

TERMWORK 9.

PROBLEM STATEMENT

Find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d .

For example

If $S = \{1, 2, 5, 6, 8\}$ and $d = 9$, there are two solutions $\{1, 2, 6\}$ and $\{1, 8\}$.

A suitable message to be displayed if given problem instance does not have a solution.

OBJECTIVE

- To introduce the concept of backtracking
- Present the work of subset-sum problem.
- To find the subset as a solution for a given positive integer.
- Analyze the Algorithm complexity.

THEORY

In Subset-Sum problem, we find a subset of a given set $S = \{s_1, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d .

Example.

$S = \{1, 2, 5, 6, 8\}$ and $d = 9$ solutions are $\{1, 8\}$ and $\{1, 2, 6\}$.
Some of the instances may have no solutions.

It is convenient to sort element in increasing order.
The state space tree can be constructed as a binary tree.
A path from the root to a node on i^{th} level of tree indicates which of first i numbers have been included in subsets represented by that node. We record the value of s , the sum of these numbers, in node if s is equal to d , we have the solution to problem.

ALGORITHM

// Gives a template on generic backtracking algorithm.

// Input : $x[1 \dots i]$

// Output : All tuples representing solutions.

If $x[1 \dots i]$ is a solution write $x[1 \dots i]$

else

for each element $x \in S_{i+1}$ consistent with $x[1 \dots i]$
and constraints do

• $x[i+1] \leftarrow x$

Backtracking ($x[1 \dots i+1]$)

PROGRAM

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define MAX 50
```

```
#define TRUE 1
```

```
#define FALSE 0
```

```
int inc[MAX], w[MAX], sum, n;
```

```
int promising (int i, int wt, int total) {  
    return (((wt + total) >= sum) && ((wt == sum) ||  
        (wt + w[i+1] <= sum)));  
}
```

```
void sumset (int i, int wt, int total) {
```

```
    int j;
```

```
    if (promising (i, wt, total))  
    {
```

```
        if (wt == sum)  
        {
```

```
            printf ("\\n { \\t");
```

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

```
                if (inc[j]) printf ("%d", w[j]);
```

```
            printf ("\\n { \\t");
```

```
        }
```

```
    else {
```

```
        inc[i+1] = TRUE;
```

```
        sumset (i+1, wt + w[i+1], total + w[i+1]);
```

```
        inc[i+1] = FALSE;
```

```
        sumset (i+1, wt, total - w[i+1]);
```

```
    }
```

```
}
```

```
}
```

```

int main (int argc, char * argv[]) {
    int i, j, n, temp, total=0;
    printf ("\n Enter how many numbers: \n");
    scanf ("%d", &n);
    printf ("\n Enter %d numbers to set: \n", n);
    for (i=0; i<n; i++)
        for (j=0; j<n-1; j++)
            if (w[j] > w[j+1])
            {
                temp = w[j];
                w[j] = w[j+1];
                w[j+1] = temp;
            }

    printf ("\n The given %d numbers in ascending order: \n", n);
    for (i=0; i<n; i++)
        printf ("%d", w[i]);
    if ((total < sum))
        printf ("\n Subset construction is not possible");
    else
    {
        for (i=0; i<n; i++)
            inc[i] = 0;
        printf ("\n The solution using backtracking is: \n");
        subset (-1, 0, total);
    }

    system ("PAUSE");
    return 0;
}

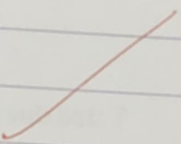
```

REFERENCES

- K. Bertman, J. Paul, "Algorithms" Cengage Learning.
- Thomas, H. (Charles) Leiserson, "Introduction to Algorithms", PHI 2nd edition.

CONCLUSION

In this teamwork we learnt the concept of Backtracking and also learned how to apply backtracking in the sum of sub-set problem.



TERMWORK 09

Name : REECHA TOPINKATTI

USN : 2GI20CS108

DATE : 16-8-22

OUTPUT:

Enter how many numbers: 6

Enter 6 numbers to the set:

1 3 5 6 4 2

Input the sum value to create sub set: 7

The given 6 numbers in ascending order:

1 2 3 4 5 6

The solution using backtracking is:

SOLUTION:1 { 1 2 4 }

SOLUTION:2 { 1 6 }

SOLUTION:3 { 2 5 }

SOLUTION:4 { 3 4 }

Press any key to continue . . .

✓
By
22/08/2022
(10)

TERMWORK - 10

PROBLEM

Implement N Queen's problem using Back Tracking.

OBJECTIVE

- To implement solution to place n queens on n by n chessboard as that no 2 queens attack each other

THEORY

In this method main aim is to place n queens on $n \times n$ chessboard such that no two queens attack each other by being in same row or in same column or on same diagonal. For $n=1$ there is a trivial solution and for $n=2$ and $n=3$ there is no solution. So in this termwork we consider $n=4$.

ALGORITHM

Procedure queen (i , $index$, n)

var $j \leftarrow index$

Begin

if promising (i) then

if $i = n$ then

write ($col[i]$ through $col[n]$)

else

for $j \leftarrow 1$ to n do

$col[i+1] \leftarrow j$

queens ($i+1$, n)

end

end

end

end

function promising ($i = index$): boolean;

var $k \leftarrow index$

begin

$k \leftarrow 1$

promising \leftarrow true;

while $k < i$ and promising do

if $col[i] = col[k]$ or $abs(col[i] - col[k]) = i - k$ then

promising \leftarrow false.

end

$k \leftarrow k + 1$

end

end

PROGRAM

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <math.h>
```

```
int a[30], count = 0;
```

```
int place (int pos)
```

```
{  
    int i;
```

```
    for (i = 1; i < pos; i++)  
    {
```

```
        if ((a[i] == a[pos]) || (abs(a[i] - a[pos]) == abs(i - pos)))
```

```
            return 0;
```

```
    }
```

```
    return 1;
```

```
}
```

```
void printsol (int n)
```

```
{
```

```
    int i, j;
```

```
    count++;
```

```
    printf ("\n\n Solution #%d \n\n", count);
```

```
    for (i = 1; i <= n; i++) {
```

```
        for (j = 1; j <= n; j++) {
```

```
            if (a[i] == j)
```

```
                printf ("Q\t");
```

```
            else
```

```
                printf ("X\t");
```

```
        }
```

```
    }  
    printf ("\n");
```

```
}
```

```
}
```



```
void queen (int n)
```

```
{
```

```
    int k=1;
```

```
    a[k]=0;
```

```
    while (k!=0)
```

```
{
```

```
    a[k]=a[k]+1;
```

```
    while (a[k] <= n && ! place(k))
```

```
        a[k]++;
```

```
    if (a[k] <= n)
```

```
{
```

```
    if (k==n)
```

```
        printsol(n);
```

```
    else {
```

```
        k++;
```

```
        a[k]=0;
```

```
    }
```

```
}
```

```
else
```

```
    k--;
```

```
}
```

```
}
```

```
void main() {
```

```
    int n;
```

```
    printf("Enter the number of queens: \n");
```

```
    scanf("%d", &n);
```

```
    queen(n);
```

```
    printf("Total Number of solutions = %d", count);
```

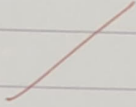
```
}
```

REFERENCES

- K. Berman J Paul 'Algorithm' Cengage learning.
- Thomas H C, Charles 'Introduction to Algorithm' PHL 2nd edition

CONCLUSIONS

In this termwork we have successfully solved the n queens problem and implemented with code.



TERMWORK 10

N QUEENS

NAME : REECHA TOPINKATTI

USN : 2GI20CS108

DATE :2-8-22

Enter the number of queens

4

Solution #1

```
*      *      *      *
*      Q      *      *
*      *      *      Q
Q      *      *      *
*      *      Q      *
```

Solution #2

```
*      *      *      *
*      *      Q      *
Q      *      *      *
*      *      *      Q
*      Q      *      *
```

Total number of sol=2

Process returned 0 (0x0) execution time : 2.257 s

Press any key to continue.

Bsg
22/10/22

(10)