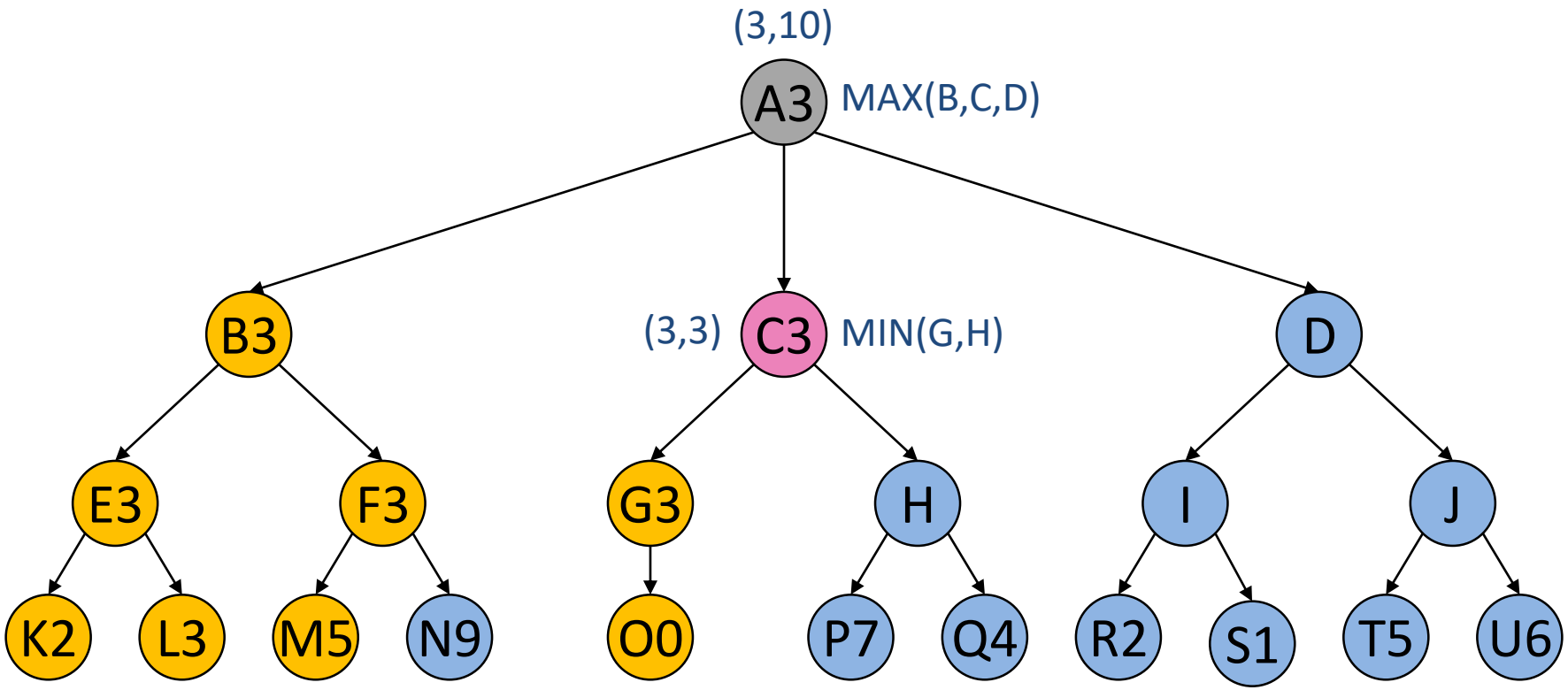
α-β odsecanje (ilustracija)



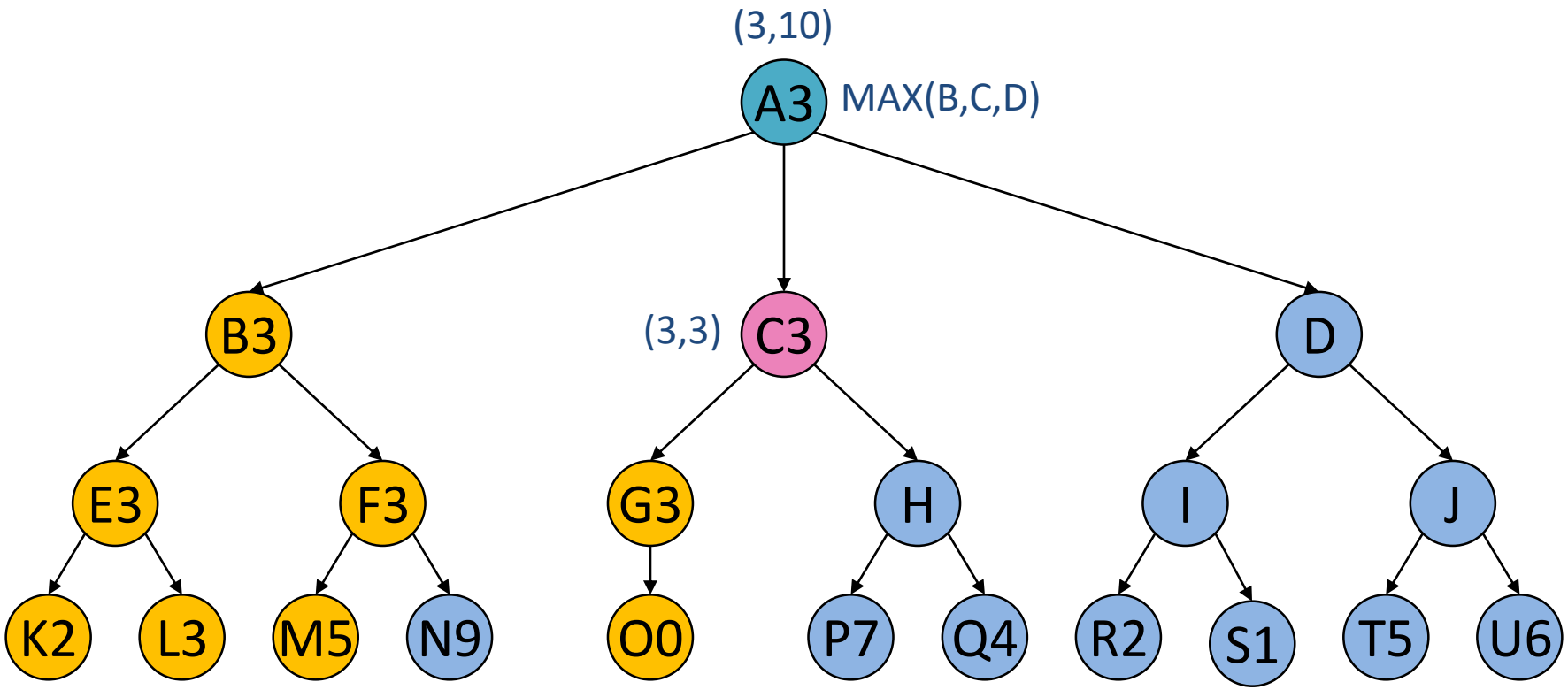
α-β odsecanje (ilustracija)



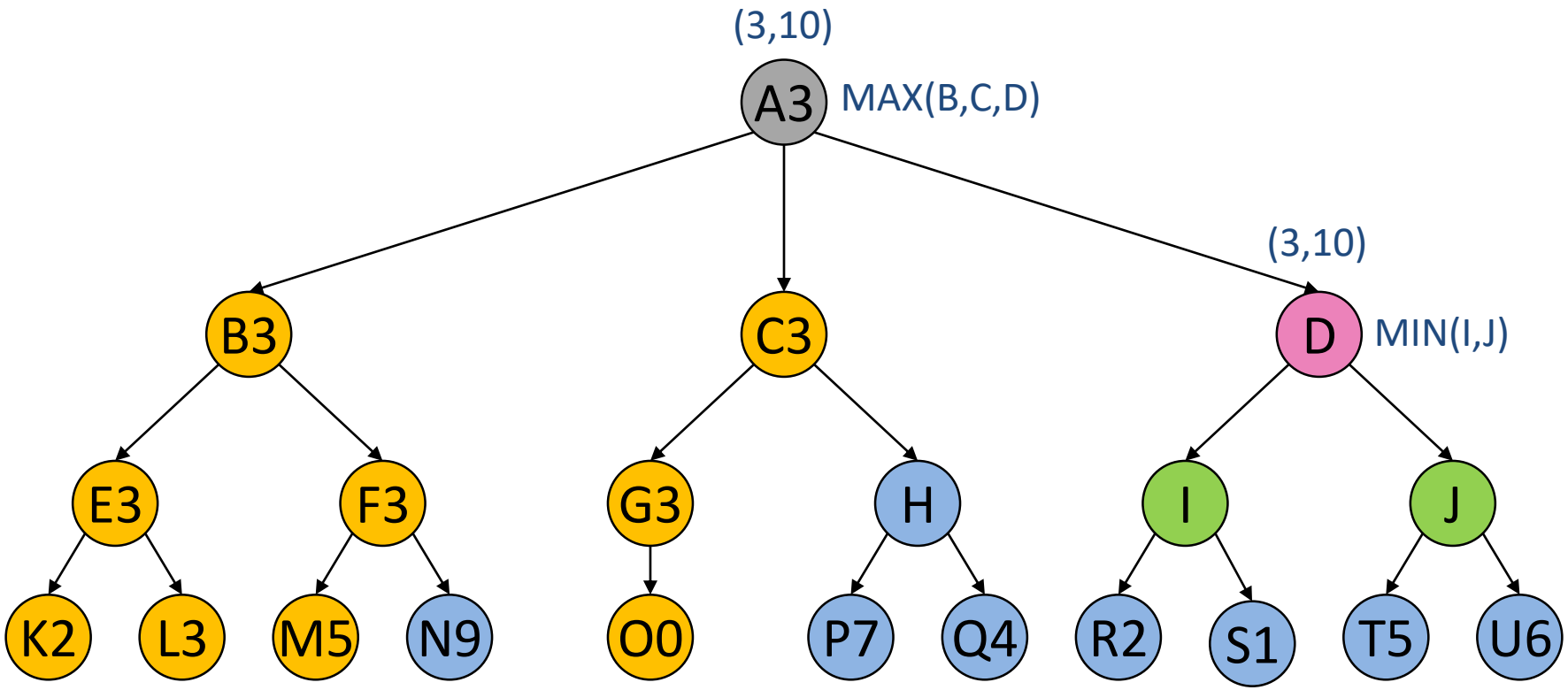
α-β odsecanje (ilustracija)



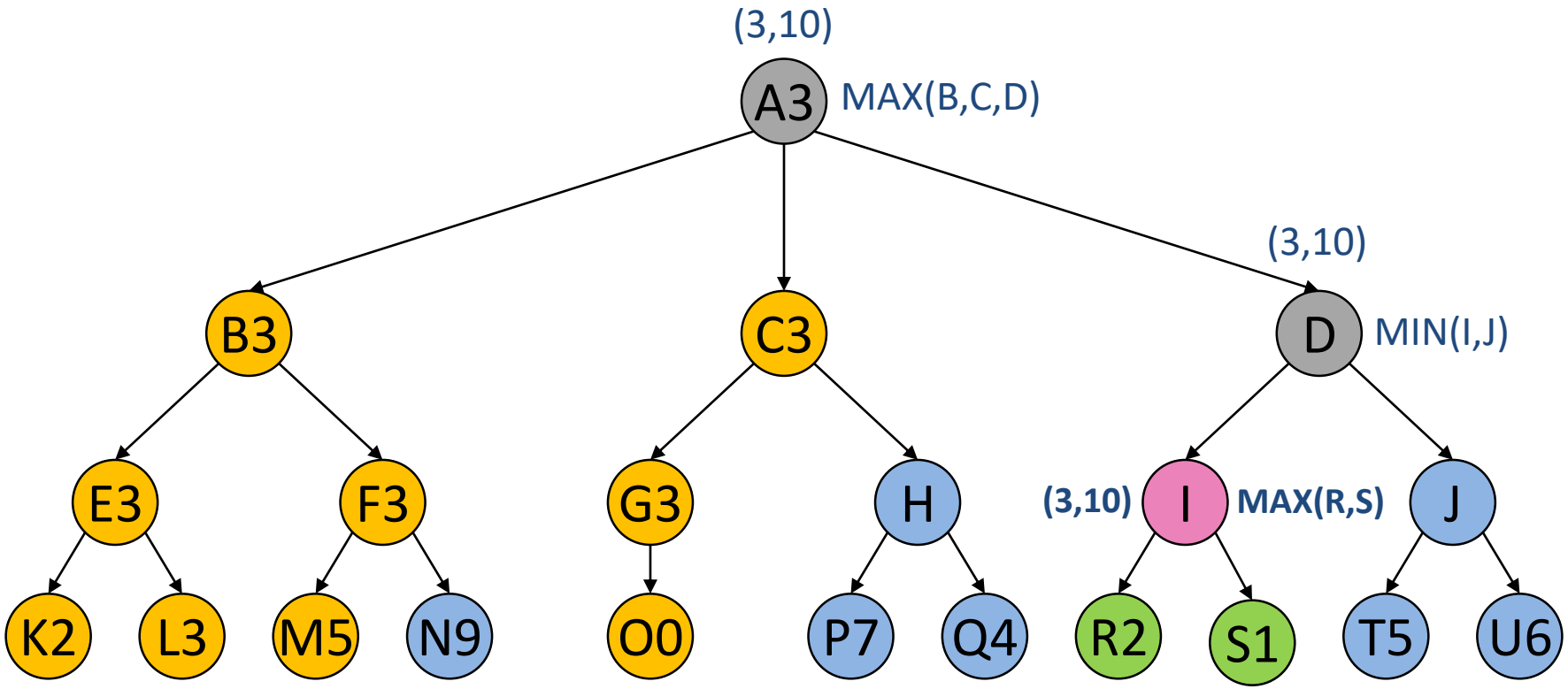
α-β odsecanje (ilustracija)



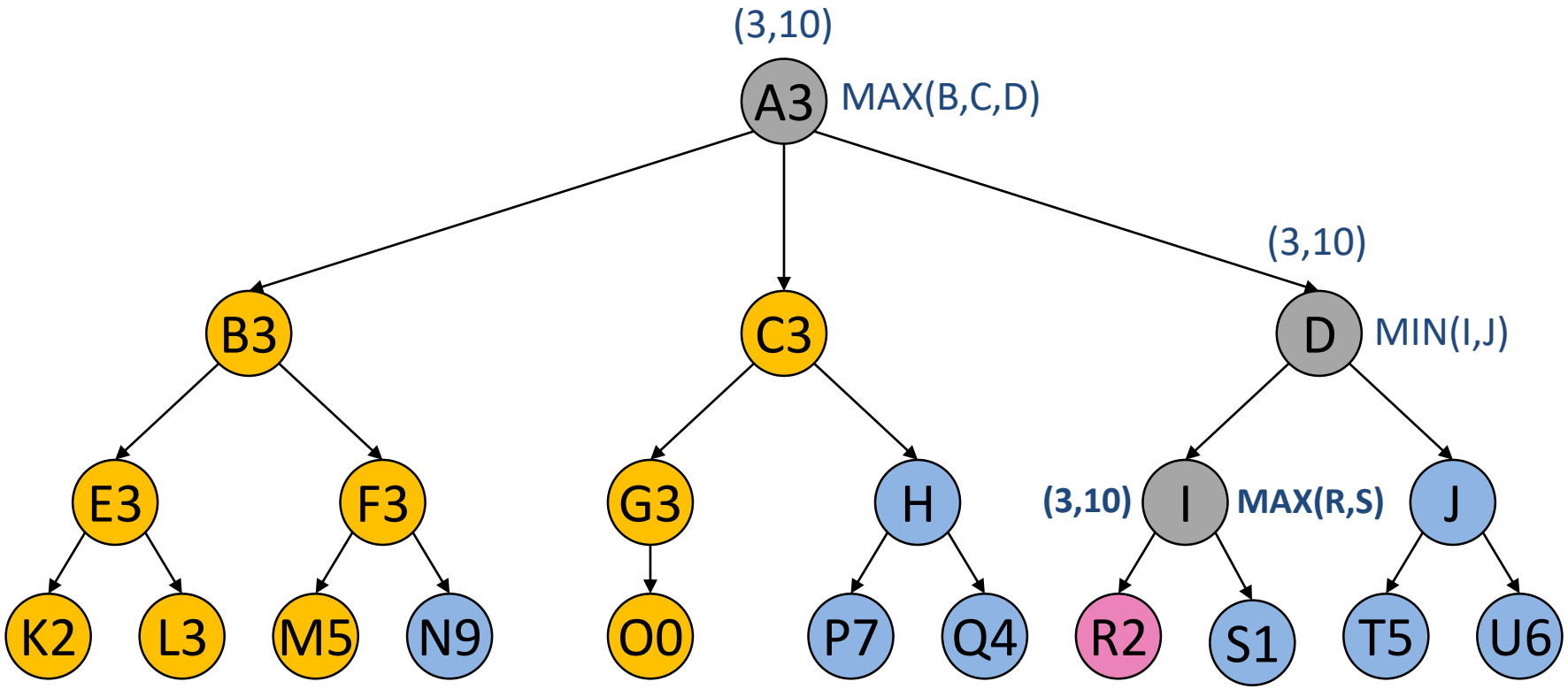
α-β odsecanje (ilustracija)



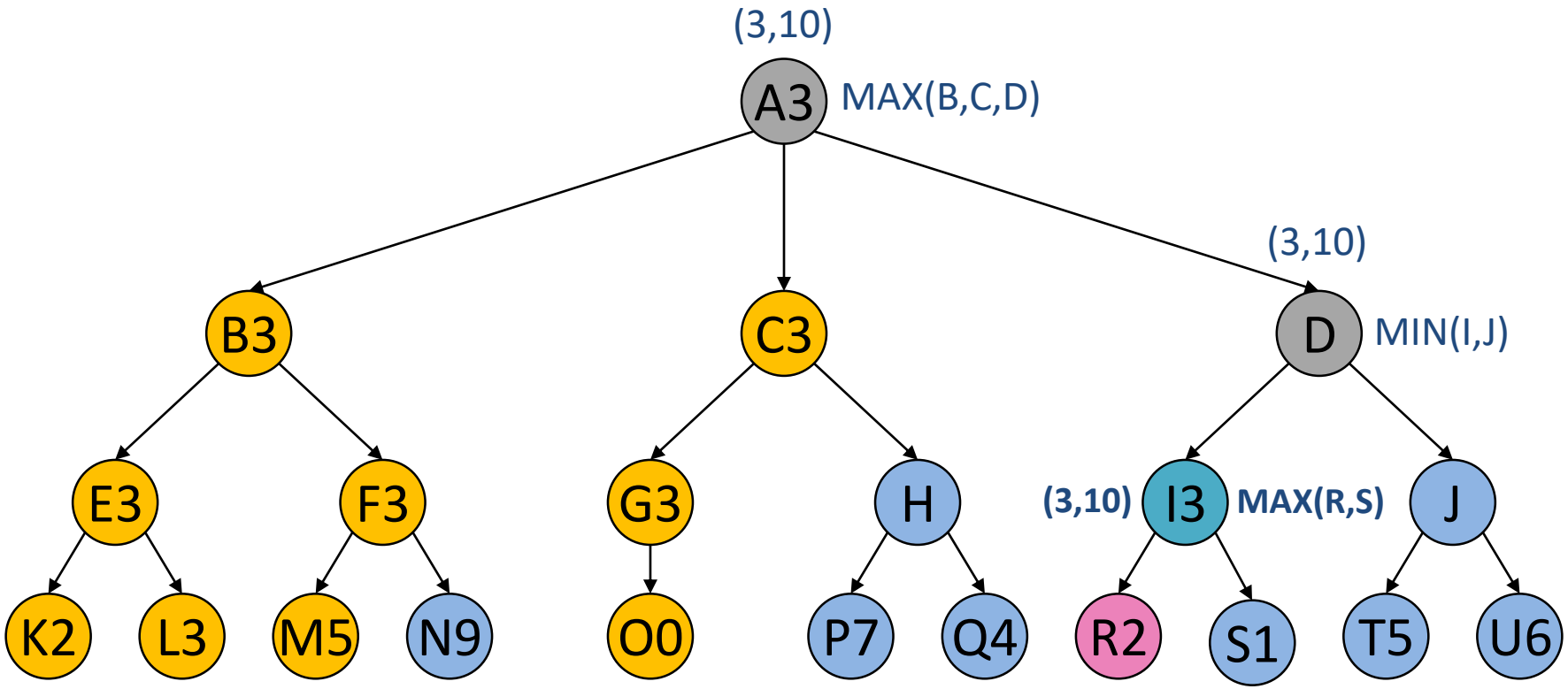
α-β odsecanje (ilustracija)



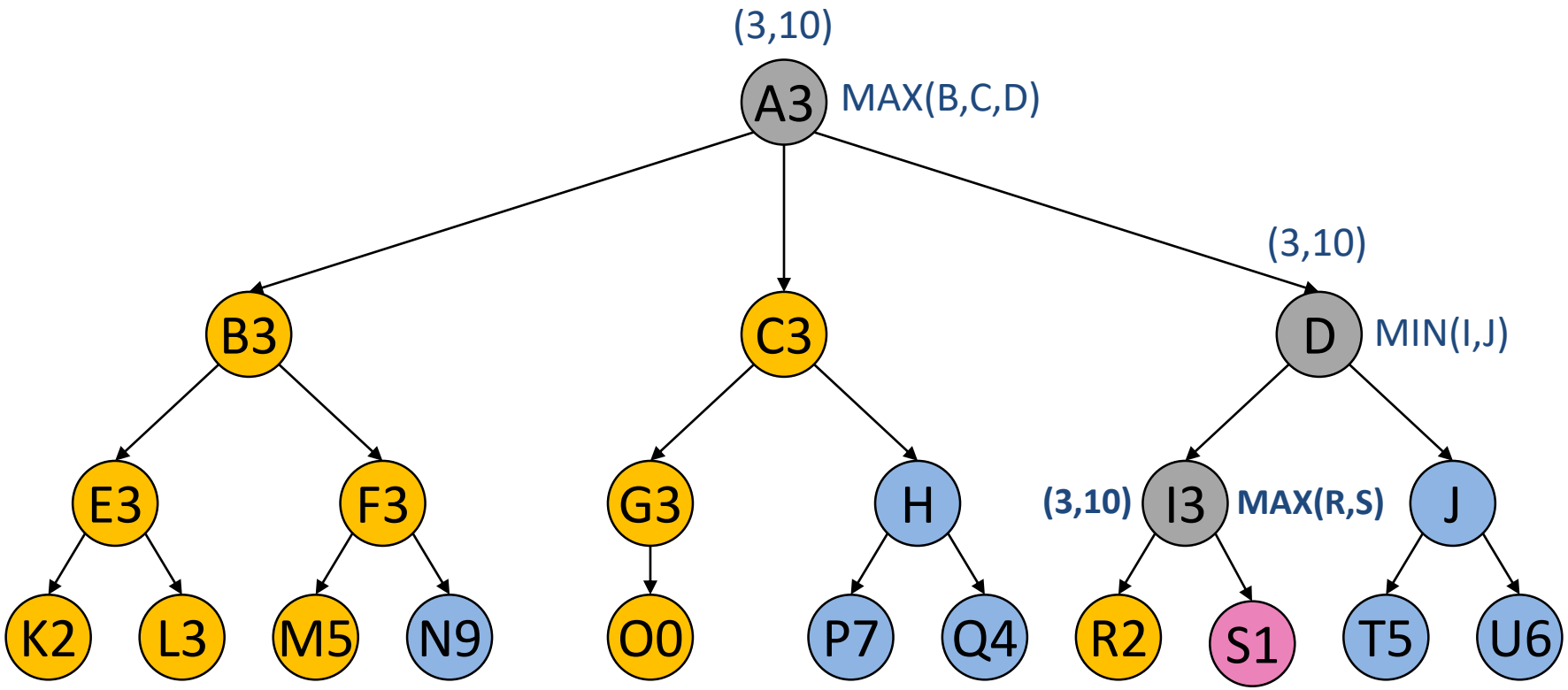
α-β odsecanje (ilustracija)



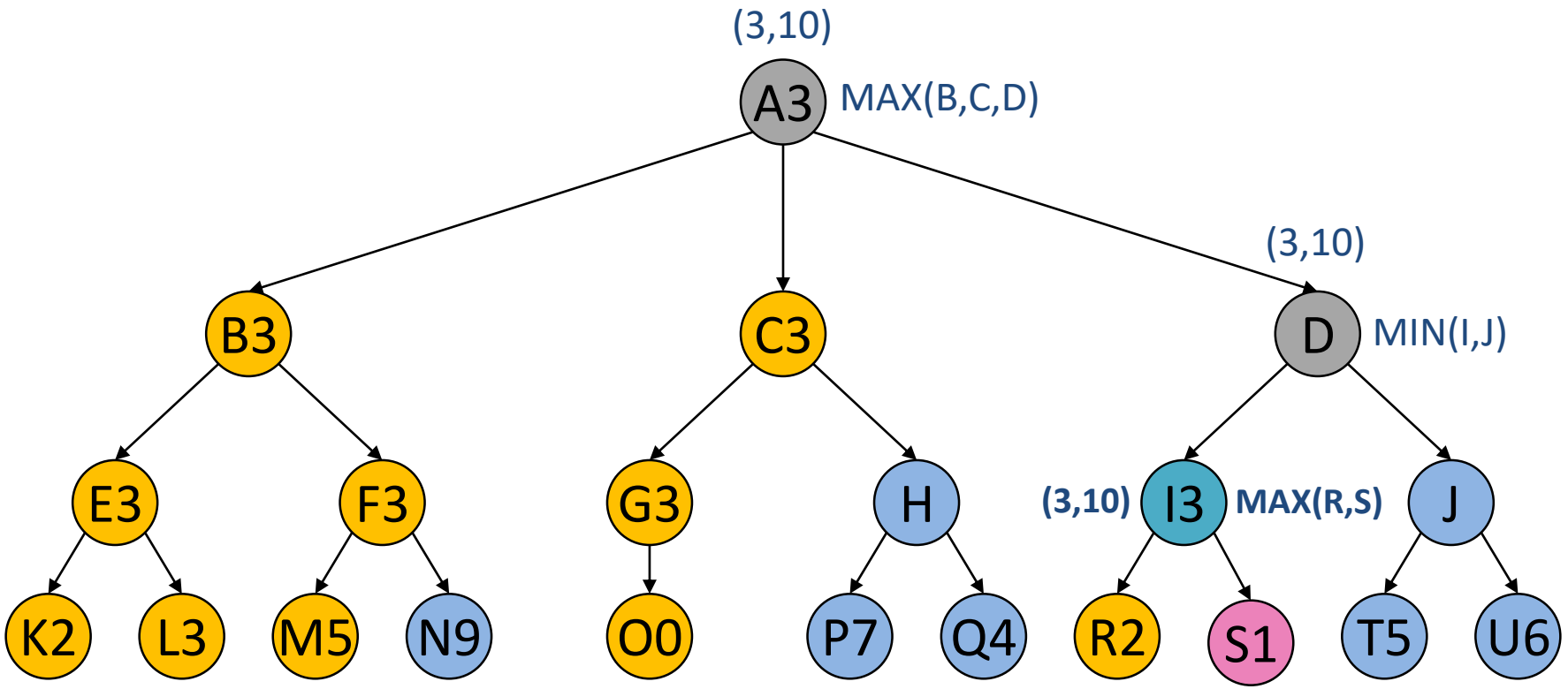
α-β odsecanje (ilustracija)



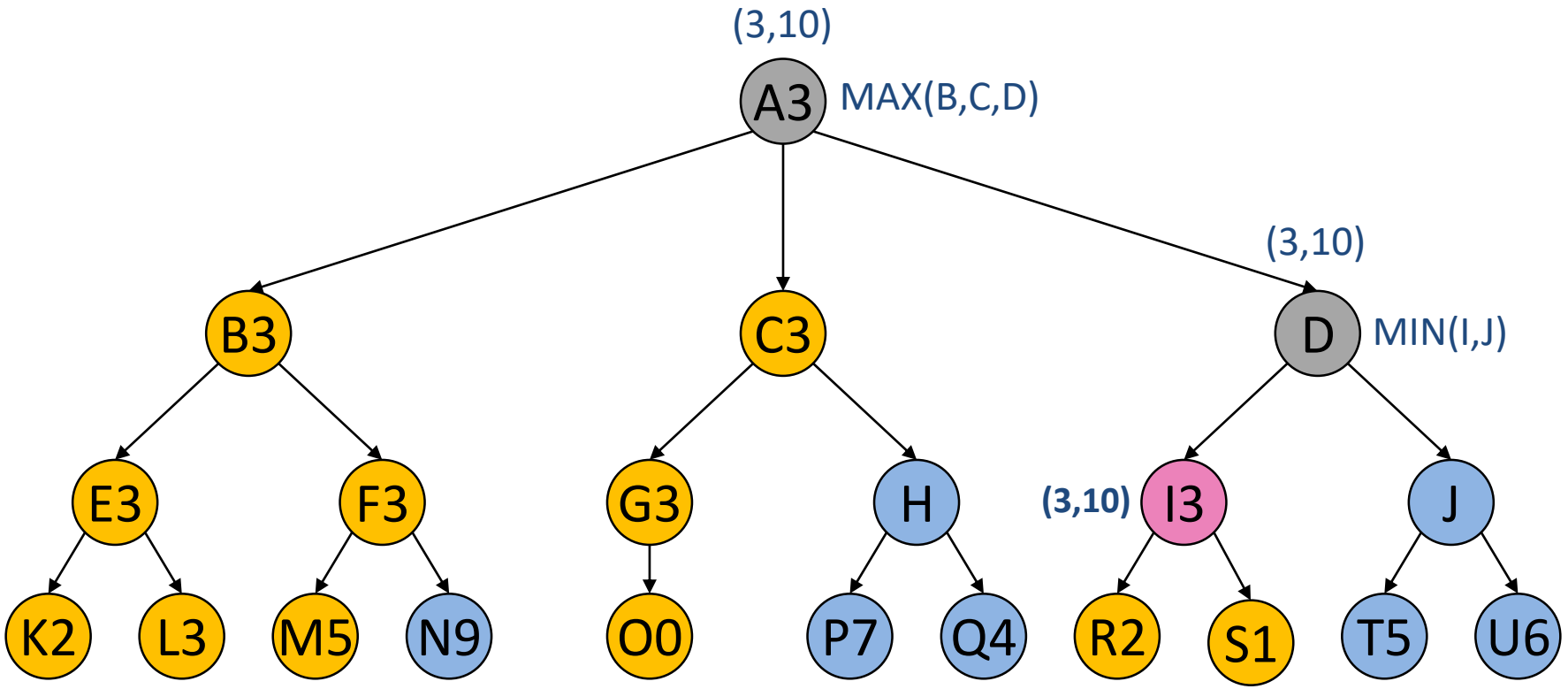
α-β odsecanje (ilustracija)



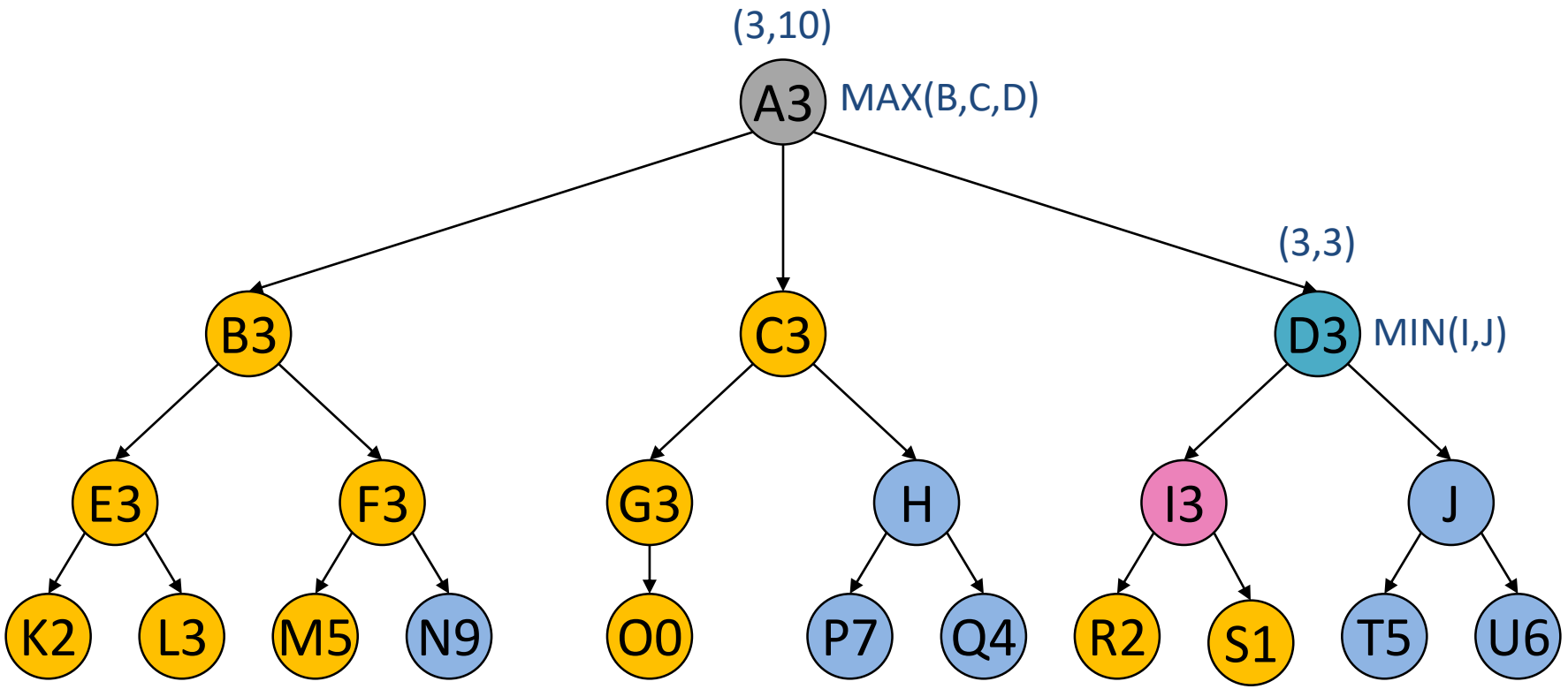
α-β odsecanje (ilustracija)



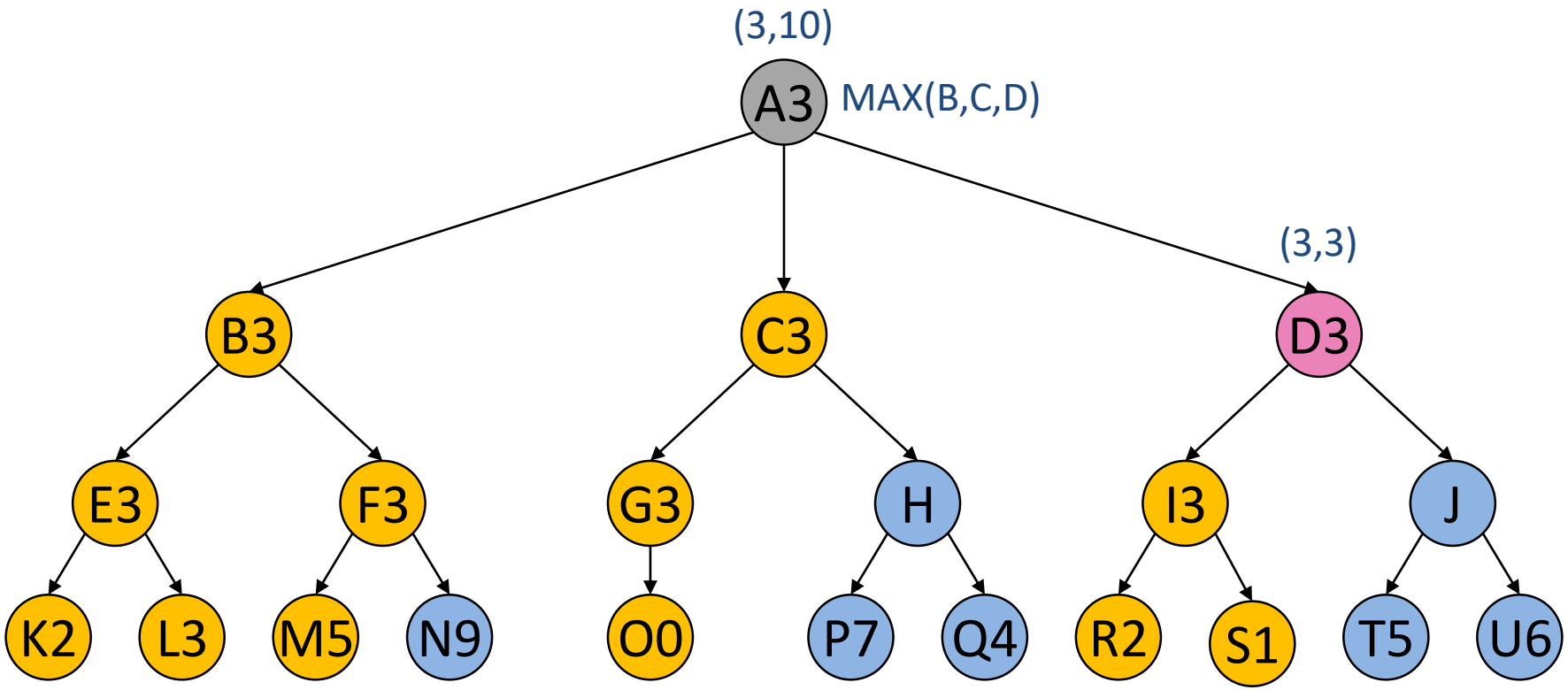
α-β odsecanje (ilustracija)



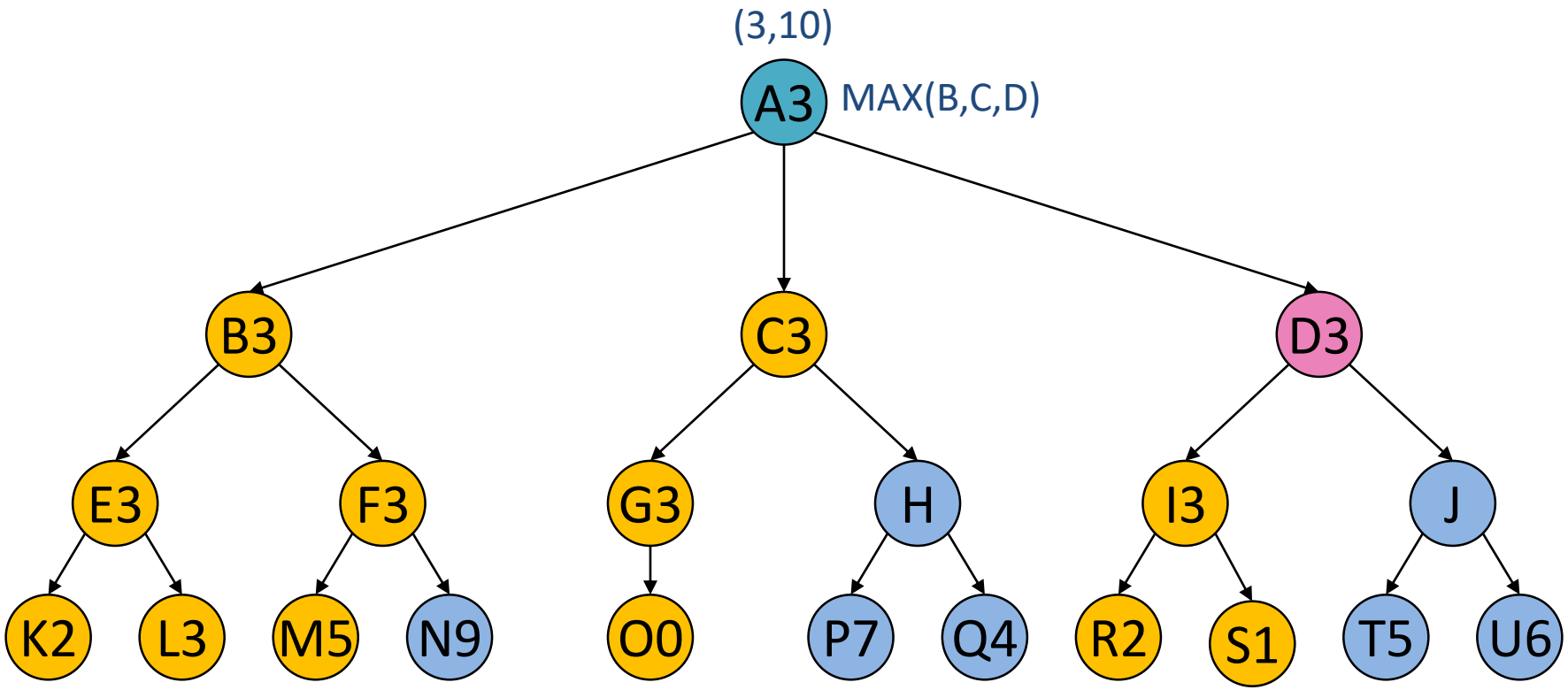
α-β odsecanje (ilustracija)



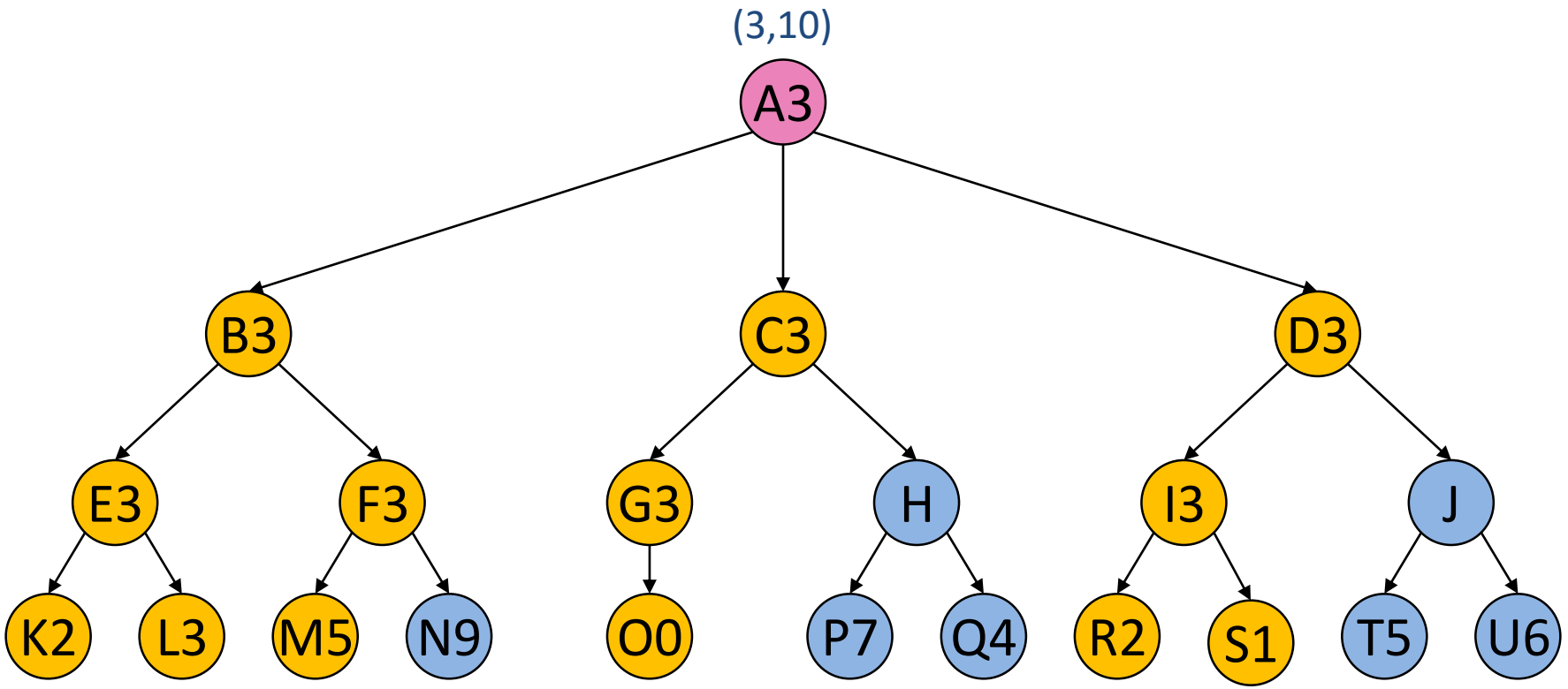
α-β odsecanje (ilustracija)



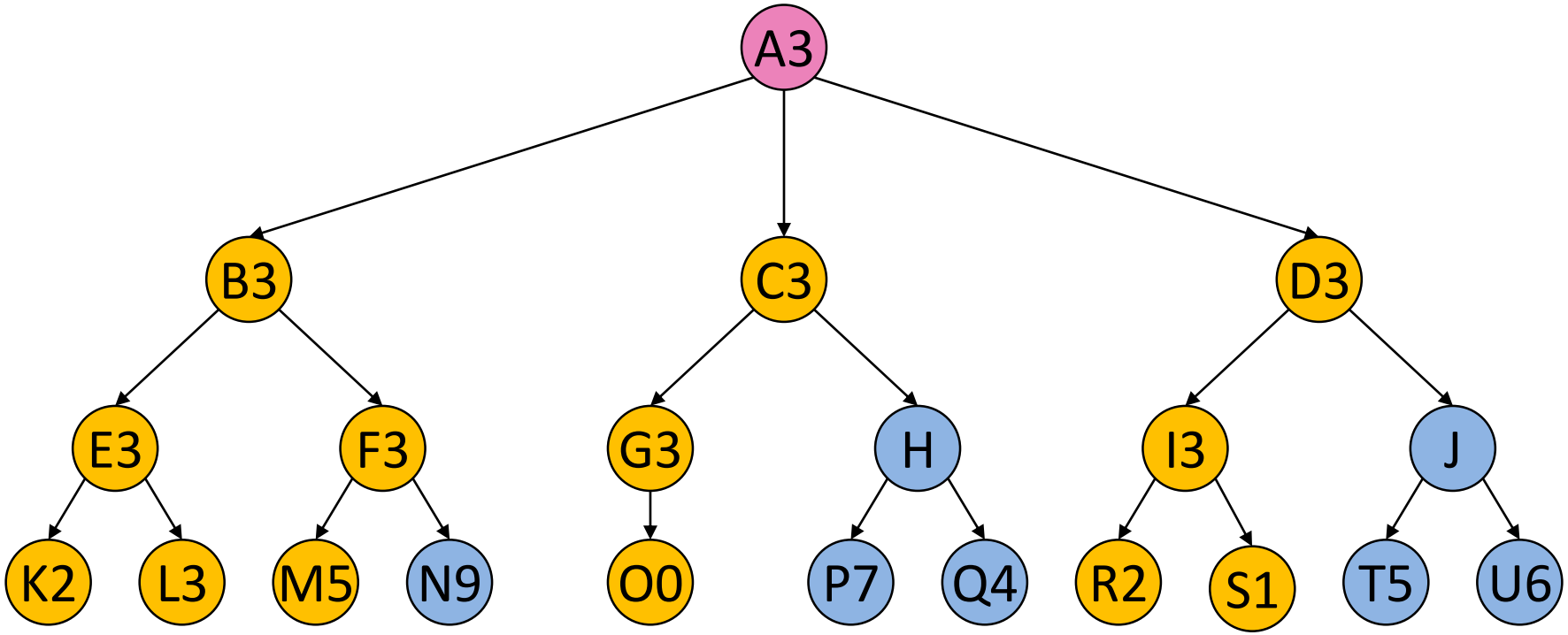
α-β odsecanje (ilustracija)



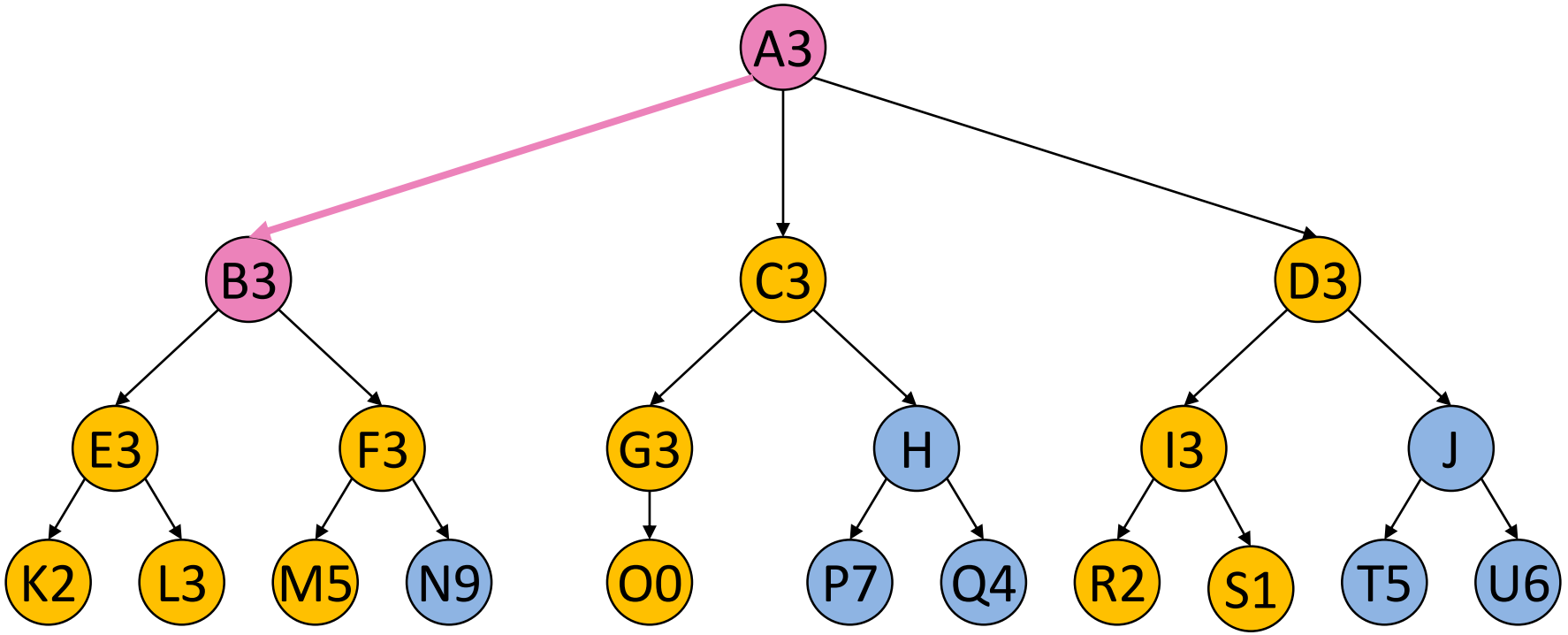
α-β odsecanje (ilustracija)



α-β odsecanje (ilustracija)



α-β odsecanje (ilustracija)

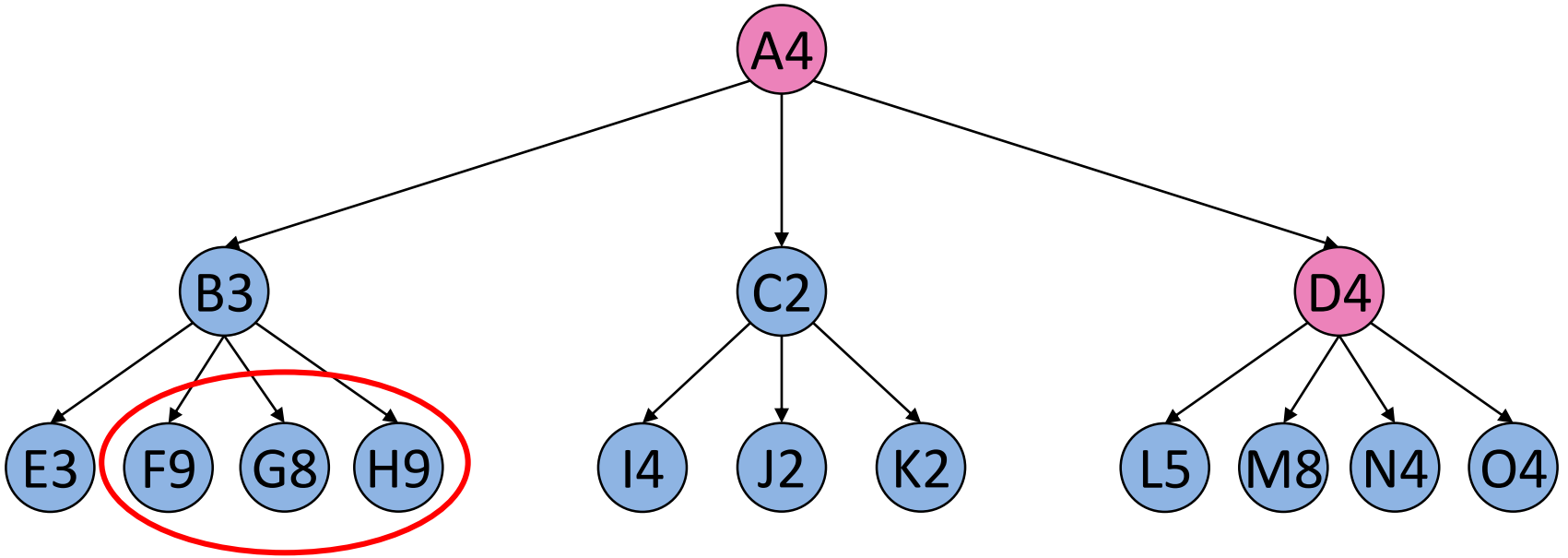


Najbolje je odigrati potez koji igru prevodi iz stanja A u stanje B.

α-β odsecanje - zaključak

- Standardni Min-Max prolazi kroz celo stablo, tj. kroz 21 čvor.
- Min-Max sa α-β odsecanjem ne prolazi kroz sve čvorove, već kroz njih 14.
- 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 algoritma je:
- **$O(b^d)$** – za standardni Min-Max
- **$O(b^{d/2})$** – za Min-Max sa α-β odsecanjem
- α-β odsecanje omogućuje obradu stabla duplo veće visine za isto vreme, kao i isti utrošak radne memorije.

Nedostatak Min-Max algoritma



Primer rešavanja problema – Min-Max

Iks-Oks

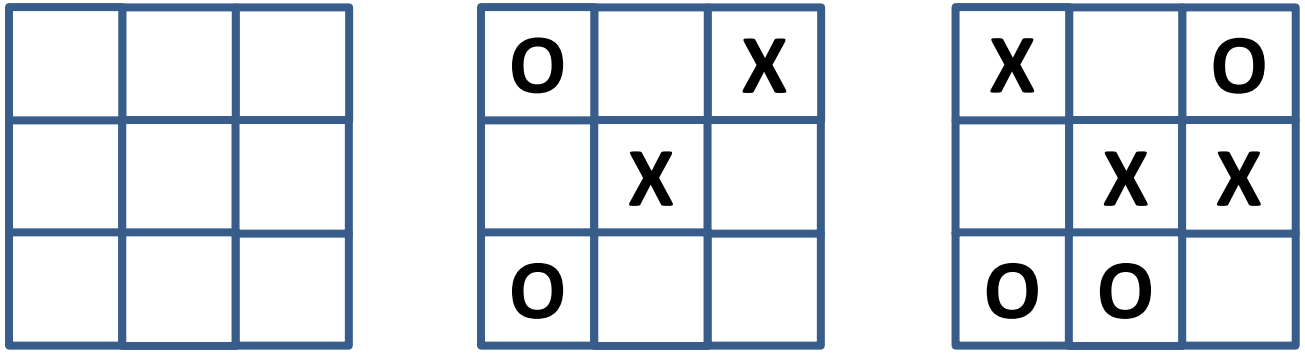
Iks-Oks

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

Koraci za rešavanje problema

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

Predstavljanje stanja

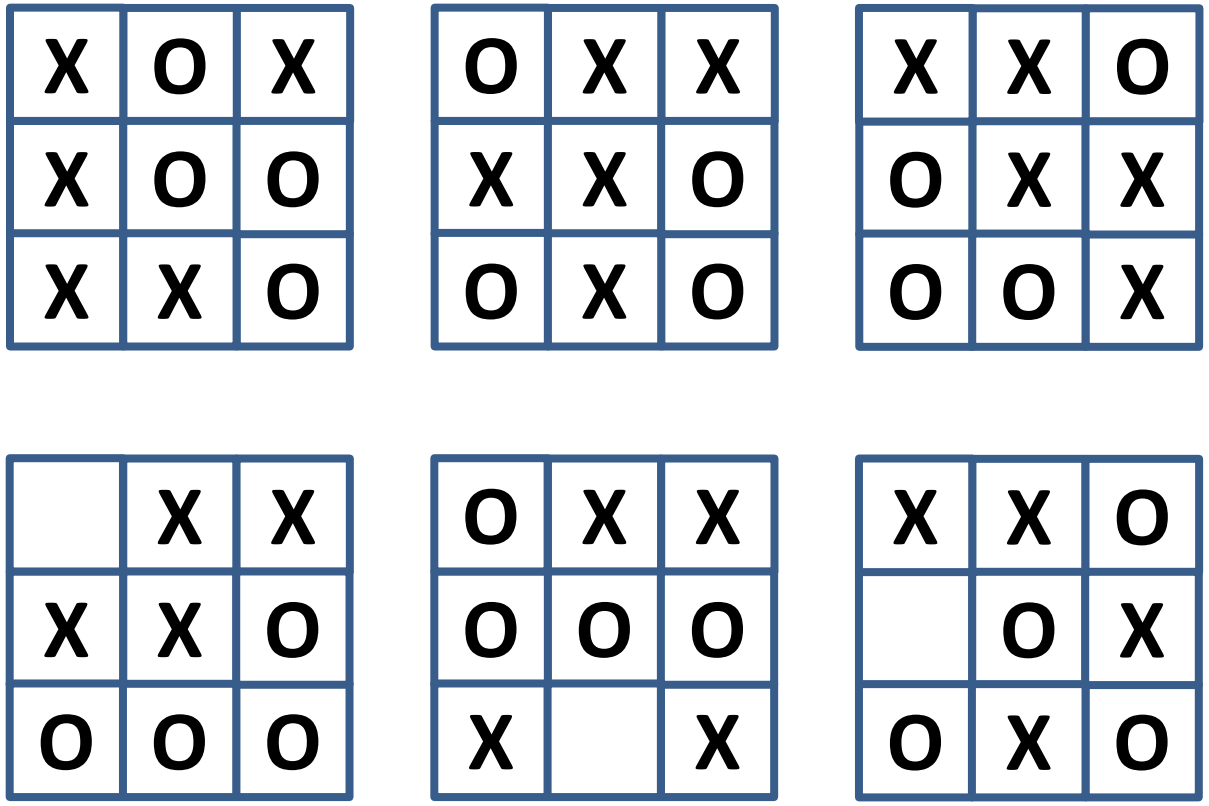


Predstavljanje stanja:

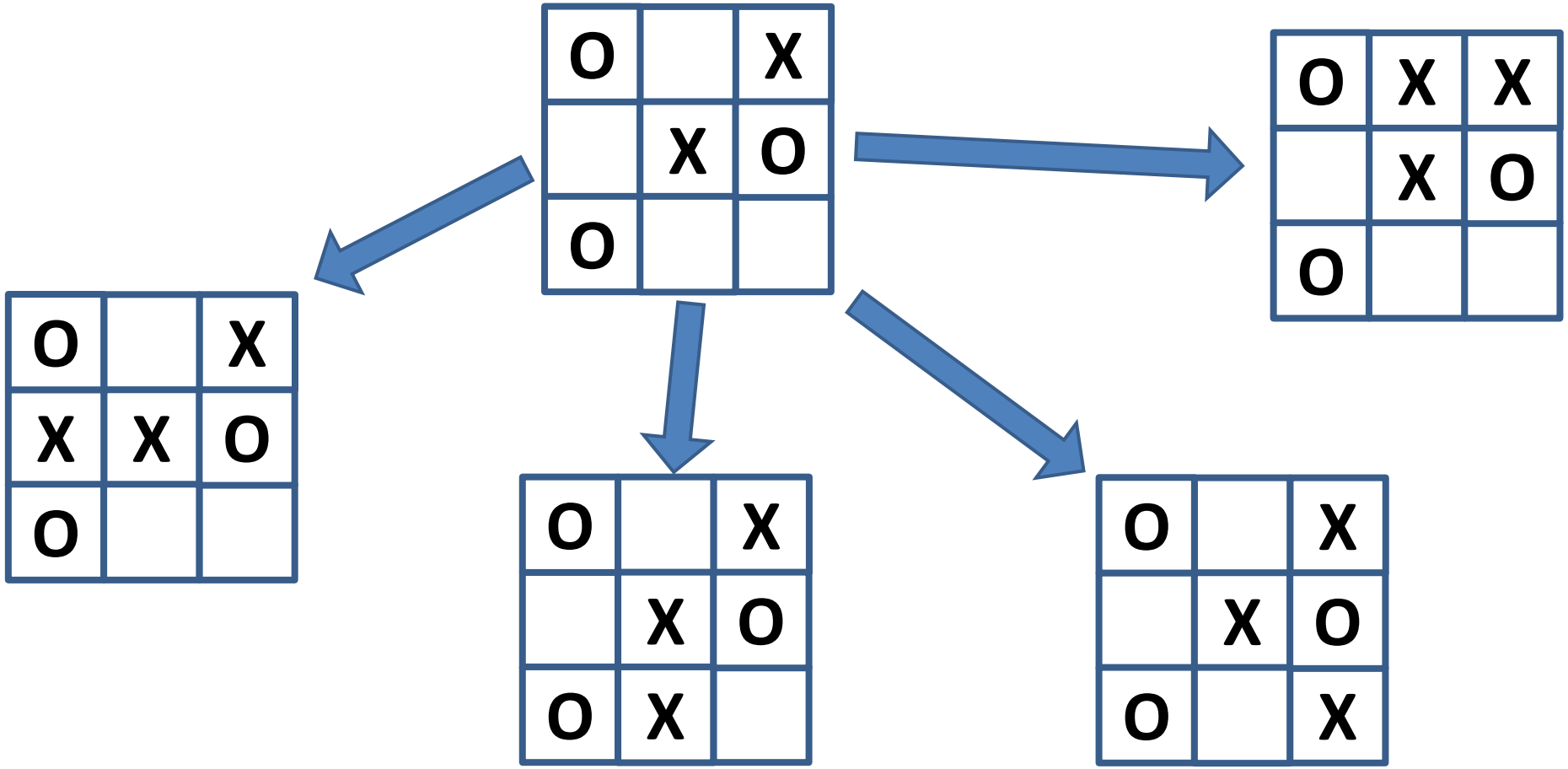
```
1. [[None, None, None], [None, None, None], [None, None, None]]
2. [O, None, X, None, X, None, O, None, None]
3. [(X, None, O), (None, X, X), (O, O, None)]
```

Bilo koji od ovih načina je pogodan

Ciljna stanja



Prelazi između stanja



Ograničenja

- Ne može se odigrati potez na polje koje je zauzeto
- Ne može se odigrati potez van table
  - (0, 4)
  - (5, 2)

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

# Omogućava proveru da li je na zadatom polju X, O ili None
def polje_ima(pos, vrednost, stanje):
    return validno(pos) and stanje[pos[0]][pos[1]] is vrednost
```

Ispitivanje polja, određivanje mogućih poteza i odigravanje poteza

```
def nova_stanja(stanje):
    for i in range(0, 3):
        for j in range(0, 3):
            if polje_ima((i, j), None, stanje):
                yield (i, j)


def igraj(pos, igrac, stanje):
    if polje_ima(pos, None, 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
```

Ispitivanja kraja igre

```
def kraj(stanje):
    glavna_dijagonala = [stanje[x][x] for x in range(0, 3)]
    sporedna_dijagonala = [stanje[x][2 - x] for x in range(0, 3)]
    transponovano_stanje = [[x[pos] for x in stanje] for pos in range(0, 3)]
    sve = [glavna_dijagonala, sporedna_dijagonala] + stanje + transponovano_stanje
    if any([x.count(X) == 3 for x in sve]):
        return 10
    if any([x.count(O) == 3 for x in sve]):
        return -10
    return 0


def full(stanje):
    return not any(None in x for x in stanje)
```

Procena heuristike stanja igre

```
def oceni(stanje):
    glavna_dijagonala = [stanje[x][x] for x in range(0, 3)]
    sporedna_dijagonala = [stanje[x][2 - x] for x in range(0, 3)]
    flatten_table = [x for sub_list in stanje for x in sub_list]
    return (flatten_table.count(X) - flatten_table.count(O) +
            glavna_dijagonala.count(X) - glavna_dijagonala.count(O) +
            sporedna_dijagonala.count(X) - sporedna_dijagonala.count(O))
```

Min-Max algoritam

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

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

def minimax(stanje, dubina, moj_potez, potez=None):
    if abs(kraj(stanje)) == 10 or full(stanje):
        return (potez, kraj(stanje))
    igrac = X if moj_potez else O
    funkcija_min_max = max_stanje if moj_potez else min_stanje
    lista_poteza = list(nova_stanja(stanje))
    if dubina == 0 or lista_poteza is None or len(lista_poteza) == 0:
        return (potez, oceni(stanje))
    return funkcija_min_max([minimax(igraj(x, igrac, stanje), dubina - 1,
           not moj_potez, x if potez is None else potez) for x in lista_poteza])
```

Min-Max algoritam (α-β odsecanje) – *max_value*

```
def max_value(stanje, dubina, alpha, beta, potez=None):
    if abs(kraj(stanje)) == 10 or full(stanje):
        return (potez, kraj(stanje))
    lista_poteza = list(nova_stanja(stanje))
    if dubina == 0 or lista_poteza is None or len(lista_poteza) == 0:
        return (potez, oceni(stanje))
    else:
        for s in lista_poteza:
            alpha = max(alpha, min_value(igraj(s, X, stanje), dubina - 1,
                        alpha, beta, s if potez is None else potez), key=lambda x: x[1])
            if alpha[1] >= beta[1]:
                return beta
    return alpha
```

Min-Max algoritam (α-β odsecanje) – *min_value*

```
def min_value(stanje, dubina, alpha, beta, potez=None):
    if abs(kraj(stanje)) == 10 or full(stanje):
        return (potez, kraj(stanje))
    lista_poteza = list(nova_stanja(stanje))
    if dubina == 0 or lista_poteza is None or len(lista_poteza) == 0:
        return (potez, oceni(stanje))
    else:
        for s in lista_poteza:
            beta = min(beta, max_value(igraj(s, O, stanje), dubina - 1,
                       alpha, beta, s if potez is None else potez), key=lambda x: x[1])
            if beta[1] <= alpha[1]:
                return alpha
    return beta
```

Min-Max algoritam (α-β odsecanje)

```
def minimax_alpha_beta(stanje, dubina, moj_potez, alpha=(None, -10), beta=(None, 10)):
    if moj_potez:
        return max_value(stanje, dubina, alpha, beta)
    else:
        return min_value(stanje, dubina, alpha, beta)
```

Štampanje

```
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):
    return (f'{symbol} '
            if symbol in [X, O]
            else '  ')
```

Pokretanje igre

```
def igra(tabla, igrac, potez):
    print_table(tabla)
    while (kraj(tabla) == 0 and not full(tabla)):
        # dubina je u ovom slučaju 9, ali može da bude i manja
        min_max_result = minimax(tabla, 9, potez)
        min_max_alpha_beta_result = minimax_alpha_beta(tabla, 9, potez)
        print(f"Min-Max: {min_max_result}")
        print(f"Min-Max α-β: {min_max_alpha_beta_result}")
        naj = min_max_result[0] if type(min_max_result) is tuple else (0, 0)
        tabla = igraj(naj, igrac, tabla)
        print_table(tabla)
        igrac = O if igrac is X else X
        potez = not potez
    pobednik = (X if kraj(tabla) == 10 else
                (O if kraj(tabla) == -10 else
                 "Nerešeno"))
    print(f"Pobednik je: {pobednik}")
```

- Pokretanje igre

```
tabla = [[None, None, None],
         [None, None, None],
         [None, None, None]]

igrac = (X if input("Uneti igrača: ") == "X"
           else O)

potez = True if igrac == X else False

igra(tabla, igrac, potez)
```