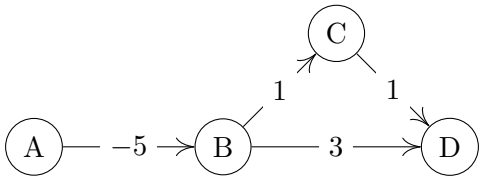
COMP20007 Design of Algorithms

Workshop 7 Solutions

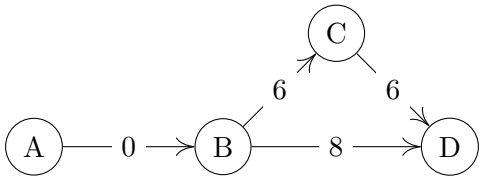
Tutorial

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

Here $a = b = 2$ and $d = 0$ so $b^d = 1 < 2 = a$, thus we get $\Theta(n^{\log_2 2}) = \Theta(n)$.

3. Mergesort Time Complexity Recurrence relation:

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

where $T(n)$ is the runtime of mergesort sorting n elements. The first $T(\frac{n}{2})$ is the time it takes to sort the left half of the input using mergesort. The other $T(\frac{n}{2})$ is the time it takes to sort the right half. $\Theta(n)$ is a bound on the time it takes to merge the two halves together.

Recall that the Master Theorem states that if we have a recurrence relation $T(n)$ such that

$$T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^d),$$
$$T(1) = c,$$

then,

$$T(n) \in \begin{cases} \Theta\left(n^d\right) & \text{if } a < b^d \\ \Theta\left(n^d \log n\right) & \text{if } a = b^d \\ \Theta\left(n^{\log_b(a)}\right) & \text{if } a > b^d \end{cases}.$$

We can recognise that the mergesort recurrence relation fits the form required by the Master Theorem, with constants $a = 2$, $b = 2$, and $d = 1$.

$$b^d = 2 = a$$

so, by the master theorem, $T(n) \in \Theta(n \log n)$.

4. 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($C = \{c_1, \ldots, c_n\}$)
    $Points \leftarrow \{(c_1, 0), \ldots, (c_n, 0)\}$
    $Distance \leftarrow$ 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 $\Omega(n \log n)$.