CPSC 320 Midterm 1
Monday, October 18th, 2010

[13] 1. Short answers

[4] a. We frequently prove an $O(n^2)$ bound on the worst-case running time $T(n)$ of an algorithm, without proving any Ω bound on $T(n)$. However, we would never prove an $\Omega(n^2)$ bound on $T(n)$ without also proving a O bound on $T(n)$. Why not?

Solution : Proving an $\Omega(n^2)$ bound on $T(n)$ simply means that the algorithm runs slowly. There is no point in giving an algorithm without providing an upper-bound on its running time (since then the algorithm might be arbitrarily bad).

[4] b. Why is Dijkstra's algorithm greedy?

Solution : Because at each iteration of the `for` loop, it adds to the tree the vertex with the smallest cost at that point, without considering what that might mean for future iterations.

[5] c. Mr. Isulo, a famous alien computer scientist, has designed a greedy algorithm to solve the *Clique* problem (you don't need to know what it is) on a type of graphs called *circular arc graphs* (you don't need to know what they are either). His algorithm starts by choosing the vertex with the most neighbours.
Mr. Isulo wants to prove the following lemma: "Every circular arc graph has a maximum clique that contains the vertex with the most neighbours", but he eventually finds a counter-example to his conjecture. What does this imply for Mr. Isulo's algorithm, and why?

Solution : All of the solutions constructed by Mr. Isulo's algorithm will contain the vertex with the most neighbours, since a greedy algorithm never removes the vertices it has chosen. Because the lemma is false, his algorithm will not return the correct answer for some graphs (that is, it doesn't work).

[5] 2. Give an example of a function $f : \mathbf{N} \to \mathbf{R}^+$ that is not in $O(n \log n)$, but is not in $\omega(n \log n)$ either. Justify your answer briefly.

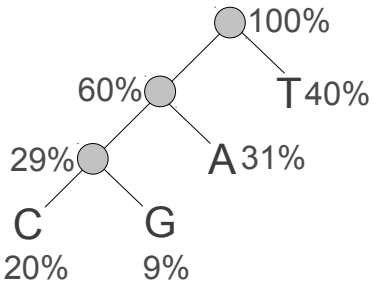
Solution : Consider the function $f : \mathbf{N} \to \mathbf{R}^+$ defined by

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ n^2 & \text{if } n \text{ is odd} \end{cases}$$

Because $f(n) = n$ for every even value of n, $f(n)$ is not in $\omega(n \log n)$. And because $f(n) = n^2$ for every odd value of n, $f(n)$ is not in $O(n \log n)$ either.

[5] 3. A long string consists of the four characters A, C, G and T; they appear with frequencies 31%, 20%, 9% and 40% respectively. What code would Huffman's algorithm return for each of these characters (that is, list the sequence of bits that would be used to represent each character)? Justify your answer.

Solution :



Character	Code
A	01
C	000
G	001
T	1

[5] 4. Recall that a simple path from a vertex s of a graph G to another vertex v of G is a path from s to G that contains each vertex at most once. A student who was interested in finding the **maximum** cost simple path from a vertex s of G to every other vertex of G decided that he could achieve this by using Dijkstra's algorithm, using a Max-Heap instead of a Min-Heap, and replacing the line

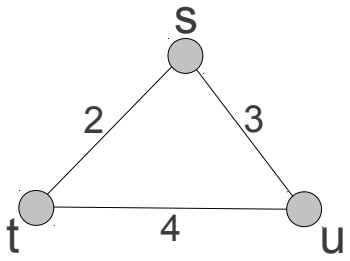
```
if (Cost(v) > Cost(u) + cost(u,v)) then
    Cost(v) ← Cost(u) + cost(u,v)
```

by

```
if (Cost(v) < Cost(u) + cost(u,v)) then
    Cost(v) ← Cost(u) + cost(u,v)
```

Give an example of a graph where this student's algorithm will fail. Show both the output from the modified version of Dijkstra's algorithm, and a path from s to v that is longer than the path found by the algorithm.

Solution : Consider the following graph:

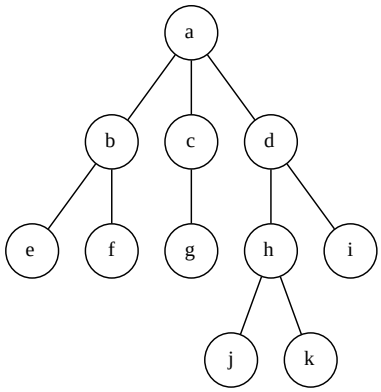


The modified version of Dijkstra's algorithm will first select vertex u (with a cost of 3) and then vertex t (with a cost of 7). However the longest simple path from s to u is the path stu which has a cost of 6.

[12] 5. Given a graph $G = (V, E)$, and a parity function $p : V \to \{\text{"odd"}, \text{"even"}\}$ (that is, we assign a value "odd" or "even" to each vertex of G), the *parity matching* problem consists in finding a subset E' of the edges of G such that for every $v \in V$, the number of edges of E' having v as an endpoint is odd if $p(v) =$ "odd", and even if $p(v) =$ "even".

For instance, if $p(v) =$ "even" for every vertex v of G, then the empty set will be a parity matching of G, p (since every vertex will belong to exactly 0 edge, and 0 is even).

[2] (a) Consider the following graph G and parity function p:



even: a, b, c, g, i, j
odd: d, e, f, h, k

For each edge, determine if the edge (1) **must** belong to a parity matching of G, p, (2) **can not** belong to a parity matching of G, p, or (3) belongs to **some** parity matchings of G, p, but **not to all** of them.

i. The edge (b, f).

Solution : The edge (b, f) must belong to every parity matching of G, p because $p(f)$ is odd, and this edge is the only edge connected to f.

ii. The edge (h, j).

Solution : The edge (h, j) can not belong to any parity matching of G, p because $p(j)$ is even, and this edge is the only edge connected to j.

[8] (b) Describe a greedy algorithm that determines whether or not a **tree** $T = (V, E)$ has a parity matching for a given function p. Hint: think about part (a).

Solution : The idea is the following: for each vertex v of T, we keep a count of the number of edges in E' currently having v as an endpoint. We then do a post-order traversal of the tree. When we get to a vertex v, we have already decided whether or not each one of the edges connecting v to its children is in E'. If the number of such edges has the same parity as $p(v)$ (both even or both odd), then we do not want to add the edge connecting v to its parent to E'. If the two parities differ, then we must add the edge connecting v to its parent to E'.

The algorithm terminates when v is the root of the tree. At that point, if the number of edges connected to v has the same parity as $p(v)$, then the tree has a parity matching and we return E'. If the parities differ, then the tree does not have a parity matching and we return the value `false`.

[3] (c) Analyze the worst-case running time of your algorithm from part (b).

Solution : We are doing a constant amount of work at each node of the tree, and the post-order traversal itselfs takes $\Theta(|V|)$ time. Therefore the worst-case running time of the algorithm is in $\Theta(|V|)$.