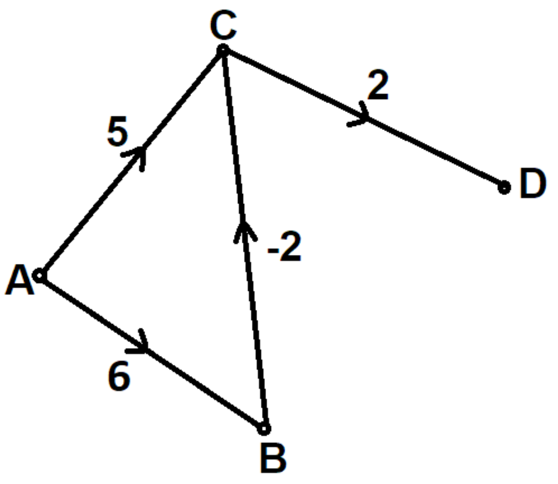
CS180 Homework 3

Due: May 24, 11:59pm

1. (10 pt) (Xiangning) Give an example where Dijkstra's algorithm fails when there are edges of negative weight even if there are no negative cycle. (A negative cycle means a cycle $v_1, \dots, v_r$ where $v_1 = v_r$ and the total weights $\sum_{i=1}^{r-1} \ell_{v_i, v_{i+1}}$ is negative).



Solution: Let's assume A as the source node, and we run the Dijkstra's algorithm as follows: Step 1, we add C to the visited node set since $5 < 6$; Step 2, we add B to the visited node set since $6 < 7$; Step 3, we add D and updates the distance as 7. However, the shortest distance from A to D is actually 6 with path $A \to B \to C \to D$.

2. (25 pt) (Lucas) Given an undirected weighted graph G with n nodes and m edges, and we have used Prim algorithm to construct a minimum spanning tree T. Suppose the weight of one of the tree edge ($(u, v) \in T$) is changed from w to w', design an algorithm to verify whether T is still a minimum spanning tree. Your algorithm should run in $O(m)$ time, and explain why your algorithm is correct. You can assume all the weights are distinct.

Solution: MSTCHECKING$(G, T, u, v, weight)$

(a) NODES$[l] \longleftarrow 0$ for every nodes l.

(b) Remove $e = \{u, v\}$ in T.

(c) Run DFS from u on T, set NODES$[l] = 1$ for every nodes l visited during DFS.

(d) Run DFS from v on T, set NODES$[l] = -1$ for every nodes l visited during DFS.

(e) For every edge $e' = \{u', v'\}$ in G: if NODES$[u']$ + NODES$[v'] = 0$ and $w(e') <$ weight Return False.

(f) Return True

Correctness: When we delete e from T, the nodes of T becomes 2 sets s_1 and s_2. Let s_1 be the cut, so the MST contains the least weight edge crossing the cut, if the updated weight edge e is still minimal, then T still remains the MST.

3. (20 pt) (Xuanqing) Given an undirected graph with nonegative edge weights, the goal of the traveling saleseman problem is to find the lowest weight tour of the graph, where a tour is a path $v_1, v_2, \ldots, v_r$ such that $v_1 = v_r$ and every node in the graph is visited at least once. Note that edges or nodes can appear more than once in the tour. The cost of the tour is the sum of the edge weights in this path, defined as $\sum_{i=1}^{r-1} \ell_{v_i, v_{i+1}}$. We aim to develop an approximation algorithm for solving this problem via MST.

   - Prove the weight of the optimal tour is at least the weight of the MST
   - Describe an algorithm to find a tour that visits every node with the cost at most twice the cost of the MST (so this will be a two-approximation algorithm for traveling salesman problem).

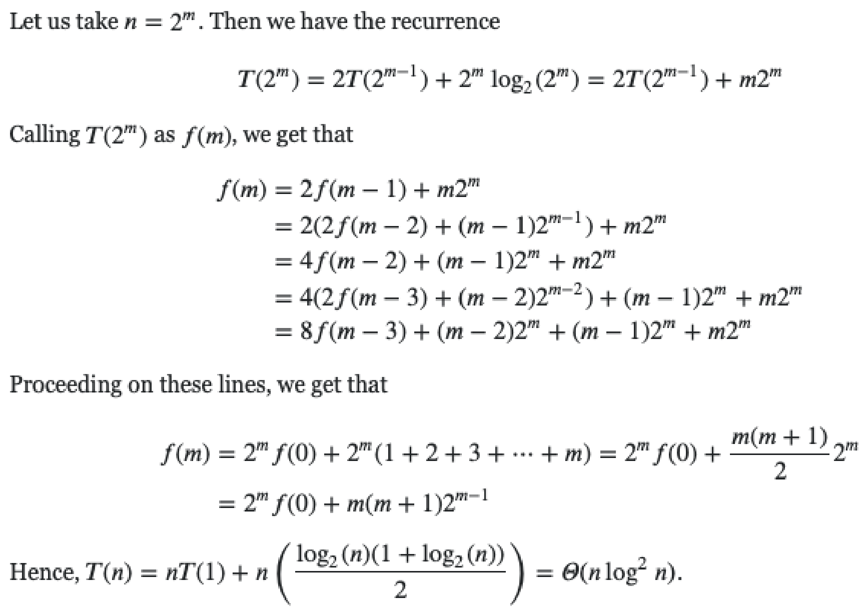
   Solution: a) This is because for any tour, we can remove some edges to make it an MST. b) Compute MST, and then run a DFS on MST to form a tour.

4. (20 pt) (Yuanhao) Derive the order of $T(n)$ for the following recurrence:

$$T(n) = 2T(\frac{n}{2}) + n \log n$$

   This one cannot be derived from the Master Theorem, so you will need to compute the sum of costs at each level directly, like what we did in the class.

   Solution: Should be $O(n(\log n)^2)$ since each layer costs $O(n \log n)$ and there are $\log n$ layers.



Let us take $n = 2^m$. Then we have the recurrence

$$T(2^m) = 2T(2^{m-1}) + 2^m \log_2(2^m) = 2T(2^{m-1}) + m2^m$$

Calling $T(2^m)$ as $f(m)$, we get that

$$\begin{aligned} f(m) &= 2f(m-1) + m2^m \\ &= 2(2f(m-2) + (m-1)2^{m-1}) + m2^m \\ &= 4f(m-2) + (m-1)2^m + m2^m \\ &= 4(2f(m-3) + (m-2)2^{m-2}) + (m-1)2^m + m2^m \\ &= 8f(m-3) + (m-2)2^m + (m-1)2^m + m2^m \end{aligned}$$

Proceeding on these lines, we get that

$$\begin{aligned} f(m) &= 2^m f(0) + 2^m(1 + 2 + 3 + \cdots + m) = 2^m f(0) + \frac{m(m+1)}{2}2^m \\ &= 2^m f(0) + m(m+1)2^{m-1} \end{aligned}$$

Hence, $T(n) = nT(1) + n\left(\frac{\log_2(n)(1 + \log_2(n))}{2}\right) = \Theta(n \log^2 n)$.

Figure 1: Problem 4

5. (25 pt) (Noor) An array with n objects, has a majority element if more than half of its entries are the same. Given an array, design an algorithm to tell whether there is a majority element, and if so, find that element. Your algorithm should run in $O(n \log n)$ time.
   Note that the elements of the array may not be ordered numbers so you can't make any query in the form of "$A[i] > A[j]$". However, you can answer questions of the form "$A[i] = A[j]$" in constant time.

Solution: Divide the array into two parts with equal size. Decide whether each part has a majority element. We can show if there's a majority element, it will be a majority element for at least one of these parts. We can then just check for each majority element of two parts whether it is a majority element for the whole array by making $O(n)$ comparisons. So the Divide-and-conquer algorithm has recurrent $T(n) = 2T(n/2)+n$ which leads to $O(n \log n)$ time complexity.

★ Homework assignments are due on the exact time indicated. Please submit your homework using the Gradescope system. Email attachments or other electronic delivery methods are not acceptable. To learn how to use Gradescope, you can:

- 1. Watch the one-minute video with complete instructions from here:
  https://www.youtube.com/watch?v=-wemznvGPfg
- 2. Follow the instructions to generate a PDF scan of the assignments:
  http://gradescope-static-assets.s3-us-west-2.amazonaws.com/help/submitting_hw_guide.pdf
- 3. **Make sure you start each problem on a new page.**

★ We recommend to use LaTeX, LyX or other word processing software for submitting the homework. This is not a requirement but it helps us to grade the homework and give feedback. For grading, we will take into account both the correctness and the clarity. Your answer are supposed to be in a simple and understandable manner. Sloppy answers are expected to receiver fewer points.

★ Unless specified, you should justify your algorithm with proof of correctness and time complexity.