Dynamic Programming | Set 24 (Optimal Binary Search Tree)

Given a sorted array *keys[0.. n-1]* of search keys and an array *freq[0.. n-1]* of frequency counts, where *freq[i]* is the number of searches to *keys[i]*. Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible.

Let us first define the cost of a BST. The cost of a BST node is level of that node multiplied by its frequency. Level of root is 1.

```
Example 1
Input:  keys[] = {10, 12}, freq[] = {34, 50}
There can be following two possible BSTs
        10                       12
          \                     /
           12                 10
```

```
         I                     II
Frequency of searches of 10 and 12 are 34 and 50 respectively.
The cost of tree I is 34*1 + 50*2 = 134
The cost of tree II is 50*1 + 34*2 = 118

Example 2
Input:  keys[] = {10, 12, 20}, freq[] = {34, 8, 50}
There can be following possible BSTs
    10                12                 20         10              20
      \             /    \              /             \            /
      12          10     20           12               20         10
        \                            /                /             \
         20                        10                12             12
     I               II             III             IV             V
Among all possible BSTs, cost of the fifth BST is minimum.
Cost of the fifth BST is 1*50 + 2*34 + 3*8 = 142
```

1) Optimal Substructure:
The optimal cost for freq[i..j] can be recursively calculated using following formula.
[Tex]optCost(i, j) = \sum\limits_{k=i}^j freq[k] + \min\limits_{r=i}^j [optCost(i, r-1) + optCost(r+1, j)] [/Tex]

We need to calculate ***optCost(0, n-1)*** to find the result.

The idea of above formula is simple, we one by one try all nodes as root (r varies from i to j in second term). When we make *rth* node as root, we recursively calculate optimal cost from i to r-1 and r+1 to j. We add sum of frequencies from i to j (see first term in the above formula), this is added because every search will go through root and one comparison will be done for every search.

2) Overlapping Subproblems
Following is recursive implementation that simply follows the recursive structure mentioned above.

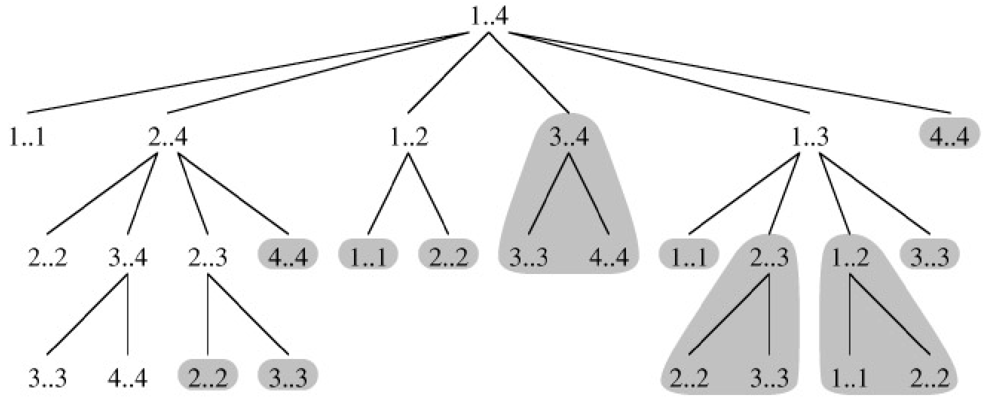
```
// A naive recursive implementation of optimal binary search tree problem
#include <stdio.h>
#include <limits.h>

// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j);

// A recursive function to calculate cost of optimal binary search tree
int optCost(int freq[], int i, int j)
{
   // Base cases
   if (j < i)      // If there are no elements in this subarray
     return 0;
   if (j == i)     // If there is one element in this subarray
     return freq[i];

   // Get sum of freq[i], freq[i+1], ... freq[j]
   int fsum = sum(freq, i, j);

   // Initialize minimum value
   int min = INT_MAX;

   // One by one consider all elements as root and recursively find cost
```

```
   // of the BST, compare the cost with min and update min if needed
   for (int r = i; r <= j; ++r)
   {
       int cost = optCost(freq, i, r-1) + optCost(freq, r+1, j);
       if (cost < min)
          min = cost;
   }

   // Return minimum value
   return min + fsum;
}

// The main function that calculates minimum cost of a Binary Search Tree.
// It mainly uses optCost() to find the optimal cost.
int optimalSearchTree(int keys[], int freq[], int n)
{
     // Here array keys[] is assumed to be sorted in increasing order.
     // If keys[] is not sorted, then add code to sort keys, and rearrange
     // freq[] accordingly.
     return optCost(freq, 0, n-1);
}

// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j)
{
    int s = 0;
    for (int k = i; k <=j; k++)
       s += freq[k];
    return s;
}

// Driver program to test above functions
int main()
{
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys)/sizeof(keys[0]);
    printf("Cost of Optimal BST is %d ", optimalSearchTree(keys, freq, n));
    return 0;
}
```

Output:

```
Cost of Optimal BST is 142
```

Time complexity of the above naive recursive approach is exponential. It should be noted that the above function computes the same subproblems again and again. We can see many subproblems being repeated in the following recursion tree for freq[1..4].



Since same suproblems are called again, this problem has Overlapping Subprolems property. So optimal BST problem has both properties (see this and this) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array cost[][] in bottom up manner.

Dynamic Programming Solution

Following is C/C++ implementation for optimal BST problem using Dynamic Programming. We use an auxiliary array cost[n][n] to store the solutions of subproblems. cost[0][n-1] will hold the final result. The challenge in implementation is, all diagonal values must be filled first, then the values which lie on the line just above the diagonal. In other words, we must first fill all cost[i][i] values, then all cost[i][i+1] values, then all cost[i][i+2] values. So how to fill the 2D array in such manner> The idea used in the implementation is same as Matrix Chain Multiplication problem, we use a variable 'L' for chain length and increment 'L', one by one. We calculate column number 'j' using the values of 'i' and 'L'.

```
// Dynamic Programming code for Optimal Binary Search Tree Problem
#include <stdio.h>
#include <limits.h>

// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j);

/* A Dynamic Programming based function that calculates minimum cost of
   a Binary Search Tree. */
int optimalSearchTree(int keys[], int freq[], int n)
{
    /* Create an auxiliary 2D matrix to store results of subproblems */
    int cost[n][n];

    /* cost[i][j] = Optimal cost of binary search tree that can be
       formed from keys[i] to keys[j].
       cost[0][n-1] will store the resultant cost */

    // For a single key, cost is equal to frequency of the key
    for (int i = 0; i < n; i++)
        cost[i][i] = freq[i];

    // Now we need to consider chains of length 2, 3, ... .
    // L is chain length.
    for (int L=2; L<=n; L++)
    {
        // i is row number in cost[][]
```

```
        for (int i=0; i<=n-L+1; i++)
        {
            // Get column number j from row number i and chain length L
            int j = i+L-1;
            cost[i][j] = INT_MAX;

            // Try making all keys in interval keys[i..j] as root
            for (int r=i; r<=j; r++)
            {
               // c = cost when keys[r] becomes root of this subtree
               int c = ((r > i)? cost[i][r-1]:0) +
                       ((r < j)? cost[r+1][j]:0) +
                       sum(freq, i, j);
               if (c < cost[i][j])
                  cost[i][j] = c;
            }
        }
    }
    return cost[0][n-1];
}

// A utility function to get sum of array elements freq[i] to freq[j]
int sum(int freq[], int i, int j)
{
    int s = 0;
    for (int k = i; k <=j; k++)
       s += freq[k];
    return s;
}

// Driver program to test above functions
int main()
{
    int keys[] = {10, 12, 20};
    int freq[] = {34, 8, 50};
    int n = sizeof(keys)/sizeof(keys[0]);
    printf("Cost of Optimal BST is %d ", optimalSearchTree(keys, freq, n));
    return 0;
}
```

Output:

```
Cost of Optimal BST is 142
```

Notes

1) The time complexity of the above solution is O(n^4). The time complexity can be easily reduced to O(n^3) by pre-calculating sum of frequencies instead of calling sum() again and again.

2) In the above solutions, we have computed optimal cost only. The solutions can be easily modified to store the structure of BSTs also. We can create another auxiliary array of size n to store the structure of tree. All we need to do is, store the chosen ‘r’ in the innermost loop.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming

Writing code in comment? Please use **ideone.com** and share the link here.

Ashish • 8 days ago

```
#include<bits stdc++.h="">

using namespace std;

int func(int i,int j,int *f)

{ int a=0 ;

for(int l = i;l<=j;l++)

a+=f[l];

return a;

}

int optimalSearchTree(int *k,int *f,int n)

{
```

int cost[n][n];

for(int i = 0;i<n;i++) cost[i][i]="f[i];" for(int="" i="n-1;i">=0;i--)

for(int j = i+1;j<n;j++) {="" cost[i][j]="INT_MAX;" for(int="" r="i;r<=j;r++){" cost[i][j]="min(cost[i][j],((i<=r-1?cost[i][r-1]:0)+" (r+1<="j?cost[r+1][j]:0)+func(i,j,f)));}" }return="" cost[0][n-1];}="" int="" main()="" {="" int="" keys[]="{10," 12,="" 20};="" int="" freq[]="{34," 8,="" 50};="" int="" n="sizeof(keys)/sizeof(keys[0]);" printf("cost="" of="" optimal="" bst="" is="" %d="" ",="" optimalsearchtree(keys,="" freq,="" n));="" return="" 0;="" }="">

• Reply • Share ›

Dean Schulze • 21 days ago

You are writing outside of your array bounds. Put this check in after computing j:

if (j >= n)
printf("L=%d, i=%d, j=%d\n", L, i, j);

1 • Reply • Share ›

sourcedelica → Dean Schulze • 19 days ago

Agreed, except you mean

int j = i+L-1;
if (j < n) {
cost[i][j] = INT_MAX;

also caught by shiwakant below.

1 • Reply • Share ›

Ashish Maheshwari → sourcedelica • 13 days ago

shiwakant caught a diff error.. both being array indices out of bounds:
1. the one you mentioned..
int j = i+L-1;
if (j < n) {
cost[i][j] = INT_MAX;
2. as mentioned by shiwakant..
// i is row number in cost[][]
for (int i = 0; i < n - L + 1; i++)
c++ code:
https://github.com/maheshwari-...

• Reply • Share ›

Mission Peace • a month ago

Check out my video on above qs
https://www.youtube.com/watch?...

1 • Reply • Share ›

pawan • 10 months ago

excellent dp problem

4 • Reply • Share ›

Lohith Ravi • a year ago

we can also put the weights in a Max Heap with key->weight & Value->index .and with a next top value of the Heap, break the array at the index recursively. to form right and left subtree.

Forming the heap and taking = nlogn and that will be the complexity of algo as well

• Reply • Share ›

NB • a year ago

Food for thought - This problem is similar to Huffman coding concept. (where you want to associate small length prefix code to symbols that occur more frequently). Huffman coding can be solved in linear time when sorted weights are given. This cant be directly be applied here because the the binary tree formed by this method wont be a binary SEARCH tree.

• Reply • Share ›

Gaurav pruthi → NB • 9 months ago

It will be BST but here we have to find the cost of every node not the leaves as the case of huffman tree

2 • Reply • Share ›

Guest • a year ago

Recursion tree for above example
opt(0,2)

root0 root1 root2
/ / \ \
opt(1,2) opt(0,0) opt(2,2) opt(0,1)
root1 root2 root0 root1
/ \ / \
/ \ / \
opt(2,2) opt(1,1) opt(1,1) opt(0,0)

• Reply • Share ›

Junyi Hu • a year ago

Very clear explanation!

• Reply • Share ›

Samar • a year ago

Following code is simple and easy to understand.

```
#include <iostream>
#include <cstdlib>

using namespace std;

struct node {
int data;
node *left;
node *right;
};

node * setroot(int n) {
node *p;
p = new node;
p->data = n;
p->left = NULL;
p->right = NULL;
```

see more

^ | v • Reply • Share ›

geekforgeeks ➦ Samar • a month ago

no way is it simpler

^ | v • Reply • Share ›

gautam • a year ago

Example that shows greedy doesn't work
keys[x,y,z,w] frequency[2, 8, 1, 9] and given that x<y<z<w so="" if="" you="" build="" the="" bst="" with="" w(maximum="" frequency)="" you="" will="" get="" total="" cost="" 34="" ,="" while="" if="" you="" build="" the="" bst="" with="" y(frequency="" 8)="" as="" a="" root="" you="" will="" get="" the="" total="" cost="" 33.="" which="" shows="" that="" greedy="" won't="" work="" here.="">

1 ^ | v • Reply • Share ›

gautam ➦ gautam • a year ago

Example that shows greedy doesn't work
keys[x,y,z,w] frequency[2,
8, 1, 9] and given that x<y<z<w. so="" choosing="" greedy="" w="" as="" a="" root="" will="" have="" cost="" 34="" but="" choosing="" the="" y="" as="" root="" have="" cost="" 33.="">

1 ^ | v • Reply • Share ›

prashant jha • a year ago

```
#include<iostream>
#define infinity 999999
```

```
#define infinity 999999
using namespace std;
struct s
{
int key;
int freq;
};
int fun(s *arr,int n)
{
if(n==0)
return 0;
if(n==1)
return (arr[0].freq);
s *left,*right;
int min=infinity;int i,j,m;
for(i=0;i<n;i++) {="" int="" r_cost="0,s1=0,s2=0,p1=0,p2=0;" for(j="0;j&lt;n;j++)" {="" if(j="=i)"
continue;="" r_cost="r_cost+arr[j].freq;" if(arr[j].key<arr[i].key)="" s1++;="" else="" s2++;="" }=""
```

see more

1 • Reply • Share ›

jv • 2 years ago

regarding

2) In the above solutions, we have computed optimal cost only. The solutions can be easily modified to store the structure of BSTs also. We can create another auxiliary array of size n to store the structure of tree. All we need to do is, store the chosen 'r' in the innermost loop.

i think this needs any array of NxN as we need to store r at every level and back trace at the end.

Can you please explain how this can be done with array of size N only.

4 • Reply • Share ›

shiwakant.bharti • 2 years ago

Awesome post with amazing code and comments. Thank you admin!

Meanwhile there is a minor bug which leads to Exception in Java java.lang.ArrayIndexOutOfBoundsException: 3

The issue is in this code we have not created and array of size n+1. Also this issue is not visible in C/C++ as there is no bound checking support there. Fixed code below.

```
            // i is row number in cost[][]
           for (int i = 0; i < n - L + 1; i++) {
```

8 • Reply • Share ›

Ashish Maheshwari shiwakant.bharti • 13 days ago

i was thinking the exact same thing..
java is nice in this context that it throws an exception for array index out of bounds :)

• Reply • Share ›

Born Actor • 2 years ago

```
 //to print the minimum cost and inorder traversal of teh tree formed



#include <iostream>
#include<string>
#include<sstream>
#include<iomanip>
#include <stdio.h>
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;
int keys[50];
int n;
int frequencies[50];
class node

{
```

see more

1 • Reply • Share ›

atul • 2 years ago

In the given example the output should be :134

(50*1)+(34*2)+(8*2)=134

i.e following tree :_

```
         20
        /  \
       10  12
```

but it seems to me the following details about the question is missing which would make this implementation correct:-

if ith value is considered as the root then 0 to i-1 elements lies on the left side of the ith node and i+1 to nth element lies on the right side of ith node.

considering the fact that this tree is also valid

see more

^ | v • Reply • Share ›

Dheeraj ➦ atul • 2 years ago

Did u read the question? You don't even need to read the complete question. See title, it says Binary **Search** Tree. The example that you have given above is not a binary search tree.

2 ^ | v • Reply • Share ›

rajat rastogi • 2 years ago

Next problem will be print structure of optimal binary search tree in O(nlogn) time?

```
/* Paste your code here (You may delete these lines if not writing code) */
```

^ | v • Reply • Share ›

rajat rastogi ➦ rajat rastogi • 2 years ago

Correction, problem statment should be...Write algorithm to find Optimal Binary Search Tree in O(nlogn) time.

^ | v • Reply • Share ›

yashraj • 2 years ago

"We add sum of frequencies from i to j (see first term in the above formula), this is added because every search will go through root and one comparison will be done for every search."

what does this mean.. i could not get. can you please explain in detail

1 ^ | v • Reply • Share ›

jv ➦ yashraj • 2 years ago

lets say
keys[]={1,2,3}
frequency[]={3,10,5}
when you are considering 2 as root and finding the min cost
mincostof(1, with size 1)+mincostof(3, with size 1) + costof 2(=10) + since now element 1 and elment 3 is moved to second level you need to add there cost also(=3+5)

so if see the total will become

cost of all keys + minat 1 + mniat 3

hope it clears this now

1 • Reply • Share ›

tejas • 2 years ago

Please change the condition to (i<=n-L).

1 • Reply • Share ›

rocker • 2 years ago

Why is the recurrence relation just considers (i,r-1) and (r+1, j) to be the subproblems. The rest of the elements apart from root can be part of any of the subtrees.
The one defined here signifies that, elements from (i to r-1) or (r+1 to j) constitute one subtree which need not be the case.

• Reply • Share ›

Unknown → rocker • 2 years ago

Because the tree here is a BST. So it is obvious that we consider the smaller elements to the left subtree of root. Same statement follows for the right.

• Reply • Share ›

sreeram → Unknown • 2 years ago

But that requires frequencies to be sorted right ?

or am i missing something ,,,

• Reply • Share ›

sreeram → sreeram • 2 years ago

Oh no not frequencies ...keys ...i think here the assumption is that keys are sorted ...

• Reply • Share ›

ammy • 2 years ago

Finding optimal BST can be done in o(n^2) using knuth's algorithm to find roots of optimal sub-trees..root(i,j-1)<=root(i,j)<=root(i+1,j)...isn&#039t it??

1 • Reply • Share ›

Piyush • 2 years ago

In example 2, tree structure II has the least code, not V

• Reply • Share ›

Kartik → Piyush • 2 years ago

Cost of IInd tree = 1*8 + 34*2 + 50*2 = 176

Which is more than cost of Vth tree.

• Reply • Share ›

Arvind B R • 2 years ago

The description is misleading "Construct a binary search tree of all keys such that the total cost of all the searches is as small as possible." Here you have not constructed any binary search tree ,you have just found the minimum cost.

1 • Reply • Share ›

Kartik → Arvind B R • 2 years ago

The main algorithm for the given problem lies in finding out the total cost. The programs can be easily augmented to construct the tree as well. We will soon add code to construct the tree also.

• Reply • Share ›

BSTlover → Kartik • 2 years ago

Can you share that solutions pls?

6 • Reply • Share ›

op • 2 years ago

good explanation

...Greedy algorithm....
make the most frequent element root, do the same for left and right subtrees

• Reply • Share ›

kapser → op • 2 years ago

As OP said,

Why haven't you used Greedy algorithm ? (sort the frequencies by descending order and build tree based on the keys).

• Reply • Share ›

Mathan Kumar → kapser • 2 years ago

I think the question is to find optimal binary "search" tree... Not an ordinary binary tree..

```
/* Paste your code here (You may delete these lines if not writing code) */
```

1 • Reply • Share ›

Kartik → kapser • 2 years ago

Consider the following example
keys[] = {10, 12, 20};
freq[] = {100, 99, 98};
Among the 5 possible BSTs, following BST has the minimum cost

Among the 5 possible BSTs. following BST has the minimum cost

```
    12
  /  \
10    20
Cost = 99*1 + 100*2 + 98*2 = 495
```

But according to Greedy, we should get following BST

```
     10
      \
       12
         \
          20

Cost = 100*1 + 99*2 + 98*3 =  595
```

The cost from Greedy approach is much more than the optimal cost.

2 • Reply • Share ›

Guru ➦ Kartik • 2 years ago

Greeedy works too. The above example has distributed the keys in a skewed fashion. We can always try to build a complete tree.

```
/* Paste your code here (You may delete these lines if not writing co
```

• Reply • Share ›

Kartik ➦ kapser • 2 years ago

Geedy algorithm doesn't always give the optimal solution. We will post the examples soon.

3 • Reply • Share ›

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding "extern" keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Follow @GeeksforGeeks

- # Recent Comments

  - lt_k

    i need help for coding this function in java...

    Java Programming Language · 1 hour ago

  - Piyush

    What is the purpose of else if (recStack[*i])...

    Detect Cycle in a Directed Graph · 1 hour ago

  - Andy Toh

    My compile-time solution, which agrees with the...

    Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

  - lucy

    because we first fill zero in first col and...

    Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

  - lucy

    @GeeksforGeeks i don't n know what is this long...

    Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

  - manish

    Because TAN is not a subsequence of RANT. ANT...

    Given two strings, find if first string is a subsequence of second · 3 hours ago

-