LTH Challenge solutions

A - Arrival time

Given a weighted graph with time tables on the edges, find the latest time t0 you have to leave your position to be able to arrive at latest at time T.

A - Arrival time

Given a weighted graph with time tables on the edges, find the latest time t0 you have to leave your position to be able to arrive at latest at time T.

- To slow: For each time t0, run Dijkstra and see if we can reach the end node before time T.

A - Arrival time

Given a weighted graph with time tables on the edges, find the latest time t0 you have to leave your position to be able to arrive at latest at time T.

- To slow: For each time t0, run Dijkstra and see if we can reach the end node before time T.
- On the limit: binary search over t0.

A - Arrival time

Given a weighted graph with time tables on the edges, find the latest time t0 you have to leave your position to be able to arrive at latest at time T.

- To slow: For each time t0, run Dijkstra and see if we can reach the end node before time T.
- On the limit: binary search over t0.
- Intended solution: Run a backwards Dijkstra from the end node “backwards in time”, inverting each time table: (u, v, P, t, d) -> (v, u, P, t + d, d).

B - Bus numbers

Given a number N, find the largest number X <= N,

such that X = a^3 + b^3 = c^3 + d^3 (a != c, a!= d)

Count the number of cube sum representations of x by looping over a, a^3 <= x

and check if x - a^3 is a cube. O(X*X^(⅓))

B - Bus numbers

Given a number N, find the largest number X <= N,

such that X = a^3 + b^3 = c^3 + d^3 (a != c, a!= d)

Faster: Use a counter

Loop over all unordered pairs of cubes i, j less than X: count i+j. Find largest element <=X with count >= 2. O(X) with an array, O(X^(⅔)) with a hashmap.

C - Cucumber Cundurum

Fit as many circles inside another circle as possible:

We need to find the largest k satisfying 3 conditions:

1) k <= n

C - Cucumber Cundurum

Fit as many circles inside another circle as possible:

consider the ratio between the cucumbers radii: f = r/s

2) Find the maximal number of cucumber slices k that we can use not using too much area

A1 = s*s*pi, A2 = k*r*r*pi, A2/A1 <= z/100

=> k*f*f <= z/100

=> k <= z/(f*f*100)

C - Cucumber Cundurum

3) Fit as many circles inside another circle as possible:

consider the ratio between the circles radii: f = r/s

if f <= ⅓ we can fit in 7 cucumbers, this is also needed for 6 cucumbers.

for 2 < k <=6 we have:

if (1-f)/sin(pi(1 - 2/k)/2) >= 2f/sin(2pi/k)

k can be 2 if f <= ½

k can be 1 if f <= 1

D - Distributing seats

Find the maximal number of passengers that can fit on the very large plane.

D - Distributing seats

Find the maximal number of passengers that can fit on the very large plane.

See passengers as segments of seats where they can sit (l, r)

Sort passengers on the last seat they can sit in. O(n) with counting sort.

Use a sorted Map or Set (TreeMap/TreeSet in java) to remember how many places are left on each row.

Iterate over the passengers and insert as early as possible with map.ceiling(l_i) => O(nlog(n))

E - Entering the time

Find the shortest path between two timestamps only using valid timestamps.

E - Entering the time

Find the **shortest path** between two timestamps only using valid timestamps.

E - Entering the time

Find the **shortest path** between two timestamps only using valid timestamps.

BFS! O(n + m), n = 24*60, m <= 8*n

E - Entering the time

Find the **shortest path** between two timestamps only using valid timestamps.

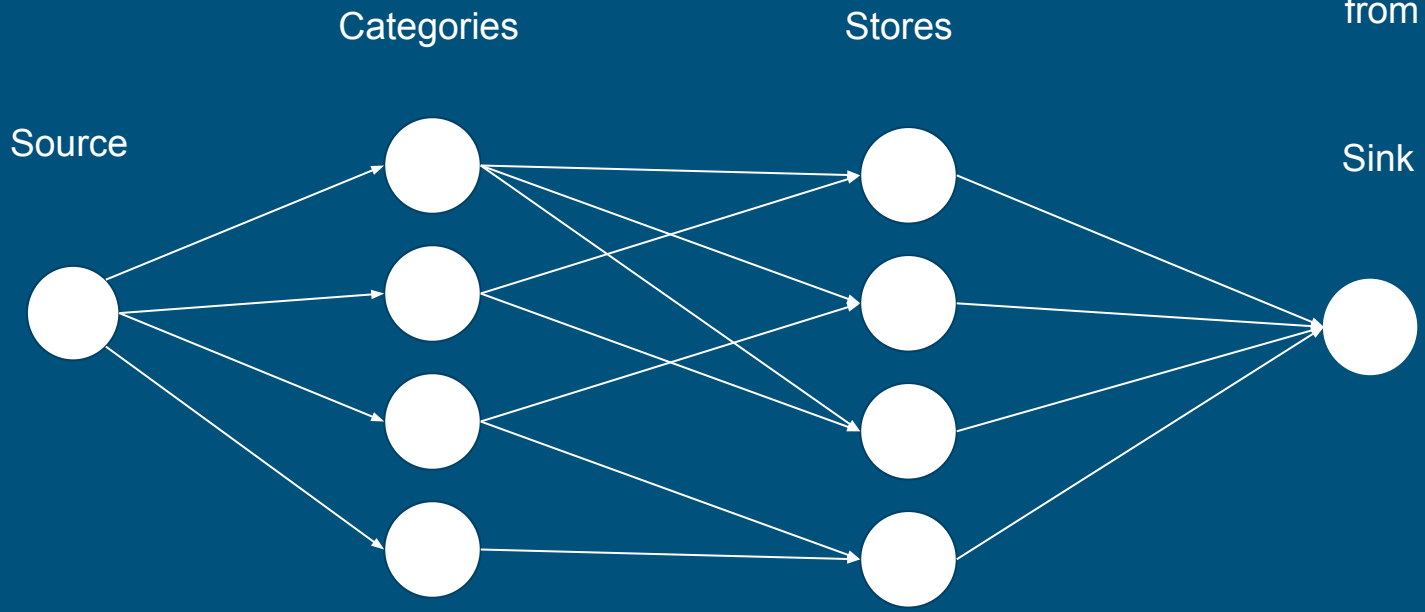
A greedy solution also exists. Make sure to do the hours in the right order. (the order depends on the values).

20:00 -> 19:00 should take 2 steps

19:00 -> 20:00 should also take 2 steps.

G - Get-Rich-Quick Schemes

If a store s has a category c in stock there exists an edge from c to s.



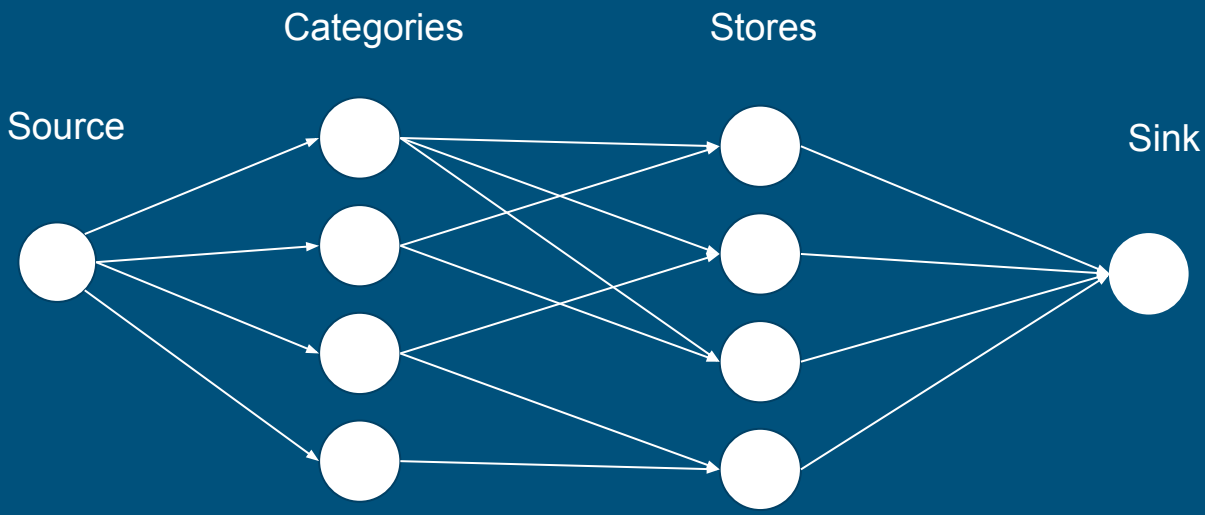
G - Get-Rich-Quick Schemes

Each edge has a gain and a capacity.

Edges from src to categories:
gain, cap = (p_i, m_i)

Edges from categories to stores:
gain, cap = (0, inf).

Edges from stores to sink:
gain, cap = (0, l_i)

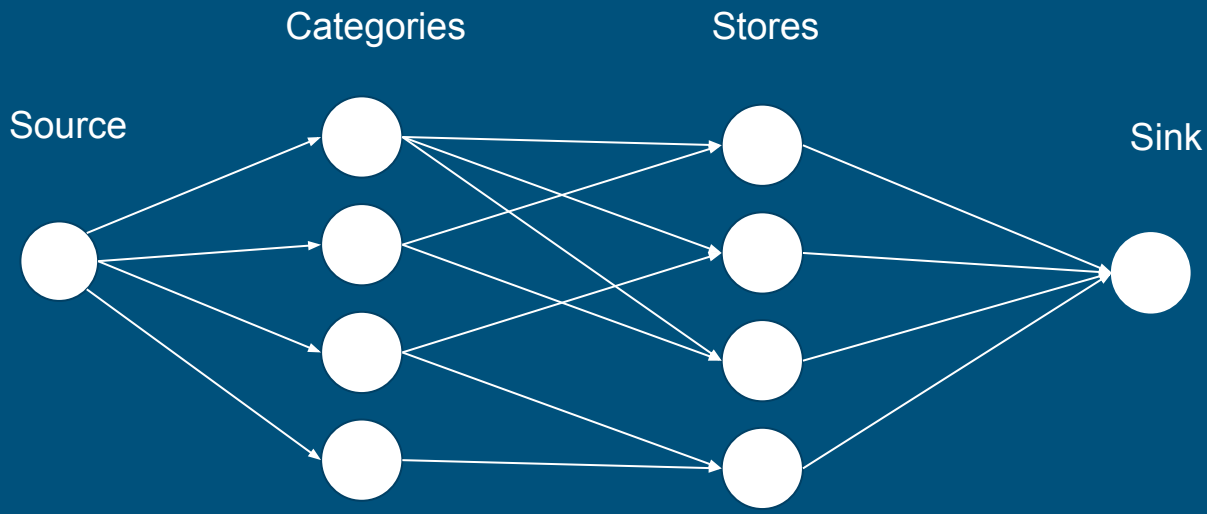


G - Get-Rich-Quick Schemes

If we want to maximize the sum of gain, we will have a maximum flow through the graph. (Otherwise we can gain more, since there are no negative gains)

Solved by max cost max flow.

Or, the more known algorithm: ‘min cost max flow’, by assigning the cost of edge e with -e.gain.



H - Hide and seek

“The m caves are connected by m - 1 tunnels (each between two distinct caves), such that there is a path between any pair of caves”

=> The graph is a Tree.

Do DP over the Tree, dp(x, y, t, canStay) answers, given that you reach node x with t time left, and you have considered y of x’s children neighbours previously, what is the maximal amount of nodes that you can visit.

H - Hide and seek

“The m caves are connected by m - 1 tunnels (each between two distinct caves), such that there is a path between any pair of caves”

=> The graph is a Tree.

Do DP over the Tree, dp(x, y, t, canStay) answers, given that you reach node x with t time left, and you have considered y of x’s children neighbours previously, what is the maximal amount of nodes that you can visit.

dp(x, y, t, canStay) = max(dp(x, y+1, t, canStay), max 0<ti <= t (

max(dp(y, 0, ti, canStay) + dp(x, y+1, t - ti - c_xy, False),

dp(y, 0, ti, False) + dp(x, y+1, t - ti - 2*c_xy, canStay)))

dp(0, 0, t, True) - 1 solves the problem for us.

H - Hide and seek

I call the amount of caves for n, and the maximal time for t.

The size of the DP is sum _ i (t * deg(i) * 2) = 2t *sum(deg(i)) = 2*t*n

However we iterate over all times to compute one element =>

O(t^2n)

I - Include Scoring

Just sort and make sure the list of scores is of length 30.

(Måns was 29, when testing the problem.)

Also make sure that you group all the participants by their score correctly.

(In Johans first solution everyone with an index larger or equal to 30 received a score of 0, even though they sometimes shared rank with someone with an index < 30.)