Huge Graphs
- Consider some really huge graphs…
  - All cities and towns in the World Atlas
  - All stars in the Galaxy
  - All ways 10 blocks can be stacked

Huh???

Implicitly Generated Graphs
- A huge graph may be implicitly specified by rules for generating it on-the-fly
- Blocks world:
  - vertex = relative positions of all blocks
  - edge = robot arm stacks one block

Blocks World
- Source = initial state of the blocks
- Goal = desired state of the blocks
- Path source to goal = sequence of actions (program) for robot arm!
- n blocks = n^v vertices
- 10 blocks = 10 billion vertices!

Problem: Branching Factor
- Cannot search such huge graphs exhaustively. Suppose we know that goal is only $d$ steps away.
- Dijkstra’s algorithm is basically breadth-first search (modified to handle arc weights)
- Breadth-first search (or for weighted graphs, Dijkstra’s algorithm) – If out-degree of each node is 10, potentially visits $10^d$ vertices
- 10 step plan = 10 billion vertices visited!

An Easier Case
- Suppose you live in Manhattan; what do you do?
Best-First Search

- The *Manhattan distance* ($\Delta x + \Delta y$) is an estimate of the distance to the goal
  - a heuristic value
- Best-First Search
  - Order nodes in priority to minimize estimated distance to the goal $h(n)$
- Compare: BFS / Dijkstra
  - Order nodes in priority to minimize distance from the start

Problem 1: Led Astray

- Eventually will expand vertex to get back on the right track

Problem 2: Optimality

- With Best-First Search, are you *guaranteed* a shortest path is found when
  - goal is first seen?
  - when goal is removed from priority queue (as with Dijkstra?)

Sub-Optimal Solution

- No! Goal is by definition at distance 0: will be removed from priority queue immediately, even if a shorter path exists!

Synergy?

- Dijkstra / Breadth First guaranteed to find *optimal* solution
- Best First often visits *far fewer* vertices, but may not provide optimal solution
  - *Can we get the best of both?*
A* ("A star")

- Order vertices in priority queue to minimize (distance from start) + (estimated distance to goal)

\[ f(n) = g(n) + h(n) \]

- \( f(n) \) = priority of a node
- \( g(n) \) = true distance from start
- \( h(n) \) = heuristic distance to goal

Optimality

- Suppose the estimated distance (h) is always less than or equal to the true distance to the goal
  - heuristic is a lower bound on true distance

- Then: when the goal is removed from the priority queue, we are guaranteed to have found a shortest path!
Proof of A* Optimality

- A* terminates when G is popped from the heap.
- Suppose G is popped but the path found isn’t optimal:
  - priority(G) > optimal path length c
- Let P be an optimal path from S to G, and let N be the last vertex on that path that has been visited but not yet popped.
  - There must be such an N, otherwise the optimal path would have been found.
  - priority(N) = g(N) + h(N) ≤ c
- So N should have popped before G can pop. Contradiction.

What About Those Blocks?

- “Distance to goal” is not always physical distance
- Blocks world:
  - distance = number of stacks to perform
  - heuristic lower bound = number of blocks out of place

# out of place = 2, true distance to goal = 3
## Other Real-World Applications

- Routing finding – computer networks, airline route planning
- VLSI layout – cell layout and channel routing
- Production planning – “just in time” optimization
- Protein sequence alignment
- Many other “NP-Hard” problems
  - A class of problems for which no exact polynomial time algorithms exist – so heuristic search is the best we can hope for

## Coming Up

- How to make Depth First Search optimal
- Other graph problems
  - Connected components
  - Spanning trees
  - Max-Flow
- Other cool data structures & algorithms
  - Search trees for graphical data
  - Huffman codes
  - Mergeable heaps