Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?

Establish the problem.

- specifications – task, input, output, legal solution, optimal solution

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?

input: *n* cities and the distance *d*(*i*,*j*) between each pair of cities *i*,*j*

output: ordering of cities

legal solution: all cities included exactly once

optimization goal: shortest total distance

- examples

Identify avenues of attack.

- known targets
- approach

Series of choices.

- paradigms and patterns

Paradigm: dynamic programming

Flavor:

Pattern:

- the series of choices

Define the algorithm.

- size
- generalize / define subproblems
  - partial solution

the order of the cities visited so far

  - alternatives – for the next choice

any city not yet visited

  - subproblem – solve the rest of the problem, returning the solution for the rest of the problem (only)

Given a list of cities and the distances between each pair of cities, what is the shortest possible route from city *c* that visits each city in *C*?

input: ***n-k* cities remaining** and the distance *d*(*i*,*j*) between each pair of cities *i*,*j* , final destination city *d*, **current city *c***

output: ordering of cities

- memoization

D = total distance of a route

D[c][C] = shortest route from city *c* (index in the original list of cities) visiting all of the cities in C and returning to the destination

C is bit vector version of a set

- base case(s)

D[c][{}] = d(c,d)

- main case

D[c][C] = min over all cities c’ in C { D[c’][C \ { c’ }]+d(c,c’) }

- top level
  - initial subproblem
  - setup
  - wrapup
- special cases
- algorithm – determine the order of computation and write the loops

Show termination and correctness.

- termination
  - making progress
  - the end is reached
- correctness
  - establish the base case(s)
  - show the main case
  - final answer

Determine efficiency.

- implementation
- time and space

backtracking = n^n (<= n alternatives for the next city, n cities to order)

dyn prog = n^2 2^n

- room for improvement