**Homework 3 – Lincoln Steber**

Chart, waterfall chart

Description automatically generated

* 1. Multi-Stage Decision Problem:

State Variables:

* t: Current time
* Jt: Jobs with start time <= t that have yet to be considered
* Vt: Max value that can be obtained up to time t

Decision Variables:

* xti: Decide to include job i at time t or not

State Transitions:

* t -> t + 1: Time as it increases by 1
* Jt -> Jt + 1: Remove job if finish time <= t + 1
* Vt -> Vt + 1: Increments as time increases by 1

Bellman Equation:

Vt + 1 = max {Vt + ∑ vi \* xti +1}

* 1. Pseudocode for Dynamic Approach:

task\_selection\_dynamic(jobs):

Sort jobs by increasing order of finish times.

n = length(jobs)

Create an array V of size n+1 to hold the maximum value of the subproblem.

Create a list of empty lists to hold the optimal jobs for each subproblem.

For i from 1 to n:

Set j to i-1.

while j > 0 and finish time of job (j-1) > start time of job i:

Decrement j by 1.

Compute the value vi + V[j].

If the value computed in the previous step is greater than V[i-1]:

Set V[i] to the maximum value computed in the previous step and V[j].

Set the list of optimal jobs for the current subproblem to the list of optimal jobs for subproblem j plus job i (i.e., best\_jobs[i] = best\_jobs[j] + [job i]).

Else:

Set V[i] to V[i-1].

Set the list of optimal jobs for the current subproblem to the list of optimal jobs for the previous subproblem (i.e., best\_jobs[i] = best\_jobs[i-1]).

Return(best\_jobs[n]).

* 1. Pseudocode for Greedy Approach:

task\_selection\_greedy(jobs):

Sort jobs by increasing order of start times.

Create an empty list greedy\_jobs to hold the selected jobs.

For each job in sorted jobs:

If the list of selected jobs is empty or the start time of the current job is greater than or equal to the finish time of the last job in the list of selected jobs:

Add the current job to the list of selected jobs.

Return the list of selected jobs.

Graphical user interface, text, application

Description automatically generated

* 1. Multi-Stage Decision Problem:

State Variables:

* Sij: State at j where the first string has been edited to i and the second string has been edited to position j

Decision Variables:

* Xj: Chosen operation for position j

State Transitions:

* Sij -> Sij + 1 and D(Sij + 1) = 0: If the string at the current state match the cost is 0
* Sij -> Sij + 1 and D(Sij + 1) = D(Sij) + Cij: If a character from the first string is substituted with a character from the second the cost is Cij
* Sij -> Sij + 1 and D(Sij + 1) = D(Sij) + Di: If a character is deleted from the first string the cost is Di
* Sij -> Sij + 1 and D(Sij + 1) = D(Sij) + Ij: If a character is deleted from the first string the cost is Di

Bellman Equation:

D(Si,j) = min {D(Si,j-1) + Ij, D(Si-1,j) + Di, D(Si-1,j-1) + Cij}

* 1. Pseudocode for Dynamic Approach:

string\_edit\_dynamic(s1, s2):

c = create 2D array with dimensions (len(s1) + 1) x (len(s2) + 1) and initialize all values to 0

for i = 0 to len(s1):

c[i][0] = i

for j = 0 to len(s2):

c[0][j] = j

for i = 1 to len(s1):

for j = 1 to len(s2):

if s1[i - 1] == s2[j - 1]:

c[i][j] = c[i - 1][j - 1]

else:

substitute = c[i - 1][j - 1] + 1

delete = c[i - 1][j] + 1

insert = c[i][j - 1] + 1

c[i][j] = min(substitute, delete, insert)

return c[len(s1)][len(s2)]

* 1. Pseudocode for Greedy Approach:

string\_edit\_greedy(s1, s2):

c = 0

i = 0

j = 0

while i < len(s1) or j < len(s2):

if i == len(s1):

c += len(s2) - j

break

else if j == len(s2):

c += len(s1) - i

break

if s1[i] == s2[j]:

i += 1

j += 1

else:

substitute = 1

delete = 1

insert = 1

if i < len(s1) - 1 and j < len(s2) - 1 and s1[i + 1] == s2[j] and s1[i] == s2[j + 1]:

substitute = 2

if j < len(s2) - 1 and s1[i] == s2[j + 1]:

insert = 2

if i < len(s1) - 1 and s1[i + 1] == s2[j]:

delete = 2

if substitute <= delete and substitute <= insert:

c += substitute

i += 1

j += 1

else if delete <= insert:

c += delete

i += 1

else:

c += insert

j += 1

return c

Graphical user interface, text, application

Description automatically generated

3) See python code

Text

Description automatically generated

4) Pseudocode for Polygon Triangulation:

polygon\_triangulation(end\_points):

n = len(end\_points)

A = [0] \* (n \* n)

for i in range(1, n – 1):

for j in range(1, n – i):

m = i + j

for k in range(j + 1, m – 1):

cost = A[j][m] + A[m][m] +distance(end\_points [j], end\_points [k] + distance(end\_points [k], end\_points [m])

A[j][m] = min(A[j][m], cost)

return A[1][n]

distance(p1, p2):

x1, y1 = p1

x2, y2 = p2

return sqrt((x2 – x1)^2 – (y2 – y1))