Algorithm Theory

Chapter 4
Amortized Analysis

Part II:
Potential Function Method

Fabian Kuhn

Potential Function Method

- Most generic and elegant way to do amortized analysis!
  - But, also more abstract than the others…
- State of data structure / system: $S \in \mathcal{S}$ (state space)

  Potential function $\boldsymbol{\Phi}: \boldsymbol{\mathcal{S}} \to \mathbb{R}_{\geq 0}$

- **Operation $\boldsymbol{i}$:**
  - $\boldsymbol{t_i}$: actual cost of operation i
  - $\boldsymbol{S_i}$: state after execution of operation i (S_0: initial state)
  - $\boldsymbol{\Phi_i} \coloneqq \Phi(S_i)$: potential after exec. of operation i
  - $\boldsymbol{a_i}$: amortized cost of operation i:

$$\boldsymbol{a_i \coloneqq t_i + \Phi_i - \Phi_{i-1}}$$

Potential Function Method

$t_i = a_i + \Phi_{i-1} - \Phi_i$

Operation i:

actual cost: t_i amortized cost: $a_i = t_i + \Phi_i - \Phi_{i-1}$

Overall cost:

$$T := \sum_{i=1}^{n} t_i = \left(\sum_i^n a_i\right) + \Phi_0 - \Phi_n \leq \left(\sum_i^n a_i\right) + \Phi_0$$

$$\begin{aligned}\sum_{i=1}^{n} t_i = \; & a_1 + \Phi_0 - \Phi_1 \\ & + a_2 \quad + \Phi_1 - \Phi_2 \\ & + a_3 \quad + \Phi_2 - \Phi_3 \\ & + a_4 \quad + \Phi_3 \cdots \\ & \vdots \\ & + a_{n-1} \quad \cdots - \Phi_{n-1} \\ & + a_n \quad + \Phi_{n-1} - \Phi_n\end{aligned}$$

Binary Counter: Potential Method

- **Potential function:**

  Φ: number of ones in current counter

- Clearly, $\Phi_0 = 0$ and $\Phi_i \geq 0$ for all $i \geq 0$

- Actual cost t_i:
  - 1 flip from 0 to 1
  - $t_i - 1$ flips from 1 to 0

- Potential difference: $\Phi_i - \Phi_{i-1} = 1 - (t_i - 1) = 2 - t_i$

- Amortized cost: $a_i = t_i + \Phi_i - \Phi_{i-1} = 2$

Example 3: Dynamic Array

- How to create an array where the size dynamically adapts to the number of elements stored?
  - e.g., Java "ArrayList" or Python "list"

Implementation:

- Initialize with initial size N_0
- Assumptions: Array can only grow by appending new elements at the end
- If array is full, the size of the array is increased by a factor $\beta > 1$

Operations (array of size N):

- read / write: actual cost $O(1)$
- append: actual cost is $O(1)$ if array is not full, otherwise the append cost is $O(\beta \cdot N)$ (new array size)

Example 3: Dynamic Array

Notation:

- n: number of elements stored
- N: current size of array

Cost t_i of ith append operation: $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$

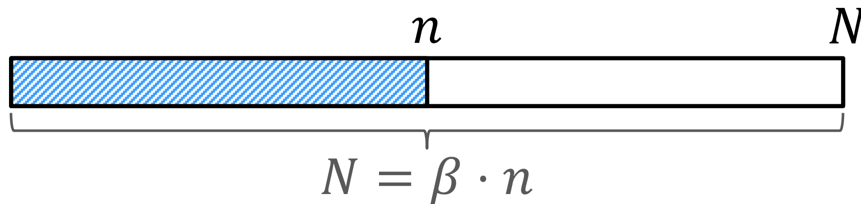
Claim: Amortized append cost is $O(1)$

Potential function Φ?

- should allow to pay expensive append operations by cheap ones
- when array is full, Φ has to be large
- immediately after increasing the size of the array, Φ should be small again

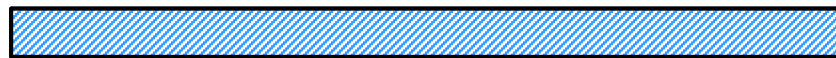
Dynamic Array: Potential Function

Cost t_i of ith append operation: $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$



Φ small $(\Phi = 0)$

$n = N$



Φ large $(\Phi \geq \beta N)$

At the start:

$N = N_0$

$n = 0$

We need: $\Phi \geq 0$

Let's try: $\Phi(n, N) = c \cdot (\beta n - N) \; + \; c \cdot N_0$

$$c(\beta N - N) \geq \beta N$$

$$c(\beta - 1) \geq \beta$$

$$c \geq \frac{\beta}{\beta - 1}$$

$$\Phi(n, N) = \frac{\beta}{\beta - 1} \cdot (\beta n - N + N_0)$$

Dynamic Array: Amortized Cost

Cost t_i of ith append operation: $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$

Potential function: $\Phi(n, N) = \frac{\beta}{\beta - 1} \cdot (\beta n - N + N_0)$

Amortized cost $a_i = t_i + \Phi_i - \Phi_{i-1}$

Case 1 ($n < N$): $a_i = 1 + \frac{\beta}{\beta-1} \cdot (\beta(n+1) - \beta n) = 1 + \frac{\beta^2}{\beta-1}$

Case 2 ($n = N$): $t_i = \beta n = \beta N$

$$a_i = \beta N + \frac{\beta}{\beta-1} \cdot [\beta(N+1) - \beta N - (\beta N - N)]$$

$$= \beta N + \frac{\beta^2}{\beta-1} - \frac{\beta}{\beta-1} \cdot (\beta - 1)N = \frac{\beta^2}{\beta-1}$$

Amortized cost $\leq 1 + \frac{\beta^2}{\beta-1}$