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

**ST-2 (SET-II)**

**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. Backtracking is an algorithmic technique for solving problems that involves:
   1. **Trying all possible solutions**
   2. Iterating through a loop
   3. Using dynamic programming
   4. Using recursion only
2. What is the main characteristic of backtracking algorithms?
   1. They guarantee the optimal solution.
   2. They use divide and conquer approach.
   3. They use recursion for efficient solutions.
   4. **They explore multiple possibilities and backtrack when needed.**
3. What is the key characteristic of a tail-recursive function?
   1. It calls itself multiple times for smaller sub-problems
   2. It uses a base case to terminate the recursion
   3. **It performs a single recursive call at the end of the function**
   4. It uses memoization to optimize the recursion
4. In a maze-solving problem, what is the role of recursion?
   1. To keep track of the visited cells
   2. **To backtrack and explore different paths**
   3. To optimize the path-finding algorithm
   4. To create a new maze configuration
5. Which linked list type can be easily implemented using an array?
   1. Singly linked list
   2. Doubly linked list
   3. Circular linked list
   4. **Array-based list**
6. Which type of recursion involves a function calling itself multiple times during a single recursion step?
   1. Tail recursion
   2. **Multiple recursion**
   3. Linear recursion
   4. Indirect recursion
7. In C++, which data structure is used to implement a recursive algorithm to find the factorial of a number?
   1. Linked list
   2. Array
   3. Queue
   4. **Stack**
8. What is a constructor in C++?
   1. **A member function that destructs objects**
   2. A member function that creates objects
   3. A special member function that initializes objects
   4. A built-in function for memory allocation
9. How many constructors can a class have?
   1. Only one
   2. At least two
   3. **As many as needed**
   4. Two, one for objects and one for pointers
10. What is dynamic binding in C++?
    1. A way to create multiple instances of a class
    2. A way to achieve static polymorphism
    3. A way to determine the type of an object at runtime
    4. **A way to choose the implementation of a function at runtime**

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

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

11) #include <iostream>

using namespace std;

int gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

int main() {

cout << gcd(48, 18);

return 0;

}

a) 48

b) 18

**c) 6**

d) 12

12) struct Node {

int data;

Node\* next;

};

int main() {

Node\* head = nullptr;

Node\* newNode = new Node{2, nullptr};

head->next = newNode;

cout << head->next->data;

return 0;

}

a) 0

b) 2

c) Garbage value

**d) Compilation error**

13) class A {

public:

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

};

class B : public A {

public:

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

};

int main() {

A \*a = new B;

B \*b = static\_cast<B \*>(a);

b->print();

return 0;

}

a) A

**b) B**

c) Compile Error

d) Undefined Behavior

14) #include <iostream>

using namespace std;

void backtrack(int n) {

if (n <= 0) return;

cout << n << " ";

backtrack(n - 2);

cout << n << " ";

}

int main() {

backtrack(6);

return 0;

}

a) 6 4 2 1 3 5

b) 6 5 4 3 2 1

**c) 6 4 2 2 4 6**

d) 6 4 2 1 3 5

15) What is the time complexity of finding the factorial of a number using recursion?

a) O(1)

**b) O(n)**

c) O(log n)

d) O(n^2)

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

Q16) Write a recursive function to count the number of digits in a positive integer.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter a positive integer: 374 | Enter a positive integer: 97 | Enter a positive integer: 16789 |
| **Output** | Number of digits: 3 | Number of digits: 2 | Number of digits: 5 |

Solution :

**#include <iostream>**

**// Recursive function to count number of digits**

**int countDigits(int n) {**

**// Base case: If the number is 0, it has no digits, so return 0.**

**if (n == 0)**

**return 0;**

**// Recursive step: Return 1 (for the current digit) plus the count of digits**

**// obtained by calling the function with the remaining digits (n / 10).**

**return 1 + countDigits(n / 10);**

**}**

**int main() {**

**int n;**

**std::cout << "Enter a positive integer: ";**

**std::cin >> n;**

**// Call the countDigits function with the input positive integer.**

**int numDigits = countDigits(n);**

**// Output the number of digits.**

**std::cout << "Number of digits: " << numDigits << std::endl;**

**return 0;**

**}**

Q17) Implement a linked list with inserting elements from the end.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | Enter elements for linked list (enter -1 to stop): 5 7 9 -1 | Enter elements for linked list (enter -1 to stop): 12 65 98 7 34 -1 | Enter elements for linked list (enter -1 to stop): 3 2 5 -1 |
| **Output** | Linked List: 5 7 9 | Linked List: 12 65 98 7 34 | Linked List: 3 2 5 |

Solution :

**#include <iostream>**

**using namespace std;**

**// Node structure for linked list**

**struct Node {**

**int data;**

**Node\* next;**

**};**

**// Function to insert an element 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 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;**

**// 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**

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

**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 maze with obstacles represented by 0, find a path for a robot to reach its destination from the top-left cell to the bottom-right cell.

The robot can only move down or right.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Test Case 1** | **Test Case 2** | **Test Case 3** |
| **Input** | {1, 0, 0, 0}  {1, 1, 0, 1}  {0, 1, 0, 0}  {1, 1, 1, 1} | {1, 0, 0, 0}  {1, 1, 0, 1}  {0, 1, 0, 0}  {1, 0, 1, 1} | {1, 0, 0, 0}  {1, 1, 1, 1}  {0, 0, 1, 0}  {1, 1, 1, 1} |
| **Output** | Path exists. Solution:  1 0 0 0  1 1 0 0  0 1 0 0  0 1 1 1 | No path exists. | Path exists. Solution:  1 0 0 0  1 1 1 0  0 0 1 0  0 0 1 1 |

Solution :

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**const int N = 4; // Size of the maze**

**bool isSafe(int maze[N][N], int x, int y) {**

**return (x >= 0 && x < N && y >= 0 && y < N && maze[x][y] == 1);**

**}**

**bool solveMazeUtil(int maze[N][N], int x, int y, vector<vector<int>>& path) {**

**if (x == N - 1 && y == N - 1) {**

**path[x][y] = 1;**

**return true;**

**}**

**if (isSafe(maze, x, y)) {**

**path[x][y] = 1;**

**if (solveMazeUtil(maze, x + 1, y, path))**

**return true;**

**if (solveMazeUtil(maze, x, y + 1, path))**

**return true;**

**path[x][y] = 0; // Backtrack if no valid path found**

**}**

**return false;**

**}**

**void printSolution(const vector<vector<int>>& path) {**

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

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

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

**}**

**cout << endl;**

**}**

**}**

**void solveMaze(int maze[N][N]) {**

**vector<vector<int>> path(N, vector<int>(N, 0));**

**if (solveMazeUtil(maze, 0, 0, path)) {**

**cout << "Path exists. Solution:\n";**

**printSolution(path);**

**} else {**

**cout << "No path exists.";**

**}**

**}**

**int main() {**

**int maze[N][N] = {**

**{1, 0, 0, 0},**

**{1, 1, 0, 1},**

**{0, 1, 0, 0},**

**{1, 1, 1, 1}**

**};**

**solveMaze(maze);**

**return 0;**

**}**