Some ideas:

Dynamic programming is an optimization technique.

Greedy vs. Dynamic Programming :

* Both techniques are optimization techniques, and both build solutions from a collection of choices of individual elements.
* The greedy method computes its solution by making its choices in a serial forward fashion, never looking back or revising previous choices.
* Dynamic programming computes its solution bottom up by synthesizing them from smaller subsolutions, and by trying many possibilities and choices before it arrives at the optimal set of choices.
* There is no a priori litmus test by which one can tell if the Greedy method will lead to an optimal solution.
* By contrast, there is a litmus test for Dynamic Programming, called [The Principle of Optimality](http://www.seas.gwu.edu/~ayoussef/cs212/dynamicprog.html#poo)

Divide and Conquer vs. Dynamic Programming:

* Both techniques split their input into parts, find subsolutions to the parts, and synthesize larger solutions from smalled ones.
* Divide and Conquer splits its input at prespecified deterministic points (e.g., always in the middle)
* Dynamic Programming splits its input at every possible split points rather than at a pre-specified points. After trying all split points, it determines which split point is optimal.

1. Greedy method: A greedy algorithm is an [algorithm](http://en.wikipedia.org/wiki/Algorithm) that follows the [problem solving](http://en.wikipedia.org/wiki/Problem_solving) [heuristic](http://en.wikipedia.org/wiki/Heuristic_(computer_science)) of making the locally optimal choice at each stage with the hope of finding a global optimum.
2. Longest common subsequence (LCS) problem: The longest common subsequence (LCS) problem is to find the longest [subsequence](http://en.wikipedia.org/wiki/Subsequence) common to all sequences in a set of sequences.
3. Optimal substructure property: a problem is said to have optimal substructure if an optimal solution can be constructed efficiently from optimal solutions of its subproblems. This property is used to determine the usefulness of dynamic programming and greedy algorithms for a problem.
4. A [spanning tree](http://en.wikipedia.org/wiki/Spanning_tree_(mathematics)) of that graph is a [subgraph](http://en.wikipedia.org/wiki/Glossary_of_graph_theory#Subgraphs) that is a [tree](http://en.wikipedia.org/wiki/Tree_graph) and connects all the [vertices](http://en.wikipedia.org/wiki/Vertex_(graph_theory)) together. A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of minimum spanning trees for its [connected components](http://en.wikipedia.org/wiki/Connected_component_(graph_theory)).
5. Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblems. To solve a given problem, we need to solve different parts of the problem (subproblems), then combine the solutions of the subproblems to reach an overall solution.
6. To calculate C(n, k), the number of k-combinations (i.e., k-element subsets) of an n-element set. Use the formulas

C(n, k) = C(n - 1, k - 1) + C(n - 1, k)

valid for 1 <= k <= n - 1, and C(n, n) = 1 = C(n, 0) valid for n>= 0

k= 1 C(n,1)= C(n-1,1)

k =2 C(n,2)=C(n-1,1) + C(n-1,2)

k=3 C(n,3)=C(n-1,2) + C(n-1,3)

….

K=n-1 C(n,n-1)=C(n-1,n-2) + C(n-1,n-1)



Combine(n, k)

for i = 0 to n do

C[i, i] = 1

C[i, 0] = 1

for i = 2 to n do

for j = 1 to n – 1 do

C[i, j] = C[i – 1, j – 1] + C[i – 1, j]

return C[n, k]

Note that C[n, k] is a symmetric matrix, so we can modify code as follows:

Combine(n, k)

m = k

if (m > n / 2) then m= n / 2

for i = 0 to n do

C[i, 0] = 1

for i = 0 to m do

C[i, i] = 1

for i = 2 to n do

for j = 1 to n – 1 do

if (j <= m)

C[i, j] = C[i – 1, j – 1] + C[i – 1, j]

if (k > n – k) then

return C[n, n – k]

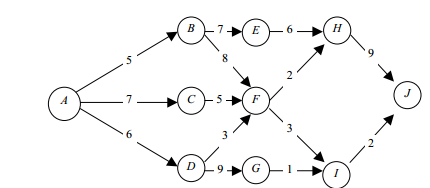
return C[n, k]

**(c) The worst-case time of this algorithm is O (n2)**

**(d) Trace your algorithm for C (7, 5):**



1. Shortest Path Problem:
2. Greedy method cannot work on Shortest Path Problem by given a counter example.



From A, by greedy algorithm, we found that going to B is the shortest at the moment, so we choose B. Similarly, we go to E, H then finally J. The total distance for choosing this path is 27.

Greedy algorithm is an easy and direct way to find out a solution, but it doesn’t promise the solution is necessarily optimal. For example, the shortest distance is actually 14 in the above case. The difference is there because greedy algorithm does not consider the consequence when choosing the best way to go. Therefore, greedy algorithm may not be a good choice for solving these kinds of questions.

b. We can use dynamic programming strategy to solve theproblem. We know A-D-F-I-J is the shortest way.Now, we find the shortest way from A to F, it is “ A-D-F”.Solving the big problem by dividing it into simpler subproblem and combine the subsolution to make general solution. That is, please explain why the problem satisfies the optimal substructure property.

c. We don’t use same strategy (dynamic programming) to solve longestPath Problem, which tries to find a longest simple path between two nodes in a graph? Why or Why not?

- The longest path problem, on the other hand, does not satisfy the Principle of Optimality. Take for example the undirected graph G of nodes a, b, c, d, and e, and edges (a,b) (b,c) (c,d) (d,e) and (e,a). That is, G is a ring. The longest (noncyclic) path from a to d to a,b,c,d. The sub-path from b to c on that path is simply the edge b,c. But that is not the longest path from b to c. Rather, b,a,e,d,c is the longest path. Thus, the subpath on a longest path is not necessarily a longest path.