* **Dynamic Programming Approaches**

1. Bottom-Up approach

Algorithm

1. set Fib[0] = 0

2. set Fib[1] = 1

3. From index 2 to n compute result using the below formula

     Fib[index] = Fib[index - 1] + Fib[index - 2]

4. The final result will be stored in Fib[n].

**Code**

#include<stdio.h>

**int** **Fibonacci**(**int** N)

{

**int** Fib[N+**1**],i;

Fib[**0**] = **0**;

Fib[**1**] = **1**;

**for**(i = **2**; i <= N; i++)

Fib[i] = Fib[i-**1**]+Fib[i-**2**];

**return** Fib[N];

}

**int** **main**()

{

**int** n;

scanf("%d",&n);

**if**(n <= **1**)

printf("Fib(%d) = %d**\n**",n,n);

**else**

printf("Fib(%d) = %d**\n**",n,Fibonacci(n));

**return** **0**;}

1. Top-Down approach

## Algorithm

1. Fib(n)
2. If n == 0 || n == 1 return n;
3. Otherwise, compute subproblem results recursively.
4. return Fib(n-1) + Fib(n-2);

**Code**

#include<stdio.h>

**int** **Fibonacci**(**int** N)

{

**if**(N <= **1**)

**return** N;

**return** Fibonacci(N-**1**) + Fibonacci(N-**2**);

}

**int** **main**()

{

**int** n;

scanf("%d",&n);

printf("Fib(%d) = %d**\n**",n,Fibonacci(n));

**return** **0**;

}

* **0/1 Knapsack**
* if wt[i] > w then
* V[i,w] = V[i-1,w]
* else if wt[i] <= w then
* V[i,w] = max( V[i-1,w], val[i] + V[i-1, w - wt[i]] )
* After calculation, the value table V

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| V[i,w] | w = 0 | 1 | 2 | 3 | 4 | 5 |
| i = 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 100 | 100 | 100 |
| 2 | 0 | 0 | 20 | 100 | 100 | 120 |
| 3 | 0 | 0 | 20 | 100 | 100 | 120 |
| 4 | 0 | 40 | 40 | 100 | 140 | 140 |

* Maximum value earned  
  Max Value = V[n,W]  
  = V[4,5]  
  = 140

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

  int i, w;

  int V[n+1][W+1];

  for(w = 0; w <= W; w++) {

    V[0][w] = 0;

  }

  for(i = 0; i <= n; i++)

  {

    V[i][0] = 0;

  }

  for(i = 1; i <= n; i++) {

    for(w = 1; w <= W; w++) {

      if(wt[i] <= w) {

        V[i][w] = getMax(V[i-1][w], val[i] + V[i-1][w - wt[i]]);

      } else {

        V[i][w] = V[i-1][w];

      }

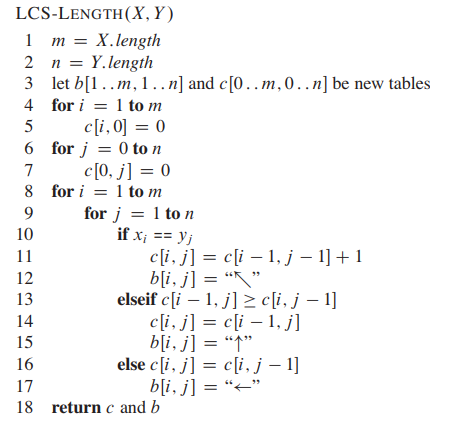
    }

  }

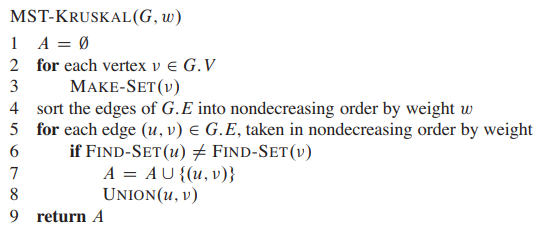
  printf("Max Value: %d\n", V[n][W]);

}

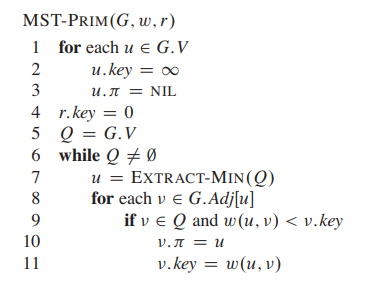
* **LCS**

****

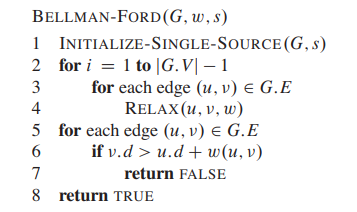
* **Kruskal**

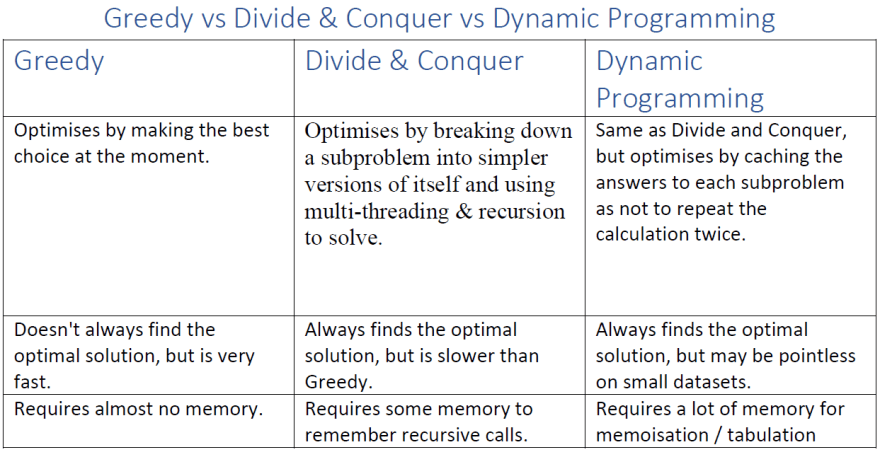
****

* **Prims**

****

* **Bellman Ford**

****



* **Dijkstra**

#include <stdio.h>

#define INFINITY 9999

#define MAX 4

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {

  int cost[MAX][MAX], distance[MAX], pred[MAX];

  int visited[MAX], count, mindistance, nextnode, i, j;

  for (i = 0; i < n; i++)

    for (j = 0; j < n; j++)

      if (Graph[i][j] == 0)

        cost[i][j] = INFINITY;

      else

        cost[i][j] = Graph[i][j];

  for (j = 0; j < n; j++) {

    distance[j] = cost[start][j];

    pred[j] = start;

    visited[j] = 0;

  }

  distance[start] = 0;

  visited[start] = 1;

  count = 1;

  while (count < n - 1) {

    mindistance = INFINITY;

    for (i = 0; i < n; i++)

      if (distance[i] < mindistance && !visited[i]) {

        mindistance = distance[i];

        nextnode = i;

      }

    visited[nextnode] = 1;

    for (i = 0; i < n; i++)

      if (!visited[i])

        if (mindistance + cost[nextnode][i] < distance[i]) {

          distance[i] = mindistance + cost[nextnode][i];

          pred[i] = nextnode;

        }

    count++;

  }

  for (i = 0; i < n; i++)

    if (i != start) {

      printf("\nDistance from source to %d: %d", i, distance[i]);

    }

}