**BÀI CÁO CÁO CUỐI KÌ MÔN CTDL&GT**

**LỚP: IT003.N27.1**

**NHÓM: IV**

**MÃ ĐỀ: IV (10 +11)**

**BẢNG PHÂN CÔNG**

|  |  |  |
| --- | --- | --- |
| Phân Công Thành Viên | | |
| STT | **Họ Tên** | **Nhiệm Vụ** |
| 1 | **Lại Quan Thiên**  **22521385** | * Làm đề 10 câu 4, đề 11 câu 1, đề 11 câu 4. * Soạn slide, phân công nhiện vụ cho nhóm. * Lên ý tưởng, duyệt, tổng hợp bài. |
| 2 | **Lê Minh Quân**  **22521181** | * Làm đề 10 câu 1, đề 10 câu 2, đề 11 câu 2. * Lên ý tưởng * Kiểm tra bài, chỉnh sửa lỗi. |
| 3 | **Trần Thế Hữu Phúc**  **22521143** | * Làm đề 10 câu 3, đề 11 câu 3. * Lên ý tưởng * Tổng hợp đề 10, 11, hình ảnh, kiểm tra bài. |

**ĐỀ 10**

**Câu 1 (2 điểm):**

a. Anh/ Chị hãy trình bày ý tưởng của thuật toán tìm kiếm *nhị phân (Binary Search)* để tìm kiếm một số trên một mảng đã có thứ tự.

b. Trình bày các bước (vẽ từng bước) theo thuật toán tìm kiếm nhị phân thực hiện tìm kiếm giá trị x = 8 trên mảng số nguyên có giá trị: 1 4 6 13 17 19 .

**GIẢI:**

**a.**

- Ý tưởng của thuật toán **Tìm kiếm Nhị phân (Binary Search)**: Tìm kiếm trong đoạn từ **[left, right]** của mảng, ở mỗi bước thuật toán tìm vị trị **mid** ở giữa đoạn **[left, right]**. Nếu phần tử cần tìm kiếm bằng phần tử ở vị trí **mid** thì kết luận là tìm thấy, nếu không ta có thể giảm 1 nửa đoạn tìm kiếm xuống và tiếp tục tìm kiếm ở bên trái hay bên phải của **mid**.

- Đây là một thuật toán cực kì hiệu quả và quan trọng, khi học lập trình ta cần nắm rõ thuật toán này với điều kiện áp dụng là mảng đã được sắp xếp.

- Độ phức tạp của thuật toán này là O(N.log(N)) với O(N.long(N)) là độ phức tạp khi chúng ta sắp xếp lại mảng và O(log(N)) là độ phức tạp của việc tìm kiếm nhị phân.

**b.** A screenshot of a computer

Description automatically generated

**Câu 2 (2.5 điểm):**

a**.** Anh/ chị hãy viết hàm cài đăt thuật toán *chọn trực tiếp (Selection Sort)* để sắp xếp một mảng các số nguyên có N phần tử theo chiều giảm dần bằng ngôn ngữ C/C++, cho biết độ phức tạp của giải thuật.

b. Trình bày các bước (vẽ từng bước) áp dụng thuật toán ở câu 2.a để sắp xếp mảng số nguyên {2, 7, 10, 9, 5, 3, 8, 40} giảm dần.

**GIẢI:**

**a.**

|  |
| --- |
| #include <iostream>  **using** **namespace** std;  **void** selectionSort(**int** a[], **int** n)  {  **for**(**int** i = 0; i < n - 1; i++)  {  **int** max\_pos = i;  **for**(**int** j = i + 1; j < n; j++)  {  **if**(a[j] > a[max\_pos])  {  max\_pos = j;  }  }  swap(a[i], a[max\_pos]);  }  }  **int** main()  {  **int** a[8] = {2, 7, 10, 9, 5, 3, 8, 40};  cout << "Mang ban dau la: ";  **for**(**int** x : a)  {  cout << x << " ";  }  cout << endl;  cout << "Mang sau khi sap xep giam dan la: ";  selectionSort(a, 8);  **for**(**int** x : a)  {  cout << x << " ";  }  cout << endl;  cout << "Do phuc tap cua thuat toan nay la: " << "O(N^2)" << endl;  system("pause");  **return** 0;  } |

**b.** Minh hoạ trong hình dưới đây:

A screenshot of a computer

Description automatically generated with medium confidence

**Câu 3 (3 điểm):** Biết rằng khi duyệt cây nhị phân tìm kiếm T theo thứ tự Left - Right - Node (LRN) thì được dãy sau: 14, 23, 17, 32, 37, 35, 29, 42, 40, 53, 44, 38. Hãy thực hiện các yêu cầu sau:

a. Vẽ lại cây nhị phân tìm kiếm T.

b. Xóa khỏi cây T lần lượt các nút 37,40, 29, 38 (vẽ hình từng trường hợp) sao cho cây vẫn là cây nhị phân tìm kiếm sau khi xóa nút.

c. Viết hàm in ra tổng giá trị các nút ở mức k trên cây nhị phân tìm kiếm T (với quy ước mức ở nút gốc là 0).

**GIẢI:**

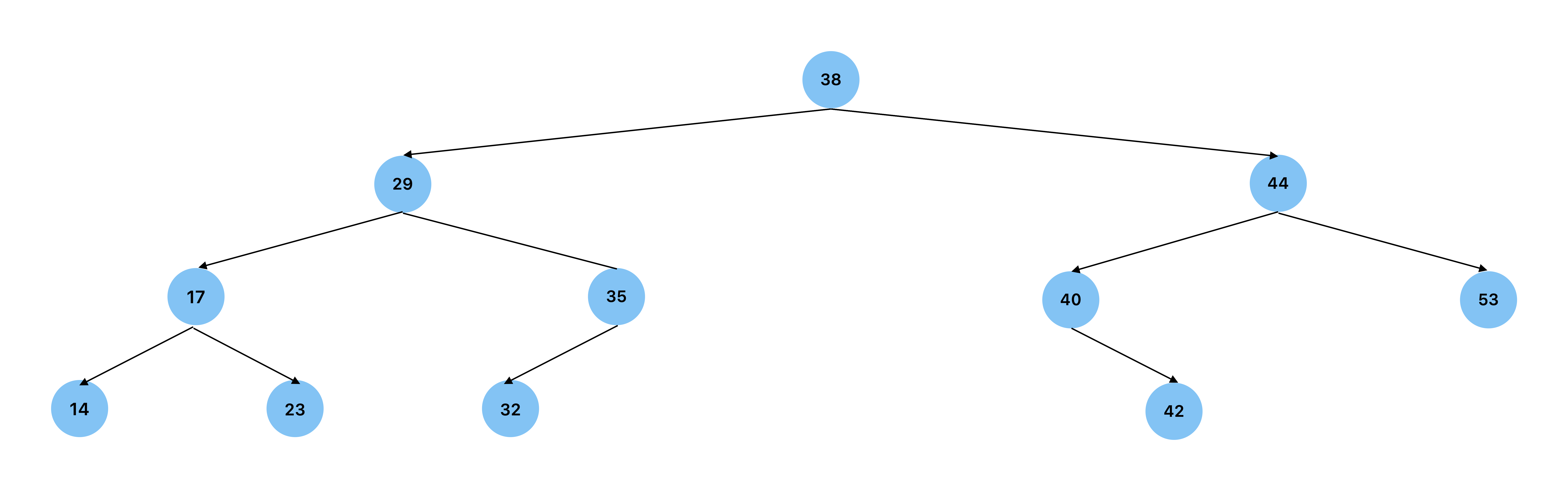
**a.** Xét dãy số được duyệt từ cây nhị phân tìm kiếm T theo thứ tự Left - Right - Node (LRN): 14, 23, 17, 32, 37, 35, 29, 42, 40, 53, 44, 38

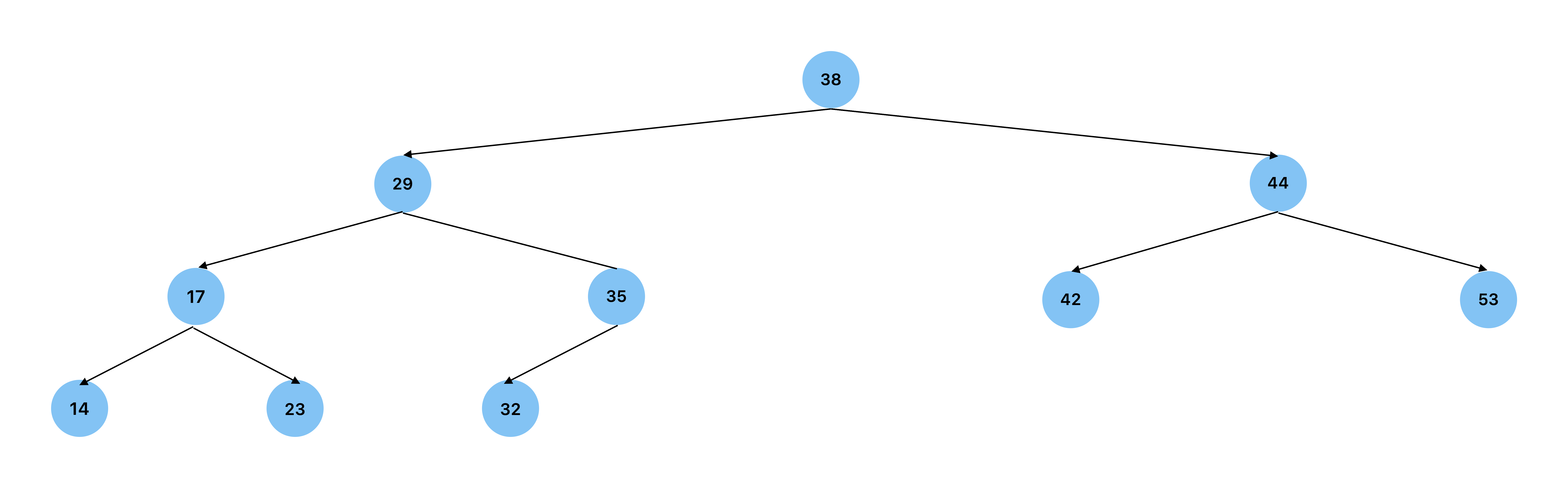
- Ta dễ thấy 38 là nút gốc với cây con trái gồm: 14, 23, 17, 32, 37, 35, 29; cây con phải gồm: 42, 40, 53, 44

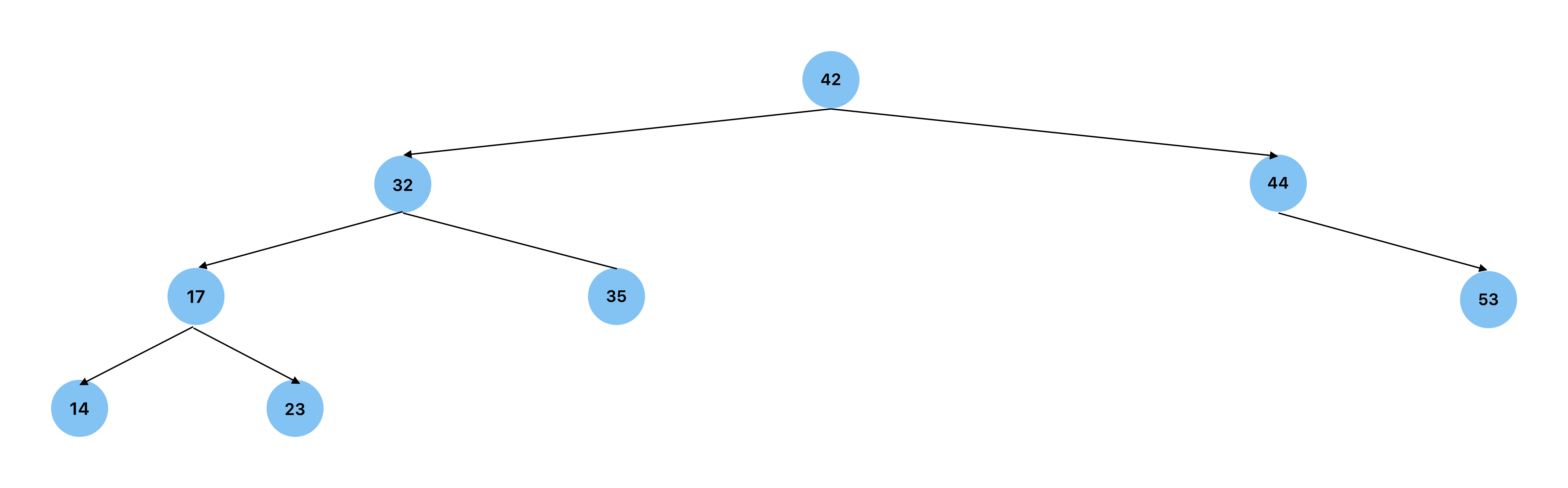
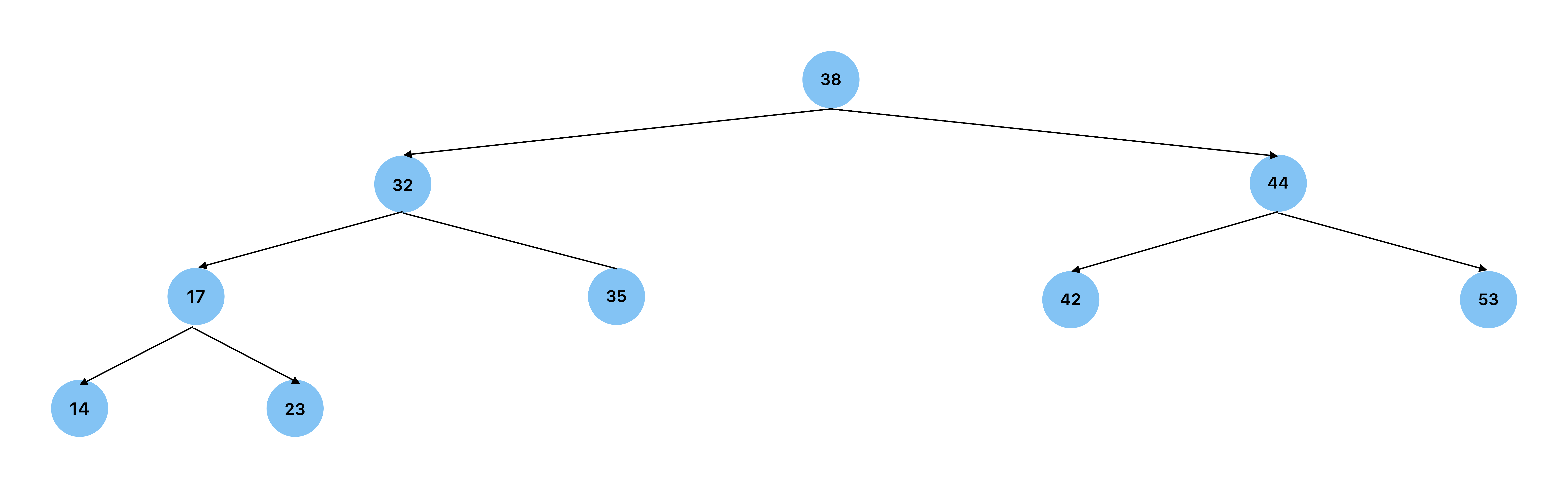
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 14 | 23 | 17 | 32 | 37 | 35 | 29 | 42 | 40 | 53 | 44 | 38 |

- Áp dụng cách làm tương tự với mỗi cây con:

**b.** Cây ban đầu: (ở trên)

- Sau khi xoá 37:

- Sau khi xoá 40:

- Sau khi xoá 29:

- Sau khi xoá 38:

**c.**

|  |
| --- |
| //Hàm tính tổng các nút trên mức k  **void** Sum(Tree root, **int** k, **int** &s) {  **if** (root != **NULL**) {  k--;  **if** (k == 0)  s += root->key;  **else** {  **if** (root->left != **NULL**)  Sum(root->left, k, s);  **if** (root->right != **NULL**)  Sum(root->right, k, s);  }  }  } |

**Câu 4 (2.5 điểm):**

a. Anh/Chị hãy cho biết ưu điểm bảng băm khi so với cấu trúc dữ liệu đã học (mảng, danh sách liên kết đơn, cây).

b. Cho bảng băm A kích thước 7 phần tử và tập khóa K={76, 40, 48, 5, 20}. Hãy vẽ bảng băm khi thêm từng khóa K vào bảng A với hàm băm hf(k) = k % 7, trong trường hợp xảy ra đụng độ, hãy sử dụng phương pháp dò bậc hai (Quadratic Probing Method) để giải quyết đụng độ.

**GIẢI:**

**a.**

Một số ưu điểm của bảng băm so với các cấu trúc dữ liệu khác như mảng, danh sách liên kết đơn và cây:

- **Tìm kiếm nhanh:** Truy cập và tìm kiếm trong bảng băm có thể được thực hiện với thời gian hằng số O(1), với điều kiện rằng hàm băm được triển khai tốt và không có quá nhiều đụng độ.

- **Dùng ít bộ nhớ:** So với cấu trúc dữ liệu như danh sách liên kết đơn và cây, bảng băm thường dùng ít bộ nhớ hơn. Bởi vì không cần lưu trữ các con trỏ hay tham chiếu giữa các phần tử, chỉ cần lưu trữ các khóa và giá trị tương ứng.

- **Xử lý đụng độ:** Bảng băm cung cấp cơ chế xử lý đụng độ, cho phép lưu trữ các khóa có cùng giá trị băm trong cùng một ô. Phương pháp xử lý đụng độ có thể được lựa chọn để đảm bảo hiệu quả tốt nhất trong trường hợp cụ thể.

- **Thao tác chèn và xóa linh hoạt:** Bảng băm cho phép thực hiện các thao tác chèn (insert) và xóa (delete) với độ phức tạp trung bình O(1). Điều này là do thời gian tìm kiếm và truy cập vào ô băm được giữ ở mức độ hằng số.

**- Lưu ý:** Trong trường hợp tốt nhất, việc tìm kiếm trong bảng băm có thể được thực hiện với thời gian hằng số O(1). Tuy nhiên, điều này đòi hỏi một số giả định:

+ Giả định về hàm băm hoàn hảo: Hàm băm hoàn hảo là một hàm băm đảm bảo không xảy ra đụng độ, có nghĩa là mỗi khóa sẽ được ánh xạ vào một vị trí duy nhất trong bảng băm. Trong trường hợp này, việc tìm kiếm chỉ cần một phép tính băm và truy cập vào vị trí tương ứng, cho nên có thể được thực hiện với thời gian hằng số O(1).

+ Giả định về truy cập ngẫu nhiên: Giả định rằng việc truy cập các khóa trong bảng băm được thực hiện ngẫu nhiên, không có mẫu tìm kiếm đặc biệt hoặc tập truy vấn có thể tạo ra đụng độ tập trung. Trong trường hợp này, việc tìm kiếm trong bảng băm vẫn có thể đạt được thời gian hằng số O(1) trung bình.

**b.**

- Với kích thước 7 ô, ta có các hàm băm sau:

+ Hàm băm: hf(k) = k % 7

+ Hàm băm lại: hfi(k) = [hf(k) + i2] % 7

- Cách làm cụ thể được minh hoạ dưới bảng sau (hàng trên là vị trí i, hang dưới là giá trị k)

+ Với **k lần lượt là** **76 và 40** thì ta được vị trí i là 6 và 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **NULL** | **NULL** | **NULL** | **NULL** | **NULL** | **40** | **76** |

+ Với **k = 48:**

* hf(k) = 6 => xảy ra đụng độ với hf(key = 76), do đó ta sẽ dò bậc 2 để xử lý.
* hfi(k) = [hf(k) + i2] % 7 = (6 + i2) % 7
* hfi(k) = (6 + i2) % 7. Với i = 1, ta được hf1(48) = 0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **48** | **NULL** | **NULL** | **NULL** | **NULL** | **40** | **76** |

**+** Với **k = 5:**

* hf(k) = 5 => xảy ra đụng độ với hf(key = 40), do đó ta tiếp tục dò bậc 2 để xử lý.
* hfi(k) = [hf(k) + i2] % 7 = (5 + i2) % 7
* hfi(k) = (5 + i2) % 7. Với i = 2, ta được hf2(5) = 2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **48** | **NULL** | **5** | **NULL** | **NULL** | **40** | **76** |

**+** Với **k = 20:**

* hf(k) = 6 => xảy ra đụng độ với hf(key = 76), do đó ta tiếp tục dò bậc 2 để xử lý.
* hfi(k) = [hf(k) + i2] % 7 = (6 + i2) % 7
* hfi(k) = (6 + i2) % 7. Với i = 2, ta được hf2(20) = 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **48** | **NULL** | **5** | **20** | **NULL** | **40** | **76** |

**=> Bảng băm sau khi được thêm các key vào là:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **48** | **NULL** | **5** | **20** | **NULL** | **40** | **76** |

**ĐỀ 11:**

**Câu 1 (4 điểm):**

Giả sử người ta có nhu cầu dùng danh sách liên kết để lưu trữ các số nguyên dương. Anh/chị hãy thực hiện :

a. Định nghĩa cấu trúc dữ liệu để có thể lưu trữ như yêu cầu (1 điểm)  
b. Viết hàm nhập các số nguyên vào danh sách, việc nhập kết thúc khi nhập giá trị -1 (1 điểm)

c. Viết hàm void TinhToan(ptr Head) in ra giá trị lớn nhất, giá trị nhỏ nhất và giá trị trung bình của danh sách. Qui ước: nếu danh sách là rỗng thì các giá trị này đều là 0. (1 điểm)  
d. Viết hàm sắp xếp danh sách theo giá trị của phần tử trong danh sách giảm dần và in ra màn hình (1 điểm)

**Giải**

**a.** Cấu trúc dữ liệu được định nghĩa như sau:

|  |
| --- |
| #include <iostream>  **using** **namespace** std;  **typedef** **struct** tagNode  {  **int** data;  **struct** tagNode\* Next;  } Node;  **typedef** **struct** tagList  {  Node\* Head;  Node\* Tail;  } List; |

**b.** Hàm nhập:

|  |
| --- |
| **void** InputList(List &l)  {  **int** x;  **do**  {  cin >> x;  **if** (x == -1)  **break**;  AddTail(l, CreateNode(x));  }  **while** (x != -1);  } |

**c.** Hàm tính toán theo yêu cầu:

Sơ lược về hàm:

- Ta khởi tạo các giá trị S, i, Min, Max, Avg và 1 Node p kiểu con trỏ để lưu giá trị l.Head của danh sách

- Trước tiên ta kiểm tra danh sách có rỗng không, nếu có thì xuất các giá trị 0 như yêu cầu đề, nếu không thì t duyệt qua danh sách.

- Mỗi lần duyệt thì biến i sẽ được tăng lên 1, để đếm số lượng phần tử có trong danh sách, đồng thời ta cũng tìm Max và Min sau mỗi lần duyệt.

- Sau khi duyệt xong, ta tiến hành tính giá trị trung bình của danh sách.

- Cuối cùng, xuất các giá trị theo yếu cầu của đề bài.

|  |
| --- |
| **void** TinhToan(List l)  {  **int** S = 0, i = 0;  **int** Min = l.Tail->data, Max = l.Head->data;  **float** Avg = 0.0;  Node\* p = l.Head;  **if** (IsEmpty(l) == **true**)  {  cout << "- The list is empty!\n";  S = 0; Min = 0; Max = 0; Avg = 0.0;  }  **else**  {  **while** (p != **NULL**)  {  S = S + p->data;  **if** (p->data > Max)  Max = p->data;  **if** (p->data < Min)  Min = p->data;  i++;  p = p->Next;  }  Avg = **float**(S/i);  }  cout << "- Tong cac phan tu trong danh sach: " << S << endl;  cout << "- Phan tu lon nhat trong danh sach: " << Max << endl;  cout << "- Phan tu nho nhat trong danh sach: " << Min << endl;  cout << "- Gia tri trung binh cac phan tu trong danh sach: " << Avg << endl;  }  /\*  Giải thích:  - S: là tổng các giá trị trong List  - i: là tổng số giá trị có trong List  - Max: là giá trị lớn nhất List  - Min: Là giá trị nhỏ nhất List  - Avg: là giá trị trung bình của List \*/ |

**d.** Hàm sắp xếp (Quick Sort): Ta viết thêm 1 hàm PrinList, sau đó là hàm QuickSort. Ta xuống hàm main() để thực hiện gọi hàm QuickSort và in danh sách ra màn hình.

|  |  |
| --- | --- |
| **void** PrintList(List l)  {  Node \*p = l.Head;  **while** (p != **NULL**)  {  cout << p->data << " ";  p = p->Next;  }  }  **void** QuickSort(List &L)  {  Node \*p, \*X;  List L1, L2;    **if** (L.Head == L.Tail)  **return**;    CreateList(L1);  CreateList(L2);    X = L.Head;  L.Head = X->Next;    **while** (L.Head != **NULL**)  {  p = L.Head;  L.Head = p->Next;  p->Next = **NULL**;  **if** (p->data > X->data)  AddTail(L1, p);  **else**  AddTail(L2, p);  } | QuickSort(L1);  QuickSort(L2);    **if** (L1.Head != **NULL**)  {  L.Head = L1.Head;  L1.Tail->Next = X;  }  **else**  L.Head = X;    X->Next = L2.Head;    **if** (L2.Head != **NULL**)  L.Tail = L2.Tail;  **else**  L.Tail = X;  }  **int** main()  {  List L;  CreateList(L);  InputList(L);  TinhToan(L);  QuickSort(L);  cout << "- List: ";  PrintList(L);  } |

**Câu 2: (2 điểm)**

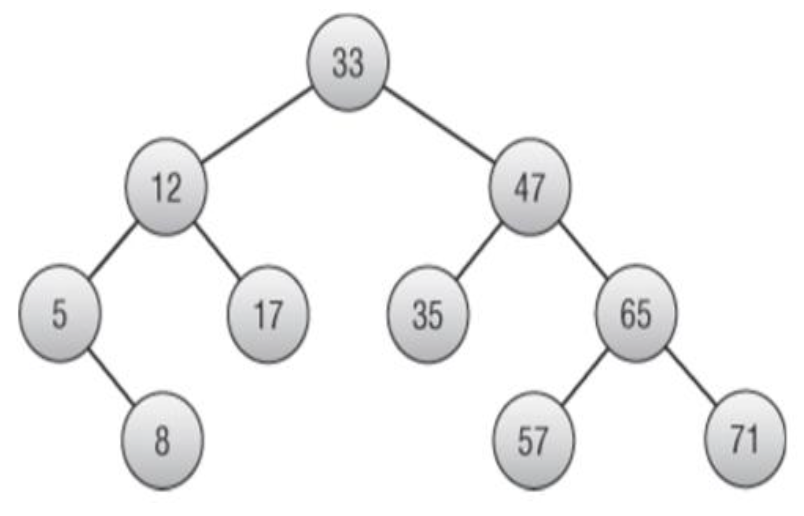
Hãy phát biểu định nghĩa cây nhị phân tìm kiếm. Vẽ cây nhị phần tìm kiếm (chỉ vẽ cây kết quả) từ dãy số nguyên khi xây dựng cây theo thứ tự từ trái qua phải của dãy số: 72; 67; 73; 58; 5; 4; 27; 53; 61; 32.

**GIẢI:**

**Cây Nhị phân Tìm kiếm (Binary Search Tree)** là một **cây nhị phân** có thể tìm kiếm sự xuất hiện của 1 phần tử trên cây với độ phức tạp **O(log(N)).**

A screenshot of a computer

Description automatically generated**Vẽ:**

**Câu 3: (3 điểm)**

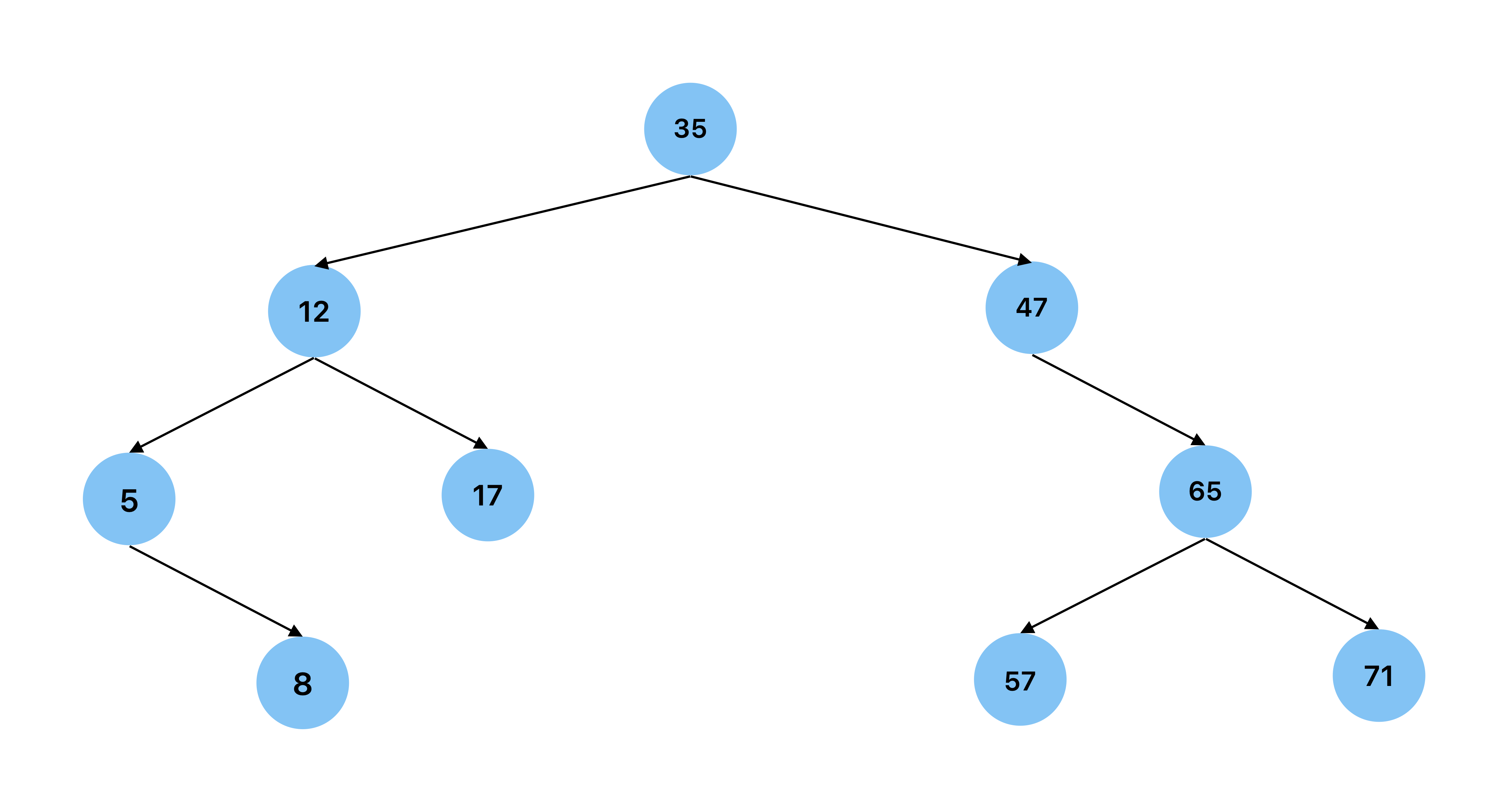
Cho cây nhị phân tìm kiếm T như hình bên.

Anh /chị hãy thực hiện:

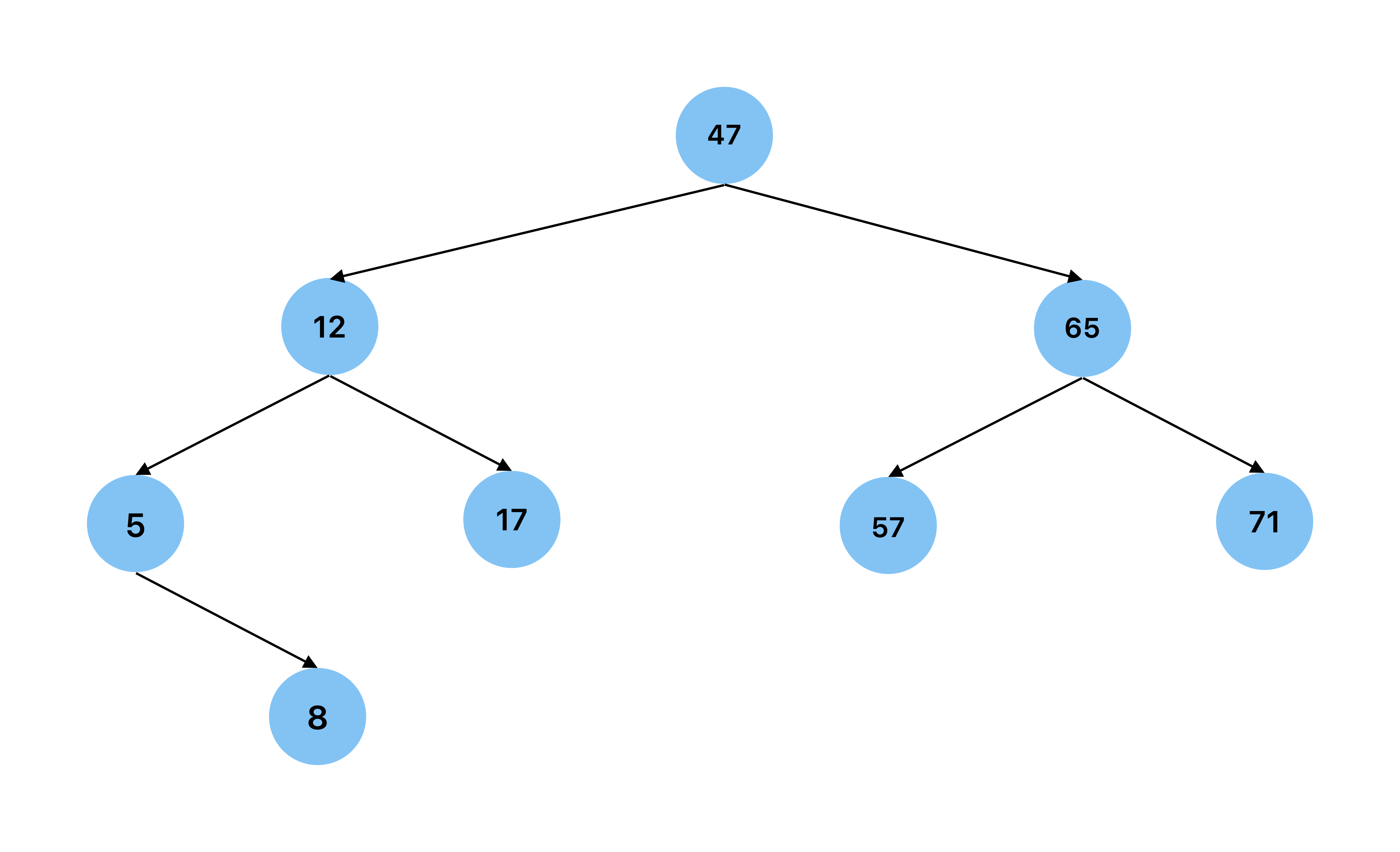
**a.** Viết hàm in ra các nút trên cây tại mức thứ k của cây. Ví dụ : nhập k = 0 (mức gốc) thì in ra 33.  
**b.** Hãy vẽ lại cây khi xóa lần lượt nút 33; 35;12; 5. Nếu khi xóa nút, cây bị mất cân bằng, hãy cân bằng lại cây và cho biết loại mất cân bằng (nếu có)

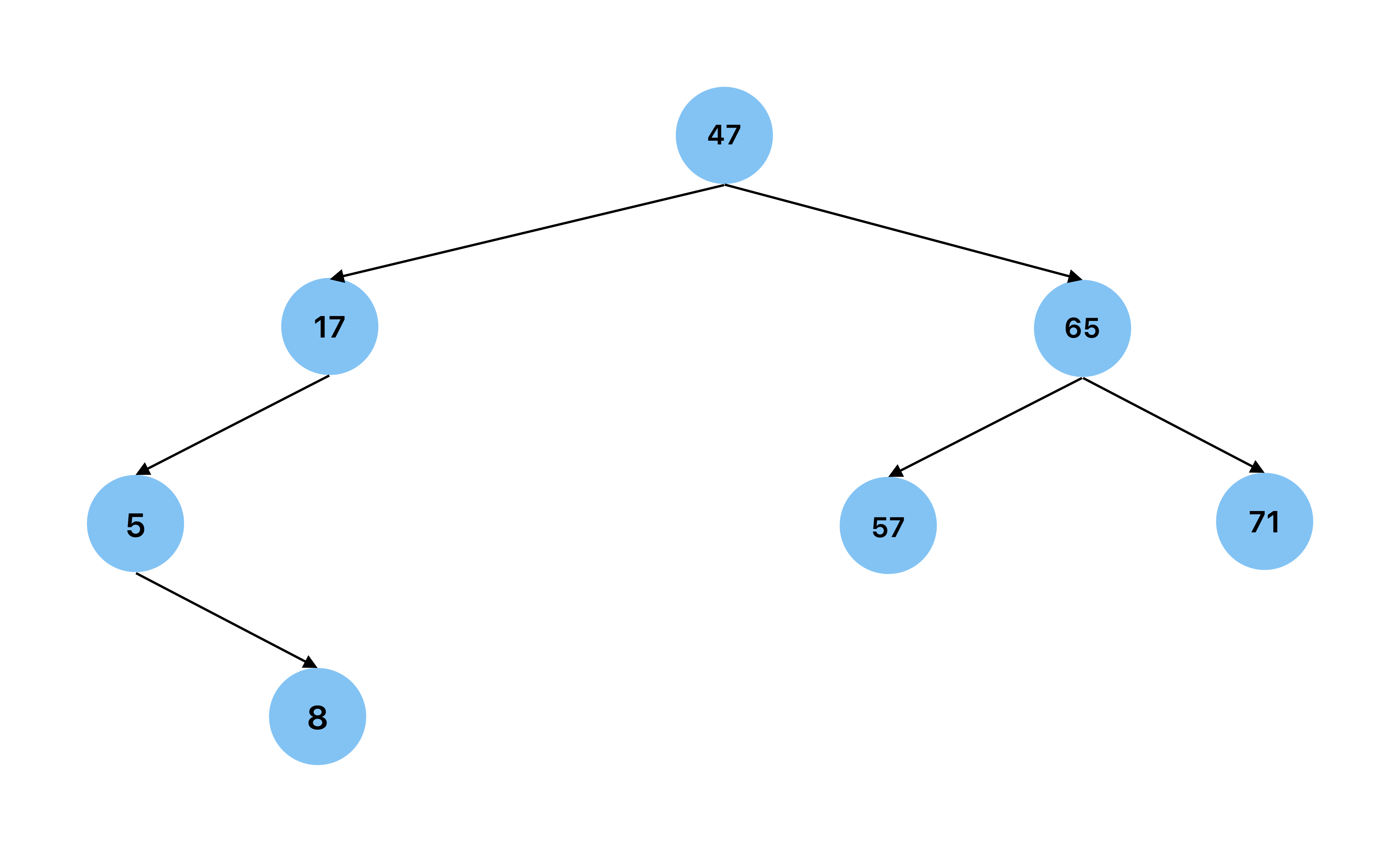
**GIẢI:**

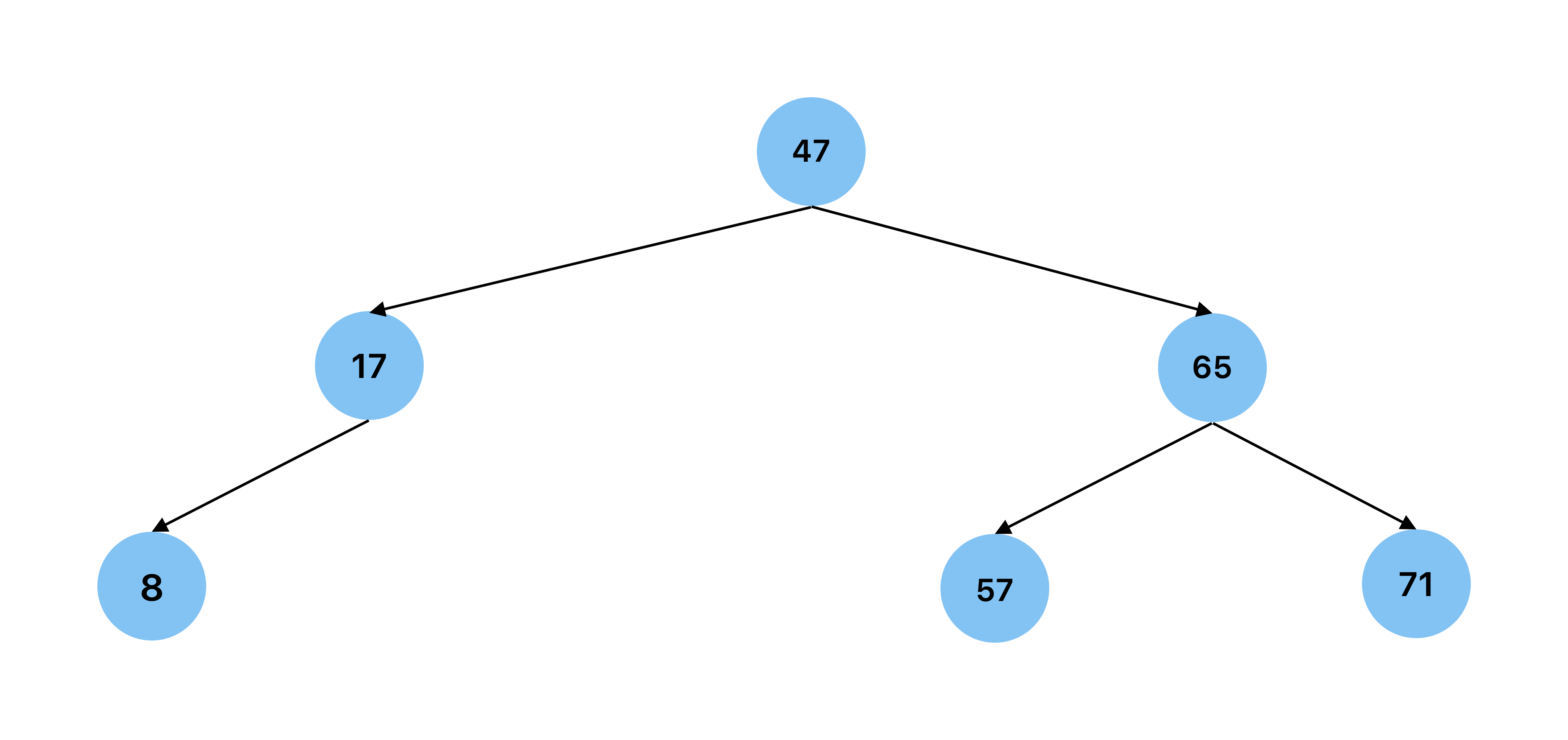
|  |
| --- |
| //Hàm in các node trên mức k  **void** Sum(Tree root, **int** k) {  **if** (root != **NULL**) {  k--;  **if** (k == 0)  cout << root->key << " ";  **else** {  **if** (root->left != **NULL**)  Sum(root->left, k);  **if** (root->right != **NULL**)  Sum(root->right, k);  }  }  } |

**b.**

Xoá 33:

Xoá 35:

Xoá 12:

Xoá 5:

**Câu 4**: (**1 điểm)**

Giả sử cho bảng băm A kích thước 7 ô và tập khóa K = {76, 93, 40, 47, 10, 55}, ta cần nạp các giá trị khóa K vào bảng A sử dụng hàm băm H1(k) = k % 7. Hãy vẽ bảng băm kết quả, trong trường hợp xảy ra đụng độ, hãy sử dụng phương pháp băm kép để xử lý, với hàm băm thứ 2 do anh/chị tự đề nghị.

**GIẢI:**

- Với kích thước 7 ô, đặt M = 7, ta có các hàm băm sau:

+ H1(k) = k % 7

+ H2(k) = (M – 2) – [k % (M – 2)] = 5 – (k % 5)

- Gọi H1(k) = a và H2(k) = b, ta có hàm băm lại lần thứ i: Hi(k) = (a + b\*i) % 7

- Cách làm cụ thể được minh hoạ dưới bảng sau (hàng trên là vị trí i, hang dưới là giá trị k)

+ Với k lần lượt là **76, 93, 40** thì ta được vị trí i: 6, 2, 5.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **NULL** | **NULL** | **93** | **NULL** | **NULL** | **40** | **76** |

+ Với **k = 47**

* H1(k) = 5 => xảy ra đụng độ với H(key = 40), do đó ta sẽ băm kép để xử lý
* H2(k) = 5 – (47 % 5) = 5 – 2 = 3.
* Hi(k) = (a + b\*i)%7 = (5 + 3\*i) % 7. Với i = 1, ta được H1(47) = 1.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **NULL** | **47** | **93** | **NULL** | **NULL** | **40** | **76** |

+ Với **k = 10**, ta được vị trí i = 3.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **NULL** | **47** | **93** | **10** | **NULL** | **40** | **76** |

+ Với **k = 55**

* H1(k) = 6 => xảy ra đụng độ với H(key = 76), do đó ta sẽ băm kép để xử lý
* H2(k) = 5 – (55 % 5) = 5 – 0 = 5.
* Hi(k) = (a + b\*i)%7 = (6 + 5\*i) % 7. Với i = 1, ta được H1(55) = 4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **NULL** | **47** | **93** | **10** | **55** | **40** | **76** |

**=> Bảng băm sau khi được thêm các key vào là:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **NULL** | **47** | **93** | **10** | **55** | **40** | **76** |