**Phần Trắc Nghiệm**

Câu 1: Thuật toán tìm kiếm nhị phân có thể phân loại vào phương pháp nào bên dưới ?

1. Quy hoạch động
2. Vết cạn
3. Tham lam
4. *Chia để trị*

Câu 2: Có 4 thuật toán A1, A2, A3, A4 có độ phức tạp lần lượt là log(n), nlog(n), log(log(n)), n/log(n). Thuật toán nào là tốt nhất?

1. A4
2. *A3*
3. A2
4. A1

Câu 3: Phương pháp quay lùi không thể giải quyết các bài toán nào?

1. Bài toán mạch Hamilton
2. *Bài toán người bán hàng*
3. Bài toán n quân hậu
4. Bài toán tổng tập con

Câu 4: Bài toán tìm tập con của các số nguyên dương của một mảng X sao cho có tổng của nó bằng một số nguyên dương cho trước là?

1. Bài toán n quân hậu
2. *Bài toán tổng tập con*
3. Bài toán balo
4. Bài toán mạch Hamilton

Câu 5: Độ phức tạp thời gian của thuật toán tìm kiếm nhị phân là?

1. O(n log n)
2. *O(log n)*
3. O(1)
4. O(n)

Câu 6: Khi chuyển biểu thức (12-a)\*(b+9)/(d\*4) sang biểu diễn ở dạng hậu tố ta được biểu thức nào?

1. /12a-b9+d4\*
2. *12a-b9+\*d4\*/*
3. 4b\*d9+a12-\*/
4. 12-a\*b+9/d\*4

Câu 7: Cho chuỗi sau S=(-10, 24, -9, 35, -21, 55, -41, 76, 84). Độ dài chuỗi con tăng dần dài nhất của chuỗi S là?

1. 7
2. *6*
3. 4
4. 5

Câu 8: Trong quy hoạch động kỹ thuật lưu trữ các giá trị tính toán trước đó được gọi là ?

1. Mapping
2. Saving value property
3. Storing value property
4. *Memoization*

Câu 9: Kết quả của chương trình sau là?

#include <bits/stdc++.h>

using namespace std;

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

    if (n == 0 || W == 0)

        return 0;

    if (wt[n - 1] > W)

        return knapSack(W, wt, val, n - 1);

    elsesao

        return max(

            val[n - 1]

                + knapSack(W - wt[n - 1],

                           wt, val, n - 1),

            knapSack(W, wt, val, n - 1));

}

int main()

{

    int val[] = { 60, 100, 120 };

    int wt[] = { 10, 20, 30 };

    int W = 50;

    int n = sizeof(val) / sizeof(val[0]);

    cout << knapSack(W, wt, val, n);

    return 0;

}

1. 100
2. Lỗi thực thi
3. 180
4. *220*

Câu 10: Chiến lược giải quyết một bài toán bằng cách kết hợp các lời giải tối ưu của các bài toán con gối nhau được gọi là ?

1. *Quy hoạch động*
2. Tham lam
3. Chia để trị
4. Đệ quy

Câu 11: Thuật toán quay lui nhanh hơn thuật toán vét cạn ?

1. *Đúng*
2. Sai

Câu 12: Hai đại lượng đo chính dùng để đánh giá tính hiệu quả của một thuật toán là gì?

1. Bộ xử lý và bộ nhớ
2. Độ phức tạp và công suất
3. Dữ liệu và không gian
4. *Thời gian và không gian*

Câu 13: Nếu 1 bài toán có thể chia thành các bài toán con được sử dụng lại nhiều lần, thì bài toán đó có tính chất là gì?

1. Cấu trúc con tối ưu
2. *Các bài toán con gối nhau*
3. Ghi nhớ
4. Tham lam

Câu 17: Độ phức tạp về thời gian khi áp dụng thuật toán vét cạn để giải quyết bài toán ba lô Knapsack là ?

1. O(n)
2. O(n!)

Câu 18: Thuật toán tham lam có thể áp dụng để giải quyết các bài toán quy hoạch động ?

1. Đúng
2. *Sai*

Câu 19: Phương trình đệ quy về tối ưu thời gian của bài toán Tháp Hà Nội với n đĩa là ?

1. T(n) = 2T(n/2)+n
2. T(n) = 2T(n/2)+c
3. T(n) = 2T(n-1)+n
4. *T(n) = 2T(n-1)+c*

Câu 20: Có 3 loại tờ tiền với các mệnh giá là 1, 3, 4. Tìm số cách ghép các tờ tiền trên để được tổng số tiền là 7 biết rằng số tờ tiền mỗi loại là không giới hạn và thức tự các tờ tiền không quan trọng?

1. 3
2. 4
3. *5*
4. 6

Câu 22: Phương pháp nào sau đâu có thể giải quyết được bài toán balo?

1. Vét cạn
2. Đệ quy
3. Quy hoạch động
4. *Cả A, B, C*

Câu 23: Bước đầu tiên trong việc giải quyết vấn đề là?

1. *Xác định vấn đề*
2. (Gì gì đó nhìn méo rõ)
3. Hiểu vấn đề
4. Không phải các đáp án trên

Câu 24: Lợi thế của phương pháp đệ qui so với phương pháp lặp là ?

1. Sử dụng ít bộ nhớ hơn
2. *Ít code hơn và dễ thực hiện*
3. Sử dụng nhiều bộ nhớ hơn
4. Phải viết nhiều code hơn

Câu 25: Cho mảng arr = [2, 5, 7, 99, 899], giả sử với key = 899. Chúng ta phải thực hiện đẹ quy bao nhiêu lần đề tìm được key (tìm kiếm nhị phân)?

1. 5
2. 2
3. *3*
4. 4

Câu 26: Một loại quy hoạch động có những tính chất nào

1. Các bài toán con gối nhau
2. *Có cấu trúc con tối ưu và các bài toán con gối nhau*
3. Cấu trúc con tối ưu
4. Cách tiếp cận tham lam

Câu 27: Thuật toán Kruskal thường dùng để?

1. *Tìm cây khung nhỏ nhất*
2. Tìm đường đi ngắn nhất
3. Duyệt đồ thị
4. Tìm chu trình Euler

Câu 28: Lời giải đệ quy của bài toán Tháp Hà Nội là một ví dụ minh họa của thuật toán được thiết kế bởi chiến lược nào?

1. Quay lui
2. Quy hoạch động
3. Tham lam
4. *Chia để trị*

Câu 30: Độ phức tạp của hàm fun() là?

int fun(int n){

int count = 0;

for (int i=n; i>0; i/=2){  
 for(int j=0; j<i; j++){

count += 1;

}

return count;

}

}

1. O(n^2)
2. O(nlogn)
3. *O(n)*
4. O(nlognlogn)

Câu 31: Cây không gian trạng thái của bài toán quay lui được xây dựng bằng cách nào?

1. Breadth – first search
2. *Depth – first search*
3. Nearest neighbour first
4. Twice around the tree

Câu 32: Khi thuật toán quay lui tìm được một lời giải hoàn chỉnh nó sẽ làm gì?

1. Không làm gì cả
2. Chuyển qua một đường đi khác
3. *Tiếp tục tìm kiếm các lời giải khả thi khác*
4. Quay lại gốc

Câu 33: Thuật toán sắp xếp nào dưới đây có thể được xem là sự cải tiến của thuật toán sắp xếp cây nhị phân?

1. *Heap sort*
2. Quick sort
3. Selection sort
4. Insertion sort

Câu 34: Mục tiêu chính của việc sử dụng phương pháp tiếp cận từ trên xuống (top-down) trong quy hoạch động đối với một bài toán là ?

1. Giảm cả độ phức tạp thời gian và không gian
2. Tăng cả độ phức tạp thời gian và không gian
3. *Giảm độ phức tạp thời gian, tăng độ phức tạp không gian*
4. Tăng độ phức tạp thời gian, giảm độ phức tạp không gian

Câu 35: Cây các lựa chọn được xây dựng để triển khai thuật toán quay lui được gọi là cây?

1. *Cây không gian trạng thái (State – space tree)*
2. Cây biểu đồ trạng thái (State – chart tree)
3. Cây nốt (Node tree)
4. Cây quay lui (Backtracking tree)

**CHUỖI CON TĂNG DÀI NHẤT**

Câu 1: Bài toán dãy con dài nhất tăng dần là bài toán tìm độ dài của dãy con từ một dãy các phần tử mảng sao cho dãy con được sắp xếp theo thứ tự tăng dần và độ dài của nó là lớn nhất. Vấn đề này có thể được giải quyết bằng cách sử dụng \_\_\_\_\_\_\_\_\_\_

1. Đệ quy
2. Quy hoạch động
3. Brute force
4. *Cả 3 đáp án trên*

Câu 2: Tìm dãy con dài nhất tăng dần trong dãy đã cho: {10, -10, 12, 9, 10, 15, 13, 14}

1. {10, 12, 15}
2. {10, 12, 13, 14}
3. {-10, 12, 13, 14}
4. *{-10, 9, 10, 13, 14}*

Câu 3: Tìm độ dài của dãy con dài nhất tăng dần trong dãy đã cho: {-10, 24, -9, 35, -21, 55, -41, 76, 84}

1. 5
2. 4
3. 3
4. *6*

Câu 4: Đối với bất kì dãy nào đã cho, luôn luôn có một dãy con tăng dần duy nhất với độ dài dài nhất.

1. Đúng
2. *Sai*

Câu 5: Số dãy con tăng dần có độ dài lớn nhất của dãy đã cho là: {10, 9, 8, 7, 6, 5}

1. 3
2. 4
3. 5
4. *6*

Câu 6: Trong triển khai thuật toán **vét cạn** để **tìm dãy con dài nhất tăng dần**, tất cả cac dãy con của một dãy đã cho đều được tìm thấy. Sau đó, tất cả các dãy con tăng dần được chọn và độ dài của dãy con dài nhất được tìm thấy. Độ phức tạp về thời gian của việc thực hiện vét cạn này là gì?

Câu 7: Hoàn thành việc triển khai quy hoạch động sau đây cho bài toán dãy con tăng dần dài nhất.

#include<stdio.h>

int longest\_inc\_sub(int \*arr, int len)

{

int i, j, tmp\_max;

int LIS[len]; *// array to store the lengths of the longest increasing subsequence*

LIS[0]=1;

for(i = 1; i < len; i++)

{

tmp\_max = 0;

for(j = 0; j < i; j++)

{

if(arr[j] < arr[i])

{

if(LIS[j] > tmp\_max)

\_\_\_\_\_\_\_\_\_\_\_;

}

}

LIS[i] = tmp\_max + 1;

}

int max = LIS[0];

for(i = 0; i < len; i++)

if(LIS[i] > max)

max = LIS[i];

return max;

}

int main()

{

int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;

int ans = longest\_inc\_sub(arr, len);

printf("%d",ans);

return 0;

}

1. *tmp\_max = LIS[j]*
2. LIS[i] = LIS[j]
3. LIS[j] = tmp\_max
4. tmp\_max = LIS[i]

Câu 8: Độ phức tạp về **thời gian** của việc triển khai quy hoạch động sau đây được sử dụng để tìm độ dài của dãy con tăng dần dài nhất là bao nhiêu?

#include<stdio.h>

int longest\_inc\_sub(int \*arr, int len)

{

int i, j, tmp\_max;

int LIS[len]; *// array to store the lengths of the longest increasing subsequence*

LIS[0]=1;

for(i = 1; i < len; i++)

{

tmp\_max = 0;

for(j = 0; j < i; j++)

{

if(arr[j] < arr[i])

{

if(LIS[j] > tmp\_max)

tmp\_max = LIS[j];

}

}

LIS[i] = tmp\_max + 1;

}

int max = LIS[0];

for(i = 0; i < len; i++)

if(LIS[i] > max)

max = LIS[i];

return max;

}

int main()

{

int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;

int ans = longest\_inc\_sub(arr, len);

printf("%d",ans);

return 0;

}

1. O(1)
2. O(n)
3. *O(n2)*
4. O(nlogn)

Câu 9: Độ phức tạp **không gian** của việc triển khai quy hoạch động sau đây được sử dụng để tìm độ dài của dãy con tăng dần dài nhất là bao nhiêu?

#include<stdio.h>

int longest\_inc\_sub(int \*arr, int len)

{

int i, j, tmp\_max;

int LIS[len]; *// array to store the lengths of the longest increasing subsequence*

LIS[0]=1;

for(i = 1; i < len; i++)

{

tmp\_max = 0;

for(j = 0; j < i; j++)

{

if(arr[j] < arr[i])

{

if(LIS[j] > tmp\_max)

tmp\_max = LIS[j];

}

}

LIS[i] = tmp\_max + 1;

}

int max = LIS[0];

for(i = 0; i < len; i++)

if(LIS[i] > max)

max = LIS[i];

return max;

}

int main()

{

int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;

int ans = longest\_inc\_sub(arr, len);

printf("%d",ans);

return 0;

}

1. O(1)
2. *O(n)*
3. O(n2)
4. O(nlogn)

Câu 10: Kết quả của chương trình sau là gì?

#include<stdio.h>

int longest\_inc\_sub(int \*arr, int len)

{

int i, j, tmp\_max;

int LIS[len]; *// array to store the lengths of the longest increasing subsequence*

LIS[0]=1;

for(i = 1; i < len; i++)

{

tmp\_max = 0;

for(j = 0; j < i; j++)

{

if(arr[j] < arr[i])

{

if(LIS[j] > tmp\_max)

tmp\_max = LIS[j];

}

}

LIS[i] = tmp\_max + 1;

}

int max = LIS[0];

for(i = 0; i < len; i++)

if(LIS[i] > max)

max = LIS[i];

return max;

}

int main()

{

int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;

int ans = longest\_inc\_sub(arr, len);

printf("%d",ans);

return 0;

}

1. 3
2. 4
3. 5
4. *6*

Câu 11: Giá trị được lưu trong LIS[5] sau khi chương trình sau được thực thi là bao nhiêu?

#include<stdio.h>

int longest\_inc\_sub(int \*arr, int len)

{

int i, j, tmp\_max;

int LIS[len]; *// array to store the lengths of the longest increasing subsequence*

LIS[0]=1;

for(i = 1; i < len; i++)

{

tmp\_max = 0;

for(j = 0; j < i; j++)

{

if(arr[j] < arr[i])

{

if(LIS[j] > tmp\_max)

tmp\_max = LIS[j];

}

}

LIS[i] = tmp\_max + 1;

}

int max = LIS[0];

for(i = 0; i < len; i++)

if(LIS[i] > max)

max = LIS[i];

return max;

}

int main()

{

int arr[] = {10,22,9,33,21,50,41,60,80}, len = 9;

int ans = longest\_inc\_sub(arr, len);

printf("%d",ans);

return 0;

}

1. 2
2. 3
3. *4*
4. 5

**KỸ THUẬT THAM LAM**

Câu 1: Bài toán knapsack phân số còn được gọi là \_\_\_\_\_\_\_\_\_\_

1. Vấn đề knapsack 0/1
2. *Vấn đề ba lô liên tục*
3. Bài toán knapsack có thể chia hết
4. Vấn đề ba lô không liên tục

Câu 2: Vấn đề knapsack phân số được giải quyết hiệu quả nhất bằng thuật toán nào sau đây?

1. Chia rẽ và chinh phục
2. Lập trình động
3. *Thuật toán tham lam*
4. Quay lui

Câu 3: Mục tiêu của vấn đề ba lô là gì?

1. *Để có được tổng giá trị lớn nhất trong ba lô*
2. Để có được tổng giá trị nhỏ nhất trong ba lô
3. Để có được trọng lượng tối đa trong ba lô
4. Để có được trọng lượng tối thiểu trong ba lô

Câu 4: Câu nào sau đây về vấn đề ba lô và ba lô phân đoạn 0/1 là chính xác?

1. Trong 0/1 các mục có vấn đề về ba lô có thể chia hết được và trong các mục ba lô phân đoạn là không thể chia nhỏ
2. Cả hai đều giống nhau
3. 0/1 knapsack được giải bằng thuật toán tham lam và knapsack phân số được giải bằng lập trình động
4. *Trong 0/1 các mục có vấn đề về ba lô là không thể chia nhỏ và trong các mục ba lô phân đoạn có thể chia hết được*

Câu 5: Độ phức tạp thời gian của bài toán knapsack phân số là \_\_\_\_\_\_\_\_\_\_\_\_

1. *O(n log n)*
2. O(n)
3. O(n2)
4. O (nW)

Câu 6: Vấn đề knapsack phân số có thể được giải quyết kịp thời O (n).

1. *Đúng*
2. Sai

Câu 7: Given items as {value,weight} pairs {{40,20},{30,10},{20,5}}. Sức chứa của ba lô = 20. Tìm đầu ra giá trị tối đa giả sử các mục có thể chia hết được.

1. *60*
2. 80
3. 100
4. 40

Câu 8. Kết quả của ba lô phân số lớn hơn hoặc bằng 0/1 ba lô.  
*a) Đúng*  
b) Sai

Câu 9. Thời gian chính thực hiện bước trong bài toán knapsack phân đoạn là \_\_\_\_\_\_\_\_\_\_\_  
a) Chia các vật phẩm thành từng phần  
b) Thêm vật phẩm vào ba lô  
*c) Sắp xếp*  
d) Lặp qua các mục đã sắp xếp

Câu 10. Given items as {value,weight} pairs {{60,20},{50,25},{20,5}}. Sức chứa của ba lô = 40. Tìm đầu ra giá trị tối đa giả sử các mục có thể chia hết và không thể điều chỉnh tương ứng.  
a) 100, 80  
b) 110, 70  
c) 130, 110  
*d) 110, 80*

**QUY HOẠCH ĐỘNG**

Câu 1. Thuộc tính nào sau đây của một vấn đề lập trình động?  
a) Cấu trúc con tối ưu  
b) Các bài toán con gối nhau  
c) Cách tiếp cận tham lam  
*d) Cả cấu trúc con tối ưu và các bài toán con gối nhau*

Câu 2. Nếu một giải pháp tối ưu có thể được tạo ra cho một vấn đề bằng cách xây dựng các giải pháp tối ưu cho các dự án con của nó, thì vấn đề đó sở hữu thuộc tính \_\_\_\_\_\_\_\_\_\_  
a) Các bài toán con gối nhau  
*b) Cấu trúc con tối ưu*  
c) Ghi nhớ  
d) Tham lam

Câu 3. Nếu một vấn đề có thể được chia thành các dự án con được tái sử dụng nhiều lần, vấn đề đó sở hữu thuộc tính \_\_\_\_\_\_\_\_\_\_\_\_\_\_ .  
*a) Các bài toán con gối nhau*  
b) Cấu trúc con tối ưu  
c) Ghi nhớ  
d) Tham lam

Câu 4. Nếu một vấn đề có thể được giải quyết bằng cách kết hợp các giải pháp tối ưu cho các vấn đề không chồng chéo, chiến lược được gọi là \_\_\_\_\_\_\_\_\_\_\_\_\_  
a) Quy hoạch động

b) Tham lam  
*c) Chia để trị*  
d) Đệ quy

Câu 5. Khi lập trình động được áp dụng cho một vấn đề, sẽ mất ít thời gian hơn nhiều so với các phương pháp khác không tận dụng được các phân đoạn chồng chéo.  
*a) Đúng*  
b) Sai

Câu 6. Một thuật toán tham lam có thể được sử dụng để giải quyết tất cả các vấn đề lập trình động.  
a) Đúng  
*b) Sai*

Câu 7. Trong lập trình động, kỹ thuật lưu trữ các giá trị được tính toán trước đó được gọi là \_\_\_\_\_\_\_\_\_\_\_  
a) Saving value property  
b) Storing value property  
*c) Memoization*  
d) Mapping

Câu 8. Khi một cách tiếp cận từ trên xuống của lập trình động được áp dụng cho một vấn đề, nó thường \_\_\_\_\_\_\_\_\_\_\_\_\_  
a) Giảm cả hai, độ phức tạp thời gian và độ phức tạp không gian  
b) Giảm độ phức tạp về thời gian và tăng độ phức tạp của không gian  
c) Tăng độ phức tạp về thời gian và giảm độ phức tạp của không gian  
d) Tăng cả hai, độ phức tạp thời gian và độ phức tạp không gian

Câu 9. Vấn đề nào sau đây KHÔNG được giải quyết bằng lập trình động?  
a) Vấn đề knapsack 0/1  
b) Bài toán nhân chuỗi ma trận  
c) Chỉnh sửa vấn đề khoảng cách  
*d) Bài toán knapsack phân số*

Câu 10. Những vấn đề nào sau đây cần được giải quyết bằng cách sử dụng lập trình động?  
a) Mergesort  
b) Tìm kiếm nhị phân  
*c) Phân đội phổ biến dài nhất*d) Quicksort

**ĐỆ QUY**

Câu 1: Đệ quy là một phương pháp trong đó lời giải của một vấn đề phụ thuộc vào \_\_\_\_\_\_\_\_\_\_\_\_  
a) Các trường hợp lớn hơn của các bài toán khác nhau  
b) Các trường hợp lớn hơn của cùng một bài toán  
*c) Các trường hợp nhỏ hơn của cùng một bài toán*  
d) Các trường hợp nhỏ hơn của các bài toán khác nhau

Giải thích: Trong đệ quy, lời giải của một bài toán phụ thuộc vào lời giải của các trường hợp nhỏ hơn của cùng một bài toán.

Câu 2, Vấn đề nào sau đây không thể được giải quyết bằng cách sử dụng đệ quy?  
 a) Giai thừa của một số  
 b) Số fibonacci thứ n  
 c) Độ dài của một chuỗi  
 *d) Các bài toán không có chữ hoa và chữ thường*

Giải thích: Các bài toán không có trường hợp cơ sở dẫn đến việc gọi đệ quy vô hạn. Nói chung, chúng ta sẽ giả sử một trường hợp cơ sở để tránh lệnh gọi đệ quy vô hạn. Các vấn đề như tìm Giai thừa của một số, số Fibonacci thứ N và Độ dài của một chuỗi có thể được giải quyết bằng cách sử dụng đệ quy.

Câu 3. Đệ quy tương tự với điều nào sau đây?  
 a) Trường hợp chuyển mạch  
 *b) Vòng lặp*  
 c) Nếu-khác

d) nếu elif khác

Giải thích: Đệ quy tương tự như một vòng lặp.

Câu 4. Trong đệ quy, điều kiện mà hàm sẽ ngừng gọi chính nó là \_\_\_\_\_\_\_\_\_\_\_\_  
 a) Trường hợp tốt nhất  
 b) Trường hợp xấu nhất  
 *c) Trường hợp cơ sở*  
 d) Không có điều kiện đó

Giải thích: Để đệ quy kết thúc tại một thời điểm nào đó, luôn phải có một điều kiện mà hàm sẽ không gọi chính nó. Điều kiện này được gọi là trường hợp cơ sở.

Câu 5. Điều gì sẽ xảy ra khi đoạn mã dưới đây được thực thi?

void my\_recursive\_function()

{

my\_recursive\_function();

}

int main()

{

my\_recursive\_function();

return 0;

}

a) Mã sẽ được thực thi thành công và không có đầu ra nào được tạo  
b) Mã sẽ được thực hiện thành công và đầu ra ngẫu nhiên sẽ được tạo  
c) Mã sẽ hiển thị lỗi thời gian biên dịch  
*d) Mã sẽ chạy trong một thời gian và dừng lại khi ngăn xếp tràn*

Giải thích: Mọi lệnh gọi hàm đều được lưu trữ trong bộ nhớ ngăn xếp. Trong trường hợp này, không có điều kiện kết thúc (trường hợp cơ sở). Vì vậy, my\_recursive\_ Chức năng () sẽ được gọi liên tục cho đến khi ngăn xếp bị tràn và không còn không gian để lưu trữ các lệnh gọi hàm. Tại thời điểm này, chương trình sẽ dừng đột ngột.

6. Đầu ra của đoạn mã sau là gì?

void my\_recursive\_function(int n)

{

if(n == 0)

return;

printf("%d ",n);

my\_recursive\_function(n-1);

}

int main()

{

my\_recursive\_function(10);

return 0;

}

a) 10  
b) 1  
c) 10 9 8… 1 0  
*d) 10 9 8… 1*

Giải thích: Chương trình in ra các số từ 10 đến 1.

7. Trường hợp cơ sở cho đoạn mã sau là gì?

void my\_recursive\_function(int n)

{

if(n == 0)

return;

printf("%d ",n);

my\_recursive\_function(n-1);

}

int main()

{

my\_recursive\_function(10);

return 0;

}

a) return  
b) printf (“% d“, n)

*c) if (n == 0)*  
d) my\_recursive\_ function (n-1)

Giải thích: Đối với trường hợp cơ sở, hàm đệ quy không được gọi. Vì vậy, “if (n == 0)” là trường hợp cơ sở.

8. Hàm đệ quy được gọi bao nhiêu lần, khi đoạn mã sau được thực thi?

void my\_recursive\_function(int n)

{

if(n == 0)

return;

printf("%d ",n);

my\_recursive\_function(n-1);

}

int main()

{

my\_recursive\_function(10);

return 0;

}

a) 9  
b) 10  
*c) 11*  
d) 12

Giải thích: Hàm đệ quy được gọi là 11 lần.

9. Đoạn mã đệ quy sau làm gì?

void my\_recursive\_function(int n)

{

if(n == 0)

return;

my\_recursive\_function(n-1);

printf("%d ",n);

}

int main()

{

my\_recursive\_function(10);

return 0;

}

a) In các số từ 10 đến 1  
b) In các số từ 10 đến 0  
*c) In các số từ 1 đến 10*  
d) In các số từ 0 đến 10

Giải thích: Đoạn mã trên in ra các số từ 1 đến 10.

10. Phát biểu nào sau đây là đúng?  
a) Đệ quy luôn tốt hơn so với lặp  
*b) Đệ quy sử dụng nhiều bộ nhớ hơn so với lặp*  
c) Đệ quy sử dụng ít bộ nhớ hơn so với lặp  
d) Lặp lại luôn tốt hơn và đơn giản hơn đệ quy

Giải thích: Phép đệ quy sử dụng nhiều bộ nhớ hơn so với phép lặp vì mỗi khi hàm đệ quy được gọi, lệnh gọi hàm được lưu trữ trong ngăn xếp.

11. Đầu ra của đoạn mã sau đây sẽ là gì?

int cnt=0;

void my\_recursive\_function(int n)

{

if(n == 0)

return;

cnt++;

my\_recursive\_function(n/10);

}

int main()

{

my\_recursive\_function(123456789);

printf("%d",cnt);

return 0;

}

a) 123456789  
b) 10  
c) 0  
*d) 9*

Giải thích: Chương trình in ra số chữ số trong dãy số 123456789 là 9.

12. Đầu ra của đoạn mã sau đây sẽ là gì?

void my\_recursive\_function(int n)

{

if(n == 0)

{

printf("False");

return;

}

if(n == 1)

{

printf("True");

return;

}

if(n%2==0)

my\_recursive\_function(n/2);

else

{

printf("False");

return;

}

}

int main()

{

my\_recursive\_function(100);

return 0;

}

a) True  
*b) False*

Giải thích: Hàm kiểm tra xem một số có phải là lũy thừa của 2. Vì 100 không phải là lũy thừa của 2 nên nó in sai.

13. Đầu ra của đoạn mã sau là gì?

int cnt = 0;

void my\_recursive\_function(char \*s, int i)

{

if(s[i] == '**\0**')

return;

if(s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u')

cnt++;

my\_recursive\_function(s,i+1);

}

int main()

{

my\_recursive\_function("thisisrecursion",0);

printf("%d",cnt);

return 0;

}

*a) 6*  
b) 9  
c) 5  
d) 10

Giải thích: Hàm đếm số nguyên âm trong một chuỗi. Trong trường hợp này, số nguyên âm là 6.

14. Đầu ra của đoạn mã sau là gì?

void my\_recursive\_function(int \*arr, int val, int idx, int len)

{

if(idx == len)

{

printf("-1");

return ;

}

if(arr[idx] == val)

{

printf("%d",idx);

return;

}

my\_recursive\_function(arr,val,idx+1,len);

}

int main()

{

int array[10] = {7, 6, 4, 3, 2, 1, 9, 5, 0, 8};

int value = 2;

int len = 10;

my\_recursive\_function(array, value, 0, len);

return 0;

}

a) 3  
*b) 4*  
c) 5  
d) 6

Giải thích: Chương trình tìm kiếm một giá trị trong mảng đã cho và in ra chỉ mục tại đó giá trị được tìm thấy. Trong trường hợp này, chương trình tìm kiếm giá trị = 2. Vì, chỉ số của 2 là 4 (lập chỉ mục dựa trên 0), chương trình sẽ in ra 4.

* 1. **KNAPSACK**

1. Bài toán Knapsack là một ví dụ của \_\_\_\_\_\_\_\_\_\_\_\_

a)Thuật toán tham lam

*b) Lập trình động 2D*

c) Lập trình động 1D

d) Chia rẽ và chinh phục

Giải thích: Bài toán Knapsack là một ví dụ về lập trình động 2D.

2.Phương pháp nào sau đây có thể được sử dụng để giải quyết vấn đề Knapsack?

a)Thuật toán vét cạn

b) Đệ quy

c) Lập trình động

*d) Vét cạn, Đệ quy và Lập trình động*

Giải thích: Vét cạn, Đệ quy và Lập trình động có thể được sử dụng để giải quyết vấn

đề ba lô.

3. Bạn được cấp một chiếc ba lô có thể mang trọng lượng tối đa là 60. Có 4 vật phẩm có

trọng lượng {20, 30, 40, 70} và các giá trị {70, 80, 90, 200}. Giá trị tối đa của các mặt hàng bạn có thể mang theo khi sử dụng ba lô là bao nhiêu?

*a) 160*

b) 200

c) 170

d) 90

Giải thích: Giá trị tối đa bạn có thể nhận được là 160. Bạn có thể đạt được điều này

bằng cách chọn các mục 1 và 3 có tổng trọng lượng là 60.

4. Bài toán nào sau đây tương đương với bài toán Knapsack 0-1?

a) Bạn được phát một chiếc túi có thể chứa tối đa trọng lượng W. Bạn được phát N món

đồ có trọng lượng là {w1, w2, w3,…., wn} và giá trị là {v1, v2, v3,… ., vn}.

Bạn có thể chia các mục thành nhiều phần nhỏ hơn. Chọn các mục theo cách mà bạn

nhận được giá trị lớn nhất

*b) Bạn đang ôn thi và bạn phải học N câu hỏi. Các câu hỏi mất {t1, t2, t3,…., Tn} thời gian (tính bằng giờ) và mang các dấu {m1, m2, m3,…., Mn}. Bạn có thể học tối đa T giờ. Bạn có thể nghiên cứu một câu hỏi hoặc để lại nó. Chọn các câu hỏi theo cách sao cho điểm của bạn được tối đa hóa*

c) Bạn nhận được vô hạn số xu có mệnh giá {v1, v2, v3,… .., vn} và một tổng S. Bạn

phải tìm số xu tối thiểu cần thiết để có tổng S

d) Bạn được tặng một vali có thể chở tối đa 15kg. Bạn nhận được 4 mục có trọng lượng là {10, 20, 15,40} và giá trị là {1, 2, 3,4}. Bạn có thể chia các mục thành nhiều phần nhỏ hơn. Chọn các mục theo cách mà bạn nhận được giá trị lớn nhất

Giải thích: Trong trường hợp này, câu hỏi được sử dụng thay cho các mục. Mỗi câu hỏi

có một số điểm giống như mỗi mục có một giá trị. Ngoài ra, mỗi câu hỏi mất một

khoảng thời gian t giống như mỗi món đồ có trọng lượng w. Bạn phải tối đa hóa

điểm trong thời gian T, bằng với việc tối đa hóa giá trị khi sử dụng một túi có

khối lượng W.

5. Độ phức tạp thời gian của thuật toán brute force được sử dụng để giải quyết vấn đề

Knapsack là bao nhiêu?

a) O(n)

b) O(n!)

c) O(2n)

d) O(n3)

Giải thích: Trong thuật toán brute force, tất cả các tập con của các mục được tìm

thấy và giá trị của mỗi tập con được tính. Tập hợp con của các mục có giá trị

lớn nhất và trọng lượng nhỏ hơn trọng lượng tối đa cho phép đưa ra câu trả lời.

Thời gian cần thiết để tính tất cả các tập con là O (2n).

6. Bài toán Knapsack 0-1 có thể được giải quyết bằng thuật toán Tham lam.

a) Đúng

*b) Sai*

Giải thích: Không thể giải quyết vấn đề Knapsack bằng thuật toán tham lam.

7. Hãy xem xét việc triển khai lập trình động sau đây của vấn đề Knapsack:

#include<stdio.h>

int find\_max(int a, int b)

{

   if(a > b)

      return a;

   return b;

}

int knapsack(int W, int \*wt, int \*val,int n)

{

  int ans[n + 1][W + 1];

  int itm,w;

  for(itm = 0; itm <= n; itm++)

      ans[itm][0] = 0;

  for(w = 0;w <= W; w++)

     ans[0][w] = 0;

  for(itm = 1; itm <= n; itm++)

  {

       for(w = 1; w <= W; w++)

       {

            if(wt[itm - 1] <= w)

               ans[itm][w] = \_\_\_\_\_\_\_\_\_\_\_\_\_\_;

            else

               ans[itm][w] = ans[itm - 1][w];

       }

  }

  return ans[n][W];

}

int main()

{

  int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;

  int ans = knapsack(W, w, v, 3);

  printf("%d",ans);

  return 0;

}

Dòng nào sau đây hoàn thành đoạn mã trên?

*a) find\_max (ans [itm - 1] [w - wt [itm - 1]] + val [itm - 1], ans [itm - 1] [w])*

b) find\_max (ans [itm - 1] [w - wt [itm - 1]], ans [itm - 1] [w])

c) ans [itm] [w] = ans [itm - 1] [w];

d) ans [itm + 1] [w] = ans [itm - 1] [w];

Giải thích: find\_max (ans [itm - 1] [w - wt [itm - 1]] + val [itm - 1], ans [itm - 1] [w]) hoàn thành đoạn mã trên.

8. Độ phức tạp về thời gian của việc thực hiện lập trình động sau đây của bài toán Knapsack với n item và trọng lượng tối đa của W?

#include<stdio.h>

int find\_max(int a, int b)

{

   if(a > b)

      return a;

   return b;

}

int knapsack(int W, int \*wt, int \*val,int n)

{

  int ans[n + 1][W + 1];

  int itm,w;

  for(itm = 0; itm <= n; itm++)

      ans[itm][0] = 0;

  for(w = 0;w <= W; w++)

     ans[0][w] = 0;

  for(itm = 1; itm <= n; itm++)

  {

       for(w = 1; w <= W; w++)

       {

            if(wt[itm - 1] <= w)

              ans[itm][w] = find\_max(ans[itm - 1][w-wt[itm - 1]]+val[itm - 1], ans[itm - 1][w]);

            else

               ans[itm][w] = ans[itm - 1][w];

       }

     }

  return ans[n][W];

}

int main()

{

  int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;

  int ans = knapsack(W, w, v, 3);

  printf("%d",ans);

  return 0;

}

a) O(n)

b) O(n + w)

*c) O(nW)*

d) O(n2)

Giải thích: Độ phức tạp về thời gian của việc triển khai lập trình động ở trên của bài toán Knapsack là O (nW).

9. Độ phức tạp không gian của việc triển khai lập trình động sau đây của bài toán Knapsack là bao nhiêu?

#include<stdio.h>

int find\_max(int a, int b)

{

   if(a > b)

      return a;

   return b;

}

int knapsack(int W, int \*wt, int \*val,int n)

{

  int ans[n + 1][W + 1];

  int itm,w;

  for(itm = 0; itm <= n; itm++)

      ans[itm][0] = 0;

  for(w = 0;w <= W; w++)

     ans[0][w] = 0;

  for(itm = 1; itm <= n; itm++)

  {

       for(w = 1; w <= W; w++)

       {

            if(wt[itm - 1] <= w)

             ans[itm][w] = find\_max(ans[itm - 1][w - wt[itm - 1]]+val[itm - 1], ans[itm - 1][w]);

               else

             ans[itm][w] = ans[itm - 1][w];

       }

  }

  return ans[n][W];

}

int main()

{

  int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;

  int ans = knapsack(W, w, v, 3);

  printf("%d",ans);

  return 0;

}

a) O(n)

b) O(n + w)

*c) O(nW)*

d) O(n2)

Giải thích: Độ phức tạp không gian của việc triển khai lập trình động ở trên của bài toán Knapsack là O (nW).

10. Đầu ra của đoạn mã sau là gì?

#include<stdio.h>

int find\_max(int a, int b)

{

   if(a > b)

     return a;

   return b;

}

int knapsack(int W, int \*wt, int \*val,int n)

{

  int ans[n + 1][W + 1];

  int itm,w;

  for(itm = 0; itm <= n; itm++)

      ans[itm][0] = 0;

  for(w = 0;w <= W; w++)

      ans[0][w] = 0;

  for(itm = 1; itm <= n; itm++)

  {

       for(w = 1; w <= W; w++)

       {

            if(wt[itm - 1] <= w)

             ans[itm][w] = find\_max(ans[itm - 1][w-wt[itm - 1]]+val[itm - 1], ans[itm - 1][w]);

          else

             ans[itm][w] = ans[itm - 1][w];

       }

  }

  return ans[n][W];

}

int main()

{

  int w[] = {10,20,30}, v[] = {60, 100, 120}, W = 50;

  int ans = knapsack(W, w, v, 3);

  printf("%d",ans);

  return 0;

}

a) 120

b) 100

c) 180

*d) 220*

Giải thích: Đầu ra của đoạn mã trên là 220.

**THÁP HÀ NỘI**

1. Mục tiêu của tháp xếp hình hà nội là gì?

*a) Để chuyển tất cả các đĩa sang một số que khác theo quy tắc sau*

b) Chia đều các đĩa cho ba que theo quy tắc sau

c) Chuyển tất cả đĩa sang que khác theo thứ tự ngẫu nhiên

d) Chia đều các đĩa cho ba que theo thứ tự ngẫu nhiên

Giải thích: Mục tiêu của bài toán tower of hanoi là di chuyển tất cả các đĩa sang một số thanh khác theo quy tắc sau-1) Mỗi ​​lần chỉ được di chuyển một đĩa. 2) Đĩa chỉ có thể được di chuyển nếu nó là đĩa trên cùng của ngăn xếp. 3) Không nên đặt đĩa lên đĩa nhỏ hơn.

2. Nội dung nào sau đây KHÔNG phải là quy tắc của tháp câu đố hà nội?

a) Không nên đặt đĩa nào trên đĩa nhỏ hơn

b) Chỉ có thể di chuyển đĩa nếu nó là đĩa trên cùng của ngăn xếp

*c) Không nên đặt đĩa nào trên đĩa lớn hơn*

d) Mỗi ​​lần chỉ có thể di chuyển một đĩa

Giải thích: Quy tắc là không đặt đĩa lên trên một đĩa nhỏ hơn. Cho phép đặt một đĩa nhỏ hơn một đĩa lớn hơn.

3. Độ phức tạp về thời gian của tháp giải bài toán hà nội sử dụng đệ quy là \_\_\_\_\_\_\_\_\_

a) O (n2 )

*b) O (2n )*

c) O (n log n)

d) O (n)

Giải thích: Độ phức tạp về thời gian của bài toán có thể được tìm ra bằng cách giải quan hệ lặp lại: T (n) = 2T (n-1) + c. Kết quả của mối quan hệ này được tìm thấy bằng 2 n . Nó có thể được giải quyết bằng cách sử dụng thay thế.

4. Phương trình lặp lại được hình thành cho bài toán tháp của Hà Nội được cho bởi \_\_\_\_\_\_\_\_\_

a) T (n) = 2T (n-1) + n

b) T (n) = 2T (n / 2) + c

*c) T (n ) = 2T (n-1) + c*

d) T (n) = 2T (n / 2) + n

Giải thích: Vì có 2 lệnh gọi đệ quy đến n-1 đĩa và một hoạt động thời gian không đổi nên quan hệ lặp lại sẽ được cho bởi T (n) = 2T (n-1) + c.

5. Số nước đi tối thiểu cần thiết để giải một bài toán tháp Hà Nội với n đĩa là \_\_\_\_\_\_\_\_\_\_

a) 2n

*b) 2n -1*

c) n2

d) n2 -1

Giải thích: Số bước di chuyển tối thiểu có thể được tính bằng cách giải quan hệ lặp lại - T (n) = 2T (n-1) + c. Ngoài ra, chúng ta có thể quan sát mô hình được hình thành bởi chuỗi số lần di chuyển 1,3,7,15… .. Dù theo cách nào thì nó cũng bằng 2 n -1.

6. Độ phức tạp không gian của lời giải đệ quy tháp câu đố Hà Nội là \_\_\_\_\_\_\_\_

a) O (1)

b) O (n)

c) O (log n)

d) O (n log n)

Giải thích: Độ phức tạp không gian của bài toán tower of hanoi có thể được tìm ra bằng cách giải quan hệ lặp lại T (n) = T (n-1) + c. Kết quả của mối quan hệ này được tìm ra là n. Nó có thể được giải quyết bằng cách sử dụng thay thế.

7. Hàm nào sau đây thể hiện đúng lời giải câu đố tower of hanoi?

một)

a)

void ToH(int n,int a,int b,int c)

{

   If(n>0)

   {

       ToH(n-1,a,c,b);

       cout<<”move a disk from” <<a<<” to”<< c;

       ToH(n-1,b,a,c);

   }

}

b)

void ToH(int n,int a,int b,int c

{

   If(n>0)

   {

       ToH(n-1,a,b,c);

       cout<<”move a disk from” <<a<<” to”<< c;

       ToH(n-1,b,a,c);

   }

}

c)

void ToH(int n,int a,int b,int c)

{

     If(n>0)

     {

         ToH(n-1,a,c,b);

         cout<<”move a disk from” <<a<<” to”<< c;

         ToH(n-1,a,b,c);

     }

}

d)

void ToH(int n,int a,int b,int c)

{

   If(n>0)

   {

       ToH(n-1,b,a,c);

       cout<<”move a disk from” <<a<<” to”<< c;

       ToH(n-1,a,c,b);

   }

}

Trả lời: a

Giải thích: Lời gọi đệ quy đầu tiên di chuyển n-1 đĩa từ a sang b bằng cách sử dụng c. Sau đó, chúng tôi di chuyển một đĩa từ a đến c. Cuối cùng, lời gọi đệ quy thứ hai di chuyển n-1 đĩa từ b sang c bằng cách sử dụng a.

8. Lời giải đệ quy của bài toán tower of hanoi là một ví dụ của thuật toán nào sau đây?

a) Lập trình động

b) Bẻ khóa

c) Thuật toán tham lam

d) Chia để trị

Giải thích: Phương pháp đệ quy sử dụng thuật toán chia và chinh phục khi chúng ta chia bài toán thành các phần nhỏ hơn, sau đó giải các phần nhỏ hơn và cuối cùng kết hợp kết quả của chúng để có được giải pháp tổng thể.

9. Bài toán Tower of hanoi có thể được giải quyết lặp đi lặp lại.

a) Đúng

b) Sai

Giải thích: Giải pháp lặp đi lặp lại cho câu đố tower of hanoi cũng tồn tại. Cách tiếp cận của nó phụ thuộc vào việc tổng số đĩa là chẵn hay lẻ.

10. Thời gian tối thiểu cần thiết để giải câu đố Tower of hanoi với 4 đĩa, giả sử một lần di chuyển mất 2 giây, sẽ là \_\_\_\_\_\_\_\_\_\_

a) 15 giây

b) 30 giây

c) 16 giây

d) 32 giây

Giải thích: Số lần di chuyển = 2 4 -1 = 16-1 = 15

Thời gian cho 1 lần di chuyển = 2 giây

Thời gian cho 15 lần di chuyển = 15 × 2 = 30 giây.

**QUAY LUI**

**1.**      Bài toán nào không giải được bằng phương pháp quay lui?

a) Bài toán n-queen

b) Bài toán tổng hợp con

c) Bài toán mạch hamiltonian

*d) Bài toán người bán hàng du lịch*

**2.**      Thuật toán backtracking được thực hiện bằng cách xây dựng một cây lựa chọn được gọi là?

*a) Cây không gian trạng thái*

b) Cây biểu đồ trạng thái

c) Cây nút

d) Cây quay lui

**3.**      Điều gì xảy ra khi thuật toán backtracking đạt được một giải pháp hoàn chỉnh?

a) Nó quay ngược trở lại gốc

*b) Nó tiếp tục tìm kiếm các giải pháp khả thi khác*

c) Nó đi qua một tuyến đường khác

d) Đệ quy đi qua cùng một tuyến đường

**4.**      Một nút được cho là \_\_\_\_\_\_\_\_\_\_\_\_ nếu nó có khả năng đạt được một giải pháp hoàn chỉnh.

a) Không hứa hẹn - Non-promising

*b) Hứa hẹn - Promising*

c) Thành công - Succeeding

d) Trước - Preceding

**5.**      Cây không gian trạng thái cho thuật toán quay lui được xây dựng theo cách nào?

*a) Tìm kiếm theo độ sâu trước tiên*

b) Tìm kiếm theo chiều rộng

c) Hai lần xung quanh cái cây

d) Người hàng xóm gần nhất trước tiên

**6.**      Những chiếc lá trong cây không gian trạng thái chỉ đại diện cho những giải pháp hoàn chỉnh.

a) đúng

*b) sai*

**7.**      Nói chung, backtracking có thể được sử dụng để giải quyết?

a) Các bài toán số

b) Tìm kiếm tận cùng

*c) Các bài toán tổ hợp*

d) Các bài toán tô màu đồ thị

**8.**      Ứng dụng nào sau đây là ứng dụng của thuật toán quay lui?

a) Tìm con đường ngắn nhất

b) Tìm số lượng hiệu quả để mua sắm

c) Ludo

*d) Trò chơi ô chữ*

**9.**      Thuật toán backtracking nhanh hơn kỹ thuật brute force

*a) đúng*

b) sai

**10.** Ngôn ngữ lập trình logic nào sau đây không dựa trên backtracking?

a) Biểu tượng

b) Lời mở đầu

c) Người lập kế hoạch

*d) Fortran*

**11.** Bài toán tìm danh sách các số nguyên trong một phạm vi xác định cho trước thỏa mãn điều kiện nào được gọi là?

a) Bài toán tổng của tập con

*b) Bài toán thỏa mãn ràng buộc*

c) Bài toán mạch Hamilton

d) Bài toán nhân viên bán hàng đi du lịch

**12.** Ai là người đặt ra thuật ngữ 'backtracking'?

*a) Lehmer*

b) Donald

c) Ross

d) Ford

**13.** \_\_\_\_\_\_\_\_\_\_\_ liệt kê một danh sách các nút đầy hứa hẹn có thể được tính toán để đưa ra các giải pháp khả thi của một vấn đề nhất định.

a) Tìm kiếm tận cùng

b) Tính bạo lực

*c) Bám sát lại*

d) Chia rẽ và chinh phục

**14.** Bài toán tìm tập hợp con các số nguyên dương mà tổng của chúng bằng một số nguyên dương đã cho được gọi là?

a) n- bài toán nữ hoàng

*b) bài toán tổng tập hợp con*

c) bài toán ba lô

d) bài toán mạch hamiltonian

**15.** Bài toán xếp n quân hậu trong bàn cờ sao cho không có hai quân hậu tấn công nhau được gọi là?

*a) Bài toán n quân hậu*

b) câu đố tám quân hậu

c) câu đố bốn quân hậu

d) Bài toán 1 quân hậu

**SẮP XẾP**

1. Xét mảng ban đầu 17 8 12 4 26. Có bao nhiêu phép so sánh cần thiết để xây dựng BST trên mảng ban đầu?

a) 5

b) 4

c) 7

*d) 10*

***Giải thích***: Mảng ban đầu là 17 8 12 4 26. BST được xây dựng trên mảng này được thể hiện trong hình dưới đây.

Để xây dựng BST, chúng tôi đi xuống cây cho đến khi đạt được một chiếc lá. Do đó, đối với mọi phần tử, chúng tôi so sánh phần tử với các nút bên trong cho đến khi chúng tôi rời khỏi và sau đó một lần nữa so sánh phần tử với cha của nó để quyết định xem nó là con phải hay con trái. Vì vậy, đối với mảng đã cho, chúng ta cần thực hiện 10 phép so sánh để xây dựng BST.

2. Trong sắp xếp cây nhị phân, trước tiên chúng ta xây dựng BST và sau đó chúng ta thực hiện \_\_\_\_\_\_\_ traversal để có được thứ tự đã sắp xếp.

*a) inorder*

b) postorder

c) preorder

d) level order

***Giải thích***: Trong cây nhị phân, sắp xếp là một thuật toán sắp xếp trong đó cây tìm kiếm nhị phân được xây dựng từ các phần tử sẽ được sắp xếp, sau đó chúng ta thực hiện duyệt theo thứ tự trên BST để lấy các phần tử theo thứ tự đã sắp xếp

3. Độ phức tạp thời gian trong trường hợp xấu nhất của sắp xếp cây nhị phân là gì?

a) O (n)

b) O (nlogn)

*c) O (n 2 )*

d) O (logn)

***Giải thích***: Đối với cây nhị phân, trường hợp xấu nhất khi BST được xây dựng là không cân bằng. BST mất cân bằng khi các yếu tố đã được sắp xếp. Vì vậy, trong trường hợp xấu nhất, cần O (n 2 ) thời gian để xây dựng BST và O (n) thời gian để đi qua cây. Do đó, độ phức tạp thời gian trong trường hợp xấu nhất là O (n 2 ) + O (n) = O (n 2 ).

4. Thủ tục insert (), được đưa ra dưới đây, xây dựng BST trên các phần tử đầu vào, là bước đầu tiên của sắp xếp cây nhị phân. Chọn đúng để điền vào điều kiện.

void insert ( Tree \* node ,  int newElement )

{

     if ( node ​​== NULL )

     {

            node = createNewNode ( ) ;

            nút -> giá trị = newElement ;

            nút -> left = NULL ;

            nút -> right = NULL ;

            trở lại ;

     }

     else  if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

     {

            insert ( node ​​-> left , newElement ) ;

     }

     else

     {

            insert ( node ​​-> right , newElement ) ;

     }

}

a) newElement> node-> value

*b) newElement <node-> value*

c) newElement == root-> value

d) newElement! = root-> value

***Giải thích***: Trong sắp xếp cây nhị phân, BST được xây dựng dựa trên các phần tử đầu vào và cây được duyệt theo thứ tự để có thứ tự đã sắp xếp. Trong khi xây dựng BST, chúng tôi đi xuống cây cho đến khi chạm được một chiếc lá. Trong khi du hành cây bình minh, chúng ta di chuyển trên cây con bên trái nếu phần tử mới nhỏ hơn nút hoặc sang bên phải nếu phần tử lớn hơn hoặc bằng nút. Vì vậy, tùy chọn đúng là giá trị newElement <node->.

5. Độ phức tạp thời gian trường hợp tốt nhất của sắp xếp cây nhị phân là gì?

a) O (n)

*b) O (nlogn)*

c) O (n 2 )

d) O (logn)

***Giải thích***: Trường hợp tốt nhất xảy ra khi BST cân bằng. Vì vậy, khi cây cân bằng, chúng ta yêu cầu O (nlogn) thời gian để xây dựng cây và O (n) thời gian để đi qua cây. Vì vậy, độ phức tạp thời gian trường hợp tốt nhất của sắp xếp cây nhị phân là O (nlogn).

6. Sắp xếp cây nhị phân là một thuật toán sắp xếp tại chỗ.

a) Đúng

*b) Sai*

***Giải thích***: Trong sắp xếp cây nhị phân, yêu cầu dành một nút cây cho mỗi phần tử mảng. Việc triển khai nó yêu cầu hai biến con trỏ cho mỗi nút. Vì vậy, nó cần thêm bộ nhớ. Độ phức tạp không gian trong trường hợp xấu nhất của sắp xếp cây nhị phân là Θ (n). Do đó, sắp xếp cây nhị phân không phải là một thuật toán sắp xếp tại chỗ.

7. Điều nào sau đây là sai?

a) Sắp xếp cây nhị phân và sắp xếp nhanh có cùng thời gian chạy

b) Sắp xếp cây nhị phân sử dụng BST làm vùng làm việc

c) Khi số phần tử cần sắp xếp càng lớn, sắp xếp cây nhị phân ngày càng hiệu quả hơn

*d) Cả sắp xếp nhanh và cây nhị phân có các thuật toán sắp xếp tại chỗ*

***Giải thích***: Sắp xếp cây nhị phân và sắp xếp nhanh có cùng thời gian chạy, tức là O (nlogn)

trong trường hợp trung bình và O (n 2 ) trong trường hợp xấu nhất. Cây nhị phân không phải là thuật toán sắp xếp tại chỗ.

8. Thuật toán sắp xếp nào sau đây có thể được coi là cải tiến đối với sắp xếp cây nhị phân?

*a) Sắp xếp đống – heap sort*

b) Sắp xếp nhanh – quick sort

c) Sắp xếp lựa chọn – selection sort

d) Sắp xếp chèn – insertion sort

***Giải thích***: Sắp xếp đống về cơ bản là cải tiến đối với sắp xếp cây nhị phân. Sắp xếp đống xây dựng một đống trên phần tử đầu vào bằng cách điều chỉnh vị trí của các phần tử trong mảng ban đầu, thay vì tạo các nút như trong sắp xếp cây nhị phân.

9. Hãy xem xét các câu lệnh sau liên quan đến sắp xếp cây nhị phân.

I. Nguyên tố có thể được thêm dần khi chúng có sẵn

II. Nó cần thêm không gian bộ nhớ

a) Câu lệnh I đúng nhưng câu lệnh II sai

b) Cả câu lệnh I và câu lệnh II đều sai

*c) Cả câu lệnh I và câu lệnh II đều đúng*

d) Câu lệnh II đúng nhưng câu lệnh I là sai

***Giải thích***: Sắp xếp cây nhị phân là sắp xếp động, nghĩa là nó trở nên hiệu quả hơn khi nhiều phần tử được thêm vào. Vì vậy, chúng tôi có thể thêm các phần tử dần dần khi chúng có sẵn. Sắp xếp cây nhị phân yêu cầu thêm không gian bộ nhớ, độ phức tạp không gian trong trường hợp xấu nhất của nó là Θ (n).

**KNAPSACK PHÂN SỐ**

1. Bài toán ba lô phân số còn được gọi là \_\_\_\_\_\_\_\_\_\_

a) Bài toán ba lô 0/1

*b) Bài toán ba lô liên tục*

c) Bài toán ba lô chia hết

d) Bài toán ba lô không liên tục

Giải thích: Bài toán ba lô phân số còn được gọi là bài toán ba lô liên tục. Knapsack phân số được giải quyết bằng cách sử dụng lập trình động.

2. Bài toán ba lô phân số được giải một cách hiệu quả nhất bằng thuật toán nào sau đây?

a) Chia để trị

b) Lập trình động

*c) Thuật toán tham lam*

d) Bẻ khóa

Giải thích: Thuật toán tham lam được sử dụng để giải quyết vấn đề này. Trước tiên, chúng tôi sắp xếp các mục theo tỷ lệ giá trị / trọng lượng của chúng và sau đó thêm mục có tỷ lệ cao nhất cho đến khi chúng tôi không thể thêm toàn bộ mục tiếp theo. Cuối cùng, chúng tôi thêm mục tiếp theo nhiều nhất có thể.

3. Mục tiêu của bài toán ba lô là gì?

*a) Để có tổng giá trị lớn nhất trong bao*

b) Để có tổng giá trị nhỏ nhất trong bao

c) Để có trọng lượng lớn nhất trong bao

d) Để có trọng lượng tối thiểu trong bao

Giải thích: Mục tiêu là để lấp đầy túi đựng của một số thể tích nhất định bằng các vật liệu khác nhau sao cho giá trị của các mặt hàng đã chọn là tối đa.

4. Phát biểu nào sau đây về bài toán cái ba lô 0/1 và ba lô phân số là đúng?

a) Trong 0/1 bài toán có ba lô là chia hết và trong ba lô phân số các mục không chia được

b) Cả hai đều giống nhau

c) Ba lô 0/1 được giải bằng cách sử dụng thuật toán tham lam và ba lô phân số được giải bằng cách sử dụng lập trình động

*d) Trong 0/1 các mục vấn đề trong túi knapsack là không thể phân chia được và trong phân số các mục có túi có thể chia được*

Giải thích: Trong bài toán ba lô phân số, chúng ta có thể đưa một phần một mục vào túi đựng trong khi trong túi 0/1, chúng ta phải bao gồm hoặc loại trừ toàn bộ mục.

5. Độ phức tạp thời gian của bài toán ba lô phân số là \_\_\_\_\_\_\_\_\_\_\_\_

*a) O (n log n)*

b) O (n)

c) O (n^ 2 )

d) O (nW)

Giải thích: Vì thời gian chính thực hiện một bước là sắp xếp nên nó xác định độ phức tạp về thời gian của mã của chúng ta. Vì vậy, độ phức tạp về thời gian sẽ là O (n log n) nếu chúng ta sử dụng sắp xếp nhanh để sắp xếp.

6. Bài toán ba lô phân số có thể được giải quyết trong thời gian O (n).

*a) Đúng*

b) Sai

Giải thích: Có thể giải quyết vấn đề trong thời gian O (n) bằng cách điều chỉnh thuật toán tìm trung phương có trọng số.

7. Cho các mục dưới dạng cặp {value, weight} {{40,20}, {30,10}, {20,5}}. Dung lượng của knapsack = 20. Tìm đầu ra giá trị lớn nhất giả sử các mục có thể chia hết.

*a) 60*

b) 80

c) 100

d) 40

Giải thích: Tỷ lệ giá trị / trọng lượng là- {2,3,4}. Vì vậy, chúng tôi bao gồm hoàn toàn các mục thứ hai và thứ ba vào trong túi. Điều này chỉ để lại 5 đơn vị thể tích cho mục đầu tiên. Vì vậy, chúng tôi bao gồm mục đầu tiên một phần.

Giá trị cuối cùng = 20 + 30 + (40/4) = 60.

8. Kết quả của ba lô phân số lớn hơn hoặc bằng 0/1 ba lô.

*a) Đúng*

b) Sai

Giải thích: Vì cái cặp phân số cho phép thêm tự do để bao gồm một phần đối tượng mà điều này không thể thực hiện được với cái cặp 0/1, do đó chúng tôi có được kết quả tốt hơn với cái cặp phân số.

9. Thời gian chính thực hiện bước trong bài toán ba lô phân số là \_\_\_\_\_\_\_\_\_\_\_

a) Chia các mục thành phân số

b) Thêm các mục vào ba lô

*c) Sắp xếp*

d) Lặp qua các mục đã sắp xếp

Giải thích: Bước thực hiện thời gian chính là sắp xếp các mặt hàng theo tỷ lệ giá trị / trọng lượng của chúng. Nó xác định độ phức tạp về thời gian của mã.

10. Cho các mục dưới dạng cặp {value, weight} {{60,20}, {50,25}, {20,5}}. Dung lượng của knapsack = 40. Tìm giá trị lớn nhất đầu ra giả sử các mục tương ứng là chia hết và không chia hết.

a) 100, 80

b) 110, 70

c) 130, 110

*d) 110, 80*

Giải thích: Giả sử các mục có thể chia hết-

Tỷ lệ giá trị / trọng lượng là {3, 2, 4}. Vì vậy, chúng tôi bao gồm toàn bộ các mục thứ ba và thứ nhất. Vì vậy, bây giờ chỉ còn lại 15 đơn vị khối lượng cho mặt hàng thứ hai. Vì vậy, chúng tôi bao gồm nó một phần.

Khối lượng cuối cùng = 20 + 60 + 50x (15/25) = 80 + 30 = 110

Giả sử các mục là không thể chia được- Trong trường hợp này chúng ta sẽ phải để lại một mục do không đủ dung lượng.

Khối lượng cuối cùng = 60 + 20 = 80.

**LẶP TÌM KIẾM NHỊ PHÂN**

1. Ưu điểm của cách tiếp cận đệ quy so với cách tiếp cận lặp là gì?

a) Sử dụng ít bộ nhớ hơn

*b) Ít mã hơn và dễ thực hiện*

c) Sử dụng nhiều bộ nhớ hơn

d) Phải viết nhiều mã hơn

***Giải thích***: Cách tiếp cận đệ quy dễ hiểu hơn và chứa ít dòng mã hơn.

3. Cho một đầu vào arr = {2,5,7,99,899}; phím = 899; Mức độ đệ quy là gì?

a) 5

b) 2

*c) 3*

d) 4

***Giải thích***: level 1: mid = 7

level 2: mid = 99

level 3: mid = 899 (đây là khóa).

4. Cho một mảng arr = {45,77,89,90,94,99,100} và key = 99; các giá trị giữa (phần tử mảng tương ứng) trong cấp đệ quy đầu tiên và thứ hai là gì?

*a) 90 và 99*

b) 90 và 94

c) 89 và 99

d) 89 và 94

***Giải thích:*** Ở cấp độ đầu tiên khóa = 90

Ở cấp độ thứ hai khóa = 99

Ở đây 90 và 99 là giá trị giữa.

5. Độ phức tạp trong trường hợp xấu nhất của tìm kiếm nhị phân sử dụng đệ quy là gì?

a) O (nlogn)

*b) O (logn)*

c) O (n)

d) O (n 2 )

***Giải thích***: Sử dụng định lý phép chia.

6. Độ phức tạp thời gian trung bình của tìm kiếm nhị phân sử dụng đệ quy là bao nhiêu?

a) O (nlogn)

*b) O (logn)*

c) O (n)

d) O (n 2 )

***Giải thích***: T (n) = T (n / 2) + 1, Sử dụng định lý chủ chia.

7. Ứng dụng nào sau đây không phải là ứng dụng của tìm kiếm nhị phân?

a) Để tìm giới hạn dưới / trên trong một chuỗi có thứ tự

b) Liên hợp các khoảng

c) Gỡ lỗi

*d) Để tìm kiếm trong danh sách không có thứ tự*

***Giải thích***: Trong tìm kiếm nhị phân, các phần tử trong danh sách nên được sắp xếp. Nó chỉ áp dụng cho danh sách đã đặt hàng. Do đó, tìm kiếm nhị phân trong danh sách không có thứ tự không phải là một ứng dụng.

9. Tìm kiếm nhị phân có thể được phân loại thành loại nào sau đây?

a) Kỹ thuật Brute Force

*b) Chia để trị*

c) Thuật toán tham lam

d) Lập trình động

***Giải thích***: Vì 'giữa' được tính cho mỗi lần lặp hoặc lần đệ quy, chúng ta sẽ chia mảng thành một nửa và sau đó cố gắng giải quyết vấn đề.

10. Cho mảng arr = {5,6,77,88,99} và key = 88; Có bao nhiêu lần lặp được thực hiện cho đến khi phần tử được tìm thấy?

a) 1

b) 3

c) 4

*d) 2*

***Giải thích***: Lặp lại 1: mid = 77; Lặp lại2: mid = 88;

11. Cho mảng arr = {45,77,89,90,94,99,100} và key = 100; Giá trị giữa (phần tử mảng tương ứng) được tạo ra trong lần lặp đầu tiên và thứ hai là gì?

*a) 90 và 99*

b) 90 và 100

c) 89 và 94

d) 94 và 99

***Giải thích***: Theo dõi đầu vào bằng mã lặp tìm kiếm nhị phân.

12. Độ phức tạp về thời gian của tìm kiếm nhị phân có lặp là bao nhiêu?

a) O (nlogn)

*b) O (logn)*

c) O (n)

d) O (n 2 )

***Giải thích***: T (n) = T (n / 2) + theta (1)

Sử dụng định lý chia và chinh, ta nhận được độ phức tạp về thời gian là O (logn).