Developing Algorithms

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
  - control-flow paradigms: iterative, recursive
  - solution-construction paradigms: decomposition, series of choices
  - design paradigms: divide-and-conquer, greedy, backtracking + branch-and-bound, 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 –

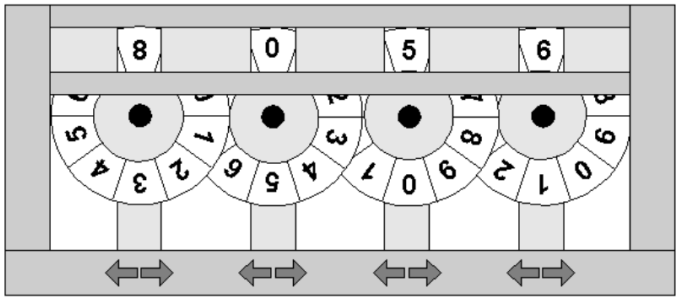
- the flavor of graph
  - connected or not connected
  - simple or containing self loops and/or multiedges
  - sparse vs dense
  - cyclic vs acyclic
  - weighted vs unweighted
    - if weighted, are there negative weights? negative weight cycles?
  - embedded vs topological
    - is there a particular 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)

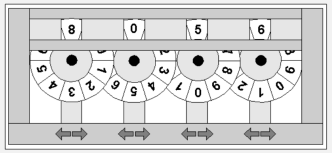
In this problem we will be considering a game played with four wheels. Digits ranging from 0 to 9 are printed consecutively (clockwise) on the periphery of each wheel. The topmost digits of the wheels form a four-digit integer. For example, in the following figure the wheels form the integer 8056. Each 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



Building a particular software package requires compiling a number of modules. In order to compile a module, any other modules that it depends on must be compiled first. Determine an order of compilation for the modules so that the whole package can be built successfully.

vertices?
→ modules

edges?
→ dependencies

directed or undirected?
→ directed – u → v means u depends on v (v must be compiled first)

edge weights?
→ no

what does a solution look like in the graph?
→ an ordering of vertices such that modules depended on are compiled before those that need them
→ topological sort

Red light cameras have been placed at strategic intersections around a city. An intersection is "strategic" if the only route between two points passes through that intersection. For example, if there are 6 intersections (A, B, C, D, E, F) and 7 streets connecting intersections (B-C, A-B, C-A, D-C, D-E, E-F, F-C), then there is a camera at C because the only route from A to E goes through C. Given a set of intersections and routes, find the speed cameras.

vertices?
→ intersections

edges?
→ roads between intersections

directed or undirected?
→ undirected

edge weights?
→ no

what does a solution look like in the graph?
→ a set of vertices such that the only route between some pair of vertices passes through that vertex
→ cut vertices

Electricity is finally coming to a (very) rural small town. Given the cost of stringing wire between every pair of houses in the town, determine how to connect the town with the smallest possible total cost.

vertices?
→ houses

edges?
→ between every pair of vertices – possible connection for wiring

directed or undirected?
→ undirected

edge weights?
→ cost of stringing wire between that pair of houses

what does a solution look like in the graph?
→ a set of edges connecting every vertex with minimum cost
→ minimum spanning tree

Within social networks, finding close-knit communities where everyone follows, directly or indirectly, everyone else in the group.

vertices?
→ members of the community

edges?
→ following relationship

directed or undirected?
→ directed – u → v means u follows v

edge weights?
→ no

what does a solution look like in the graph?
→ subsets of vertices where there is a path from every vertex to every other vertex within that subset
→ strongly connected components

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.

vertices?
→ cities

edges?
→ existence of a bus service between cities

directed or undirected?
→ undirected (two-way roads)

edge weights?
→ yes (bus capacity)

what does a solution look like in the graph?
→ routes from one city to another which together get all of the passengers to their destination without exceeding any bus capacities

algorithm?
→ flow network / max flow – add a source vertex *s* connected to the start city with edge capacity equal to the number of tourists, *t* Is the destination city

Modeling With Graphs

- what the solution looks like in the graph may not be a familiar algorithm

An *edit step* is a transformation from one word x to another word y such that x and y are words in the dictionary, and x can be transformed to y by adding, deleting, or changing one letter. So the transformation from *dig* to *dog* or from *dog* to *do* are both edit steps. An *edit step ladder* is a lexicographically ordered sequence of words $w_1, w_2, \ldots, w_n$ such that the transformation from w_i to w_{i+1} is an edit step for all i from 1 to $n - 1$.

For a given dictionary, you are to compute the length of the longest edit step ladder.

vertices?
→ words

edges?
→ words one edit step apart

directed or undirected?
→ directed because while both an action and its opposite are valid edit steps, an edit step ladder is a lexicographically ordered sequence and we only want edit steps that could be part of a ladder

edge weights?
→ no, each step counts as one

what does a solution look like in the graph?
→ longest path between two words

algorithm?
→ longest path in a DAG

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

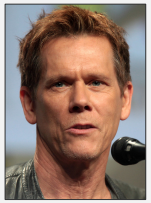
- make the graph directed
  - every undirected edge (u,v) in the original graph will turn into two directed edges (u,v) and (v,u)
- add new start vertex s' with edge s'→s
- give edge i→j the weight of vertex j
- run Dijkstra's algorithm starting from s'

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

Six Degrees of Kevin Bacon and Bacon Numbers



Six degrees of Kevin Bacon: find a way to link a particular actor to Kevin Bacon through movies they have in common.

The **Bacon number** of an actor is the number of degrees of separation he or she has from Bacon, as defined by the game. This is an application of the Erdős number concept to the Hollywood movie industry. The higher the Bacon number, the greater the separation from Kevin Bacon the actor is.

- Kevin Bacon himself has a Bacon number of 0.
- Those actors who have worked directly with Kevin Bacon have a Bacon number of 1.
- If the lowest Bacon number of any actor with whom X has appeared in any movie is N, X's Bacon number is N+1.

Examples [edit]

Elvis Presley:

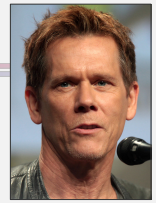
- Elvis Presley was in *Change of Habit* (1969) with Edward Asner
- Edward Asner was in *JFK* (1991) with Kevin Bacon

Therefore, Asner has a Bacon number of 1, and Presley (who never appeared in a film with Bacon) has a Bacon number of 2.

Ian McKellen:

- Ian McKellen was in *X-Men: Days of Future Past* (2014) with Michael Fassbender and James McAvoy
- McAvoy and Fassbender were in *X-Men: First Class* (2011) with Kevin Bacon

Therefore, McAvoy and Fassbender have Bacon numbers of 1, and McKellen has a Bacon number of 2.



Compute the Bacon number for an actor / determine if their Bacon number ≤ 6.

vertices?
→ actors and movies

edges?
→ between actor and movie if an actor is in that movie

directed or undirected?
→ undirected

edge weights?
→ no – graph construction requires two edges to connect two actors (actor→movie→actor) – while that only adds 1 to the Bacon number, the graph is bipartite so every path contains twice as many edges with respect to the Bacon number

what does a solution look like in the graph?
→ shortest path from Kevin Bacon to the actor

algorithm?
→ shortest unweighted path, with Bacon number = path length / 2

a graph with just actors as vertices – and edges connecting two actors if they were in a movie together – is an alternative but the number of edges is much larger (the k actors in movie M each have edges to all of the other k-1 actors in that movie, while in the bipartite graph version, there are k edges from actors in M to M and another k edges from M to actors in M – $2k$ vs $k(k\text{-}1)$