University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn

Algorithm Theory

Thursday, March 28, 2019, 10:00-12:00

Name: .....

Matriculation No.: .....

Signature: .....

Do not open or turn until told so by the supervisor!

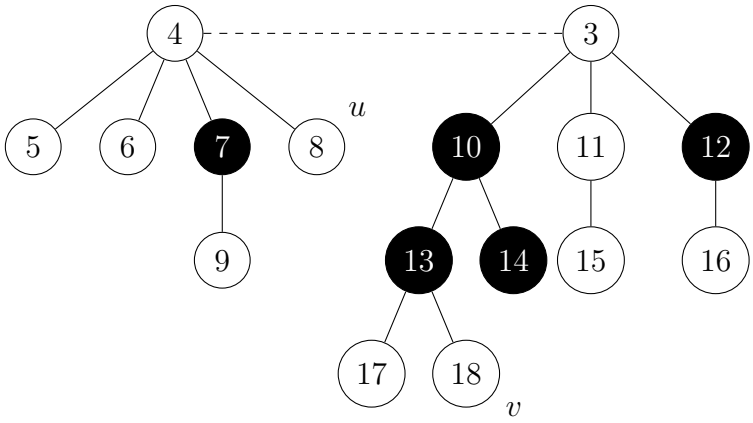
- Put your **student ID** on the table next to you so we can check it.
- Write your **name** and **matriculation number** on this page and **sign** the document.
- Your **signature** confirms that you have answered all exam questions without any help, and that you have notified exam supervision of any interference.
- You are allowed to use a summary of **five (single-sided) A4 pages**.
- Write legibly and only use a pen (ink or ball point). Do **not use red**! Do **not use a pencil**!
- You may write your answers in **English or German** language.
- **No electronic devices** are allowed.
- Only **one solution per task** is considered! Make sure to strike out alternative solutions, otherwise the one yielding the minimal number of points is considered.
- **Detailed steps** might help you to get more points in case your final result is incorrect.
- The keywords **Show...**, **Prove...**, **Explain...** or **Argue...** indicate that you need to prove or explain your answer carefully and in sufficient detail.
- The keywords **Give...**, **State...** or **Describe...** indicate that you need to provide an answer solving the task at hand but without proof or deep explanation (except when stated otherwise).
- You may use information given in a **Hint** without further explanation.
- **Read each task thoroughly** and make sure you understand what is expected from you.
- **Raise your hand** if you have a question regarding the formulation of a task.
- Write your name on **all sheets**!

Task	1	2	3	4	5	**Total**
Maximum	42	27	12	18	21	120
Points						

Task 1: Short Questions *(42 Points)*

(a) *(6 Points)* Consecutively apply the following five operations on the given Fibonacci Heap (black nodes are marked, white nodes are unmarked). How does the Fibonacci Heap look like after the *third* operation and how does it look like after the *fifth* operation?

1. `decrease-key`$(v, 1)$
2. `delete-min`$()$
3. `decrease-key`$(u, 2)$
4. `delete-min`$()$
5. `delete-min`$()$



(b) *(8 Points)* Given an integer $n \geq 1$, your goal is to compute the number of n-bit binary strings with *no consecutive* $1s$. Show how this number can be computed in time linear in n

Remark: 10011010 *is an* 8*-bit string that has consecutive* 1*s, whereas* 10010010 *is an* 8*-bit string that has no consecutive* 1*s.*

(c) *(7 Points)* For this question, we consider the EREW PRAM model. Assume that we are given an array A of length n filled with 0s and 1s. The array is initially stored in the shared memory. Show that a parallel algorithm can determine the smallest number $j \leq n$ such that the number of 1s contained in $A[0, \ldots, j]$ is at least 42 (the result shall be 0 if there is no such j) in time $O(\log n)$. How many processors do you need for this?

(d) *(7 Points)* Let $G = (V, E)$ be a weighted undirected graph and let $e \in E$ be an edge that is not the heaviest edge of any cycle in G (i.e., any cycle containing e also contains an edge of strictly larger weight). Show that e is contained in *every* minimum spanning tree of G.

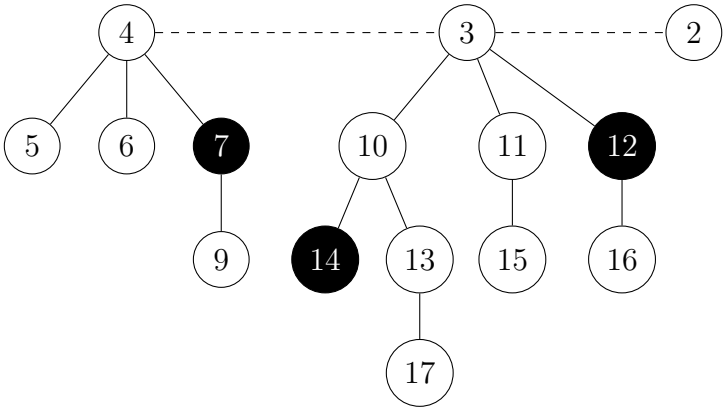
(e) *(6 Points)* Let G be a graph with a maximum clique of size C. Assume that there is an algorithm $\mathcal{A}$ that is a $(1/2)$-approximation algorithm for the maximum independent set problem. Show how one can use $\mathcal{A}$ to find a clique of size at least $C/2$ of G.

(f) *(8 Points)* We are considering the following online minimum vertex cover problem. The nodes V of a graph G are given initially and the edges E of G arrive in an online fashion. An online algorithm has to always maintain a valid vertex cover S of G. Initially $S = \emptyset$ and as new edges appear, an algorithm has to add nodes to S to keep all edges covered. Once a node v has been added to S, v has to remain in S. Give a deterministic online algorithm for this problem that has a competitive ratio of 2. Argue why your algorithm is 2-competitive.

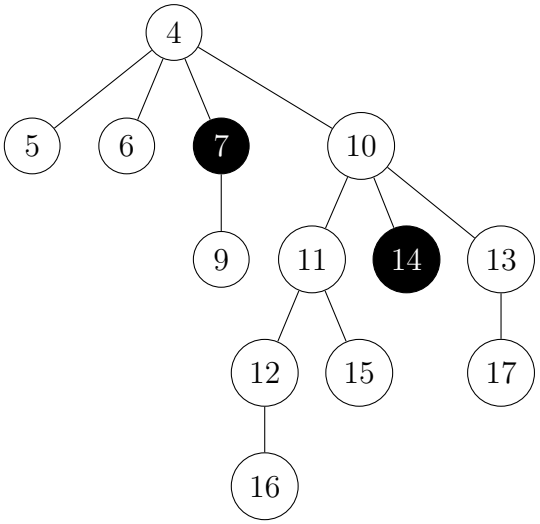
Remark: You can use thereoms proved in the lecture without reproving them.

Sample Solution

(a) State after three Operations:



State after five Operations:



(b) *First Solution:* Let $T_1(j)$ be the number of j-bit strings with no consecutive 1's and most significant bit of 1. Similarly let $T_0(j)$ be the number of j-bit strings with no consecutive 1's and most significant bit of 0. We obtain the following recursive relation:

$$T_0(n) = T_0(n-1) + T_1(n-1), \quad T_1(n) = T_0(n-1)$$

With base cases $T_0(1) = T_1(1) = 1$. The final result is $T_0(n) + T_1(n) = T_0(n+1)$. This recursive function can be computed with the standard dynamic programming approach in $O(n)$, because we have to compute $2n$ values.

Second Solution: From the above equations we obtain

$$T_0(n) = T_0(n-1) + T_1(n-1) = T_0(n-1) + T_0(n-2)$$

with $T_0(1) = 1$ and $T_0(2) = T_0(1) + T_1(1) = 2$, so basically $T_0(n)$ is the $(n+1)$-th Fibonacci number. The result is as above $T_0(n+1)$, i.e. the $(n+2)$-th Fibonacci number. The n-th Fibonacci number can be computed in $O(n)$ bottom up, by simply memorizing the respective last two values and get the next by summing them up.

(c) *First step:* We apply the algorithm from the lecture to compute the partial sums $s_i = \sum_{\ell=0}^{i} A[\ell]$ in $\mathcal{O}(\log n)$ steps in the EREW model using $\mathcal{O}(n/\log n)$ processors. As result we obtain $A[i] := s_i$. *(2 Points)*

Second step: (solution A) Processor i writes $A[i] := n - i$ if $A[i] \geq 42$ and $A[i] := 0$, else. This takes $\mathcal{O}(1)$ steps and $\mathcal{O}(n)$ processors. Then we compute the maximum of A and write it to $A[0]$. This can be done in $\mathcal{O}(1)$ on a strong CRCW PRAM machine with $\mathcal{O}(n)$ processors (processor i writes value $A[i]$ to $A[0]$ and the largest value "wins"). We can simulate a CRCW PRAM machine on an EREW PRAM machine with a time factor $\log n$, hence this is possible in $\mathcal{O}(\log n)$ on an EREW machine. The result is $j := 0$ if $A[0] = 0$, else $j := n - A[0]$. *(3 Points)*

Second step: (solution B) Processor i writes $A[i] := 1$ if $A[i] \geq 42$ and $A[i] := 0$, else. This takes $\mathcal{O}(1)$ steps and $\mathcal{O}(n)$ processors. Then we compute the sum $s := \sum_{\ell=0}^{n} A[\ell]$ of all entries again in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ processors. If $s = 0$ then we set $j := 0$, hence no index as required in the exercise. Otherwise we set $j := n - s + 1$. *(3 Points)*

Second step: (solution C) Conduct a binary search for the smallest index j with $A[j] \geq 42$. Takes $O(\log n)$ time on one processor. *(3 Points)*

In summary the computations requires $\mathcal{O}(\log n)$ steps and $\mathcal{O}(n)$ processors. *(2 Points)*

(d) Assume that T' is a MST of G that does not contain e. Then $T' \cup \{e\}$ has a cycle C. Let e' be an edge on C with $w(e') > w(e)$. Then $T := \big(T \setminus \{e'\}\big) \cup \{e\}$ is a spanning tree and $w(T) = w(T') - w(e') + w(e) < w(T')$, a contradiction to the assumption that T' is a MST.

(e) Let $\overline{G} = (V, \overline{E})$ be the inverted graph of G, that has an edge between two nodes if G has none and vice versa (formally $\overline{E} := \binom{V}{2} \setminus E$).

A maximum clique of G is a maximum subset C of V that fulfills the requirement that for all $u, v \in C$ we have $\{u, v\} \in E$. The latter is equivalent to the fact that for all $u, v \in C$ we have $\{u, v\} \notin \overline{E}$, which is the definition that C is a maximum independent set in $\overline{G}$. *(3 Points)*

First we compute $\overline{G}$. Then we use $\mathcal{A}$ to compute a an independent set C' of $\overline{G}$ with $|C'| \geq |C|/2$ where C is a maximum independent set. According to our argument above, C' is a clique and C a maximal clique $|C'| \geq |C|/2$. *(3 Points)*

(f) We define a greedy algorithm as follows. Initially $S := \emptyset$. For each incoming edge $\{u, v\}$, if $u \notin S$ and $v \notin S$ then $S := S \cup \{u, v\}$. *(5 Points)*

The set S always covers all edges we know so far, because as soon as an edge appears that has no endpoint in S, we add both endpoints of that edge to S. After all edges of G have arrived, the resulting set S forms a maximal matching of G. From the lecture we know that a maximal matching is a $1/2$-approximation of a vertex cover of G. Hence our algorithm is 2-competetive. *(3 Points)*

Task 2: Greedy Approximation Algorithm *(27 Points)*

Let $G = (V, E)$ be a *complete, undirected* graph with positive edge weights $d : E \to \mathbb{R}^+$ that satisy the *triangle inequality*. That is, for any $u, v, w \in V$, we have $d(u, w) \leq d(u, v) + d(v, w)$ (where $d(x, y)$ is the weight of $\{x, y\} \in E$). Given an input parameter $k \geq 1$, our goal is to select a set $S \subseteq V$ of k nodes, such that the maximum distance from a node to its nearest node in S is as small as possible. You can imagine this as building k warehouses in set of cities V, such that the maximum distance between a city and its nearest warehouse is minimized.

For a set $S \subseteq V$ of nodes, we define

$$d(S) := \max_{u \in V} \min_{v \in S} d(u, v).$$

Our goal thus is to find a set S of size $|S| = k$ that minimizes $d(S)$. Let $S^* \subseteq V$ be an *optimal solution*.

Consider the following *greedy algorithm*, which computes a set $\widehat{S}$ of size k. The algorithm first chooses an arbitrary node $u \in V$ and adds it to $\widehat{S}$. Then, as long as $|\widehat{S}| < k$, the algorithms chooses a node $v \in V$ that has the farthest distance to its current closest node in $\widehat{S}$ and the algorithm adds v to $\widehat{S}$. Our goal is to show that the described greedy algorithm has an approximation ratio of at most 2, i.e., that $d(\widehat{S}) \leq 2d(S^*)$.

(a) *(6 Points)* First consider the case $k = 1$. In this case, $\widehat{S} = \{w\}$ for an arbitrary node $w \in V$ and the optimal solution $S^* = \{w^*\}$ such that the maximum distance $d(S^*)$ between w^* and any other node in V is minimized. Show that in this case $d(\widehat{S}) \leq 2d(S^*)$ holds.

(b) *(6 Points)* For the general case (i.e., for arbitrary k), first argue why at the end of the algorithm, the distance between any two nodes in $\widehat{S}$ is at least $d(\widehat{S})$.

(c) *(10 Points)* Show that $d(\widehat{S}) \leq 2d(S^*)$ for general k.

Hint: Assume that after greedily picking k nodes, one node still has a distance bigger than $2d(S^*)$ to its nearest node in $\widehat{S}$. Show that together with (b) this is not possible for a set S^* of size k.

(d) *(5 Points)* Give a weighted complete graph for which the triangle inequality holds and a possible solution $\widehat{S}$ for the above greedy algorithm for which $d(\widehat{S}) = 2d(S^*)$ (you can choose the parameter k).

Sample Solution

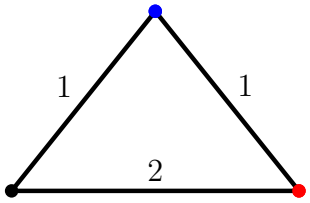
(a) Let $v \in V$ a node with $d(v, w) = d(\widehat{S})$. By definition of $d(S^*)$, we have $d(w, w^*) \leq d(S^*)$ and $d(w^*, v) \leq d(S^*)$. We obtain:

$$d(\widehat{S}) = d(v, w) \overset{\text{trian.-ineq.}}{\leq} d(v, w^*) + d(w^*, w) \leq 2d(S^*)$$

(b) After picking k warehouses greedily according to the strategy above, let u be the node with the maximum distance $d(\widehat{S})$ to its nearest warehouse. Assume there would be two nodes $w_1, w_2 \in \widehat{S}$ with distance *smaller* than $d(\widehat{S})$ (assume w.l.o.g. w_1 was picked before w_2). Since u is farther from any warehouse than w_2 we would have picked u instead of w_2, a contradiction.

(c) We do as told in the hint and assume a node v has a bigger distance than $2d(S^*)$ to its closest warehouse in $\widehat{S}$. Then, according to part (b), all pairs of "greedy"-warehouses $w \in S$ have distance bigger than $2d(S^*)$ as well. Each "greedy" warehouse in $\widehat{S}$ and also the point v must have an "optimal" warehouse in S^* within distance $d(S^*)$. Since the pairwise distances in $\widehat{S} \cup \{v\}$ are all bigger than $2d(S^*)$, the closest "optimal" warehouses in S^* to each node in $\widehat{S} \cup \{v\}$ are ***unique***. However, $|\widehat{S} \cup \{v\}| = k + 1$ and we have only $|S^*| = k$ "optimal" warehouses, a contradiction.

(d) We choose $k = 1$ and $\widehat{S}$ consists of the red node and S^* consists of the blue node.



Task 3: Amortization *(12 Points)*

Consider some data structure D that supports the operations $D.\texttt{insert}(x)$ to insert object x into D and $D.\texttt{lookup}(y)$ to read data from D. The operation $D.\texttt{lookup}()$ always has a cost of 1. The cost of the operation $D.\texttt{insert}(x)$ depends on the number of objects in D. Assume that before carrying out the operation, the number of objects stored in D is $k-1$. If k is an integer power of 3, the cost of the operation equals k and otherwise the cost of the operation equals 1.

A sequence of n `insert`/`lookup` operations is performed on D (which is initially empty). Use the *potential function* method to show that each operation has constant amortized cost.

Hint: For an integer k, the largest integer power of 3 that is at most k is $3^{\lfloor \log_3 k \rfloor}$.

Remark: If you did not find a potential function to show that each operation has constant amortized cost, you can use the accounting *method and get up to 6 points instead.*

Sample Solution

With potential function: Let D_i be the data structure after the i^{th} operation. Define the potential function

$$\Phi(D_i) = \frac{3}{2}\left(|D_i| - 3^{\lfloor \log_3 |D_i| \rfloor}\right) \qquad \textit{(3 Points)}$$

and the amortized cost of the i^{th} operation as $a_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$, where c_i is the actual cost.

As $\Phi(D_i) \geq 0$ for all i and $\Phi(D_0) = 0$, the total amortized costs are upper bounded by the total actual costs. *(2 Points)*

If the i^{th} operation is `lookup`, we have for the amortized cost

$$a_i = c_i + \Phi(D_{i-1}) - \Phi(D_{i-1}) = c_i = 1\ . \qquad \textit{(2 Points)}$$

If the i^{th} operation is `insert` and $|D_i|$ is not a power of 3, we have

$$\begin{aligned} a_i &= 1 + \frac{3}{2}\left(|D_i| - 3^{\lfloor \log_3 |D_i| \rfloor}\right) - \frac{3}{2}\left(|D_i| - 1 - 3^{\lfloor \log_3 (|D_i|-1) \rfloor}\right) \\ &\overset{(*)}{=} 1 + \frac{3}{2}\big(|D_i| - (k-1)\big) - \frac{3}{2}\big(|D_i| - (k-1)\big) = 1 + \frac{3}{2} \end{aligned}$$

If the i^{th} operation is `insert` and $|D_i| = 3^k$ for some $k \geq 1$, we have

$$\begin{aligned} a_i &= 3^k - \big(\Phi(D_{i-1}) - \Phi(D_i)\big) = 3^k - \frac{3}{2}\left(3^k - 1 - 3^{\lfloor \log_3 (3^k-1) \rfloor}\right) \\ &\overset{(*)}{=} 3^k - \frac{3}{2}\left(3^k - 1 - 3^{k-1}\right) = \frac{3}{2}\ . \end{aligned}$$

(5 Points)

$(*)$: For $k \geq 1$ we have $3^k > 3^k{-}1 > 3^{k-1} \Rightarrow k > \log_3(3^k{-}1) > k-1 \Rightarrow \lfloor \log_3(3^k{-}1) \rfloor = k{-}1$.

With accounting method: For each `lookup` we pay nothing and subtract nothing from the account. For each `insert` where $|D_i|$ is no power of three, we pay $3/2$ coins to the account. For each `insert` where $|D_i|$ is a power of three, we subtract $|D_i| - 3/2$ from the bank to (almost) pay for the cost of this operation. *(2 Points)*

We show that the account is always at least 0 by induction. Initially the account is empty, i.e. ≥ 0. We subtract only from the account when an `insert` occurs, where $|D_i| = 3^k$ for a $k \in \mathbb{N}$. We assume that (by the induction hypothesis) the account was at least 0 after the last "expensive" `insert`, where we had 3^{k-1} elements in D. Since then there must have been at least $3^k - 1 - 3^{k-1}$ "cheap" `insert` operations for each of which we paid $3/2$ to the account. That means we have at least $\frac{3}{2}(3^k - 1 - 3^{k-1}) = 3^k - 3/2$ on the account, which is exactly what we subtract for this operation. *(3 Points)*

In that setting `lookup` has amortized cost of 1. Each `insert` where $|D_i|$ is no power of three has amortized cost $1 + 3/2 = 5/2$. Each `insert` where $|D_i|$ is a power of three, we have amortized cost $|D_i| - (|D_i| -) = 3/2$. *(1 Point)*

Task 4: Edge Coloring and Matching *(18 Points)*

The edge chromatic number of a graph G is defined as the smallest number of colors that suffice to color the edges of G such that no two edges incident to the same node have the same color. Consider an arbitrary bipartite graph $G = (A \cup B, E)$, where Δ denotes the maximum degree of any node of G.

(a) *(7 Points)* Show that the edge chromatic number of a bipartite graph G is Δ if G is Δ-regular.

(b) *(6 Points)* Show that the edge chromatic number of a bipartite graph G with maximum degree Δ is at most Δ even if G is a non-regular. You can assume that (a) is true if you could not prove it.

(c) *(5 Points)* Show that every bipartite graph G with maximum degree Δ has a matching of size at least $|E|/\Delta$. You can assume that (b) is true if you could not prove it.

Sample Solution

(a) Consider the following algorithm to color the graph with at most Δ colors. Since G is a regular bipartite graph, it has a perfect matching by the Hall's theorem. Therefore, Let M_1 be the perfect matching of G. Then, color all the edges in M_1 by color c_1, and remove all these edges from G Let the new graph denoted by G_2. Then, since G was Δ-regular, G_2 is $(\Delta - 1)$-regular. We repeat this for $\Delta - 1$ more times to color all the edges of the graph to $c_1, \ldots, c_\Delta$.

(b) We transform the given by partite graph to a regular bipartite graph by adding extra nodes to the side with less number of nodes. *(2 Points)*

Then, each time pick two nodes from in the left and the right side that have degree less than Δ and connect them. Repeat this until there is no node left with degree less than Δ. *(2 Points)*

Then we apply the same technique in (a) to color this graph. Note that a coloring of graph is a coloring of the original one. *(2 Points)*

(c) Consider the edge coloring with at most Δ colors, which was shown to exist in part b). On average each color class contains $|E|/\Delta$ edges, thus there is at least one color class that has size at least $|E|/\Delta$. Since edges of one color form a matching, any class with size at least $|E|/\Delta$ is a matching of that size for G.

Task 5: Unique-Set-Cover Problem *(21 Points)*

We are given a set of n elements and a collection S of m subsets of the elements. The *Unique-Set-Cover Problem* requires to choose a collection $S' \subseteq S$ of sets that maximizes the number of uniquely covered elements. An element is uniquely coved by S' if the element is in *exactly one* of the sets in S'. Note that the sets in S' do not necessarily need to cover all n elements at least once, i.e., S' does not need to be a set cover.

Consider the following algorithm $\mathcal{A}$: Initialize $S' := \emptyset$. For each subset $s \in S$, add s independently with some given probability p to S'.

(a) *(6 Points)* First assume that every element is in exactly x subsets of S (for some value x). Provide the expected number of uniquely covered elements in the solution returned by Algorithm $\mathcal{A}$ (as a function of n, x, and p).

(b) *(7 Points)* Assume now that each element is contained in at most x subsets and in at least $x/2$ subsets of S (for a given value x). Show that by choosing the probability p appropriately, the above randomized algorithm can be guaranteed to produce a constant factor approximation to the Unique-Set-Cover problem in expectation.

(c) *(8 Points)* Provide a randomized algorithm with approximation ratio $1/O(\log m)$ for the Unique-Set-Cover problem.

Sample Solution

(a) $n\binom{x}{1}p(1-p)^{x-1}$

(b) Let us fix $p = 1/x$. For an arbitrary integer d, fix an element s which is in d subsets of S. Then the probability that s is uniquely covered by the solution of $\mathcal{A}$ is at least

$$\binom{d}{1}\frac{1}{x}\left(1-\frac{1}{x}\right)^{d-1} \geq \frac{x}{2}\frac{1}{x}\left(1-\frac{1}{x}\right)^{x-1} \geq \frac{1}{2}\left(1-\frac{1}{x}\right)^{x} \geq \frac{1}{2e}.$$

In expectation we have at least $n/2e$ uniquely covered elements by the solution of $\mathcal{A}$ with parameter $p := 1/x$. Obviously the best solution can have at most n uniquely covered elements. The approximation ratio is therefore $1/2e$.

(c) Let us partition the nodes into $\lceil \log m \rceil + 1$ sets $I_0, \ldots, I_{\lceil \log m \rceil}$, where each element s in set I_i is in d subsets of S for $2^i \leq d < 2^{i+1}$. Then, there is a set I_k with cardinality at least $\Omega(n/\log m)$. We apply Algorithm $\mathcal{A}$. For the elements in I_k the condition of part (b) applies ($x := 2^{k+1}$), meaning that we have an approximation ratio c (where $0 < c < 1$ is constant) for the elements in I_k. That is our greedy solution uniquely covers at least $c|I_k| = \Omega(n/\log m)$ nodes. Since the optimal solution can have at most n uniquely covered elements we have the required approximation ratio of $c|I_k|/n = O(1/\log m)$.