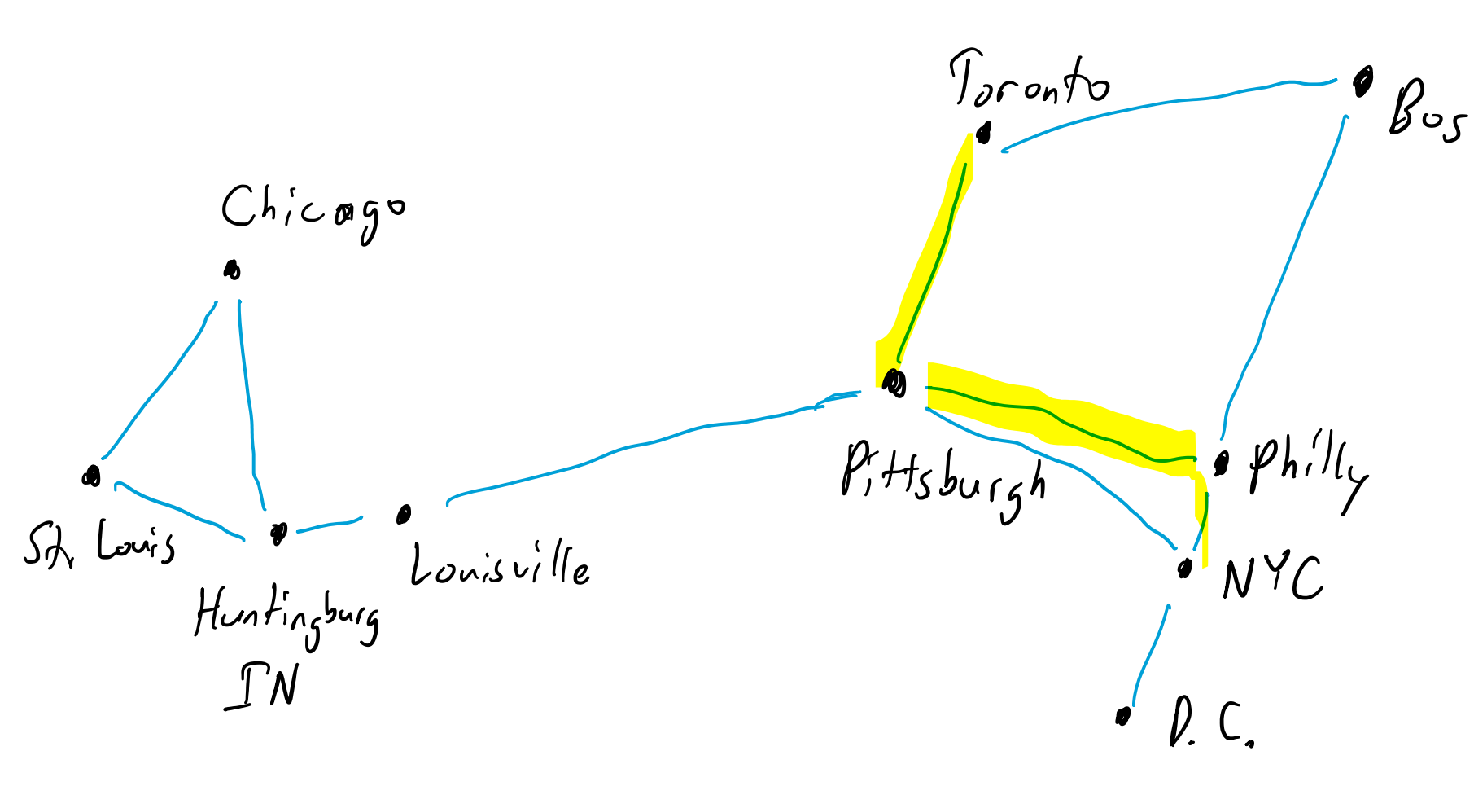
Lec08-A-star

Tuesday, September 8, 2026 11:02 PM

Can we do better if we specifically want the $s \to t$ shortest path? Dijkstra gives all shortest paths from source, and we can start after finishing t (not first-seeing), but that's overkill.

A* algorithm: Faster heuristic for finding shortest $s \to t$ path in graph with positive edge weights

Dijkstra chooses nextEdge by tentative distance from s $d[u] + length(u,v)$.
└ could be exploring in opposite direction from t.



Pittsburg to Huntingburg trip, but Dijkstra has to compute all shortest distances in the East first, which feels inefficient.

Problem: Dijkstra knows nothing about unreached nodes

Heuristic: Give an estimate $h(u)$ of distance from target as part of minimization.

For railroad paths, perhaps Euclidean distance.

A* alg:

- Maintain $d[u]$ = current best distance from s to u found so far
- Need function $h(u)$ = estimate of u to t distance
- Let $f(u) = d[u] + h(u) \to$ estimate of best $s \to u \to t$ distance so far

Modify Dijkstra to use $f(u)$ as key instead of $d[u]$ as key

Pseudocode:

tentative $s \to u$ dist; tentative parent; frontier; tentative $u \to t$ distance

```
for $u \in V$, $d[u] = \infty$, $p[u] = \emptyset$,   $F = V$,   $h[u]$,   $f[u] = d[u] + h[u]$
                                        ~~$F$=makeheap$(V,d)$~~
                                        $F$=makeheap$(V,f)$
$d[s] = 0$
while $F \neq \emptyset$:
    $u \in$ vertex in F with min ~~$d[u]$~~ $f[u]$ — always explore ~~next nearest node~~ node we think is best path to t
    remove u from F

    for each neighbor v of u in F:
        if $d[u] + length(u,v) < d[v]$:
            $p[v] = u$
            $d[v] = d[u] + length(u,v)$
            $f[v] = d[v] + h[v]$
return $d[u]$, $p[u]$
```

Choice of $h(u)$:

Def. Let $h^*(u)$ be the real shortest path from u to t.
A heuristic is admissible if $h(u) \le h^*(u)$ $\forall u$.
└ overestimating bad because you might never go the right way, but underestimating gets corrected as we explore.

Which of the following are NOT admissible:

$h(u) = 0$ → recovers Dijkstra from A*
$h(u) = h^*(u)$ → not useful because shortest path is trivial if we know real distance
$h(u)$ = Euclidean distance for train network
All of the above are admissible

Theorem: If $h(u)$ is admissible, then A* is guaranteed to find an optimal route.

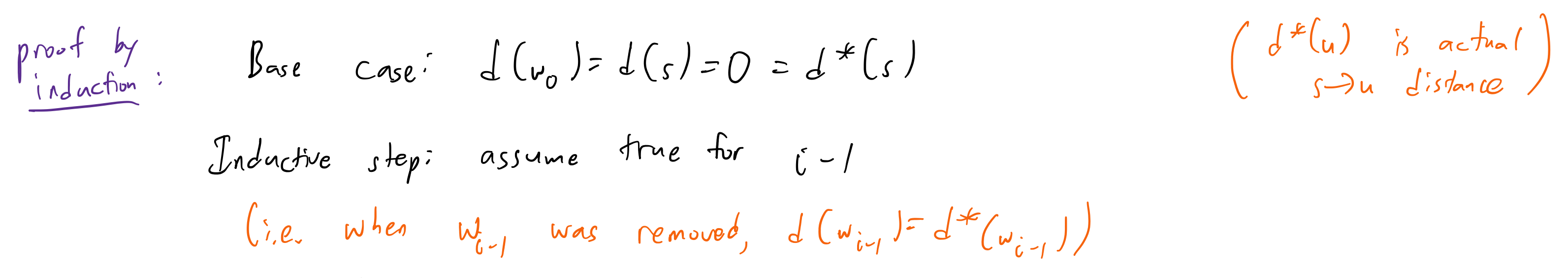
- Want $h(u)$ to be admissible
- Want to minimize $h^*(u) - h(u)$

Intuition: Guessed distances always underestimates, so order of exploration is:

1
2 paths that are
3 larger than they
4 appear
⋮
best path

We will never search a path that is better than our guess, so once we have found an optimal path, all other paths will look (and be) worse.

proof: Suppose not, and let P^* be an optimal path.
Consider the state of nodes along P^* when t is the next node removed from the heap.



At least one node on P^* is on the heap (e.g. t) nodes on P^* still in heap

Let w_m be the first such node.

Lemma: When each node $w_0, \dots, w_{m-1}$ was last removed from heap, $d(w_i) = d^*(w_i)$ ($d^*(u)$ is actual $s \to u$ distance)

proof by induction: Base case: $d(v_0) = d(s) = 0 = d^*(s)$

Inductive step: assume true for $i-1$
(i.e. when w_{i-1} was removed, $d(w_{i-1}) = d^*(w_{i-1})$)
After w_{i-1}'s removal, we updated neighbors, including w_i:

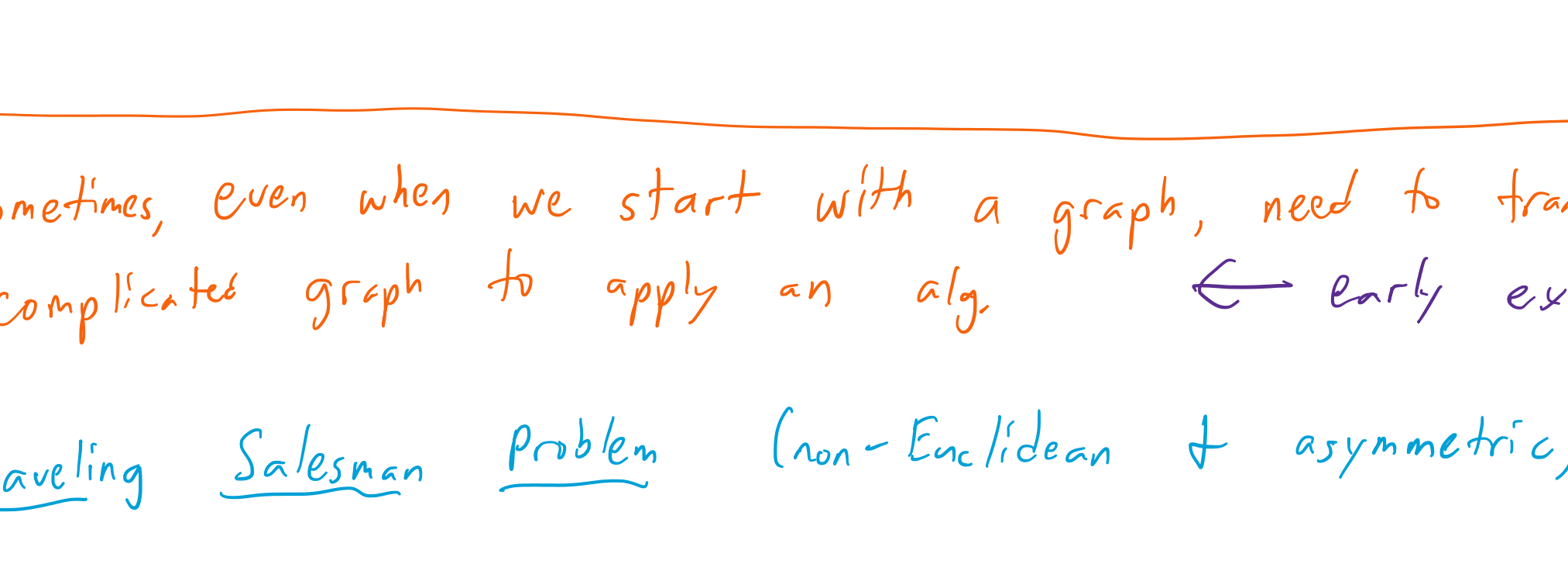
$$d(w) \leftarrow d(w_{i-1}) + length(w_{i-1}, w_i)$$
$$= d^*(w_{i-1}) + length(w_{i-1}, w_i)$$
$$= d^*(w_i) \text{ because } P^* \text{ is optimal.} \quad \square$$

Corollary: As we remove t, $d(w_m) = d^*(w_m)$.
(because w_{m-1} was removed)

back to main proof: Let $w = w_m$. Since $f(t) = d(t) + h(t)$,

$$f(t) = d(t) + \cancel{h(t)}^{\,0}$$
$$\le f(w) \quad \text{b/c } f \text{ is min of paths through frontier}$$
$$= d(w) + h(w)$$
$$= d^*(w) + h(w) \quad \text{since } P^* \text{ is optimal}$$
$$\le d^*(w) + h^*(w) \quad \text{by admissibility}$$
$$= length(P^*) = \text{optimal}$$

$\Rightarrow d(t)$ is optimal when t is removed from heap, so A* did find an optimal path. →/← ▨



Sometimes, even when we start with a graph, need to transform to a more complicated graph to apply an alg. ← early example of a reduction

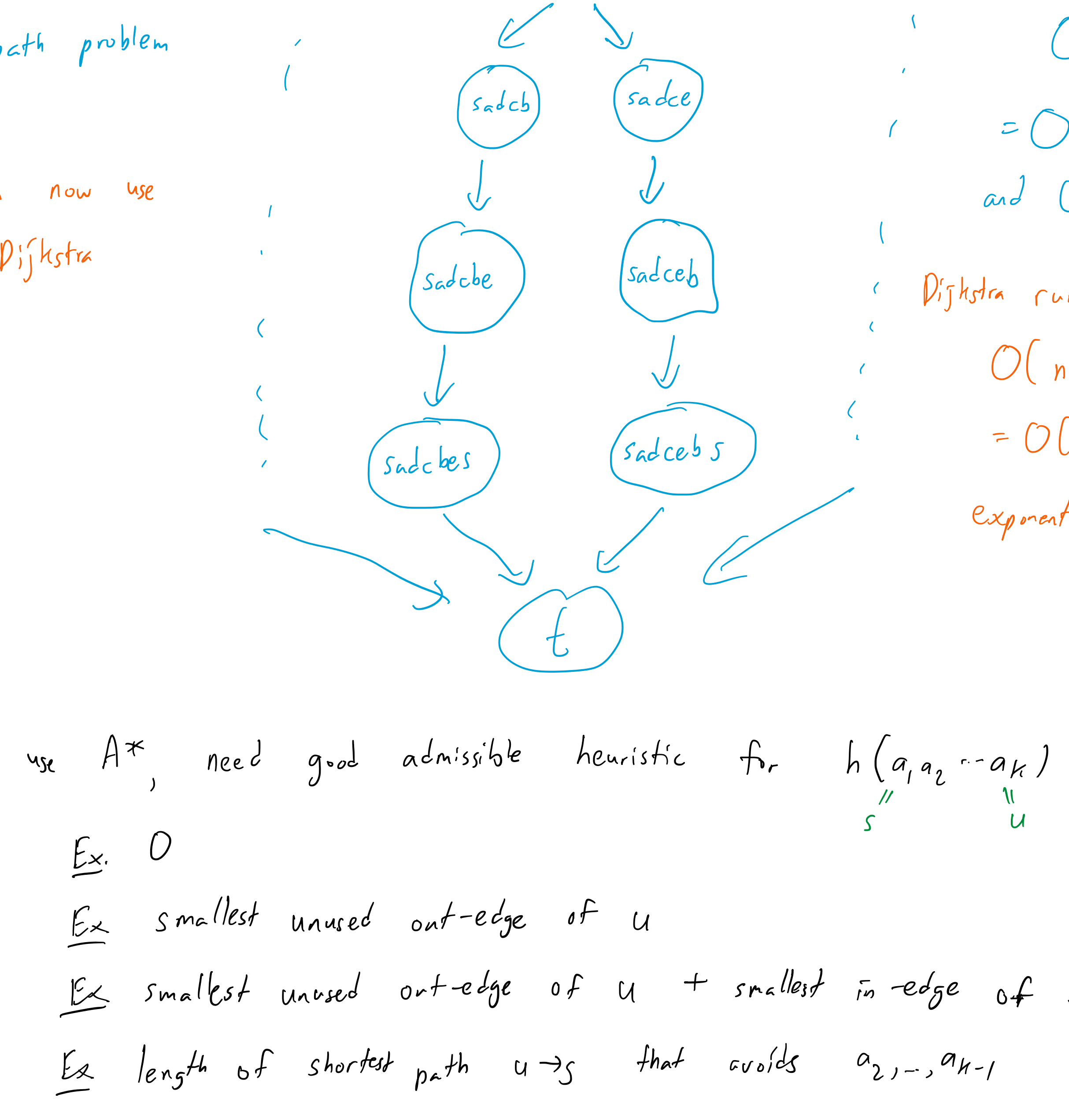
Traveling Salesman Problem (non-Euclidean + asymmetric) (returning back to starting city after visiting every city exactly once)

$\exists$ dist b/t any two cities but $dist(i,j) \neq dist(j,i)$ (one-way train routes)

No triangle inequality
(so can't go back through already visited city)

Reduction: convert graph to state graph that encodes partial tours

Graph with n nodes



Convert TSP to $s \to t$ shortest path problem

Can now use Dijkstra

Graph with $O(n!)$ nodes $= O(n^n)$ nodes and $O(n^n)$ edges

Dijkstra runtime $O(n^n \log n^n)$
$= O(n^{n+1} \log n)$
exponential runtime

To use A*, need good admissible heuristic for $h(\underset{s}{\overset{\shortparallel}{a_1}}, a_2 \cdots \underset{u}{\overset{\shortparallel}{a_k}})$

Ex. 0
Ex. smallest unused out-edge of u
Ex. smallest unused out-edge of u + smallest in-edge of s
Ex. length of shortest path $u \to s$ that avoids $a_2, \dots, a_{k-1}$
Ex. cost(MST) on all nodes except $a_2, \dots, a_{k-1}$

- We can use shortest paths for combinatorial problems by constructing state graph.
- A* incorporates heuristics to speed things up
- Can guarantee optimality using heuristics w/ admissibility.

Summary of shortest paths $n = |V|$, $m = |E|$

Alg	Runtime	Application
BFS	$O(n+m)$	unweighted edges
Dijkstra	$O(m \log n)$	positive edge weights
A*	heuristic?	need heuristic $h(u)$
Bellman-Ford	$O(mn)$	arbitrary edge weights (incl. negative)

Algorithm design techniques recap

- Based on BFS/DFS (bipartite testing, topo sort)
- Greedy tree growing (Prim's, Dijkstra)
- A* (design admissible heuristic; state graphs)