Developing Algorithms

Strategies –

- realize your problem is another well-known problem in disguise
  - it is searching or sorting
  - there's a data structure for that
  - it is a graph problem
- develop a new algorithm
  - divide-and-conquer
  - iterative
  - series of choices – greedy, recursive backtracking, dynamic programming

Modeling With Graphs

Things to decide –

- what the vertices represent
  - vertices represent things
- what the edges represent
  - edges represent connections (relationships) between pairs of things
    - does the relationship simply exist or not exist? – indicated by presence or absence of edge
    - does the relationship have a strength? – modeled by edge weights
    - is the relationship one-way? – indicated by directed or undirected edges
- what a solution to the original problem looks like in the graph
  - ideally it is a concept for which there is a known algorithm
    - e.g. reachability, shortest path, minimum spanning tree, cut vertices

Modeling With Graphs

As you think about graph algorithms, identify –

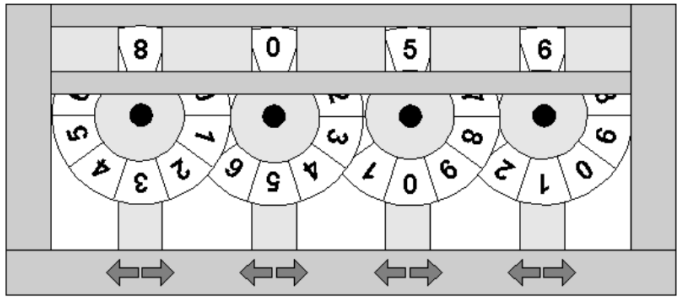
- the flavor of graph
  - connected or not connected
  - simple or containing self loops and/or multiedges
  - sparse vs dense
  - cyclic vs acyclic
  - weighted vs unweighted
    - if weighted, are there negative weights? negative weight cycles?
  - embedded vs topological
    - is there a particular ordering of incident edges for each vertex?
  - explicit vs implicit
    - does a graph data structure need to be built?

these properties are consequences of the modeling choices rather than independent decisions

they are important to recognize because they can affect the algorithms used and the choice of implementation for the graph structure (and other data structures)

e.g. representing an embedded graph would require an adjacency list implementation
e.g. (recursive) DFS doesn't require explicit graph structure, but BFS and Dijkstra's do (at least for vertices)

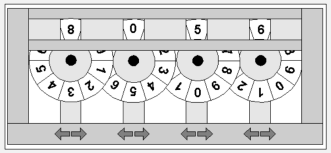
In this problem we will be considering a game played with four wheels. Digits ranging from 0 to 9 are printed consecutively (clockwise) on the periphery of each wheel. The topmost digits of the wheels form a four-digit integer. For example, in the following figure the wheels form the integer 8056. Each wheel has two buttons associated with it. Pressing the button marked with a *left arrow* rotates the wheel one digit in the clockwise direction and pressing the one marked with the *right arrow* rotates it by one digit in the opposite direction.



The game starts with an initial configuration of the wheels. Say, in the initial configuration the topmost digits form the integer $S_1S_2S_3S_4$. You will be given some (say, n) forbidden configurations $F_{i_1}F_{i_2}F_{i_3}F_{i_4}$ $(1 \leq i \leq n)$ and a target configuration $T_1T_2T_3T_4$. Your job will be to write a program that can calculate the minimum number of button presses required to transform the initial configuration to the target configuration by never passing through a forbidden one.

How to Model With Graphs

- the task is to find the minimum number of button presses to get from one configuration to another
  - a strategy is to find the shortest such sequence of configurations and then count the number of steps
    - → a sequence of configurations sounds like a path
    - vertices represent allowed configurations
      - no vertices for forbidden configurations
    - edges represent button presses
      - undirected because a button press is reversible



Mr. G. works as a tourist guide. His current assignment is to take some tourists from one city to another. Some two-way roads connect the cities. For each pair of neighboring cities there is a bus service that runs only between those two cities and uses the road that directly connects them. Each bus service has a limit on the maximum number of passengers it can carry. Mr. G. has a map showing the cities and the roads connecting them. He also has the information regarding each bus service. He understands that it may not always be possible for him to take all the tourists to the destination city in a single trip.

Adapting Algorithms

If you don't have an exact algorithm for the problem you need to solve, there are two options –

Option #1 – adapt an existing algorithm to work with your situation

Option #2 – build an instance of a known problem so that the solution produced by a standard algorithm solves your problem

Adapting Algorithms

A concrete example –

Find the shortest path from s to every other vertex in a graph with vertex weights.

- cost of path = sum of weights of vertices on the path

Option #1 – adapt Dijkstra's algorithm to work with vertex weights instead of edge weights.

- requires proving that the new algorithm is correct

Option #2 – transform the vertex-weighted graph into an edge-weighted graph so that the shortest paths in the edge-weighted graph have the same cost as the shortest paths in the vertex weighted graph.

- make the graph directed
  - every undirected edge (u,v) in the original graph will turn into two directed edges (u,v) and (v,u)
- add new start vertex s' with edge s' → s
- give edge i → j the weight of vertex j
- run Dijkstra's algorithm starting from s'

Adapting Algorithms

Advantages of option #2 –

- can be easier to argue why a solution in the new graph is equivalent to a solution in the original graph than to argue correctness of a new algorithm
- can use a library implementation of the algorithm instead of having to code your own variation

7-14. *[3]* Plum blossom poles are a Kung Fu training technique, consisting of n large posts partially sunk into the ground, with each pole p_i at position (x_i, y_i). Students practice martial arts techniques by stepping from the top of one pole to the top of another pole. In order to keep balance, each step must be more than d meters but less than $2d$ meters. Give an efficient algorithm to find a safe path from pole p_s to p_t if it exists.