DYNAMIC PROGRAMMING & OTHER ALGORITHMS

Dynamic Programming is mainly an optimization over plain recursion.

The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later.

EX: FIBONACCI

SEQUENCE **1, 1, 2, 3, 5, 8, 13, 21,**

...

FIBONACCI IN CODE:

```
void fib ()

{

fibresult[0] = 1;

fibresult[1] = 1,

for (int i = 2; i<n; i++)

fibresult[i] = fibresult[i-1] + fibresult[i-2];

}
```

DYNAMIC PROGRAMMING VS GREEDY ALGORITHM

GREEDY ALGORITHM

In a greedy Algorithm, we make whatever choice seems best at the moment in the hope that it will lead to global optimal solution.

DYNAMIC PROGRAMMING

In Dynamic Programming we make decision at each step considering current problem and solution to previously solved sub problem to calculate optimal solution.

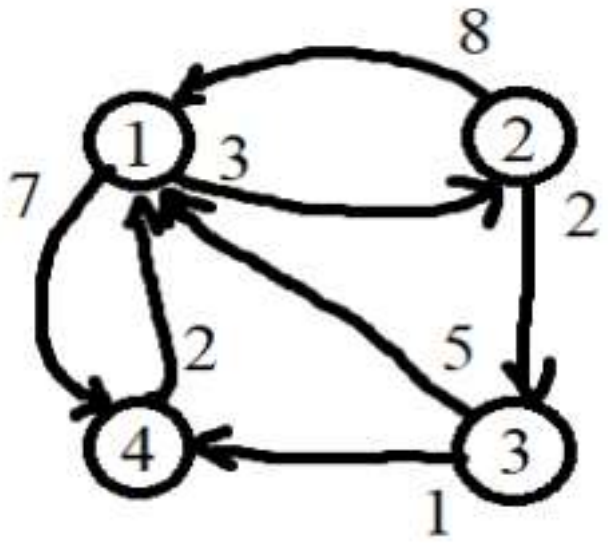
FLOYD-WARSHALL ALGORITHM

The Floyd-Warshall Algorithm is for solving all pairs shortest path problems. The problem is to find the shortest distances between every pair of vertices in a given edge-weighted directed Graph.

FLOYD-WARSHALL ALGORITHM

Given Graph:

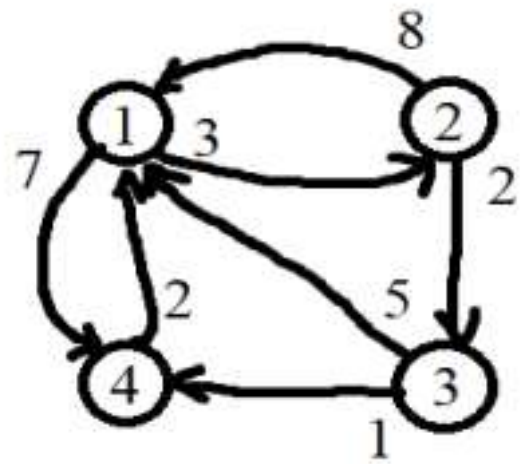
edges



vertex

vertices

HOW TO FIND THE SHORTEST PATH?



$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 4 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Step 1:

Create a matrix A0 of dimension n*n where n is the number of vertices. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph.

Each cell A[i][j] is filled with the distance from the ith vertex to the jth vertex. If there is no path from ith vertex to jth vertex, the cell is left as infinity.

$A^1 =$

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	8	∞	0

$A^1 =$

	1	2	3	4
1	0	3	∞	7
2	8	0		
3	5		0	
4	2			0

Step 2: Now, create a matrix A1 using matrix A0. The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way.

Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is

filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]).

That is, if the direct distance from the source to the destination is greater than the path through the vertex k, then the cell is filled with A[i][k] + A[k][j].

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & \infty \\ 5 & \infty & 0 & 1 \\ 4 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & 2 & 15 \\ 5 & 8 & 0 & 1 \\ 2 & 8 & \infty & 0 \end{bmatrix} \end{array}$$

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 7 \\ 8 & 0 & & \\ 5 & & 0 & \\ 2 & & & 0 \end{bmatrix} \end{array}$$

SOURCE CODE:

```
for(i=1; i<=n; i++)
 {
    for (j=1; j<=n; j++)
     {
        A[I,j] =min(A[i,j] , A[i,k] + A[k,j]); }
        }
```

LET US

UNDERSTAND LCS WITH AN EXAMPLE.

THE FOLLOWING STEPS ARE FOLLOWED FOR FINDING THE LONGEST COMMON

SUBSEQUENCE.

1. Create a table of dimension n+1*m+1 where n and m are the lengths of X and Y respectively. The first row and the first column are filled with zeros.

3. If the character corresponding to the current row and current column are matching, then fill the current cell by adding one to the diagonal element. Point an arrow to the diagonal cell.

5. Step 2 is repeated until the table is filled.

Thus, the longest common subsequence is CA.

CONSTRAINTS

GRAPH COLORING

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET PROBLEM

CONSTRAINTS

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET PROBLEM

GRAPH COLORING

CONSTRAINTS

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET PROBLEM

GRAPH COLORING

PROBLEM

GRAPH COLORING

CONSTRAINTS

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET

CONSTRAINTS

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET PROBLEM

GRAPH COLORING

Subset Sum Problem Solution using Backtracking Algorithm

The main idea is to add the number to the stack and track the sum of stack values.

CONSTRAINTS

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET PROBLEM

GRAPH COLORING

A graph coloring is an assignment of labels traditionally called "colors" to each vertex.

Example:
In this approach, we color a single vertex and then move to its adjacent (connected) vertex to color it with different color. After

coloring, we again move to another adjacent vertex that is uncolored and repeat the process until all vertices of the given graph are colored.

GRAPH COLORING

CONSTRAINTS

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET PROBLEM

	C1	C2	C1	C2	C1	C2

CONSTRAINTS

APPLICATIONS OF BACKTRACKING

SUM OF SUBSET PROBLEM

GRAPH COLORING

Hamilton Path is a path that goes through every Vertex of a graph exactly once.

A Hamilton Circuit is a Hamilton Path that begins and ends at the same vertex.

Example:
How does hashing work:
-It uses functions or algorithms to map object data to a representative integer value.

-A
hash
can
then
be
used
to
narrow down
searches when
locating these
items on that
object data map.

A problem-solving technique that works by maintaining a window, a contagious part of the data. Its disadvantage is that if we want to calculate something about the next window, we don’t need to do it from scratch, we just take the value of the actual window and update in in some way, which is faster.

Rabin-Karp algorithm is an

algorithm used for searching/matching patterns in the text using a hash function. (

value)

Unlike Naive string-matching algorithm, it does not travel through every character in the initial phase rather it filters the characters that do not match and then performs the comparison.

Hashing the process of converting a key to a fixed-size

A sequence of characters is taken and checked for the possibility of the presence of the required string. If the possibility is found then, character matching is performed.

Example:

Given a string s of size n and a pattern p of size m characters of s with p

We have “string s” of n characters, and “string p” of m characters.
Two equal ‘string’ must have the same number of characters, m in this case because we are searching for p, So, the idea is to compare all substring of s of m characters with p.

finding the value ‘cab’ in ‘abcabfdeabaccabebf’.

We have “string s” of n characters, and “string p” of m characters.

Two equal ‘string’ must have the same number of characters, m in this case because we are searching for p,
So, the idea is to compare all substring of s of m characters with p.
Example. Values of ‘s’ and ‘p’, we are