Weighted Interval Scheduling (pre-class version)

January 31, 2025

[1]:
```python
import random

def random_request():
    return [sorted(random.sample(range(100),2)), random.random()*10]
```

[ ]:
```python
R = random_request()
print(R[0])
print(R[1])
```

[3]:
```python
def make_requests(n):
    return [random_request() for i in range(n)]
```

[ ]:
```python
make_requests(3)
```

[5]:
```python
def compatible(r1, r2):
    return r2[0][1] <= r1[0][0] or r2[0][0] >= r1[0][1]

def is_compatible(request, solution):
    return all(compatible(request, r) for r in solution)
```

[6]:
```python
def plot_requests(requests):
    for r in sorted(requests, key=lambda x : x[0][1]):
        print(" "*(r[0][0]) + "-"*(r[0][1]-r[0][0]) + "  (" +␣
 ↪str(round(r[1],2)) + ")")
    #print("total value:",sum(r[1] for r in requests))
    total_value = sum(r[1] for r in requests)
    print(f"total value: {total_value}")
```

[9]:
```python
# best = most valuable
def greedy(requests):
    sorted_requests = sorted(requests, key=lambda r: r[1], reverse=True)
    solution = []
    solution.append(sorted_requests.pop(0))

    while len(sorted_requests) > 0:
        request = sorted_requests.pop(0)
        if is_compatible(request, solution):
            solution.append(request)

    return solution
```

[10]:
```
requests = make_requests(100)
```

[ ]:
```
plot_requests(requests)
```

[ ]:
```
sol = greedy(requests)
print(sol)
print(len(sol))
```

[ ]:
```
plot_requests(sol)
```

[ ]:
```

```

[ ]:
```
sum(s[1] for s in sol)
```

[ ]:
```
# best = most valuable
# best = shortest
# best = most value-dense (highest value/duration)
```

[16]:
```
def greedy(requests, sort_function):
    sorted_requests = sorted(requests, key=sort_function)
    solution = []
    solution.append(sorted_requests.pop(0))

    while len(sorted_requests) > 0:
        request = sorted_requests.pop(0)
        if is_compatible(request, solution):
            solution.append(request)

    return solution
```

[ ]:
```

```

[17]:
```
# request = [[start, end], value]
most_value = lambda req : -req[1]
shortest = lambda req : req[0][1] - req[0][0]
density = lambda req : -req[1]/(req[0][1] - req[0][0])
```

[ ]:
```
print(most_value)
print(most_value([[10,20], 15.1]))
```

[18]:
```
requests = make_requests(1000)
```

[19]:
```
s1 = greedy(requests, most_value)
s2 = greedy(requests, shortest)
s3 = greedy(requests, density)
```

[ ]:
```
plot_requests(s1)
```

[ ]:
```
plot_requests(s2)
```

[ ]:
```
plot_requests(s3)
```

[ ]:
```

```

[ ]:
```
requests = make_requests(100_000)
s1 = greedy(requests, most_value)
s2 = greedy(requests, shortest)
s3 = greedy(requests, density)
def score(sol):
    return sum(s[1] for s in sol)
print([score(s1), score(s2), score(s3)])
```

[ ]:
```

```

[ ]:
```

```