**CHAPTER 1**

**INTRODUCTION**

* 1. **Tower of Hanoi**

The **Tower of Hanoi** (also called the **Tower of Brahma** or **Lucas' Tower** [[1]](https://en.wikipedia.org/wiki/Tower_of_Hanoi#cite_note-1) and sometimes pluralized) is a [mathematical game](https://en.wikipedia.org/wiki/Mathematical_game)r [puzzle](https://en.wikipedia.org/wiki/Puzzle). It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a [conical](https://en.wikipedia.org/wiki/Cone) shape

The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No larger disk may be placed on top of a smaller disk.



**Fig 1.1 Towers of Hanoi Disk Moving**

**1.2 Time Complexity**

The time needed by an algorithm expressed as a function of the size of a problem is called the time complexity of the algorithm. The time complexity of a program is the amount of computer time it needs to run to completion.

**1.3 Space Complexity**

The space complexity of a program is the amount of memory it needs to run to completion. The space need by a program has the following components:

* 1. **Environment stack space**

The environment stack is used to save information needed to resume execution of partially completed functions.

* 1. **Some Mathematical Problems Using C**
     1. **Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.**

1. **Spanning tree of a connected graph G**

A connected acyclic sub graph (tree) of *G* that includes all of *G*’s vertices.

1. **Minimum Spanning Tree** **of a weighted, connected graph G**

A spanning tree of *G* of minimum total weight.

* Kruskal’s algorithm finds the minimum spanning tree for a weighted connected graph G=(V,E) to get an acyclic subgraph with |V|-1 edges for which the sum of edge weights is the smallest.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 Towers of Hanoi**

The **Tower of Hanoi** (also called the **Tower of Brahma** or **Lucas' Tower** [[1]](https://en.wikipedia.org/wiki/Tower_of_Hanoi#cite_note-1) and sometimes pluralized) is a [mathematical game](https://en.wikipedia.org/wiki/Mathematical_game)r [puzzle](https://en.wikipedia.org/wiki/Puzzle). It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a [conical](https://en.wikipedia.org/wiki/Cone) shape.

**2.1.1 Iterative solution**

A simple solution for the toy puzzle is to alternate moves between the smallest piece and a non-smallest piece. When moving the smallest piece, always move it to the next position in the same direction (to the right if the starting number of pieces is even, to the left if the starting number of pieces is odd). If there is no tower position in the chosen direction, move the piece to the opposite end, but then continue to move in the correct direction. For example, if you started with three pieces, you would move the smallest piece to the opposite end, then continue in the left direction after that. When the turn is to move the non-smallest piece, there is only one legal move. Doing this will complete the puzzle in the fewest moves.

### 2.1.2 Recursive solution

### Illustration of recursive solution for the Towers of Hanoi puzzle with 4 disks

### The key to solving a problem recursively is to recognize that it can be broken down into a collection of smaller sub-problems, to each of which that same general solving procedure thatwe are seeking applies, and the total solution is then found in some simple way from those sub-problems' solutions. Each of thus created sub-problems being "smaller" guarantees that the base case(s) will eventually be reached. Thence, for the Towers of Hanoi:

* label the pegs A, B, C,
* let *n* be the total number of disks,
* Number the disks from 1 (smallest, topmost) to n (largest, bottom-most).

Assuming all n disks are distributed in valid arrangements among the pegs; assuming there are m top disks on a source peg, and all the rest of the disks are larger than m, so they can be safely ignored; to move m disks from a source peg to a target peg using a spare peg, without violating the rules:

1. Move m − 1 disks from the **source** to the **spare** peg, by the same general solvingprocedure. Rules are not violated, by assumption. This leaves the disk *m* as a top disk on the source peg.
2. Move the disk *m* from the **source** to the **target** peg, which is guaranteed to be a valid move, by the assumptions — asimplestep.
3. Move the m − 1 disks that we have just placed on the spare, from the **spare** to the **target** peg by the same general solving procedure, so they are placed on top of the disk m without violating the rules.
4. The base case being to move *0* disks (in steps 1 and 3), that is, do nothing – which obviously doesn't violate the rules.

The full Tower of Hanoi solution then consists of moving *n* disks from the source peg A to the target peg C, using B as the spare peg.

This approach can be given a rigorous mathematical proof with [mathematical induction](https://en.wikipedia.org/wiki/Mathematical_induction) and is often used as an example of recursion when teaching programming.

**2.1.3 Recursive Equation**

T (n) = 2T (n-1) + 1 ——-equation-1

Solving it by Back-Substitution:

T (n-1) = 2T (n-2) + 1 ———–equation-2

T (n-2) = 2T (n-3) + 1 ———–equation-3

Put value of T (n-2) in equation–2 with help of equation-3

T (n-1) = 2 (2T (n-3) + 1) + 1 ——equation-4

Put value of T (n-1) in equation-1 with help of equation-4

T (n) = 2(2(2T (n-3) + 1) + 1) + 1

T (n) = 2^3 T (n-3) + 2^2 + + 2^1 + 1

After Generalization:

T(n)= 2^k T(n-k) + 2^{(k-1)} + + 2^{(k-2)} + ............ +2^2 + + 2^1 + 1

Base condition T(0) == 1

n – k = 0

n = k;

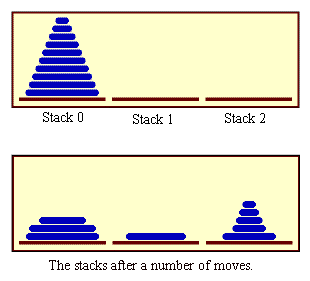
put, k = n

T(n) = 2^n T(0) + 2^{(n-1)} + + 2^{(n-2)} + ............ +2^2 + + 2^1 + 1

It is GP series, and sum is 2^ {(n+1)} - 1

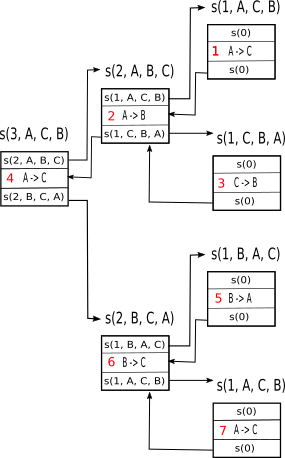
T(n)= O( 2^{(n+1)} - 1), or you can say O(2^n) which is exponential

**2.1.4 Tower of Hanoi Using Stack**

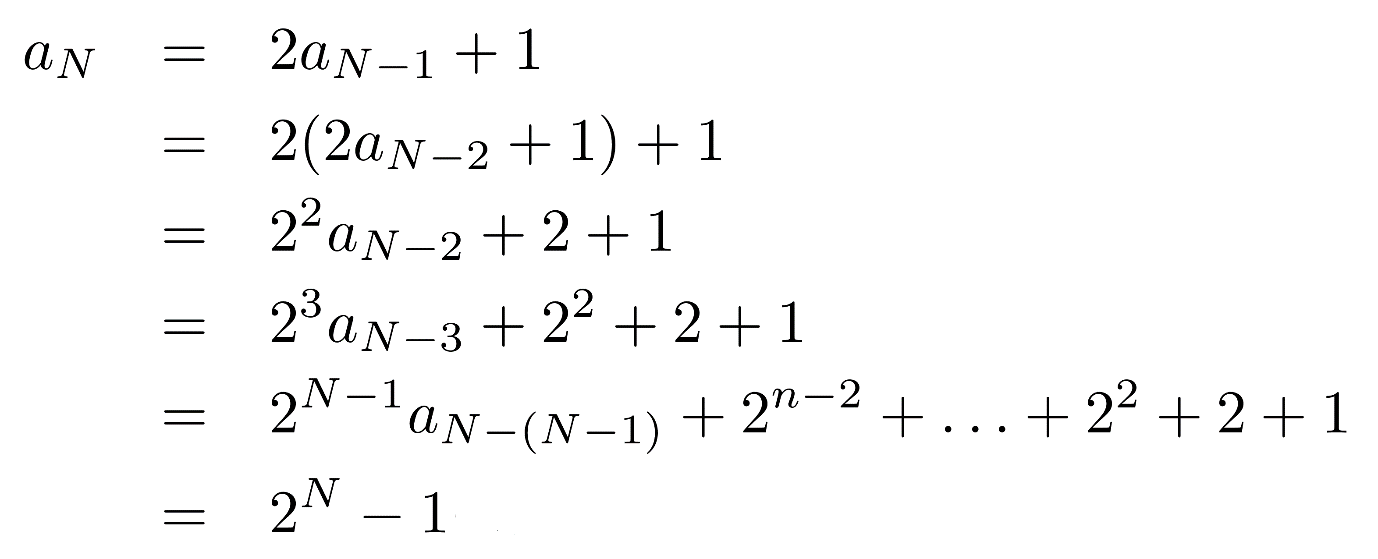
****

**Fig 2.1 Tower of Hanoi Using Stack**

**2.1.5 Working of Tower of Hanoi**

****

**Fig 2.2 Working of Tower of Hanoi**

**2.1.6 Tower of Hanoi Formula to calculate how many number of disk moves**

**2.1.7 Algorithm for Tower of Hanoi**

START

Procedure Hanoi (disk, source, dest, aux)

IF disk == 1, THEN

move disk from source to dest

ELSE

Hanoi (disk - 1, source, aux, dest) // Step 1

move disk from source to dest // Step 2

Hanoi (disk - 1, aux, dest, source) // Step 3

END IF

END Procedure

STOP

**2.1.8 Time Complexity**

Let the time required for n disks is T (n).

There are 2 recursive call for n-1 disks and one constant time operation to move a disk from ‘from’ peg to ‘to’ peg. Let it be k1.

Therefore,

T (n) = 2 T (n-1) + k1

T (0) = k2 , a constant.

T (1) = 2 k2 + k1

T (2) = 4 k2 + 2k1 + k1

T (2) = 8 k2 + 4k1 + 2k1 + k1

Coefficient of k1 =2n

Coefficient of k2 =2n-1

Time complexity is O(2n) or O(an) where a is a constant greater than 1.

So it has exponential time complexity. For single increase in problem size the time required is double the previous one. This is computationally very expensive. Most of the recursive programs takes exponential time that is why it is very hard to write them iteratively.

**2.1.9 Space Complexity**

Space for parameter for each call is independent of n i.e., constant. Let it be k .

When we do the 2nd recursive call 1st recursive call is over. So, we can reuse the space of 1st call for 2nd call. Hence,

T(n) = T(n-1) + k

T(0) = k

T(1) = 2k

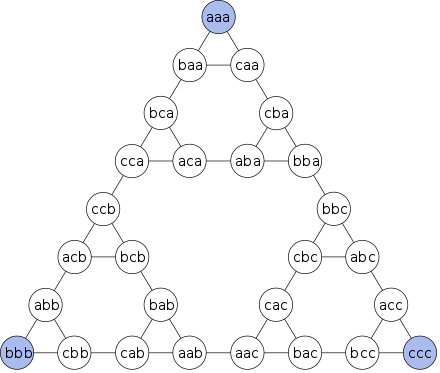
T(2) = 3k

T(3) = 4k

So the space complexity is O(n).

Here time complexity is exponential but space complexity is linear. Often there is a trade-off between time and space complexity.

**2.1.10 Graph Representation**

****

**Fig 2.3 Graph Representation**

**2.2 Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.**

A greedy algorithm makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. The choice made at each step must be:

* Feasible: Satisfy the problem’s constraints
* locally optimal: Be the best local choice among all feasible choices
* Irrevocable: Once made, the choice can’t be changed on subsequent steps.

**2.2.1 Spanning tree of a connected graph G**

A connected acyclic sub graph (tree) of Gthat includes all of G’s vertices.

**2.2.2** **Minimum Spanning Tree** **of a weighted, connected graph G**

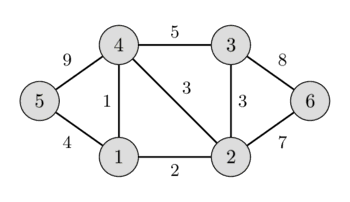
A spanning tree of *G* of minimum total weight.

* Kruskal’s algorithm finds the minimum spanning tree for a weighted connected graph G=(V,E) to get an acyclic subgraph with |V|-1 edges for which the sum of edge weights is the smallest.
* Consequently the algorithm constructs the minimum spanning tree as an expanding sequence of subgraphs, which are always acyclic but are not necessarily connected on the intermediate stages of algorithm.
* The algorithm begins by sorting the graph’s edges in non-decreasing order of their weights. Then starting with the empty subgraph, it scans the sorted list adding the next edge on the list to the current sub graph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

**2.2.3 Formulation of the Problem:**

Let directed weight graph G = (V, E), where V is the set of n vertices; E is the set of m edges and W is the set of weights associated from vi to vj of the graph. Let’s assume, eij = the edge from vertices vi to vj. wij = The weight of the edge eij. From the following rule the weight matrix W of the G is constructed: If there is an edge from vi to vj presented in G then Set, W[i,j] = wij else Set, W[i,j] = 0 Figure 1 shows the directed weight graph G1

**Example:**

****

**2.2.4 Complexity Analysis of the Algorithm:**

The nxn weight matrix W is the input to the algorithm, where n is the number of the vertices in the graph. It selects n-1 elements to produce T from n2 elements. Basically it searches n2-1 elements to produce T. Hence the complexity of this algorithm is O(n2).

**2.2.5 Best Case Analysis**

The situation for the best case is, when, only the elements in first row or first column are available for mark. Other rows or columns are marked as 0. In this situation the complexity for this algorithm will be O(n). When constructing the Directed Minimum Cost Spanning tree with this type of element the algorithm will consume O(n) time. Hence the best case will be O(n) and instead of n, if m replaced, the best case time complexity will be in order of O(m), where m = n-1

**2.2.6 Worst Case Analysis**

The situation for the worst case is, when all the elements in matrix W is consider for searching and marking suitable edges. In this situation the complexity will be O(n2). When constructing the Directed Minimum Cost Spanning tree with consider all the elements of matrix W, the algorithm will consume O(n2) time. Hence the best case will be O(n2) and instead of n, if m replaced, the worst

case time complexity will be in order of O(m2), where m = n-1. If n is large, the complexity O(m) and O(m2) are better than O(n) and O(n2), where m = n-1. Include conclusion

**2.3 Find the Binomial Co-efficient using Dynamic Programming.**

**2.3.1 Theorem Statement**

In [elementary algebra](https://en.wikipedia.org/wiki/Elementary_algebra), the **binomial theorem** (or **binomial expansion**) describes the algebraic expansion of [powers](https://en.wikipedia.org/wiki/Exponentiation) of a [binomial](https://en.wikipedia.org/wiki/Binomial_(polynomial)). According to the theorem, it is possible to expand the polynomial (*x* + *y*)n into a [sum](https://en.wikipedia.org/wiki/Summation) involving terms of the form *a*xb yc, where the exponents *b* and *c* are [nonnegative integers](https://en.wikipedia.org/wiki/Nonnegative_integer) with b + c = n, and the [coefficient](https://en.wikipedia.org/wiki/Coefficient) *a* of each term is a specific [positive integer](https://en.wikipedia.org/wiki/Positive_integer) depending on n and b.

For example (for n = 4), (x+y)4=x4+4x3y+6x2y2+4xy3+y4

**2.3.2 FORMULA**

The formal expression of the Binomial Theorem is as follows:

(a + b)^n = sum[k=0,n][(n over k)a^(n-k)b^k]

The parenthetical bit above has these equivalents:

(n over k) = nCk = n!/[(n-k)!k!]

Example

**a+b** is a binomial (the two terms are **a** and **b**)

Let us multiply **a+b** by itself using [Polynomial Multiplication](https://www.mathsisfun.com/algebra/polynomials-multiplying.html) :

(a+b)(a+b) = **a2 + 2ab + b2**

Now take that result and multiply by **a+b** again:

(a2 + 2ab + b2)(a+b) = **a3 + 3a2b + 3ab2 + b3**

And again:

(a3 + 3a2b + 3ab2 + b3)(a+b) = **a4 + 4a3b + 6a2b2 + 4ab3 + b4**

**2.3.3 ALGORITHM Binomial (*n*,*k*)**

Computes *C*(*n*, *k*) by the dynamic programming algorithm

//Input: A pair of nonnegative integers n ≥ k ≥ 0

//Output: The value of *C*(*n* ,*k*)

for *i🡨*0 to n do

for *j🡨*⇓0 to min (*i* ,*k*) do

if *j* = 0 or *j* = *k*

C [*i* , *j*]🡨 1

else *C* [*i* , *j*] 🡨 *C*[*i*-1, *j*-1] + *C*[*i*-1, *j*]

return C [n, k]

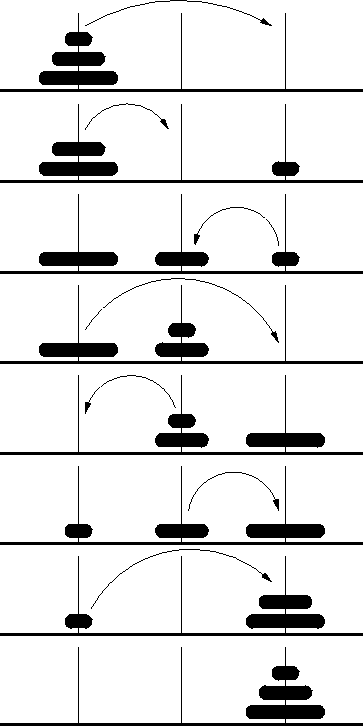
Number of Additions=A(n,k)

**CHAPTER 3**

**PROJECT DESCRIPTION**

**3.1 Tower of Hanoi**

The **Tower of Hanoi** (also called the **Tower of Brahma** or **Lucas' Tower** [[1]](https://en.wikipedia.org/wiki/Tower_of_Hanoi#cite_note-1) and sometimes pluralized) is a [mathematical game](https://en.wikipedia.org/wiki/Mathematical_game)r [puzzle](https://en.wikipedia.org/wiki/Puzzle). It consists of three rods and a number of disks of different sizes, which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a [conical](https://en.wikipedia.org/wiki/Cone) shape

****

**3.2 Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm**

Let directed weight graph G = (V, E), where V is the set of n vertices; E is the set of m edges and W is the set of weights associated from vi to vj of the graph. Let’s assume, eij = the edge from vertices vi to vj. wij = The weight of the edge eij. From the following rule the weight matrix W of the G is constructed: If there is an edge from vi to vj presented in G then Set, W[i,j] = wij else Set, W[i,j] = 0 Figure 1 shows the directed weight graph G1

**3.3 BINOMIAL THEOREM**

In [elementary algebra](https://en.wikipedia.org/wiki/Elementary_algebra), the **binomial theorem** (or **binomial expansion**) describes the algebraic expansion of [powers](https://en.wikipedia.org/wiki/Exponentiation) of a [binomial](https://en.wikipedia.org/wiki/Binomial_(polynomial)). According to the theorem, it is possible to expand the polynomial (*x* + *y*)n into a [sum](https://en.wikipedia.org/wiki/Summation) involving terms of the form *a*xb yc, where the exponents *b* and *c* are [nonnegative integers](https://en.wikipedia.org/wiki/Nonnegative_integer) with b + c = n, and the [coefficient](https://en.wikipedia.org/wiki/Coefficient) *a* of each term is a specific [positive integer](https://en.wikipedia.org/wiki/Positive_integer) depending on n and b.

For example (for n = 4), (x+y)4=x4+4x3y+6x2y2+4xy3+y4

**CHAPTER 4**

**CODE SNIPPET**

**4.1 Source Code**

**4.1.1 Towers of Hanoi Using Stack**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct \_MyStack

{

int \*m\_data;

int m\_numElements;

} MyStack;

static int movecount = 0;

static int countA = 0;

static int countB = 0;

static int countC = 0;

static MyStack \*A = 0;

static MyStack \*B = 0;

static MyStack \*C = 0;

int push(MyStack \*s, int data)

{

if(s->m\_data == NULL) // root node

{

s->m\_numElements = 1;

s->m\_data = (int\*) malloc(sizeof(int));

}

else

{

s->m\_numElements++;

s->m\_data = realloc(s->m\_data, s->m\_numElements \* sizeof(int));

memmove(&s->m\_data[1], s->m\_data, (s->m\_numElements - 1) \* sizeof(int));

}

s->m\_data[0] = data;

return 1;

}

int pop(MyStack \*s)

{

int result = -1;

if(s->m\_data == NULL) // root node

{

s->m\_numElements = 0;

return result;

}

else

{

result = top(s);

if(s->m\_numElements == 1)

{

// last item

s->m\_numElements = 0;

free(s->m\_data);

s->m\_data = NULL;

}

else

{

s->m\_numElements--;

memmove(s->m\_data, &s->m\_data[1], s->m\_numElements \* sizeof(int));

s->m\_data = (int\*) realloc(s->m\_data, s->m\_numElements \* sizeof(int));

}

}

return result;

}

int top(MyStack \*s)

{

if(s->m\_numElements > 0)

return s->m\_data[0];

return 0;

}

int size(MyStack \*s)

{

return s->m\_numElements;

}

void PrintStack(MyStack \*s)

{

int i = 0;

printf("[");

for(i = s->m\_numElements - 1; i >= 0; i--)

{

printf("%d", s->m\_data[i]);

}

printf("]");

}

void PrintStacks()

{

if (countA != A->m\_numElements ||

countB != B->m\_numElements ||

countC != C->m\_numElements)

{

int diffA = A->m\_numElements - countA;

int diffB = B->m\_numElements - countB;

int diffC = C->m\_numElements - countC;

if (diffA == 1)

{

if (diffB == -1)

printf("Move Disc %d From B To A", top(A));

else

printf("Move Disc %d From C To A", top(A));

}

else if (diffB == 1)

{

if (diffA == -1)

printf("Move Disc %d From A To B", top(B));

else

printf("Move Disc %d From C To B", top(B));

}

else //if (diffC == 1)

{

if (diffA == -1)

printf("Move Disc %d From A To C", top(C));

else

printf("Move Disc %d From B To C", top(C));

}

countA = A->m\_numElements;

countB = B->m\_numElements;

countC = C->m\_numElements;

printf("\n");

}

PrintStack(A);

printf(" , ");

PrintStack(B);

printf(" , ");

PrintStack(C);

printf(" , ");

}

void Solve2DiscsTOH(MyStack \*source, MyStack \*temp, MyStack \*dest)

{

push(temp, pop(source));

movecount++;

PrintStacks();

push(dest, pop(source));

movecount++;

PrintStacks();

push(dest, pop(temp));

movecount++;

PrintStacks();

}

int SolveTOH(int nDiscs, MyStack \*source, MyStack \*temp, MyStack \*dest)

{

if (nDiscs <= 4)

{

if ((nDiscs % 2) == 0)

{

Solve2DiscsTOH(source, temp, dest);

nDiscs = nDiscs - 1;

if (nDiscs == 1)

return 1;

push(temp, pop(source));

movecount++;

PrintStacks();

//new source is dest, new temp is source, new dest is temp;

Solve2DiscsTOH(dest, source, temp);

push(dest, pop(source));

movecount++;

PrintStacks();

//new source is temp, new temp is source, new dest is dest;

SolveTOH(nDiscs, temp, source, dest);

}

else

{

if (nDiscs == 1)

return 0;

Solve2DiscsTOH(source, dest, temp);

nDiscs = nDiscs - 1;

push(dest, pop(source));

movecount++;

PrintStacks();

Solve2DiscsTOH(temp, source, dest);

}

return 1;

}

else if (nDiscs >= 5)

{

SolveTOH(nDiscs - 2, source, temp, dest);

push(temp, pop(source));

movecount++;

PrintStacks();

SolveTOH(nDiscs - 2, dest, source, temp);

push(dest, pop(source));

movecount++;

PrintStacks();

SolveTOH(nDiscs - 1, temp, source, dest);

}

return 1;

}

int main()

{

int sz, i, maxdisc;

A = malloc(sizeof(MyStack));

B = malloc(sizeof(MyStack));

C = malloc(sizeof(MyStack));

while(1)

{

printf("\nEnter the number of discs (-1 to exit): ");

scanf("%d", &maxdisc);

if(maxdisc == -1)

break;

if(maxdisc < 2 || maxdisc > 9)

{

printf("Enter between 2 - 9");

continue;

}

movecount = 0;

memset((void\*)A, 0, sizeof(MyStack));

memset((void\*)B, 0, sizeof(MyStack));

memset((void\*)C, 0, sizeof(MyStack));

for (i = maxdisc; i >= 1; i--)

push(A, i);

countA = A->m\_numElements;

countB = B->m\_numElements;

countC = C->m\_numElements;

PrintStacks();

SolveTOH(maxdisc, A, B, C);

printf("Total Moves = %d", movecount);

free(C->m\_data);

}

return 0;

}

**4.1.2 Undirected graph using Kruskal's algorithm**

#include<stdio.h>

#define MAX 30

typedef struct edge

{

int u,v,w;

} edge;

typedef struct edgelist

{

edge data[MAX];

int n;

} edgelist;

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist;

void kruskal();

int find(int belongs[],int vertexno);

void union1(int belongs[],int c1,int c2);

void sort();

void print();

void main()

{

int i,j,total\_cost;

printf("\nEnter number of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

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

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

scanf("%d",&G[i][j]);

kruskal();

print();

}

void kruskal()

{

int belongs[MAX],i,j,cno1,cno2;

elist.n=0;

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

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

{

if(G[i][j]!=0)

{

elist.data[elist.n].u=i;

elist.data[elist.n].v=j;

elist.data[elist.n].w=G[i][j];

elist.n++;

}

}

sort();

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

belongs[i]=i;

spanlist.n=0;

for(i=0;i<elist.n;i++)

{

cno1=find(belongs,elist.data[i].u);

cno2=find(belongs,elist.data[i].v);

if(cno1!=cno2)

{

spanlist.data[spanlist.n]=elist.data[i];

spanlist.n=spanlist.n+1;

union1(belongs,cno1,cno2);

}

}

}

int find(int belongs[],int vertexno)

{

return(belongs[vertexno]);

}

void union1(int belongs[],int c1,int c2)

{

int i;

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

if(belongs[i]==c2)

belongs[i]=c1;

}

void sort()

{

int i,j;

edge temp;

for(i=1;i<elist.n;i++)

for(j=0;j<elist.n-1;j++)

if(elist.data[j].w>elist.data[j+1].w)

{

temp=elist.data[j];

elist.data[j]=elist.data[j+1];

elist.data[j+1]=temp;

}

}

void print()

{

int i,cost=0;

for(i=0;i<spanlist.n;i++)

{

printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);

cost=cost+spanlist.data[i].w;

}

printf("\n\nCost of the spanning tree=%d",cost);

}

**4.1.3 Binomial Co-efficient using Dynamic Programming**

#include<stdio.h>

// Returns value of Binomial Coefficient C(n, k)

int binomialCoeff(int n, int k)

{

// Base Cases

if (k==0 || k==n)

return 1;

// Recur

return binomialCoeff(n-1, k-1) + binomialCoeff(n-1, k);

}

int main()

{

int n,k;

printf("enter the value");

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

if(n<k)

printf("Invalid input\n");

printf("Value of C(%d, %d) is %d ", n, k, binomialCoeff(n, k));

return 0;

}

**CHAPTER 5**

**REFRENCES**

**Tower of Hanoi**

1. Hofstadter, Douglas R. (1985). Metamagical Themas : Questing for the Essence of Mind and Pattern. New York: Basic Books. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*978-0-465-04540-2*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-465-04540-2).
2. Spitznagel, Edward L. (1971). Selected topics in mathematics. Holt, Rinehart and Winston. p. 137. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-03-084693-9](https://en.wikipedia.org/wiki/Special:BookSources/978-0-03-084693-9).
3. [Moscovich, Ivan](https://en.wikipedia.org/wiki/Ivan_Moscovich) (2001). 1000 playthinks: puzzles, paradoxes, illusions & games. Workman. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-7611-1826-8](https://en.wikipedia.org/wiki/Special:BookSources/978-0-7611-1826-8).
4. Petković, Miodrag (2009). Famous Puzzles of Great Mathematicians. AMS Bookstore. p. 197. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-8218-4814-2](https://en.wikipedia.org/wiki/Special:BookSources/978-0-8218-4814-2).
5. Troshkin, M. "Doomsday Comes: A Nonrecursive Analysis of the Recursive Towers-of-Hanoi Problem". Focus (in Russian). 95 (2): 10–14.
6. Mayer, Herbert; Perkins, Don (1984). "Towers of Hanoi Revisited". SIGPLAN Notices. 19 (2): 80–84. [*doi*](https://en.wikipedia.org/wiki/Digital_object_identifier):[*10.1145/948566.948573*](https://doi.org/10.1145%2F948566.948573).
7. Warren, Henry S. (2003). "Section 5-4: Counting Trailing 0's.". Hacker's delight (1st ed.). Boston MA: Addison-Wesley. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*978-0-201-91465-8*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-201-91465-8).
8. Miller, Charles D. (2000). [*"Ch. 4: Binary Numbers and the Standard Gray Code"*](https://web.archive.org/web/20040821062630/http:/occawlonline.pearsoned.com/bookbind/pubbooks/miller2_awl/chapter4/essay1/deluxe-content.html#tower). [*Mathematical Ideas*](http://occawlonline.pearsoned.com/bookbind/pubbooks/miller2_awl/chapter4/essay1/deluxe-content.html) (9 ed.). Addison Wesley Longman. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*978-0-321-07607-6*](https://en.wikipedia.org/wiki/Special:BookSources/978-0-321-07607-6). Archived from the original on 2004-08-21.

**Kruskal’s Algorithm**

1. [Kruskal, J. B.](https://en.wikipedia.org/wiki/Joseph_Kruskal) (1956). "On the shortest spanning subtree of a graph and the traveling salesman problem". [Proceedings of the American Mathematical Society](https://en.wikipedia.org/wiki/Proceedings_of_the_American_Mathematical_Society). **7**: 48–50. [doi](https://en.wikipedia.org/wiki/Digital_object_identifier):[10.1090/S0002-9939-1956-0078686-7](https://doi.org/10.1090%2FS0002-9939-1956-0078686-7). [JSTOR](https://en.wikipedia.org/wiki/JSTOR) [2033241](https://www.jstor.org/stable/2033241).
2. Quinn, Michael J.; Deo, Narsingh (1984). "Parallel graph algorithms". ACM Computing Surveys (CSUR) 16.3: 319–348.
3. Grama, Ananth; Gupta, Anshul; Karypis, George; Kumar, Vipin (2003). Introduction to Parallel Computing. pp. 412–413. [*ISBN*](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [*978-0201648652*](https://en.wikipedia.org/wiki/Special:BookSources/978-0201648652).

**Binomial Co-efficient**

1. Ash, Robert B. (1990) [1965]. [Information Theory](https://www.amazon.com/Information-Theory-Dover-Books-Mathematics/dp/0486665216). Dover Publications, Inc. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-486-66521-6](https://en.wikipedia.org/wiki/Special:BookSources/0-486-66521-6).
2. [Benjamin, Arthur T.](https://en.wikipedia.org/wiki/Arthur_T._Benjamin); [Quinn, Jennifer J*.*](https://en.wikipedia.org/wiki/Jennifer_Quinn) (2003). Proofs that Really Count: The Art of Combinatorial Proof. Dolciani Mathematical Expositions. **27**. [Mathematical Association of America](https://en.wikipedia.org/wiki/Mathematical_Association_of_America). [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [978-0-88385-333-7](https://en.wikipedia.org/wiki/Special:BookSources/978-0-88385-333-7).
3. [Bryant, Victor](https://en.wikipedia.org/w/index.php?title=Victor_Bryant&action=edit&redlink=1) (1993). Aspects of combinatorics. Cambridge University Press. [ISBN](https://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-521-41974-3](https://en.wikipedia.org/wiki/Special:BookSources/0-521-41974-3).