**Pieces in GalaxyTrip**

Start with ["...", "2 3", "1 4", "...", "2 5" ...]

- Machine 1 depends on 2 and 3
  - What if 2 depends on 1 and 4
  - And 4 depends on 2 and 5

- If we bring machine 1, what other machines do we need?
  - How big is this machine component?
  - How do we find the chain of dependencies?

- Once we find this connected component, we don’t care about the vertices in it anymore

**Another piece in GalaxyTrip**

- Supposed we have found all the connected components, accounting for all the vertices and we have these sizes:
  
  1, 3, 5

- We can bring one machine, or three, but how can we bring four? Six? Eight? Nine?
  - Bring component 1 and 3 means we bring 4 machines

- We need to calculate all sums generated by component sizes

**Simple All Sums**

- If we know we can generate these sums using components 0..k (e.g., the first k components) then what changes when we process the k+1\textsuperscript{st} component?
  
  1, 2, 3, 5

- If next component has size 4, then we can generate
  - 5+4, 3+4, 2+4, 1+4 or 9, 7, 6, 5
  - We can also bring 4 machines, the component itself

**All sums continued**

- We can keep a boolean array, if b[k] == true, we can bring k machines

- Have 1 and 2 set so far, what happens with new value of 3?
  - Now will have 5, 4, and 3 why?
  - What if instead of 3 we get 2, what values will be true?
Why do we have to go backwards?

process new value \(i\) given what we have so far:

\[
\text{for(int } k = \text{ok.length-1}; k >= 0; k--){
    \text{if (ok[k] && i+k < ok.length)}{
        \text{ok[i+k] = true;}
    }
    \text{ok[i] = true;}
}
\]

- If we go front to back with new value 2 we’ll see we can get 5 and 7 if we already have 3, not the case if we go back-to-front
  - Don’t want to include new values in calculation

Last piece, parsing the graph

- If we use adjacency list, what do we need?
  - What is a vertex? String or int?

\[
\text{for(int } k = 0; k < \text{dependencies.length}; k++){
    \text{String[]} \text{adj = dependencies[k].split(" ");}
    \text{graph.put("+k,adj");}
    \text{// or}
    \text{int[]} \text{iajd = new int[adj.length];}
    \text{// loop: iajd[0] = Integer.parseInt(adj[0]);}
    \text{graph.put(k,iajd);}
}
\]

DFS arrives


defs\langle Graph.\text{Vertex} \rangle \text{dfs(Graph.\text{Vertex} start)}{\n    \text{Set<Graph.\text{Vertex}> visited = new TreeSet<Graph.\text{Vertex}>();}
    \text{Stack<Graph.\text{Vertex}> qu = new Stack<Graph.\text{Vertex}>();}
    \text{visited.add(start);}
    \text{qu.push(start);}
    \text{while (qu.size() > 0)}{\n        \text{Graph.\text{Vertex} v = qu.pop();}
        \text{for(\text{Graph.\text{Vertex} adj : myGraph.getAdjacent(v)}){\n            if (! visited.contains(adj)) {\n                visited.add(adj);
                \text{qu.push(adj);}
            }
        }}
    }\n    \text{return visited;}
}

Putting everything together

- Create graph
  - Watch out for no-dependencies, adjacency list has nothing in it, but parsing ...

- Find all connected component sizes
  - DFS and a list of values

- Generate all sums
  - Use boolean ok idea, backwards

- Create array of values in which ok[k] == true
  - Return and done!