CS584
HW#3

Due: Monday October 15

1. (6 points) Design an algorithm to guess an **unknown** positive integer n using $O(\log n)$ benchmark operations, where a benchmark operation is a comparison of the form

- $k > n$?
- $k = n$?
- $k < n$?

The values of k will be determined by your algorithm. Another way to view this problem is to assume that there's an **oracle** who knows the value of n. The **oracle** will not tell you the value of n, but given any k she will reply correctly to any of the benchmark operations. You may only ask the **oracle** at most $O(\log n)$ questions.

2. (16 points) In the MINIMUM SET COVER PROBLEM, we are given a set of sets, $\{S_1,...,S_n\}$, and we seek a minimum cardinality, k, subset of these sets $S_{j_1},...,S_{j_k}$ such that $\bigcup_{1\le i\le n} S_i = \bigcup_{1\le i\le k} S_{j_i}$. This problem is NP-Hard and testing if there exists set cover of cardinality k is NP-Complete. We are going to study the following Greedy Algorithm, which greedily adds to the cover a set with a maximum number of uncovered elements until every element is covered.

GREED($\{S_1,...,S_n\}$)
  COVER $\leftarrow \varnothing$
  UNCOVERED $\leftarrow \bigcup_{1\le i\le n} S_i$
  UNUSED $\leftarrow \{S_1,...,S_n\}$
  while UNCOVERED $\neq \varnothing$ **do**
    Select $S^* \in$ UNUSED of maximum cardinality
    for each $S \in$ UNUSED **do** $S \leftarrow S - S^*$
    UNCOVERED $\leftarrow$ UNCOVERED-S^*
    UNUSED $\leftarrow$ UNUSED $- \{S^*\}$
    COVER $\leftarrow$ COVER $\cup \{S^*\}$
  return COVER

a Describe an instance of the problem for which GREED fails to find a minimum cover.

For the rest of this problem, we want to find a bound on how poorly GREED can do. We define COVEROPT to be an optimum cover and we define τ =|COVEROPT|. We now bound how well the first τ sets added to COVER approximate COVEROPT.

b What fraction of $\bigcup_{1 \le i \le m} S_i$ (as a function of τ) do we know will be covered by the first set added to COVER?

c What fraction of $\bigcup_{1 \le i \le m} S_i$ (as a function of τ) do we know will be covered by the first τ sets added to COVER? Hint: $1/e \ge (1-1/\tau)^{\tau}$

3. (8 points) As we presented the minimum spanning tree problem, the Blue Rule and the Red Rule assumed that all edge weights are positive, $w : E \to \Re^+$. What happens if we relax this assumption?

In class we showed that if *G* is connected and $w : E \to \Re^+$ and we apply the Blue Rule and the Red Rule until they can't be applied anymore, then the red edges and the blue edges partition *E* and the blue edges are a minimum weight spanning subgraph of *G*.

For the rest of this problem, assume you are given a connected weighted graph $G = (V, E)$ with $w : E \to \Re$.

a Show that applying the Blue Rule and the Red Rule until they can't be applied anymore does not necessarily construct a set of blue edges which comprise a minimum weight spanning subgraph of *G*.

b Modify the Blue Rule and the Red Rule such that when we apply them until they can't be applied anymore, then the red edges and the blue edges partition *E* and the blue edges are a minimum weight spanning subgraph of *G*.

4. (5 points) Consider the following divide-and-conquer algorithm for computing a minimum spanning tree.

INPUT: Connected weighted graph $G = (V, E)$ with $w : E \to \Re^+$.

OUTPUT: The edges of a minimum spanning tree of *G*

$MST(V, E)$

if $|V| = 1$ **then return** $\varnothing$

if $|V| = 2$ **then return** *E*

Partition *V* into V_1 and V_2 such that $\big||V_1| - |V_2|\big| \le 1$

Let E_1 be the edges incident only with vertices of V_1

Let E_2 be the edges incident only with vertices of V_2

Let *e* be a lightest edge incident with a vertex of V_1 and a vertex of V_2

return $MST(V_1, E_1) \cup MST(V_2, E_2) \cup \{e\}$

Does MST always produce a minimum spanning tree? Justify your answer.

CS584

HW#3 SOLUTIONS

1. $k \leftarrow 1$

while $k < n$ **do**

$k \leftarrow 2k$

/* $\frac{k}{2} \le n \le k$ */

return $Binary\_Search(k/2, k)$

Binary_Search(*lo*,*hi*)

$k \leftarrow \lfloor (lo + hi)/2 \rfloor$

if *k*=*n* **then return** *k*

if *n* > *k* **then return** *Binary_Search*(*k*+1,*hi*)

return *Binary_Search*(*lo*,*k*-1)

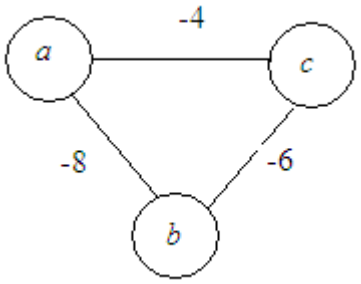
For the initial loop, we note that 1 will be doubled $\lceil \lg n \rceil$ times until it equals or exceeds *n*. Then *Binary_Search* will be executed $O(\lg n)$ times.

2. ***a*** A minimal cover of $\{\{1,2,3,4\},\{1,2,6\},\{3,4,5\}\}$ of cardinality 2 is COVEROPT=$\{\{1,2,6\},\{3,4,5\}\}$, but GREED constructs COVER=$\{\{1,2,3,4\},\{1,2,6\},\{3,4,5\}\}$ of cardinality 3.

b Some set in COVEROPT must contain $\bigcup_{1 \le i \le m} S_i / \tau$ elements, so there must be a set in $\{S_1, ..., S_n\}$ with at least this many elements. And GREED will choose a set at least this large.

c After the first execution of the **while**-loop in GREED, the uncovered fraction of $\bigcup_{1 \le i \le m} S_i$ is at most $\frac{\tau - 1}{\tau}$. The next execution of the **while**-loop will cover at least the fraction $\frac{1}{\tau}$ of this fraction, or $\frac{1}{\tau}\frac{\tau - 1}{\tau}$ of $\bigcup_{1 \le i \le m} S_i$, and the uncovered fraction will be at most $\left(\frac{\tau - 1}{\tau}\right)^2$. After τ iterations the uncovered fraction will be at most $\left(\frac{\tau - 1}{\tau}\right)^\tau$, and the covered fraction will be at least $1 - \left(\frac{\tau - 1}{\tau}\right)^\tau \ge 1 - \frac{1}{e}$.

3. ***a*** Application of the rules to the graph

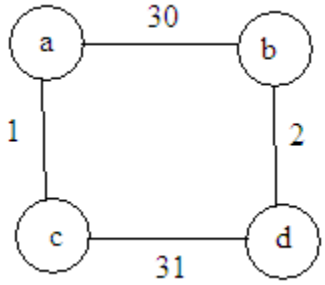


yields the tree with edges $\{ab, bc\}$ of weight -14, but the minimum weight spanning subgraph has edges $\{ab, bc, ac\}$ of weight -18.

b. **Blue Rule**: For any edge *e*, color it *blue* if its weight is negative or if there is a cut for which every edge in the cut is *red* or uncolored and *e* has minimum weight of all edges in the cut.

Red Rule: Color any edge *red* if its weight is positive and it belongs to a cycle with no *red* edges and it is the *uncolored* edge in the cycle of maximum weight.

4. The algorithm does **not** always work. Consider the graph



If the partition is $\{V_1, V_2\} = \{\{a, b\}, \{c, d\}\}$, then the algorithm will return the set of edges $\{ab, ac, cd\}$ of weight 62. The minimum spanning tree is $\{ac, bd, ab\}$ of weight 33.