EECS 3101 - A Tutorial Notes

Shahin Kamali

York University

Fall 2023

1. Consider the following input for Bucket-Sort which is NOT uniformly sorted:

   - one-third of the items are uniformly distributed in $[0, 0.3]$.
   - one-third of items are uniformly distributed in $[0.3, 0.4]$.
   - the remaining one-third are uniformly distributed in $(0.4, 1]$.

   Repeat the analysis from the class to describe the expected running time of Bucket-Sort with parameter $k = \sqrt{n}$ for the above input. You may assume that we a comparison-based sorting algorithm with running time $\Theta(n \log n)$ to sort items within each bucket.

   Answer: As before, distributing items between buckets takes $\Theta(n)$.

   - The first group of $0.3k = 0.3\sqrt{n}$ buckets receive one-third of the numbers, that is, $n/3$ items. The expected number of items in each bucket is thus $(n/3)/(0.3\sqrt{n}) = \sqrt{n}/0.9 = \Theta(\sqrt{n})$. Therefore, sorting each of these buckets takes $\Theta(\sqrt{n} \log \sqrt{n}) = \Theta(\sqrt{n} \log n)$.
   - The second group of $0.1k = 0.1\sqrt{n}$ buckets receive one-third of the numbers, that is, $n/3$ items. The expected number of items in each bucket is thus $(n/3)/(0.1\sqrt{n}) = \sqrt{n}/0.3 = \Theta(\sqrt{n})$. Therefore, sorting each of these buckets takes $\Theta(\sqrt{n} \log \sqrt{n}) = \Theta(\sqrt{n} \log n)$.
   - The third group of $0.6k = 0.6\sqrt{n}$ buckets receive one-third of the numbers, that is, $n/3$ items. The expected number of items in each bucket is thus $(n/3)/(0.6\sqrt{n}) = \sqrt{n}/1.8 = \Theta(\sqrt{n})$. Therefore, sorting each of these buckets takes $\Theta(\sqrt{n} \log \sqrt{n}) = \Theta(\sqrt{n} \log n)$.

   Therefore, sorting all buckets take $\sqrt{n} \times \Theta(\sqrt{n} \log n) = \Theta(n \log n)$. The overall complexity of the algorithm is thus $\Theta(n \log n)$.

2. **Egg Dropping Puzzle:** Suppose that we have n eggs, and wish to know which stories in a k-story building are safe to drop eggs from, and which will cause the eggs to break on landing. We make a few assumptions:

   - An egg that survives a fall can be used again.But a broken egg must be discarded.
   - The effect of a fall is the same for all eggs.
   - If an egg breaks when dropped, then it would break if dropped from a higher floor. Similarly, if an egg survives a fall then it would survive a shorter fall.
   - It is not ruled out that the first-floor windows break eggs, nor is it ruled out that the kth-floor do not cause an egg to break.

Use a DP approach to find the floors from which eggs should be dropped so that the total number of trials are minimized. In other words, we need to find a *critical floor*, which is the lowest floor such that the egg breaks dropping from it.

Answer:

- **Step 1:** Let $eggDrop(m, j)$ denote the minimum number of trials required to detect the critical floor among j floors $(1 \leq j \leq k)$ using m eggs $(1 \leq m \leq n)$; we want to find $eggDrop(n, k)$.
  Setting $eggDrop(m, j)$: suppose we drop from floor x (where $1 \leq x \leq j$) in the first trial.
  - If the egg breaks, the critical floor is in $[1, x]$. We need to search for the critical floor among $x - 1$ floors using $m - 1$ eggs.
  - if it survives, the critical floor is in $[x + 1, j]$. We need to search between $j - x$ floors using m eggs.
- **Step 2:** Write a recursive formula for the minimum number of trials using m eggs and for j stories. We can write:

$$eggDrop(m, j) = \begin{cases} j & \text{if } j \leq 1 \\ j & \text{if } m = 1 \\ \min\limits_{x \in [1,j]} \max\{eggDrop(m-1, x-1), eggDrop(m, j-x)\} \end{cases}$$

3. **Step 3:** Fill the DP table bottom up. In this case, for setting $eggDrop(m, j)$ we look at the previous rows and columns. We can use the following simple code:

```
EggDropTable(n, k)
1.   for $j = 1$ to $k$
2.       for $m = 1$ to $n$
3.           if $j \leq 1$ or $m = 1$
4.               $eggDrop(m, j) \leftarrow j$
5.           else
6.               $eggDrop[m, j] \leftarrow \infty$
7.               for $x = 1$ to $j$
8.                   $candidate[x] \leftarrow \max\{eggDrop[m-1, x-1], eggDrop[m, j-x]\}$
9.                   if $candidate[x] < eggDrop[m, j]$
10.                      $eggDrop[m, j] \leftarrow candidate[x]$
11.  return eggDrop
```

4. **Step 4:** In Step 4, we usually move backward in the table to find the actual solution. Here, the question only asks for the number of trials. If the actual stories were intended, we had to store a flag in Line 10: $flag[m, j] \leftarrow x$

5. Given a rope of length n meters, cut the rope in different parts of integer lengths in a way that maximizes product of lengths of all parts. You must make at least one cut. Assume that the length of rope is more than 2 meters.

   Answer:
   Step 1: Let $maxProd(m)$ be the maximum product for a rope of length m for $m \in [1, n]$; we want to find $maxProd(n)$.

Step 2: To set $maxProd[m]$, suppose the first cut has length i. If the remainder of $n-i$ is not cut any more, the profit is $i \times m - i$; otherwise, the profit is $i \times maxProd[m-i]$. So, for a cut of length i, the profit will be $\max\{i \times (m-i), i \times maxProd[m-i]\}$. Therefore $maxProd(m)$ can be written as following.

$$maxProd(m) = \begin{cases} 0 & \text{if } m \leq 1 \\ \max\limits_{i \in [1,m]} (\max\{i \times (m-i), i \times maxProd[m-i]\}) \end{cases}$$

Step 3: Fill the DP table:

```
RopeCutTable(n)
1.  for $m = 1$ to $n$
2.          if $m \leq 1$
3.              $maxProd(m) \leftarrow 0$
4.          else
5.              $maxProd[m] \leftarrow -\infty$
6.              for $i = 1$ to $m$
7.                  $candidate[i] \leftarrow \max\{i \times (m-i), i \times eggDrop[m-i]\}$
8.                  if $candidate[i] > maxProd[m]$
9.                      $maxProd[m] \leftarrow candidate[i]$
10.                     $flag[m] \leftarrow i$
11. return $maxProd$
```

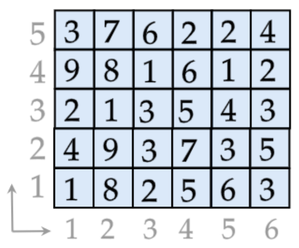
Step 4: Retrieve the actual solution, i.e., the length of the cuts:

```
RetrieveCuts($n, maxProd$)
1.  $m \leftarrow n$
2.  while $m \geq 2$
3.      print $flag[m]$
4.      $m \leftarrow m - flag[m]$
```

3. Shahin's Board Game

Consider the following game between two players. We are given an $m \times n$ square formed by mn tiles. Each tile has a positive integer value written on it. At their turn, a player selects either the top row, bottom row, left column, or right column, and collects (removes) all the tiles of their selection. For example, in the example below, where $m = 6$ and $n = 5$, a player may collect the top row for a total value of $3+7+6+2+2+4 = 24$, the bottom row for a total value of $1+8+2+5+6+3 = 25$, the left row for a total value of $1+4+2+9+3 = 19$, or the right row for a total value $4+2+3+5+3 = 17$. Of course, the players tend to play in a way to maximize the total value of their collected tiles. The game ends when there is no more tile to collect

Devise a DP algorithm that reports the maximum profit (the total value of the collected tiles in the entire game) of the player that plays first, assuming both players play optimally to maximize their profit. You may use $S[l,b,r,t]$ to denote the total sum of tiles within the rectangle that finds its left at index l, its bottom at index b, its right at index r, and its top at index t (we have $1 \leq l \leq r \leq m$ and $1 \leq b \leq t \leq n$). For example, $S[1,2,3,4] = 4+9+3+2+1+3+9+8+1$.



Answer: Let $Profit[l,b,r,t]$ indicates the maximum profit of the player whose turn is to play, assuming the set of remaining tiles is formed by the rectangles findings its left at index l, its bottom at index b, its right at index r, and its top at index t (we have $1 \leq l \leq r \leq m$ and $1 \leq b \leq t \leq n$).

- If the first player takes the left column, it collects $S[l,b,l,t]$, and the profit of the second player for the remaining tiles will be $Profit[l+1,b,r,t]$ and therefore, the profit of the first player will be the rest, that is, $S[l+1,b,r,t] - Profit[l+1,b,r,t]$. Therefore, the total profit of the first player for choosing the left column is $S[l,b,l,t] + S[l+1,b,r,t] - Profit[l+1,b,r,t]$.
- With a similar argument, the total profit of the first player for choosing the bottom row is $S[l,b,r,b] + S[l,b+1,r,t] - Profit[l,b+1,r,t]$.
- Similarly, the total profit of the first player for choosing the right column is $S[r,b,r,t] + S[l,b,r-1,t] - Profit[l,b,r-1,t]$.
- With a similar argument, the total profit of the first player for choosing the top row is $S[l,t,r,t] + S[l,b,r,t-1] - Profit[l,b,r,t-1]$.

Note that if $r < l$ or $t < b$, then the set of remaining tiles is empty, and we have $Profit[l,b,r,t] = 0$. In summary, we can write:

$$Profit[l,b,r,t] = \begin{cases} 0 & \text{if } r < l \text{ or } t < b \\ \max\{S[l,b,l,t] + S[l+1,b,r,t] - Profit[l+1,b,r,t] & \\ \qquad S[l,b,r,b] + S[l,b+1,r,t] - Profit[l,b+1,r,t] & \\ \qquad S[r,b,r,t] + S[l,b,r-1,t] - Profit[l,b,r-1,t] & \text{otherwise} \\ \qquad S[l,t,r,t] + S[l,b,r,t-1] - Profit[l,b,r,t-1]\} & \end{cases}$$