AL101A Final Product

Arino, Lalaine S. Mr. Alex Pasion

Compilation of Algorithm and Complexity Activities

1. **Knapsack Algorithm Complete Approach**

*Bottom-up Approach: Top-down Approach:*

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

[Code zip file](file:///C:\Users\arino\OneDrive\Documents\NetBeansProjects\knapsackALGO.zip)

* The highest total amount from the four when using the **top-down approach** is the fractional knapsack, which costs a total of **510**. On the other hand, the total amount in fractional and 0-1 knapsack using **bottom-up approach** is both **150**, therefore, there is no higher or lower total amount as they are equal.

Code:

package knapsackalgo;

public class KnapsackALGO {

    public static void main(String[] args) {

        String item[] = {"A", "B", "C", "D"};

        int w[] = {1, 3, 2, 5};

        int c[] ={200, 240, 140, 150};

        int v[] = {0, 0, 0, 0};

        int n = c.length;

        int k = 5;

        for(int i = 0; i < n; i++){

            v[i] = c[i] / w[i];

        } //step1

        System.out.println("Unsorted:");

        System.out.println("ITEM\tWEIGHT\tCOST\tVALUE");

        System.out.println(item[0] + "\t" + w[0] + "\t" + c[0] + "\t" + v[0]);

        System.out.println(item[1] + "\t" + w[1] + "\t" + c[1] + "\t" + v[1]);

        System.out.println(item[2] + "\t" + w[2] + "\t" + c[2] + "\t" + v[2]);

        System.out.println(item[3] + "\t" + w[3] + "\t" + c[3] + "\t" + v[3]);

        sort(item, w, c, v, n);

        System.out.println("SORTED:");

        System.out.println("ITEM\tWEIGHT\tCOST\tVALUE");

        System.out.println(item[0] + "\t" + w[0] + "\t" + c[0] + "\t" + v[0]);

        System.out.println(item[1] + "\t" + w[1] + "\t" + c[1] + "\t" + v[1]);

        System.out.println(item[2] + "\t" + w[2] + "\t" + c[2] + "\t" + v[2]);

        System.out.println(item[3] + "\t" + w[3] + "\t" + c[3] + "\t" + v[3]);

         Fknapsack(item, c, w, n, k);

         zerooneknapsack(item, c, w, n, k);

     }

    public static void sort(String item[], int w[], int c[], int v[], int n){ //step2

        int tempC, tempW, tempV;

        String tempI;

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

             for (int j = i + 1; j < n; j++) {

                 if (v[i] < v[j]) {

                     tempC = c[i];

                     c[i] = c[j];

                     c[j] = tempC;

                     tempW = w[i];

                     w[i] = w[j];

                     w[j] = tempW;

                     tempV = v[i];

                     v[i] = v[j];

                     v[j] = tempV;

                     tempI = item[i];

                     item[i] = item[j];

                     item[j] = tempI;

                 }

             }

         }

    }

    public static void Fknapsack(String item[], int c[], int w[], int n, int k){ // step 3

        System.out.println("FRACTIONAL KNAPSACK");

        int sum = 0, fraction;

        System.out.println("ITEM\tWEIGHT\tCOST");

        for (int i = 0; i < n; i++) {

            if (k >= w[i]) {

                System.out.println(item[i] + "\t" + w[i] + "\t" + c[i]);

                k -= w[i];

                sum += c[i];

            }

            else if (k < w[i]) {

                fraction = k \* (c[i] / w[i]); // or v[i]

                sum += fraction;

                if (k == 0) {

                    break;

                }

                System.out.println(item[i] + "\t" + k + "\t" + fraction);

                break;

            }

        }

        System.out.println("Total: " + sum);

    }

    public static void zerooneknapsack(String item[], int c[], int w[], int n, int k){ // step 3

        System.out.println("O-1 KNAPSACK");

        int sum = 0, fraction;

        System.out.println("ITEM\tWEIGHT\tCOST");

        for (int i = 0; i < n; i++) {

            if (k >= w[i]) {

                System.out.println(item[i] + "\t" + w[i] + "\t" + c[i]);

                k -= w[i];

                sum += c[i];

            }

        }

        System.out.println("Total: " + sum);

    }

}

1. **Activity Selection Algorithm**

A screenshot of a computer

Description automatically generated

[Code zip file](file:///C:\Users\arino\OneDrive\Documents\NetBeansProjects\activityselection)

Code:

class ActivitySelection {

    public static void printMaxActivities(String subj[], String s[], String f[], int sTime[], int fTime[], int n) {

        int i, j;

        System.out.println("The following activities are selected:");

        System.out.println("Subject\t|| Start Time\t|| Finish Time");

        System.out.println("--------------------------------------------");

        i = 0;

        System.out.println(subj[i] + "\t\t|| " + s[i] + "\t|| " + f[i]);

        for (j = 1; j < n; j++) {

            if (sTime[j] >= fTime[i]) {

                System.out.println(subj[j] + "\t\t|| " + s[j] + "\t|| " + f[j]);

                i = j;

            }

        }

    }

    public static void main(String[] args) {

        String[] subj = {"Eng", "Fil", "Math", "Sci", "PE", "Physics"};

        String[] s = {"01:00 pm", "03:00 pm", "12:00 pm", "05:00 pm", "08:00 pm", "05:00 pm"};

        String[] f = {"02:00 pm", "04:00 pm", "06:00 pm", "07:00 pm", "09:00 pm", "09:00 pm"};

        int[] sTime = {13, 15, 12, 17, 20, 17};

        int[] fTime = {14, 16, 18, 19, 21, 21};

        int n = sTime.length;

        printMaxActivities(subj, s, f, sTime, fTime, n);

    }

}

1. **Weighted Job Scheduling**

A screen shot of a schedule

Description automatically generated

Code:

import java.util.\*;

class Job {

    int start, finish, profit;

    Job(int start, int finish, int profit) {

        this.start = start;

        this.finish = finish;

        this.profit = profit;

    }

}

public class Main {

    public static int findLastNonConflictingJob(List<int[]> indexedJobs, int n) {

        int low = 0;

        int high = n - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            if (indexedJobs.get(mid)[2] <= indexedJobs.get(n)[1]) {

                if (mid + 1 <= high && indexedJobs.get(mid + 1)[2] <= indexedJobs.get(n)[1]) {

                    low = mid + 1;

                } else {

                    return mid;

                }

            } else {

                high = mid - 1;

            }

        }

        return -1;

    }

    public static void findMaxProfitJobs(List<Job> jobs) {

        List<int[]> indexedJobs = new ArrayList<>();

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

            indexedJobs.add(new int[]{i, jobs.get(i).start, jobs.get(i).finish, jobs.get(i).profit});

        }

        indexedJobs.sort(Comparator.comparingInt(x -> x[2]));

        int n = indexedJobs.size();

        if (n == 0) {

            return;

        }

        int[] maxProfit = new int[n];

        List<List<Integer>> tasks = new ArrayList<>();

        tasks.add(new ArrayList<>());

        maxProfit[0] = indexedJobs.get(0)[3];

        tasks.get(0).add(0);

        for (int i = 1; i < n; i++) {

            int index = findLastNonConflictingJob(indexedJobs, i);

            int currentProfit = indexedJobs.get(i)[3];

            if (index != -1) {

                currentProfit += maxProfit[index];

            }

            if (maxProfit[i - 1] < currentProfit) {

                maxProfit[i] = currentProfit;

                tasks.add(index != -1 ? new ArrayList<>(tasks.get(index)) : new ArrayList<>());

                tasks.get(i).add(i);

            } else {

                maxProfit[i] = maxProfit[i - 1];

                tasks.add(new ArrayList<>(tasks.get(i - 1)));

            }

        }

        int totalProfit = maxProfit[n - 1];

        System.out.println("WEIGHTED JOB SCHEDULING");

        System.out.println("-----------------------");

        System.out.println("Job | Start | Finish | Profit");

        System.out.println("----------------------------");

        List<Integer> selectedJobs = tasks.get(n - 1);

        for (int i : selectedJobs) {

            System.out.printf("%-4d | %-5d | %-6d | %-6d\n", indexedJobs.get(i)[0] + 1, indexedJobs.get(i)[1], indexedJobs.get(i)[2], indexedJobs.get(i)[3]);

        }

        System.out.println("----------------------------");

        System.out.println("Total Profit: " + totalProfit);

    }

    public static void main(String[] args) {

        List<Job> jobs = Arrays.asList(

            new Job(0, 6, 60),

            new Job(1, 4, 30),

            new Job(3, 5, 10),

            new Job(5, 7, 30),

            new Job(5, 9, 50),

            new Job(7, 8, 10)

        );

        findMaxProfitJobs(jobs);

    }

}

1. **Job Sequencing Problem**

A white background with black text

Description automatically generated

[Code zip file](file:///C:\Users\arino\Downloads\jobSequencing.zip)

Code:

package jobsequencing;

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

import java.util.stream.Collectors;

class Job

{

    public int taskId, deadline, profit;

    public Job(int taskId, int deadline, int profit)

    {

        this.taskId = taskId;

        this.deadline = deadline;

        this.profit = profit;

    }

}

class JobSequencing

{

    // Function to schedule jobs to maximize profit

    public static void scheduleJobs(List<Job> jobs, int T)

    {

        // stores the maximum profit that can be earned by scheduling jobs

        int profit = 0;

        // array to store used and unused slots info

        int[] slot = new int[T];

        Arrays.fill(slot, -1);

        // arrange the jobs in decreasing order of their profits

        Collections.sort(jobs, (a, b) -> b.profit - a.profit);

        // consider each job in decreasing order of their profits

        for (Job job: jobs)

        {

            // search for the next free slot and map the task to that slot

            for (int j = job.deadline - 1; j >= 0; j--)

            {

                if (j < T && slot[j] == -1)

                {

                    slot[j] = job.taskId;

                    profit += job.profit;

                    break;

                }

            }

        }

        // print the scheduled jobs

        System.out.println("The scheduled jobs are " +

                Arrays.stream(slot).filter(val -> val != -1).boxed()

                        .collect(Collectors.toList()));

        // print total profit that can be earned

        System.out.println("The total profit earned is " + profit);

    }

    public static void main(String[] args) {

    {

        List<Job> jobs = Arrays.asList(

                new Job(1, 9, 15), new Job(2, 2, 2), new Job(3, 5, 18),

                new Job(4, 7, 1), new Job(5, 4, 25), new Job(6, 2, 20),

                new Job(7, 5, 8), new Job(8, 7, 10), new Job(9, 4, 12),

                new Job(10, 3, 5));

        final int T = 15;

        scheduleJobs(jobs, T);

    }

    }

}

1. **Hungarian Algorithm**

A screenshot of a computer

Description automatically generated

[Code zip file](file:///C:\Users\arino\Downloads\Test.zip)

Code:

package test;

import java.util.Arrays;

import java.util.LinkedHashSet;

import java.util.Set;

public class HungarianAlgorithm {

int[][] matrix;

int[] squareInRow, squareInCol, rowIsCovered, colIsCovered, staredZeroesInRow;

public HungarianAlgorithm(int[][] matrix) {

if (matrix.length != matrix[0].length) {

try {

throw new IllegalAccessException("The matrix is not square!");

} catch (IllegalAccessException ex) {

System.err.println(ex);

System.exit(1);

}

}

this.matrix = matrix;

squareInRow = new int[matrix.length];

squareInCol = new int[matrix[0].length];

rowIsCovered = new int[matrix.length];

colIsCovered = new int[matrix[0].length];

staredZeroesInRow = new int[matrix.length];

Arrays.fill(staredZeroesInRow, -1);

Arrays.fill(squareInRow, -1);

Arrays.fill(squareInCol, -1);

}

public int[][] findOptimalAssignment() {

step1();

step2();

step3();

while (!allColumnsAreCovered()) {

int[] mainZero = step4();

while (mainZero == null) {

step7();

mainZero = step4();

}

if (squareInRow[mainZero[0]] == -1) {

step6(mainZero);

step3();

} else {

rowIsCovered[mainZero[0]] = 1;

colIsCovered[squareInRow[mainZero[0]]] = 0;

step7();

}

}

int[][] optimalAssignment = new int[matrix.length][];

for (int i = 0; i < squareInCol.length; i++) {

optimalAssignment[i] = new int[]{i, squareInCol[i]};

}

return optimalAssignment;

}

private boolean allColumnsAreCovered() {

for (int i : colIsCovered) {

if (i == 0) {

return false;

}

}

return true;

}

private void step1() {

// rows

for (int i = 0; i < matrix.length; i++) {

int currentRowMin = Integer.MAX\_VALUE;

for (int j = 0; j < matrix[i].length; j++) {

if (matrix[i][j] < currentRowMin) {

currentRowMin = matrix[i][j];

}

}

for (int k = 0; k < matrix[i].length; k++) {

matrix[i][k] -= currentRowMin;

}

}

for (int i = 0; i < matrix[0].length; i++) {

int currentColMin = Integer.MAX\_VALUE;

for (int j = 0; j < matrix.length; j++) {

if (matrix[j][i] < currentColMin) {

currentColMin = matrix[j][i];

}

}

for (int k = 0; k < matrix.length; k++) {

matrix[k][i] -= currentColMin;

}

}

}

private void step2() {

int[] rowHasSquare = new int[matrix.length];

int[] colHasSquare = new int[matrix[0].length];

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix.length; j++) {

if (matrix[i][j] == 0 && rowHasSquare[i] == 0 && colHasSquare[j] == 0) {

rowHasSquare[i] = 1;

colHasSquare[j] = 1;

squareInRow[i] = j;

squareInCol[j] = i;

continue;

}

}

}

}

private void step3() {

for (int i = 0; i < squareInCol.length; i++) {

colIsCovered[i] = squareInCol[i] != -1 ? 1 : 0;

}

}

private void step7() {

int minUncoveredValue = Integer.MAX\_VALUE;

for (int i = 0; i < matrix.length; i++) {

if (rowIsCovered[i] == 1) {

continue;

}

for (int j = 0; j < matrix[0].length; j++) {

if (colIsCovered[j] == 0 && matrix[i][j] < minUncoveredValue) {

minUncoveredValue = matrix[i][j];

}

}

}

if (minUncoveredValue > 0) {

for (int i = 0; i < matrix.length; i++) {

for (int j = 0; j < matrix[0].length; j++) {

if (rowIsCovered[i] == 1 && colIsCovered[j] == 1) {

matrix[i][j] += minUncoveredValue;

} else if (rowIsCovered[i] == 0 && colIsCovered[j] == 0) {

matrix[i][j] -= minUncoveredValue;

}

}

}

}

}

private int[] step4() {

for (int i = 0; i < matrix.length; i++) {

if (rowIsCovered[i] == 0) {

for (int j = 0; j < matrix[i].length; j++) {

if (matrix[i][j] == 0 && colIsCovered[j] == 0) {

staredZeroesInRow[i] = j; // mark as 0\*

return new int[]{i, j};

}

}

}

}

return null;

}

private void step6(int[] mainZero) {

int i = mainZero[0];

int j = mainZero[1];

Set<int[]> K = new LinkedHashSet<>();

K.add(mainZero);

boolean found = false;

do {

if (squareInCol[j] != -1) {

K.add(new int[]{squareInCol[j], j});

found = true;

} else {

found = false;

}

if (!found) {

break;

}

i = squareInCol[j];

j = staredZeroesInRow[i];

if (j != -1) {

K.add(new int[]{i, j});

found = true;

} else {

found = false;

}

} while (found);

for (int[] zero : K) {

if (squareInCol[zero[1]] == zero[0]) {

squareInCol[zero[1]] = -1;

squareInRow[zero[0]] = -1;

}

if (staredZeroesInRow[zero[0]] == zero[1]) {

squareInRow[zero[0]] = zero[1];

squareInCol[zero[1]] = zero[0];

}

}

Arrays.fill(staredZeroesInRow, -1);

Arrays.fill(rowIsCovered, 0);

Arrays.fill(colIsCovered, 0);

}

}

package test;

public class Test {

public static void main(String[] args) {

int[][] costMatrix = {

{70, 40, 20, 55},

{65, 60, 45, 90},

{30, 45, 50, 75},

{25, 30, 55, 40}

};

int[][] matrixCopy = new int[costMatrix.length][costMatrix[0].length];

for (int i = 0; i < costMatrix.length; i++) {

System.arraycopy(costMatrix[i], 0, matrixCopy[i], 0, costMatrix[i].length);

}

String[] taskList = {

"Clean the Bathroom",

"Wash the Dishes",

"Sweep the Floor",

"Wash the Windows"

};

String[] personList = {

"Roland",

"Eric",

"Marian",

"Lalaine"

};

HungarianAlgorithm hungarian = new HungarianAlgorithm(matrixCopy);

int[][] optimalAssignment = hungarian.findOptimalAssignment();

int totalExpense = 0;

String[] resultOutput = new String[taskList.length];

if (optimalAssignment.length > 0) {

for (int i = 0; i < optimalAssignment.length; i++) {

int worker = optimalAssignment[i][0];

int task = optimalAssignment[i][1];

int cost = costMatrix[worker][task];

totalExpense += cost;

resultOutput[task] = taskList[task] + " (" + cost + ") = " + personList[worker];

}

} else {

System.out.println("No assignment found!");

}

for (String result : resultOutput) {

System.out.println(result);

}

System.out.println("Total Expense: " + totalExpense);

}

}

1. **Face Recognition**

[Screen recording link](https://drive.google.com/file/d/1cCugQ29U8LjesBua1RsznikQOXLrZx5F/view?usp=sharing)

[Code zip file](file:///C:\Users\arino\Downloads\FaceRec_Luxand_SDK_6.1.0.0.zip)

1. **Apriori**

A screenshot of a spreadsheet

Description automatically generated

[APRIORI.xlsx file](file:///C:\Users\arino\Downloads\APRIORI.xlsx)

1. Hand Gesture 1

[Screen recording link](file:///C:\Users\arino\Downloads\Screen%20Recording%20-%20Made%20with%20FlexClip%20(1).webm)

[Code zip file](file:///C:\Users\arino\Downloads\HandGesturree1.zip)

1. Hand Gesture 2

[Screen recording link](https://drive.google.com/drive/folders/1bdSD8tfv2EzRyZ_e2wKvHsCYUrCBulE-?usp=sharing)

[Code zip file](file:///C:\Users\arino\Downloads\facetouchmonitor-master.zip)

10.) Voice Recognition

[Screen recording link](file:///C:\Users\arino\Downloads\Screen%20Recording%20-%20Made%20with%20FlexClip%20(2).webm)

[Code zip file](file:///C:\Users\arino\Downloads\VoiceRecognition.zip)