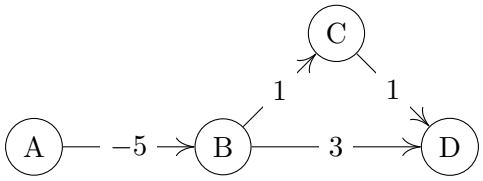
COMP20007 Design of Algorithms

Workshop 7 Solutions

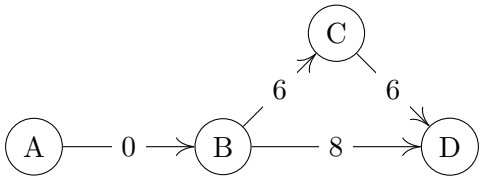
Tutorial

1. Negative edge weights Your friend's algorithm might sound like a good idea, but it sadly won't cure Dijkstra's algorithm of its inability to handle negative edge weights properly. Simply adding a constant value to the weight of each edge distorts the length of paths differently depending on how many edges they contain. Therefore the shortest paths found by Dijkstra's algorithm in the modified graph might not correspond to true shortest paths in the original graph.

As an example, consider the graph below.



The shortest path from A to D is A, B, C, D. However, when you add 5 to every edge, the shortest path becomes A, B, D:



(Interestingly, however, a similar idea forms the basis of a fast *all pairs, shortest path* algorithm called Johnson's algorithm. See `https://en.wikipedia.org/wiki/Johnson%27s_algorithm` for details, it's an interesting read!)

2. Master Theorem Note for this question that comparing a and b^d is the same as comparing $\log_b a$ and d.

(a) $T(n) = 9T\left(\frac{n}{3}\right) + n^3,\ T(1) = 1$

We have $a = 9, b = 3, d = 3$ and $c = 1$. So $\log_b(a) = \log_3(9) = 2$.

Also $2 < 3 = c$, so the $\Theta(n^3)$ is the dominating term. So $T(n) \in \Theta(n^3)$.

(b) $T(n) = 64T\left(\frac{n}{4}\right) + n + \log n,\ T(1) = 1$

We have $a = 64$ and $b = 4$, so $\log_b(a) = \log_4(64) = 3$.

Also, since $n + \log n \in \Theta(n^1)$, $d = 1 < 3$, so $T(n) \in \Theta(n^3)$.

(c) $T(n) = 2T\left(\frac{n}{2}\right) + n,\ T(1) = 1$

We have $a = b = 2$ so $\log_b(a) = \log_2(2) = 1$. Also $n \in \Theta(n^1)$ so $d = 1$ as well.

So $T(n) \in \Theta(n^d \log n) = \Theta(n \log n)$.

(d) $T(n) = 2T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2}\right) + 0,\ T(1) = 1$

Here $a = b = 2$ and $n^d = 0$ so $b^d = 0 < 2 = a$, thus we get $\Theta(n^{\log_2 2}) = \Theta(n)$.

3. Lower bound for the Closest Pairs problem We can use the closest pair algorithm from class to solve the element distinction problem like so, where the output is a collection of elements $C = \{c_1, \ldots, c_n\}$ and the output is DISTINCT or NOTDISTINCT:

```
function ElementDistinction(C = {c_1, ..., c_n})
    Points ← {(c_1, 0), ..., (c_n, 0)}
    Distance ← ClosestPair(Points)
    if Distnace is 0 then
        return NotDistinct
    else
        return Distinct
```

So, we can see that we can solve the element distinct problem using the closest pair algorithm. This is called a *reduction* from element distinction to closest pair.

We know that ELEMENTDISTINCTION is $\Omega(n \log n)$, and want to prove that CLOSESTPAIR is also $\Omega(n \log n)$.

We assume for the sake of contradiction that CLOSESTPAIR can be solved in a time complexity smaller (*i.e.*, asymptotically faster) that $n \log n$. As we have exhibited (provided) a reduction from ELEMENTDISTINCTION to CLOSESTPAIR then this must also give us an algorithm for ELEMENTDISTINCTION which is asymptotically faster than $n \log n$.

This contradicts the statement that ELEMENTDISTINCTION is $\Omega(n \log n)$, and as a result our assumption that CLOSESTPAIR can be solved in a time complexity smaller (*i.e.*, asymptotically faster) that $n \log n$ must be false.

Hence CLOSESTPAIR can not be solved in faster than $n \log n$ time, and is therefore $\Omega(n \log n)$.

4. Horspool's Algorithm For that pattern we calculate the shifts: $S[G] = 3, S[O] = 2, S[R] = 1, S[x] = 4$ for all other letters x. So the first shift (when the string's O fails to match E) is 2 positions, bringing E under the string's I. The next shift is 4, which will take us beyond the end of the string, so the algorithm halts (after just two comparisons), reporting failure.

5. Horspool's Algorithm Continued

(a) The pattern's last 1 will be compared against every single 0 in the text (except of course the first four), since the skip will be 1. So 999,996 comparisons.

(b) Here we will will make two comparisons between shifts, and each shift is of length 2. So the answer is again 999,996 comparisons.

(c) For the last pattern, the skip is 4. So we will make 249,999 comparisons.

6. Horspool's Worst-Case Time Complextity Consider the case where the text contains n zeros, and the pattern contains a 1 in the first position, followed by $m - 1$ zeros. In this case each skip will be just a single position, and between skips, m comparisons will be made.

Skips will occur until the pattern is starting at index $n - m$. Since we start at index 0 and end at index $n - m$ we do m comparisons $n - m + 1$ times, giving $m(n - m + 1) \in O(nm)$ comparisons.

We can see that if our pattern contains a 1 followed by $n/2$ zeros this time complexity becomes $O(n^2)$.