Homework 4a

2021315385
이건 / Gun Daniel Lee

Problem explanation

- Elevator Optimization Problem

- Given only *k* number of possible stops, find the stop for each rider such that the total number of floors required to move up or down is minimized

- Given the choice, always prioritize the highest floor

Problem explanation

Input:

- Number of riders (1 - 500)
- Number of stops (1 - 20)
- Each rider’s intended destination (1 - 100)

Problem explanation

Output:

- Floor each rider stops at

**floor is assumed to be the most optimal solution among the possible stops*

Solution explanation

- Not much has to be altered from the original elevator optimization problem explained in Lecture 11

- The only change needed is to consider the higher floor possible

Solution explanation

- Required global variables (adjusted to the problem)

```
#define NFLOORS 100 //buiding height in floors
#define MAX_RIDERS 500 //elevator capacity

int stops[MAX_RIDERS]; //floor that each passenger gets off
int nriders; //num of riders
int nstops; //num of allowed stops
int MAXINT = 100000; //a decently sized maxint
int count = 0; //

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

Solution explanation

- Min function (outputs the minimum between to integers)

```
int min(int a, int b)
{
  if(a < b)
  {
    return a;
  }

  return b;
}
```

Solution explanation

- Floors_walked function

- Outputs the total number of steps required for the riders between the *previous* and *current* floor

```
int floors_walked(int previous, int current)
{
  int nsteps = 0;
  int i;

  for(i = 0; i < nriders; i++)
  {
    if((stops[i] > previous) && (stops[i] <= current))
    {
      nsteps += min(stops[i] - previous, current - stops[i]);
    }
  }
  return nsteps;
}
```

Solution explanation

- Optimize_floors function

- Optimizes the number of stops made based on the rider's stops

```
int optimize_floors()
{
  int i, j, k;
  int cost;
  int laststop;

  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);
```

Solution explanation

- Optimize_floors function (CONTINUED)

- Optimizes the number of stops made based on the rider's stops

```
            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;
}
```

Solution explanation

- Reconstruct_path function

- Finds and stores the optimal stops in the floor array through recursion

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

  floor[count] = lastfloor;
  count++;
}
```

Solution explanation

- Main function
- Scan variables

```
int main()
{
  scanf("%d", &nriders);
  scanf("%d", &nstops);

  int floor[nstops];

  for(int i = 0; i < nriders; i++)
  {
    scanf("%d", &stops[i]);
  }
```

Solution explanation

- Main function (CONTINUED)

- Find optimal solution for the stops

```
int opt = optimize_floors();

reconstruct_path(opt, nstops, floor);
```

Solution explanation

- Main function (CONTINUED)

- For each rider, find the optimal stop among all the possible stops and print this optimal stop

- Since highest stop is preferred if number of steps needed are the same, take the higher stop

```
  for(int i = 0; i < nriders; i++)
  {
    int temp_min = MAXINT;
    int temp_floor;
    for(int j = 0; j < nstops; j++)
    {
      if(abs(stops[i] - floor[j]) <= temp_min)
      {
        temp_min = abs(stops[i] - floor[j]);
        temp_floor = floor[j];
      }
    }
    printf("%d\n", temp_floor);
  }
}
```

Solution explanation

- Changes made:

- In all cases were minimum floor had to be found, the if conditions were all changed from (if less than) to (if less than or equal to)
- This allows the highest floor preference to be considered

Thank you!