ADA Homework 4

許崴棠

December 23, 2023

Problem 0

- **Problem 1:**
- **Problem 2:**
- **Problem 3:**

Problem 1

(a) To determine if a 2-CNF problem is in 2-SAT, we can regard the 2-CNF formula as a set of clause, take ϕ_1 as an example we can regard ϕ_1 as

$$\phi_1 = \{(a_1 \lor \neg a_2), (a_3 \lor a_1), (\neg a_1 \lor \neg a_1)\}$$

and by De Morgan law, we can rewrite ϕ_1 as

$$\phi_1' = \{\neg(\neg a_1 \land a_2), \neg(\neg a_3 \land \neg a_1), \neg(a_1 \land a_1)\}$$

Consider a directed graph G where $V(G) = \{a_1, a_2, \dots, a_n, \neg a_1, \neg a_2, \dots, \neg a_n\}$. And to make $\neg(x \land y)$ in ϕ_1' true, we can add the edge of $x \to \neg y$ and $y \to \neg x$ since that if x is right, $\neg y$ must be right, if y is right, $\neg x$ must be right. After then, we can find the SCCs by the kosaraju's algorithm, and we can check if a 2-CNF formula is solvable simply by checking if there exist a a_i where a_i and $\neg a_i$ are in the same SCC, since if x is able to reach y, it means that "if x is true, then y must be true ", and if a_i and $\neg a_i$ are in the same SCC it means that the statement a_i will be true if and only if statement $\neg a_i$ is true, which is obviously wrong. Thus we can know that there will be at least one set of a that can solve the answer, only if there is no a_i and $\neg a_i$ in the same scc.

(b) (b-1)

(b-2)

(c) (c-1)

(c-2)

Problem 2

(a) By observe, we can tell that after we put α items into the current box, we'll had to spend 2α time to make a new box, so when we're putting new items into the box, we can save 6 dollars to the bank per item, therefore, after we put α items into the box with the size 2α, we can directly spend the 4α dollars we've saved in the bank to make a new box with the size of 4α, and spend 2α dollars to move the 2α items in the original box. By doing this, we can say that the time complexity of putting n items into the box are $O(n)$.

(b) It is impossible to amortize the time complexity to $O(n)$ because that when $n = s+1$, we need to spend s^2 time to make a new box, and it would be impossible to amortize a s^2 step to s step to make the overall time complexity $O(n)$.

(c) By observe, we can tell that after we put α items into the current box, we'll had to spend α^2 time to make a new box. When the box is full when we put the i_{th} item into the box, we had to make sure the previous $i - \sqrt{i}$ items can afford the price of making new box and move the every items into the new box. So when we're moving n^2 items to a new box, we had to pay $(n^2)^2 + n^2$ dollars, and it should be pay by the $n + 1_{th}$ to ${n^2}_{th}$ items. By math, we can know that to put the if we assume every i_{th} item had to pay $\alpha i + \beta$ dollars to bank. Since we already knew the first box (size 2) can be enlarge to the second box if every one put $2n$ dollars into the bank.
We had to calculate the general case, which is

$$(n^2)^2 + n^2 \leq \alpha\frac{(n^2+n+1)\cdot(n^2-n)}{2} + \beta(n^2-n)\ \forall\ n \geq 2$$

where n^2 is the current box size.
By math, we can tell that the tightest α is 2 and the tightest β is 3. Therefore, if every i_{th} insert put $2i + 3$ dollars into the bank, we can safely make a n^2 box and move the previous n_{th} items into the new box only by using the $n^2 + n$ the previous $\sqrt{n} + 1_{th}$ to n_{th} items had saved.

(d) First, we assume the worst case, which happens when the manager come and check after every time Nathan put an item into the box.
In the cost section of the below table, it represent the (cost of insert + cost of move)

i	cost	i	cost	i	cost
1	1+0	6	1+1	11	1+1
2	1+1	7	1+1	12	1+1
3	1+3	8	1+1	13	1+1
4	1+1	9	1+9	14	1+1
5	1+5	10	1+1	15	1+1

By the table above, we can find out that the maximum averagetime complexity happens when we put $n = 2^k + 1$ item into the boxes, which can take at most $2n-1+2(2^k-1) = 4n-3$ times, and we can tell the time complexity is $O(4n-3) = O(n)$.

(e)