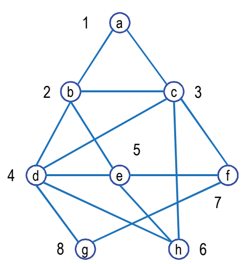
Greedy Algorithms

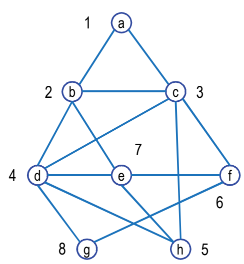
Design and Analysis of Algorithms
Andrei Bulatov

"Greed … is good. Greed is right.
Greed works."
"Wall Street"

Graph Traversal, BFS and DFS



BFS



DFS

BFS and DFS

Theorem
The running time of BFS and DFS is O(m + n) where n is the number of vertices in the graph, and m the number of edges

Bipartiteness

Use BFS to check if a graph is bipartite

Shortest Path

Suppose that every arc e of a digraph G has length (or cost, or weight, or …) len(e)
Then we can naturally define the length of a directed path in G, and the distance between any two nodes

The s-t-Shortest Path Problem
Instance:
Digraph G with lengths of arcs, and nodes s,t
Objective:
Find a shortest path between s and t

Single Source Shortest Path

The Single Source Shortest Path Problem

Instance:

Digraph G with lengths of arcs, and node s

Objective:

Find shortest paths from s to all nodes of G

Greedy algorithm:

Attempts to build an optimal solution by small steps, optimizing locally, on each step

Dijkstra's Algorithm

Input: `digraph G with lengths len, and node s`

Output: `distance d(u) from s to every node u`

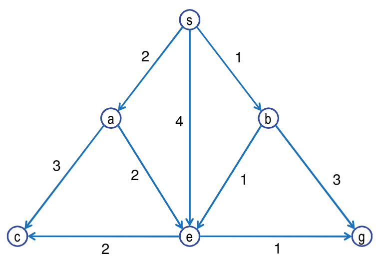
Method:

let S be the set of explored nodes

for each $v \in S$ let d(v) be the distance from s to v

```
set S:={s} and  d(s):=0
while S≠V do
  pick a node v not from S such that the value
```
$d'(v) := \min_{e=(u,v), u \in S}\{d(u)+len(e)\}$
```
  is minimal
  set S:=S∪{v}, and  d(v):=d'(v)
endwhile
```


Example



Questions

What if G is not connected?

there are vertices unreachable from s?

How can we find shortest paths from s to nodes of G?

Dijkstra's Algorithm

Input: `digraph G with lengths len, node s`

Output: `distance d(u) from s to every node u and predecessor P(u) in the shortest path`

Method:

```
set S:={s}, d(s):=0, and P(s):=null
while S≠V do
  pick a node v not from S such that the value
```
$d'(v) := \min_{e=(u,v), u \in S}\{d(u)+len(e)\}$
```
  is minimal
  set S:=S∪{v} and d(v):=d'(v)
  set P(v):= u (providing the minimum)
endwhile
```


Dijkstra's Algorithm Analysis: Soundness

Theorem

For any node v the path s, … P(P(P(v))), P(P(v)), P(v), v is a shortest s – v path

Method: Algorithm stays ahead

Soundness

Proof

Induction on |S|

Base case: If |S| = 1, then S = {s}, and d(s) = 0

Induction case:

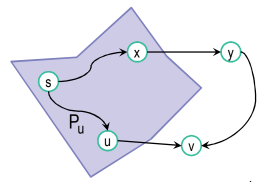
Let P_u denote the path s, … P(P(P(u))), P(P(u)), P(u), u

Suppose the claim holds for |S| = k, that is for any $u \in S$ P_u is the shortest path

Let v be added on the next step.

Consider any path P from s to v other than P_v

Soundness (cntd)



There is a point where P leaves S for the first time

Let it be arc (x,y)

The length of P is at least the length of P_x + the length of (x,y) + the length of y – v

However, by the choice of v

$$len(P_v) = len(P_u) + len(u,v) \le len(P_x) + len(x,y) \le len(P)$$

QED

Running Time

Let the given graph have n nodes and m arcs

n iterations of the while loop

Straightforward implementation requires checking up to m arcs

that gives O(mn) running time

Improvements:

For each node v store $d'(v) := \min_{e=(u,v), u \in S} \{d(u) + len(e)\}$

and update it every time S changes

When node v is added to S we need to change deg(v) values

m changes total

O(m+n) `calls' Properly implemented this gives O(m log n)

Recall heaps and priority queues