ADA hw3

B08902065 資工二 洪易

Discuss with

B08902127 林歆凱 B08902047 汪昊新 B08902013 張永達 B08202029 楊欣翰 B08902075 林耘平 B08902072 黃國銘 B08902063 陳羿穎 B08902020 陳明信

Ref: https://logic.pdmi.ras.ru/~elena/alon.pdf

5. Amortized Analysis

(1)

The recursive function acts as a monotonic stack which is always increasing. For every new index i, the stack will pop all the numbers that is smaller than a[i], choose the index of the top number left in the stack as dp[i] and push (i, a[i]) into the stack. Take the following charts for example, the current i = 4, a[i] = 7 and we want to find dp[i]. Operations the recursive function will do is call H(3, 7), H(1, 7) and find 8 > a[4] = 7, make dp[4] = 1. On the other hand, the stack operation will be popped out a[3] = 4 (equivalent to call H(3, 7)), and find a[1] = 8 > a[4] = 7 (equivalent to call H(1, 8)). As a result, push (4, 7) into the stack.

The current dp table.

i	a[i]	dp[i]
4	7	?
3	4	1
2	3	1
1	8	0
0	12	0

The current equivalent stack

Index	Number
3	4
1	8
0	12

Therefore, we can transform the process of finding the time complexity of recursive function into finding the time complexity of a stack with operation push and pop as two algorithms are equivalent. By using the method demonstrated in class, let the potential function $\Phi(D_i)$ equals to the number of elements in equivalent stack after i operations. For the validity check, it's trivial that for all $i > 0$, $\Phi(D_i) >= \Phi(D_0)$. For push operation, the amortized cost is 2, and for pop operation, the amortized cost is 0 (As shown in class). All operations have O(1) amortized cost, so total cost of n operations is O(n).

(2)

I maintain a deque of classes to keep track of all the people in line, and an array ans with size of total types to store the answer. Each class represents a group, and I store the t_i, population of the group, and the time the group is pushed into the deque in the class. For each new group i added into the line, I pop the first or last element from the deque and compare the population in it with k_i. Not until k_i is smaller than the population of the most recently popped item, pop the front or back (determined by s_i) and subtract k_i by the population in it. In addition, add ans[Type of the class] with (current time – the time of the class) * population of the class. While if k_i is smaller than the population of the most recently popped item, push back the left people and current time and type of the class. At last, push a class of t_i, $k_i + c_i$, and i into the deque and proceed to the next group. While finish all group, pop out all the group and calculate the ans[] with the same method above. In the ans[] array, the correct answer will be stored properly.

Time complexity analysis

Operation	Actual Cost	Amortized Cost
push_back() and push_front()	1	3
pop_back() and pop_front()	1	0
Population modification	1	0

Modify population means the operation when k_i is smaller than the population of the most recently popped item, modify the population in the last popped item and push back to the deque. For each group push into the deque, many pops may be done but at most one population modification will be done. From another perspective, each group the deque will be popped after pushed, as a result, the vault of every group will always be sufficient. Therefore, the amortized cost of each operation is O(1), after n group, the total time complexity of computing is O(Q). In addition, the time complexity of printing out the answer of each type is O(N). To sum up, the total time complexity is O(Q + N).

(3)

We have 2 parts of time complexity need to analysis, rebalance and find insertion point.

1. Rebalance amortized cost

We consider how many numbers need to be insert into a prefect balanced tree. Without lost of generality, let the total number of nodes equal to x, and size(root.left) = x / 2. Suppose after b times of insertion, the tree is unbalanced, and consider the extreme case, which all the nodes are insert to root.left. Therefore, we can have inequality $\mathrm{a} * (\mathrm{x} + \mathrm{b}) - \frac{x}{2} \leq b$. We can see that $\mathrm{b} \geq \frac{(a - 0.5)x}{1 - a}$. Calculate the total time cost for n = $\left((1 - \frac{a - 0.5}{1 - a})^k \mathrm{x}\right)$. Take Log at each side and we can find out as 0.5 < a < 1, and x = 1 initially, k = O(log(n)). By problem description, rebalance take O(n) time. Hence, the amortized cost of insertion is O(n) * O(log(n)) / n = O(log(n)).

2. Find insertion point time cost.

We go through each node from root, and compare left and right of each node which have a larger size(node). Check if it's larger than a * size(node), if not and go that side. In worst case, we need to go to the leaf which is log(n) depth to make sure if it needed rebuild. Hence, the time to find insertion cost is O(log(n)).

Therefore, the total amortized time cost of insertion is O(log(n)) + O(log(n)) = O(log(n)).

(4)

Correction of Algorithm

As “and” operation only change from 1 to 0, keep doing more “and” only make 64 bits into more 0s but never changing any bit into 1. Therefore, we can state that B[j] <= B[i] for any j < i, and those bits in B[i] are 0 must be 0 in B[j], and those biys in B[j] are 1 must also be 1 is B[i].

For every new i, B[j] = B[j] & A[i], this is quite trivial, in this case the answer is correct. If we break, that means all the bits that are 1 in B[j] are also 1 in A[i]. Hence, all other k < j, B[k] must also hold the same value as their 1s must be less than those in B[j] and won’t be change to 0. Therefore, this algorithm is correct.

Time Complexity

Accounting method.

	Actual cost	Amortized cost
B[i] = A[i], in first loop	1	200
1 bit to 0 bit	1	0
Check if A[i] & B[j] == B[j]	1	0

For each bit, we can only change it from 0 to 1 first and then change it from 1 to 0. In this algorithm, bits only change from 1 to 0 once for every i and then don’t change at all, and there are 64 bits. Moreover, we only check if A[i] & B[j] == B[j] at most 64 times either, as if they are the same, the loop break, and & only make 1 to 0. Therefore, the money stored in the bank is always sufficient.

Accordingly, the time complexity has O(1) amortized time for every new i, so the time complexity is O(n).

6. Robert the Archaeologist

(1)

Proof by induction on E.

Base case, for E = 1:

V = 2 and F = 1, V - E + F = 2, true.

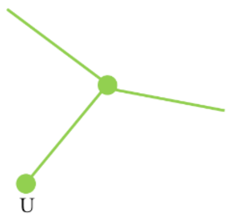
Induction hypothesis, for E = m:

Assume V - E + F = 2.

Induction step, Consider E = m + 1:

Case 1:

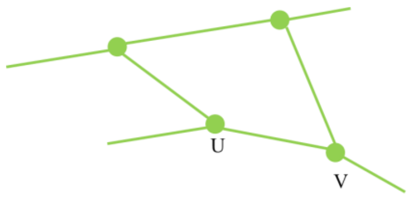
G contains a vertex u with degree 1.



remove u, then (V - 1) - (E - 1) + F = 2, then V - E + F = 2.

Case 2:

G contains no vertex of degree 1.



remove (u, v), then V - (E - 1) + (F – 1) = 2, then V - E + F = 2.

QED

Suppose $E <= 3 * V$ for all $E >= 1$.

Case 1, $F = 1$:
$E = V - 1$ (G is a tree)
$E >= 1$, then $V >= 2$.
Then, $3 * V - E = 3 * V - (V - 1) = 2 * V + 1 >= 1$.
We can see that $3 * V >= E$.

Case 2, $F > 1$:
It's trivial that every region is bounded by at least 3 edges and every edge is shared by at most 2 regions. Therefore, every face consumes at least 1.5 edges.
$F <= 2 * E / 3$.
Then by $V – E + F = 2$, $V – E + 2 * E / 3 >= 2$.
Then $V – E / 3 >= 2$, then $3 * V >= E$.

By case 1 and 2, $3 * V >= E$, $E = O(V)$.
QED

(2)

Using dijkstra algorithm with adjacency list and min-heap. While in the ordinary start condition, we have a start point with distance 0 in the heap, and set the distance of all other points into INT_MAX and also push them into the heap, we now set all the vertexes' distance that are secured to 0 and others to INT_MAX. Next, we start the Dijkstra computation. After smart computing of the algorithm, we can see the minimum distance to a secured vertex of every vertex is found.

Proof of Correctness:

Same as the proof of Dijkstra. As we may have many sources this time, so we can set their distance to 0 initially.

Time complexity:

Time complexity of Dijkstra with adjacency list and min-heap is $O((E + V) * \log(V))$. As $E = O(V)$, $O((E + V) * \log(V)) = O(2V * \log(V)) = O(V * \log(V))$

QED

(3) Claim 1
Proof by contradiction.

Suppose there exist a path inside the Saiko Psycho Cycle, that the number of vertices strictly inside the cycle is greater than 2/3V, that is shorter than the shortest path on the cycle for any pair(u, v) on the cycle. Without lost of generality, we can assume the shorter path inside the graph is I and the shortest path on the cycle is O, which length(I) < length(O). However, we can replace O with I for a new Saiko Psycho Cycle that also fulfill the first two conditions, and the number of vertices strictly inside the cycle minus the number of vertices is even smaller.

1. The number of vertices strictly outside the cycle is at most 2/3V.

The number of vertexes initially is greater than 2/3V. By replacing the shortest path on the cycle(O) with I, the number of vertexes left in the new cycle must be greater than 1/2 the number initially. As a result, the number of vertexes left in the new cycle must be strictly greater than 1/2 * 2/3V = 1/3V. Therefore, the number of vertexes outside must be strictly less than 2/3V.

2. The cycle comprises at most 4k vertices, where k is abs(sqrt(V + 1)).

Trivial, the number of vertices on the cycle decrease.

Accordingly, there exist another cycle fulfill the first two conditions of Saiko Psycho Cycle and the number of vertices strictly inside the cycle minus the number of vertices is even smaller. The origin cycle can’t be the Siako Psycho cycle.
Contradiction. The claim 1 is true.

QED

(4) Claim 2
Proof by contradiction.

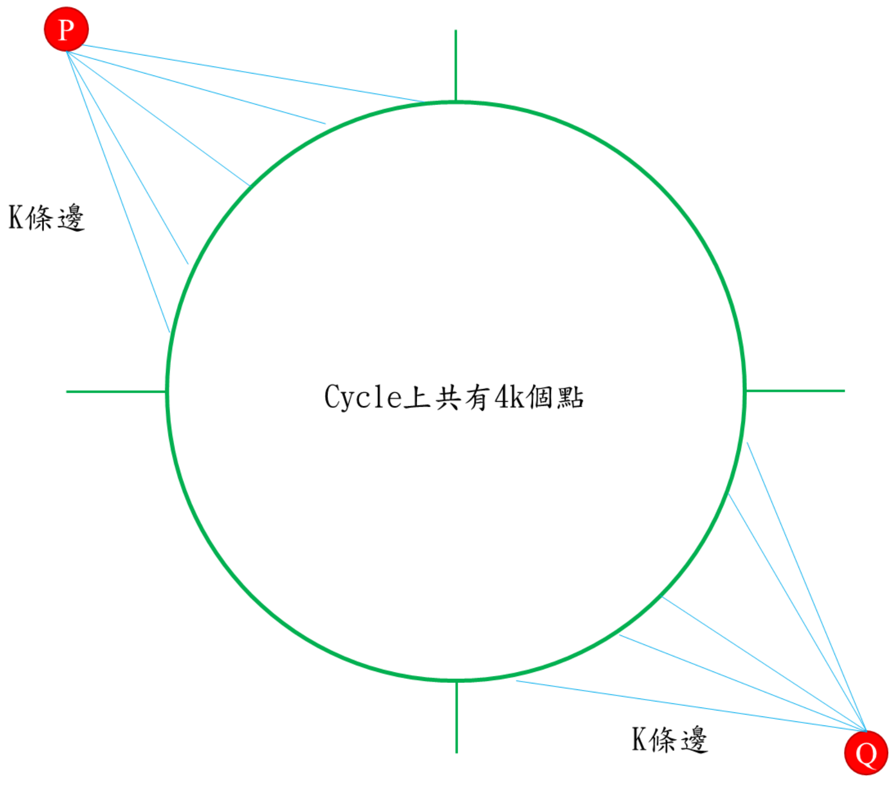
Suppose vertices on the cycle is smaller than 4k and fulfill all other requirement of Saiko Psycho Cycle, we must can find a path P that length is 2, and connect only u, v, which is both on the cycle, and a vertex that is not on the cycle. Let the path of u, v on the cycle is Q, and the rest of the cycle is Q'. In this case, we can replace Q with P and make union(P, Q') as the new cycle. We can see that the new cycle is also fulfill their first two conditions of Saiko Psycho Cycle. Moreover, it has a smaller number of vertices inside the cycle minus number of vertices outside cycle. Therefore, the initial cycle isn't a Saiko Psycho Cycle.
Contradiction. The claim 2 is true.
QED.

(5)

Proof by contradiction.

Assume there exist a Saiko Psycho Cycle with 2/3V or more of vertices inside it. From Claim 2, we see that their must exist 4k points on the cycle. As the example shown below, we can split the cycle into 4 quadrants with k vertices on each quadrant. Next, we add two vertices P, Q that are in the opposite quadrants and outside the cycle, which both connect and only connect to the k points on the cycle of their quadrant.

Let's suppose we want to find a path from P to Q. By Menger's theorem, their must exist k distinct paths as there are both unique k points connect to P and Q. By Jordan curve theorem, those paths must path through the cycle (go inside and ouside). Next, by Claim 1, length of every path inside the cycle connect two vertices on the cycle won't be shorter than the length of path on the cycle. As the distance of these points on the cycle are at least k + 1 (there are at least a quadrant, which are k vertices, between them), these k distinct paths must have a length not smaller than k + 1. Therefore, there must exist at least k points inside the cycle for each path, and all of them are different. Accordingly, k * k = V + 1 points must exist inside the cycle. However, the number of total vertices are V, contradiction.



We can't have a Saiko Psycho Cycle with 2/3V or more of vertices inside it. =
QED.