**Roll No…………….. Total No. of Pages:……**

**ST-2 (SET-V)**

**6th SEMESTER 2023-24**

**CS192- Advanced Data Structures**

**Time allowed: 90 Minutes Max. Marks: 40**

**General Instructions:**

* **Follow the instructions given in each section.**
* **Make sure that you attempt the questions in order.**

**SECTION-A (10\*1 mark=10 marks)**

***(All questions are compulsory)***

1. In C++, which of the following data structures is implemented using recursion?
   1. Linked list
   2. **Binary search tree**
   3. Array
   4. Queue
2. What is the difference between direct and indirect recursion in C++?
   1. **Direct recursion involves one function calling itself directly, while indirect recursion involves multiple functions calling each other.**
   2. Direct recursion is more efficient than indirect recursion.
   3. Indirect recursion involves one function calling itself directly, while direct recursion involves multiple functions calling each other.
   4. There is no difference; the terms are used interchangeably.
3. What is the primary difference between iteration and recursion?
   1. Iteration uses function calls, while recursion uses loops
   2. **Iteration doesn't involve function calls, while recursion does**
   3. Iteration always requires a stack, while recursion doesn't
   4. Iteration is more memory-efficient than recursion
4. The "Traveling Salesman Problem" can be effectively solved using:
   1. Greedy algorithms
   2. Divide and conquer
   3. Dynamic programming
   4. **Backtracking**
5. When implementing a problem involving movement on a grid, how can you ensure that each cell is visited exactly once?
   1. **Use a visited array**
   2. Use a counter variable
   3. Visit cells randomly
   4. Visit cells in a spiral pattern
6. In a recursive function that traverses a grid, what should be the criteria for making a recursive call?
   1. Moving to the cell with the smallest value
   2. **Moving to the unvisited neighboring cell**
   3. Moving to the cell with the largest value
   4. Moving to a random neighboring cell
7. What happens if you don't provide a destructor for a class in C++?
   1. **The compiler will automatically generate a default destructor**
   2. The class cannot be instantiated
   3. The program will fail to compile
   4. Memory leaks can occur as resources won't be properly deallocated
8. Which of the following is not a valid constructor?
   1. MyClass()
   2. MyClass(int x)
   3. **void MyClass()**
   4. explicit MyClass(int x)
9. Which operation requires updating both the "next" and "previous" pointers in a doubly linked list?
   1. Insertion at the beginning
   2. Insertion at the end
   3. Deletion from the beginning
   4. **Deletion from the end**
10. What is the time complexity to reverse a singly linked list iteratively?
    1. O(1)
    2. **O(n)**
    3. O(log n)
    4. O(n^2)

**SECTION-B (5\*2 mark=10 marks)**

***(All questions are compulsory)***

11) struct Node {

int data;

Node\* next;

Node(int val) : data(val), next(nullptr) {}

};

int main() {

Node\* head = new Node(3);

cout << head->data;

return 0;

}

1. 0
2. **3**
3. Error
4. Garbage Value

12) #include <iostream>

using namespace std;

int fun(int n) {

if (n <= 1)

return n;

return fun(n - 1) + fun(n - 2);

}

int main() {

cout << fun(5);

return 0;

}

**a) 5**

b) 8

c) 3

d) 13

13) class A {

public:

virtual void print() { cout << "A"; }

};

class B : public A {

public:

void print() { cout << "B"; }

};

int main() {

B obj;

A \*ptr = &obj;

ptr->print();

return 0;

}

1. A
2. **B**
3. Compile Error
4. Undefined Behavior

14) #include <iostream>

using namespace std;

int countWays(int n) {

if (n <= 1) {

return 1;

}

return countWays(n - 1) + countWays(n - 2);

}

int main() {

int n = 4;

cout << "Ways: " << countWays(n) << endl;

return 0;

}

**a) Ways: 5**

b) Ways: 8

c) Ways: 4

d) Ways: 6

15) Consider the following recursive function that calculates the nth power of x using a divide-and-conquer approach:

double power(double x, int n) {

if (n == 0)

return 1.0;

double halfPower = power(x, n / 2);

if (n % 2 == 0)

return halfPower \* halfPower;

else

return x \* halfPower \* halfPower;

}

What is the time complexity of this function?

a) O(1)

**b) O(log n)**

c) O(n)

d) O(n^2)

**SECTION-C(Coding Question) (2x5 marks=5 marks)**

Q16) Multiply two matrices using recursion.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Matrix A:  1 2 3  4 5 6  7 8 9  Matrix B:  9 8 7  6 5 4  3 2 1 | Matrix A:  12 4 3  6 5 6  3 8 9  Matrix B:  1 7 8  3 5 5  1 4 8 | Matrix A:  0 1 3  7 3 1  12 6 4  Matrix B:  11 7 32  9 2 9  3 2 8 |
| **Output** | Matrix C (Product of A and B):  30 24 18  84 69 54  138 114 90 | Matrix C (Product of A and B):  27 116 140  27 91 121  36 97 136 | Matrix C (Product of A and B):  18 8 33  107 57 259  198 104 470 |

Solution :

**#include <iostream>**

**using namespace std;**

**const int N = 3;**

**void multiplyMatrices(int A[][N], int B[][N], int C[][N], int rowA, int colA, int colB) {**

**if (rowA == N) {**

**return;**

**}**

**if (colA == N) {**

**multiplyMatrices(A, B, C, rowA + 1, 0, 0);**

**return;**

**}**

**if (colB == N) {**

**multiplyMatrices(A, B, C, rowA, colA + 1, 0);**

**return;**

**}**

**C[rowA][colB] = 0;**

**for (int k = 0; k < N; k++) {**

**C[rowA][colB] += A[rowA][k] \* B[k][colB];**

**}**

**multiplyMatrices(A, B, C, rowA, colA, colB + 1);**

**}**

**int main() {**

**int A[N][N] = {{0, 1, 3}, {7, 3, 1}, {12, 6, 4}};**

**int B[N][N] = {{11, 7, 32}, {9, 2, 9}, {3, 2, 8}};**

**int C[N][N] = {0};**

**multiplyMatrices(A, B, C, 0, 0, 0);**

**cout << "Matrix A:" << endl;**

**for (int i = 0; i < N; i++) {**

**for (int j = 0; j < N; j++) {**

**cout << A[i][j] << " ";**

**}**

**cout << endl;**

**}**

**cout << "Matrix B:" << endl;**

**for (int i = 0; i < N; i++) {**

**for (int j = 0; j < N; j++) {**

**cout << B[i][j] << " ";**

**}**

**cout << endl;**

**}**

**cout << "Matrix C (Product of A and B):" << endl;**

**for (int i = 0; i < N; i++) {**

**for (int j = 0; j < N; j++) {**

**cout << C[i][j] << " ";**

**}**

**cout << endl;**

**}**

**return 0;**

**}**

Q17) Given a linked list of size N and a key. The task is to insert the key in the middle of the linked list.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Linked list= 7 8 5 1 9  Key= 99 | Linked list= 12 54 97 5 7 98  Key= 100 | Linked list= 1 2 3 4 5  Key= 66 |
| **Output** | 7 8 5 99 1 9 | 12 54 97 100 5 7 98 | 1 2 3 66 4 5 |

Solution :

**#include <iostream>**

**using namespace std;**

**// Node structure for singly linked list**

**struct Node {**

**int data;**

**Node\* next;**

**};**

**// Function to insert a new node at the end of the linked list**

**Node\* insertAtEnd(Node\* head, int data) {**

**Node\* newNode = new Node;**

**newNode->data = data;**

**newNode->next = nullptr;**

**if (head == nullptr) {**

**return newNode;**

**}**

**Node\* current = head;**

**while (current->next != nullptr) {**

**current = current->next;**

**}**

**current->next = newNode;**

**return head;**

**}**

**// Function to find the middle node of the linked list**

**Node\* findMiddleNode(Node\* head) {**

**if (head == nullptr || head->next == nullptr) {**

**return head;**

**}**

**Node\* slow = head;**

**Node\* fast = head->next;**

**while (fast != nullptr && fast->next != nullptr) {**

**slow = slow->next;**

**fast = fast->next->next;**

**}**

**return slow;**

**}**

**// Function to insert the key in the middle of the linked list**

**Node\* insertInMiddle(Node\* head, int key) {**

**if (head == nullptr) {**

**return nullptr;**

**}**

**// Find the middle node**

**Node\* middle = findMiddleNode(head);**

**// Create a new node with the key**

**Node\* newNode = new Node;**

**newNode->data = key;**

**// Insert the new node after the middle node**

**newNode->next = middle->next;**

**middle->next = newNode;**

**return head;**

**}**

**// Function to display the linked list**

**void displayLinkedList(Node\* head) {**

**Node\* current = head;**

**while (current != nullptr) {**

**cout << current->data << " ";**

**current = current->next;**

**}**

**}**

**int main() {**

**Node\* head = nullptr;**

**int data, key;**

**// Create the linked list with user input**

**cout << "Enter elements for linked list (enter -1 to stop):\n";**

**while (true) {**

**cin >> data;**

**if (data == -1)**

**break;**

**head = insertAtEnd(head, data);**

**}**

**// Display the linked list before inserting the key**

**cout << "Linked List before insertion: ";**

**displayLinkedList(head);**

**// Prompt the user to enter the key to be inserted**

**cout << "\nEnter the key to be inserted in the middle: ";**

**cin >> key;**

**// Insert the key in the middle of the linked list**

**head = insertInMiddle(head, key);**

**// Display the linked list after inserting the key**

**cout << "Linked List after insertion: ";**

**displayLinkedList(head);**

**// Free memory by deleting nodes**

**while (head != nullptr) {**

**Node\* temp = head;**

**head = head->next;**

**delete temp;**

**}**

**return 0;**

**}**

**SECTION-D (Coding Question)(1x10 mark=10 mark)**

Q18) Given a collection of candidate numbers and a target value, find all unique combinations in the candidates where the chosen numbers sum to the target. Each number in candidates may only be used once.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | candidates = {10, 1, 2, 7, 6, 1, 5}  target = 8 | candidates = {10, 1, 2, 7, 6, 1, 5}  target = 12 | candidates = {5,9,16,4,5,6,13}  target = 13 |
| **Output** | Combinations that sum up to 8:  1 1 6  1 2 5  1 7  2 6 | Combinations that sum up to 12:  1 1 10  1 5 6  2 10  5 7 | Combinations that sum up to 13:  4 9  13 |

Solution :

**#include <iostream>**

**#include <vector>**

**#include <algorithm>**

**using namespace std;**

**void findCombinations(vector<int>& candidates, int target, int start, vector<int>& current, vector<vector<int>>& result) {**

**if (target < 0)**

**return;**

**if (target == 0) {**

**result.push\_back(current);**

**return;**

**}**

**for (int i = start; i < candidates.size(); i++) {**

**if (i > start && candidates[i] == candidates[i - 1])**

**continue; // Skip duplicates**

**current.push\_back(candidates[i]);**

**findCombinations(candidates, target - candidates[i], i + 1, current, result);**

**current.pop\_back();**

**}**

**}**

**vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {**

**sort(candidates.begin(), candidates.end());**

**vector<vector<int>> result;**

**vector<int> current;**

**findCombinations(candidates, target, 0, current, result);**

**return result;**

**}**

**int main() {**

**vector<int> candidates = {5,9,16,4,5,6,13};**

**int target = 13;**

**vector<vector<int>> result = combinationSum2(candidates, target);**

**cout << "Combinations that sum up to " << target << ":\n";**

**for (const vector<int>& combination : result) {**

**for (int num : combination) {**

**cout << num << " ";**

**}**

**cout << endl;**

**}**

**return 0;**

**}**