CSE 326: Data Structures
Lecture #19
Approaches to Graph Exploration

Bart Niswonger

Summer Quarter 2001

Today's Outline

- Stuff Bart didn't finish Friday
- Algorithm Design
  - Divide & Conquer
  - Greedy
  - Dynamic Programming
  - Randomized
  - Backtracking

Divide & Conquer

- Divide problem into multiple smaller parts
- Solve smaller parts
  - Solve base cases directly
  - Otherwise, solve subproblems recursively
- Merge solutions together (Conquer!)

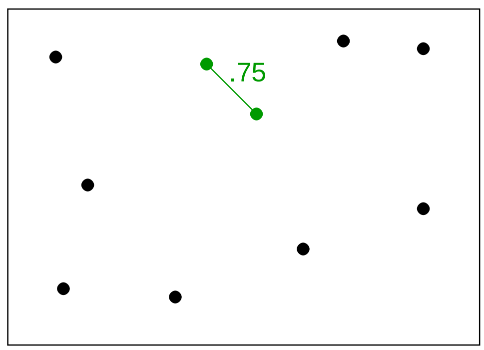
Often leads to elegant and simple recursive implementations.

Divide & Conquer in Action

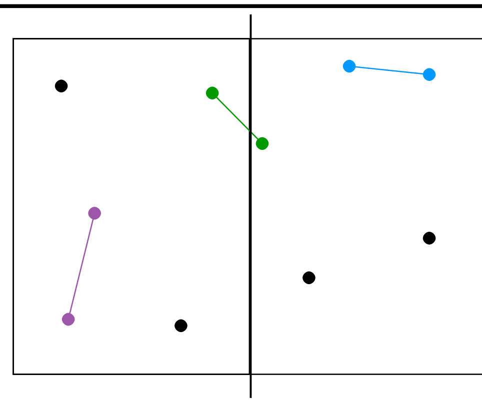
- Binary Search
- Mergesort
- Quicksort
- buildHeap
- buildTree
- Closest points

Closest Points Problem

- Given:
  - a group of points $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Return the distance between the closest pair of points



Closest Points Algorithm

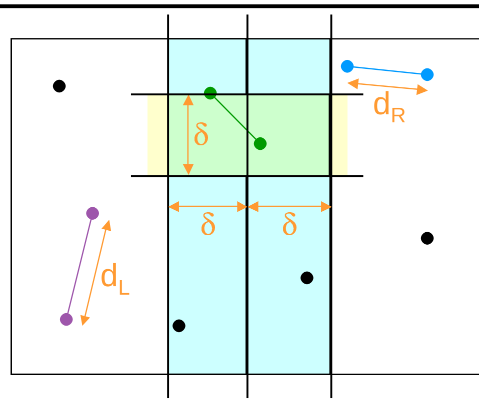


Closest pair is:

- closest pair on left ***or***
- closest pair on right ***or***
- closest pair spanning the middle

runtime:

Closest Points Algorithm



$\delta = \min(d_L, d_R)$

runtime:

Closest pair is:
- closest pair on left ***or***
- closest pair on right ***or***
- closest pair in middle strip within one δ of each other horizontally and vertically

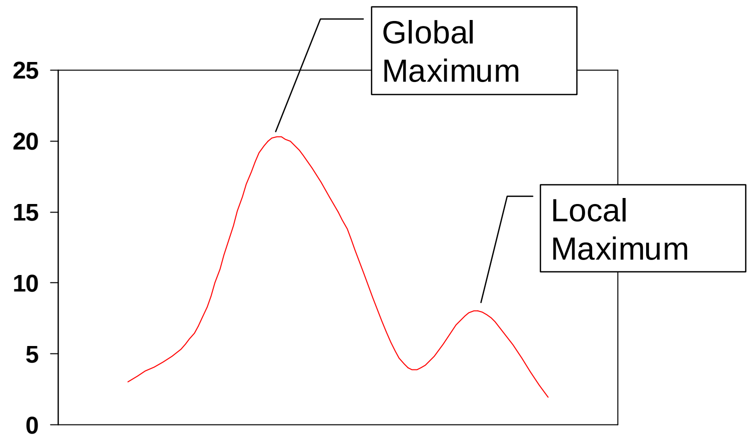
Greedy Algorithms

Repeat until problem is solved:
- Measure options according to *marginal* value
- Commit to maximum

Greedy algorithms are normally fast and simple.

Sometimes appropriate as a *heuristic* solution or to approximate the optimal solution.

Hill-Climbing



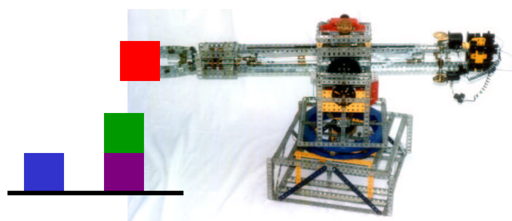
Greed in Action

- Kruskal's Algorithm
- Dijkstra's Algorithm
- Prim's Algorithm
- Huffman Encodings
- Scheduling
- Best First Search
- A^* Search

Huge Graphs

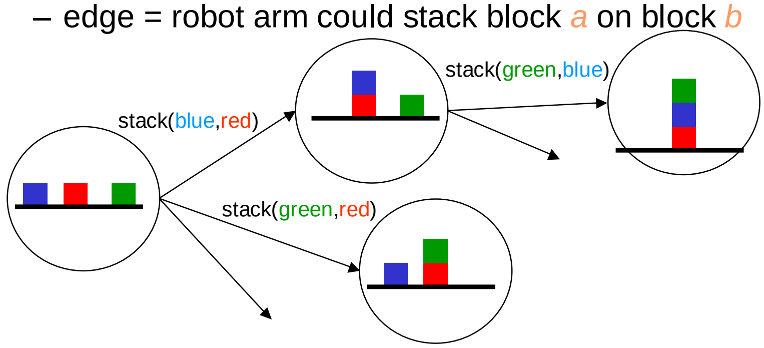
- Consider some really *huge* graphs…
  - All cities and towns in the World Atlas
  - All stars in the Galaxy
  - All ways 10 blocks can be stacked

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 could stack block *a* on block *b*



Blocks World

source: initial state of the blocks

goal: desired state of the blocks

path from source to goal = sequence of actions (*program*) for robot arm

- n blocks ≈ n^n states
- 10 blocks ≈ 10 billion states

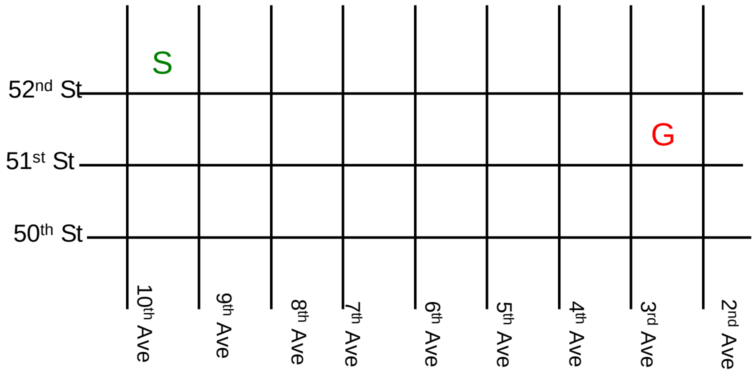
Problem: Branching Factor

- Dijkstra's algorithm is basically breadth-first search (modulo arc weights)
  - Visits all nodes (exhaustive search)
- Suppose we know that goal is only *d* steps away.
- If out-degree of each node is 10, potentially visits 10^d vertices
  - 10 step plan => 10 billion vertices!

Cannot search such huge graphs exhaustively!

An Easier Case

- Suppose you live in Manhattan; what do you do?



Best-First Search

The *Manhattan distance* (Δ x+ Δ y) is an estimate of the distance to the goal

- a *heuristic value*
  - *heuristic*: involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods (Merriam Webster – www.m-w.com)

Best-First Search

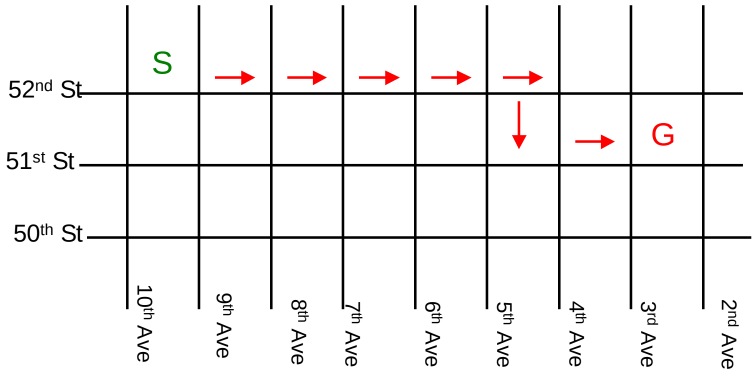
- Order nodes in priority to minimize estimated distance to the goal

Compare: *Dijkstra*

- Order nodes in priority to minimize distance from the start

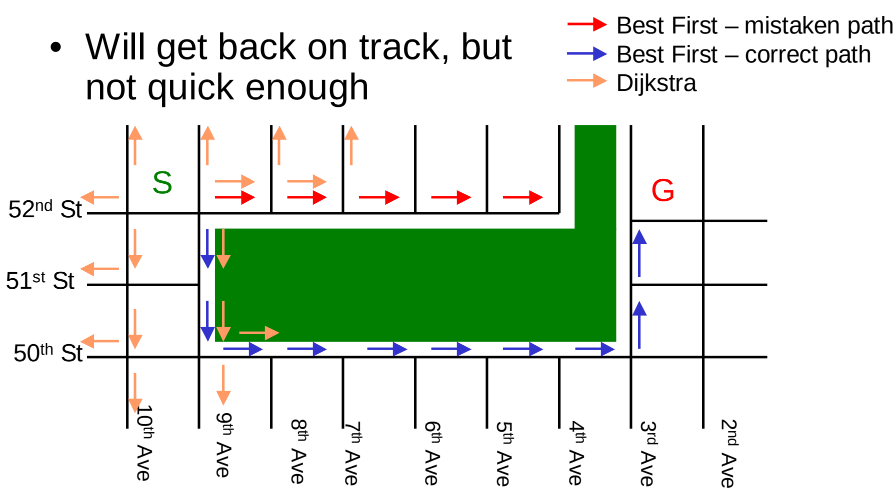
Best First in Action

- Suppose you live in Manhattan; what do you do?



Being Mislead

- Will get back on track, but not quick enough

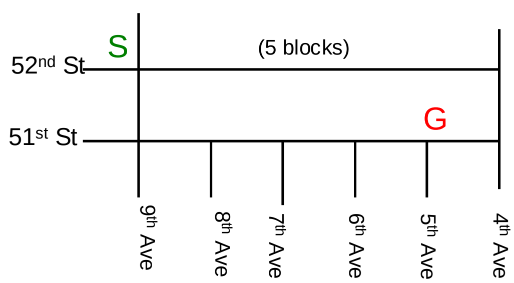


Optimality

- Does Best-First Search find the shortest path
  - when the goal is first seen?
  - when the goal is removed from priority queue?

Sub-Optimal Solution

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

$$f(n) = g(n) + h(n)$$

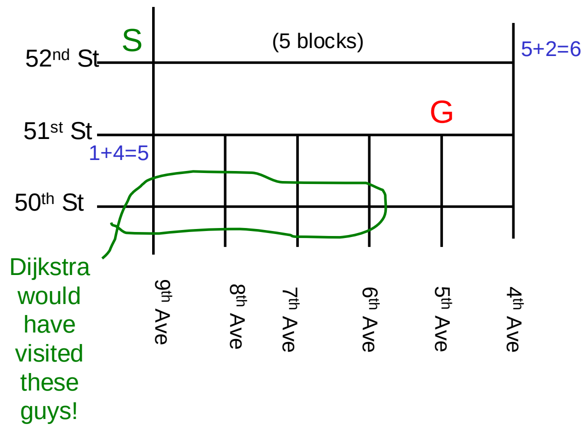
f(n) = priority of a node
g(n) = true distance from start
h(n) = heuristic distance to goal

Optimality

- Suppose the estimated distance (h) is ≤ the true distance to the goal
  - (heuristic is a lower bound)

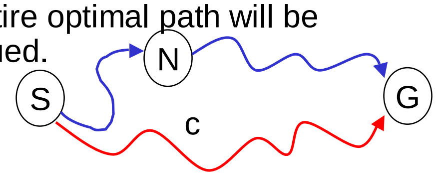
- Then: when the goal is removed from the priority queue, we are guaranteed to have found a shortest path!

Optimality Revisited



Revised Cloud Proof

- Suppose have found a path of cost c to G which is not optimal
  - priority(G) = f(G) = g(G) + h(G) = c + 0 = c
- Say N is the last vertex on an optimal path P to G which has been added to the queue but not yet dequeued.
  - There must be such an N, otherwise the optimal path would have been found.
  - priority(N) = f(N) = g(N) + h(N) $\leq$ g(N) + actual cost N to G = cost of path P < c
- So N will be dequeued before G is dequeued
- Repeat argument to show entire optimal path will be expanded before G is dequeued.

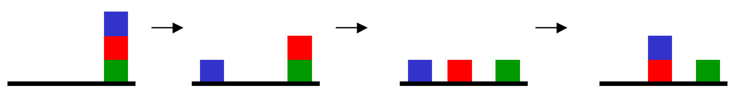


A Little History

- A* invented by Nils Nilsson & colleagues in 1968
  - or maybe some guy in Operations Research?
- Cornerstone of artificial intelligence
  - still a hot research topic!
  - iterative deepening A*, automatically generating heuristic functions, …
- Method of choice for search large (even infinite) graphs when a good heuristic function can be found

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 Examples

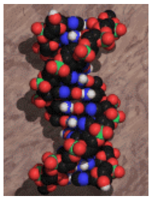
- Simplifying Integrals
  - vertex = formula
  - goal = closed form formula without integrals
  - arcs = mathematical transformations

$$\int x^n dx \rightarrow \frac{x^{n+1}}{n+1}$$

  - heuristic = number of integrals remaining in formula

DNA Sequencing

- Problem: given chopped up DNA, reassemble
- Vertex = set of pieces
- Arc = stick two pieces together
- Goal = only one piece left
- Heuristic = number of pieces remaining - 1



Solving Simultaneous Equations

- Input: set of equations
- Vertex = assignment of values to some of the variables
- Edge = Assign a value to one more variable
- Goal = Assignment that simultaneously satisfies all the equations
- Heuristic = Number of equations not yet satisfied

What ISN'T A*?

essentially, nothing.

Greedy Summary

- Greedy algorithms are not always optimal
  - Some greedy algorithms give provably optimal solutions (Dijkstra)
  - Others do not
- Notion of minimizing some function
  - Dijkstra – minimizes distance from start
  - Best First – minimizes distance to finish
  - Kruskal – minimizes edge costs
  - Hill Climbing – minimizes distance to the sky

Dynamic Programming (Memoizing)

- Define problem in terms of smaller subproblems
- Solve and record solution for base cases
- Build solutions for subproblems up from solutions to smaller subproblems

Can improve runtime of divide & conquer algorithms that have shared subproblems with *optimal substructure*.

Usually involves a table of subproblem solutions.

Dynamic Programming in Action

- Sequence Alignment
- Optimal Binary Search Tree
- *All* pairs shortest path
- Many, many optimization problems
  - Databases: finding the optimal way to answer a query
  - Workflow: the optimal order of operations to construct some complex object
- Fibonacci numbers

Fibonacci Numbers

```
F(n) = F(n - 1) + F(n - 2)
F(0) = 1
F(1) = 1
```

```
int fib(int n) {
  if (n <= 1)
    return 1;
  else
    return fib(n - 1) +
           fib(n - 2);
}
```

runtime:

Fibonacci Numbers

Observation: every Fibonacci number depends on the previous two

```
int fib(int n) {
  static vector<int> fibs;
  if (n <= 1)
    return 1;

  if (fibs[n] == 0)
    fibs[n] = fib(n - 1) +
              fib(n - 2);

  return fibs[n];
}
```

recurrence:

runtime:

To Do

- Project IV
  - Write an algorithm over your Graph Data Structure!
- Finish reading Chapter 10
- Come to the movie Friday

Coming Up

- Other Data Structures
- No Quiz tomorrow
- Movie!! (& pizza)