Chapter 16: 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
- We also have an easy way to check the savings
- 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 use savings to help the payment
- Often, savings can be checked easily based on the objects in the current data structure
- 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 k time

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

  PUSH = $2

  POP or SUPER-POP = $0

Super Stack (Take 2)

- Questions:
- Which operation “saves money to the bank“ when performed?
- Which operation “needs money from the bank“ when performed?
- How to check the savings in the bank ?

Super Stack (Take 2)

- Does our bank have enough to pay for each SUPER-POP operation?

- Ans. When SUPER-POP is performed, each pushed item donates its corresponding $1 to help the payment

  ➔ Enough $$ to pay for each SUPER-POP

Super Stack (Take 2)

- Conclusion:
  - Amortized cost of PUSH = 2
  - Amortized cost of POP/SUPER-POP = 0

- Meaning:
  - ✓ For any sequence of operations with
  - ✓ #PUSH = n_1, #POP = n_2, #SUPER-POP = n_3,

$$\text{total actual cost} \leq 2n_1$$

➔ amortized cost = $O(1)$ per operation

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 assign amortized cost for each increment = \$2

- Recall: actual cost = #bits flipped

Binary Counter (Take 2)

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

E.g.,

0	0	1	0	0

→

0	0	1	0	1

count = 4 → count = 5

0	0	1	1	1

→

0	1	0	0	0

count = 7 → count = 8

Fig. 16-2

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Binary Counter (Take 2)

- Observation: Savings = # of 1's in the counter
- To show amortized cost = \$2 is enough,
  - ✓ we use \$1 to pay for flipping some bit x from 0 to 1, and store the excess \$1
  - ✓ 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

Proof

- Basis Counter = 0 : increment counter = 1 , savings = 1 hold
- Assume counter has k contiguous ones from 1st bit (rightmost) to the kth bit = 01…1 after 2^k -1 increment and we save k credit
- We add one to the counter it should be 10…0 we saving a credit in the leftmost bit. Therefore, after 2^k -1 increment we will save another k credit and obtain k+1 credit for 1…1 (k+1 ones)

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 $\neq 0$?

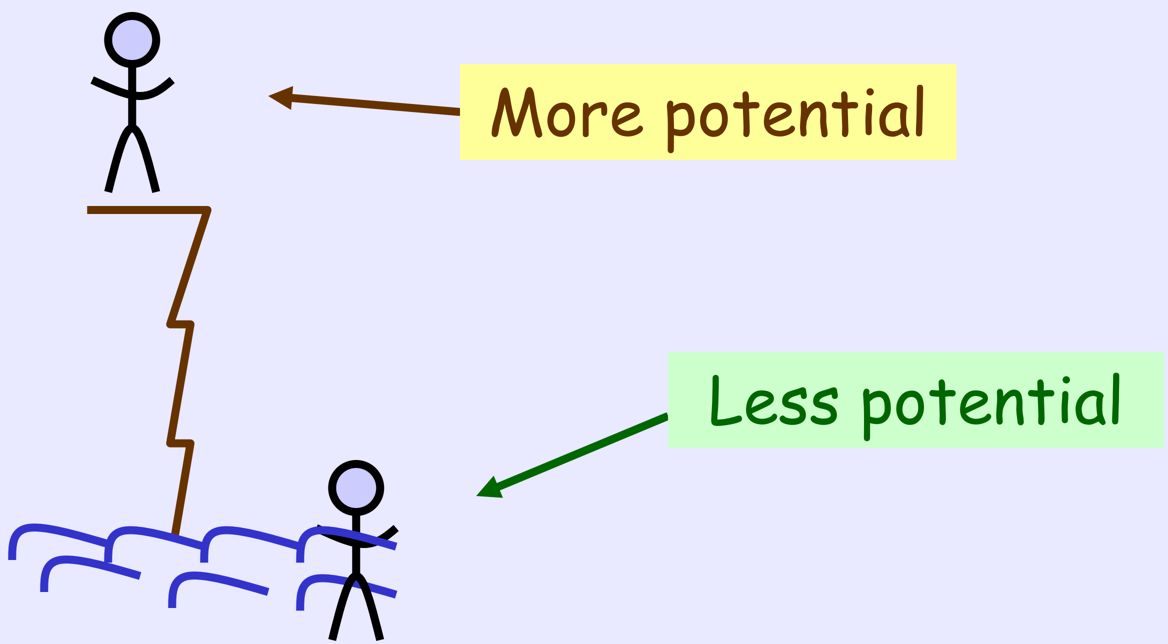
Accounting Method (Remarks)

- 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

Potential Method

- Each operation pays an amortized cost, and
  - ✓ If potential increases by d after an operation, we need:

    amortized cost $\geq$ actual cost + d

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

    amortized cost $\geq$ actual cost - d

Potential Method

- To combine the above, we 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:

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

(Potential Change)

- Consequently, after n operations,

  total amortized cost

  = total actual cost + $\Phi(D_n) - \Phi(D_0)$

Potential Method

- Any Φ such that

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

  should work, as it implies

  total amortized cost at any time

  $\geq$ total actual cost at any time

- Our target is to find the best such Φ so that amortized cost can be minimized

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

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

- Thus we have:

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

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 according to potential function: amortized cost = 2= 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

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

- Thus we have:

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

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 $c_i = t_i + 1$

  → potential change $= (-t_i) + 1$

  → amortized cost α_i

  $= c_i +$ potential change $= 2$, for all i

Binary Counter (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:

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)

Homework

- Exercises: 16.2-1, 16.2-3, 16.3-1, 16.3-3, 16.3-4, 16.3-5, 16.3-6