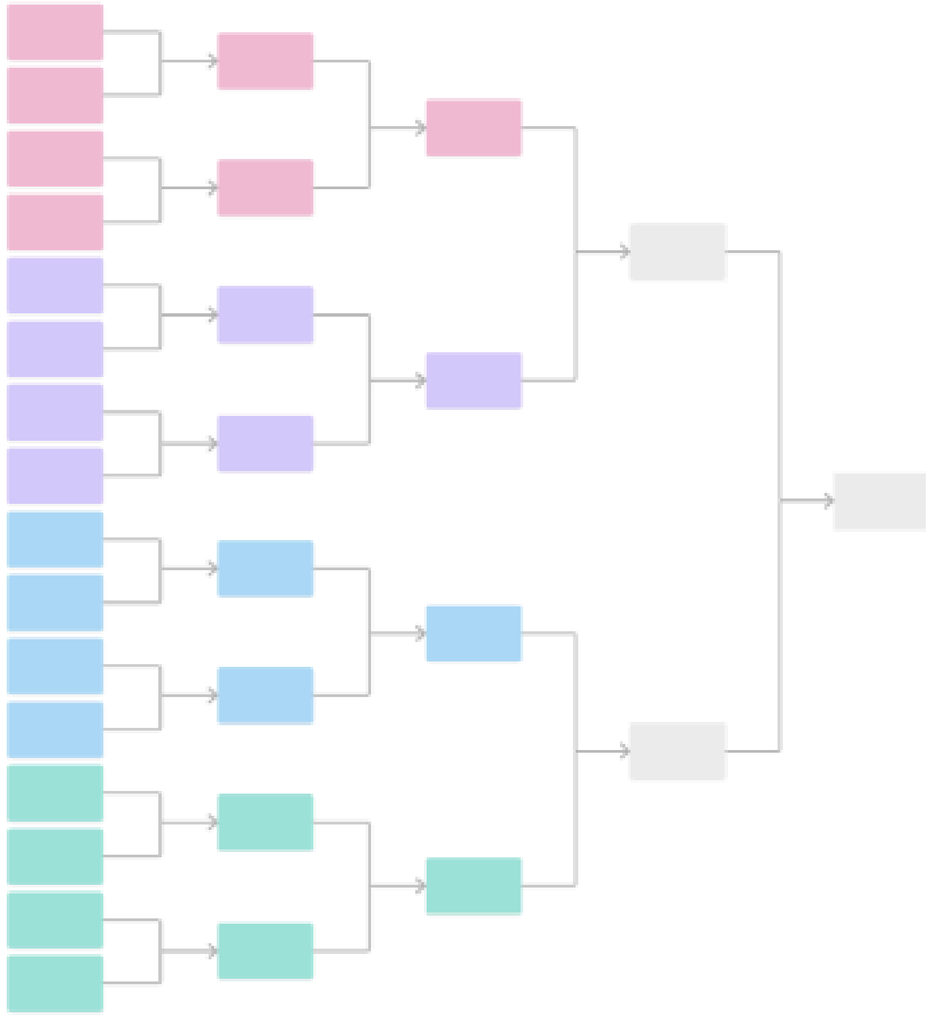
CADT បណ្ឌិត្យសភាបច្ចេកវិទ្យាឌីជីថលកម្ពុជា
Cambodia Academy of Digital Technology

Tournament Bracket

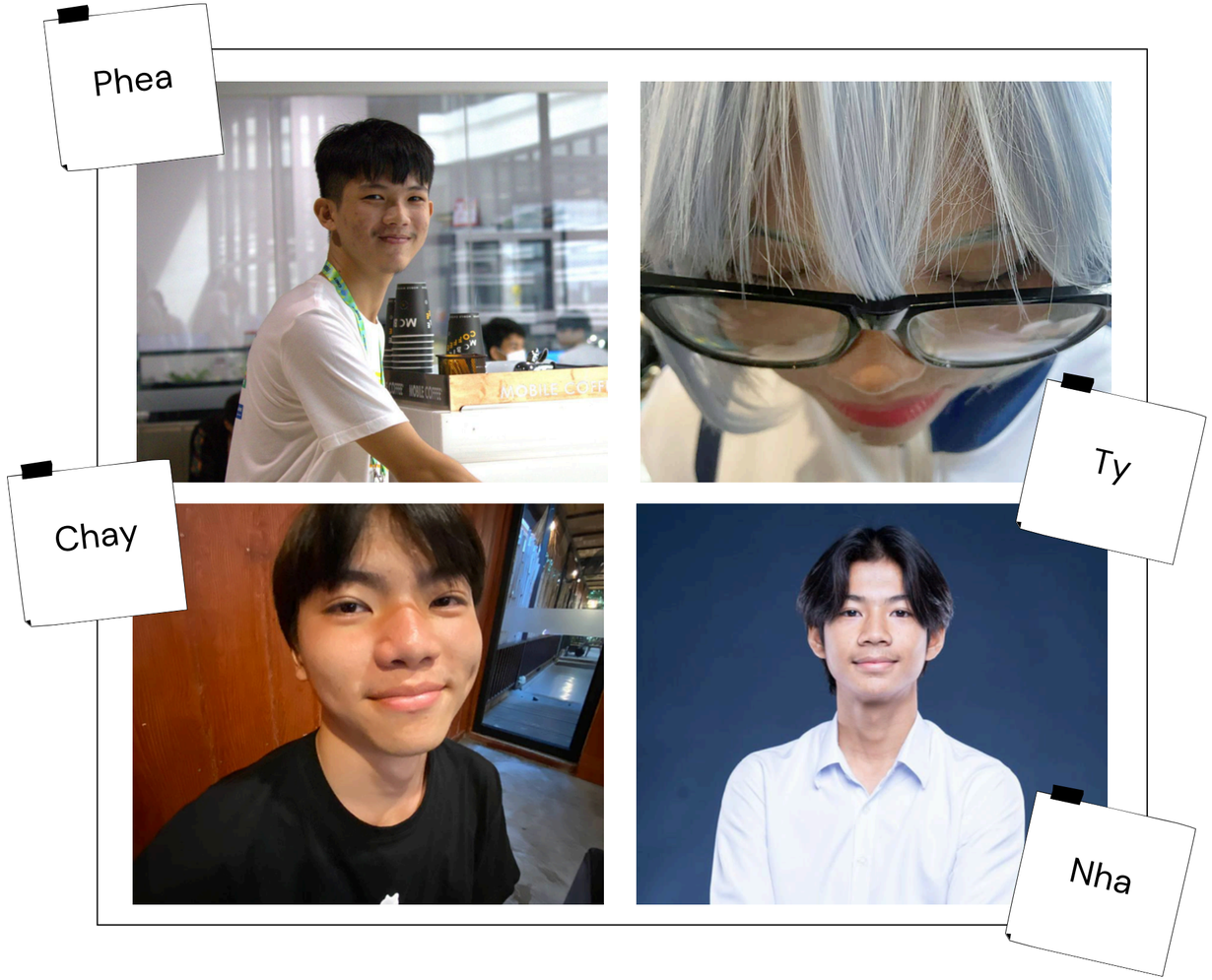


Lecturer: Natt Korat Date: 25/11/2025

CADT បណ្ឌិត្យសភាបច្ចេកវិទ្យាឌីជីថលកម្ពុជា
Cambodia Academy of Digital Technology

Presented By:

- Kong Sophea
- Chey Naryvety
- Vong sovanpanha
- Lim Vinchay



November 25, 2025

CADT បណ្ឌិត្យសភាបច្ចេកវិទ្យាឌីជីថលកម្ពុជា
Cambodia Academy of Digital Technology

Content

November 25, 2025

Overview

Implementing Knock-Out Tournament Bracket Manager

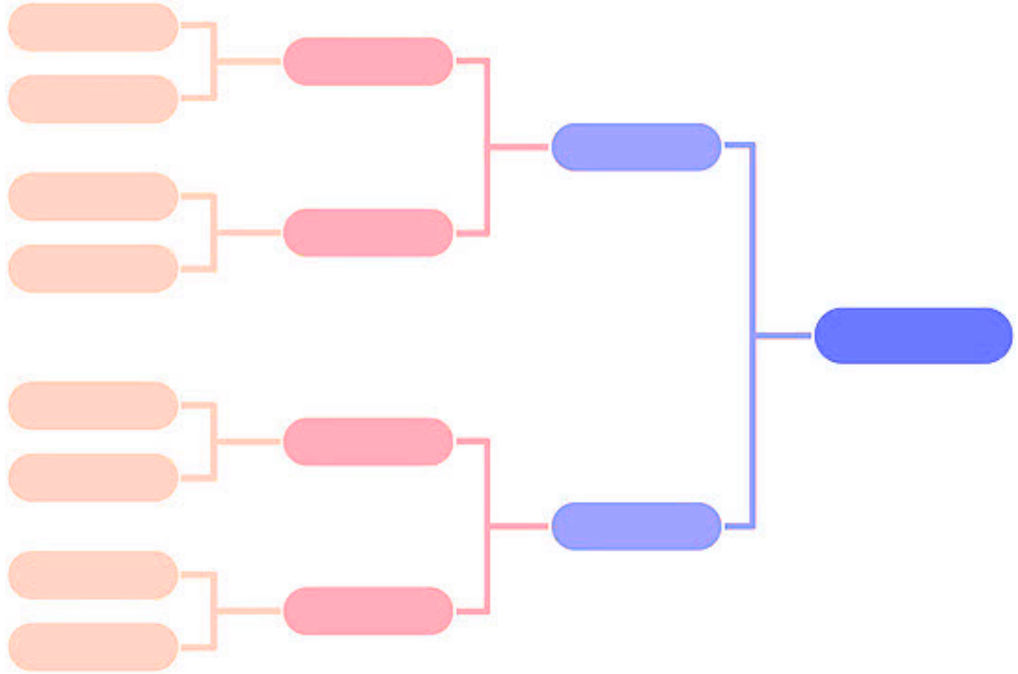
1 System Purpose

2 Function Capabilities

3 Validation Mechanism

In this project we will be implementing a single-elimination tournament using a **binary tree** structure. **Leaves:** Players, **internal nodes:** matches with winner and pad with **BYE** entries if necessary.

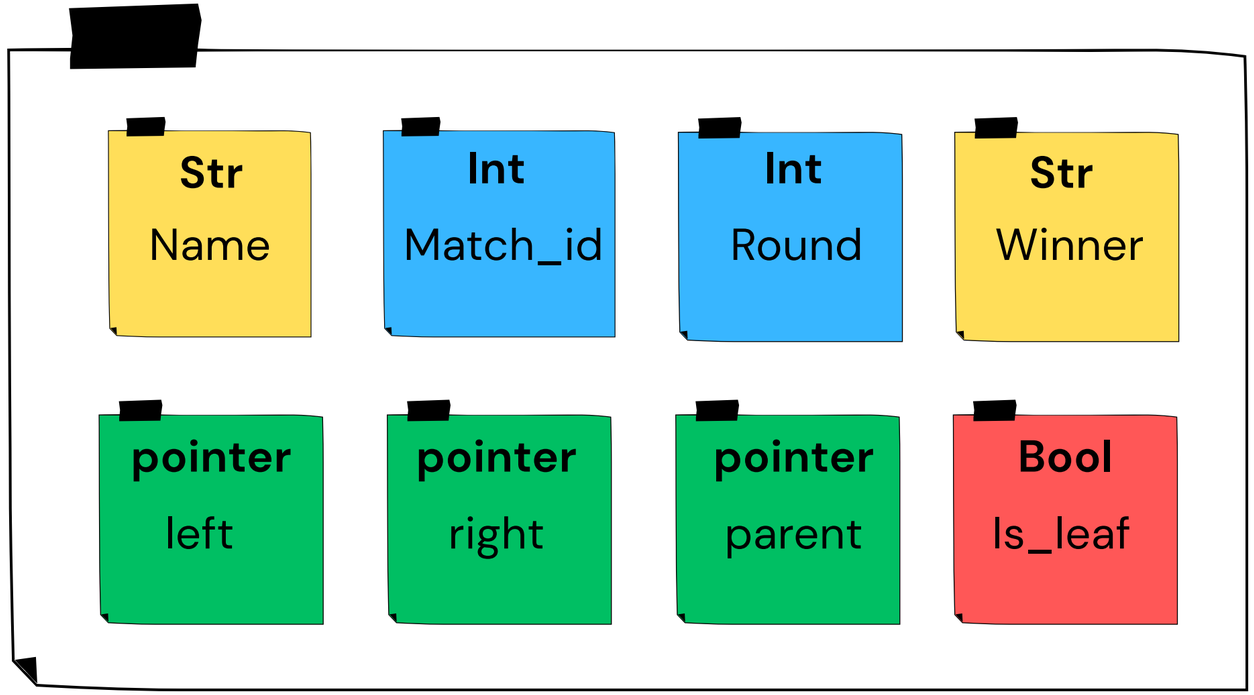
The model supports **5 core functions** to ensures correctness, validation, consistency, and efficient traversal through the tournament framework.



November 25, 2025

Data Structure Design

Using pointer-based binary tree.
Node contains:

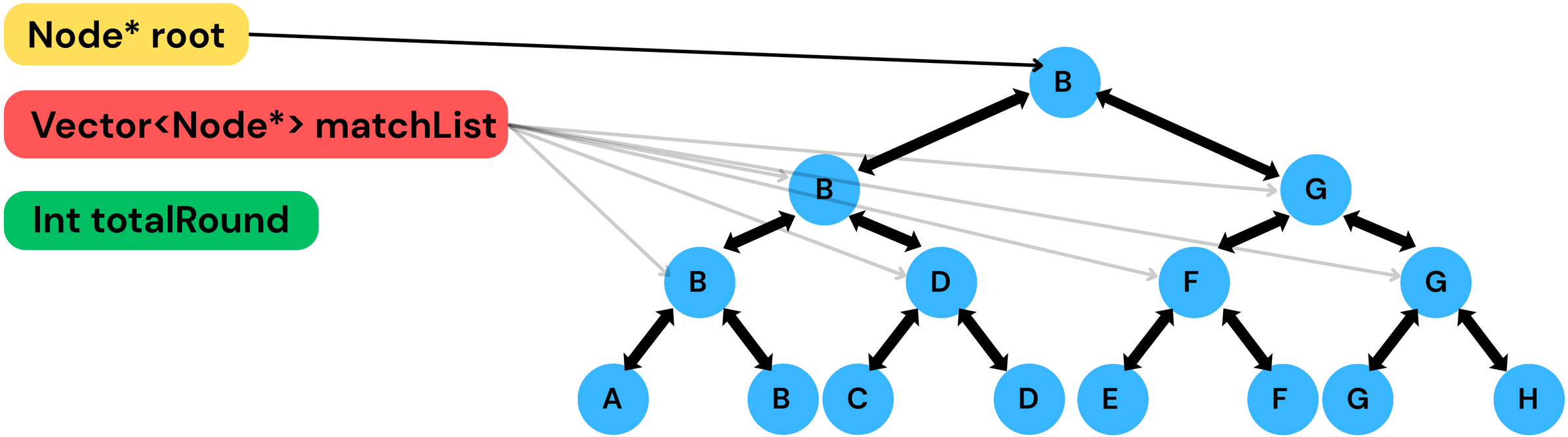



```
struct Node {
    string name;
    int matchId;
    int round;
    string winner;
    Node* left;
    Node* right;
    Node* parent;
    bool isLeaf;
```

November 25, 2025

Data Structure Design

Global variable:



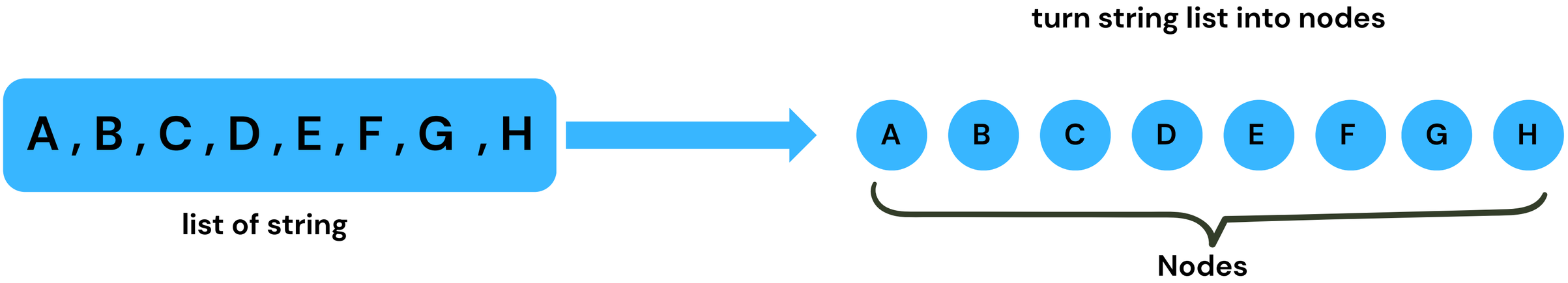
Core Functions

Functions needed to implement

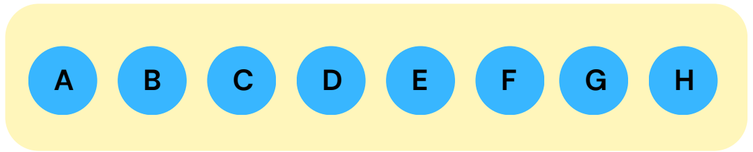
1. Build bracket
2. Record a result
3. Path query
4. Meeting query (LCA)
5. Print bracket (level-order)

November 25, 2025

Build bracket

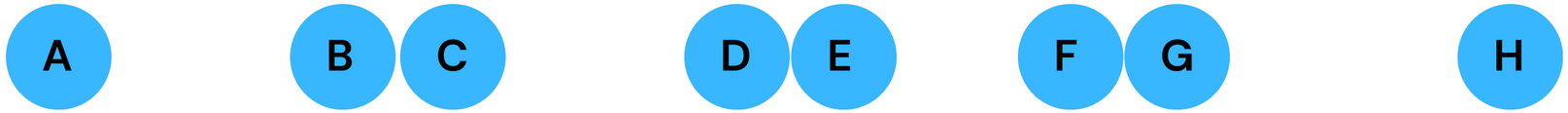


Build bracket

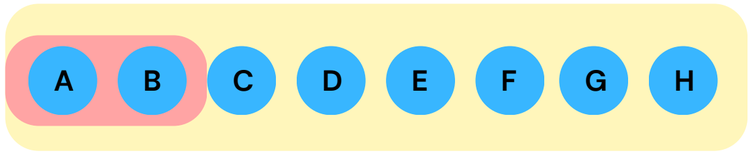


List of Nodes pointer

List of Nodes pointer (Parent)



Build bracket

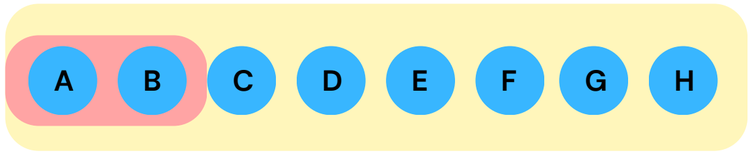


List of Nodes pointer

List of Nodes pointer (Parent)

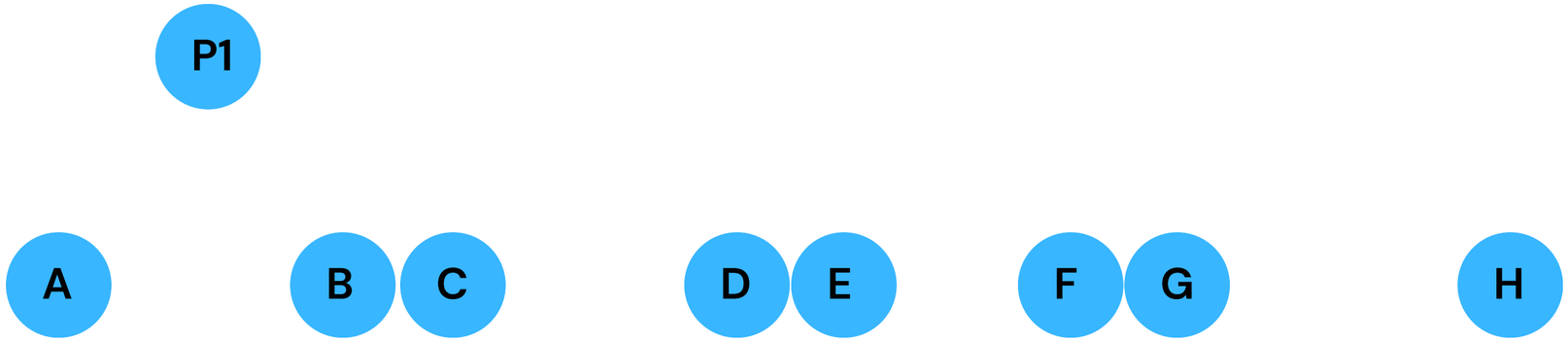
A B C D E F G H

Build bracket

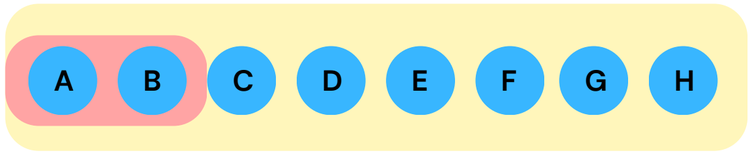


List of Nodes pointer

List of Nodes pointer (Parent)

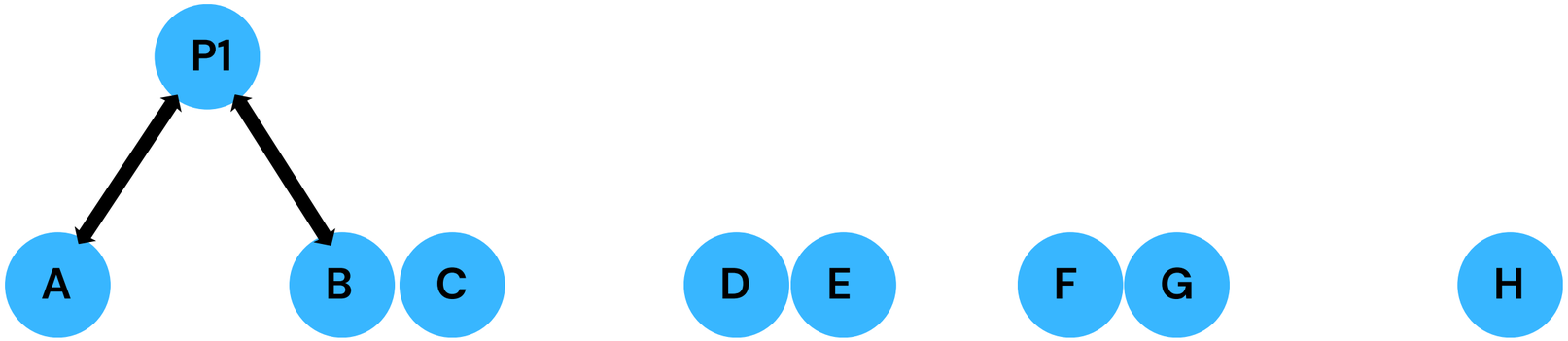


Build bracket

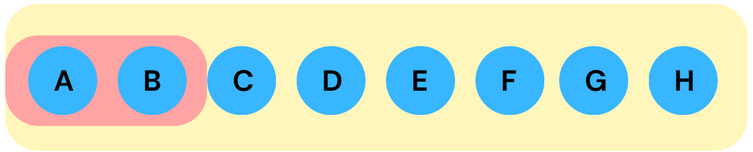


List of Nodes pointer

List of Nodes pointer (Parent)

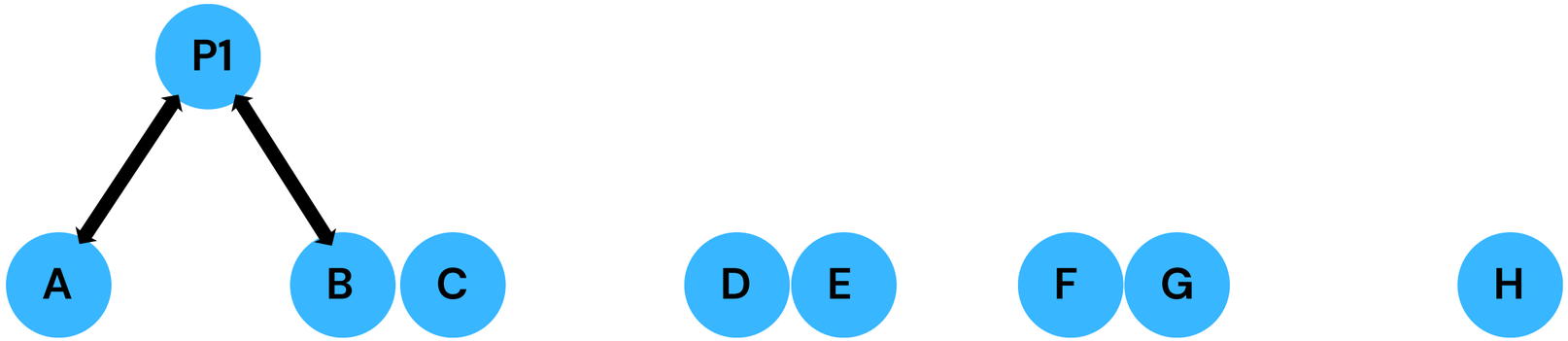


Build bracket

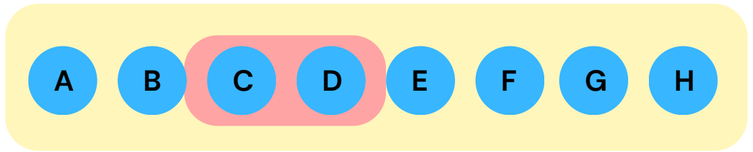


List of Nodes pointer

List of Nodes pointer (Parent)

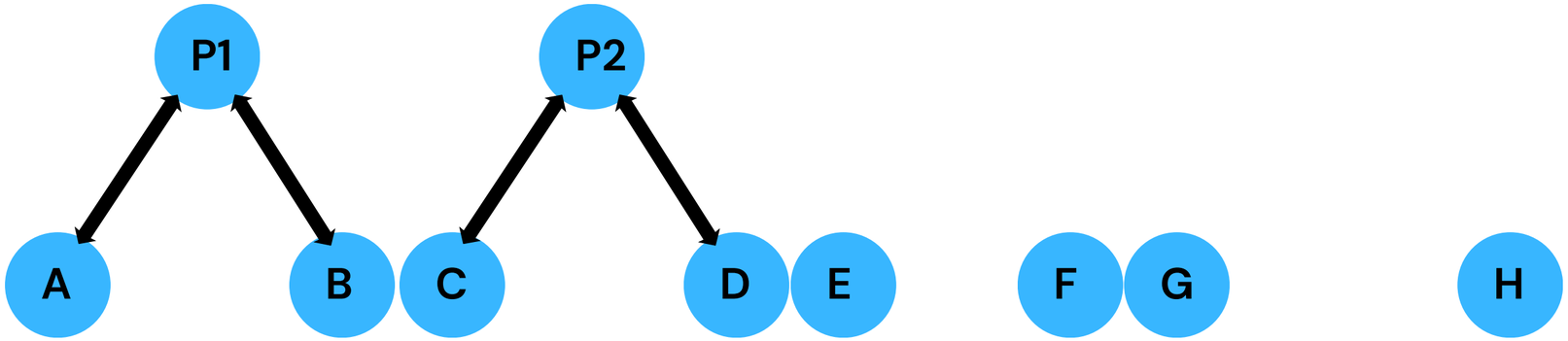


Build bracket

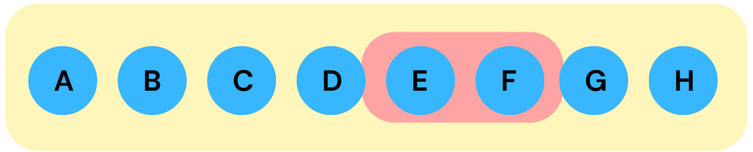


List of Nodes pointer

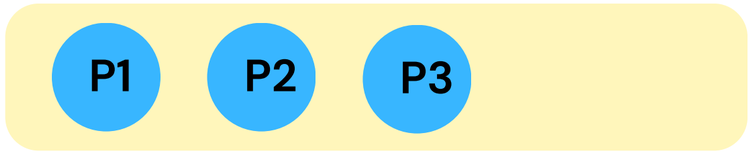
List of Nodes pointer (Parent)



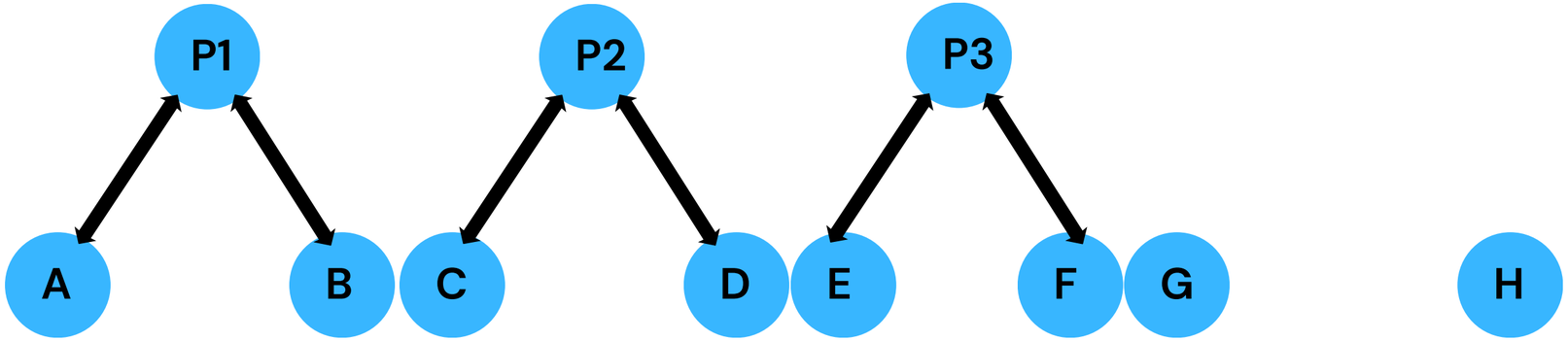
Build bracket

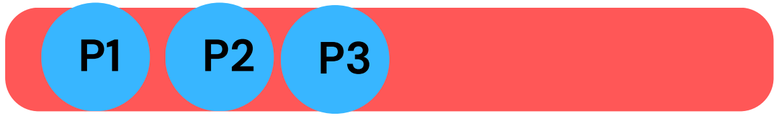


List of Nodes pointer

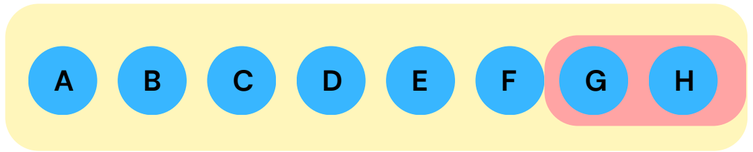


List of Nodes pointer (Parent)

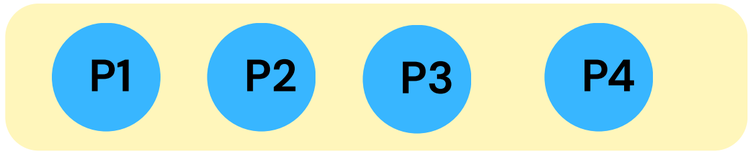




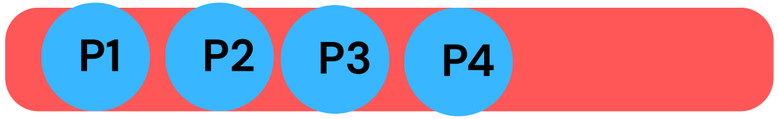
Build bracket

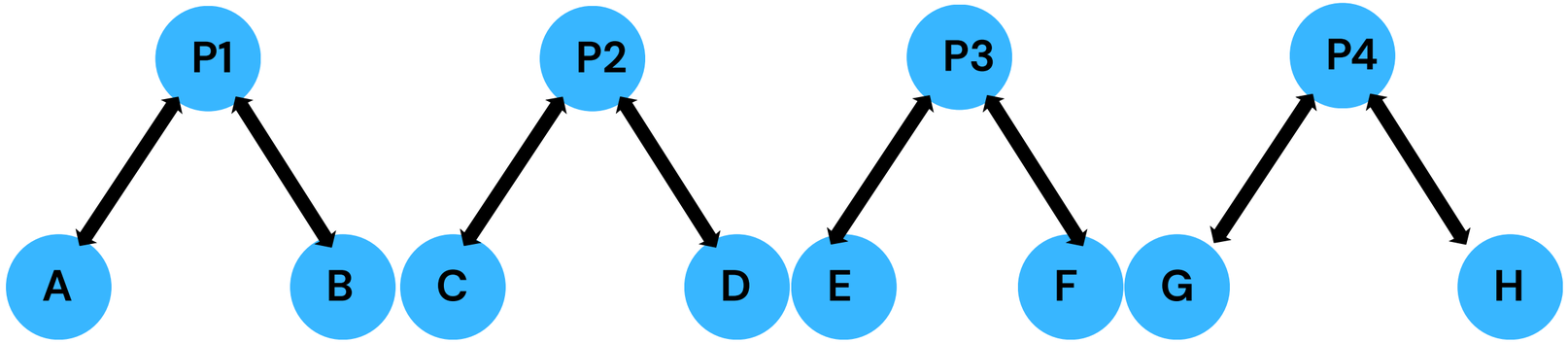


List of Nodes pointer

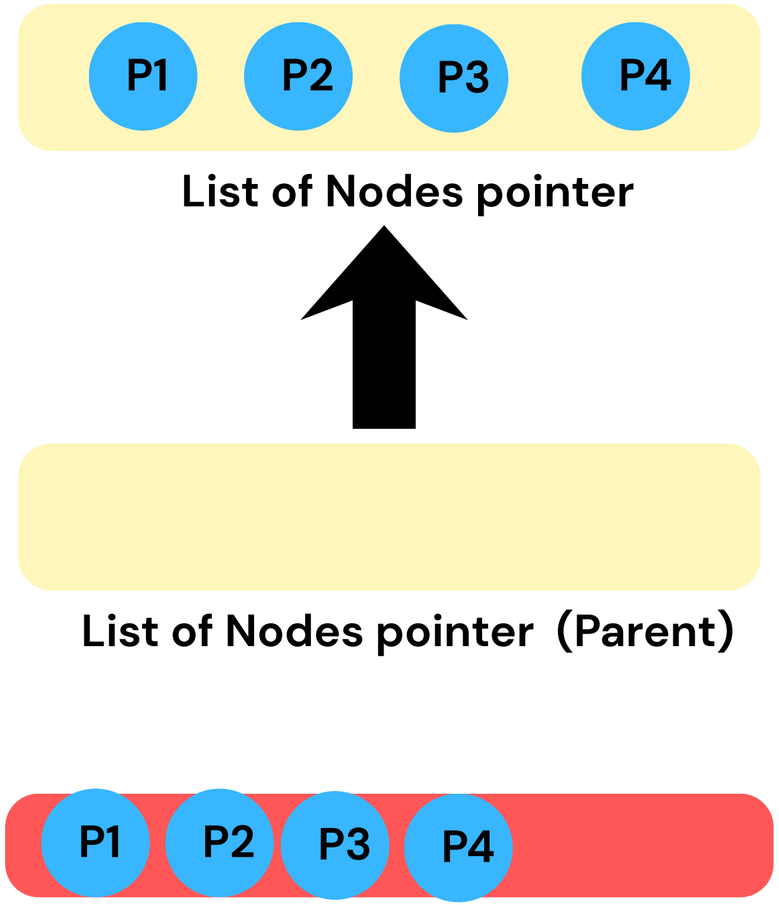


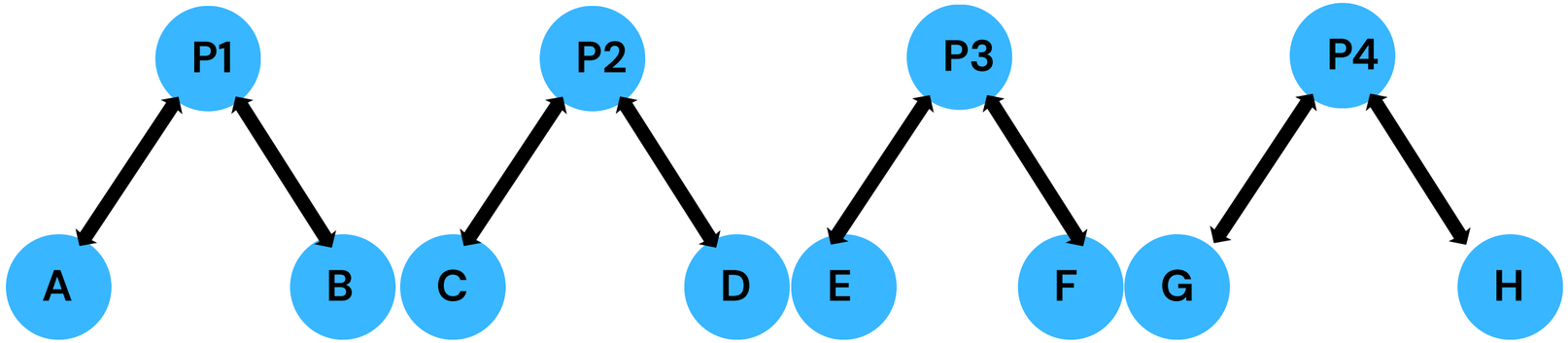
List of Nodes pointer (Parent)



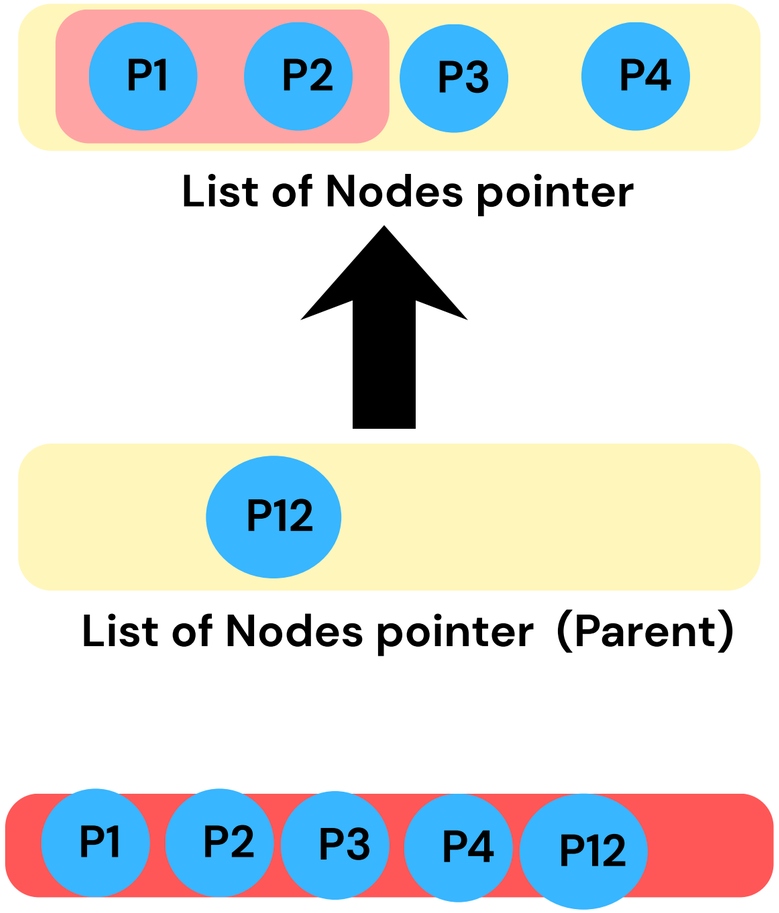


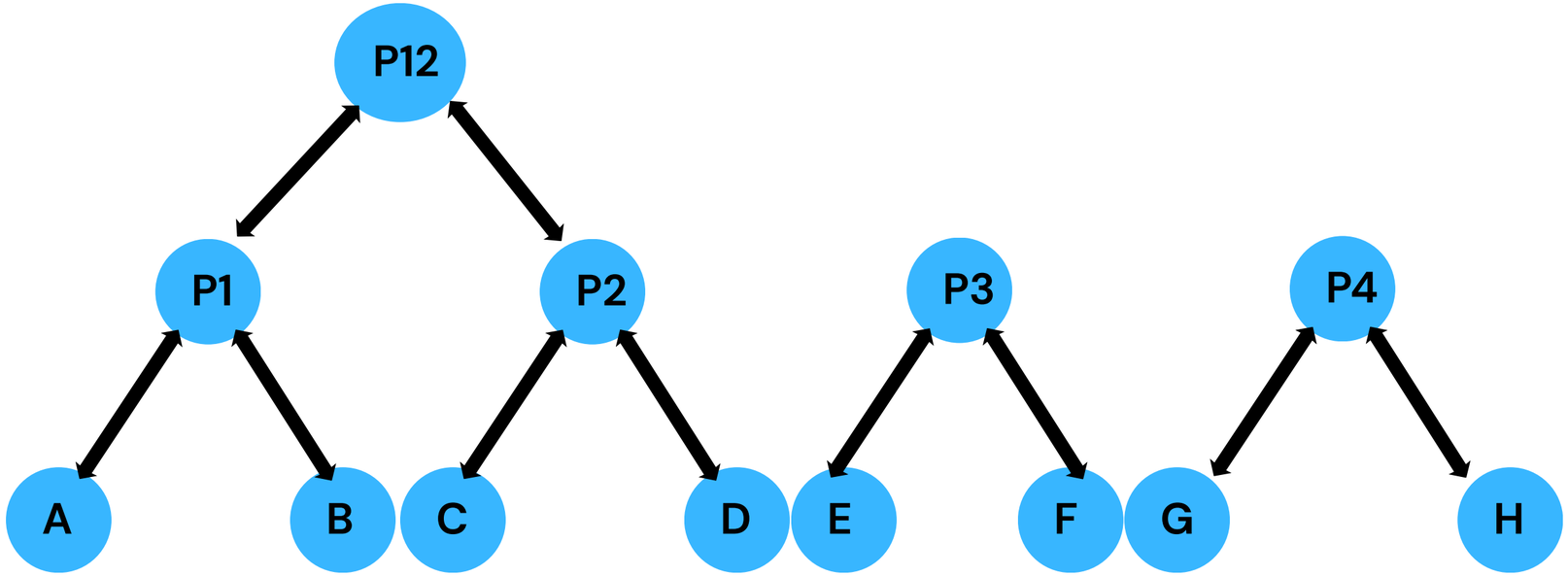
Build bracket



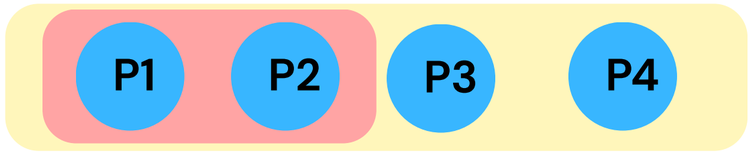


Build bracket

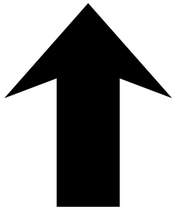


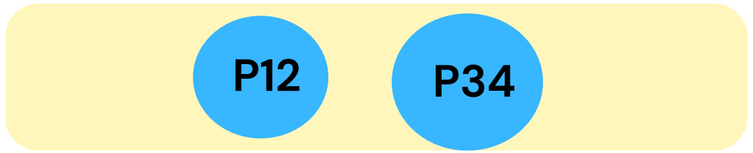


Build bracket

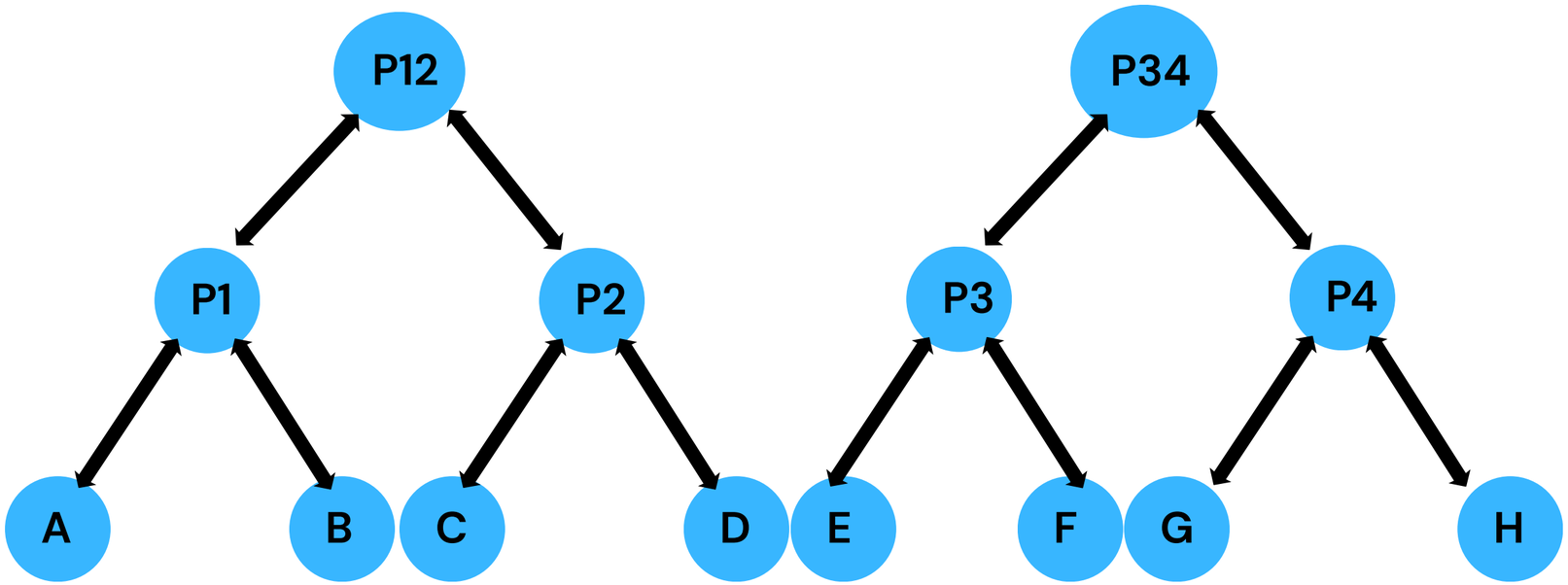


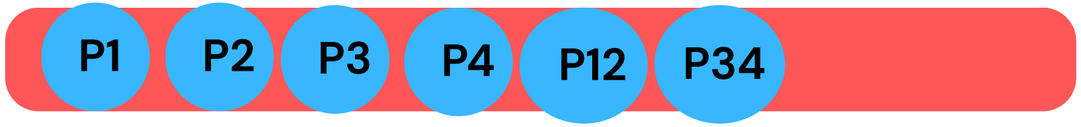
List of Nodes pointer



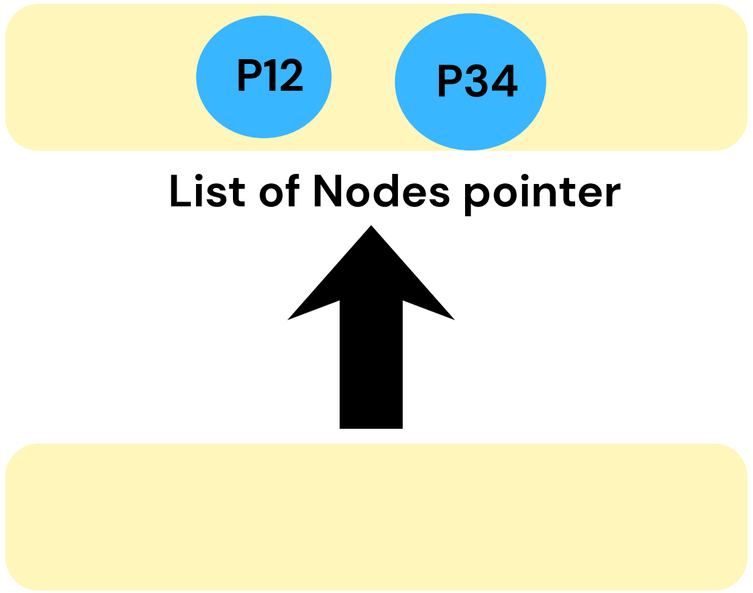


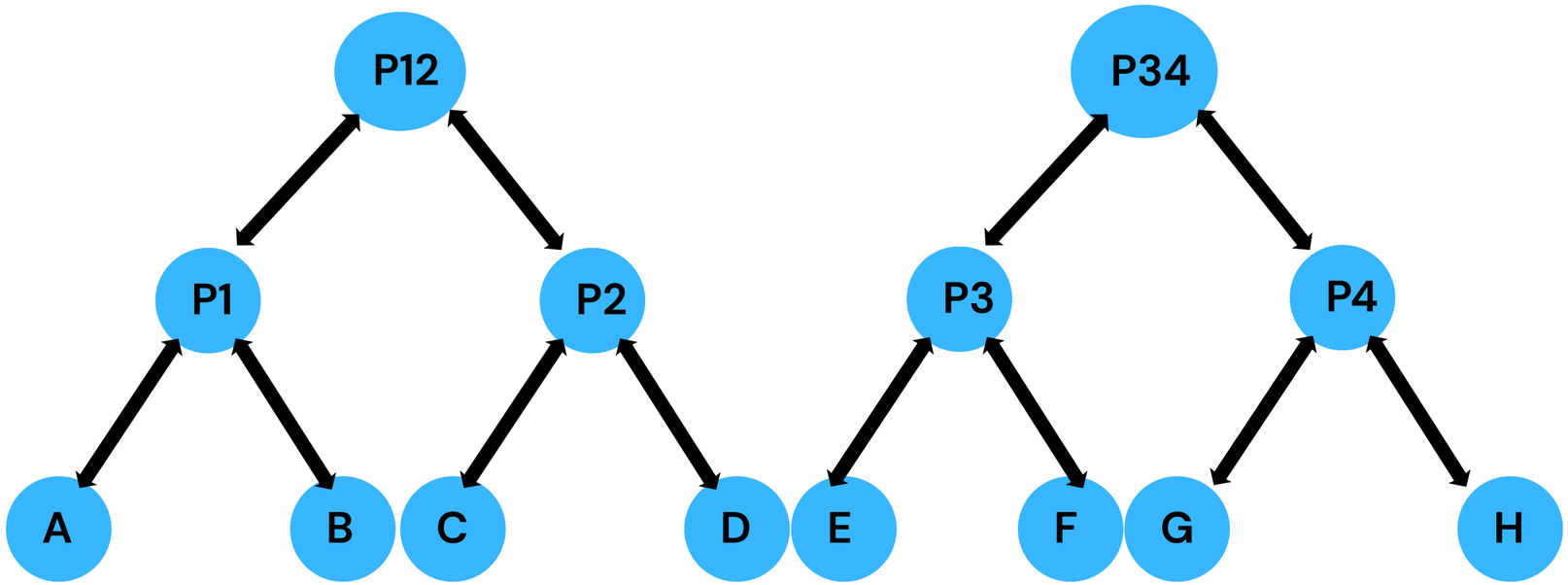
List of Nodes pointer (Parent)

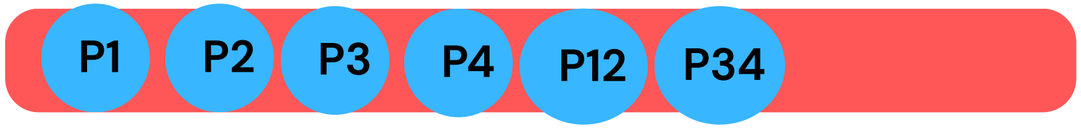




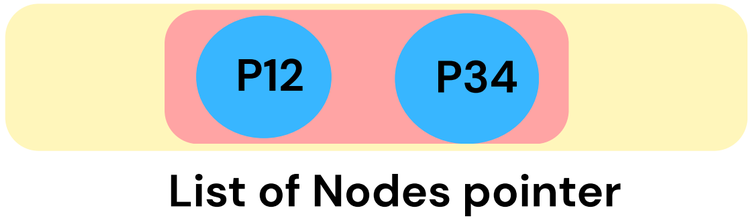
Build bracket



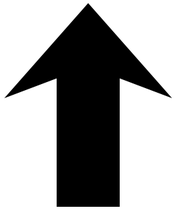


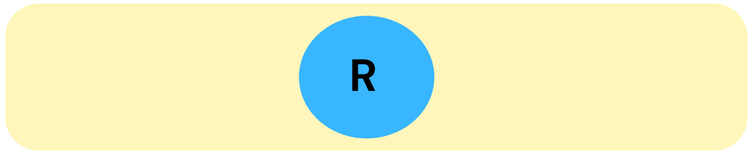


Build bracket

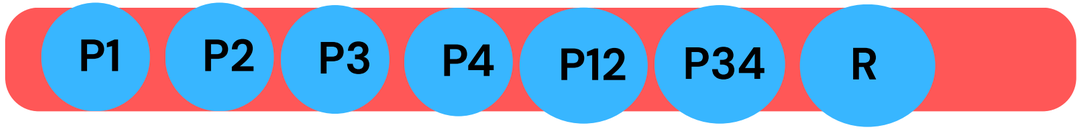


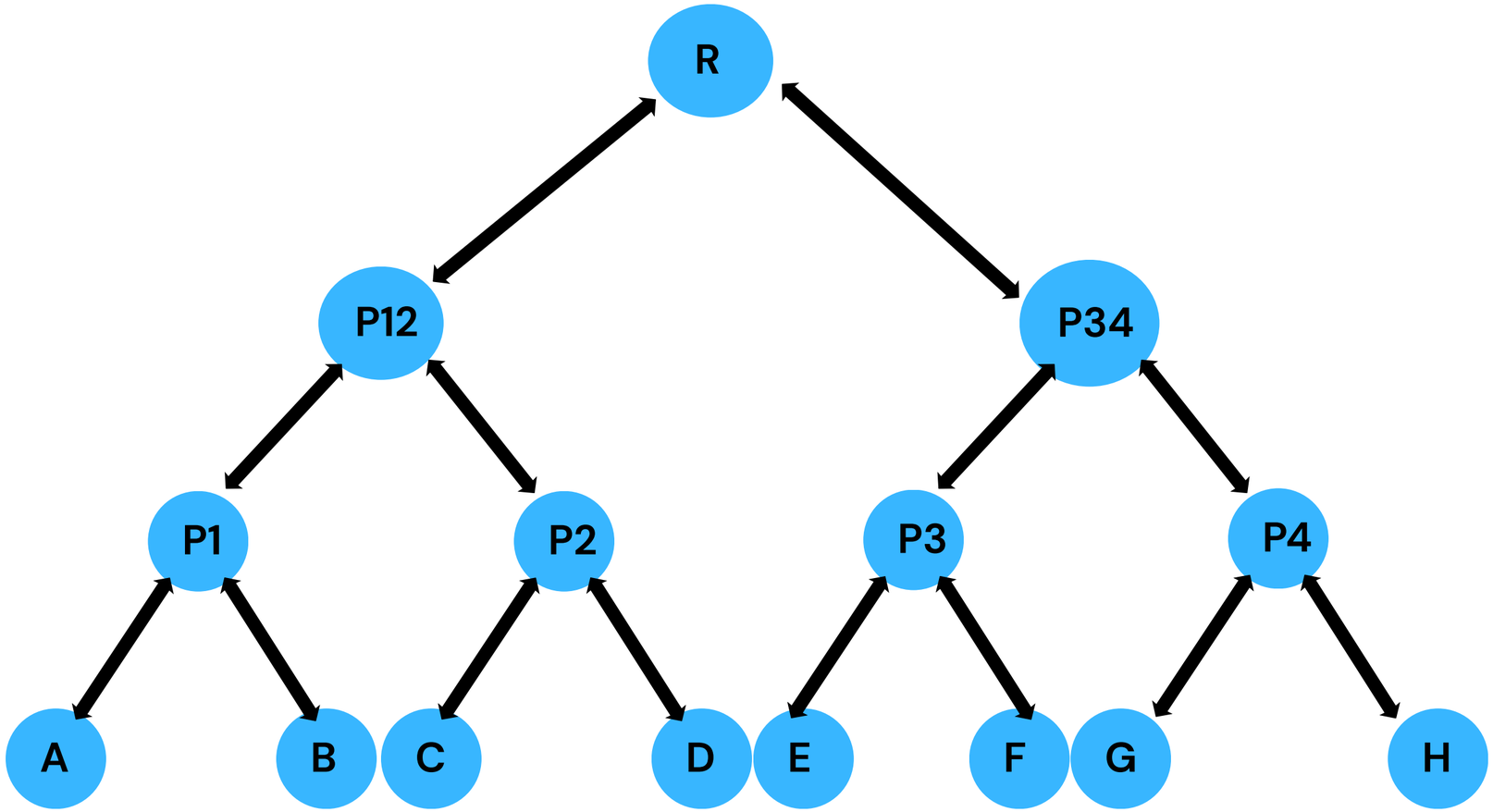
List of Nodes pointer



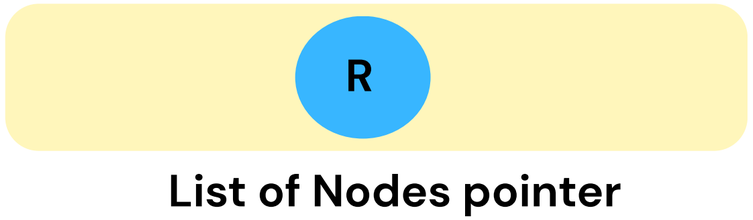


List of Nodes pointer (Parent)

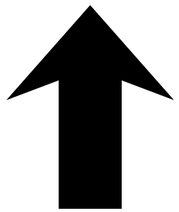




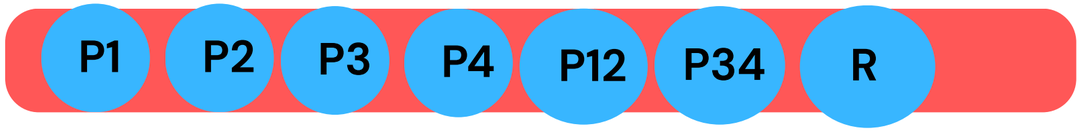
Build bracket

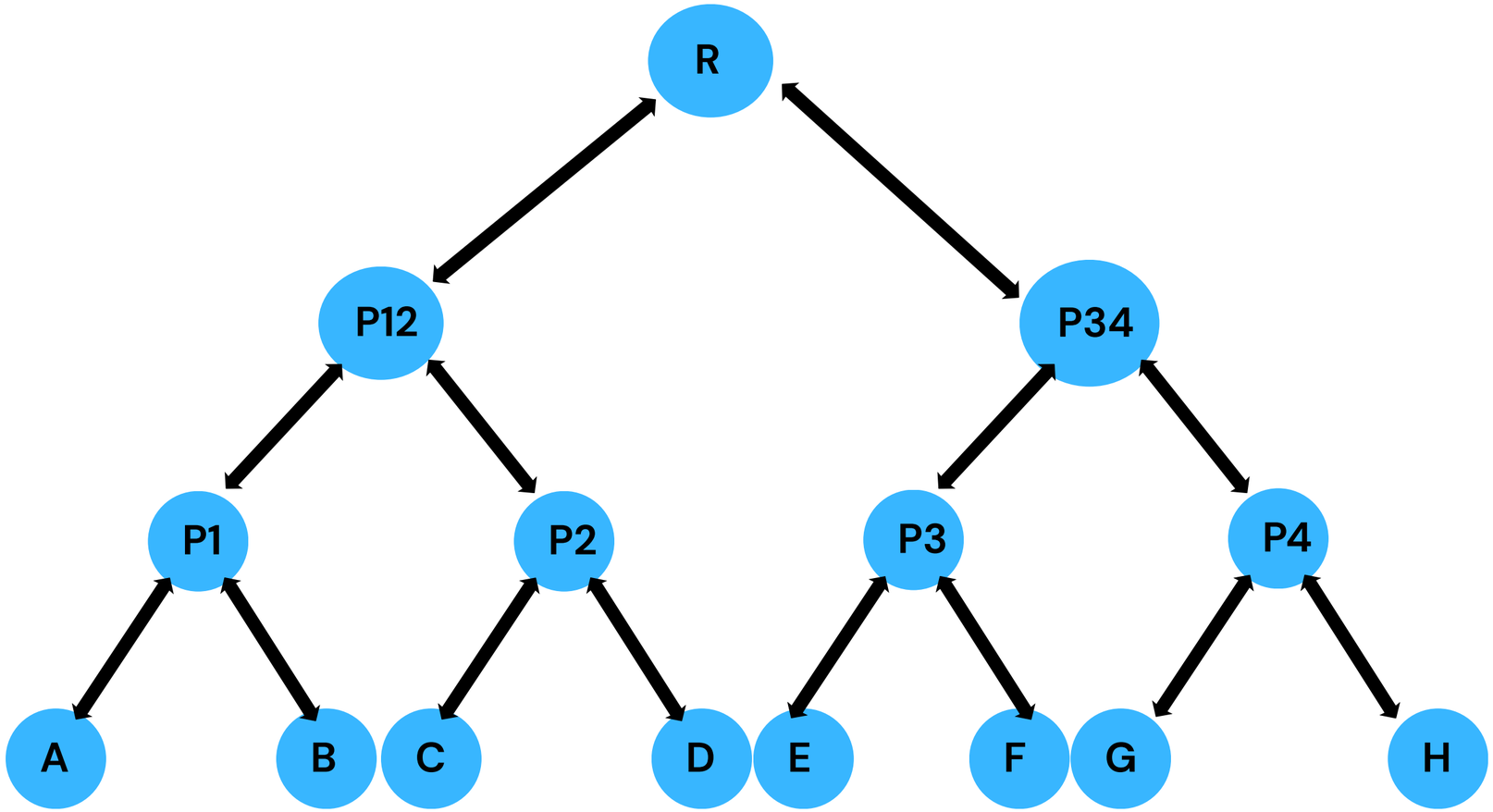


List of Nodes pointer



List of Nodes pointer (Parent)



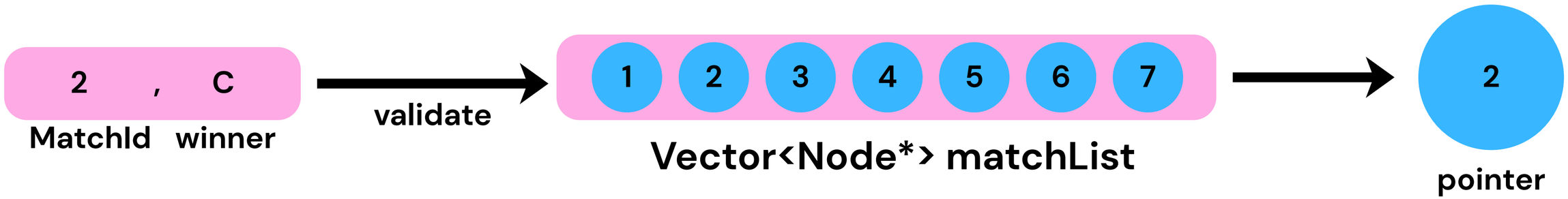


Codes

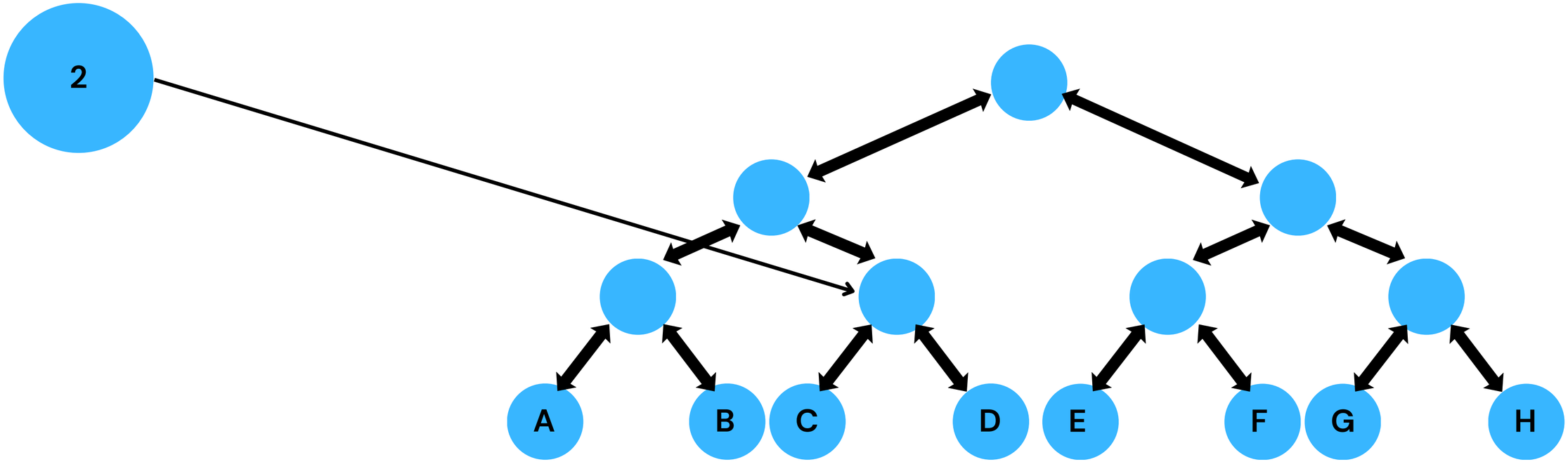
1 Build Bracket

```
Node* buildBracket(const vector<string>& players) {
    int n = players.size();
    int m = 1;
    while (m < n) m *= 2;

    vector<Node*> leaves;
    for (int i = 0; i < m; i++) {
        leaves.push_back(new Node(i < n ? players[i] : "BYE"));
    }

    matchList.clear();
    int matchCounter = 1;
    int round = 0;

    vector<Node*> curr = leaves;
    while (curr.size() > 1) {
        vector<Node*> next;
        round++;

        for (size_t i = 0; i < curr.size(); i += 2) {
            Node* m = new Node();
            m->isLeaf = false;
            m->left = curr[i];
            m->right = curr[i+1];
            curr[i]->parent = m;
            curr[i+1]->parent = m;
            m->matchId = matchCounter++;
            m->round = round;
            matchList.push_back(m);
            next.push_back(m);
        }
        curr.swap(next);
    }

    totalRounds = round;
    root = curr[0];
    return root;
}
```

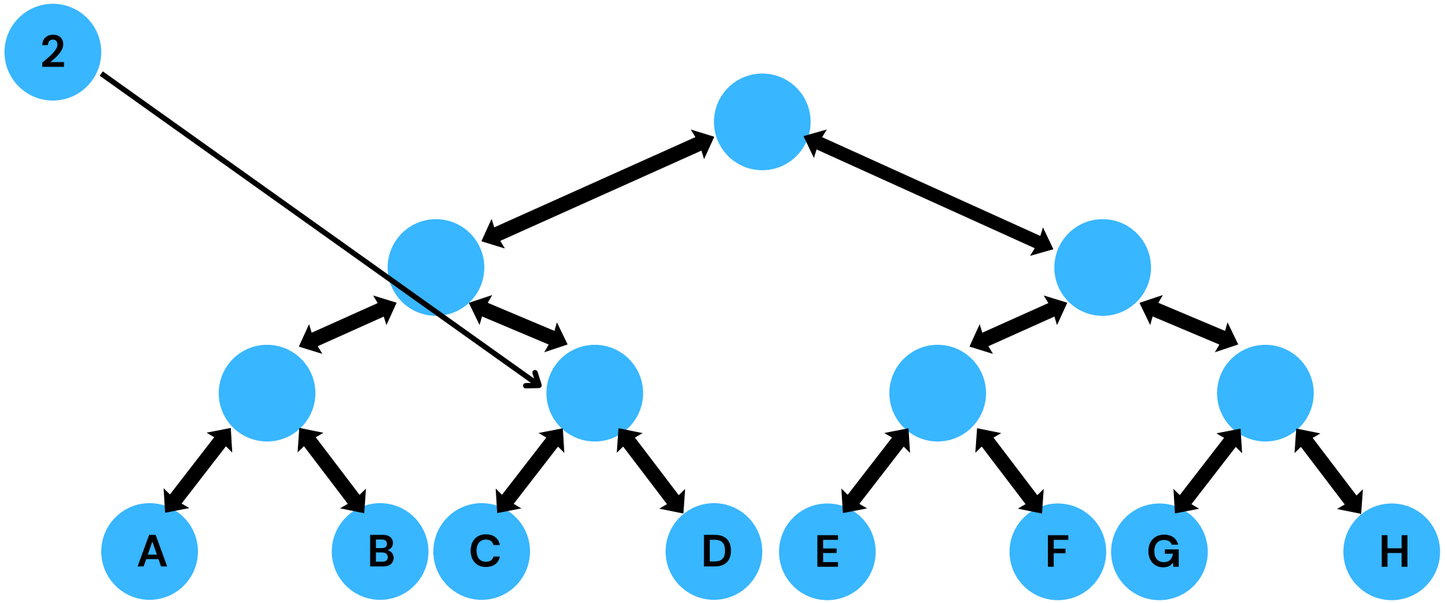
Record a result



Record a result



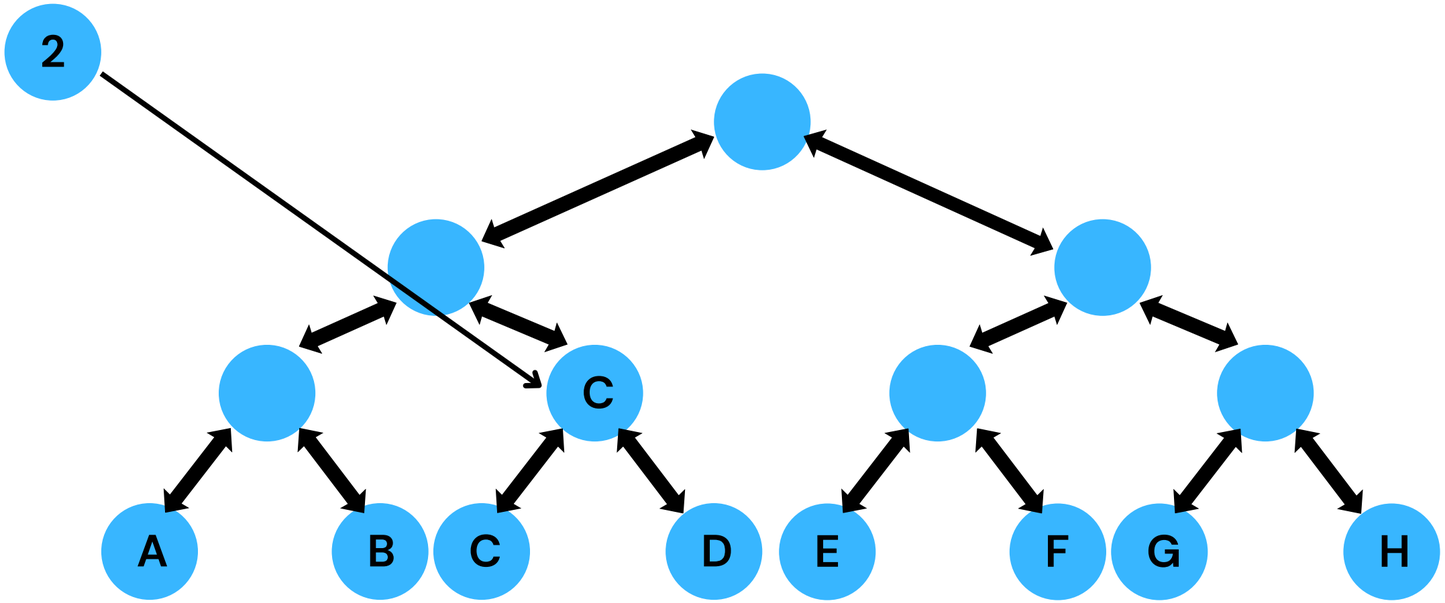
Record a result



Record a result

we assign the other child as the winner.

assign the winner name according to the parameter.



Codes

2 Record Match

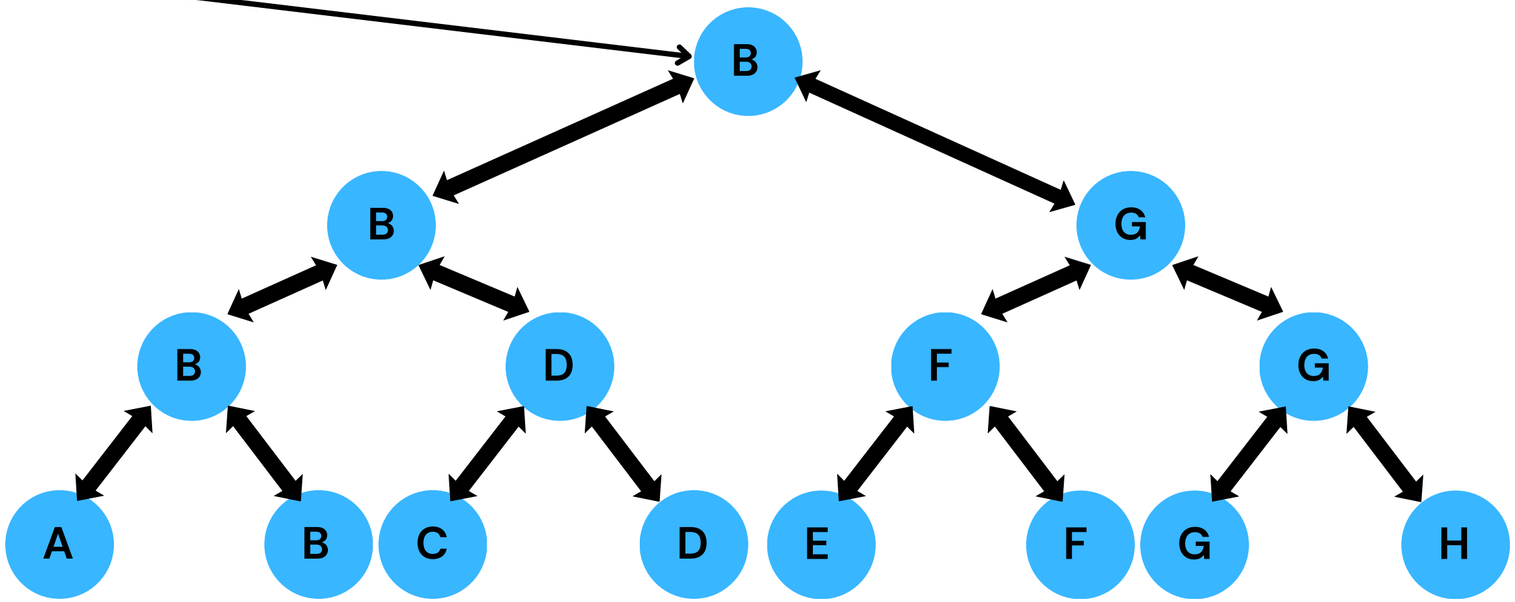
```
bool recordResult(int matchId, const string& winnerName) {
    if (matchId <= 0 || matchId > (int)matchList.size()) return false;
    Node* m = matchList[matchId-1];
    if (m->winner != "?") return false;

    string L = m->left->isLeaf ? m->left->name : m->left->winner;
    string R = m->right->isLeaf ? m->right->name : m->right->winner;

    if (L == "?" || R == "?") return false;

    if (L == "BYE") m->winner = R;
    else if (R == "BYE") m->winner = L;
    else if (winnerName == L || winnerName == R) m->winner = winnerName;
    else return false;
```

```
// Push winner up the tree automatically for BYEs
Node* parent = m->parent;
while (parent) {
    string a = parent->left->isLeaf ? parent->left->name : parent->left->winner;
    string b = parent->right->isLeaf ? parent->right->name : parent->right->winner;

    if (a == "?" || b == "?") break;
    if (parent->winner != "?") break;

    if (a == "BYE") parent->winner = b;
    else if (b == "BYE") parent->winner = a;
    else break;

    parent = parent->parent;
}

return true;
```

Find Node

node , D

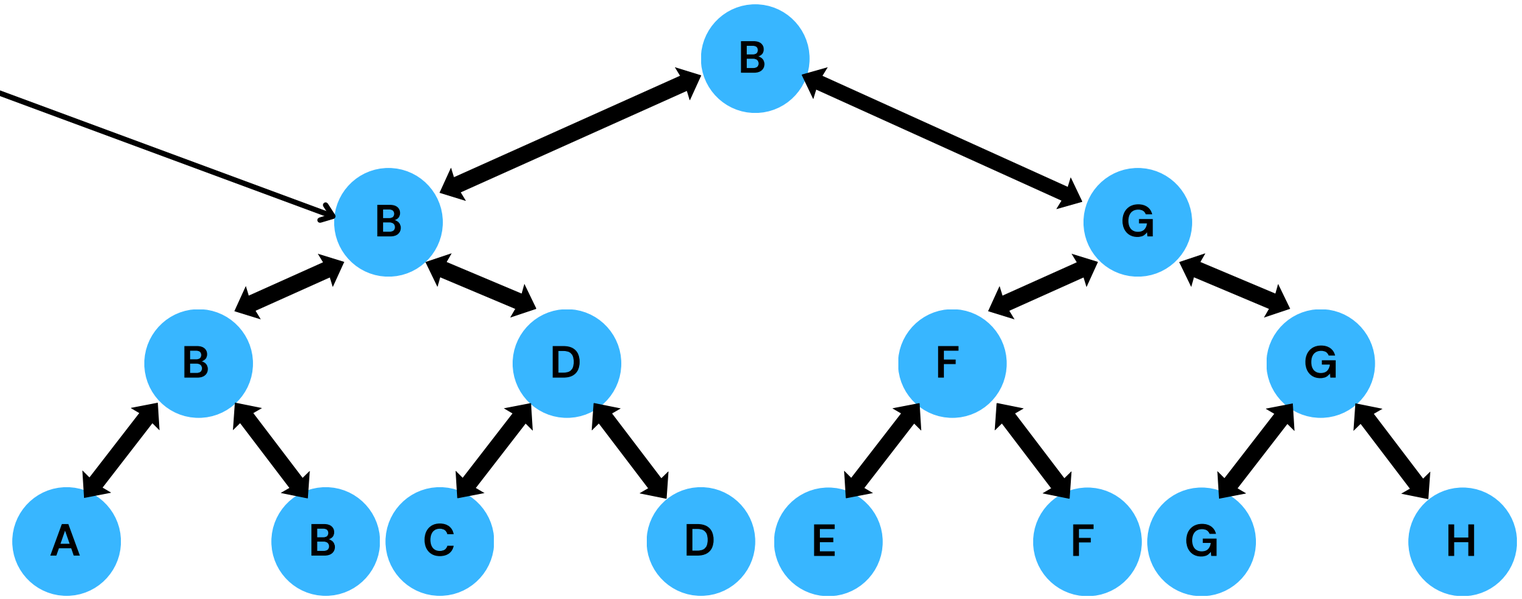
```
Node* findLeaf(Node* node, const string& name) {
    if (!node) return nullptr;
    if (node->isLeaf) return node->name == name ? node : nullptr;
    Node* L = findLeaf(node->left, name);
    return L ? L : findLeaf(node->right, name);
}
```



Find Node

node : D

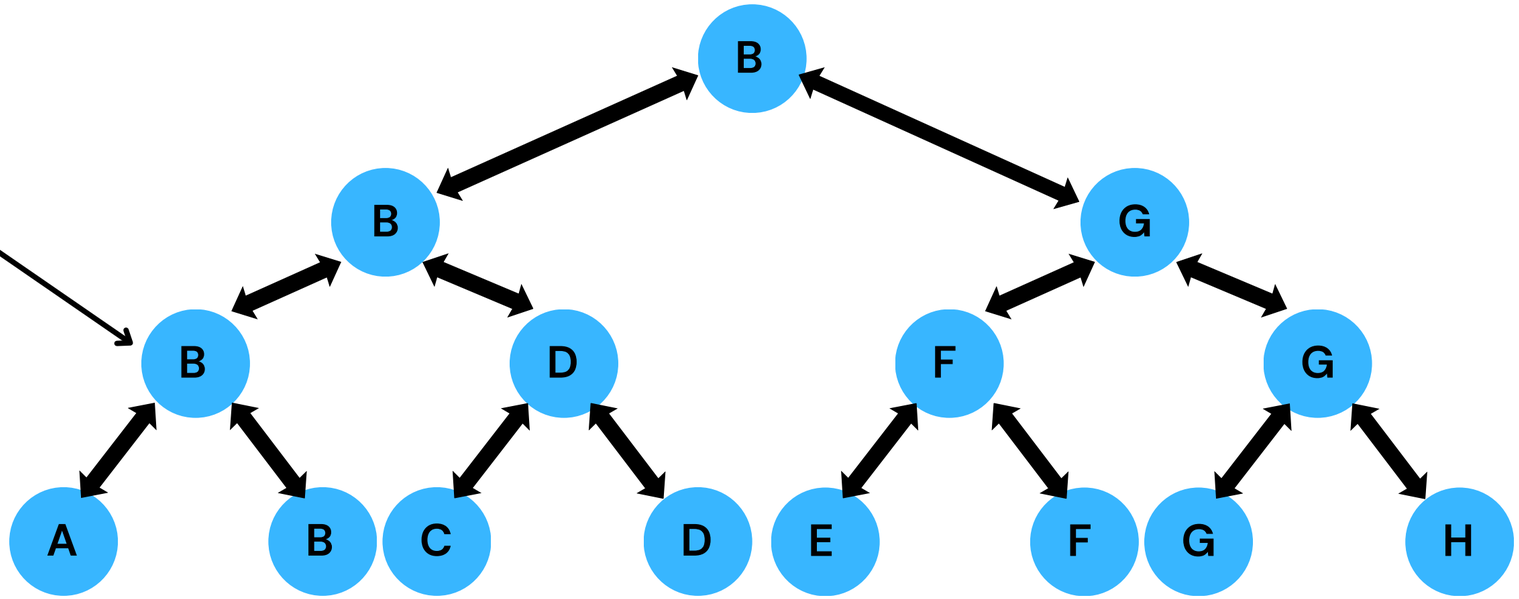
```
Node* findLeaf(Node* node, const string& name) {
    if (!node) return nullptr;
    if (node->isLeaf) return node->name == name ? node : nullptr;
    Node* L = findLeaf(node->left, name);
    return L ? L : findLeaf(node->right, name);
}
```



Find Node

node , D

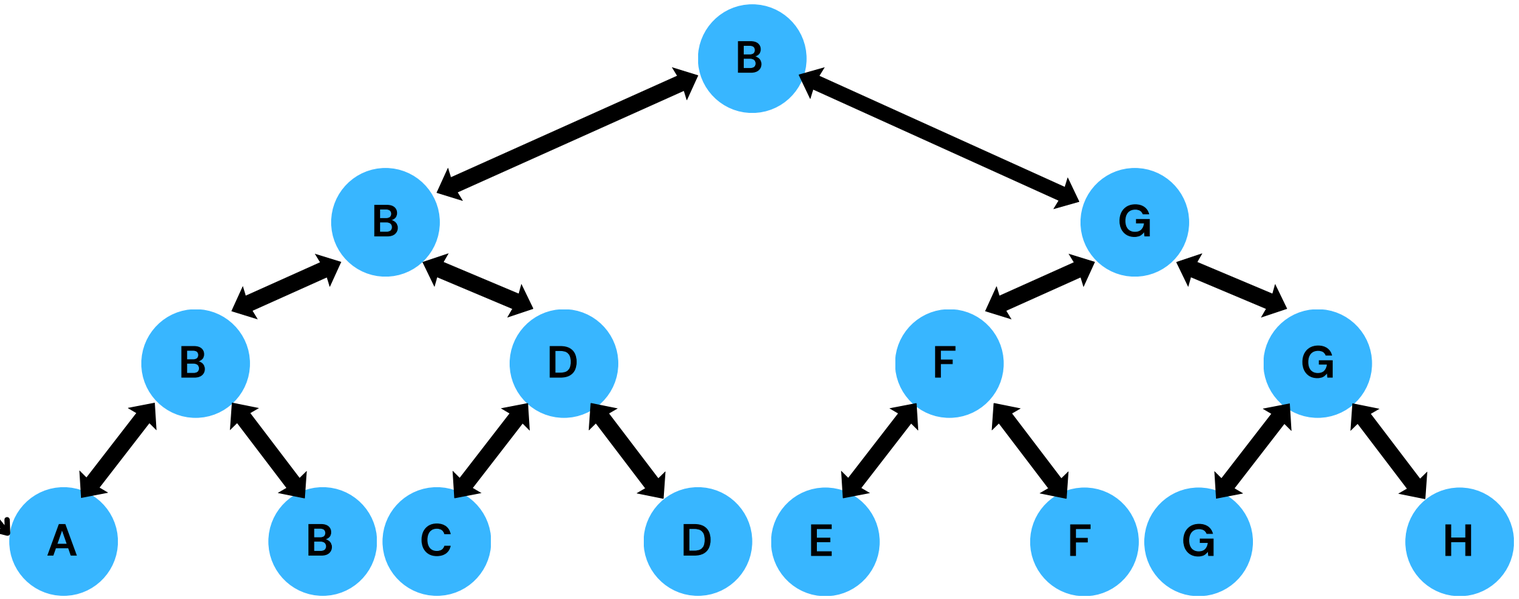
```
Node* findLeaf(Node* node, const string& name) {
    if (!node) return nullptr;
    if (node->isLeaf) return node->name == name ? node : nullptr;
    Node* L = findLeaf(node->left, name);
    return L ? L : findLeaf(node->right, name);
}
```



Find Node

node , D

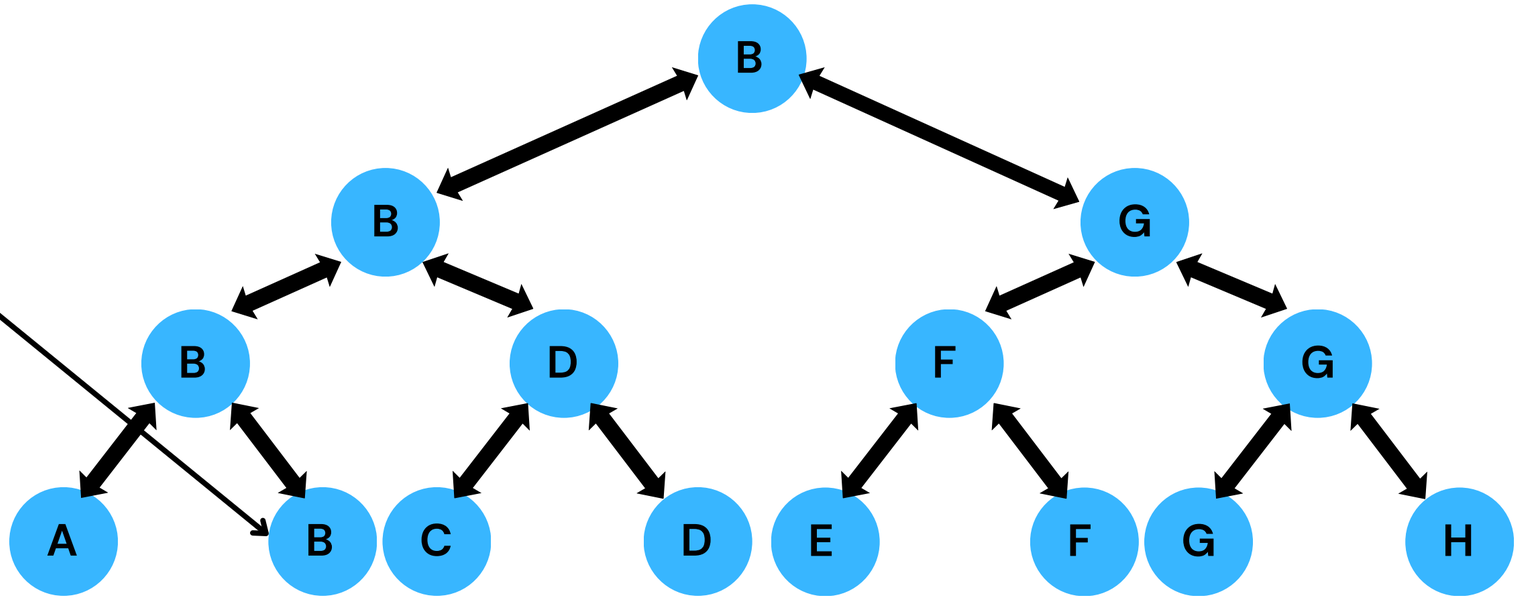
```
Node* findLeaf(Node* node, const string& name) {
    if (!node) return nullptr;
    if (node->isLeaf) return node->name == name ? node : nullptr;
    Node* L = findLeaf(node->left, name);
    return L ? L : findLeaf(node->right, name);
}
```



Find Node

node , D

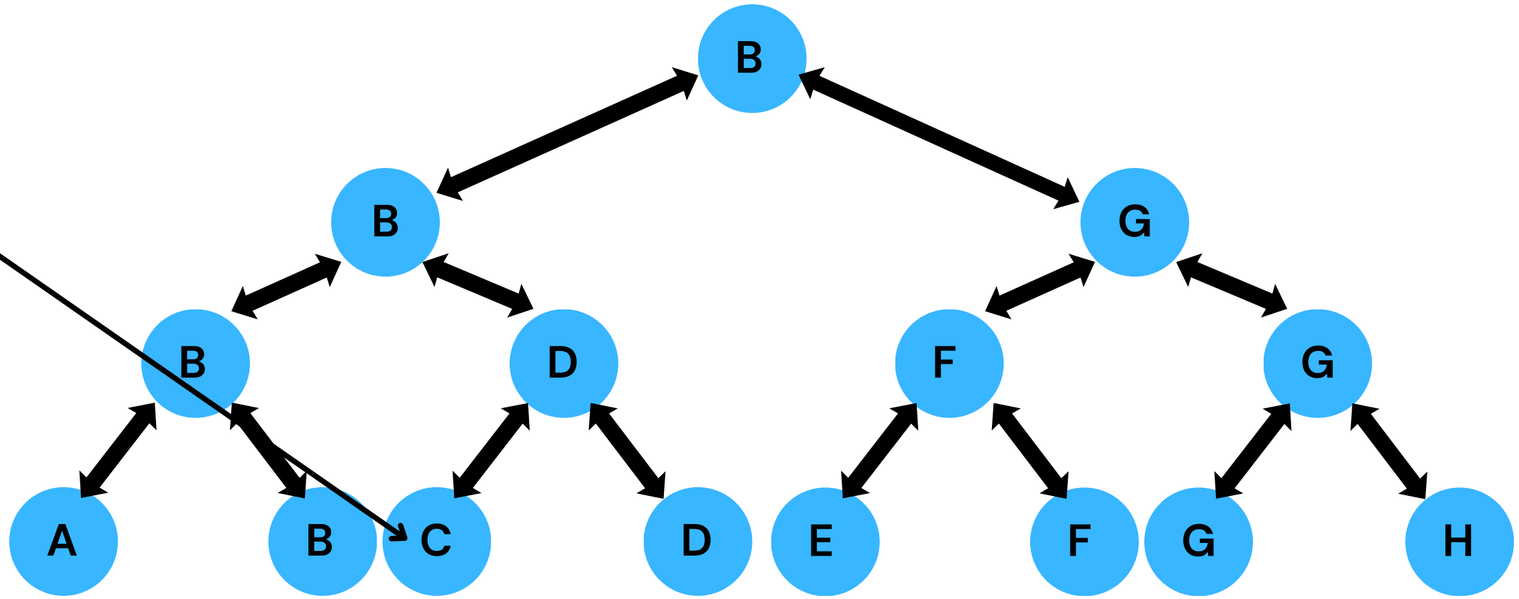
```
Node* findLeaf(Node* node, const string& name) {
    if (!node) return nullptr;
    if (node->isLeaf) return node->name == name ? node : nullptr;
    Node* L = findLeaf(node->left, name);
    return L ? L : findLeaf(node->right, name);
}
```



Find Node

node , D

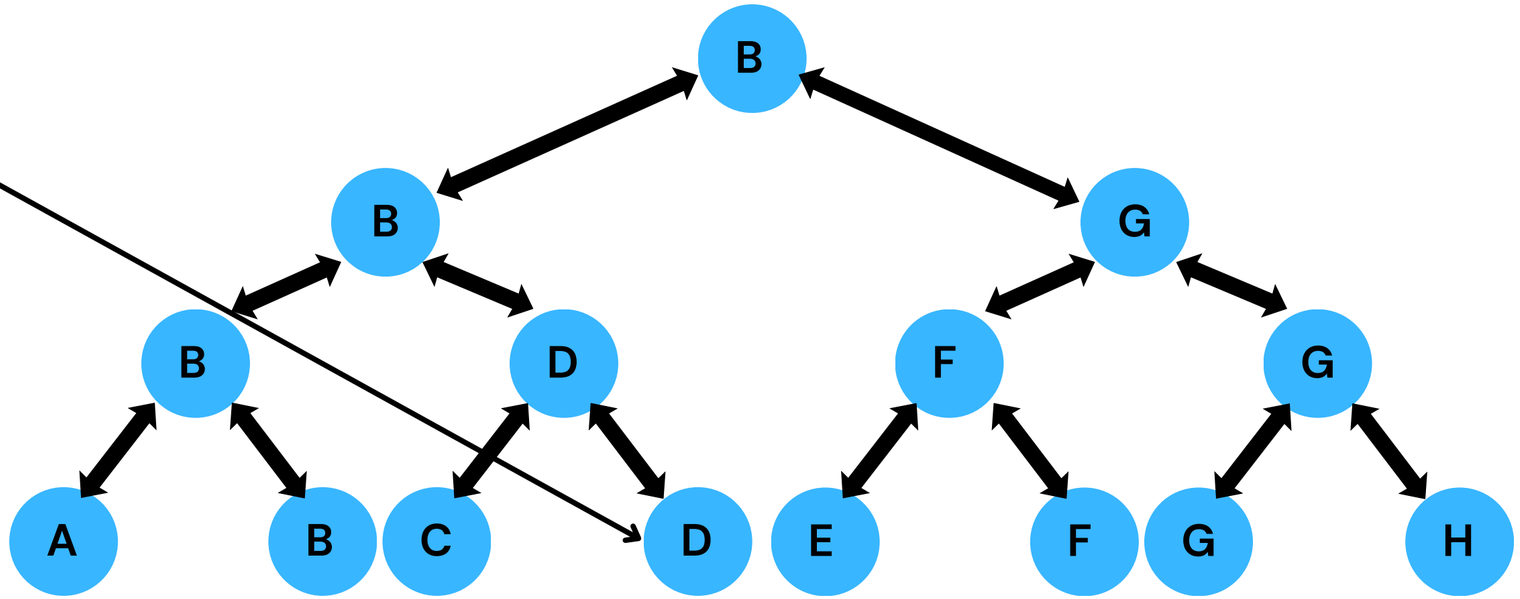
```
Node* findLeaf(Node* node, const string& name) {
    if (!node) return nullptr;
    if (node->isLeaf) return node->name == name ? node : nullptr;
    Node* L = findLeaf(node->left, name);
    return L ? L : findLeaf(node->right, name);
}
```

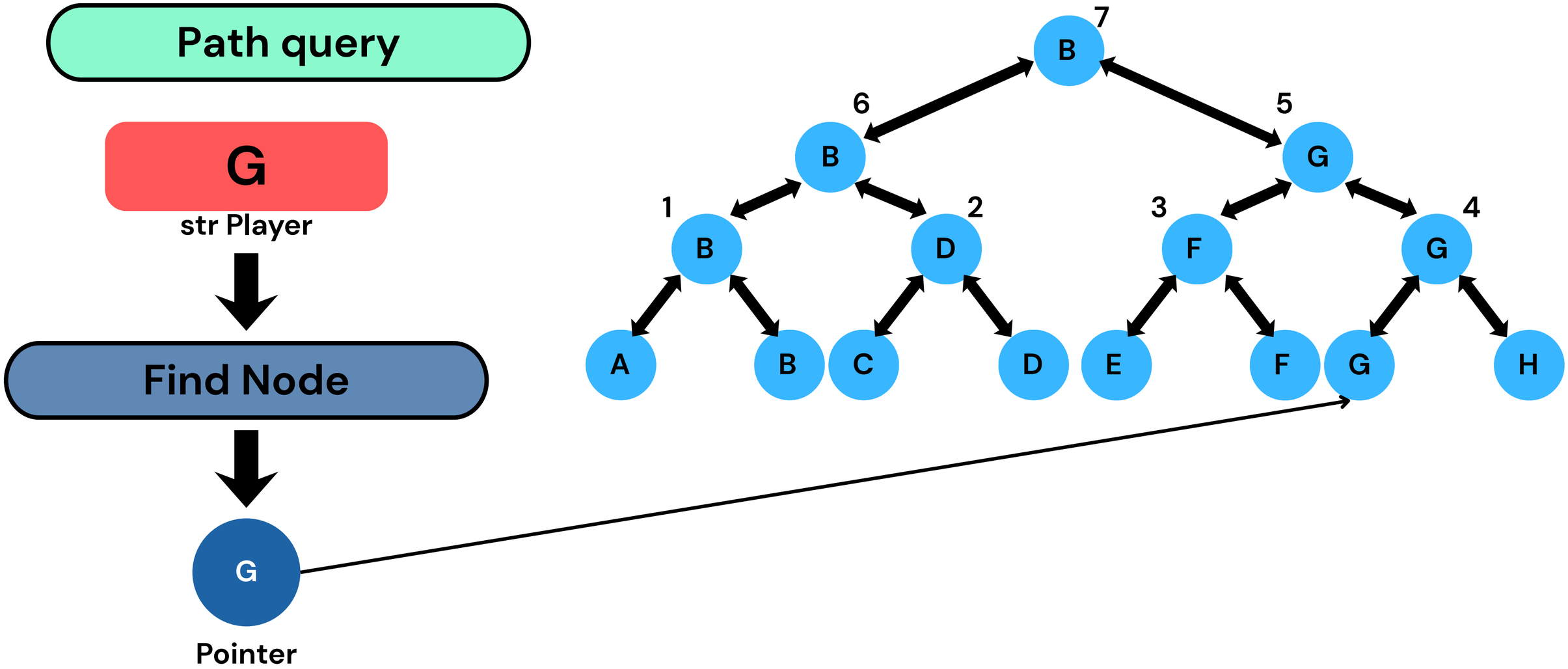


Find Node

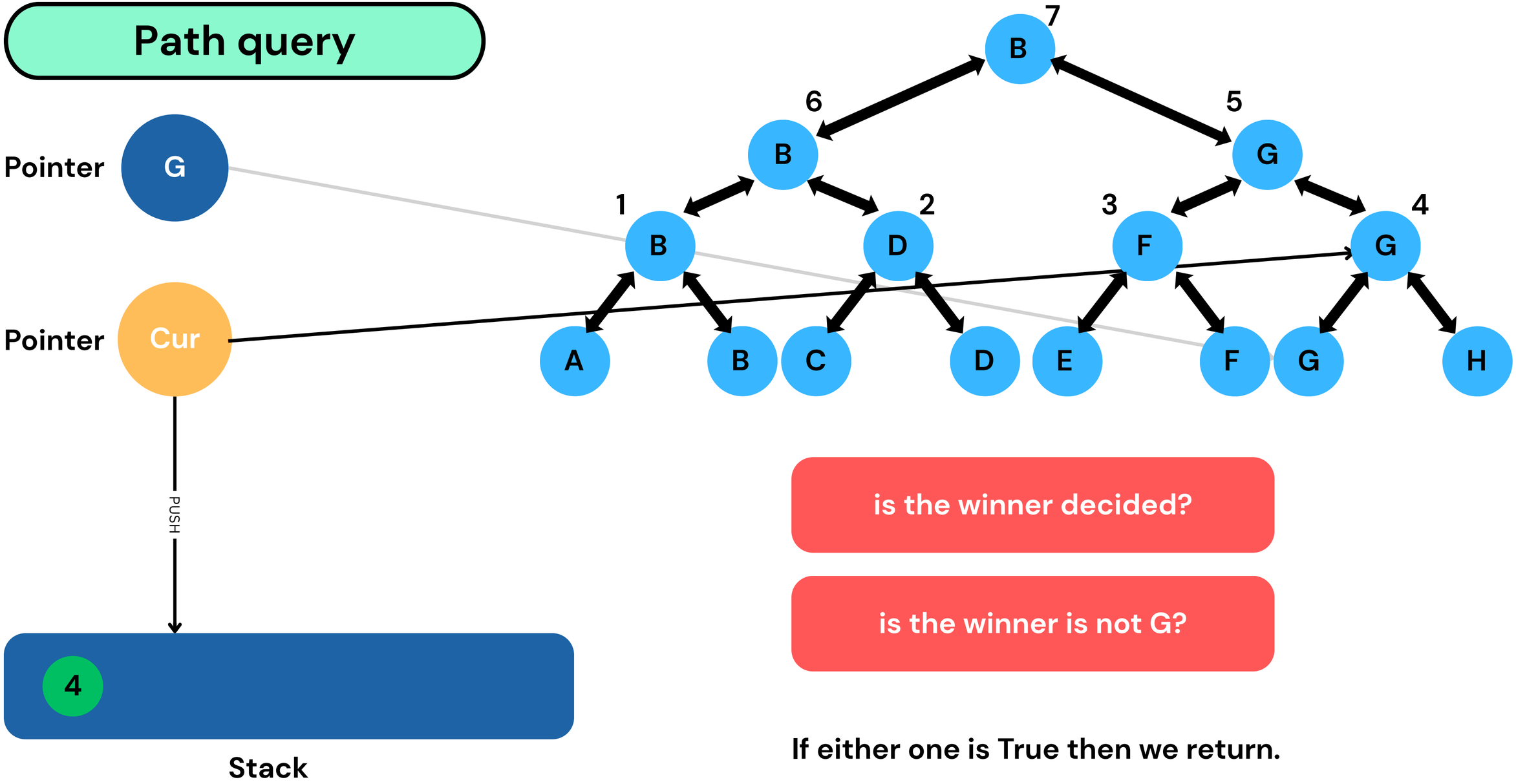
node , D

```
Node* findLeaf(Node* node, const string& name) {
    if (!node) return nullptr;
    if (node->isLeaf) return node->name == name ? node : nullptr;
    Node* L = findLeaf(node->left, name);
    return L ? L : findLeaf(node->right, name);
}
```




Path query
G
str Player
Find Node
G
Pointer
7
B
6
B
5
G
1
B
2
D
3
F
4
G
A
B
C
D
E
F
G
H

Path query

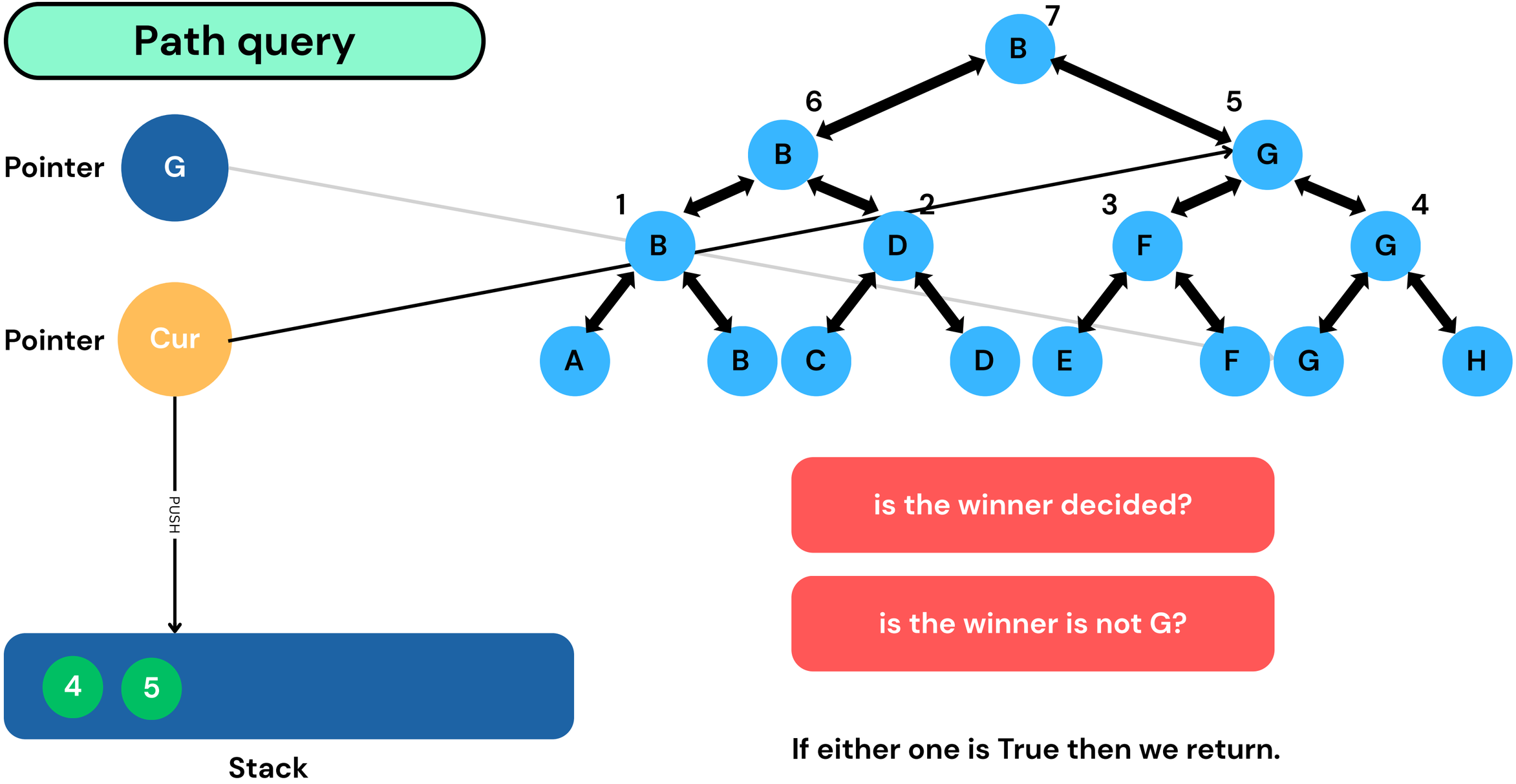


is the winner decided?

is the winner is not G?

If either one is True then we return.

Path query

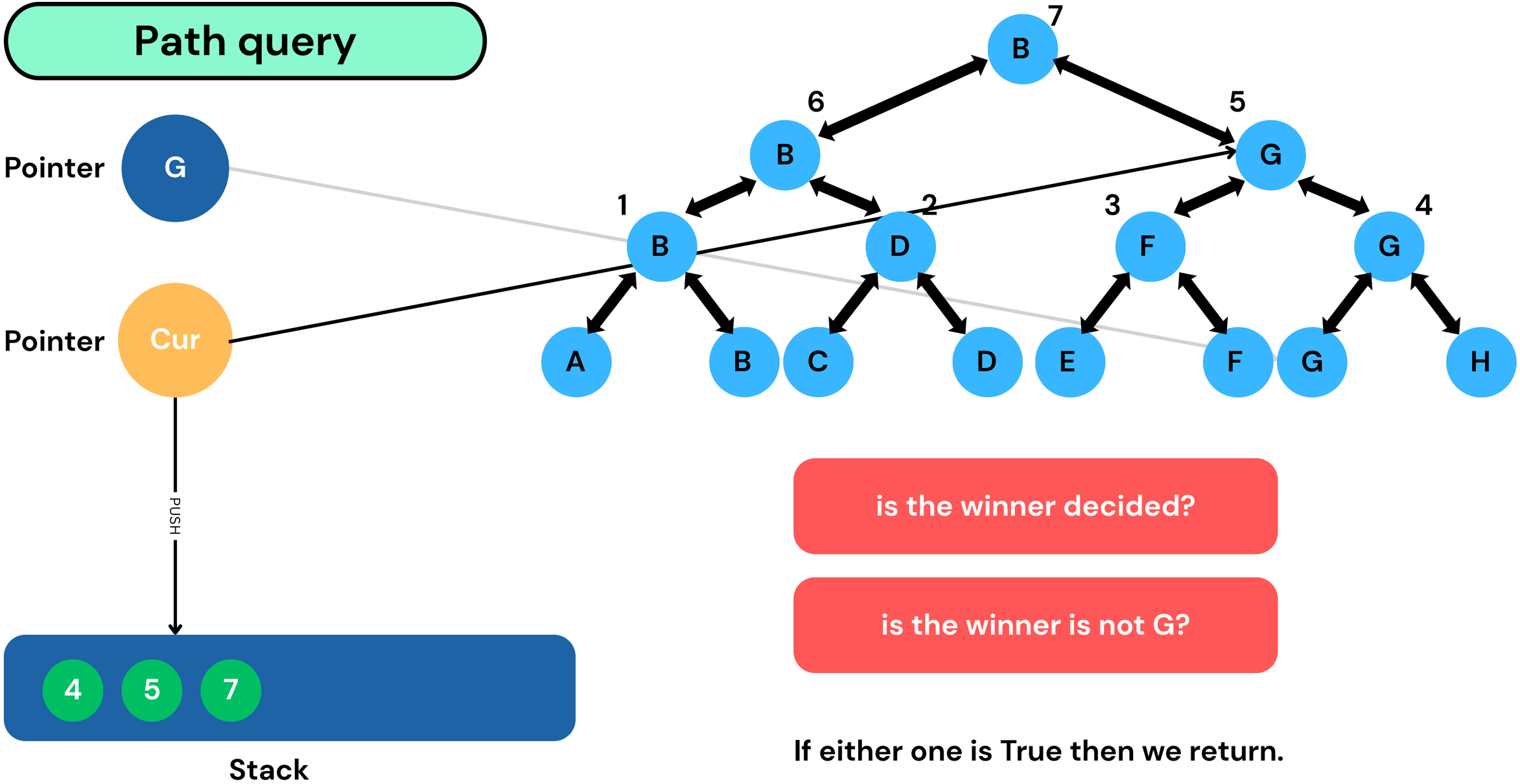


is the winner decided?

is the winner is not G?

If either one is True then we return.

Path query



is the winner decided?

is the winner is not G?

If either one is True then we return.

Codes

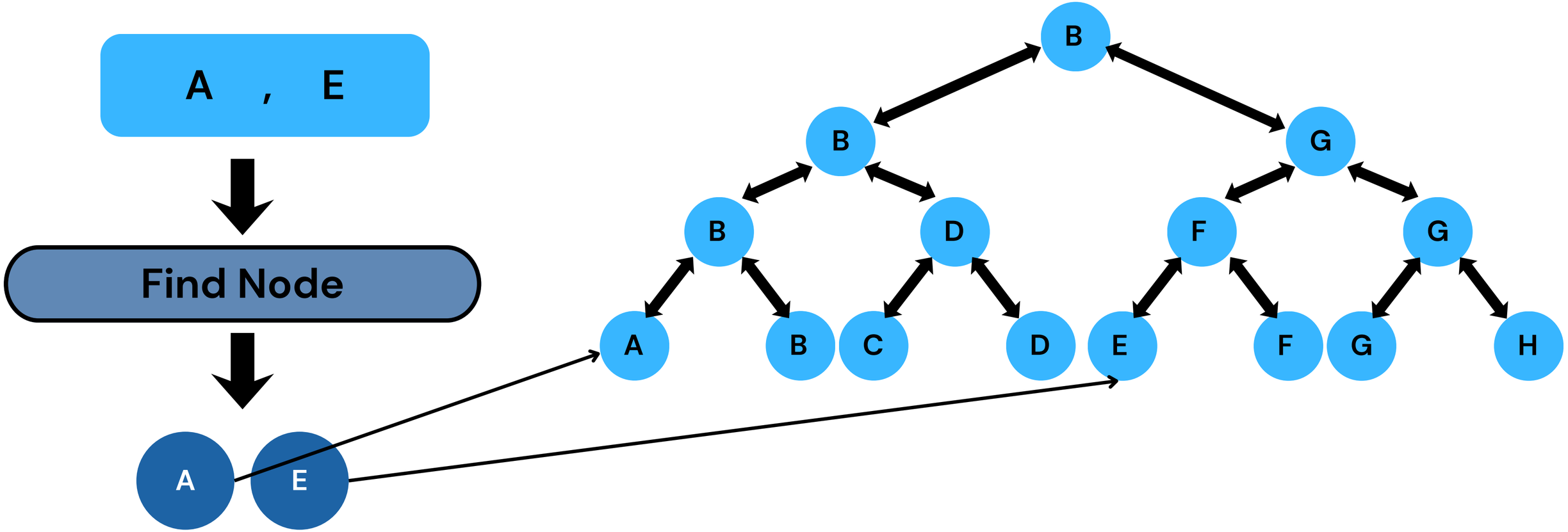
3 Path Query

```
vector<int> pathToFinal(const string& player) {
    vector<int> path;
    Node* leaf = findLeaf(root, player);
    if (!leaf) return path;
    Node* cur = leaf->parent;
    while (cur) {
        path.push_back(cur->matchId);
        if (cur->winner != "?" && cur->winner != player) break;
        cur = cur->parent;
    }
    return path;
}
```

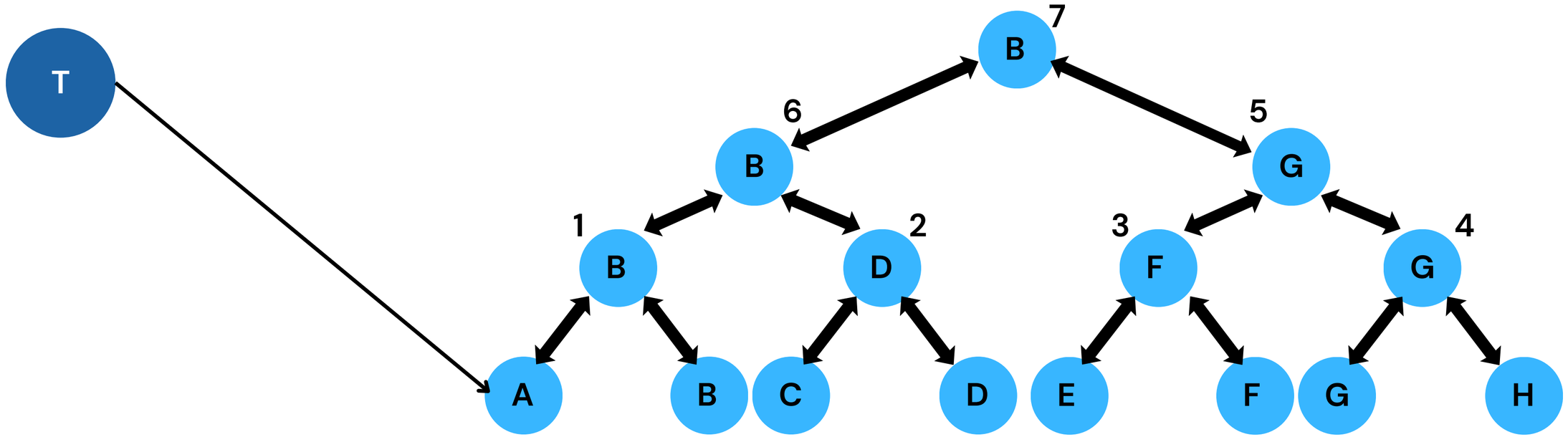
Meeting query (LCA)

A , E

Find Node

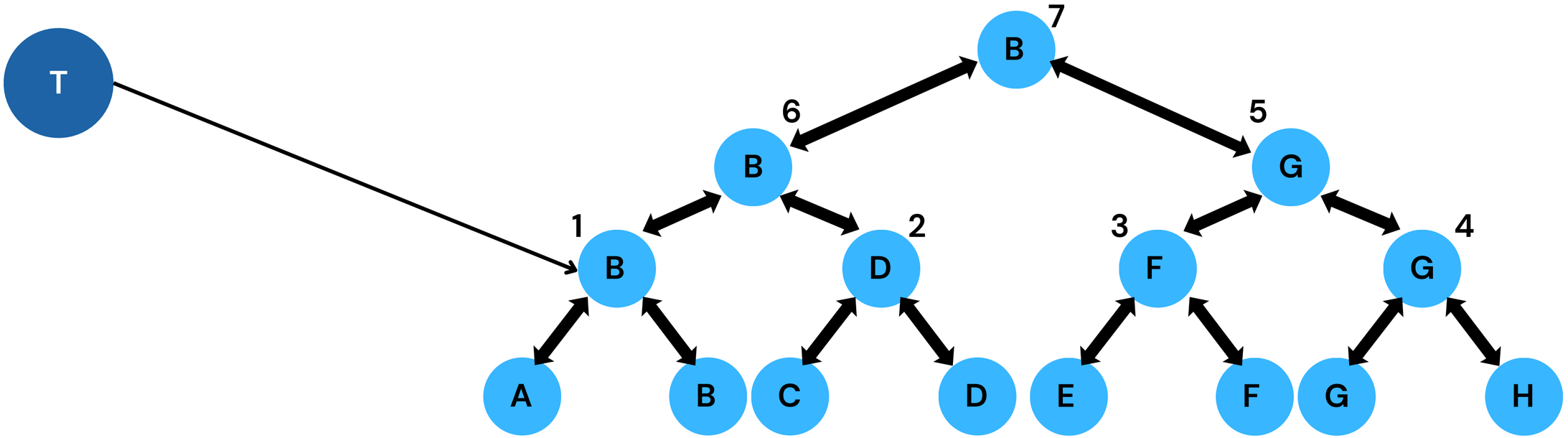


Meeting query (LCA)



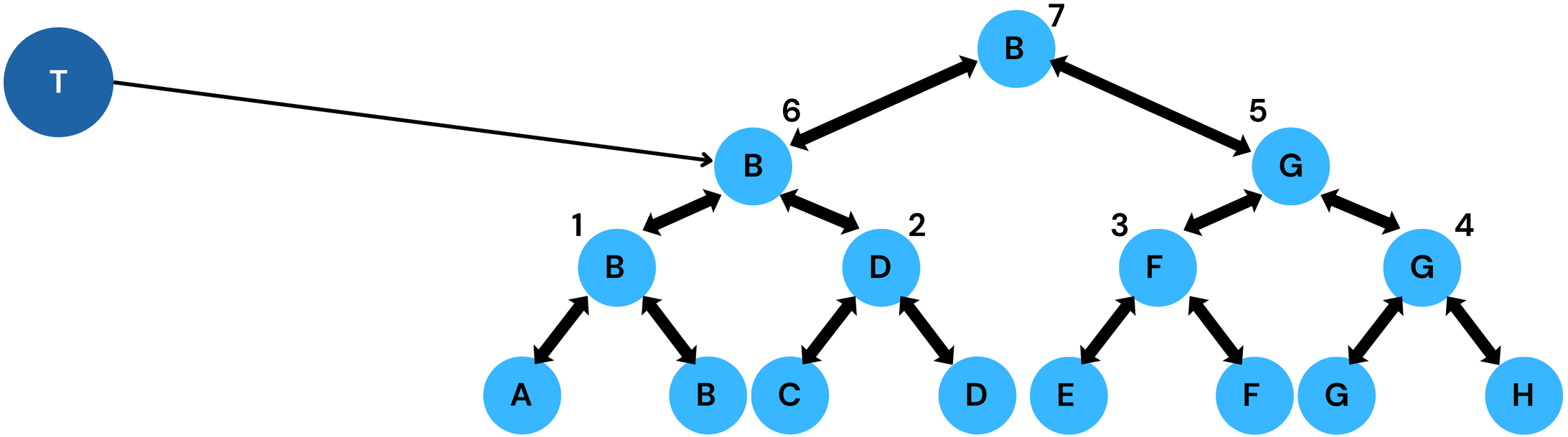
list

Meeting query (LCA)

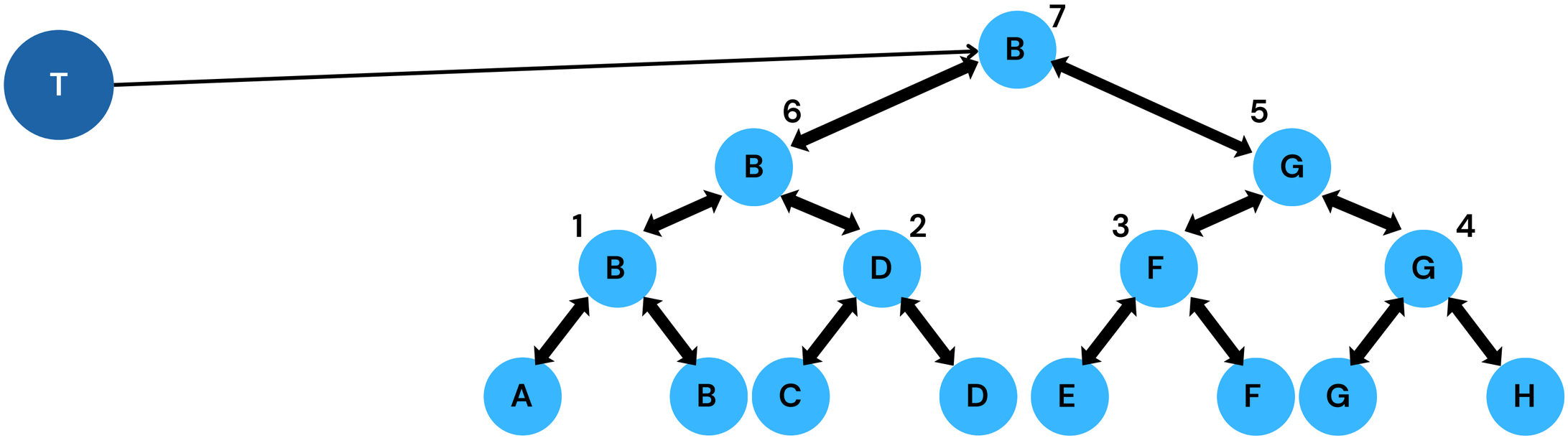


list

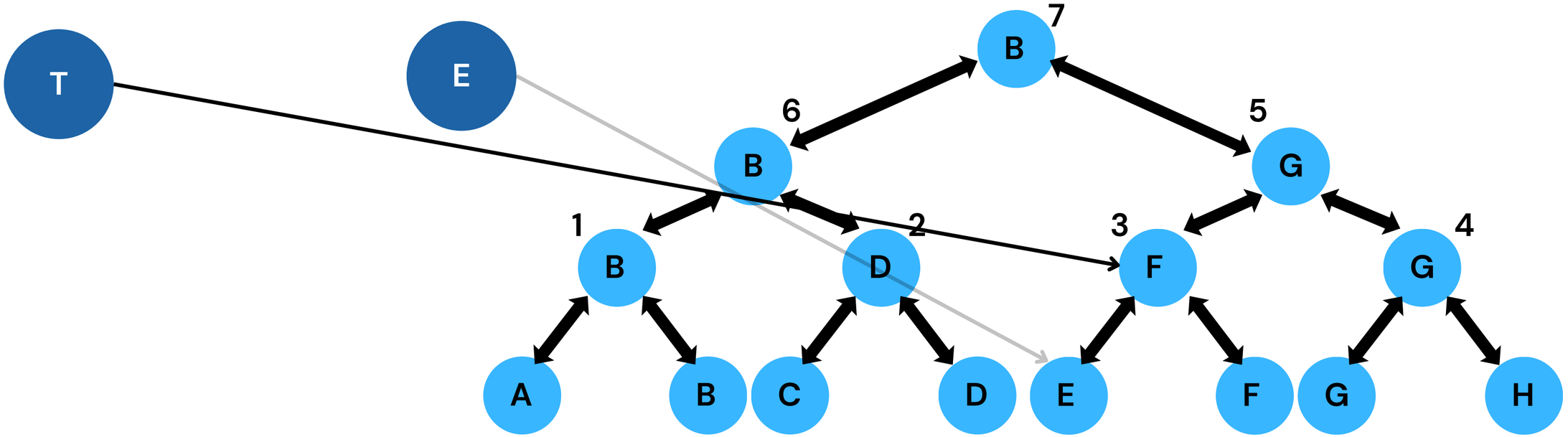
Meeting query (LCA)



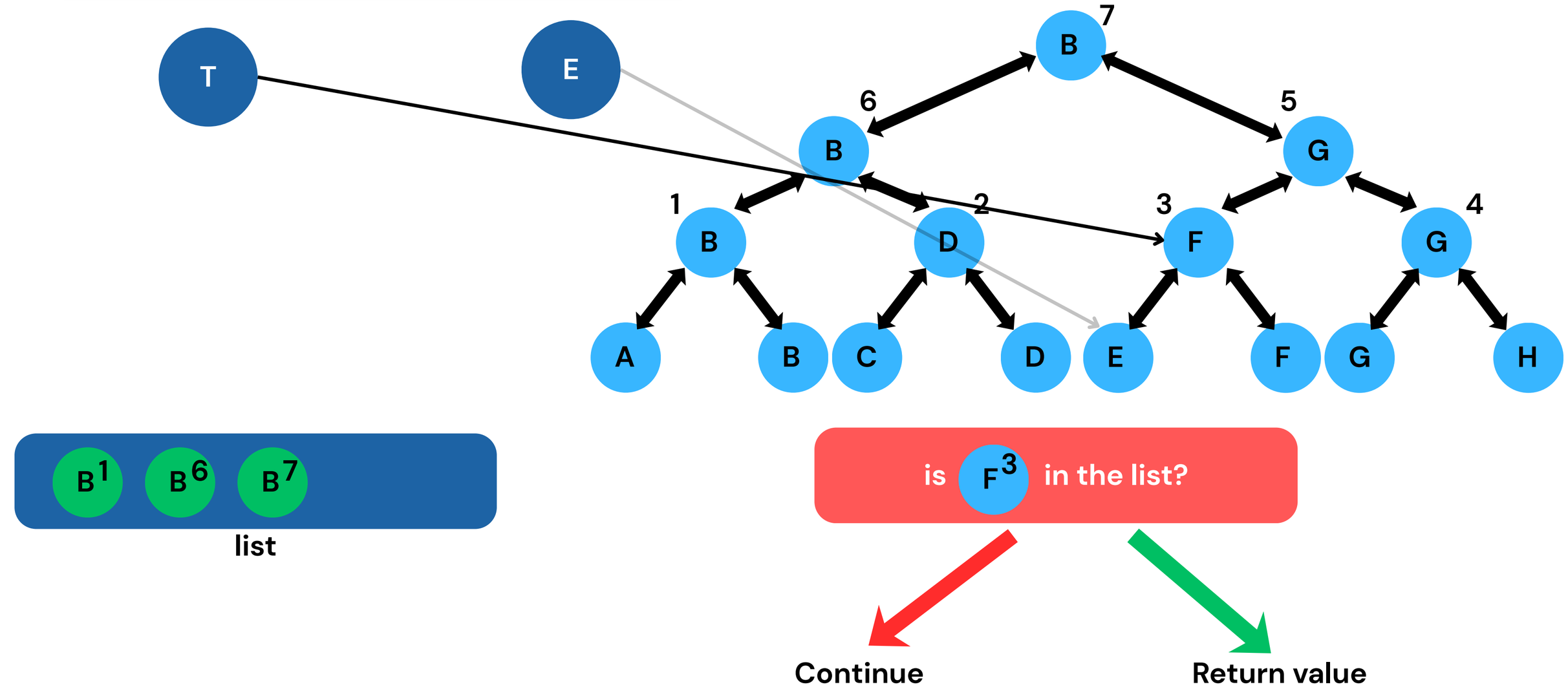
Meeting query (LCA)



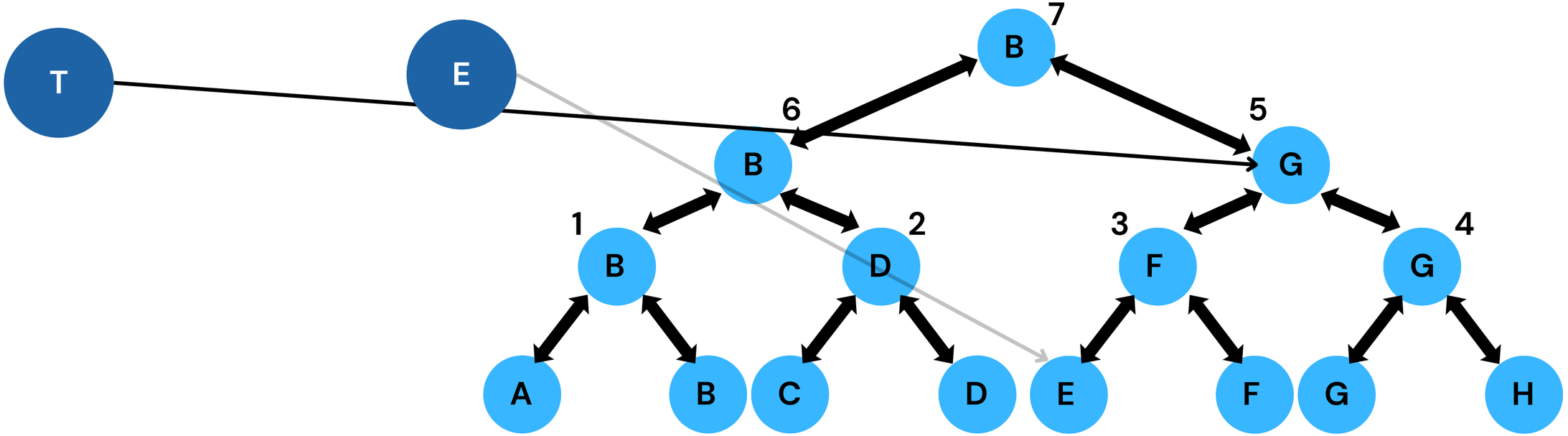
Meeting query (LCA)

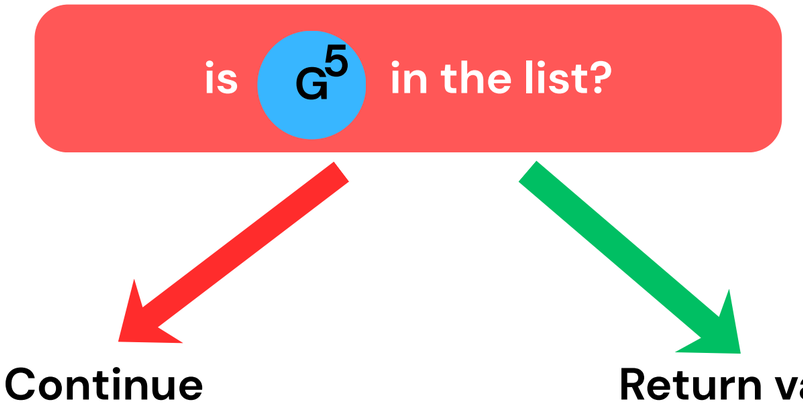


Meeting query (LCA)

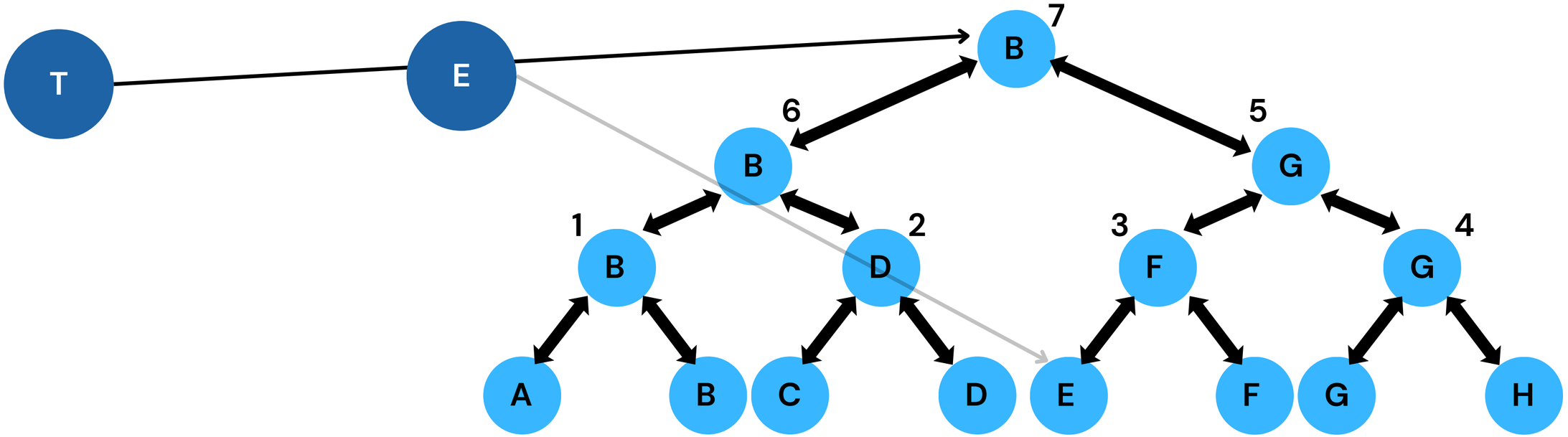


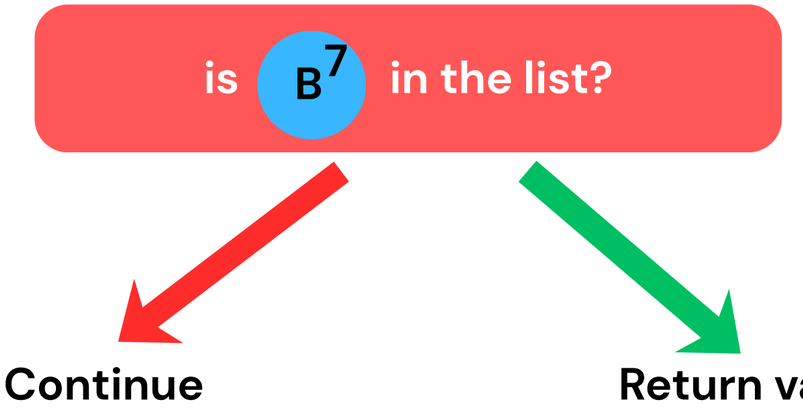
Meeting query (LCA)





Meeting query (LCA)





Codes

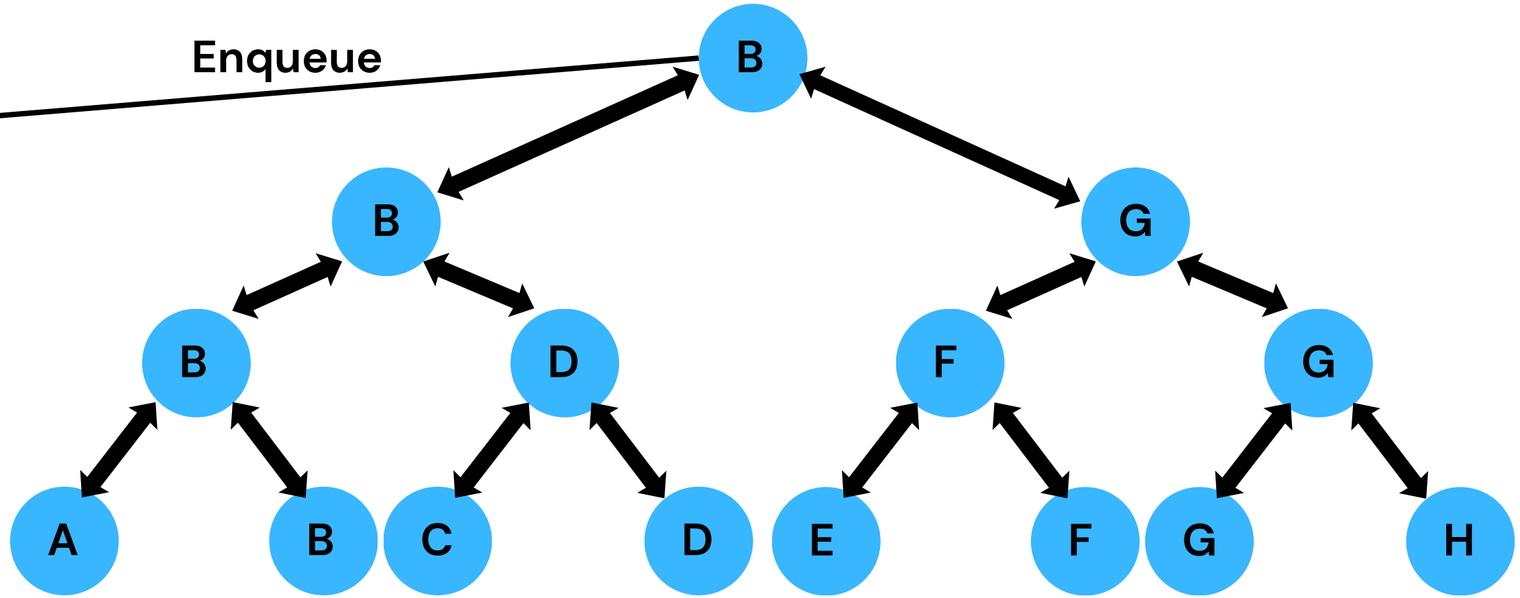
4 Meeting Query

```cpp
pair<int,int> wouldMeet(const string& p1, const string& p2) {
    Node* a = findLeaf(root, p1);
    Node* b = findLeaf(root, p2);
    if (!a || !b) return {0,0};

    vector<Node*> chainA;
    for (Node* t = a; t; t = t->parent) chainA.push_back(t);

    for (Node* t = b; t; t = t->parent) {
        for (Node* x : chainA) {
            if (x == t && !t->isLeaf) return {t->matchId, t->round};
        }
    }
    return {0,0};
}
```

Print bracket (level-order)

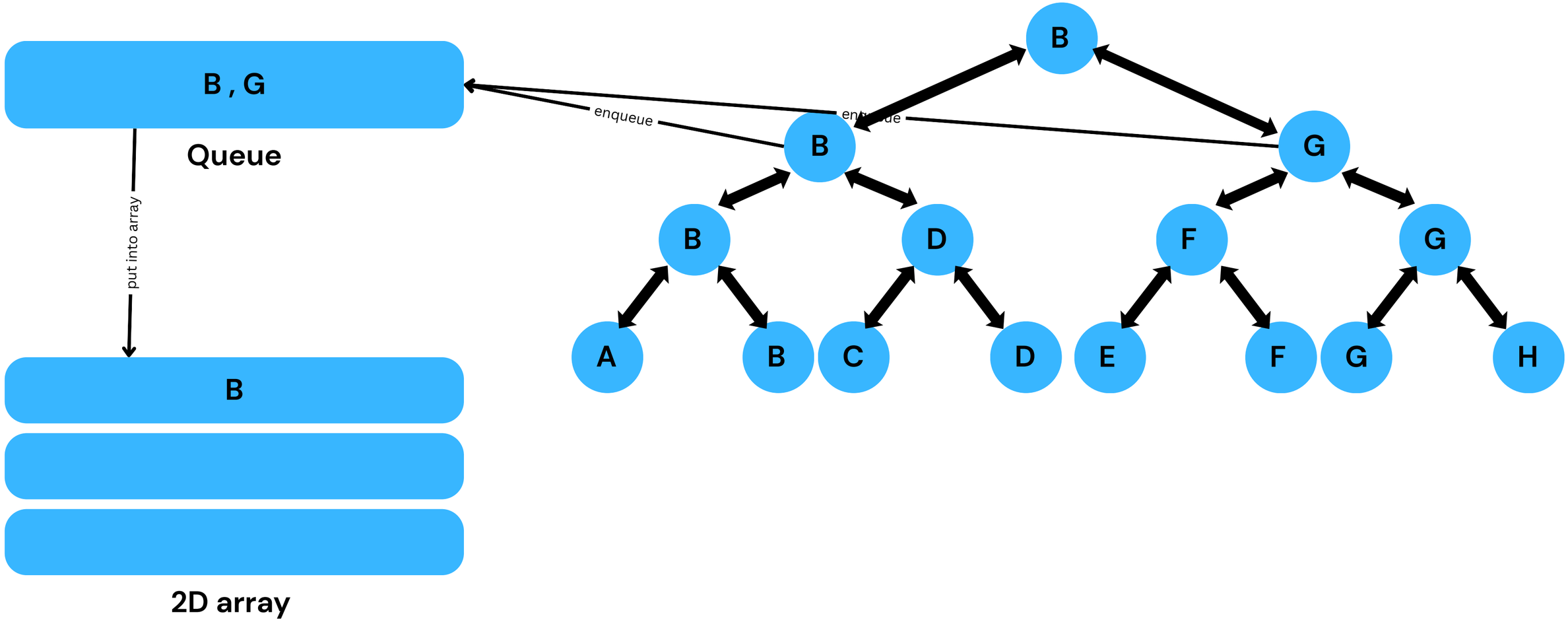
Queue

Enqueue

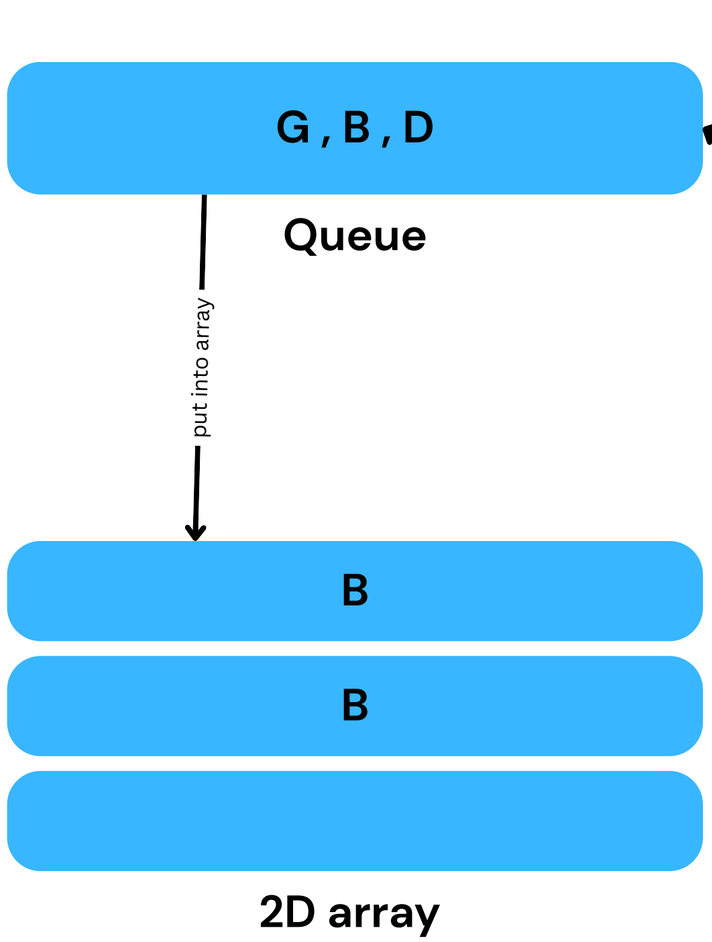


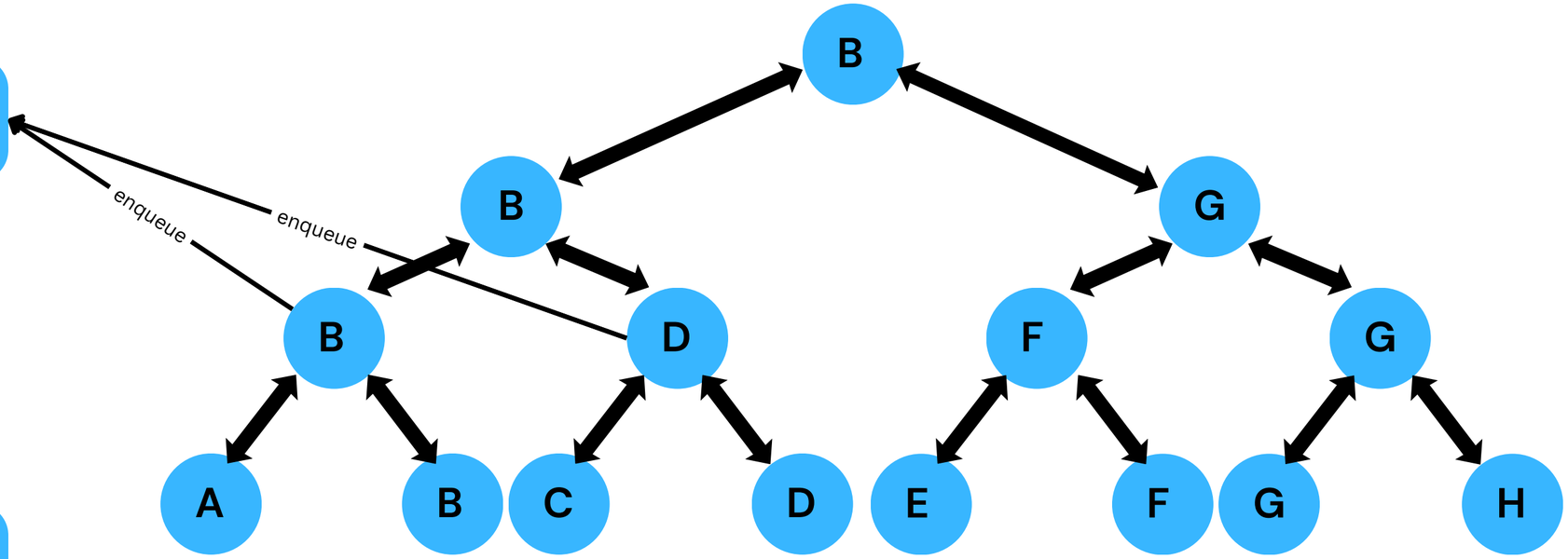
2D array

Print bracket (level-order)

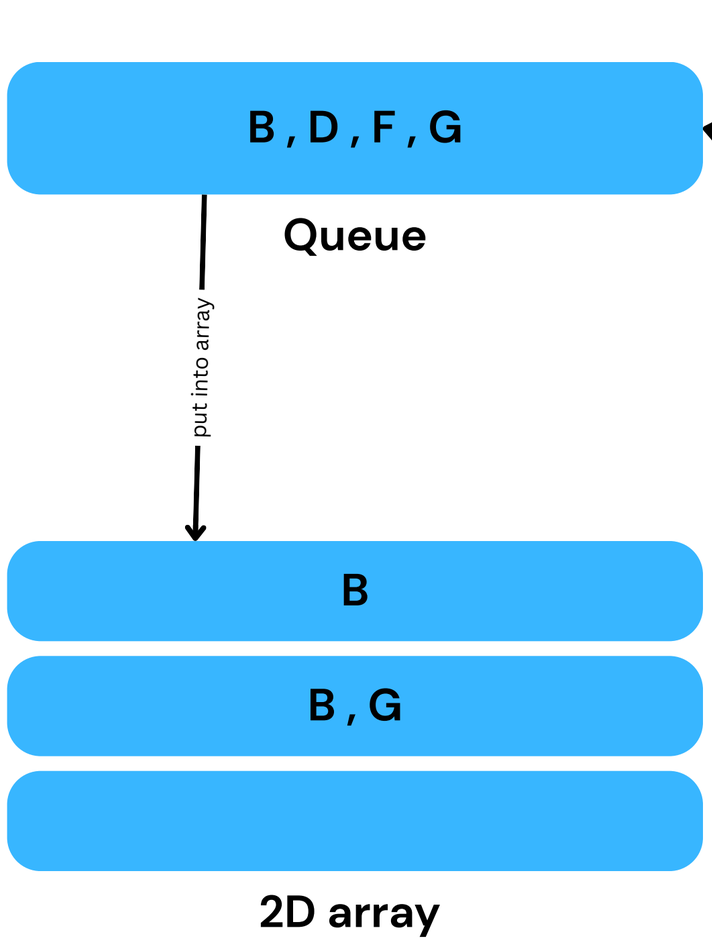


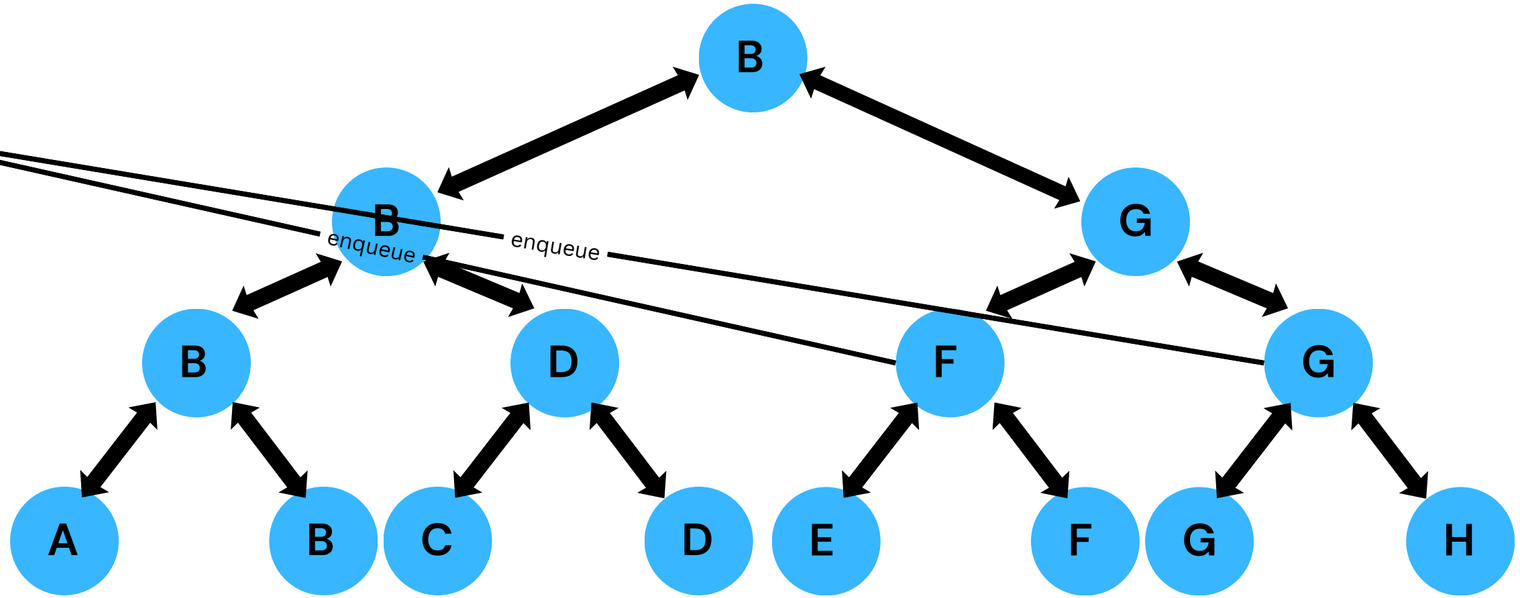
Print bracket (level-order)



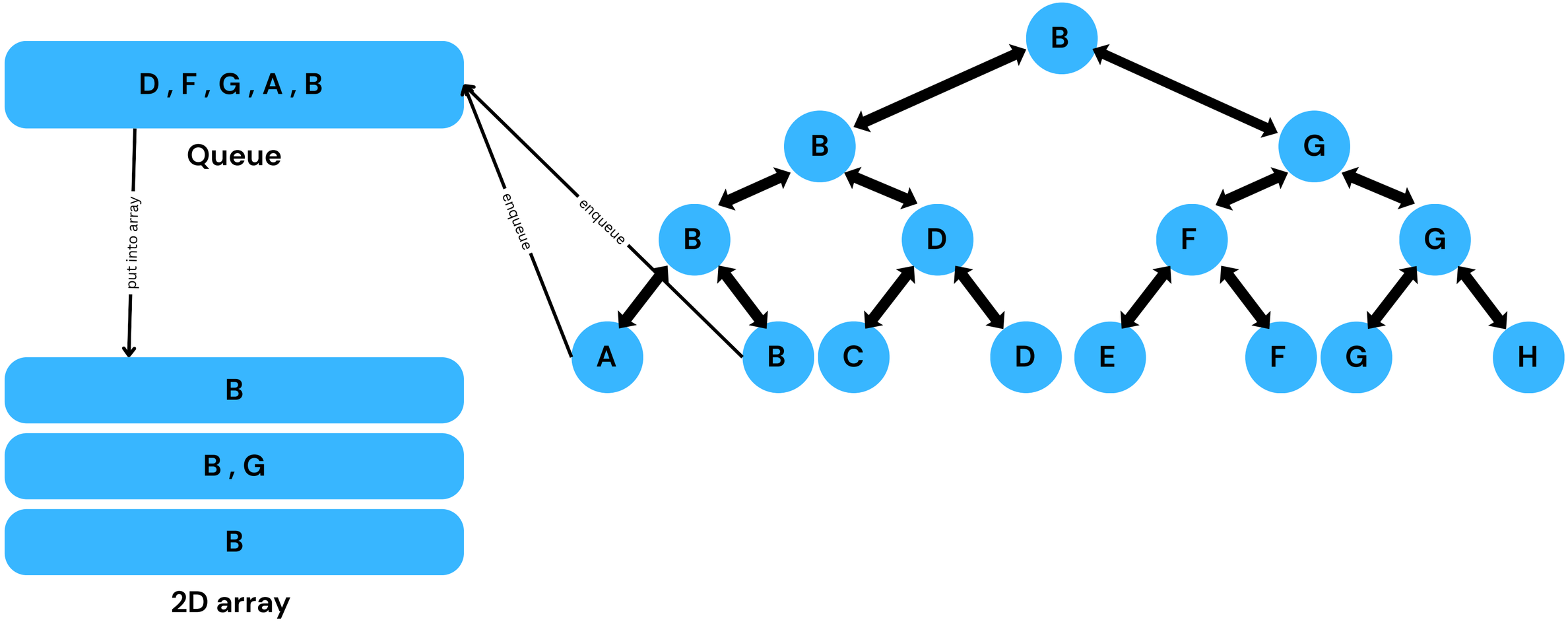


Print bracket (level-order)

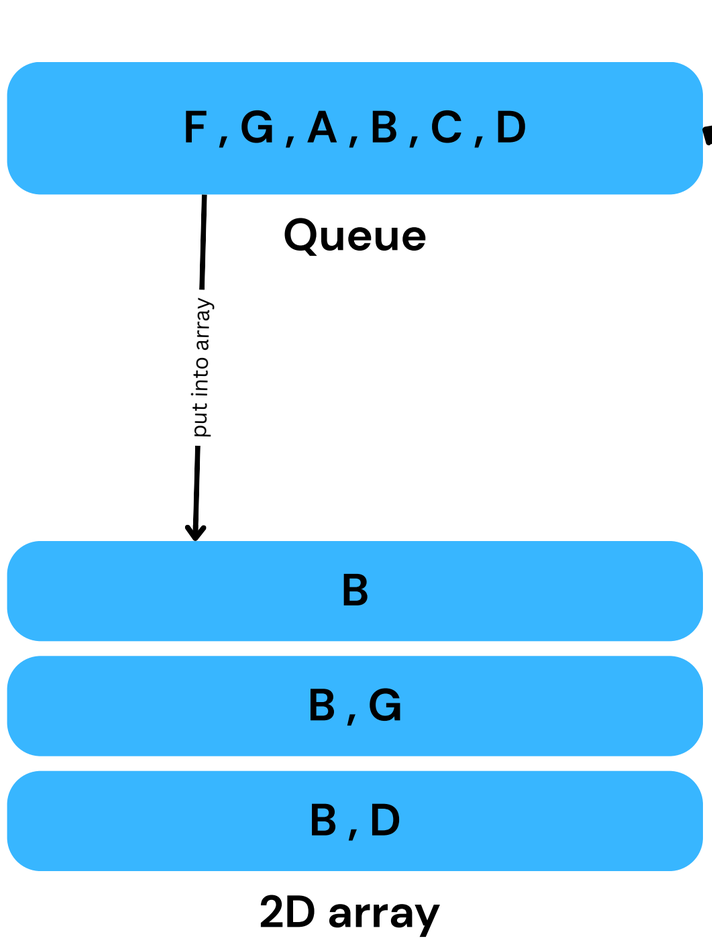


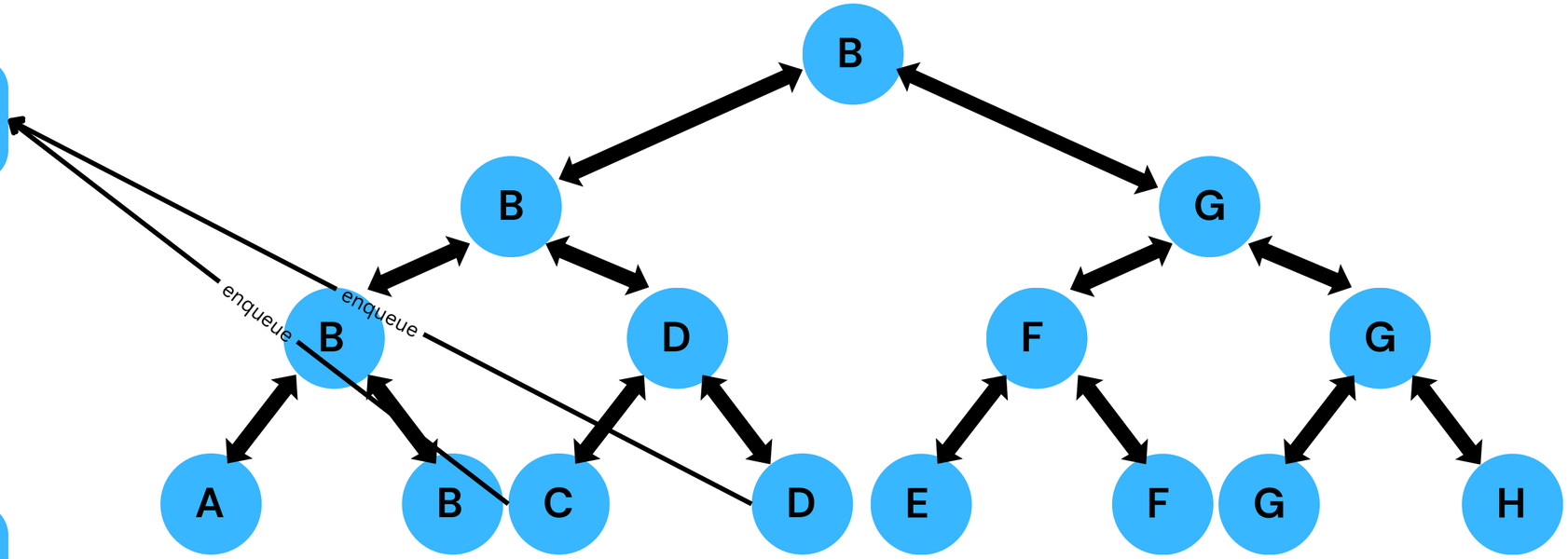


Print bracket (level-order)

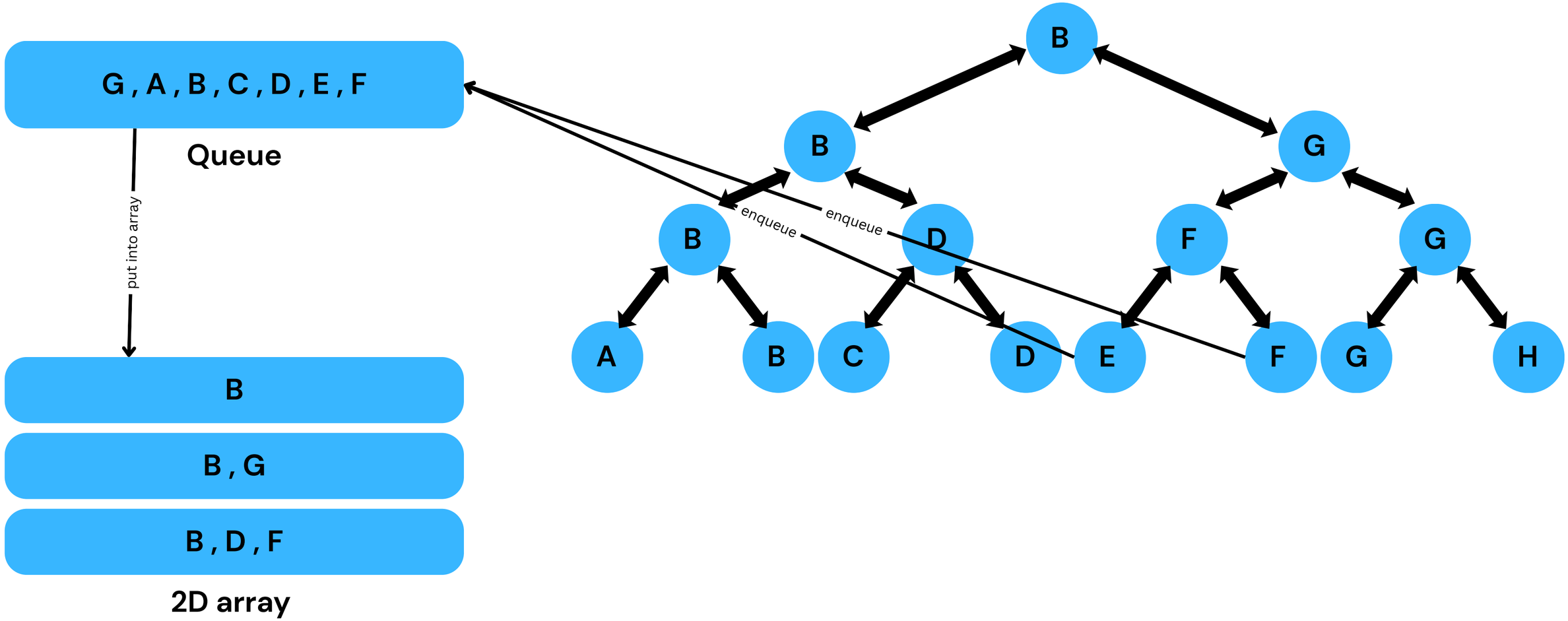


Print bracket (level-order)

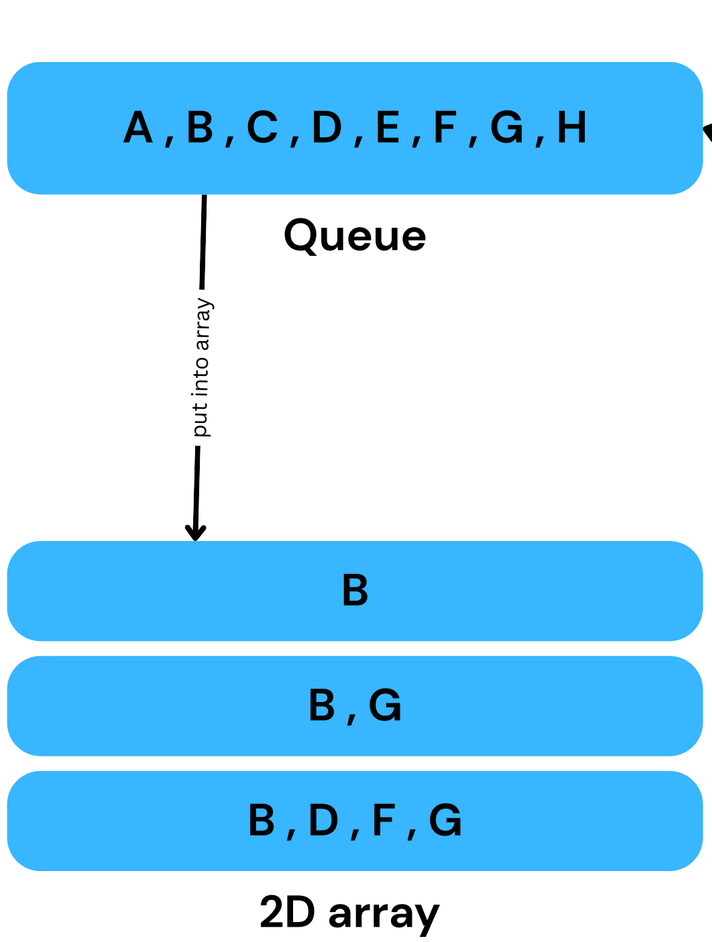


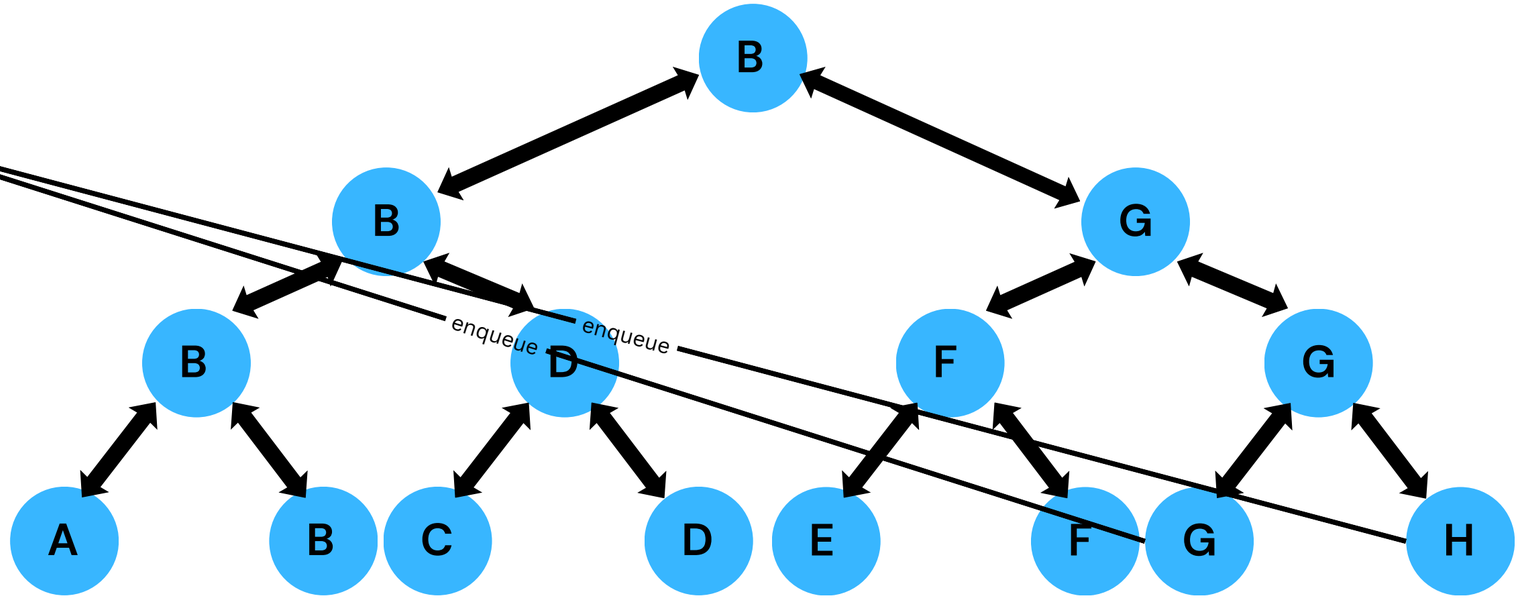


Print bracket (level-order)



Print bracket (level-order)





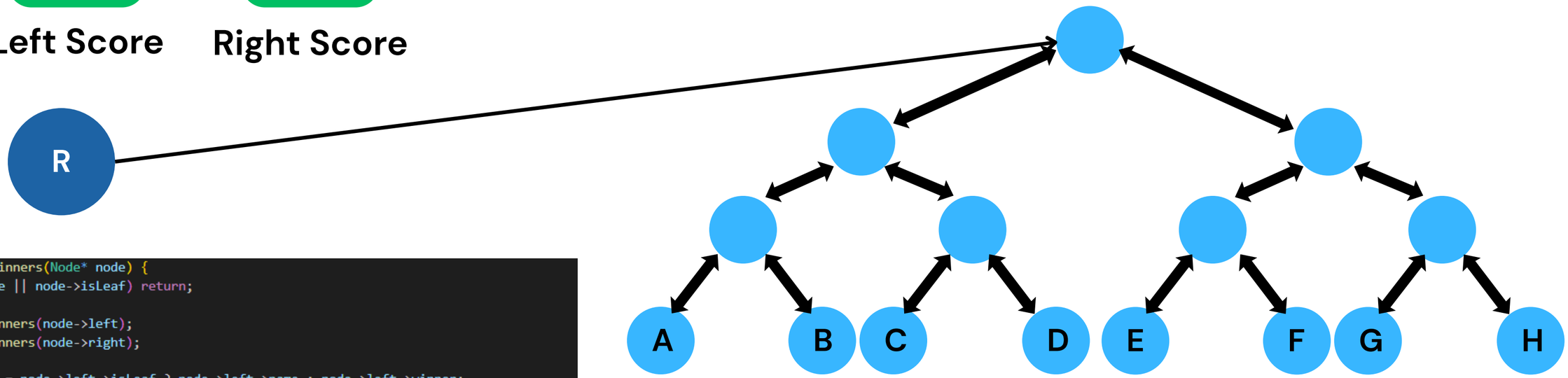
Codes

Print Bracket

```
void printBracket() {
    vector<vector<Node*>> rounds(totalRounds+1);
    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        Node* n = q.front(); q.pop();
        if (!n->isLeaf) {
            rounds[n->round].push_back(n);
            q.push(n->left);
            q.push(n->right);
        }
    }

    for (int r = 1; r <= totalRounds; r++) {
        cout << "\nRound " << r << ":\n";
        for (Node* m : rounds[r]) {
            string L = m->left->isLeaf ? m->left->name : m->left->winner;
            string R = m->right->isLeaf ? m->right->name : m->right->winner;
            cout << "  matchId = " << m->matchId
                 << " | " << L << " vs " << R
                 << " -> winner = " << m->winner << "\n";
        }
    }
}
```

Decide winner

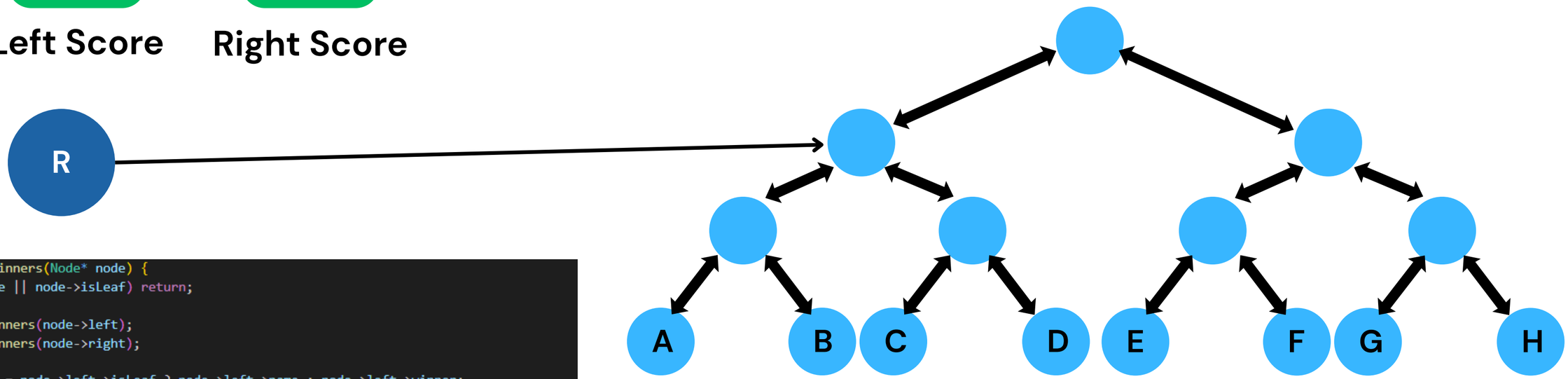
Left Score Right Score




```cpp
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

Decide winner

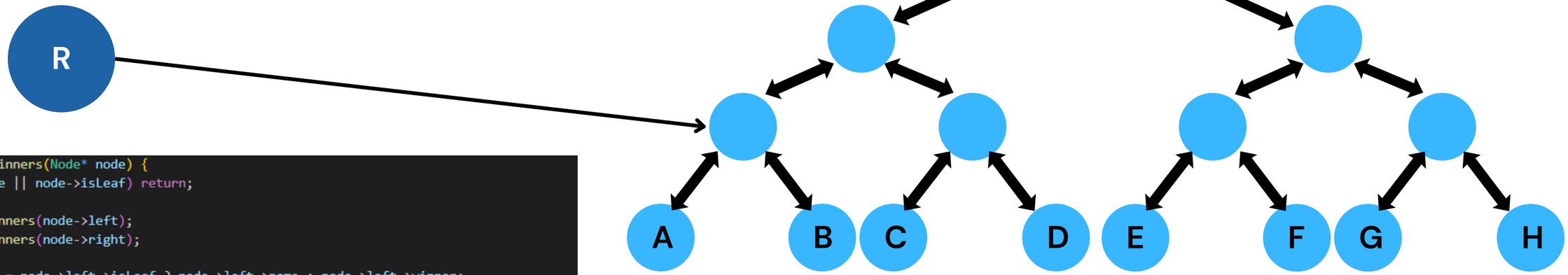
Left Score Right Score




```cpp
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

Decide winner

Left Score Right Score




```cpp
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

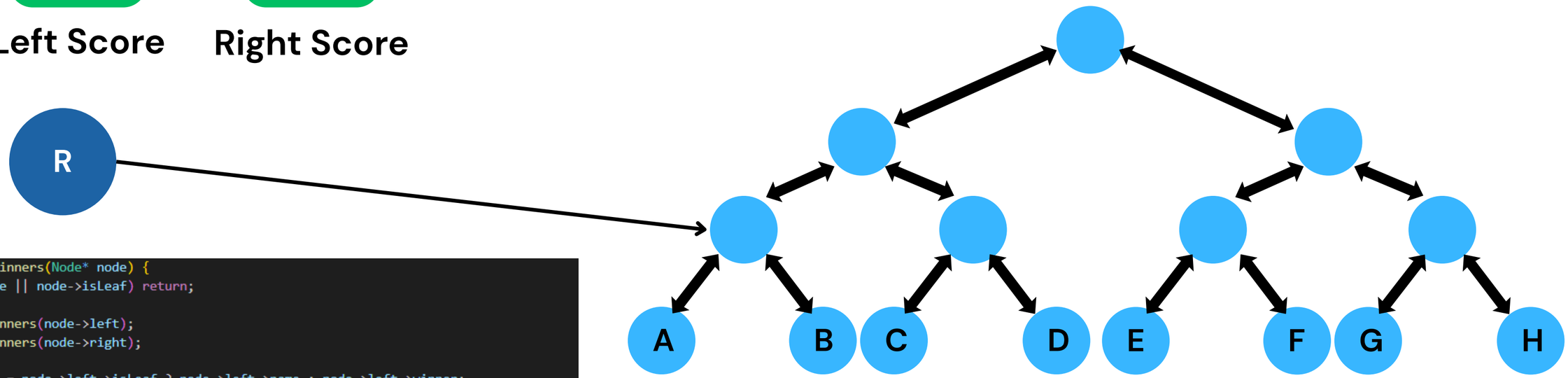
Decide winner

45

Left Score

27

Right Score




```cpp
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

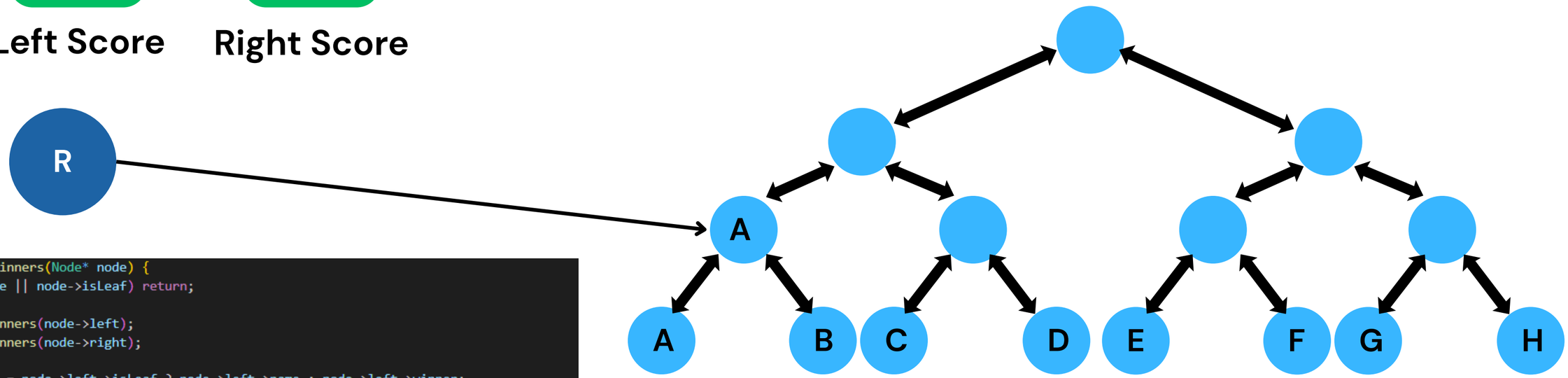
Decide winner

45

Left Score

27

Right Score




```
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

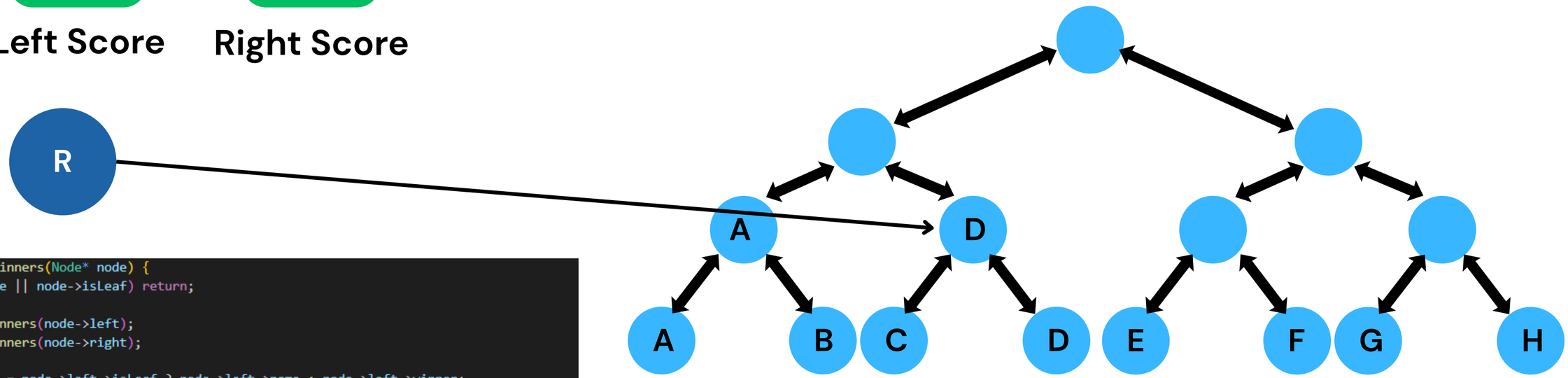
Decide winner

5

Left Score

20

Right Score




```
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

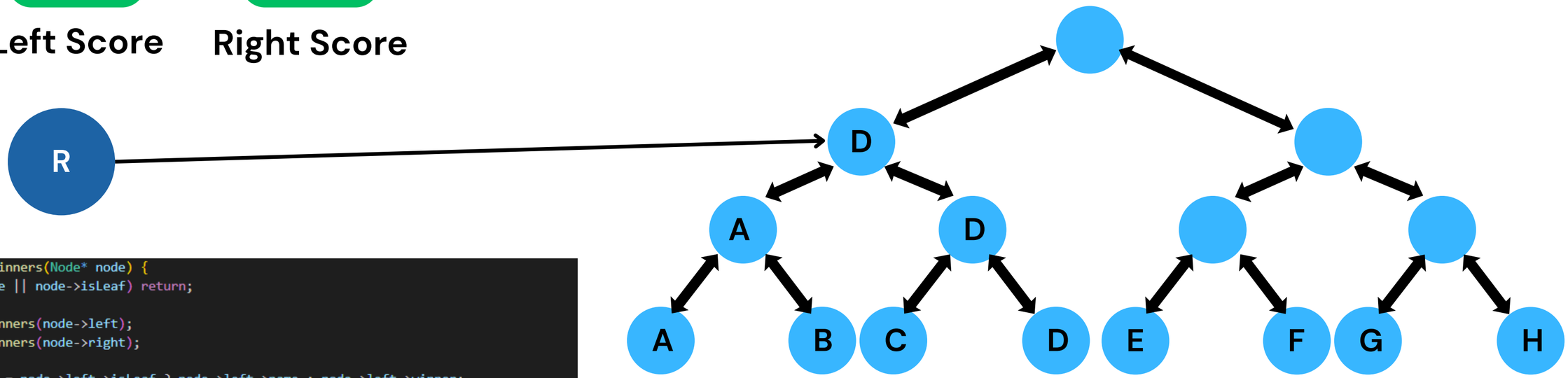
Decide winner

6

Left Score

87

Right Score




```cpp
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

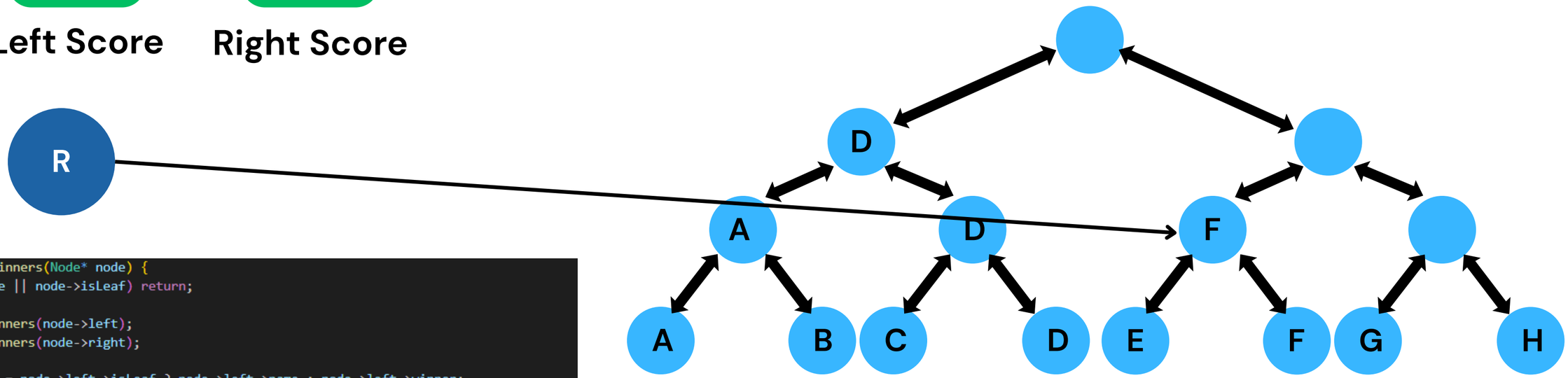
Decide winner

2

Left Score

47

Right Score




```
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

Decide winner

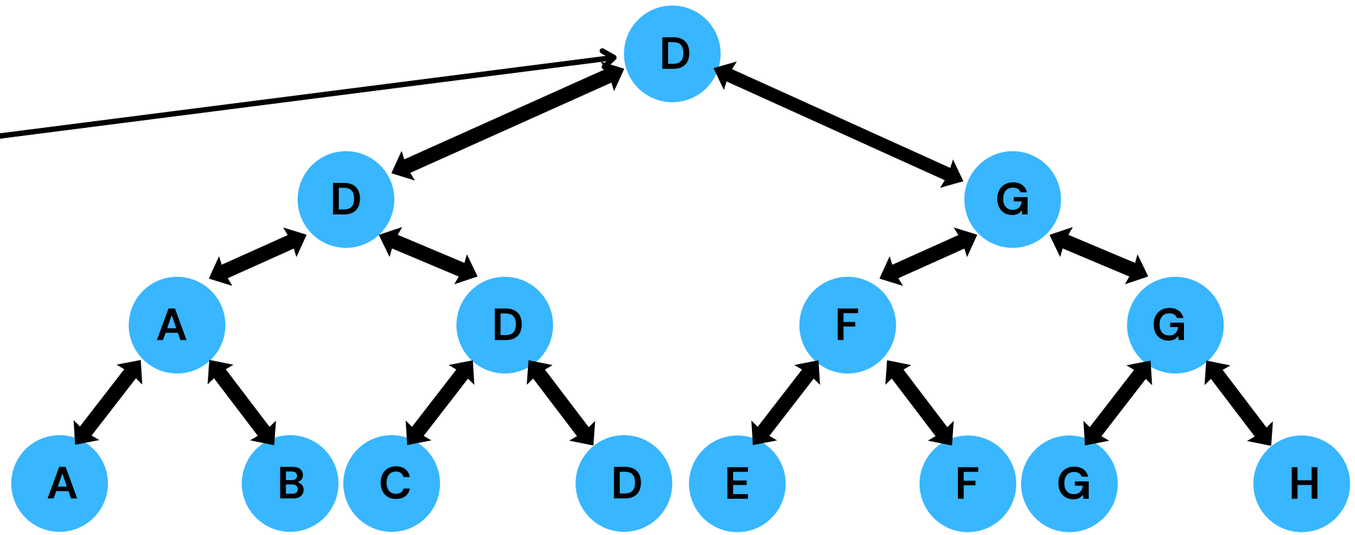
2

Left Score

47

Right Score

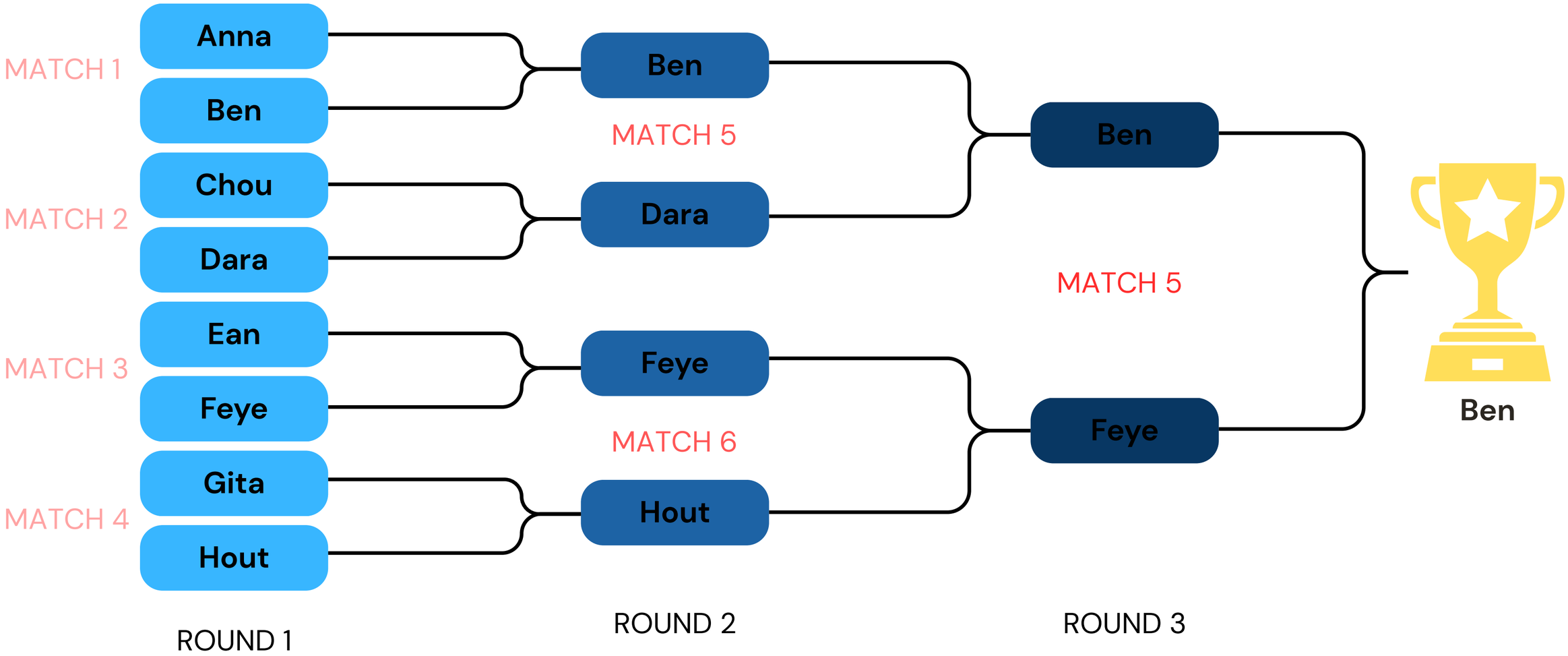
R




```
void decideWinners(Node* node) {
    if (!node || node->isLeaf) return;

    decideWinners(node->left);
    decideWinners(node->right);

    string L = node->left->isLeaf ? node->left->name : node->left->winner;
    string R = node->right->isLeaf ? node->right->name : node->right->winner;

    if (L == "BYE") recordResult(node->matchId, R);
    else if (R == "BYE") recordResult(node->matchId, L);
    else {
        int scoreL = rand() % 100;
        int scoreR = rand() % 100;
        string winner = (scoreL >= scoreR ? L : R);
        recordResult(node->matchId, winner);
        cout << "Match " << node->matchId << " (Round " << node->round << "): "
             << L << " [" << scoreL << "] vs " << R << " [" << scoreR << "] -> Winner: "
             << winner << "\n";
    }
}
```

Examples

Scenario Demonstration



```cpp
#include "implementation/tournament.cpp"

int main() {
    srand(time(0));
    vector<string> players = {"Anna","Ben","Chou","Dara","Ean","Faye","Gita","Hout"};

    buildBracket(players);
    cout << "Bracket built.\n";

    cout << "Match Results:\n";
    decideWinners(root);

    cout << "\nFinal Bracket:\n";
    printBracket();

    auto meet = wouldMeet("Anna", "Dara");
    cout << "\nwouldMeet(Anna, Dara) => matchId = " << meet.first
         << ", round = " << meet.second << "\n";

    auto path = pathToFinal("Faye");
    cout << "pathToFinal(Faye): ";
    for (int id : path) cout << id << " ";
    cout << "\n";

    cout << "Champion: " << root->winner << "\n";
}
```

```
[Running] cd "c:\Users\U-ser\Documents\CADT-2nd-year\c++\jak_ball\" && g++ main.cpp -o main && "c:\Users\U-ser\Documents\CADT-
Bracket built.
Match Results:
Match 1 (Round 1): Anna [16] vs Ben [38] -> Winner: Ben
Match 2 (Round 1): Chou [6] vs Dara [70] -> Winner: Dara
Match 5 (Round 2): Ben [97] vs Dara [32] -> Winner: Ben
Match 3 (Round 1): Ean [10] vs Faye [51] -> Winner: Faye
Match 4 (Round 1): Gita [51] vs Hout [69] -> Winner: Hout
Match 6 (Round 2): Faye [73] vs Hout [65] -> Winner: Faye
Match 7 (Round 3): Ben [61] vs Faye [20] -> Winner: Ben

Final Bracket:

Round 1:
  matchId = 1 | Anna vs Ben -> winner = Ben
  matchId = 2 | Chou vs Dara -> winner = Dara
  matchId = 3 | Ean vs Faye -> winner = Faye
  matchId = 4 | Gita vs Hout -> winner = Hout

Round 2:
  matchId = 5 | Ben vs Dara -> winner = Ben
  matchId = 6 | Faye vs Hout -> winner = Faye

Round 3:
  matchId = 7 | Ben vs Faye -> winner = Ben

wouldMeet(Anna, Dara) => matchId = 5, round = 2
pathToFinal(Faye): 3 6 7
Champion: Ben
```

CADT បណ្ឌិត្យសភាបច្ចេកវិទ្យាឌីជីថលកម្ពុជា
Cambodia Academy of Digital Technology

Thank you!

November 25, 2025