Assignment 11

R-5.1 Let S = {a, b, c, d, e, f, g} be a collection of objects with benefit-weight values as follows: a:(12,4), b:(10,6), c:(8,5), d:(11,7), e:(14,3), f:(7,1), g:(9,6). What is an optimal solution to the fractional knapsack problem for S assuming we have a knapsack that can hold objects with total weight 15? Show your work.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | S | a(12,4) | b(10,6) | c(8,5) | d(11,7) | e(14,3) | f(7,1) | g(9,6) |
|  | b | 12 | 10 | 8 | 11 | 14 | 7 | 9 |
|  | w | 4 | 6 | 5 | 7 | 3 | 1 | 6 |
|  | b/w | 3 | 1.666667 | 1.6 | 1.571429 | 4.666667 | 7 | 1.5 |
| **Knapsack** |  |  |  |  |  |  |  |  |
| 15 | Ratio | 1 | 1 | 0.2 | 0 | 1 | 1 | 0 |
| 0 | w | 4 | 6 | 1 | 0 | 3 | 1 | 0 |
| 44.6 | b | 12 | 10 | 1.6 | 0 | 14 | 7 | 0 |

Optional Solution = {a(12,4), b(10,6), c(0.2,1), e(14,3), f(7,1)}

Total benefits: 44.6

R-5.3 Suppose we are given a set of tasks specified by pairs of the start times and finish times as T = {(1,2),(1,3),(1,4),(2,5),(3,7),(4,9),(5,6),(6,8),(7,9)}. Solve the task scheduling problem for this set of tasks.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Task** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** |
| **(1,2)** | **(1,3)** | **(1,4)** | **(2,5)** | **(3,7)** | **(4,9)** | **(5,6)** | **(6,8)** | **(7,9)** |
| **Start** | 1 | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **Finish** | 2 | 3 | 4 | 5 | 7 | 9 | 6 | 8 | 9 |
|  |  |  |  |  |  |  |  |  |  |
| **Time** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **M1** | A | D | | | G | H | |  |  |
| **M2** | B | | E | | | | I | |  |
| **M3** | C | | | F | | | | |  |

R-5-11 Solve Exercise R-5.1 above for the 0-1 Knapsack Problem.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Weight** | | | | | | | | | | | | | | | |
| W | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| (0, 0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| a(12,4) | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 |
| b(10,6) | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 12 | 22 | 22 | 22 | 22 | 22 | 22 |
| c(8,5) | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 20 | 22 | 22 | 22 | 22 | 22 | 30 |
| d(11,7) | 0 | 0 | 0 | 0 | 12 | 12 | 12 | 12 | 12 | 20 | 22 | 23 | 23 | 23 | 23 | 30 |
| e(14,3) | 0 | 0 | 0 | 14 | 14 | 14 | 14 | 26 | 26 | 26 | 26 | 26 | 34 | 36 | 37 | 37 |
| f(7,1) | 0 | 7 | 7 | 14 | 21 | 21 | 21 | 26 | 33 | 33 | 33 | 33 | 34 | 41 | 43 | 44 |
| g(9,6) | 0 | 7 | 7 | 14 | 21 | 21 | 21 | 26 | 33 | 33 | 33 | 33 | 34 | 41 | 43 | 44 |

The result is (7,1), (14,3), (11,7), (12,4), the maximum benefits are 7 + 14 + 11 + 12 = 44

R-5-12 Sally is hosting an Internet auction to sell n widgets. She receives m bids, each of the form “I want ki widgets for di dollars,” for i = 1, 2, …, m. Characterize her optimization problem as a knapsack problem. Under what conditions is this a 0-1 versus fractional problem?

Answer: Here we can't split the widgets into partial part. That's why it is not fractional knapsack problem. So this is should be 0-1 fractional problem.

Do the following on paper and hand in next week:

A. Based on the characterizing equations only, give a pseudo code recursive algorithm for the 0-1 knapsack problem (try to do this without looking at my solution in the notes), then memoize it so it is efficient. Compare your algorithm to the two given in the lecture notes (iterative dynamic programming version and recursive non-memoized algorithm) in terms of time and space complexity.

|  |  |
| --- | --- |
| Algorithm Knapsack01(S, maxW)  M <- Array[S.size(), maxW + 1] // 0<=i<=n, 0<=j<=maxW  for w <- 0 to maxW+1 do  M[0, w] <- 0  for k <- 0 to S.size() do  M[k, 0] <- 0    //k <- 0  for k <- 1 to S.size() do  processKnapsack(S, k, S.elementAtRank(0).w)  Algorithm processKnapsack(S, k, wR)  if k = 0 \/ w = 0 do  return M[k, w]    if M[k, wR] > 0 then  return M[k, wR]    bk <- S.elementAtRank(k-1).benefit()  wk <- S.elementAtRank(k-1).weight()    for w <- 1 to maxW do  if w < wk then  M[k, w] = processKnapsack(S, k-1, w) //M[k-1,w]  else  M[k, w] = max(processKnapsack(S, k-1, w), processKnapsack(S, k-1, w-wk) + bk)    return M[k, wR] | O(n + MaxW)  maxW  n  n  maxW  maxW  maxW  maxW  1 |

Total running time is O(n + maxW), maxW is the maximum weight of the Knapsack

C-5.9 How can we modify the dynamic programming algorithm from simply computing the best benefit value for the 0-1 knapsack problem (like A) to computing the assignment (subset) that gives the maximum benefit? Design a pseudo code algorithm.

Answer: In the pseudo code at A. above, R is subset of items with the maximum benefit.

B. Suppose we have a set of objects that have different sizes s1, s2, …, sn, and we have some positive upper limit L. Design an efficient pseudo code algorithm to determine the subset of objects that produces the largest sum of sizes that is no greater than L. Hint: dynamic programming similar to 0-1 knapsack problem, except only size/weight and no benefit

|  |
| --- |
| Algorithm Knapsack01(S, L)  n <- S.size()  M <- Array[i, j], 0<=i<=n, 0<=j<=L  m[k, l] <- 0, k=0, 0<=l<=L  m[k, l] <- 0, 0<=k<=n, l=0    r <- 0  for each (b,w) in S do  r++  for c<-1 to L do  if w > c then  M[r,c] <- M[r-1, c]  else  lw <- c - w  M[r, c] <- max(M[r-1, c], M[r-1, lw] + w)    leftW <- maxW  R <- new Sequence  for i <- n down to 1 do  if leftW > 0 then  (b,w) <- S.elementAtRank(i-1)  if M[i, leftW] <> M[i-1, leftW] do  leftW <- leftW - w  R.insertItem((b,w))  return R |