**Fractional Knapsack:**

**Code**

#include <cstdio>

#include <algorithm>

using namespace std;

struct Item {

int profit, weight;

};

static bool cmp(Item a, Item b) {

double r1 = (double)a.profit / (double)a.weight;

double r2 = (double)b.profit / (double)b.weight;

return r1 > r2;

}

int main() {

int W;

printf("Enter the maximum capacity of the knapsack: ");

scanf("%d", &W);

int N;

printf("Enter the number of items: ");

scanf("%d", &N);

Item arr[N];

printf("Enter profit and weight of each item:\n");

for (int i = 0; i < N; ++i) {

printf("Item %d:\n", i + 1);

printf("Profit: ");

scanf("%d", &arr[i].profit);

printf("Weight: ");

scanf("%d", &arr[i].weight);

}

sort(arr, arr + N, cmp);

double finalvalue = 0.0;

for (int i = 0; i < N; i++) {

if (arr[i].weight <= W) {

W -= arr[i].weight;

finalvalue += arr[i].profit;

} else {

finalvalue += arr[i].profit \* ((double)W / (double)arr[i].weight);

break;

}

}

printf("Maximum profit that can be achieved: %.2lf\n", finalvalue);

return 0;

}  
**Psudeo Code:**

knapsack( arr[], size, capacity) {

sort(arr, arr + size, cmp);

double finalvalue = 0.0;

for (int i = 0; i < size; i++) {

if (arr[i].weight <= capacity) {

W -= arr[i].weight;

finalvalue += arr[i].profit;

} else {

finalvalue += arr[i].profit \* ((double)capacity / (double)arr[i].weight);

break;

}

}

return finalvalue;

}

**Algorithm:**

v = values of items

w = weights of items

c = capacity

I = sorted items according to v/w in descending order

n = number of items

i= current item

KS(c, v, w)

sort the item according to v/w in descending order and store in I

1. i =0, frac=1;
2. tVal=0;
3. n = I.length;
4. while (c>0 && i<n)
5. if(w[i]<=c) frac = 1;
6. else frac = c/w[i]
7. c = c – frac\*w[i]
8. tVal += fract\*v[i]
9. return tVal;

Fractional Knapsack Algorithm - recursive

1. KS(c, I, i, n)
2. if (c<=0 or i>n) return 0;
3. if (c < I[i].weight)
4. frac = c/I[i].weight
5. return frac\* I[i].value + KS(0, I, i+1, n)
6. else
7. return I[i].value + KS(c-I[i].weight, I, i+1, n)

Time Complexity:

* Sorting time complexity 🡪O(nlogn)
* Overall time complexity🡪O(nlogn)+O(n)= O(nlogn)

**Simulation:**

Step1: P/w ratio

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Items: | 1 | 2 | 3 | 4 | 5 |
| Weight (kg) | 3 | 3 | 2 | 5 | 1 |
| Profit | 10 | 15 | 10 | 20 | 8 |
| Pi/w | 3.3 | 5 | 5 | 4 | 8 |

Step2: Decreasing sorting (based on P/w)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Items: | 5 | 2 | 3 | 4 | 1 |
| Weight (kg) | 1 | 3 | 2 | 5 | 3 |
| Profit | 8 | 15 | 10 | 20 | 10 |
| Pi/w | 8 | 5 | 5 | 4 | 3.3 |
| Knapsack: | 1 | 1 | 1 | 4/5 | 0 |

Step3: Item Choose (C=10 kg)

Remaning  
 10-1=9KG  
 9-3 =6KG  
 6-2 =4KG  
 4-4=0

Weights:1+3+2+5 \* 4/5=10

Maximum profit:8+15+10+20\*4/5=

Items:{ 5, 2 , 3 ,4\*4/5 ,1}

Items:{I2, I3 , 4/5I , I5}

Example2:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Items: | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Profit | 12 | 5 | 16 | 7 | 9 | 11 | 6 |
| Weight (kg) | 3 | 1 | 4 | 2 | 9 | 4 | 3 |
| Pi/w | 4 | 5 | 4 | 3.5 | 1 | 2.75 | 2 |

n=7,c=15

|  |  |  |  |
| --- | --- | --- | --- |
| Sorted Items | Profit | Weight | Remaining weight |
| 2 | 5 | 1 | 15-1=14 |
| 1 | 12 | 3 | 14-3=11 |
| 3 | 16 | 4 | 11-4=7 |
| 4 | 7 | 2 | 7-2=5 |
| 6 | 11 | 4 | 5-4=1 |
| 7 | 6\*1/3=2 | 1 | 1-1=0 |
|  | Max=53 | 15 |  |

**Job Sequence with dealines**

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

typedef struct {

char id;

int dead;

int profit;

} Job;

bool comp(Job job1, Job job2) {

return (job1.profit > job2.profit);

}

void printJobScheduling(Job arr[], int n) {

qsort(arr, n, sizeof(Job), comp);

bool slot[n];

int maxProfit = 0;

for (int i = 0; i < n; i++) slot[i] = false;

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

for (int j = arr[i].dead - 1; j >= 0; j--) {

if (!slot[j]) {

printf("%c ", arr[i].id);

slot[j] = true;

maxProfit += arr[i].profit;

break;

}

}

}

printf("\nMaximum Profit: %d\n", maxProfit);

}

int main() {

int n;

printf("Enter the number of jobs: ");

scanf("%d", &n);

Job arr[n];

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

printf("Enter job name for job %d: ", i + 1);

scanf(" %c", &arr[i].id);

printf("Enter Deadline for job %c: ", arr[i].id);

scanf("%d", &arr[i].dead);

printf("Enter Profit for job %c: ", arr[i].id);

scanf("%d", &arr[i].profit);

}

printf("Following is maximum profit sequence of jobs:\n");

printJobScheduling(arr, n);

return 0;

}  
**Psudeo Code:**   
void printJobScheduling(arr, n) {

Quicksort (arr, n, size, compare using profit);;

bool slot[n];

int maxProfit = 0;

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

slot[i] = false;

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

for (int j = arr[i].dead - 1; j >= 0; j--) {

if (!slot[j]) {

printf("%c ", arr[i].id);

slot[j] = true;

maxProfit += arr[i].profit;

break;

}

}

}

**Algorithm:**

Job sequence:

JobScheduling(arr[],n)

Step 1: Sort the Array of jobs by profit.

Step 2: Create a Boolean array slot[] of size n and initialize all entries as false.

Step 3: maxProfit = 0.

Step 4: for i 0 to n i++ and j = arr[i].dead – 1 to 0 j--

Step 5: if slot[j] is false

Step 6: print arr[i].id

Step 7: set slot[j] is true

Step 8: maxProfit += arr[i].profit;

Step 9: break the loop;

**Simulation:** Example 1,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Job: | J1 | J2 | J3 | J4 | J5 |
| Deadline: | 2 | 1 | 2 | 1 | 3 |
| Profit: | 100 | 50 | 75 | 80 | 90 |

Step1: Sort according to profit

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Job: | J1 | J5 | J4 | J3 | J2 |
| Deadline: | 2 | 3 | 1 | 2 | 1 |
| Profit: | 100 | 90 | 80 | 75 | 50 |
| 0 | 1 | 2 | 3 | 4 | 5 |

Step2: Gantt Chart

|  |  |  |
| --- | --- | --- |
| J4 | J1 | J5 |
| 0 | 1 | 2 3 |

Max Profit = 80+100+90=270

**:.** Job sequence J4🡪J1🡪J5

Example 2,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Job: | J1 | J2 | J3 | J4 | J5 | J6 |
| Deadline: | 5 | 3 | 3 | 2 | 4 | 2 |
| Profit: | 200 | 180 | 190 | 300 | 120 | 100 |

Step1: Sort according to profit

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Job: | J4 | J1 | J3 | J2 | J5 | J6 |
| Deadline: | 2 | 5 | 3 | 3 | 4 | 2 |
| Profit: | 300 | 200 | 190 | 180 | 120 | 100 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

Step2: Gantt Chart

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| J2 | J4 | J3 | J5 | J1 |
| 0 | 1 | 2 3 | 4 | 5 |

Max Profit = 180+300+190+120+200=990

**:.** Job sequence J2🡪J4🡪J3🡪J5🡪J1

**Coin Change**

**Code:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

int i,Size,value,MAX=100,c = 0;

printf("Enter size:");

scanf("%d",&Size);

printf("\nEnter value:");

scanf("%d",&value);

int coins[Size];

printf("Enter Coins:");

for (i = 0; i < Size; i++) {

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

}

int ans[MAX];

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

{

while(value >= coins[i])

{

value -= coins[i];

ans[c] = coins[i];

c++;

}

if(value == 0)

break;

}

printf("Total Coins Needed = %d\n",c);

printf("Coins are:\t");

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

printf("%d ", ans[i]);

return 0;

}

**Algorithm:**

Coin change:

Step 1: take an array[] with max value and count=0;

Step 2: for i 0 to size

Step 3: while searching value > coins[i]

Step 4: value = value –coin[i]

Step 5: array[c] =coins[i] c++

Step 6: if value =0

Step 7: break the loop;

**Simulation:**

Given a set of coins and a value, we have to find the minimum number of coins which satisfies the value.

**Example**

coins[] = {5,10,20,25}

value = 50

### **Possible Solutions**

{coin \* count}

{5 \* 10} = 50 [10 coins]

{5 \* 8 + 10 \* 1} = 50 [9 coins] goes on.

{10 \* 5} = 50 [5 coins]

{20 \* 2 + 10 \* 1} = 50 [3 coins]

{20 \* 2 + 5 \* 2} = 50 [4 coins]

{25 \* 2} = 50 [2 coins]

etc etc

### **Best Solution**

Two 25 rupees. Total coins two.

25 \* 2 = 50

**Psudeo Code:**

calculateCoins(coins[],Size, value,ans[]) {

int c = 0;

for (int i = 0; i < Size; i++) {

while (value >= coins[i]) {

value -= coins[i];

ans[c] = coins[i];

c++;

}

if (value == 0)

break;

}

return c;

}

**BFS:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

const int MAX\_VERTICES = 100;

void addEdge(int adjMatrix[][MAX\_VERTICES], int k, int h) {

adjMatrix[k][h] = 1;

adjMatrix[h][k] = 1;

}

void BFS(int adjMatrix[][MAX\_VERTICES], bool visited[], int numVertices, int startVertex) {

int queue[MAX\_VERTICES];

int front = 0, rear = -1;

visited[startVertex] = true;

queue[++rear] = startVertex;

while (front <= rear) {

int current = queue[front++];

printf("%d ", current);

for (int i = 0; i < numVertices; ++i) {

if (adjMatrix[current][i] && !visited[i]) {

visited[i] = true;

queue[++rear] = i;

}

}

}

}

int main() {

int numVertices, numEdges, k, h;

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

printf("Enter the number of edges: ");

scanf("%d", &numEdges);

int adjMatrix[MAX\_VERTICES][MAX\_VERTICES] = {0};

bool visited[MAX\_VERTICES] = {false};

printf("Enter edges (node1 node2):\n");

for (int i = 0; i < numEdges; ++i) {

printf("Edge %d: ", i + 1);

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

addEdge(adjMatrix, k, h);

}

int startVertex;

printf("Enter the starting vertex: ");

scanf("%d", &startVertex);

printf("BFS traversal: ");

BFS(adjMatrix, visited, numVertices, startVertex);

return 0;

}

**Algorithm:**

BFS(G,s)

1.**for** each vertex u in V[G] – {s}

2.**do** color[u] ← white

3.d[u] ← ∝

4.π[u] ← nil

5.color[s] ← gray

6.d[s] ← 0

7.π[s] ← nil

8.Q ← Φ

9.enqueue(Q,s)

10.**while** Q ≠ Φ

11.do u ← dequeue(Q)

12.**for** each v in Adj[u]

13.**do** if color[v] = white

14.**then** color[v] ← gray

15.d[v] ← d[u] + 1

16.π[v] ← u

17.enqueue(Q,v)

18.color[u] ← black

**Simulation:**

**Shortest path:**

3

1

2

5

2

3

Available paths:

A🡪B🡪D🡪C=1+2+3=6

A🡪B🡪E🡪D🡪C=1+2+5+3=11

A🡪E🡪D🡪C=3+5+3=11

Shortest among all available path=6

A🡪B🡪D🡪C=6

3

1

2

5

2

3

**TREE:**

**DFS:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

const int MAX\_VERTICES = 1000;

void addEdge(int adjList[][MAX\_VERTICES], int k, int h) {

adjList[k][h] = 1;

adjList[h][k] = 1;

}

void DFS(const int adjList[][MAX\_VERTICES], bool visited[], int current, int numVertices) {

visited[current] = true;

printf("%d ", current);

for (int i = 0; i < numVertices; ++i) {

if (adjList[current][i] && !visited[i]) {

DFS(adjList, visited, i, numVertices);

}}}

int main() {

int numVertices, numEdges;

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

printf("Enter the number of edges: ");

scanf("%d", &numEdges);

int adjList[MAX\_VERTICES][MAX\_VERTICES] = {0};

bool visited[MAX\_VERTICES] = {false};

printf("Enter edges (node1 node2):\n");

for (int i = 0; i < numEdges; ++i) {

int k, h;

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

addEdge(adjList, k, h);

}int startVertex;

printf("Enter the starting vertex: ");

scanf("%d", &startVertex);

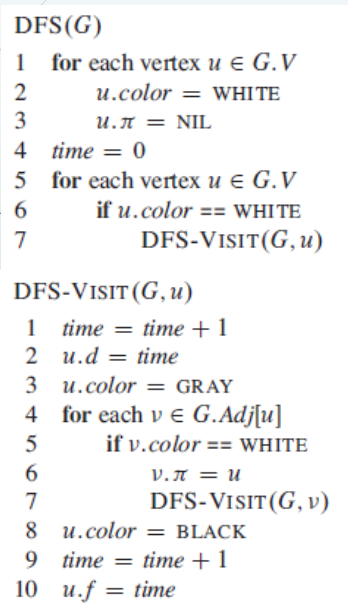
printf("DFS traversal: ");

DFS(adjList, visited, startVertex, numVertices);

return 0;

}

**Algorithm:**

****

**Algorithm 2:**

void DFS(const int adjList[][MAX\_VERTICES], visited[],current, numVertices)

1.visited[current] = true;

2.printf("%d ", current);

3.for (int i = 0; i < numVertices; ++i)

4.if (adjList[current][i] && !visited[i])

5.DFS(adjList, visited, i, numVertices);

void addEdge(adjList[][MAX\_VERTICES],k,h)

1.adjList[k][h] = 1;

2.adjList[h][k] = 1;

**Simulation:**

**Dijkstra:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

int V;

int graph[100][100];

int minDistance(int dist[], bool sptSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[]) {

printf("Vertex \t Distance from Source\n");

for (int i = 0; i < V; i++)

printf("%d \t\t\t\t%d\n", i, dist[i]);

}

void dijkstra(int src) {

int dist[V];

bool sptSet[V];

for (int i = 0; i < V; i++)

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist);

}

int main() {

printf("Enter the number of vertices: ");

scanf("%d", &V);

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

for(int i = 0; i < V; i++)

for(int j = 0; j < V; j++)

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

int src;

printf("Enter the source vertex: ");

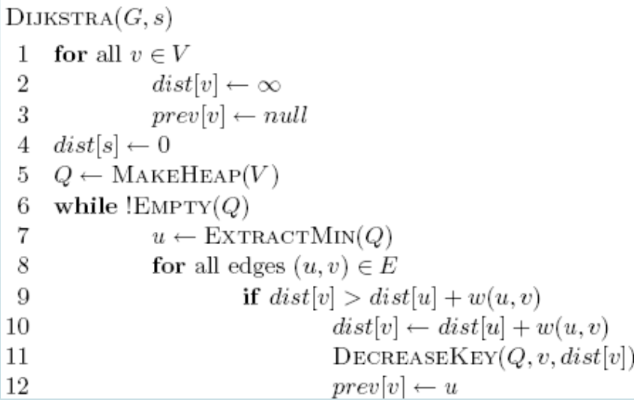
scanf("%d", &src);

dijkstra(src);

return 0;

}

**Algorithm:**



**Algorithm 2:**

Djikstra(G,s)

Step1: For Each vertex V in the Graph

1. Set Distance dist[v] to Infinity
2. Set prev[v] to null

Step2:set distance to the starting node s to 0: dist[s] to 0

Step3: Q 🡨MAKEHEAP(V)

Step4: while Q is not empty  
Step 5: Extract the vertex u with the smallest distance from the Q  
Step 6: for all edges (u,v) to E

If ( dist[v]>dist[u]+w(u,v) )  
 dist[v] 🡨 dist[u]+w(u,v)  
 prev[v]🡨u

Step7: Decrease the priority of v in the priority Q to the new distance

**Simulation:**

∞

∞

3

3

0

1

1

2

1

∞

∞

4

If( d(u) +c(u,v) < d(v)   
 d(v) = d(u) + c(u,v) } Relaxation

∞

∞

3

3

0

1

1

2

1

∞

∞

4

🡺A🡪B || Where 3< ∞  
=>0+3=3  
=> B= 3 Updated;

🡺A🡪C || Where 1< ∞  
=>0+1=1  
=> C= 1 Updated

A visited;

🡺GO to minimum C< B

🡺C🡪B || Where 1+1< 3  
=> B= 2 Updated;

🡺 C🡪E || Where 1+4< ∞  
=>1+4=5  
=> E= 5 Updated;

C visited;

🡺 B🡪D || Where 2+3< ∞  
=>2+3=5  
=> D= 5 Updated;

🡺 B🡪E || Where 2+1< 5  
=>2+1=3  
=> E= 3 Updated;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| visited | A | B | C | D | E |
| A | 0 | ∞ | ∞ | ∞ | ∞ |
| C | 0 | 3 | 1 | ∞ | ∞ |
|  | 0 | 2 | 1 | ∞ | 5 |
| B | 0 | 2 | 1 | 5 | 3 |
| E | 0 | 2 | 1 | 5 | 3 |
| D | 0 | 2 | 1 | 5 | 3 |

∞

∞

7

1

2

0

1

2

5

4

∞

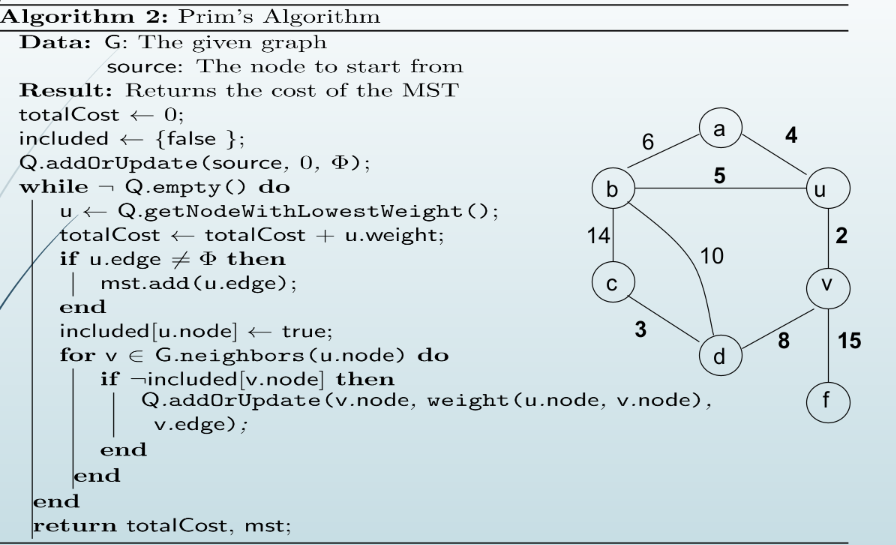
∞

3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| VISITED | A | B | C | D | E | F |
| A | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| B | 0 | 2 | 4 | ∞ | ∞ | ∞ |
| C | 0 | 2 | 3 | 9 | ∞ | ∞ |
| E | 0 | 2 | 3 | 9 | 6 | ∞ |
| D | 0 | 2 | 3 | 8 | 6 | 11 |
|  | 0 | 2 | 3 | 8 | 6 | 9 |

**Prim’s Algorithm:**

**Algorithm:**



**Simulation:**

74

54

88

75

29

31

47

55

79

23

68

66

32

80

93

47

🡺

47

🡺

23

47

🡺

23

32

29

31

47

🡺

23

32

54

29

31

47

🡺

32

23

66

**kruskal’s Algorithm:**

**Algorithm:**

MST-Kruskal(G,w)

01. A ← ∅

02. for each vertex v ∈ V[G] do

03. Make-Set(v)

04. Sort the edges of E by non-decreasing weight w

05. for each edge (u,v) ∈ E, in order by

non-decreasing weight do

06. if Find-Set(u) ≠ Find-Set(v) then

07. A ← A ∪ {(u,v)}

08. Union(u,v)

09. return A

**Simulation:**

2

5

6

3

7

4

8

3

3

2

5

2

7

4

8

3

3

2

Weight of the edge of the above graph is given in below table:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Edge | BF | CD | DF | CE | AD | AB | AE | AC |
| Weight | 2 | 2 | 3 | 3 | 4 | 5 | 7 | 8 |

23

4

3

3

2

MST Cost:2+2+3+3+4=14

**N-Queen:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

int N;

int board[100][100];

void printSolution() {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (board[i][j] == 1) {

printf("Q ");

} else {

printf(". "); }}

printf("\n");}}

bool isSafe(int row, int col) {

int i, j;

for (i = 0; i < col; i++) {

if (board[row][i]) {

return false; }}

for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {

if (board[i][j]) {

return false; }}

for (i = row, j = col; j >= 0 && i < N; i++, j--) {

if (board[i][j]) {

return false; }}

return true;}

bool solveNQUtil(int col) {

if (col >= N) {

return true;

}

for (int i = 0; i < N; i++) {

if (isSafe(i, col)) {

board[i][col] = 1;

if (solveNQUtil(col + 1)) {

return true;

}

board[i][col] = 0;

}

}

return false;

}

bool solveNQ() {

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

board[i][j] = 0;

}

}

if (solveNQUtil(0) == false) {

printf("Solution does not exist");

return false;

}

printSolution();

return true;

}

int main() {

printf("Enter the size of the chessboard: ");

scanf("%d", &N);

solveNQ();

return 0;

}

**Algorithm:**

Place (k, i)

{

For j ← 1 to k - 1

do if (x [j] = i)

or (Abs x [j]) - i) = (Abs (j - k))

then return false;

return true;

}

N - Queens (k, n)

{

For i ← 1 to n

do if Place (k, i) then

{

x [k] ← i;

if (k ==n) then

write (x [1....n));

else

N - Queens (k + 1, n);

}}

**Simulation:**

|  |  |  |  |
| --- | --- | --- | --- |
| Q | - | - | - |
| - | - |  |  |
| - |  | - |  |
| - |  |  | - |

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 | - | - | - |
| - | x | Q2 | x |
| - | x | - | - |
| - |  | x | - |

|  |  |  |  |
| --- | --- | --- | --- |
| Q1 | - | - | - |
| - | - | - | Q2 |
| - | Q3 | - | - |
| - | - | x | - |

|  |  |  |  |
| --- | --- | --- | --- |
| - | Q1 | - | - |
| - | - | - |  |
|  | - |  | - |
|  | - |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| - | Q1 | - | - |
| - | - | - | Q2 |
|  | - | x | - |
|  | - |  | x |

|  |  |  |  |
| --- | --- | --- | --- |
| - | Q1 | - | - |
| - | - | - | Q2 |
| Q3 | - | - | - |
| - | - |  | - |

|  |  |  |  |
| --- | --- | --- | --- |
| - | Q1 | - | - |
| - | - | - | Q2 |
| Q3 | - | - | - |
| - | - | Q4 | - |

**Native:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

void match(char st[100], char pat[100]) {

int n, m, i, j;

n = strlen(st);

m = strlen(pat);

for (i = 0; i <= n - m; i++) {

bool found = true;

for (j = 0; j < m; j++) {

if (st[i + j] != pat[j]) {

found = false;

break;

}}

if (found){

printf(" %d", i+1);

//break;

}}

printf("\n");

}

int main() {

char st[100], pat[100];

printf("\*\*\* Naive String Matching Algorithm \*\*\*\n");

printf("Enter the String:\n");

scanf("%s", st);

printf("Enter the pattern to match:\n");

scanf("%s", pat);

printf("Pattern found at positions:");

match(st, pat);

return 0;}

**Algorithm:**

void match(char arr[], char pat[]) {

1. n = length of arr, m = length of pat

2. for (i = 0; i <= n - m; i++)

3. bool found = true;

4. for (j = 0; j < m; j++)

5. if (arr[i + j] != pat[j])

6. found = false; break;

7. if (found){

8.print index

**Simulation:**

T=1011101110=10  
P=111=3

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

S=2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

S=7

**Rabin-Karp:**

**Algorithm:**

1. n = length of T

2. m =length of P

3. h = dm-1 mod q

4. u = 0

5. v = 0

6. for i = 1 to m

7. u = (du + P[i]) mod q

8. v = (dv+T [i]) mod q

9. for j = 0 to n-m

10. if u = ts

11. if P [1.....m] = T [j+1.....j+ m]

12. then "Pattern occurs with shift" j

13. If j < n-m

14. then ts+1 ← (d (ts-T [j+1]h)+T [j+m+1])mod q

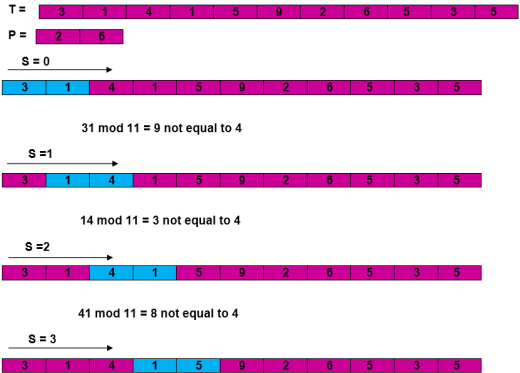
**Example:** For string matching, working module q = 11, how many spurious hits does the Rabin Karp matcher encounters in Text T = 31415926535.......

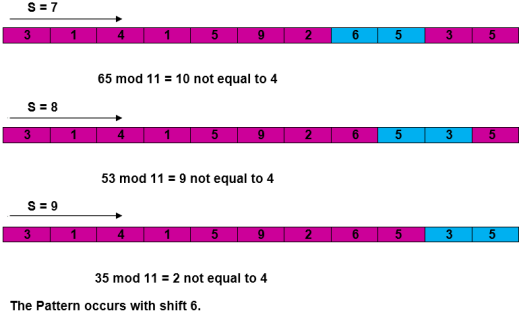
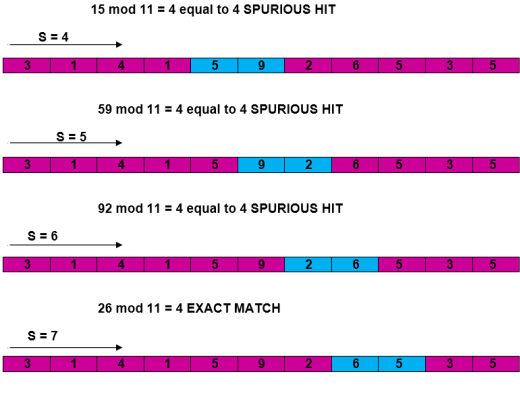
  T = 31415926535.......

  P = 26

 Here T.Length =11 so Q = 11

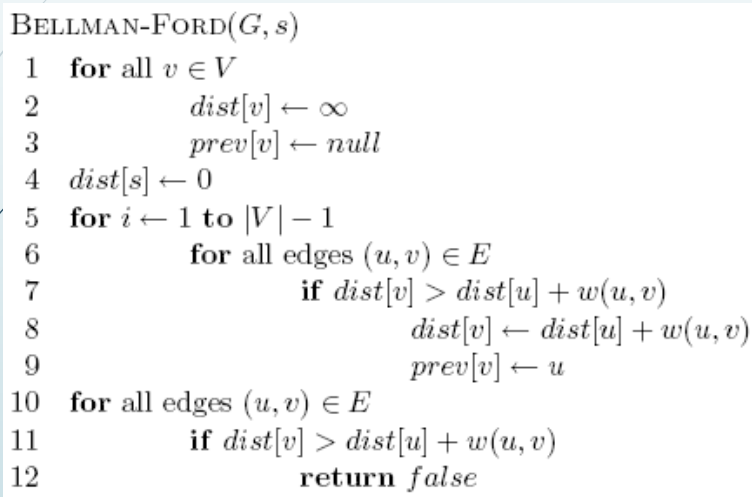
 And P mod Q = 26 mod 11 = 4  





**Bellman Ford:**

**Algorithm:**



**Complexity:**

0/1 Knapsack: O(nlogn)

Job change:O(N^2)

Coin Change:O(N\*C)

BFS:O(V+E)

DFS: O(V+E)

Dijkstra:--O(V^2) (linear array) // O((V+E)log V)(binary heap) // O(VlogV+E)(fib)

Prims:--

Kruskal:O(VE)

N-Queen:O(M^V)

Native:O(n-m+1)

Rabin: O(n+m)m

BellmanFord:--