# Lab04-Amortized &Graph 1

CS2308-Algorithm Design and Analysis, Xiaofeng Gao, Spring 2023.

∗ Contact TA Jiale Zhang for any questions. Include both your .pdf and .tex files in the uploaded .rar or .zip file.

∗ Name:张祥东 Time Spent on this Lab:____

1. (Amortized Analysis) **Amy's Album:** Amy wishes to have an electronic photo album to store photographs of herself. When Amy takes a photo, she put it into the album. When Amy shares her life with friends, she takes out the best one and throw away all photos which are taken later than this one. To define "the best", she gives each photo a score. She also requires the album to be as fast as possible.

   As the software developer for Amy's album, the essential problem you faced here is to develop a data structure $S$ to satisfy the following two operations, with average time complexity O(1) for each operation:

   - Operation 1: $Push(a)$, the user put a photo with a score into the album. For simplicity, here $a$ is a number, denoting the score.
   - Operation 2: $Pop\_best()$, output the largest number $a^*$ in $S$ and remove all number that pushed later than $a^*$ in $S$ (Including $a^*$ it self).

   Please give a design of $S$ with two stacks and prove by Amortized Analysis that the $S$ can satisfy the two operations above with average time complexity O(1) for each $Push(a)$ or $Pop\_best()$.

   **Solution.** Here comes the structure for S with two stacks.

**Algorithm 1:** struct S with two stacks

```
1:  struct S
2:  variables
3:      stackOne which stores integer
4:      stackTwo which stores integer
5:
6:  initialization function S()
7:      stackOne ← empty;
8:      stackTwo ← empty;
9:
10: function Push(a)
11:     stackOne.push(a);
12:     if stackTwo.empty() or a ≥ stackTwo.topElement:
13:         statckTwo.push(a)
14:
15: function Pop_best()
16:     if stackOne.empty():
17:         return ;
18:     value ← stackTwo.topElement;
19:     stackTwo.pop();
20:     while stackOne.topElement ≠ value:
21:         stackOne.pop();
22:     stackOne.pop();
23:     print value
24:     return ;
```

In the **struct S**,the stackOne stores all photos that Amy takes and the stackTwo only record a sequence of photos with non-decreasing score sequence.So when we need do the Pop_best(),the best element must be the top element in the stackTwo. Thus,by simply using the top element of stackTwo,we can pop the elements in stackOne until we meet the same element as the top element of stackTwo.

Assume there've been n times operations before including $n_1$ Push() and $n_2$ Pop_best().Namely

$$n = n_1 + n_2$$

Let $T(n)$ denotes the time needed for n operations and $C_i$ denotes the cost of $i_{th}$ operation while $\hat{C}_i$ denotes the amortized cost of $i_{th}$ operation.Here comes:

$$T(n) = \sum_{i=1}^{n} C_i \leq \sum_{i=1}^{n} \hat{C}_i$$

$$T(n) = \#Push + cost\ of\ pop\ best$$

$$cost\ of\ pop\ best = \#Pop$$

Obviously

$$\#Pop \leq \#Push$$

$$T(n) = \#Pop + \#Push \leq 2\#Push = 2n_1 \leq 2n$$

Then the amortized cost of each operaion $\frac{1}{n}\sum_{i=1}^{n} \hat{C}_i = \frac{2n}{n} = O(1)$.

Above all, the S can satisfy the two operations with average time complexity O(1) for each operation. □

2. (DFS) Draw the DFS tree of Graph 1, starting from $s$.(By lexicographical order if there are multiple vertices to be visited next). How many vertices are there in the stack at most (According to the discussion in class)?

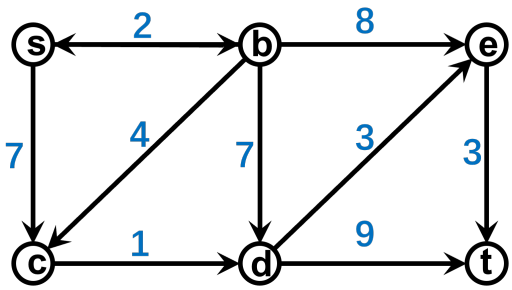


图 1: Graph 1

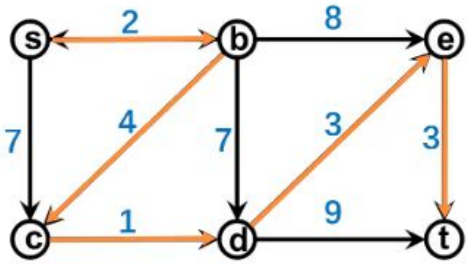


图 2: Graph 2

图 3: Two graphs

**Solution.** The DFS tree is shown in the Graph 2.

There are 5 vertices in the stack at most. □

3. (BFS) Write the BFS visiting order of Graph 1, starting from $s$.(By lexicographical order if there are multiple vertices to be visited next)

**Solution.** The BFS visiting order:

**s b c d e t**

□

4. (Dijkstra) Write the distance (the length of the shortest path) from $s$ to every vertex in Graph 1 in the order that it is extracted-min from the priority queue by Dijkstra algorithm.Compare 3 and 4, discuss the relationship between BFS and dijkstra algorithm.

**Solution.**

| Vertex | s | b | c | d | e | t |
|---|---|---|---|---|---|---|
| Distance | 0 | 2 | 6 | 7 | 10 | 13 |

For BFS,we use a queue and in every operation,we pop an element and search the adjacent elements of it and then push them all to the queue.We repeat this process until the queue is empty,namely all elements connected being visited.

For dijkstra algorithm,we use a priority queue to solve the shortest path problem.In every operation,we do extract-min to get the node whose shortest distance is fixed and then we visit its adjacent elements to update the value in the priority queue.We stop when all the distances are fixed,namely the moment the priority queue is empty.

Similarity between them:

- They are both graph algorithm.
- They both can be used as shortest path search algorithm.BFS can search shortest path when the weight of the edge is 1,and dijkstra algorithm can be used when the weight of edges is non-negative.BFS can be seen as an specific instance of dijkstra algorithm from this perspective.
- They both use a method of adjacent spansion.Both BFS and dijkstra algorithm use the adjacent nodes' information of the current node to expand their search field by using the adjacent information step by step.
- They both use a queue to maintain their elements to be operated.BFS use queue to maintain the upcoming sequence that needed to be coped and dijkstra algorithm use a priority queue to find the element whose distance is minimum is the queue.

Difference between them:

- BFS choose element by the added order of elements entirely and the "current" adjacent information is enough for its use.It's often used as a graph traversal algorithm.
- dijkstra algorithm choose element by the value of added elements and it uses the all elements' adjacent information to support its operation which is different from BFS.It's always used as an efficient algorithm to search shortest path in a non-negative weighted graph.

□