CS4311
Design and Analysis of Algorithms

Lecture 15: Amortized Analysis II

About this lecture

- Previous lecture shows Aggregate Method
- This lecture shows two more methods:

(2) Accounting Method

(3) Potential Method

Accounting Method

- In real life, a bank account allows us to save our excess money, and the money can be used later when needed

- In amortized analysis, the accounting method is very similar …

Accounting Method

… each operation pays an amortized cost

- If amortized cost $\geq$ actual cost, we save the excess in the bank
- Else, we try to use our savings to pay

Equivalently, we can always save the amortized cost to the bank first, and ask the bank to pay for the operation

Lemma: For a sequence of operations, if we have enough to pay for each operation,

total actual cost $\leq$ total amortized cost

Super Stack (Take 2)

- Recall that apart from PUSH/POP, a super stack, supports:

  SUPER-POP(k): pop top k items in $O(k)$ time

- Let us now assign the amortized cost for each operation as follows:

  PUSH = \$2

  POP or SUPER-POP = \$1

Super Stack (Take 2)

Questions:

- Which operation "saves money to the bank" when performed?

  Ans. PUSH. Each saves an excess $1 to the bank when performed

- Which operation "needs money from the bank" when performed?

  Ans. SUPER-POP. Its amortized cost is 1, while its actual cost can be larger

Super Stack (Take 2)

Questions:

- Are there enough to pay for each SUPER-POP operation?

  Ans. When an item x is pushed, we may "link" the excess \$1 with item x.

  When SUPER-POP is performed, each popped item donates its corresponding \$1 to help in paying the operation

  ➔ Enough \$\$ to pay for each SUPER-POP

Super Stack (Take 2)

Conclusion:

- Amortized cost of PUSH = 2
- Amortized cost of POP/SUPER-POP = 1

Meaning:

For any sequence of operations with

#PUSH = n_1, #POP = n_2, #SUPER-POP = n_3,

total actual cost $\leq 2n_1 + n_2 + n_3$

Binary Counter (Take 2)

- Let us use accounting method to analyze increment operation in a binary counter, whose initial count = 0

0	0	0	0	0

- We shall assign an amortized cost of $2 for each increment
- Recall that:

actual cost of increment = #bits flipped

Binary Counter (Take 2)

Observation: In each increment operation, at most one bit is set from 0 to 1 (whereas the remaining bits are set from 1 to 0).

E.g.,

0	0	1	0	0

count = 4

$\Rightarrow$

0	0	1	0	1

count = 5

0	0	1	0	1

count = 5

$\Rightarrow$

0	0	1	1	0

count = 6

Binary Counter (Take 2)

- To show amortized cost = \$2 is enough, in each increment operation:
  - we use \$1 to pay for flipping some bit x from 0 to 1, and store the excess \$1 (as usual, we link this excess \$1 with bit x)
  - For other bits being flipped (from 1 to 0), each donates its corresponding \$1 to help in paying the operation

➔ Enough to pay for each increment

Binary Counter (Take 2)

Conclusion:

- Amortized cost of increment = 2

Meaning:

For n increments (with initial count = 0)

total actual cost $\leq 2n$

Question: What's wrong if initial count $\neq 0$?

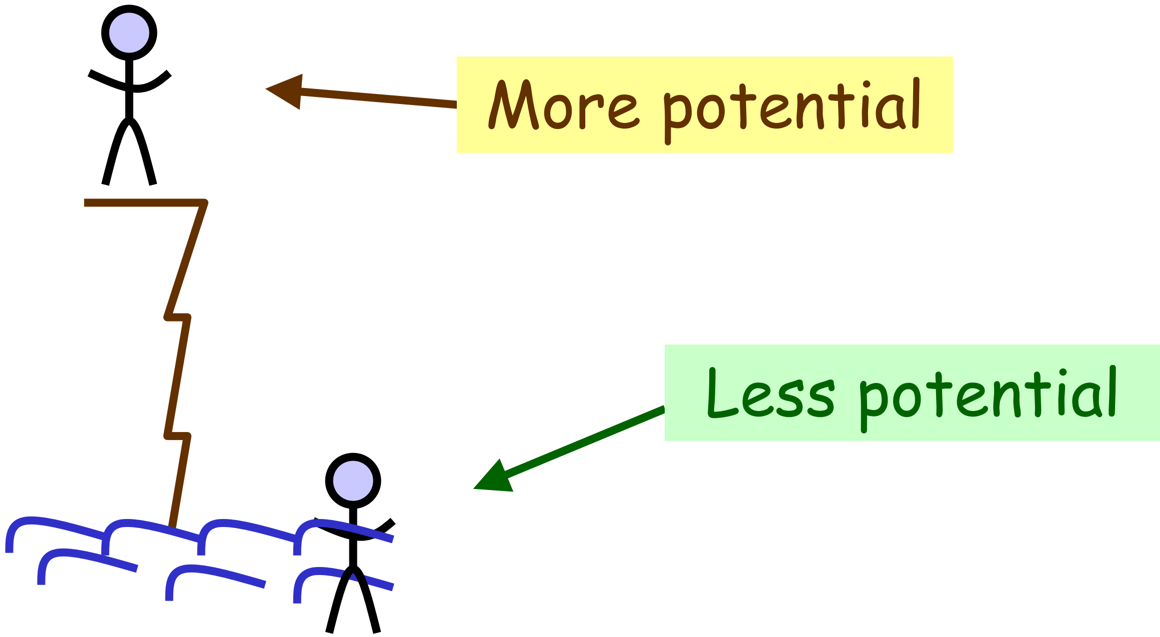
Accounting Method (Remarks)

- In contrast to the aggregate method, the accounting method may assign different amortized costs to different operations
- Another thing: To help the analysis, we usually link each excess $ to a specific object in the data structure (such as an item in a stack, or a bit in a binary counter)

  → called the credit stored in the object

Potential Method

- In physics, an object at a higher place has more potential energy (due to gravity) than an object at a lower place



Potential Method

- The potential energy can usually be measured by some function of the status of the object (in fact, its height)
- In amortized analysis, the potential method is very similar …
  - It uses a potential function to measure the potential of a data structure, based on its current status

Potential Method

- Thus, potential of a data structure may increase or decrease after an operation
- The potential is similar to the $ in the accounting method, which can be used to help in paying an operation
- While accounting method can use any amount of money in the bank (as long as the bank has money), in potential method, we have a more specific concern…

Potential Method

Precisely, each operation pays an amortized cost, and

- If potential increases by d after an operation, we need:

  amortized cost $\geq$ actual cost + d

- Else, if potential decreases by d after an operation, we need:

  amortized cost + d $\geq$ actual cost

Potential Method

In other words, let

- Φ = potential function
- D_i = data structure after i^{th} operation
- c_i = actual cost of i^{th} operation
- α_i = amortized cost of i^{th} operation

Then, we always need:

$$\alpha_i \geq c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Potential Method

- Because smaller amortized cost gives better (tighter) analysis, so in general, we set:

$$\alpha_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

- Consequently, after n operations, total amortized cost
  $= \text{total actual cost} + \Phi(D_n) - \Phi(D_0)$

Potential Method

- Thus, if we can find Φ such that

$$\Phi(D_n) \geq \Phi(D_0)$$

then, we obtain the desired relationship:

total amortized cost $\geq$ total actual cost

so that amortized cost α_i works correctly

- In fact, we want the relationship holds for any number of operations, so we want

$$\Phi(D_i) \geq \Phi(D_0) \quad \text{for all } i$$

Super Stack (Take 3)

- Let us now use potential method to analyze the operations on a super stack
- Define Φ such that for a super stack S

$$\Phi(S) = \#\text{items in } S$$

- So, if we denote D_i to be the super stack after i^{th} operation, we have:

$$\Phi(D_0) = 0, \text{ so that } \Phi(D_i) \geq \Phi(D_0) \text{ for all } i$$

Super Stack (Take 3)

- PUSH increases potential by 1

  ➔ amortized cost of PUSH = 1 + 1 = 2
- POP decreases potential by 1

  ➔ amortized cost of POP = 1 + (-1) = 0
- SUPER-POP(k) decreases potential by k

  ➔ amortized cost of SUPER-POP

  = k + (-k) = 0

[Assume: Stack has enough items before POP/SUPER-POP]

Super Stack (Take 3)

Conclusion:

Because

$\Phi(D_0) = 0$, and $\Phi(D_i) \geq \Phi(D_0)$ for all i,

➔ total amortized cost $\geq$ total actual cost

Then, by setting amortized cost for each operation accordingly (according to what??):

amortized cost = O(1)

Binary Counter (Take 3)

- Let us now use potential method to analyze the increment in a binary counter
- Define Φ such that for a binary counter B

  $$\Phi(B) = \text{\#bits in B which are 1}$$

- So, if we denote D_i to be the binary counter after i^{th} operation, we have:

  $$\Phi(D_0) = 0, \text{ so that } \Phi(D_i) \geq \Phi(D_0) \text{ for all } i$$

Assume: initial count = 0

Binary Counter (Take 3)

- From our previous observation, at most 1 bit is set from 0 to 1, the corresponding increase in potential is at most 1
- Now, suppose the i^{th} operation resets t_i bits from 1 to 0

➔ actual cost $c_i \leq t_i + 1$

➔ potential change $\leq (-t_i) + 1$

➔ amortized cost α_i

$= c_i +$ potential change ≤ 2

Binary Counter (Take 3)

Conclusion:

Because

$\Phi(D_0) = 0$, and $\Phi(D_i) \geq \Phi(D_0)$ for all i,

→ total amortized cost ≥ total actual cost

Then, by setting amortized cost for each operation accordingly:

amortized cost ≤ 2 = $O(1)$

Potential Method (Remarks)

- Potential method is very similar to the accounting method: we can save something ($/potential) now, which can be used later
- It usually gives a neat analysis, as the cost of each operation is very specific
- However, finding a good potential function can be extremely difficult (like magic)
  - Analyzing Union-Find data structure