Path finding

Goldsmiths Computing

Motivation

- Exploration of known, partially known or unknown surroundings
- Component of various AI solutions
  - especially agents exploring some space:
    - ... game enemies
    - ... NPCs
    - ... self-driving cars

Definition

Single-source shortest path

- from a single source node:
  - find the shortest path to every node in the graph
  - stop early if we have a specific target node

Basic approach

> *"Begin at the beginning," the King said gravely, "and go on till you come to the end: then stop."*

Initial state

Start at the start node

Goal state

Stop when we have found a path (optimally: shortest) to the target node

- (if no target: stop when there are no nodes without the shortest path known)

State expansion

Explore the graph using neighbours of already-visited nodes

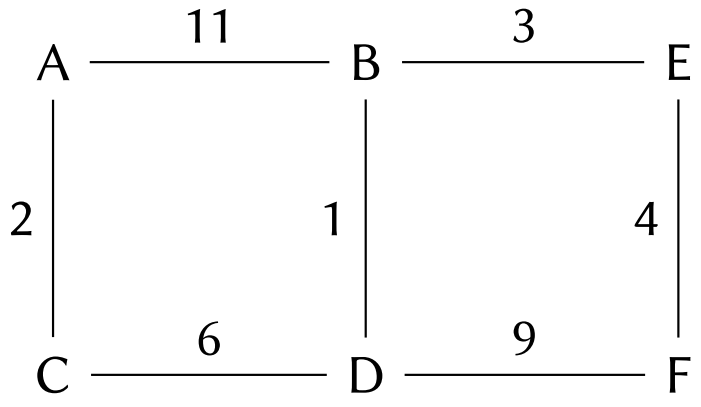
Greedy best-first search

Explore the graph using the neighbour of the current node which is closest to the target.

function GBFS(G,start,end)
    current ← start
    result ← **new** queue()
    while current ≠ end **do**
        ENQUEUE(result,current)
        ns ← NEIGHBOURS(G,current)
        current ← $\arg\min_{n \in \text{ns}} d(n, \text{end})$
    end while
    return result
end function

Example

Node	d(n,F)
A	14
B	6
C	12
D	7
E	4
F	0



Result

path A → B → E → F

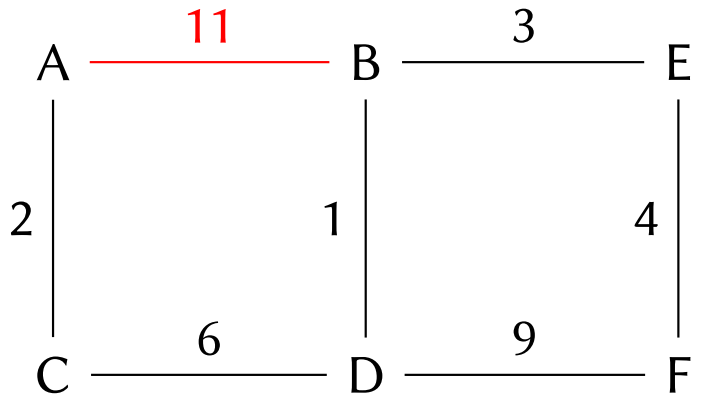
distance 18

Problems

- does not necessarily find a solution!
- not guaranteed optimal

Example

Node	d(n,F)
A	14
B	6
C	12
D	7
E	4
F	0



Result

path A → B → E → F

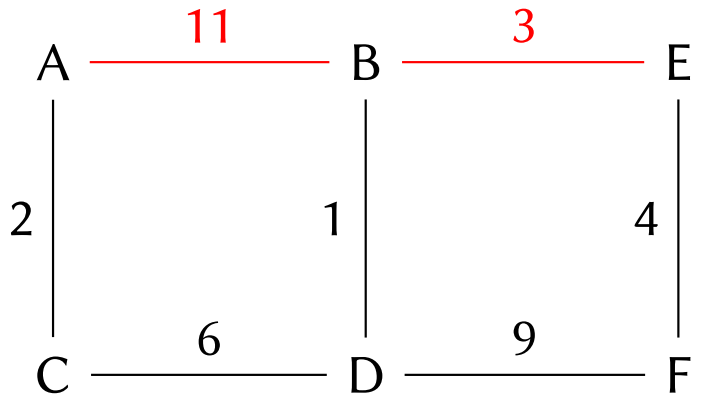
distance 18

Problems

- does not necessarily find a solution!
- not guaranteed optimal

Example

Node	d(n,F)
A	14
B	6
C	12
D	7
E	4
F	0



Result

path A → B → E → F

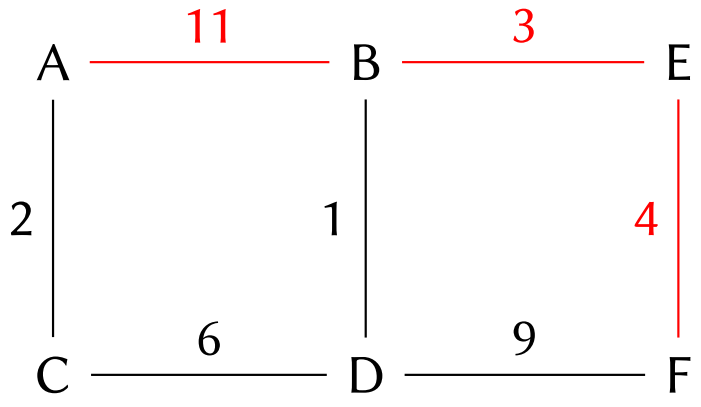
distance 18

Problems

- does not necessarily find a solution!
- not guaranteed optimal

Example

Node	d(n,F)
A	14
B	6
C	12
D	7
E	4
F	0



Result

path A → B → E → F

distance 18

Problems

- does not necessarily find a solution!
- not guaranteed optimal

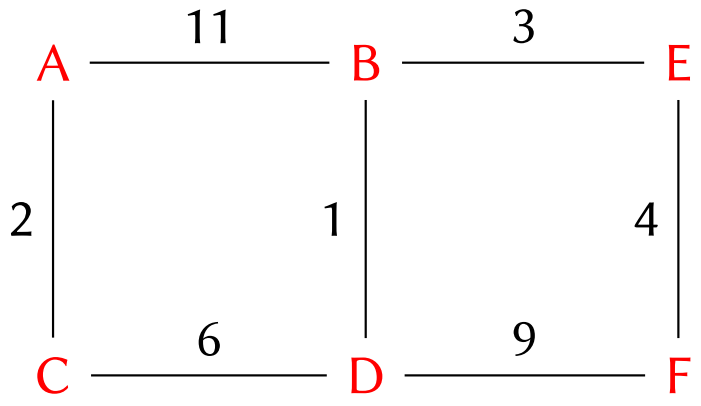
Dijkstra's algorithm

Explore the graph using the neighbour of the already-visited nodes with the smallest distance from the start node

```
function DIJKSTRA(G,start,end)
    dist ← new table(); prev ← new table()
    Q ← new min-heap(dist)
    for v ∈ G do
        dist[v] ← 0 if v = start else ∞; INSERT(Q,v)
    end for
    while ¬ EMPTY(Q) do
        u ← EXTRACT-MIN(Q)
        if u = end then
            s ← new stack()
            while u ≠ start do
                PUSH(s,u); u ← prev[u]
            end while
            return s
        end if
        for v ∈ NEIGHBOURS(G,u) do
            d ← dist[u] + WEIGHT(G,u,v)
            if d < dist[v] then
                dist[v] ← d; prev[v] ← u; DECREASE-KEY(Q,v,dist[v])
            end if
        end for
    end while
end function
```

Example

Node	dist[n]	prev[n]
A	0	
B	∞	
C	∞	A
D	∞	C
E	∞	B
F	∞	



Result

path A → C → D → B → E → F

distance 16

Note

- requires all non-negative weights
- guaranteed to find shortest path
- need priority queue (min-heap) for efficient operation
- does not use distance estimate information

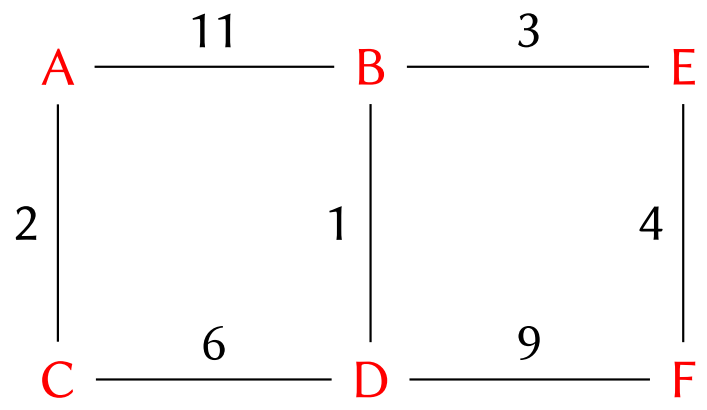
A*

Explore the graph using the neighbour of the already-visited nodes with the smallest estimated distance from the start node to the target node

```
function A*(G,start,end)
    dist ← new table(); prev ← new table()
    Q ← new min-heap(dist)
    for v ∈ G do
        dist[v] ← 0 if v = start else ∞; INSERT(Q,v)
    end for
    while ¬ EMPTY(Q) do
        u ← EXTRACT-MIN(Q)
        if u = end then
            s ← new stack()
            while u ≠ start do
                PUSH(s,u); u ← prev[u]
            end while
            return s
        end if
        for v ∈ NEIGHBOURS(G,u) do
            d ← dist[u] + WEIGHT(G,u,v) + H(v)
            if d < dist[v] then
                dist[v] ← d; prev[v] ← u; DECREASE-KEY(Q,v,dist[v])
            end if
        end for
    end while
end function
```

Example

Node	d(n,F)	dist[n]	prev[n]
A	14	0	
B	6	∞	
C	12	∞	A
D	7	∞	C
E	4	∞	B
F	0	∞	



Result

path A → C → D → B → E → F

distance 16

Note

- generalisation of Dijkstra's algorithm
- distance estimation H must be admissible
  - lower bound
  - non-negative
  - (Dijkstra's algorithm is A* with H(n) = 0)

Work

1. Reading
   - CLRS, chapter 24
   - Drozdek, sections 8.2, 8.3
2. Questions from CLRS

   Exercises 24.3-1