CSE 326: Data Structures

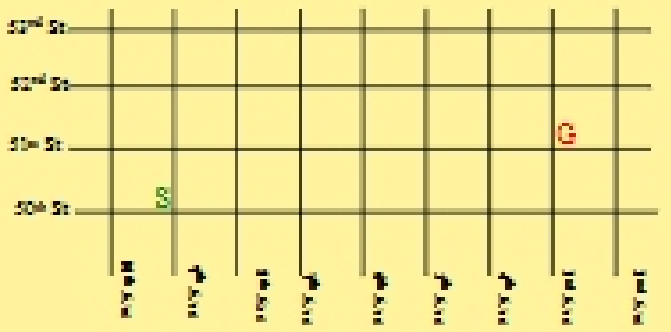
Graph Algorithms
Graph Search

Lecture 23

Problem: Large Graphs

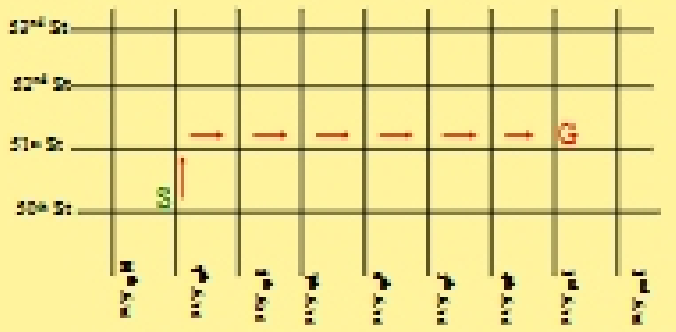
- It is expensive to find optimal paths in large graphs, using BFS or Dijkstra's algorithm (for weighted graphs)
- How can we search large graphs efficiently by using "commonsense" about which direction looks most promising?

Example



Plan a route from 9th & 50th to 3rd & 51st

Example



Plan a route from 9th & 50th to 3rd & 51st

Best-First Search

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

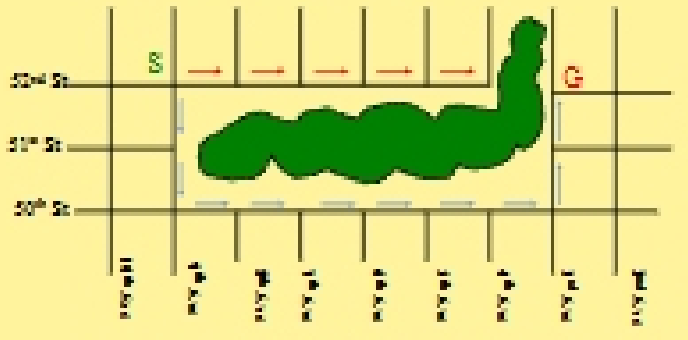
Best-First Search

Open – Heap (priority queue)
Criteria – Smallest key (highest priority)
h(n) – heuristic estimate of distance from n to closest goal

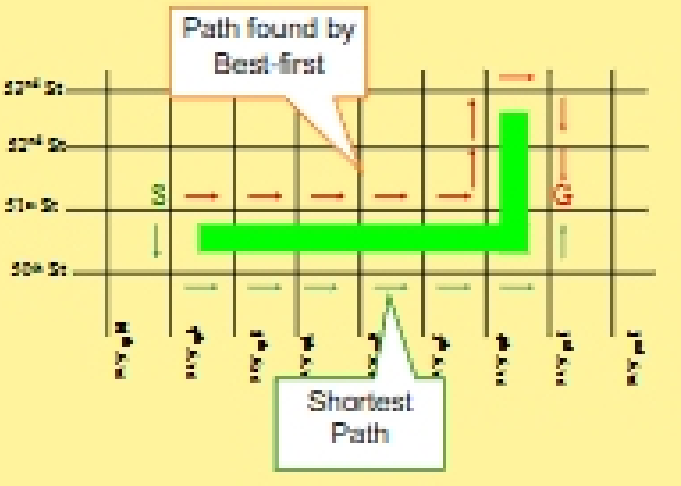
```
Best_First_Search( Start, Goal_Test)
    insert(Start, h(Start), heap);
    repeat
        if (empty(heap)) then return fail;
        Node := deleteMin(heap);
        if (Goal_test(Node)) then return Node;
        for each child of node do
            if (Child not already visited) then
                insert(Child, h(Child),heap);
        end
        Mark Node as visited;
end
```

Obstacles

- Best-FS eventually will expand vertex to get back on the right track



Non-Optimality of Best-First



Improving Best-First

- Best-first is often tremendously faster than BFS/Dijkstra, but might stop with a non-optimal solution
- How can it be modified to be (almost) as fast, but guaranteed to find optimal solutions?
- A* - Hart, Nilsson, Raphael 1968
  - One of the first significant algorithms developed in AI
  - Widely used in many applications

A*

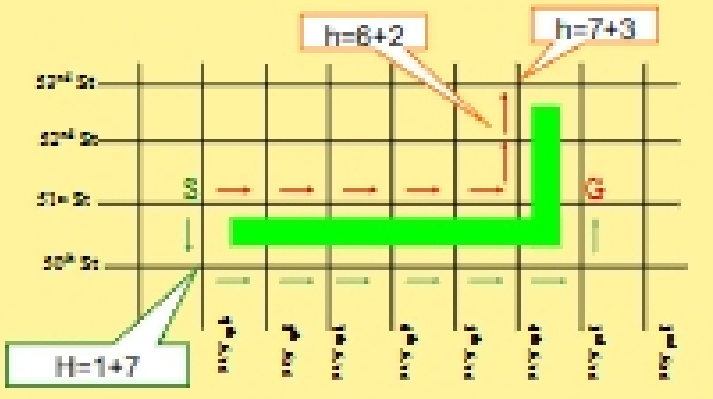
- Exactly like Best-first search, but using a different criteria for the priority queue:
- minimize (distance from start) + (estimated distance to goal)
- priority f(n) = g(n) + h(n)

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

Optimality of A*

- Suppose the estimated distance is *always* less than or equal to the true distance to the goal
  - heuristic is a lower bound
- Then: when the goal is removed from the priority queue, we are guaranteed to have found a shortest path!

A* in Action

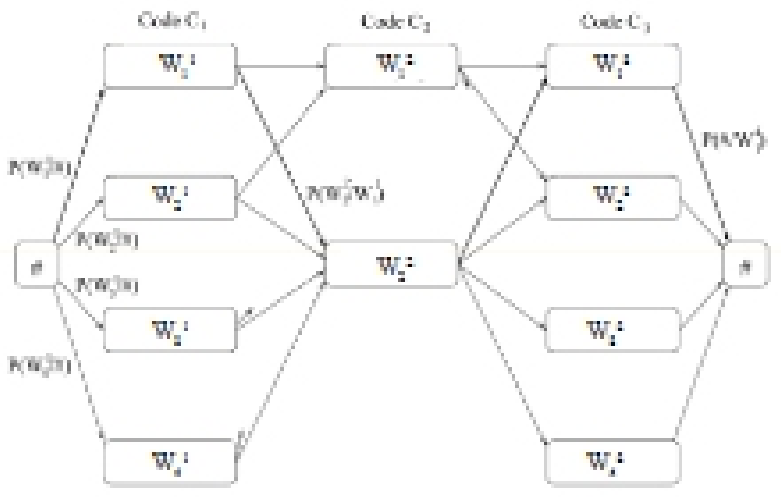


Application of A*: Speech Recognition

- (Simplified) Problem:
  - System hears a sequence of 3 words
  - It is unsure about what it heard
    - For each word, it has a set of possible "guesses"
    - E.g.: Word 1 is one of { "hi", "high", "I" }
  - What is the most likely sentence it heard?

Speech Recognition as Shortest Path

- Convert to a shortest-path problem:
  - Utterance is a "layered" DAG
  - Begins with a special dummy "start" node
  - Next: A layer of nodes for each word position, one node for each word choice
  - Edges between every node in layer i to every node in layer i+1
    - Cost of an edge is smaller if the pair of words frequently occur together in real speech
      - Technically: - log probability of co-occurrence
  - Finally: a dummy "end" node
  - Find shortest path from start to end node



Summary: Graph Search

- Depth First
  - Little memory required
  - Might find non-optimal path
- Breadth First
  - Much memory required
  - Always finds optimal path
- Iterative Depth-First Search
  - Repeated depth-first searches, little memory required
- Dijkstra's Short Path Algorithm
  - Like BFS for weighted graphs
- Best First
  - Can visit fewer nodes
  - Might find non-optimal path
- A^*
  - Can visit fewer nodes than BFS or Dijkstra
  - Optimal if heuristic estimate is a lower-bound

Dynamic Programming

- Algorithmic technique that systematically records the answers to sub-problems in a table and re-uses those recorded results (rather than re-computing them).
- **Simple Example**: Calculating the Nth Fibonacci number.

  Fib(N) = Fib(N-1) + Fib(N-2)

Floyd-Warshall

```
for (int k = 1; k =< V; k++)
 for (int i = 1; i =< V; i++)
   for (int j = 1; j =< V; j++)
    if ( ( M[i][k]+ M[k][j] ) < M[i][j] )
     M[i][j] =  M[i][k]+ M[k][j]
```

Invariant: After the kth iteration, the matrix includes the shortest paths for all pairs of vertices (i,j) containing only vertices 1..k as intermediate vertices