(Connect n ropes with minimal cost)

```
#include <stdio.h>
#include <stdlib.h>

// Function to compare two integers (for min-heap)
int compare(const void* a, const void* b) {
  return (*(int*)a - *(int*)b);
}

// Function to calculate the minimum cost of connecting ropes
int minCost(int ropes[], int n) {
  // Create a min-heap
  qsort(ropes, n, sizeof(int), compare);

  int totalCost = 0;

  // Iterate until only one rope remains
  while (n > 1) {
    // Extract two smallest ropes from the heap
    int min1 = ropes[0];
    qsort(ropes + 1, n - 1, sizeof(int), compare); // Re-heapify the heap
    int min2 = ropes[1];

    // Connect the two ropes and add their length to total cost
    int sum = min1 + min2;
    totalCost += sum;

    // Remove the two smallest ropes and insert their sum back to the heap
    ropes[0] = sum;
    for (int i = 1; i < n - 1; i++) {
```

```
            ropes[i] = ropes[i + 1];
        }
        n--;
    }

    return totalCost;
}

int main() {
    int ropes[] = {5, 4, 2, 8};
    int n = sizeof(ropes) / sizeof(ropes[0]);
    printf("The minimum cost is %d\n", minCost(ropes, n));
    return 0;
}
```

(Replace each array element by its corresponding rank)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

// Structure to store element-index pairs
typedef struct {
    int element;
    int index;
} Pair;

// Compare function for sorting Pairs by element value
int compare(const void* a, const void* b) {
    return ((Pair*)a)->element - ((Pair*)b)->element;
}
```

```c
// Function to replace array elements with their ranks
void replaceWithRanks(int arr[], int n) {
    // Create an array of pairs to store element-index pairs
    Pair pairs[n];
    for (int i = 0; i < n; i++) {
        pairs[i].element = arr[i];
        pairs[i].index = i;
    }

    // Sort pairs based on element values
    qsort(pairs, n, sizeof(Pair), compare);

    // Assign ranks to elements
    int rank = 1;
    for (int i = 0; i < n; i++) {
        arr[pairs[i].index] = rank++;
    }
}

// Function to print an array
void printArray(int arr[], int n) {
    printf("[ ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("]\n");
}

// Main function
int main() {
```

```
    int arr[] = {10, 8, 15, 12, 6, 20, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original array: ");
    printArray(arr, n);

    replaceWithRanks(arr, n);

    printf("Array after replacing with ranks: ");
    printArray(arr, n);

    return 0;
}
```

(Convert max heap to min heap in linear time)

```
#include <stdio.h>

// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to heapify a subtree rooted at index i (min-heapify)
void minHeapify(int arr[], int n, int i) {
    int smallest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] < arr[smallest])
```

```
        smallest = left;

    if (right < n && arr[right] < arr[smallest])
        smallest = right;

    if (smallest != i) {
        swap(&arr[i], &arr[smallest]);
        minHeapify(arr, n, smallest);
    }
}

// Function to convert a max-heap to a min-heap
void convertMaxHeapToMinHeap(int arr[], int n) {
    // Start from the last non-leaf node and heapify each node
    for (int i = (n / 2) - 1; i >= 0; i--) {
        minHeapify(arr, n, i);
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    printf("[ ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("]\n");
}

// Main function
int main() {
    int arr[] = {9, 4, 7, 1, -2, 6, 5};
```

```c
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Original max-heap array: ");
    printArray(arr, n);

    convertMaxHeapToMinHeap(arr, n);

    printf("Min-heap array after conversion: ");
    printArray(arr, n);

    return 0;
}
```

Depth-First Search (DFS)

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_VERTICES 5

void DFS(int vertex, bool visited[], int adjMatrix[MAX_VERTICES][MAX_VERTICES], int vertices) {
    visited[vertex] = true;
    printf("%d ", vertex);

    for (int i = 0; i < vertices; i++) {
        if (adjMatrix[vertex][i] == 1 && !visited[i]) {
            DFS(i, visited, adjMatrix, vertices);
        }
    }
}

int main() {
```

```
    int vertices = MAX_VERTICES;
    int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

    // Example edges for an undirected graph
    adjMatrix[0][1] = 1;
    adjMatrix[1][0] = 1;
    adjMatrix[1][2] = 1;
    adjMatrix[2][1] = 1;
    adjMatrix[2][3] = 1;
    adjMatrix[3][2] = 1;
    adjMatrix[3][4] = 1;
    adjMatrix[4][3] = 1;
    adjMatrix[4][0] = 1;
    adjMatrix[0][4] = 1;

    bool visited[MAX_VERTICES] = {false};

    printf("Depth-First Search starting from vertex 0:\n");
    DFS(0, visited, adjMatrix, vertices);

    return 0;
}
```

Breadth-First Search (BFS

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define MAX_VERTICES 5

void BFS(int startVertex, int adjMatrix[MAX_VERTICES][MAX_VERTICES], int vertices) {
    bool visited[MAX_VERTICES] = {false};
```

```
    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    visited[startVertex] = true;
    queue[rear++] = startVertex;

    while (front != rear) {
        int currentVertex = queue[front++];
        printf("%d ", currentVertex);

        for (int i = 0; i < vertices; i++) {
            if (adjMatrix[currentVertex][i] == 1 && !visited[i]) {
                visited[i] = true;
                queue[rear++] = i;
            }
        }
    }
}

int main() {
    int vertices = MAX_VERTICES;
    int adjMatrix[MAX_VERTICES][MAX_VERTICES] = {0};

    // Example edges for an undirected graph
    adjMatrix[0][1] = 1;
    adjMatrix[1][0] = 1;
    adjMatrix[1][2] = 1;
    adjMatrix[2][1] = 1;
    adjMatrix[2][3] = 1;
    adjMatrix[3][2] = 1;
    adjMatrix[3][4] = 1;
```

```
    adjMatrix[4][3] = 1;
    adjMatrix[4][0] = 1;
    adjMatrix[0][4] = 1;

    printf("Breadth-First Search starting from vertex 0:\n");
    BFS(0, adjMatrix, vertices);

    return 0;
}
```

Prim's Algorithm Example in C

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 5

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, minIndex;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
            min = key[v], minIndex = v;

    return minIndex;
}

void printMST(int parent[], int graph[V][V]) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}
```

```
void primMST(int graph[V][V]) {
    int parent[V];
    int key[V];
    bool mstSet[V];

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;

        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph);
}

int main() {
    int graph[V][V] = {
        {0, 2, 0, 6, 0},
        {2, 0, 3, 8, 5},
        {0, 3, 0, 0, 7},
        {6, 8, 0, 0, 9},
        {0, 5, 7, 9, 0},
```

```
    };

    primMST(graph);

    return 0;
}
```

Dijkstra's Algorithm Example in C

```
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 5

int minDistance(int dist[], bool sptSet[]) {
    int min = INT_MAX, minIndex;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min

Index = v;

    return minIndex;
}

void printSolution(int dist[]) {
    printf("Vertex \tDistance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}
```

```
void dijkstra(int graph[V][V], 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() {
    int graph[V][V] = {
        {0, 10, 20, 0, 0},
        {10, 0, 30, 50, 10},
        {20, 30, 0, 20, 0},
        {0, 50, 20, 0, 60},
        {0, 10, 0, 60, 0},
    };
```

```
    dijkstra(graph, 0);


    return 0;
}
```

Fibonacci using Dynamic Programming (Tabulation

```
#include <stdio.h>


// Function to calculate Fibonacci number using dynamic programming (tabulation)
int fibonacci(int n) {
    if (n <= 1) {
        return n; // Base case: F(0) = 0, F(1) = 1
    }
    int dp[n + 1];
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2]; // Fill the dp array from bottom-up
    }
    return dp[n];
}


int main() {
    int n = 10;
    printf("Fibonacci number at position %d is %d\n", n, fibonacci(n));
    return 0;
}
```

Edit Distance using Dynamic Programming (Tabulation)

```
#include <stdio.h>
#include <string.h>


// Function to find minimum of three numbers
```

```c
int min(int x, int y, int z) {
    return (x < y) ? ((x < z) ? x : z) : ((y < z) ? y : z);
}

// Function to calculate Edit Distance using dynamic programming (tabulation)
int editDistance(char* str1, char* str2, int m, int n) {
    int dp[m + 1][n + 1];

    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0) {
                dp[i][j] = j; // If first string is empty, insert all characters of second string
            } else if (j == 0) {
                dp[i][j] = i; // If second string is empty, remove all characters of first string
            } else if (str1[i - 1] == str2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1]; // If characters are the same, no operation needed
            } else {
                dp[i][j] = 1 + min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]); // Min of insert, delete, replace
            }
        }
    }

    return dp[m][n];
}

int main() {
    char str1[] = "sunday";
    char str2[] = "saturday";
    printf("Minimum edit distance is %d\n", editDistance(str1, str2, strlen(str1), strlen(str2)));
    return 0;
}
```

Knapsack using Dynamic Programming (Tabulation)

```
#include <stdio.h>


// Function to calculate the maximum of two integers

int max(int a, int b) {

    return (a > b) ? a : b;

}


// Function to solve the Knapsack problem using dynamic programming (tabulation)

int knapsack(int val[], int wt[], int n, int W) {

    int dp[n + 1][W + 1];


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

        for (int w = 0; w <= W; w++) {

            if (i == 0 || w == 0) {

                dp[i][w] = 0; // Base case: If no items or no capacity, value is 0

            } else if (wt[i - 1] <= w) {

                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i - 1][w]); // Include or exclude the item

            } else {

                dp[i][w] = dp[i - 1][w]; // Exclude the item

            }

        }

    }


    return dp[n][W];

}


int main() {

    int val[] = {60, 100, 120};

    int wt[] = {10, 20, 30};

    int W = 50;
```

```
   int n = sizeof(val) / sizeof(val[0]);

   printf("Maximum value in Knapsack = %d\n", knapsack(val, wt, n, W));
   return 0;
}
```

LCS using Dynamic Programming (Tabulation

```
#include <stdio.h>
#include <string.h>

// Function to calculate LCS using dynamic programming (tabulation)
#include <stdio.h>
#include <string.h>

int LCS_Tabulation(char *X, char *Y, int m, int n) {
   int L[m+1][n+1];
   for (int i = 0; i <= m; i++) {
      for (int j = 0; j <= n; j++) {
         if (i == 0 || j == 0)
            L[i][j] = 0;
         else if (X[i-1] == Y[j-1])
            L[i][j] = L[i-1][j-1] + 1;
         else
            L[i][j] = (L[i-1][j] > L[i][j-1]) ? L[i-1][j] : L[i][j-1];
      }
   }
   return L[m][n];
}

int main() {
   char X[] = "ad";
   char Y[] = "abcd";
```

```
    int m = strlen(X);
    int n = strlen(Y);
    printf("Length of LCS (Tabulation) is %d\n", LCS_Tabulation(X, Y, m, n));
    return 0;
}
```