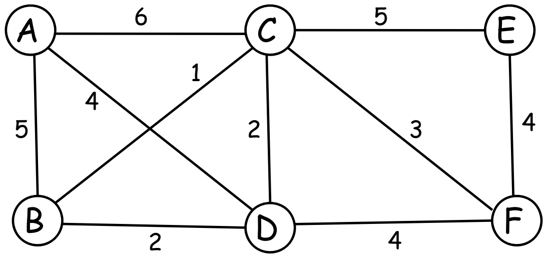
Prof. Salma Elmalaki
Fall 2021

Design and Analysis of Algorithms
EECS215

Homework 2 - Solution Sketch
Practicing Chapters 4 (Greedy) and 5 (Divide-and-Conquer)

1. *(20 Points)* **Chapter 4 - Trees on Graphs (Midterm, Fall 2011).**

   Consider the undirected graph shown in the following figure. It consists of six nodes A,B,C,D,E,F and nine edges with the shown edge costs.



   (a) *(5 Points)* Run Dijkstra's algorithm to find the shortest paths from node A to all other nodes. (Show the final answer and briefly describe the intermediate steps.)

   Answer:

   Dijkstra's algorithm computes a tree with the following edges:

$$(A, B), (A, D), (A, C), (D, F), (C, E)$$

   During the execution, we break ties arbitrarily, which may lead to different SPTs. However, the shortest paths from A to all nodes have always the same cost (distance):

$$d(A) = 0, d(B) = 5, d(C) = 6, d(D) = 4, d(F) = 8, d(E) = 11$$

   (b) *(5 Points)* Run an algorithm of your choice (*e.g.,* Kruskal, Prim, Reverse-Delete) and find a minimum spanning tree. (Show the final answer and briefly describe how you got there.)

   Answer:

   Let's running Kruskal's algorithm, which considers the edges in order of increasing cost:

$$(B, C), \quad (C, D), (B, D), \quad (C, F), \quad (D, F), (E, F), (A, D), \quad (A, B), (C, E) \quad (A, C)$$

   Depending on how cycles appear and on how we break ties between edges with the same cost, we end up with different trees.

   One MST1 consists of edges $(A, D), (B, C), (C, D), (C, F), (E, F)$ and has cost 14.

   Another MST2 consists of edges $(A, D), (B, C), (B, D), (C, F), (E, F)$ and has cost 14.

   Any other algorithm would be acceptable as well.

(c) *(5 Points)* Is the minimum spanning tree of this graph unique? Justify your answer, *i.e.,* if the answer is yes, provide a proof; if the answer is no, provide a counter-example and explain why this is the case.

Answer:

In the previous question we provided already 2 MSTs. This is no surprise, as there are several sets of edges with the same cost. Depending on how we break the ties between them, and in what order cycles tend to appear, we end up with different MSTs.

(d) *(5 Points)* Consider the average distance from A to all other nodes, first by following edges on the shortest path tree (a), let's call it d^{avg}_{SPT}; and then following edges on the minimum spanning tree found in (b), let's call it d^{avg}_{MST}. Which one is greater, d^{avg}_{SPT} or d^{avg}_{MST}? Does the same answer hold for any graph $G = (V, E)$ and node $A \in V$, or is it specific to this example?

Answer:

The distance of all shortest path between node A and all other nodes were calculated in (a), we have: $d^{avg}_{SPT} = (5 + 6 + 4 + 8 + 11)/5 = 34/5$. As found in (b), the MST of G is not unique, the distances of the shortest paths from A to all other nodes using edges in SPT, MST_1, and MST_2 (as named in (b)) are:

	SPT	MST_1	MST_2
A-B	5	7	6
A-C	6	6	7
A-D	4	4	4
A-E	11	13	14
A-F	8	9	10
Total	34	39	41

Hence $d^{avg}_{MST_1} = 39/5$ and $d^{avg}_{MST_2} = 41/5$ and, for graph G, the average distance using SPT is smaller that using any of the possible MSTs. In general, $d^{avg}_{SPT} \leq d^{avg}_{MST}$ since SPT will, by definition, find the shortest path from the node of origin to all other nodes.

2. *(15 points)* **Chapter 4 - Midterm - Fall 2019** - Consider the undirected graph below.

(a) *(5 Points)*Use an algorithm of your choice to find a Minimum Spanning Tree (MST) for this graph. Show the final answer (draw the tree and write down its cost) and briefly describe how you got the answer (which algorithm you used and some intermediate steps).

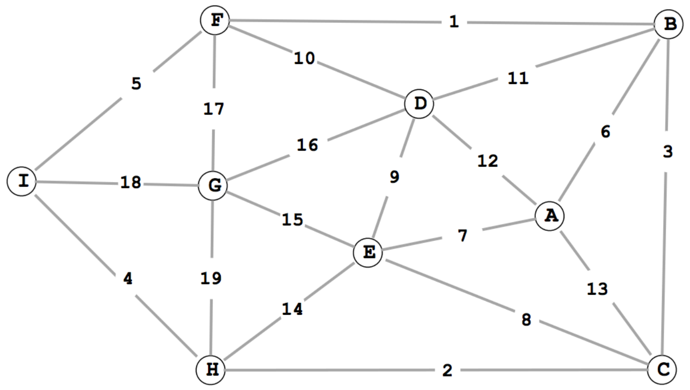
Answer:

You can use any algorithm Use Kruskal's algorithm and arrange the edges in ascending order of cost:
(F,B), add
(H,C), add
(B,C), add
(I,H), add
(I,F), creates a cycle
(A,B), add
(A,E), add
(E,C), creates a cycle

(E,D), add
(F,D), creates a cycle
(D,B), creates a cycle
(D,A), creates a cycle
(A,C), creates a cycle
(H,E), creates a cycle
(G,E), add
(G,D), creates a cycle
(F,G), creates a cycle
(I,G), creates a cycle
(G,H) creates a cycle
Cost = 47

(b) *(10 Points)* Is the MST for this graph unique? Justify your answer, i.e., if the answer is yes, provide a proof; if the answer is no, provide a second MST as a counter-example.

Answer: It is Unique: All edge costs are distinct (Proof by contradiction). Suppose there exist two MSTs for G, T and T*. Then there is some edge e in T that is not in T*. Now add edge e to T*. Now T*+e has a cycle. Consider the edge e' in this cycle with maximum cost (guaranteed to exist by distinct edge cost assumption). Now use the cycle property proved in lecture which is: the MST does not contain e'.



3. *(10 Points)* **Combinatorial Structure of Spanning Trees:** Let $\mathcal{G}$ be a connected graph, and $\mathcal{T}$ and $\mathcal{T}'$ two different spanning trees of $\mathcal{G}$. We say that $\mathcal{T}$ and $\mathcal{T}'$ are *neighbors* if $\mathcal{T}$ contains exactly one edge that is not in $\mathcal{T}'$, and $\mathcal{T}'$ contains exactly one edge that is not in $\mathcal{T}$.

Now, from any graph $\mathcal{G}$, we can build a (large) graph $\mathcal{H}$ as follows. The nodes of $\mathcal{H}$ are the spanning trees of $\mathcal{G}$, and there is an edge between two nodes of $\mathcal{H}$ if the corresponding spanning trees are neighbors.

Is it true that, for any connected graph $\mathcal{G}$, the resulting graph $\mathcal{H}$ is connected? Give a proof that $\mathcal{H}$ is always connected, or provide an example (with explanation) of a connected graph $\mathcal{G}$ for which $\mathcal{H}$ is not connected.

Answer:

Yes, H will always be connected. To show this, we prove the following fact:

(1) Let $T = (V, F)$ and T'=(V,F') be the two spanning trees of G so that $|F - F'| = |F' - F| = k$. Then there is a path in H from T to T' of length k.

What the above statement says is that we can transform T to T' in k steps, swapping one edge at a time. Also notice that this exercise is about spanning trees, *not* about minimum spanning trees.

Proof. We prove this by induction on k. It is true for $k = 1$, by the definition of edges on H. Let's assume that it is true for $k - 1$. Now consider $|F - F'| = k > 1$ and choose an edge $f' \in F' - F$. The tree $T \cup \{f'\}$ contains a cycle C and this cycle must contain an edge $f \notin F'$. The tree $T \cup \{f'\} - \{f\} = T'' = (V, F'')$ has the property that $|F'' - F'| = |F' - F''| = k - 1$. Thus, by induction, there is a path of length $k - 1$ from T'' to T'. Since T and T'' are neighbors, it follows that there is a path of length k from T to T'.

4. *(20 Points)* **Chapter 5:** Computing the asymptotic running time for divide-and-conquer:

   (a) *(10 Points)* **Master Theorem.** If $T(n) = aT(n/b) + O(n^d)$ for some constants $a > 0, b > 1, d \geq 0$, then prove that it is:

$$T(n) = \begin{cases} O(n^d) & \text{if } d > log_b a \\ O(n^d logn) & \text{if } d = log_b a \\ O(n^{log_b a}) & \text{if } d < log_b a \end{cases}$$

   Answer:

   Work at each step:cn^d, $ac(n/b)^d$, $a^2c(n/b^2)^d$, ... $(a/b^d)^i cn^d$. Summing up the work at all levels:

$$T(n) \leq \sum_{i=0}^{log_b n-1} (a/b^d)^i cn^d = cn^d \sum_{i=0}^{log_b n-1} (a/b^d)^i$$

   In order to compute the sum, we need to distinguish three cases for $r = a/b^d$:

   - If $r = 1$, then $d = log_b a$ and:

$$T(n) \leq cn^d \sum_{i=0}^{log_b n-1} (a/b^d)^i = cn^d \sum_{i=0}^{log_b n-1} (a/a)^i = cn^d * log_b n$$

     Therefore, $T(n) = O(n^d logn)$.
   - If $r = a/b^d > 1$, then: $T(n) \leq cn^d \sum_{i=0}^{log_b n-1} (a/b^d)^i = cn^d (r^{log_b n} - 1)/(r - 1) = c/(r-1)n^d(n^{log_b r} - 1) \leq c/(r-1)n^d n^{log_b r} = c/(r-1)n^d n^{log_b a - log_b b^d} = c/(r-1)n^{log_b a}$
   - Therefore, $T(n) = O(n^{log_b a})$
     If $r = (a/b^d) < 1$ then:

$$T(n) \leq cn^d \sum_{i=0}^{log_b n-1} r^i =\leq cn^d \sum_{i=0}^{\infty} r^i = cn^d * 1/(1-r) = c/(1-r)n^d$$

     Therefore, $T(n) = O(n^d)$

   (b) *(10 Points)* Then show that the divide-and-conquer algorithms we did in class Mergesort, Binary Search, Matrix multiplication (classic) and Fast Matrix Multiplication (Strassen) are all special cases of the Master Theorem. In particular, for each of these problems: (i) give the parameters a, b, d for each algorithm and (ii) write down the recursive and closed form equation for $T(n)$.

   Answer:

- Mergesort: $T(N) \leq 2T(N/2) + N = \sum_{i=1}^{log_2 N} N = Nlog_2 N = O(NlogN)$, $a = 2, b = 2, d = 1$ (case 2 in Master Theorem)
- Binary Search: $T(N) \leq T(N/2) + 1 = \sum_{i=1}^{log_2 N} 1 = log_2 N = O(logN)$, $a = 1, b = 2, d = 0$ (case 2 in Master Theorem)
- Matrix multiplication (classic): $T(N) \leq 8T(N/2) + O(N^2) \leq O(N^2) \sum_{i=0}^{log_2 N} 2^i = O(N^2) \cdot \frac{2^{log_2 N+1}-1}{2-1} = O(N^2) \cdot (2N-1) = O(N^3)$, $a = 8, b = 2, d = 1$ (case 3 in Master Theorem: $2 < log_2 8 = 3$ and $O(N^{log_2 8}) = O(N^3)$).
- Fast Matrix Multiplication (Strassen): $T(N) \leq 7T(N/2)+O(N^2) \leq O(N^2) \sum_{i=0}^{log_2 N} (\frac{7}{4})^i = O(N^2) \cdot \frac{(\frac{7}{4})^{log_2 N+1}-1}{\frac{7}{4}-1} = O(N^{log_2 7})$, $a = 7, b = 2, d = 1$ (case 3 in Master Theorem: $2 < log_2 7$ and $O(N^{log_2 7})$).

Note: Yes, we did cover all these cases in class!

5. *(20 Points)* **More recursions:** Solve the following recurrence relations and give a O bound for each of them:

   (a) $T(n) = 2T(n/3) + 1$
   Answer: Use Master's theorem with $a = 2, b = 3, d = 0$. Therefore, $d = 0 < 0.6 < log_3 2$ and
   $$T(n) = O(n^{\log_3 2})$$

   (b) $T(n) = 5T(n/4) + n$
   Answer: Use Master's theorem with $a = 5, b = 4, d = 1$. Therefore, $d = 1 < 1.1 = log5_4$ and
   $$T(n) = O(n^{\log_4 5})$$

   (c) $T(n) = 9T(n/3) + n^2$
   Answer: Use Master's theorem with $a = 9, b = 3, d = 2$. Therefore, $d = 2 = log_3 9$ and
   $$T(n) = O(n^2 \log n)$$

   (d) $T(n) = 2T(n-1) + 1$
   Answer: In this case, the Master's theorem does NOT apply. So, we simply unroll the recursion until we reach $T(0) = 0$.
   $$\begin{aligned} T(n) &= 2T(n-1) + 1 & (1)\\ &= 2(2T(n-2)+1) + 1 = 4T(n-2) + 3 & (2)\\ &= 4(2T(n-3)+1) + 3 = 8T(n-2) + 7 & (3)\\ &= ...T(n) = 2^k T(n-k) + (2^k - 1) & (4) \end{aligned}$$
   This will continue until $k = n$, when: $T(n) = 2^n T(n-n) + (2^n - 1) = 2^n - 1 = \Theta(2^n)$.
   Note: It is ok if you considered other initial conditions, e.g., $T(1) = 0$, $T(1) = c$, $T(0) = c$. The asymptotic answer will be the same.

6. *(15 Points)* **Maximum Sum Subarray**

   Answer: Here we mention two possible solution, the first one is using divide and conquer approach:

- We can find the maximum subarray by dividing the array in two sub-arrays. Then we need to find the max(answer of the left half, answer of the right half, answer such that the subarray crosses the midpoint). In order to find the answer of the right and left halves, we can just call our recursive function on those subarrays. To find the maximum subarray which crosses the midpoint, we should find the maximum sum subarray starting from the mid point to the right side of the array. Also we should find the maximum sum subarray starting from mid-1 to the left side as well. Finally, we need to combine these two maximum sums to have the final answer for the crossing subarray. The running time of this algorithm would be $T(N) = 2T(N/2) + \theta(N)$ which is $\theta(NlogN)$.
- The other solution is to start from the right side of the array and keep track of the current maximum. The Python implementation of this algorithm is stated below:

  ```
  def maxSum(a):
          found_max = a[0]
          current_max = a[0]

          for i in range(1, len(a)):
                  current_max = max(a[i], current_max + a[i])
                  found_max = max(found_max, current_max)

          return max_so_far
  ```

  The running time of such an implementation is $\theta(N)$.