CS-E3190 Principles of Algorithmic Techniques

05. Greedy Algorithms – Tutorial Exercise

1. **Maximum weight matching.** Consider a graph $G = (V, E)$, where $|V| = n$, and $|E| = m$. Suppose that the graph is weighted i.e. each edge $e \in E$ is assigned a weight $w(e) \in \mathbb{N}$. Recall that a matching $M \subseteq E$ is a set of edges such that none of them share an endpoint. A matching is called maximal if no edges can be added to it.

   The goal is to design a greedy 2-approximation algorithm for finding a maximum weight matching that runs in $O(m)$.

   (a) Give a greedy algorithm algorithm that fits the goal.
   Hint: you can assume that the input graph is given as a list of edges, sorted in a decreasing weights order.

   Solution.
   By slight abuse of notation, assume $E = (e_1, \dots, e_m) = (\{u_1, v_1\}, \dots, \{u_m, v_m\})$ is the list of edges ordered by weight in descending order. For a set of edges $A \subseteq E$ let $V(A)$ be the set of vertices covered by the edges in A.

   Algorithm 1: Greedy algorithm for maximum weight matching

   ```
   Initialize matching M = ∅
   for i = 1, ..., m do
       if e_i ∩ V(M) = ∅ then
           Add e_i to M
       end
   end
   return M
   ```

   (b) Prove that this algorithm is correct (returns a matching that has a maximal number of edges) and that it gives a 2-approximation of a maximum matching.

   Solution.
   We need to show that the algorithm returns a valid matching (corectness) and that the weight of this matching is at least $1/2$ of the optimum (2-approximation).
   Correctness. By induction on $i \in \{1, \dots, m\}$, M is a matching. Indeed,

   - The initial set $M = \emptyset$ is a matching.
   - Assume that M is a matching after processing edge e_i. During the $(i+1)^{th}$ step, the edge e_{i+1} is only added to M if $e_i \cap M = \emptyset$. Hence after processing e_{i+1} the set M is still a matching.

   Approximation. Let $w(M)$ be the weight of M and let M^* be a maximum weight matching. We want to show that $w(M^*) \leq 2 \cdot w(M)$.
   If $M = M^*$, we are done. Otherwise, $\exists e^* \in M^* \setminus M$.

   - We will first show that $\exists e \in M$ s.t. $w(e) \geq w(e^*)$ and $e \cap e^* \neq \emptyset$ by contradiction. Suppose there is no such edge. Then $\forall e \in M$, $w(e) < w(e^*)$ or $e \cap e^* = \emptyset$.
     If $\forall e \in M, e \cap e^* = \emptyset$, e^* would be added to M by the algorithm which is impossible since $e^* \in M^* \backslash M$. Hence, $\exists e \in M$ s.t. $e \cap e^* \neq \emptyset$.
     Suppose that $\forall e \in M$ s.t. $e \cap e^* \neq \emptyset$, $w(e) < w(e^*)$. Since the algorithm runs over a descending weight list of edges, this means that when the algorithm considers e^*, there is no intersection between the end points of e^* and the vertices in M, ie. $e^* \cap V(M) = \emptyset$. So if e^* is free then the algorithm must add e^* to M before considering the edges $e \in M$, s.t. $e \cap e^* \neq \emptyset$. This is impossible since $e^* \in M^* \backslash M$.

Hence for each $e^* \in M^*$, there exists an edge $e \in M$ s.t. $w(e) \geq w(e^*)$ and $e \cap e^* \neq \emptyset$.

- Let us now prove that M is a 2-approximation. For $e^* \in M^*$, let $f(e^*)$ be an edge in M as described above. Then,

$$w(M^*) = \sum_{e \in M \cap M^*} w(e) + \sum_{e^* \in M^* \backslash M} w(e^*) \leq \sum_{e \in M \cap M^*} w(e) + \sum_{e^* \in M^* \backslash M} w(f(e^*))$$

Note that two edges e' and e'' in M^* can have $f(e') = f(e'')$ but not three or more (more than two edges in M^* cannot be adjacent to the same edge without violating the fact that M^* is a matching). Let F be the set of edges $e \in M$ s.t. $\exists e^* \in M^* \backslash M$ and $e = f(e^*)$. The previous observation implies that,

$$\sum_{e^* \in M^* \backslash M} w(e^*) \leq 2 \cdot \sum_{e \in F} w(e).$$

Hence,

$$w(M^*) = \sum_{e \in M \cap M^*} w(e) + \sum_{e^* \in M^* \backslash M} w(e^*) \leq \sum_{e \in M \cap M^*} w(e) + 2 \cdot \sum_{e \in F} w(e) \leq 2 \cdot w(M).$$

(c) Assuming a sorted array, prove that it runs in time $O(m)$.

Solution.

There are exactly m iterations since the algorithms goes through the ordered list of nodes exactly once. Hence, we need to show that each iteration can be performed in constant time i.e. $(e \cup M = \emptyset)$ can be tested in constant time.

To show this, we will give a practical encoding of M. Let U_i denote the i^{th} element of $U \in \{0,1\}^n$ s.t.

$$U_i = \begin{cases} 0 & \text{if } u_i \notin V(M) \\ 1 & \text{otherwise} \end{cases}$$

If U is up to date, checking whether $e = \{u_i, u_j\}$ can be added to M is done by checking that both U_i and U_j are zero, which is done in constant time.

Moreover, maintaining U up to date requires setting $U_i = U_j = 1$ after adding $\{u_i, u_j\}$, which is also done in constant time.

Finally, U can be initialized in $O(m)$ time since $n \leq 2m$. Hence it is possible to initialize such a structure before going through all the edges, using $O(m)$ time to initialise, $O(m)$ time to go though all iterations that are all performed in constant time.