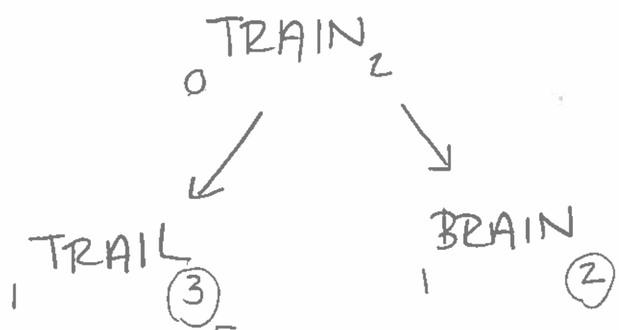
Heuristic Search

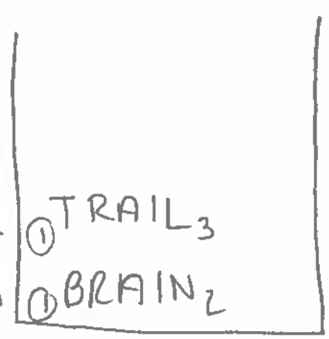
① So why were we making these guesses anyway?



3 letters are different from PRAWN, so let's guess we can reach PRAWN in 3 steps

2 letters are different from PRAWN, so let's guess we can reach PRAWN in 2 steps

② So far, the most "intelligent" container we had was a priority queue that prioritized nodes by g-value:



but these are the same

Why wouldn't we want to factor in our guess about how much it should cost to complete our solution?

HEURISTIC SEARCH

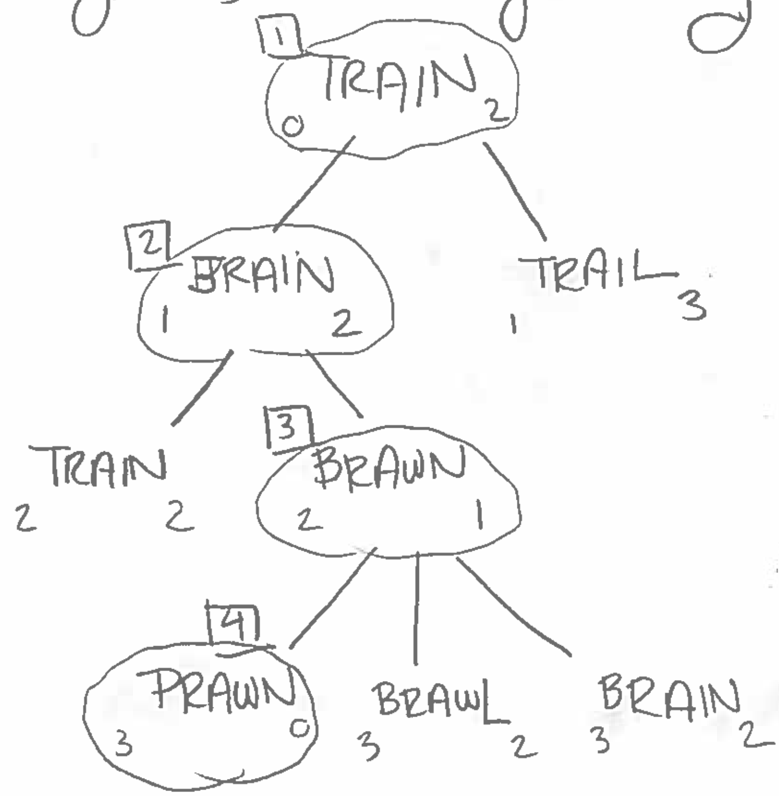
3) It would seem to make sense to have a priority queue that prioritizes nodes by:

the cost so far + the cost to reach the goal from this point on (guessed)

g-value

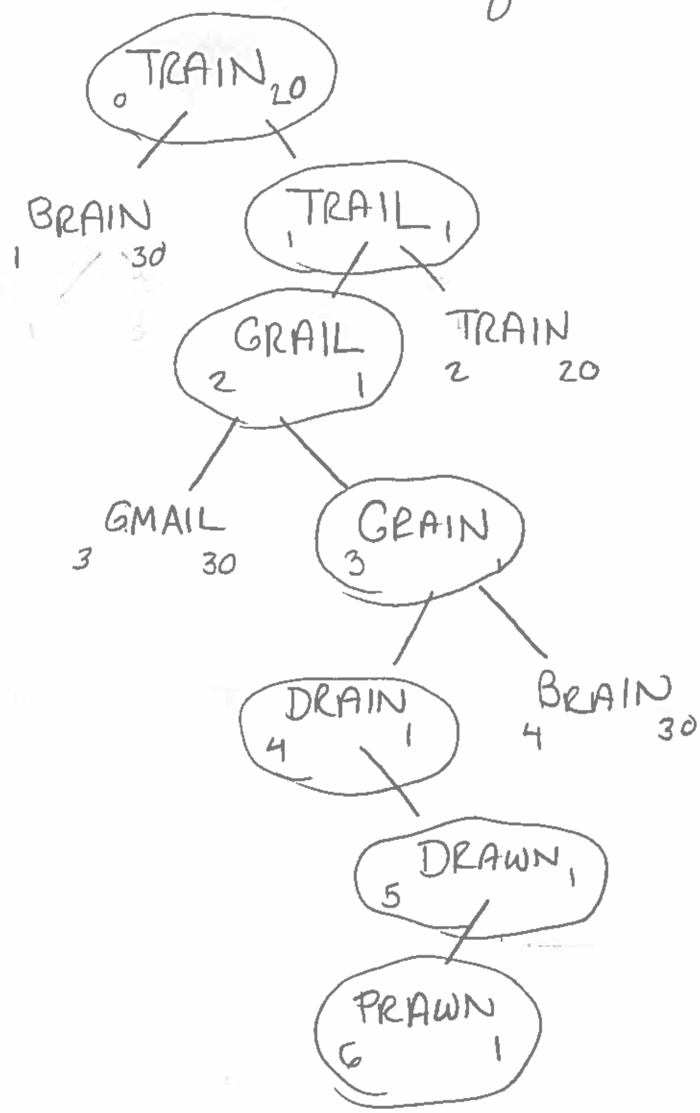
h-value

4) If we use a priority queue that prioritizes search nodes based on $g + h$, something amazing happens



We go straight to the optimal solution!

5) If our guesses are terrible, bad things can happen:



6) How do we choose a heuristic function such that amazing things, as opposed to bad things, happen?

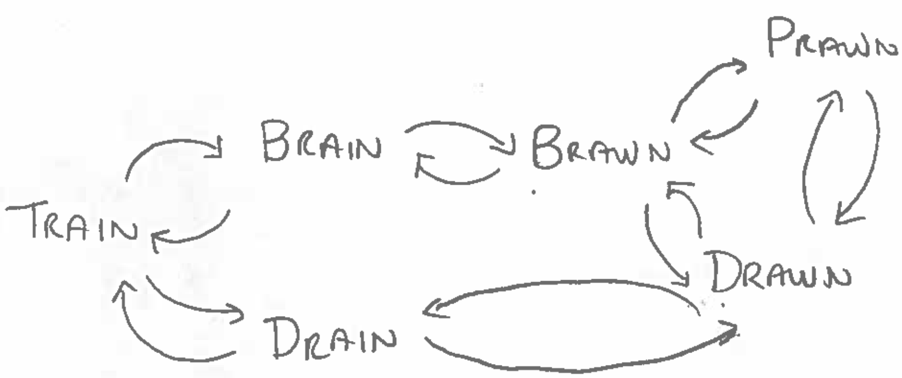
(7) To study this question, it will help to have some extra terminology.

Suppose we have a weighted state machine $M = (Q, \Sigma, \Delta, q_0, F, \omega)$.

Define a <u>completion path</u> from state $q \in Q$ as a sequence $\langle \delta_1, \ldots, \delta_K \rangle$ of transitions where:

- $\forall 1 \le i \le K$, $\delta_i \in \Delta$ and $\delta_i = (q_i, \sigma_i, q_{i+1})$
- $q_1 = q$
- $q_{k+1} \in F$

e.g.



If $q_0 =$ TRAIN and $F = \{$PRAWN$\}$, then the following are examples of completion paths from DRAIN:

< (DRAIN, 4W, DRAWN), (DRAWN, 1P, PRAWN) >

< (DRAIN, 4W, DRAWN), (DRAWN, 1B, BRAWN), (BRAWN, 1P, PRAWN) >

HEURISTIC SEARCH

8) The cost of a completion path is the sum of the weights of its transitions.

The optimal completion path (from state q) is the completion path of minimal cost.

Define $H^*(q)$ as the cost of the optimal completion path.

e.g. $H^*(\text{DRAIN}) = 2$ (i.e. the cost of completion path

< DRAIN, 4W, DRAWN>; < DRAWN, 1P, PRAWN>)

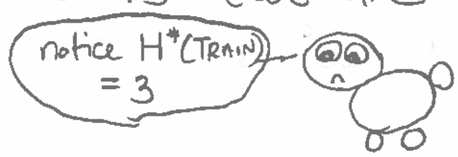
---

9) Let's consider what happens if we use an "optimistic" heuristic function H, in other words, one that never overestimates the cost of the optimal completion cost.

In other, other words:

$$H(q) \leq H^*(q) \quad \text{for all } q \in Q$$

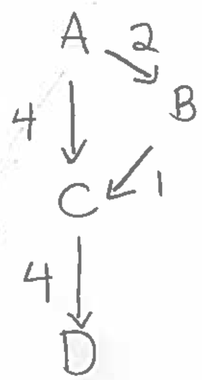
An example (for word ladder) would estimate the cost of completion to be the number of letters that are different from the solution word!



TRAIN (T: different, I: different), so $H(\text{TRAIN}) = 2$

HEURISTIC SEARCH

(10) Such a heuristic is called <u>admissible</u>. Let's show a simple example. Consider the state machine:



$Q = \{A, B, C, D\}$

$q_0 = A$

$F = \{D\}$

Suppose our heuristic function is defined:

$H(A) = 7$

$H(B) = 5$

$H(C) = 1$

$H(D) = 0$

We can check easily that it is admissible:

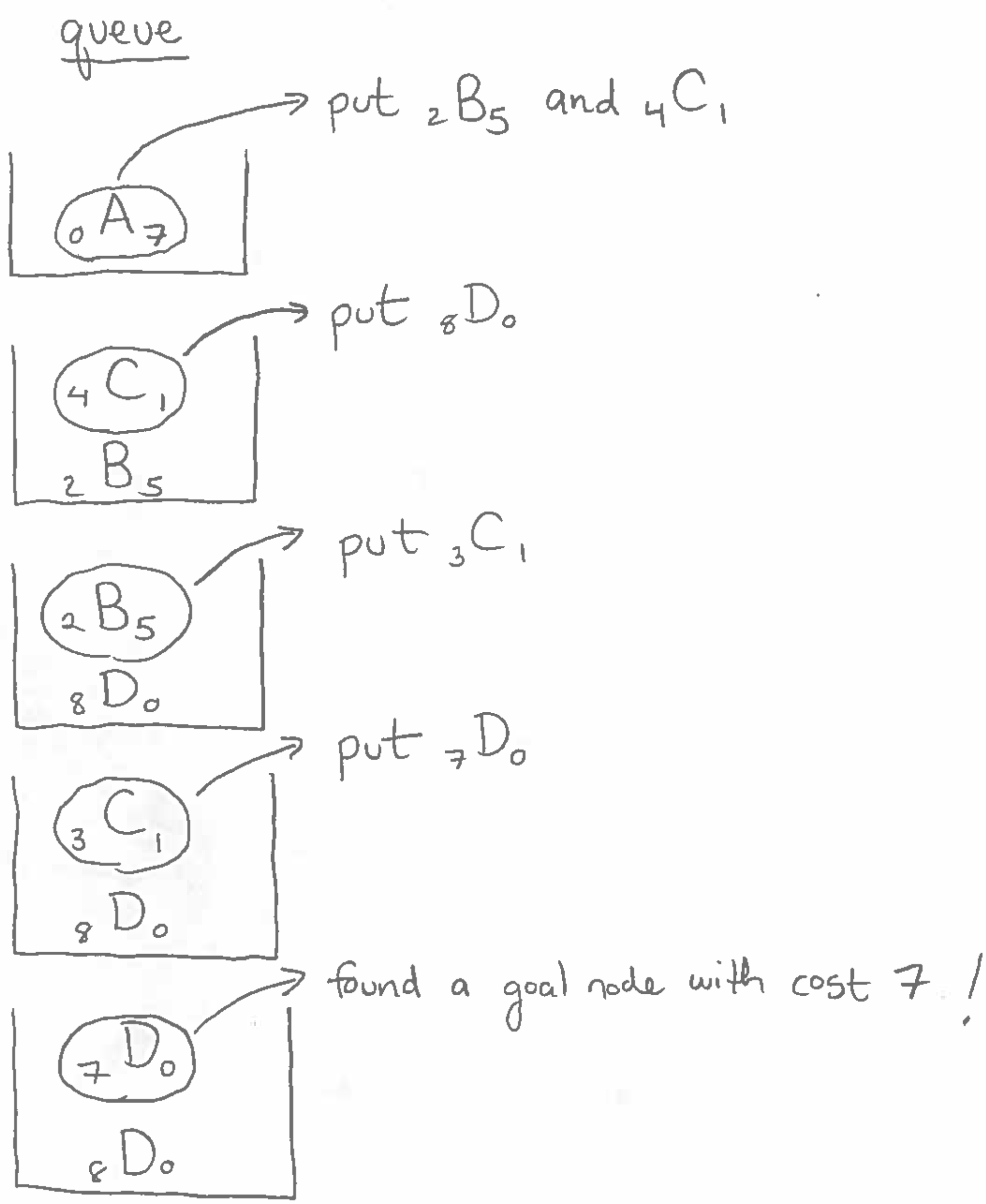
$H^*(A) = 8 \geq 7 = H(A)$

$H^*(B) = 5 \geq 5 = H(B)$

$H^*(C) = 4 \geq 1 = H(C)$

$H^*(D) = 0 \geq 0 = H(D)$

(11) Let's see how the master SEARCH algorithm performs if we use this heuristic, and a priority queue prioritized by $g+h$:



It returns the correct (optimal) solution! In fact, this is true in general. If H is admissible, and we run SEARCH using a priority queue prioritized by $g+h$, then we are guaranteed the optimal solution.

Heuristic Search

(12) Guaranteed, that is, as long as we <u>don't</u> memoize.
Suppose we run MemoizedSearch with the same container and heuristic.

container		visited
${}_0A_7$	put ${}_2B_5$ and ${}_4C_1$	A
${}_4C_1$, ${}_2B_5$	put ${}_8D_0$	A, C
${}_2B_5$, ${}_8D_0$	put ${}_3C_1$	A, B, C
${}_3C_1$, ${}_8D_0$	IGNORE; already visited	A, B, C
${}_8D_0$	found a goal node with cost 8	

Clearly admissibility is not enough to ensure optimality for MemoizedSearch.

HEURISTIC SEARCH

(13) Turns out we need a slightly stronger condition for MEMOIZEDSEARCH.

A heuristic function H is <u>consistent</u> if $H(q) = 0 \ \forall q \in F$ and

$$H(q) \leq w(q, \sigma, q') + H(q')$$

for all $q, q' \in Q$, $\langle q, \sigma, q' \rangle \in \Delta$.

In our simple example, H is not consistent.

$$H(A) = 7 > 5 = \underbrace{w(A, \cdot, C)}_{4} + \underbrace{H(C)}_{1}$$

---

(14) We can prove that if H is consistent, then both SEARCH and MEMOIZEDSEARCH, using a priority queue prioritized by g+h, will be optimal.

First, observe that is node $n' \in \text{SUCCESSORS}_{M,H}(n)$, then:

$$\begin{aligned} g(n') + h(n') &= \left[g(n) + w(q(n), \sigma, q(n'))\right] + h(n') \\ &= g(n) + \left[w(q(n), \sigma, q(n')) + h(n')\right] \\ &\geq g(n) + \left[w(q(n), \sigma, q(n')) + H(q(n'))\right] \\ &\geq g(n) + H(q(n)) \quad \text{[b/c H is consistent]} \\ &= g(n) + h(n) \end{aligned}$$

(15) Next, let's show that if $n_1, n_2, \ldots, n_k$ are the search nodes visited by SEARCH or MEMOIZEDSEARCH (in order), then $j > i \Rightarrow g(n_j) + h(n_j) \geq g(n_i) + h(n_i)$.

Assume (for contradiction) that there exists $j > i$ for which $g(n_j) + h(n_j) < g(n_i) + h(n_i)$.

When we visit node n_i, some ancestor of node n_j must still be in the container (or else n_j will never be visited). Call this ancestor n.

From (14), we know
$$g(n) + h(n) \leq g(n_j) + h(n_j) < g(n_i) + h(n_i)$$

So the priority queue should have visited n, not n_i. Contradiction!

---

(16) Finally, consider the goal nodes. Suppose goal node n_i is visited before goal node n_j. From (15):

$$i < j \Rightarrow g(n_i) + h(n_i) \leq g(n_j) + h(n_j)$$
$$\Rightarrow g(n_i) \leq g(n_j)$$

So we visit goal nodes in order of their cost. Thus the algorithm is optimal!

HEURISTIC SEARCH

(17) The master SEARCH algorithm whose container is a priority queue prioritized by g+h is called A*.

What we've asserted so far:

- if heuristic H is <u>admissible</u>, then the nonmemoized A* returns the optimal solution
- if heuristic H is <u>consistent</u>, then A* (memoized or nonmemoized) returns the optimal solution