Problem Solving Techniques 문제해결

Jinkyu Lee

Dept. of Computer Science and Engineering,
Sungkyunkwan University (SKKU)

Contents

- Chapter 11 – Dynamic Programming
  - Program design example

Design Example: Elevator Optimization <0>

❖ Problem Description

- I work in a very tall building with a very slow elevator.
  It is frustrating for me when people press the buttons for many consecutive floors.
- Thus, we need to write an elevator optimization program
- The riders all enter their intended destinations at the beginning of the trip.
- We limit the elevator to making at most k stops on any given run.
- We assume that the penalty for walking up and down is same.
- Management proposes to break ties among equal-cost solutions by given preference to stopping the elevator at the lowest floor.
- Elevator does not necessary to stop at one of the floors the riders specified.
  - Ex.) If riders specify floors 27 and 29, it can be decided to stop at floor 28.
- The aim is to select the floors to be stopped, so as to minimize the total number of floors people have to walk either up or down.

Design Example: Elevator Optimization 〈1〉

- **Can you solve this problem by backtracking?**
  - **For example,**
    - **NFLOORS=110, MAX_RIDERS=50**
    - **nriders=5, nstops=3**

3

16

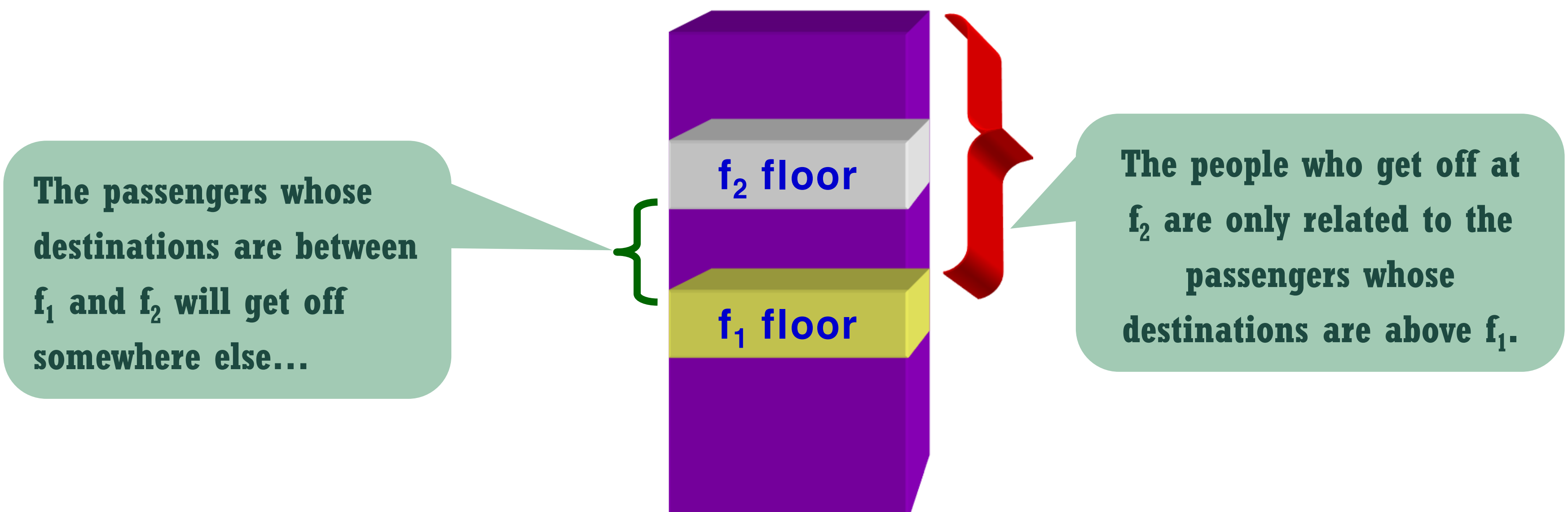
2

10

15

Design Example: Elevator Optimization ‹2›

- **Problem-solving strategy using Dynamic Programming**
  - Recall that Dynamic programming is based on Recursive algorithms!
  - Thus, deciding the best place to put the kth stop depends on the cost of all possible solutions with k-1th stops.
  - Consider a trip that stops at floor f_2, after an initial stop at f_1.
  - The second stop (f_2) can be of no interest to any passenger whose destination is at or below f_1; Thus, the problem can be decomposed into two pieces.
  - At this juncture, we smell Dynamic programming!



Design Example: Elevator Optimization <2>

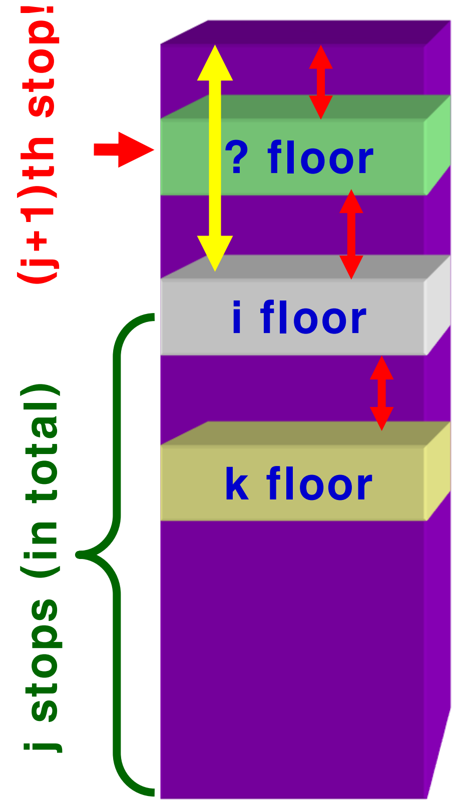
- **Solution approach**
  - Incrementally add a new stop, which is higher than all the selected stops
  - We will store the minimum cost of the situation where the number of stops is j and the last (highest) stop is at floor i.

- **Why is it possible to apply dynamic programming for this decomposition?**

Design Example: Elevator Optimization <3>

Recurrence Relation for Dynamic Prog.

- The following characteristics can be obtained from the previous investigation.
  - At first, define $m_{i,j}$ as the minimum cost of serving *all* the riders using exactly j stops, the last of which is at floor i.
  - Then, the (j+1)th stop must be higher than the previous jth stop at floor i. Further the new stop will only be of interest to the passengers seeking to get above the ith floor.
  - We must properly divide the passengers between the new stop, i.e., (j+1)th stop, and floor i based on which stop they are closer to.



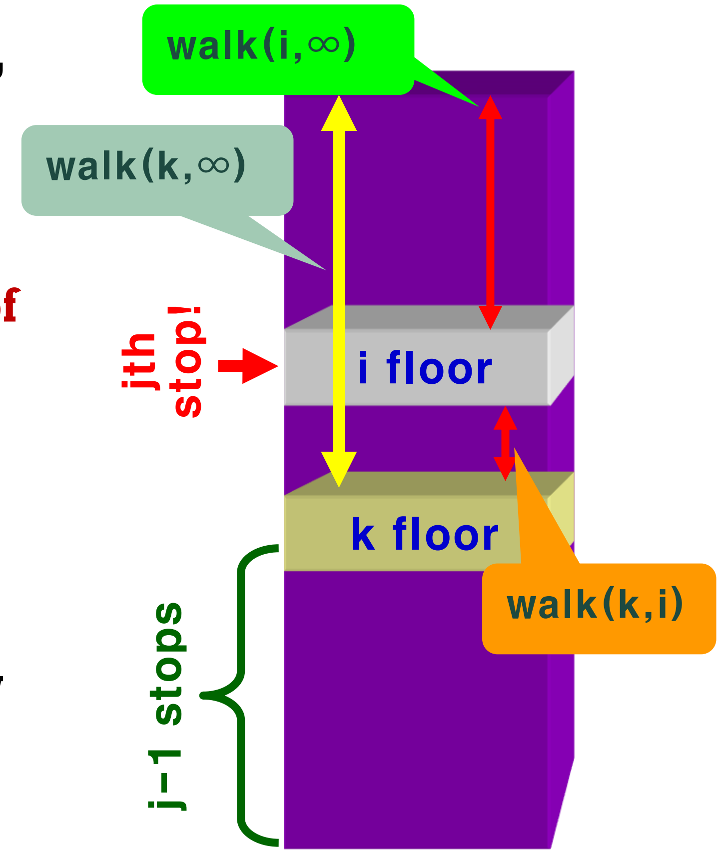
Design Example: Elevator Optimization <4>

- **Recurrence Relation for Dynamic Programming (cont.)**
  - **The previous idea defines the following recurrence**
  - $m_{i,j}=\min_{\{k=0\ldots i\}}\{m_{k,j-1} - walk(k, \infty) + walk(k, i) + walk(i, \infty)\}$
    - **$m_{i,j}$ as the minimum cost of serving *all* the riders using exactly j stops, the last of which is at floor i.**
    - **The key is the function walk(a,b), which counts the total number of floors walked by passengers whose destinations are between floor a and floor b.**

Design Example: Elevator Optimization <4>

Recurrence Relation for Dynamic Programming (cont.)

- The previous idea defines the following recurrence
- $m_{i,j}=\min_{\{k=0\ldots i\}}\{m_{k,j-1} - walk(k, \infty) + walk(k, i) + walk(i, \infty)\}$
  - If the last stop is at floor i (at the jth stop), the previous stop has to be at some floor k (at the (j-1)th stop) lower than floor i.
  - Then, $m_{i,j}$ is given by subtracting the cost of serving all passengers above floor k (walk(k,∞)) from $m_{k,j-1}$, and adding the cost of stopping at floor i (walk(k,i)+walk(i,∞)).
  - The key is the function walk(a,b), which counts the total number of floors walked by passengers whose destinations are between floor a and floor b.



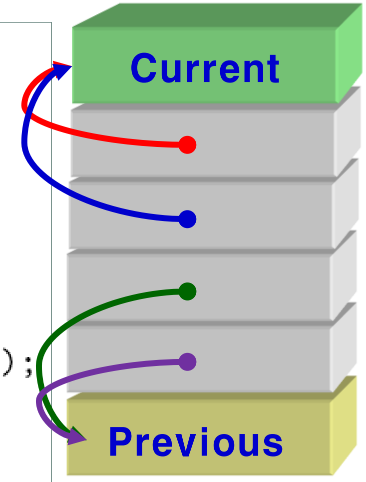
Design Example: Elevator Optimization <5>

- Set up global matrices to hold the dynamic programming table

```
#define NFLOORS         110     /* the building height in floors */
#define MAX_RIDERS      50      /* what is the elevator capacity? */

int stops[MAX_RIDERS];          /* what floor does everyone get off? */
int nriders;                    /* number of riders */
int nstops;                     /* number of allowable stops */

int m[NFLOORS+1][MAX_RIDERS];   /* dynamic programming cost table */
int p[NFLOORS+1][MAX_RIDERS];   /* dynamic programming parent table */
```

- The function that counts the total number of floors walked by passengers when the elevator stops at consecutive previous and current floors

```
floors_walked(int previous, int current)
{
        int nsteps=0;           /* total distance traveled */
        int i;                  /* counter */

        for (i=1; i<=nriders; i++)
                if ((stops[i] > previous) && (stops[i] <= current))
                        nsteps += min(stops[i]-previous, current-stops[i]);

        return(nsteps);
}
```

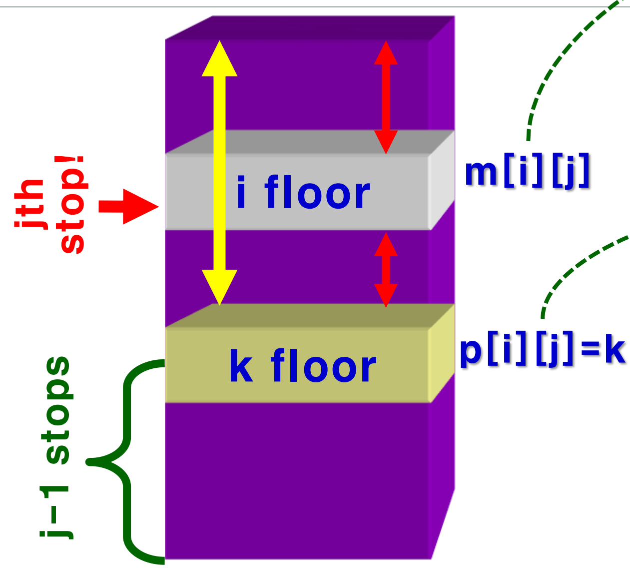


Design Example: Elevator Optimization <6>

- A direct implementation of the recurrence, with care taken to order the loops

```
int optimize_floors()
{
        int i,j,k;                  /* counters */
        int cost;                   /* costs placeholder */
        int laststop;               /* the elevator's last stop */

        for (i=0; i<=NFLOORS; i++) {
                m[i][0] = floors_walked(0,MAXINT);
                p[i][0] = -1;
        }

        for (j=1; j<=nstops; j++)
              for (i=0; i<=NFLOORS; i++) {
                   m[i][j] = MAXINT;
                   for (k=0; k<=i; k++) {
                        cost = m[k][j-1] - floors_walked(k,MAXINT) +
                           floors_walked(k,i) + floors_walked(i,MAXINT);
                        if (cost < m[i][j]) {
                           m[i][j] = cost;
                           p[i][j] = k;
                        }
                   }
              }

        laststop = 0;
        for (i=1; i<=NFLOORS; i++)
                if (m[i][nstops] < m[laststop][nstops])
                        laststop = i;

        return(laststop);
```

Find the floor at the (nstops)th stop!



Design Example: Elevator Optimization 〈7〉

❖ **Finally, we need to reconstruct solution by following the parent pointers and work backward.**

```
reconstruct_path(int lastfloor, int stops_to_go)
{
        if (stops_to_go > 1)
            reconstruct_path(p[lastfloor][stops_to_go], stops_to_go-1);

        printf("%d\n",lastfloor);
}
```