**San Skulrattanakulchai**
Gustavus Adolphus College


# Flight Problem

### Problem

We are given $n$ cities $1, 2, \ldots, n$ in that order, with cost $c(i, j)$ for flying a plane from city $i$ to city $j$ for all $1 \le i < j \le n$. We wish to fly from city 1 to city $n$, stopping at any number of these intermediate cities if we wish. The cost function is arbitrary. Compute a cheapest route to fly from city 1 to city $n$.

### Solution by Dynamic Programming

For each $1 \le i \le n$, let $m(i)$ be the cheapest cost of flying from city 1 to city $i$.
We know that $m(1) = 0$ since it costs nothing to fly to from city 1 to city 1.
For each $i$ such that $1 < i \le n$, we know that the last leg of the cheapest flight from 1 to $i$ goes from some city $k$ to city $i$, where $1 \le k < i$. So $m(i) = m(k) + c(k, i)$, since we must have flown from the city 1 to city $k$ as cheapest as possible. We don't know what $k$ is, but it must be some value between 1 and $i - 1$ inclusive. So we know that $m(i) = \min\{\, m(k) + c(k, i) : 1 \le k < i \,\}$.
Therefore, $m(i)$ satisfies the recurrence

$$m(i) = \begin{cases} 0 & \text{if } i = 1 \\ \min\{\, m(k) + c(k, i) : 1 \le k < i \,\} & \text{if } i > 1 \end{cases}$$

The following code solves the recurrence by filling a table $M[\cdot]$.

$M[1] \leftarrow 0$
**for** $i \leftarrow 2$ **to** $n$ **do** {
  $M[i] \leftarrow \infty$
  **for** $k \leftarrow 1$ **to** $i - 1$ **do**
    $M[i] \leftarrow \min(M[i], M[k] + C[k, i])$
}

Suppose the cost function $c(\cdot, \cdot)$ is given as the following 2d-array of numbers

| | $i:2$ | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $k:1$ | 100 | 200 | 300 | 400 | 500 | 600 |
| 2 | | 50 | 100 | 150 | 200 | 300 |
| 3 | | | 60 | 100 | 115 | 240 |
| 4 | | | | 40 | 70 | 100 |
| 5 | | | | | 40 | 40 |
| 6 | | | | | | 20 |

The filled-in table $M$ then looks like

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $M[i]$ | 0 | 100 | 150 | 200 | 240 | 265 | 280 |

This is Dynamic Programming, a technique for solving a problem by

(i) developing a recurrence, and

(ii) solving the recurrence by filling in a table.

The value of $M[n]$ is the cheapest cost to fly from city 1 to city $n$.
In order to find the corresponding route, we can record the minimizers in our dynamic programming table like so:

| $i$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $M[i]$ | 0 | 100 | 150 | 200 | 240 | 265 | 280 |
| $K[i]$ | | 1 | 2 | 2 | 4 | 3 | 5 |

The $K[\cdot]$ table is the minimizer table.
Since 5 is the miminizer for $m(7)$, the cheapest way to fly to 7 is to first fly to 5 in the cheapest way, and then fly directly to 7.
Similarly, we reach 5 by flying directly from 4.
We reach 4 by flying directly from 2.
We reach 2 by flying directly from 1.
I.e, we find the cheapest route by scanning the table backwards via the $K[i]$ "pointers."
This shows that the cheapest route costs 280 with itinerary $1 \to 2 \to 4 \to 5 \to 7$.
**Question:** What is curious about the cheapest way to fly to city 6?
We can print the cheapest route with a simple recursive algorithm.

```
/* print out the cheapest route from 1 to k, starting at city 1 */
STOPS(k) {
    if k = 1 then
        print 1
    else {
        STOPS(K[k])
        print k
    }
}
```

To print out the cheapest route to city $n$, call STOPS($n$).
The running time for filling out the $M[\cdot]$ and $K[\cdot]$ tables is $O(n^2)$.
The running time for printing out the cheapest route is $O(n)$.

**Notes**

1. The cost function does not have to be given as a table. It may be given as a formula, for example. We assume in this handout that given any cities $i$, $j$, the cost $c(i, j)$ can be found in $O(1)$ time.

2. Instead of making the destination city the variable, we can also make the starting city the variable.

3. We can solve the problem with extra conditions by either modifying the cost function, or the recurrence.

4. Problems admitting solutions by dynamic programming exhibit the *Optimal Substructure Property*—embedded within an optimal solution are optimal solutions to subproblems.