Refactoring Bsuccessors

```
def bsuccessors3(state):
    """Return a dict of {state:action} pairs.  State is (here, the
    where here and there are frozen sets of people, light is 0 if
    on the here side and 1 if it is on the there side.
    Action is a tuple (travelers, arrow) where arrow is '->' or '<
    _,_,light = state
    return dict(bsuccessor3(state, set([a,b]))
                for a in state[light]
                for b in state[light])

def bsuccessors3(state, travelers):
    _,_,light = state
    start = state[light] - travelers
    dest = state[1-light] | travelers
    if light == 0:
        return (start, dest, 1), (travelers, '->')
    else:
        return (dest, start, 0), (travelers, '<-')

def test():
```

More Pour Problem

```
    def is_goal(state): return goal in state

    def more_pour_successor(state):
        indices=range(len(state))
        succ={}
        for i in indices:
            succ[replace(state,i,capacities[i])]=('fill,i')
            succ[replace(state,i,0)]=('empty',i)
            for j in indices:
                amount = min(state[i],capacities[j]-state[j])
                state2=replace(state,i,state[i]-amount)
                succ[replace(state2, j, state[j] + amount)]=('pour
            return succ
    if start is None : start = (0,)*len(capacities)
    return shortest_path_search(start,more_pour_successors, is_goa
```

Subway Plannig

```
def subway(**lines):
    """Define a subway map. Input is subway(linename='station1 st
    Convert that and return a dict of the form: {station:{neighbo
    successors = collections.defaultdict(dict)
    for linename,stops in lines.items():
        for a,b in overlapping_pairs(stops.split()):
            successors[a][b]=linename
            successors[b][a]=linename
    return successors

def overlapping_pairs(items):
    return [items[i:i+2] for i in range(len(items)-1)]

boston = subway(
```