Project Orientation

…the sequel

Krysta Yousoufian

CSE 331 Section, 2/9/2012

Announcements

- Email course staff if using late day for HW4
- Office hours switchup
  - Me: 4-5 today, CSE 002
  - Hal: 3:30-4:30 tomorrow, CSE 002(?)

HW5

- Build a route-finder between UW buildings
- Given data:
  - Map of campus (won't need for this HW5)
  - Names, pixel coords of buildings
  - Start, end coords of path segments
- Nodes: points on the graph
  - Buildings
  - Non-building path endpoints
- Edges: path segments between points
  - Cost: length of path

Comparability

- Multiple routes between buildings
- Want route with shortest distance
- Natural *ordering* to routes:
  - route1 < route2 if route1 is faster
  - route1 == route2 if same distance
  - route1 > route2 if route1 is slower
- Routes are *comparable*

Comparable<E>

- Comparable<E>: Java interface for objects with a natural ordering
- Method: a.compareTo(b)
  - Returns < 0 if a < b
  - Returns 0 if a.equals(b)
  - Returns > 0 if a > b

- http://docs.oracle.com/javase/6/docs/api/index.html?overview-summary.html

Equality

- compareTo()
  - Classes with a natural ordering that implement Comparable
  - Should returns 0 iff equals() returns true
- equals()
  - Defined in all classes
  - Default: Object.equals() – reference equality
  - Can be overriden:

```
@Override
public void equals(Object other)
```

HashMaps

- You've seen dictionaries/maps
  - Set of <Key, Value> pairs
  - Lookup by key, get corresponding value
  - Student directory: <name, contact info>
  - *Webster's Dictionary*: <word, definition>
  - Time schedule: <SLN, class information>
- **Hashmaps**: O(1) lookup of keys
  - Lookup time is independent of size of map
  - Too good to be true?!
  - Java: HashMap or Hashtable (keys→values), HashSet (just keys)

Hash functions

- Hash tables use a **hash function**
  - Function from key (any object) to hash value (int)
- Used to index into array storing <key, value> pairs
- If you don't find the key at this index (or nearby), assume it's not in the list
- If "equal" objects have different hash values, lookups will fail – incorrectly report that object is not present

public int hashCode()

- Method provided by all Java objects
- Returns object's hash value
- Inherited from Object: if a == b, then a.hashCode() == b.hashCode()
- More generally: if a.equals(b), then a.hashCode() == b.hashCode()
  - If !a.equals(b), then it doesn't strictly matter either way
- **If you override equals(), you must override hashCode()**
  - Equal objects must have same hash code

Parsing the Data

- You write the parser (probably)
- Buildings:

  shortName longName xCoord yCoord

  CSE Paul G. Allen Center 1903.7201 1952.4322
- Paths:
  - Map from coord. pairs → connected coords. and distance

  (x,y) -> { (x,y,dist), … }

Path data example

- (1200, 1800) → {

    (1212, 1823): 10.5,

    (1250, 1795): 9.8,

    (1169, 1850): 12.4

  }

Path data example

- (1200, 1800) → {

    (1212, 1823): 10.5,

    (1250, 1795): 9.8,

    (1169, 1850): 12.4

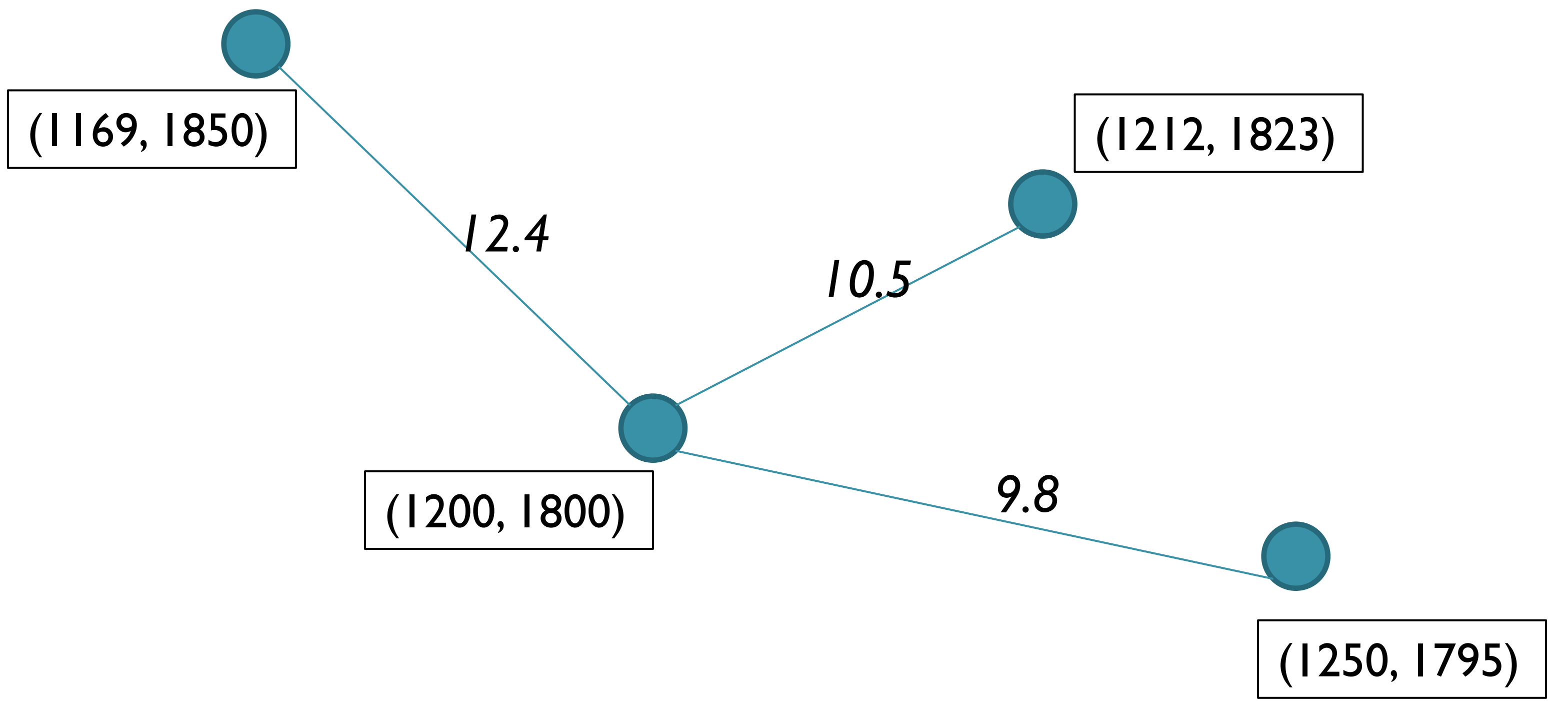
  }

There is a path segment to this point…

…with this length

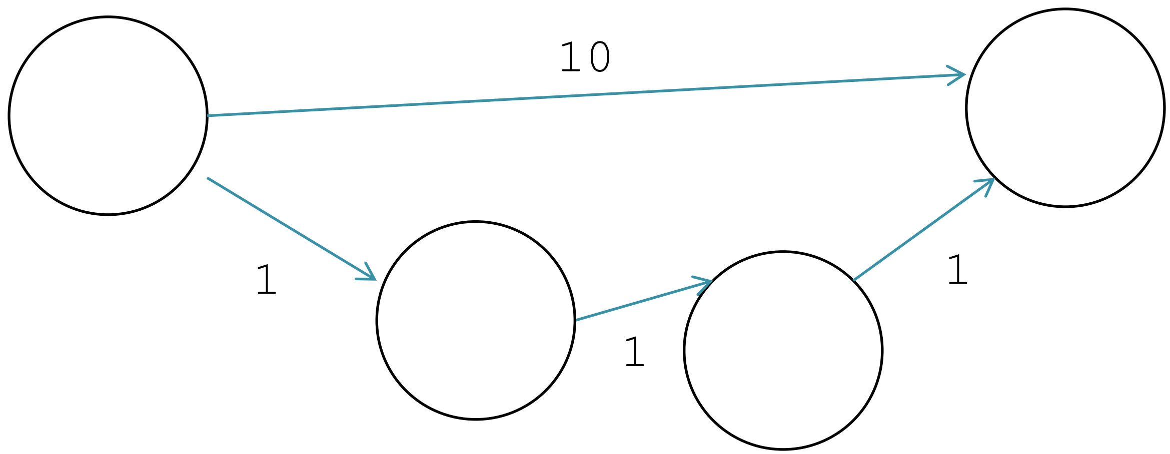
Visualization

(1200, 1800) → {
(1212, 1823): 10.5,
(1250, 1795): 9.8,
(1169, 1850): 12.4
}

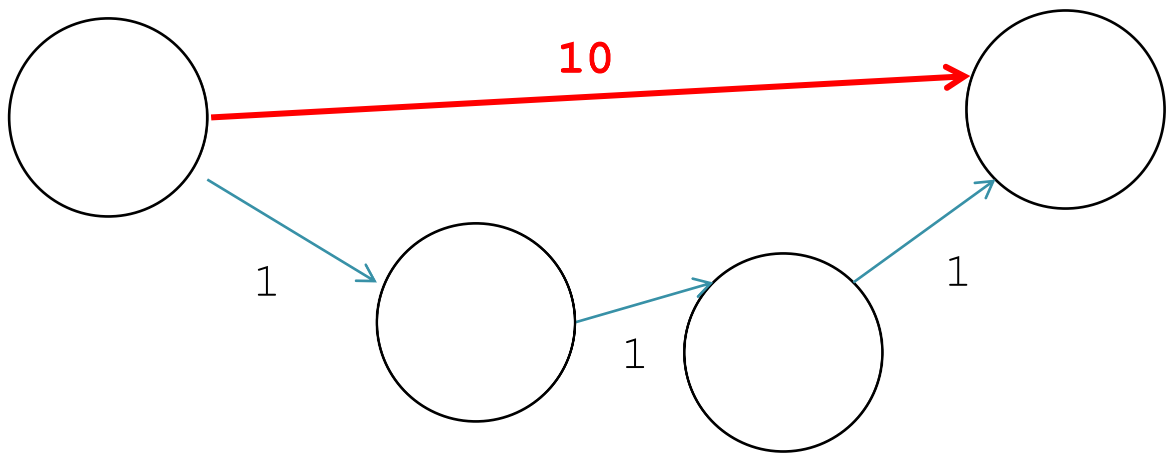


Minimum-Cost Paths

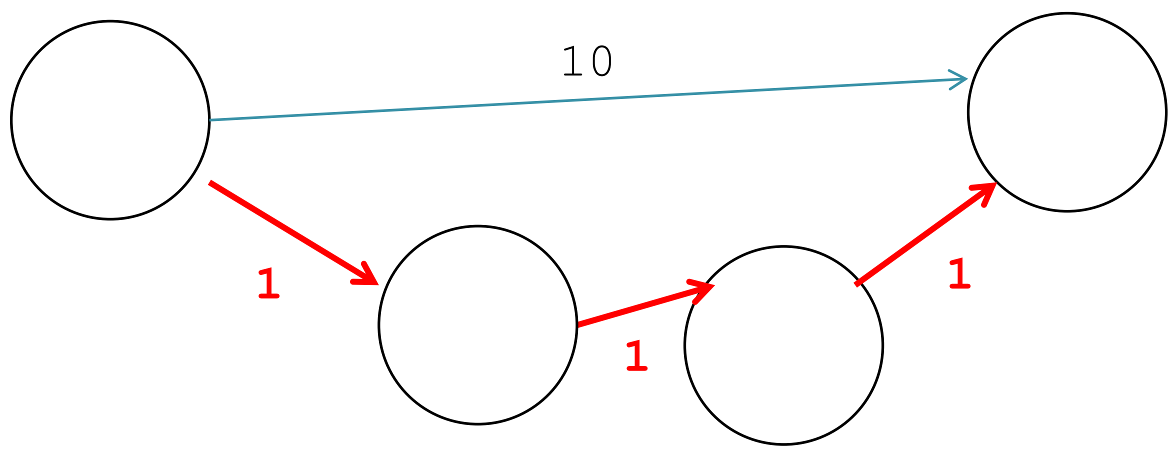
- Edge label is a *cost*
  - Money, time, …
  - Here, cost represents distance
- Want to find minimum-cost path
  - Not necessarily shortest path (in # of edges)


10
1
1
1

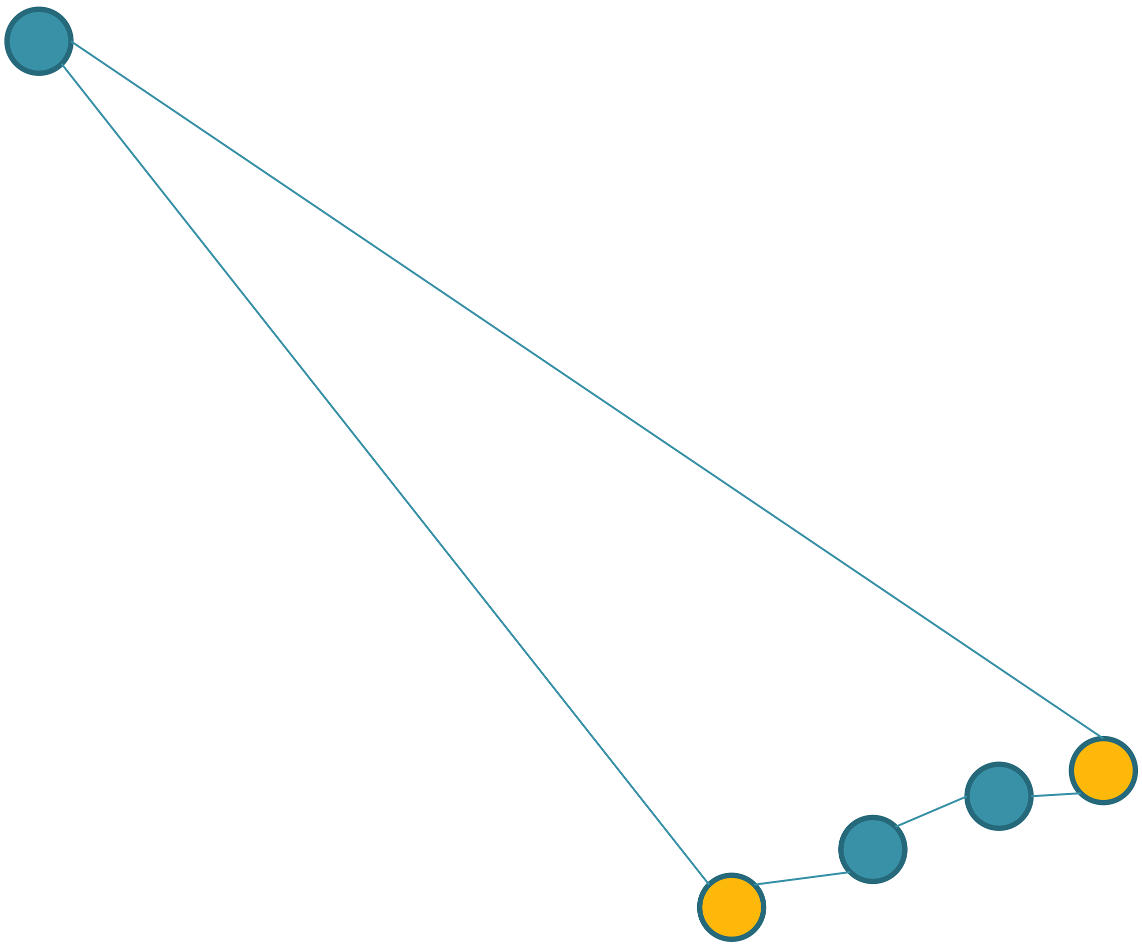
Shortest Path



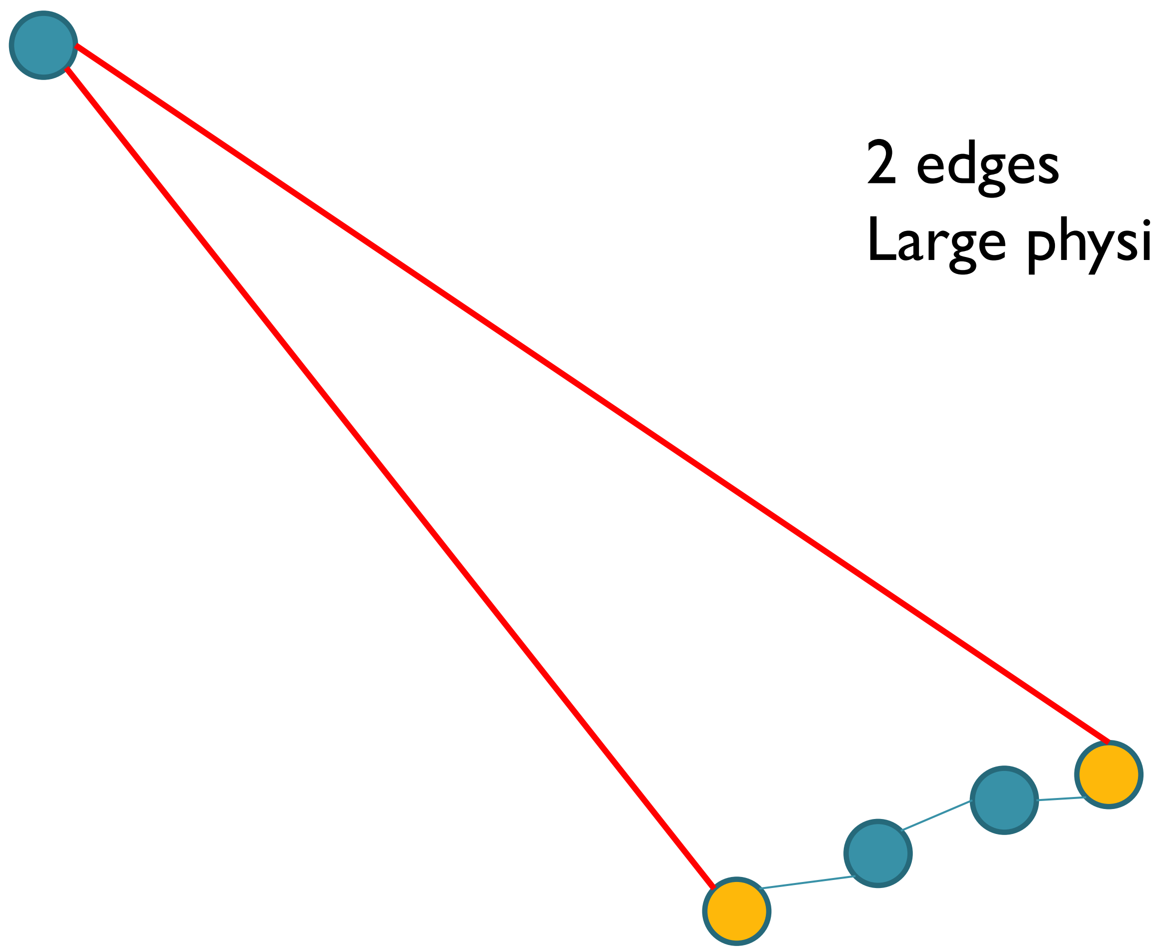
Min-Cost Path



Physical Map as Graph

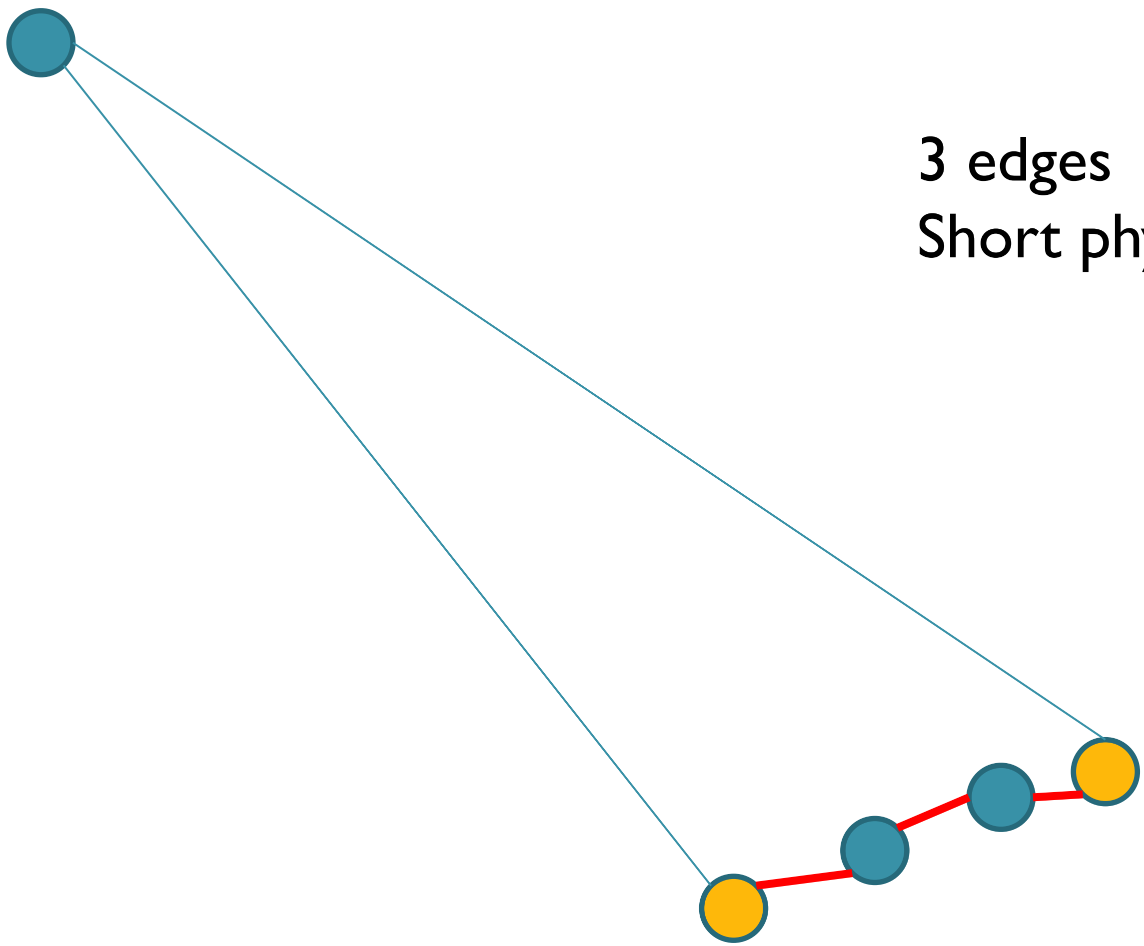


Physical Map as Graph



2 edges
Large physical distance

Physical Map as Graph



3 edges
Short physical distance

Priority Queues

- Set of objects with some ordering
- Knows the minimum object in the list
- Operations: add(), removeMin()
- (Assuming review; come to OH if not)

Priority Queues

- Application: Dijkstra's algorithm
  - Find min-cost path from source node `s`
  - Data: nodes
  - Ordering: min-cost path from `s` so far
- Java:
  - PriorityQueue

http://docs.oracle.com/javase/6/docs/api/index.html?overview-summary.html

  - Comparable

http://docs.oracle.com/javase/6/docs/api/index.html?overview-summary.html

  - Create `PriorityQueue<MyClass>` where

    `MyClass implements Comparable<MyClass>`

Dijkstra's Algorithm

- Want: min-cost path from s to t
- Get: min-cost path from s to all other nodes (traditionally)
- Optimized: stop when we find min-cost path for t

Dijkstra's Algorithm

Invariants:

- p(s,n) = min-cost path known *so far* from s to n for any node n
- c(s,n) = cost of p(s,n)
- `PriorityQueue active` = { n | we know *some* path p(s,n), but maybe not optimal path }
- `active` ordered by c(s,n)
- `Set finished` = { n | we know optimal path (s…n) }

Initial setup:

c(s,s) = 0

c(s,n) = infinity/unknown for s != n

Insert s into active

Dijkstra's Algorithm

While `active` is not empty:

Remove the element n with minimum c(s,n)

For each neighbor m of n:

If m is not in finished (i.e. haven't found optimal path):

Let e denote the edge <n,m>

If c(s,m) is unknown or is > c(s,n) + c(e)

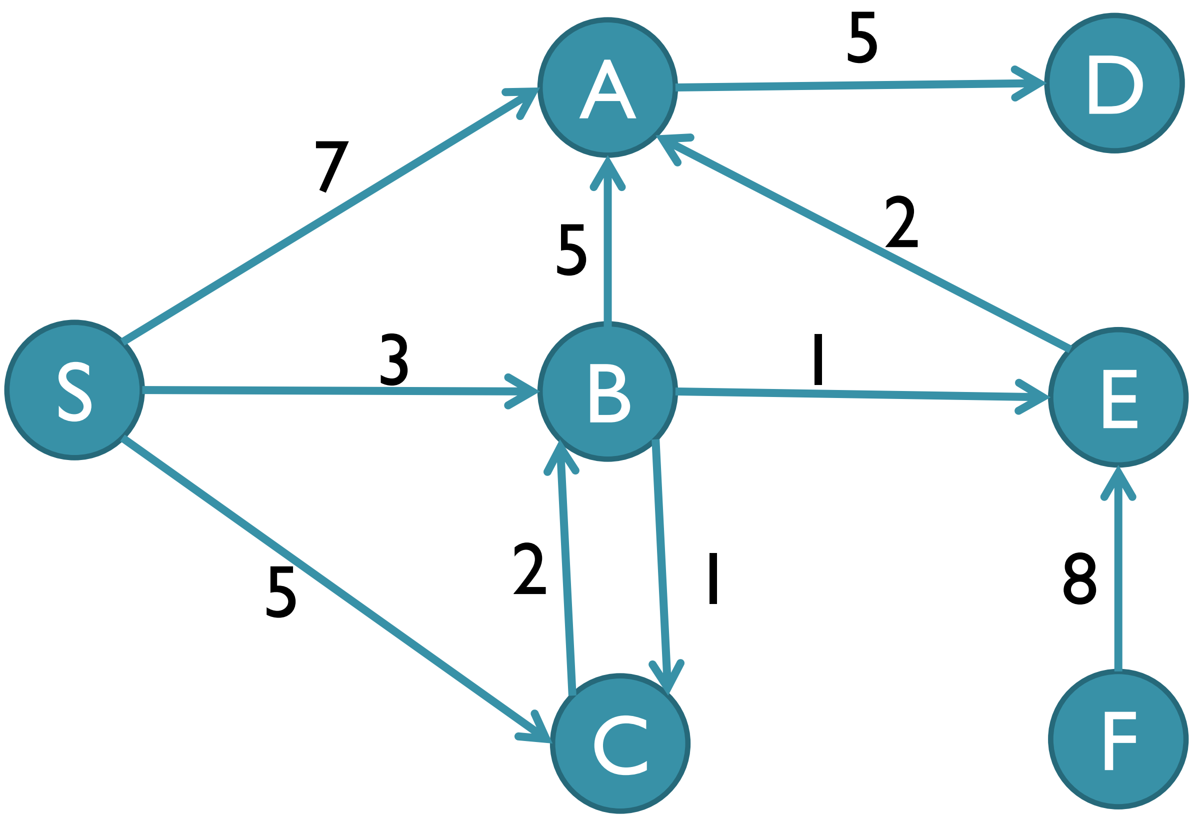
p(s,m) = p(s,n) + e

c(s,m) = c(s,n) + c(e)

Add or reorder m in `active`

Add n to `finished`

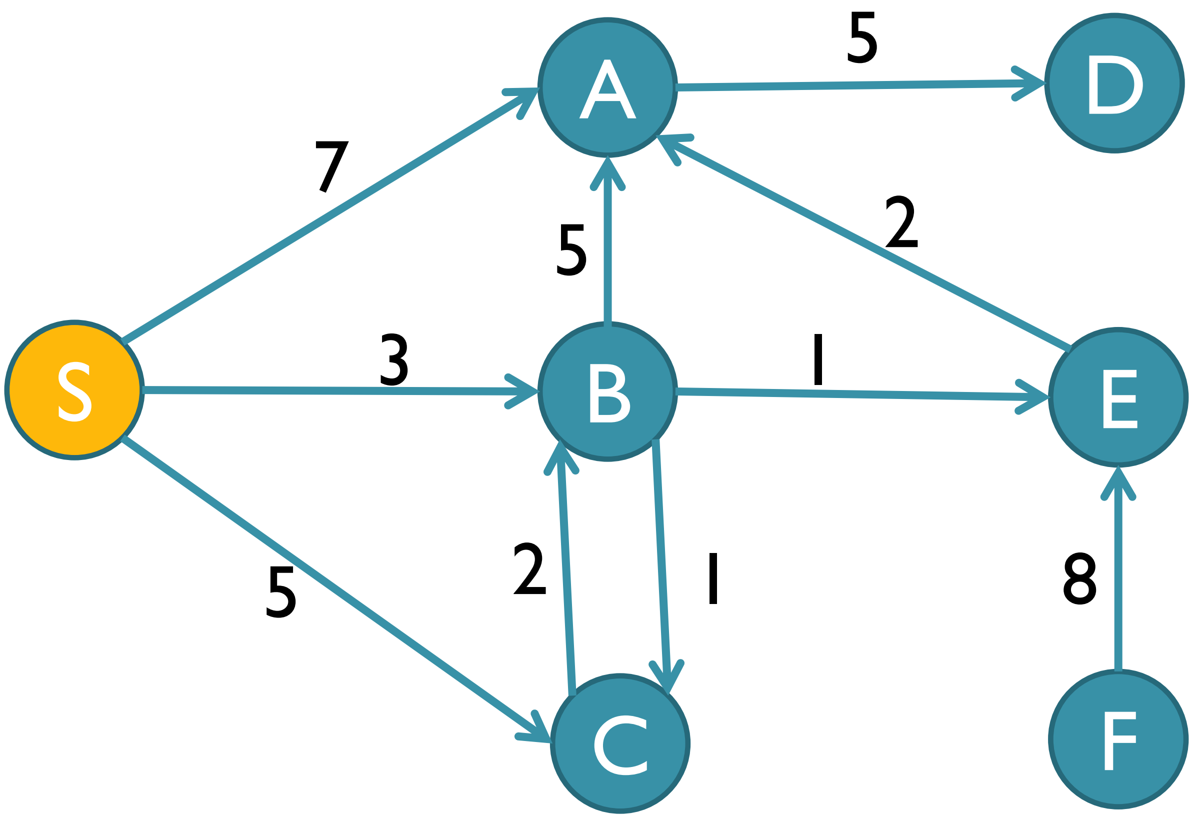
Example




```
active = {S}
finished = { }
Min-cost paths:
     A: ?
     B: ?
     C: ?
     D: ?
     E: ?
     F: ?
     S: 0
```

(from *Data Structures & Their Algorithms*, Lewis & Denenberg, 1991)

Example



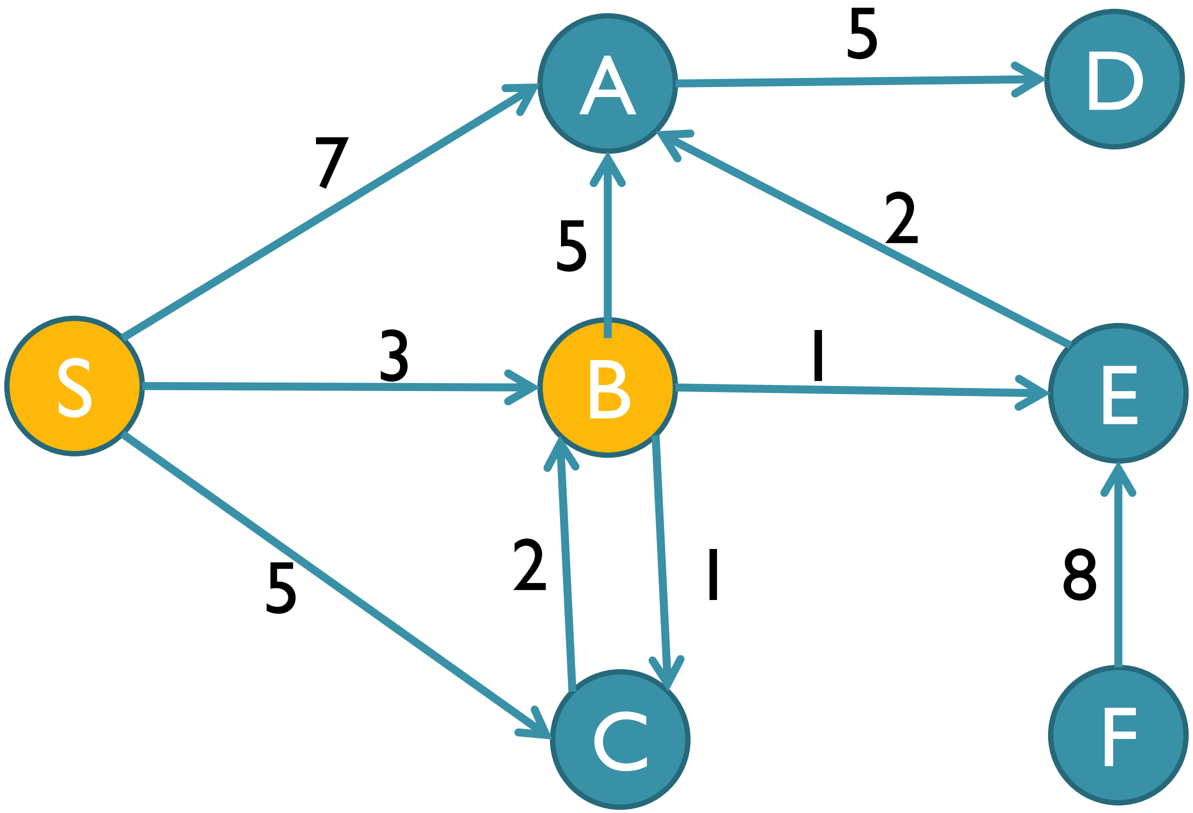

```
active = {A,B,C}
finished = {S}
```

```
Min-cost paths:
```

A: ~~?~~ 0+7 = 7
B: ~~?~~ 0+3 = 3
C: ~~?~~ 0+5 = 5
D: ?
E: ?
F: ?
S: 0

(from *Data Structures & Their Algorithms*, Lewis & Denenberg, 1991)

Example



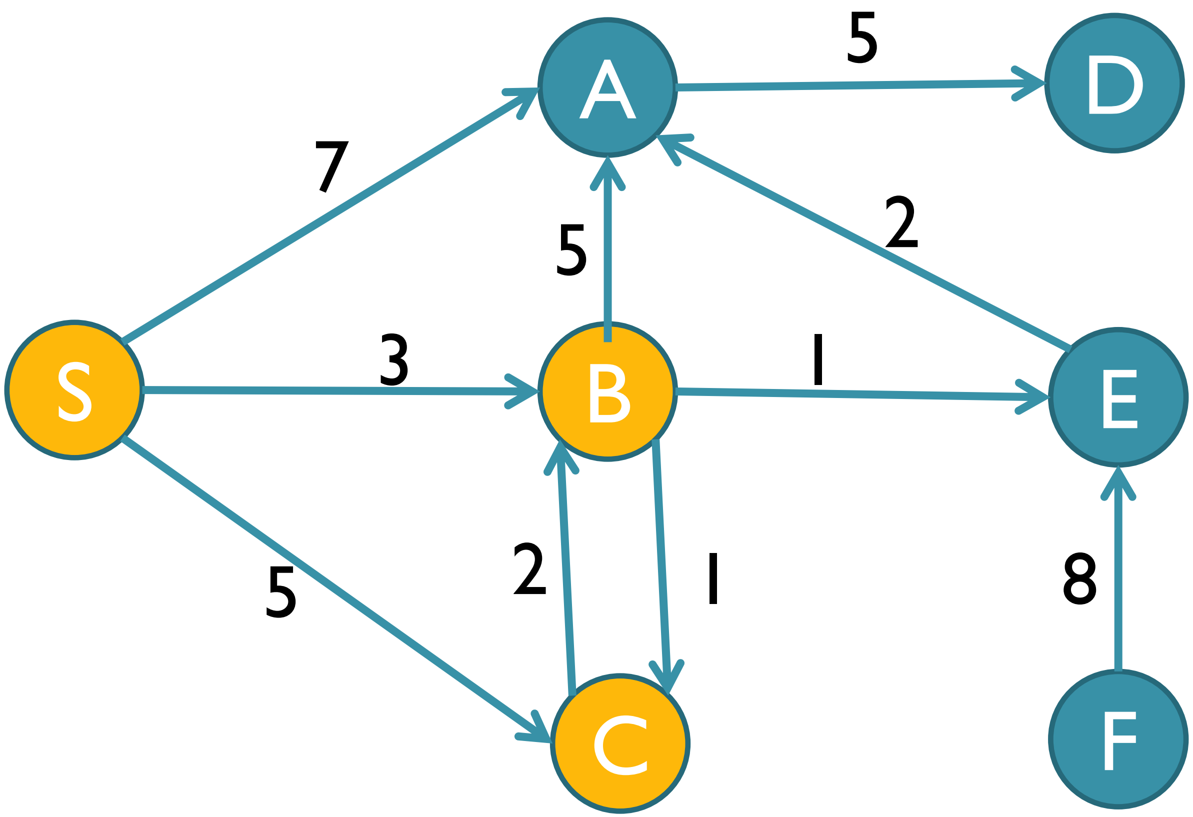

```
active = {A,C,E}
finished = {B,S}
```

```
Min-cost paths:
    A:  7
    B:  3
    C:  ~~5~~  3+1=4
    D:  ?
    E:  ~~?~~  3+1=4
    F:  ?
    S:  0
```

(from *Data Structures & Their Algorithms*, Lewis & Denenberg, 1991)

Example

```
active = {A,E}
finished = {B,C,S}
```

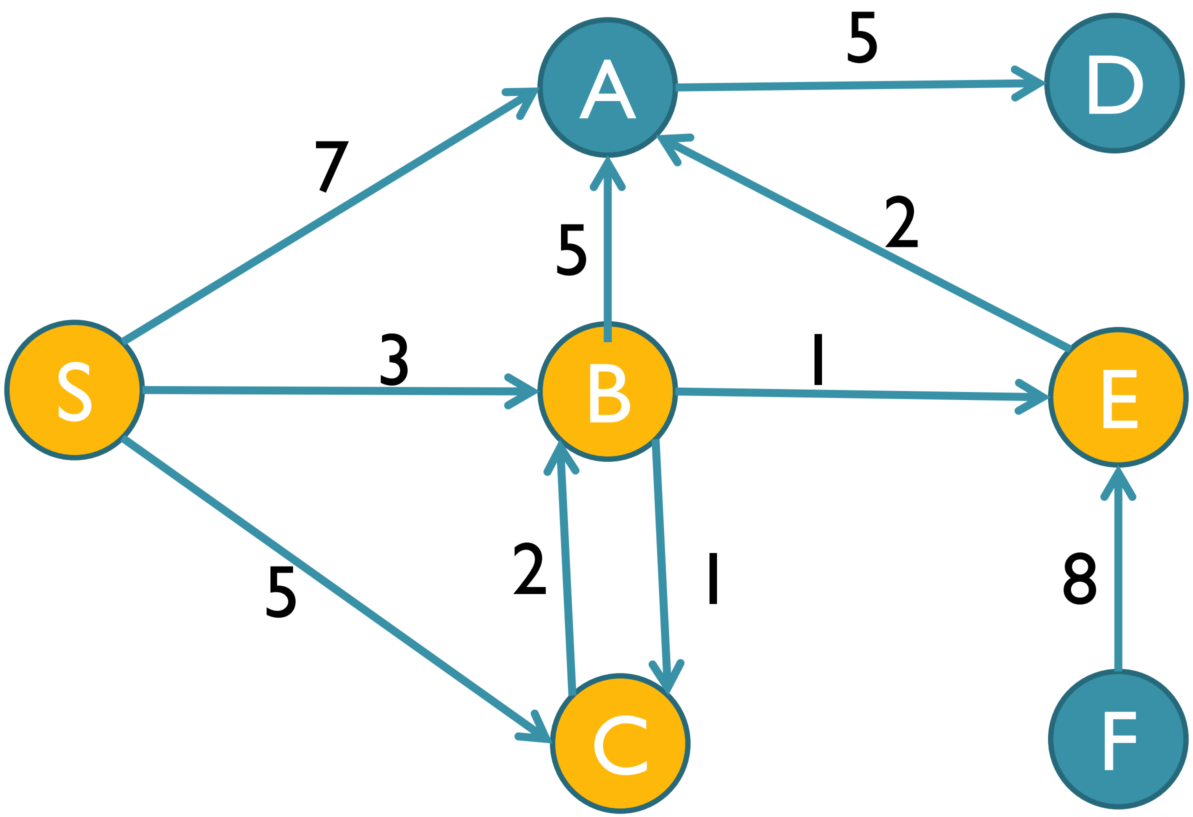



```
Min-cost paths:
    A: 7
    B: 3
    C: 4
    D: ?
    E: 4
    F: ?
    S: 0
```

(from *Data Structures & Their Algorithms*, Lewis & Denenberg, 1991)

Example

```
       active = {A}
finished = {B,C,E,S}
```

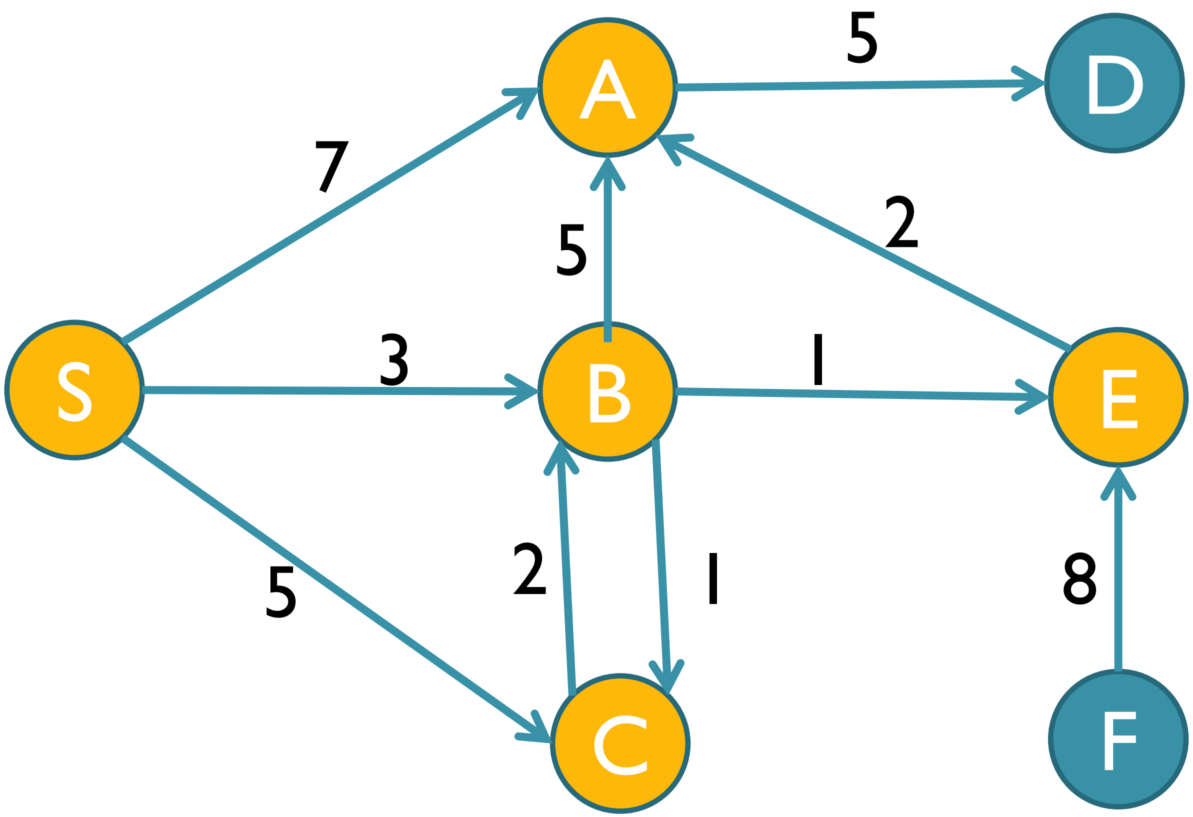



```
Min-cost paths:
    A: 7 4+2=6
    B: 3
    C: 4
    D: ?
    E: 4
    F: ?
    S: 0
```

(from *Data Structures & Their Algorithms*, Lewis & Denenberg, 1991)

Example

```
active = {D}
finished = {A,B,C,E,S}
```

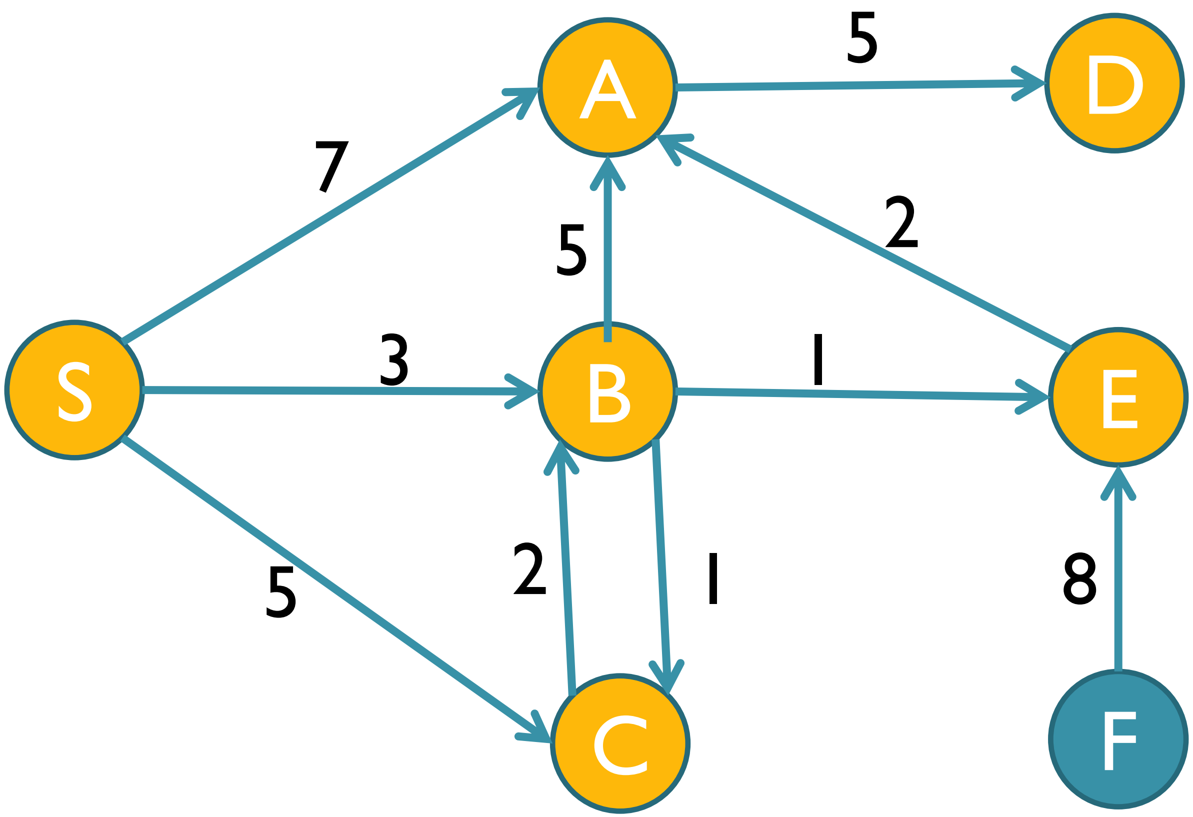



```
Min-cost paths:
    A: 6
    B: 3
    C: 4
    D: ? 6+5=12
    E: 4
    F: ?
    S: 0
```

(from *Data Structures & Their Algorithms*, Lewis & Denenberg, 1991)

Example

```
active = {}
finished = {A,B,C,D,E,S}
```




```
Min-cost paths:
    A: 6
    B: 3
    C: 4
    D: 11
    E: 4
    F: ?
    S: 0
```

(from *Data Structures & Their Algorithms*, Lewis & Denenberg, 1991)