24/6/21

Q → Floyd algorithm

```
# include <stdio.h>
# include <limits.h>
# define max 10000
int n;
void display ( int d[][n]) {
      for (int i = 0 ; i < n ; i++) {
         for (int j = 0 ; j < n ; j++) {
               if ( d[i][j] == max)
                     printf (" %6s", INF");
               else
                     printf (" %6d ", d[i][j];
         } printf ("\n"); }
}
void floyd ( int g[][n]) {
     int d[n][n], i, j, k;
     for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                 if ( graph[i][j] == -1)
                       d[i][j] = max;
                 else { d[i][j] = g[i][j]; }}}
     printf (" Matrix D(0) : \n");
     display (d);
     for (k = 0; k < n; k++) {
           for (i = 0; i < n; i++) {
            for( j=0 ; j< n; j++ ) {
                if (d[i][j] > d[i][k] + d[k][j])
                    d[i][j] = d[i][k] + d[k][j];
            } }
            printf("\n D(%d) Matrix : \n", k+1);
            display (d); }
}
int main () {
    printf(" No of vertices :- ");
    scanf ("%d", &n);
    int D[n][n];
    printf (" Enter matrix (-1 for no
                direct path) :- \n");
    for(int i=0 ; i<n; i++)
        for(int j=0 ; j<n; j++)
            scanf ("%d", &D[i][j]);
    floyd (D);
    return 0;
}
```

d-> Warshall algorithm

```
# include <stdio.h>
const int Max = 100;
void Warshall (int graph [Max][Max], int n)
{ int i, j, K;
    for (K=0 ; K<n ; K++){
        for (i=0; i<n; i++){
            for (j=0; j<n; j++){
                if (graph[i][j] || (graph [i][K] &&
                            graph [K][j]))
                  graph [i][j] = 1; } } }
}
int main() {
    int i, j, n;
    int g [Max][Max];
    printf (" No. of vertices :: ");
    scanf (" %d", &n);
    printf (" Enter adjacency matrix");
        for (i=0; i<n; i++)
            for (j=0; j<n; j++)
                scanf (" %d", &g[i][j]);
    Warshall (g, n);
    printf (" Transitive Closure :: ");
    for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
            printf("%d\t", g[i][j]);}
        printf("\n");
    }
    return 0;
}
```

Q-> Knapsack algorithm

```
# include <stdio.h>
int max (int a, int b) {
    return (a>b) ? a : b;
}
int Knapsack (int W, int wt[], int
                    val [], int n) {
    int i, w; K[n+1][W+1];
    for (i=0; i<=n ; i++) {
        for (w=0; w <= W ; w++) {
            if (i==0 || w==0)
                K[i][w] = 0;
            else if ( wt[i-1] <= w)
                K[i][w] = max (K[i-1][w],
                (val[i-1] + K[i-1][w-wt[i-1]]));
            else
                K[i][w] = K[i-1][w];
    } }
    return K[n][W];
}
int main () {
    int n, max;
    printf ("Enter no. of Items : ");
    scanf ("%d", &n);
    int val[n], wt[n];
printf(" Enter wt & profit :- ");
for(int i=0; i<n; i++){
    printf(" wt & Value of %d : ", i+1);
    scanf(" %d %d ", &wt[i], &val[i]);
}
printf(" Enter capacity :- ");
scanf(" %d", &max);
printf(" Max profit :- %d",
              KnapSack(max, wt, val, n));
return 0;
}
```