Anu Varshneya

Homework 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. (4.2a)   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **S** | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | | **0** | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | | **1** | 0 | 7 | ∞ | 6 | ∞ | 6 | 5 | ∞ | ∞ | ∞ | | **2** | 0 | 7 | 11 | 5 | 7 | 6 | 4 | ∞ | 9 | ∞ | | **3** | 0 | 7 | 11 | 5 | 7 | 6 | 4 | 9 | 7 | ∞ | | **4** | 0 | 7 | 11 | 5 | 7 | 6 | 4 | 8 | 7 | 7 | |
| 2. (4.8)  Professor F’s solution is not valid. Considering the following as a counter example:    In this example, the path from A🡪C is -6, and the path from A🡪B🡪C is -7. Therefore, the path from A🡪B🡪C is clearly the shortest path. If we follow Professor F’s solution, we would add 7 to every edge so that A🡪B = 4, B🡪C = 3, and A🡪C = 1. Once 7 is added to every edge, the path from A🡪C is 1 and A🡪B🡪C = 7, meaning that the straight path from A🡪C is the shortest, but we know this not to be true. Therefore, Professor F’s solution is incorrect. |
| 3. (4.17a)  Understood and adapted from <http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm> and <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/dijkstraAlgor.htm>.  In this example, every edge has a length between 0 and W. Because any shortest path contains at most |V|-1 edges, where V is the number of vertices, all distance values will be in the range {0, 1,…, W(|V|-1), infinity}.Therefore, we can maintain an array size of W(|V|-1)+2 in a heap, and index all possible values of distance where each entry is a pointer to a linked list of elements with all distance values equal to that entry. If this is done, we can perform insert operations in constant time by appending the beginning of the linked list that corresponds to its value. Therefore, the makeheap function outlined in the textbook will only take O(|V|). Then, when we perform the deletemin function, instead of scanning the entire heap for the next smallest value, we can start looking at the previous minimum value. This means that we will look at each value in the array once at most, so all the deletemin functions will only take O(W|V|). The decreasekey function can be implemented such that any new element is being added, it can be added to the list corresponding its value without deleting any duplicates that may already exist. This alters the deletemin function because now that function must check whtehr the current minimum is a coy of an element that has already been processed, and in that case, to ignore it. Because decreasekey will run at most |E| times, where E signifies the edges, there are at most |E| copties we need to ignore, so all decreasekey runs will take O(|E|). This implementation can therefore be described with: O(|V|) + O(W|V|) + O(|E|) = O(W|V| + |E|), which is what the problem asked for. |
| 4. (5.2)  a)   |  |  |  | | --- | --- | --- | | **Edge Included** | **Intermediate Value** | **Cost** | | AB | 1 | 1 | | BC | 2 | 3 | | CG | 2 | 5 | | GD | 1 | 6 | | GH | 1 | 7 | | GF | 1 | 8 | | AE | 4 | 12 |   b)   |  |  | | --- | --- | | Edges | Distance | | AB | 1 | | GD | 1 | | GH | 1 | | GF | 1 | | BC | 2 | | CG | 2 | | CD | 3 | | DH | 4 | | AE | 4 | | EF | 5 | | BF | 6 | | BG | 6 | | AF | 8 |  |  | | --- | | Step 1: | | Step 2: | | Step 3: | | Step 4: | |
| 5. Subset Sum Algorithm  x = [1,4,6,8,5]  target\_sum = 16  A = 0  B = 0  for i in range(len(x)):  if (x[i] < 0):  A += x[i]  else:  B += x[i]  N = len(x);  #initialize intermediate results array with initial values  prevQvalues = {}  for j in range(A,B+1):  prevQvalues[(0,j)] = (x[0] == j)  #this computes values of "is there a subset of length i, which sums to s"  #Q(i,s) := Q(i-1,s) || (x[i] == s) || Q(i-1,s-x[i])  def Q(i,s):  if ((s<A) or (s>B)):  return False  if (((s-x[i]) < A) or ((s-x[i]) > B)):  res = prevQvalues[(i-1,s)] or (x[i] == s) # skip Q(i-1,s-x[i]) as it is False  else:  res = prevQvalues[(i-1,s)] or (x[i] == s) or prevQvalues[(i-1),(s-x[i])]  return res  for i in range(1,N):  for j in range(A,B+1):  prevQvalues[(i,j)] = Q(i,j)  print "Given list = ", x, ", Target sum = ", target\_sum  print "... and the result is"  print Q(N-1,target\_sum) |
| 6. Kirkman Schoolgirl Problem |