Justin Cabral

CS 5084

Assignment #6

Dynamic Programming

Text, letter

Description automatically generated

* Let us consider a graph that has 3 vertices, where the 1st and 3rd vertices have a weight W and the second has a weight W + 1.

A picture containing text, clock, watch

Description automatically generated

* In this case, the algorithm will select the heaviest node first (W + 1) and then delete both adjacent vertices. The output will be the independent set {W+1} when the maximum weighted independent set should be {W, W}. (2W > W + 1)

Text

Description automatically generated

A picture containing text, clock

Description automatically generated

* Consider the graph from figure 6.28
* S1 = {1,6,6} = 13 > S2 = {8, 3} = 11
* The algorithm will output S1, but the independent set should be {8, 4} = 14

Graphical user interface, text, application

Description automatically generated

Max\_Indpendent\_Set (vertrices v1, … , vn with weights w1, …, wn) {

A = array of weights indexed from 0 to n

B = array of sets indexed from 0 to n

A[0] = 0

A[1] = w1

B[0] = {}

B[2] = {v1}

For i = 2 to n

A\_i\_with = wi + a[i-2]

A\_i\_without = a[i=1]

If (A\_i\_with > A\_i\_without)

A[i] = A\_i\_with

B[i] = B[i-2] U {vi}

Else

A[i] = A\_i\_without

B[i[ = B[i-1]

Return B\_n

A page of a book

Description automatically generated with low confidence

Text

Description automatically generated with medium confidence

The algorithm given is short-sighted, meaning that it might take a high-stress job too early and then an even better one later.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Week 1 | Week 2 | Week 3 |
| L | 2 | 2 | 2 |
| h | 1 | 5 | 10 |

In this case the algorithm in (a) would take the high-stress job in week 2 when the best solution would take the low-stress job in week 1, nothing in week 2, and then take the high-stress job in week 3.



Optimal\_Job({L1, L2, …, Ln}, {h1, h2, …, hn})

OptimalArray = array of index 0 to n

OptimalArray[0] = 0

OptimalArray[1] = max(L1, h1)

For i = 2 to n

OptimalArray = max(Li + OptimalArray[i-1], hi + OptimalArray[i-2])

Return OptimalArray[n]

Text, letter

Description automatically generated

Diagram

Description automatically generated

A picture containing text, clock

Description automatically generated

* In this example the algorithm will take edge (v1, v2) and (v2, v5).
* However, the correct answer for the longest path is (v1, v3), (v3,v4), (v4,v5)

Text

Description automatically generated

Longest\_Path(v1,v1, …, vn)

OptimalArray = array of indexes from 1 to n

OptimalArray[1] = 0

For i = 2 to n

For all edges (vi, vj)

OptimalArray[i] = max(OptimalArray[j] + 1) where j > I and vi and vj are connected

Return OptimalArray[n]

Text

Description automatically generated

Text, table

Description automatically generated

N = 3, M = 10

|  |  |  |  |
| --- | --- | --- | --- |
|  | Month 1 | Month 2 | Month 3 |
| NY | 5 | 10 | 5 |
| SF | 10 | 5 | 10 |

The plan of minimum costs in this example would be the sequence of locations {NY, SF, NY} which has a cost of 5 + 5 + 5 + 10 + 10 = 35. If we stayed in NY for all 3 months then the optimal cost would be {NY, NY, NY} which has a cost of 5 + 10 + 5 = 20. Therefore, 20 < 35.

Text

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Month 1 | Month 2 | Month 3 | Month 4 |
| NY | 10 | 100 | 10 | 100 |
| SF | 100 | 10 | 100 | 10 |

The optimal plan {NY, SF, NY, SF} is 3 location moves. The reason being that the difference in operating costs is so big that even after paying the moving cost every time, we arrive at a cheaper overall total cost.

A picture containing text

Description automatically generated

Current\_city = (empty)

If NY[1] < SF[1]

Current\_city = NY

Else

Current\_city = SF

For i = 2 to n

If Current\_city == SF

If NY[i] + M < SF[i]

Current\_city = NY

Output = NY in month i

Else

Output = SF in month i

Else

If SF[i] + M < NY[i]

Current\_city = SF

Output = SF in Month i

Else

Output = NY in Month i

Text, letter

Description automatically generated

Consider a sequence 1,4,2,3. In this case, the greedy algorithm produces the rising trend of 1,4, while the optimal solution is 1,2,3.



Longest\_RT(p1,p2, … , pn)

OptimalArray = array of indexes 1 to n

OptimalArray[n] = 1

For i = n - 1 to 1

OptimalArray[i] = 1

For j = i + 1 to n

If pj > pi && OptimalARray[j] + 1 > OptimalArray[i]

OptimalArray[i] = OptimalArray[j] + 1

Return OptimalArray[1]

Text

Description automatically generated

Let J be the optimal subset of jobs wherein all scheduling jobs J can be scheduled to satisfy their deadlines. By the definition of J, we know that the minimal lateness is zero. This means that all jobs can be scheduled to finish by their deadlines. Thus, according to the Greedy Idea #4 which was proven to minimize/maximize the lateness, ordering jobs by increasing deadlines is optimal for min/maxing lateness. Then an ordering of jobs in J by this method would give us a schedule for this subset that allows each job to finish by the deadline. Therefore, there exists an optimal solution J.

Text

Description automatically generated

Sched\_Jobs({j1, j2, …, jn}), D) {

M = Array [0….n, 0…D]

Initialize M[0,d] = 0 for each d = 0,1, …, D

For i = 1 to in

For d, d0 to dn

If M[i-1, d] > M[i-1, d – ti ] + 1

M[i, d] = M[i-1, d - ti]

Else

M[i,d] = M[i-1, d - ti] + 1

Return M[n, dn]