Assignment-4

Design and Analysis of Algorithms

Trishan Mondal

⚠ **Disclaimer.** Consider the following set of students

$$\mathcal{P} = \{\text{Deepta Basak, Aaratrick Basu, Soumya Dasgupta, Priyatosh Jana}\}$$

Discussion of solutions to the assignment problems are limited to the people of set $\mathcal{P}$ only. I have shared some of my solutions to vedanta. Most of the problems in this assignment has general solution. If any other person have same solution as mine is not my fault.

---

§ Problem 15

Problem. Consider the Bellman-Ford algorithm (see the code below) for determining the shortest distance in a weighted graph from a source vertex s to all other vertices. For all vertices v, the algorithm maintains `v.dist` and `v.parent`. Assume the graph has no negative-weight cycle. Prove or disprove (to disprove provide a counter example) the following statements:
(a) at every point in the execution of the algorithm, for every vertex v with `v.dist`$< \infty$, the directed path (written backwards here):

$$\text{v} \leftarrow \text{ v.parent } \leftarrow \text{ v.parent.parent } \leftarrow \cdots$$

leads from s to v;
(b) the cost of this path is `v.dist`.

Solution. (a) Let $G = (V, E)$ be the directed graph and s is the source. We will prove this part by induction on the number of iteration step. For the base case we will consider the 1-st iteration. In this step only the vertices $v \in V$ with $\{s \to v\} \in E$ can have `v.dist` $< \infty$. All such vertices satisfy `v.parent` $=$ `s`. Thus the base case is true. Let, the statement is true at n-th iteration step. We will look at the $(n+1)$-th iteration.

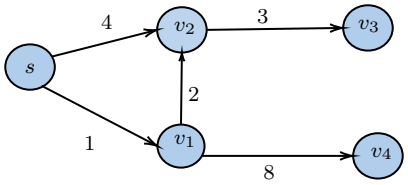
- Take $v \in V$ such that `v.dist` $< \infty$. Let v $\leftarrow$ v.parent $\leftarrow$ v.parent.parent $\leftarrow \cdots$ be the path at n-th iteration. If no vertex of it get updated at $(n+1)$-th iteration step, by the induction hypothesis we can say this will leads us to a path from $s \to v$.

- Any vertex of the path v $\leftarrow$ v.parent $\leftarrow$ v.parent.parent $\leftarrow \cdots$ get updated at $(n+1)$-th iteration. Let u be the first vertex from left in the above path get updated at this iteration. We have the path $\mathcal{P}$: v $\leftarrow$ v.parent $\leftarrow \cdots \leftarrow u$ so that `u.parent.dist` $< \infty$. This `u.parent` is updated at some iteration step $m \leq n$. So there is a path u $\leftarrow$ u.parent $\leftarrow$ u.parent.parent $\leftarrow \cdots$, which is path from $s \to u$. If we concatenate this path with $\mathcal{P}$. We will get the following path give us a path from $s \to v$,

$$\text{v} \leftarrow \text{ v.parent } \leftarrow \text{ v.parent.parent } \leftarrow \cdots u \leftarrow \text{ u.parent } \leftarrow \text{ u.parent.parent } \leftarrow \cdots$$

Thus our induction step is complete. ■

(b) **For every outer loop iteration the cost will be equal to the distance but at every update it may not be equal**. Consider the following directed graph,



At first iteration (we will consider only this updates): (considering the vertex set $V = \{s, v_1, v_2, v_3, v_4\}$)

- (s, v_1) gets updated, v_1.dist= 1, `parent.`v_1 $=s$.
- (s, v_2) get updated, v_2.dist= 4, `parent.`v_2 $=s$
- (v_2, v_3) get updated, v_3.dist= 7, `parent.`v_3 $=v_2$
- (v_1, v_2) get updated, v_2.dist= 3, `parent.`v_2 $=v_1$

Here, v_3.dist= 7 but, v_3.parent $= v_2$ and v_2.parent $= v_1$, v_1.parent $= s$, corresponding path $s \to v_1 \to v_2 \to v_3$ has cost 6.

For outer loop iteration/ each iteration step: By the part(a), we can say at every step every vertex v with `v.dist` $< \infty$, there is a path from $s \to v$ given by,

$$\mathcal{P} : \text{v} \leftarrow \text{ v.parent } \leftarrow \text{ v.parent.parent } \leftarrow \cdots$$

We will induct on the vertices on the path, for the base case we have `s.dist` $= 0$. Let, v_1 be the vertex in this path so that `v_1.parent` $= s$, we have `cost`$(v_1 \leftarrow s)$ = `v_1.dist`. Let v_k be the k-th vertex in the path $\mathcal{P}$ which satisfy

$$\texttt{cost}(v_k \leftarrow v_k.\text{parent} \leftarrow \cdots s) = \texttt{v\_k.dist}$$

v_{k+1} be the vertex in $\mathcal{P}$ with `v_(k+1).parent` $= v_k$. When v_{k+1} was updated for the last time we must have,

$$\begin{aligned} v_{k+1}.\text{dist} &= v_{k+1}.\text{parent.dist} + \ell(v_{k+1}, v_{k+1}.\text{parent}) \\ &= v_k.\text{dist} + \ell(v_{k+1}, v_k) \\ &= \texttt{cost}(v_k \leftarrow v_k.\text{parent} \leftarrow \cdots s) + \ell(v_{k+1}, v_k) \\ &= \texttt{cost}(v_{k+1} \leftarrow v_{k+1}.\text{parent} \leftarrow \cdots s) \end{aligned}$$

By induction we can say cost of such path is `v.dist`. ■

§ Problem 16

Problem. In the Bellman-Ford algorithm, one picks an edge (u,v) such that

$$\text{u.dist } + \ell(u, v) < \text{ v.dist},$$

and sets v.dist = u.dist $+\ell(u, v)$. (If no such edges exist, we stop.) To locate the next edge to perform this update operation, the algorithm scans the edges in a fixed order in each iteration. Could we have picked the next edge to update arbitrarily? Show that for all large n, there is an acyclic weighted graph with n vertices and an order of updates (each update should reduce v.dist for some vertex v) so that the algorithm performs $2^{\Omega(n)}$ updates before it terminates.

Solution. Consider the graph $G = (V, E)$ with the vertices are labeled with numbers $\{1, \cdots, n\}$ and let E contains the edges $k \to \ell$, whenever $k \leq \ell$ and let $w(k, \ell) = 2^\ell$ are the weights of the corresponding edges. So,

$s = 1$ be the source vertex. For every vertex $m \in \{1, \cdots n\}$ consider the set $M = \{1, \cdots, m\}$. Every subset of M containing 1 and m gives us a path between 1 and m. Let us define the order of updates as following,

$$\text{order}(S) = \sum_{x \in S} 2^x \,,\; S \subseteq \{1, \cdots, n\}$$

Consider all the subsets $S_1, \cdots, S_{2^{m-2}}$ of M containg 1 and m, they are labeled in a way such that,

$$\text{order}(S_1) < \cdots < \text{order}(S_{2^{m-2}})$$

Consider the following sequence of subsets,

$$S_{2^{m-2}}, S_{2^{m-2}-1}, \cdots, S_1$$

Let P_i be the path from 1 to m corresponding to the subset S_i. If we take ordering of the updates according to the above sequence of vertices, we can note the weight of the paths P_i decreases in each step and hence it's a valid update. For each $1 < m \leq n$ thus we need 2^{m-2} updates.

$$\begin{aligned} \#\text{total updates} &= \sum_{m=2}^{n} 2^{m-2} \\ &= 2^{n-1} - 1 \\ &\geq 2^{\frac{n}{2}} \\ \Rightarrow \log_2(2^{n-1} - 1) &\sim \Omega(n) \end{aligned}$$

So in this case we need $2^{\Omega(n)}$ updates to find the shortest path between 1 to n. ■

§ Problem 17

Problem. Consider the following part of the code for the Bellman-Ford algorithm.

```
1       s.dist = 0
2        s.parent = None
3         for i = 1, 2, ..., T: # the outer for loop
4            for v in V:
5                for (w, ℓ) in adj[v]:
6                    if v.dist + ℓ < w.dist:
7                        w.dist = v.dist + ell # update distance
8                        w.parent = v # update parent
9
```

Show the following (when appropriate, use induction; state the induction hypothesis precisely):

(a) At all times, for all vertices v, if $u =$ `v.parent` is not `None`, then `u.dist` $+ \ell(u, v) \leq$ `v.dist`.

(b) If `v.dist` was updated in iteration k of the outer for loop, then the first $k + 1$ elements of the sequence

$$\texttt{v}, \texttt{v.parent}, \texttt{v.parent.parent}, \ldots \quad (1)$$

are not `None`.

(c) Suppose $T = |V|$ and v.dist was updated in the last iteration. Argue that the path described in eq. (1) ends in a cycle, and the sum of the lengths of the edges of that cycle is negative (beware of ∞).

(d) Suppose there is a negative-weight cycle C reachable from vertex s. Argue that for some vertex v in C, v.dist will be updated in iteration $|V|$ of the outer for loop.

Solution.

(a) Let, $u =$ `v.parent` which is not `None`. At the iteration step where we updated `v.parent` to u, we must have `u.dist` $+\ell(u, v) <$ `v.dist` (before this iteration this inequality was mentained). After it get updated we must have `u.dist` $+\ell(u, v) =$ `v.dist`. If at any further iteration, `u.parent` got updated or `u.dist` got updated, it can be the case that `v.dist` didn't get updated. At each such update `u.dist` must decrease. And hence the given inequality hold at each time.

(b) We establish this through induction on k. Our induction hypothesis asserts that if `v.dist` is updated in the k^{th} iteration of the outer loop, then the first $k+1$ elements of the sequence $(v, \texttt{v.parent}, \dots)$ are not `None`. To initiate the induction, let's consider $k = 1$ and assume `v.dist` is updated in the first iteration. Such an update can only occur if there exists a vertex u such that $u.\texttt{dist} + \ell(u, v)$ is finite because `v.dist` $= \infty$ before this iteration. Consequently, `v.parent` $= u$ is not `None`," and thus, the first 2 elements of $(v, \texttt{v.parent}, \dots)$ are not `None`. This demonstrates the validity of the hypothesis for the base case $k = 1$. Now, we assume the hypothesis holds for some $k \geq 1$ and consider the scenario in which `v.dist` is updated in the $(k+1)^{\text{th}}$ iteration.

This event occurs if and only if there exists a vertex u such that $u.\texttt{dist} + \ell(u, v) < d$, where d represents the value of `v.dist` at the conclusion of the k^{th} iteration. Due to the sequence of loop executions, it must be the case that $u.\texttt{dist}$ is updated in the k^{th} iteration either after v or in the $(k+1)^{\text{th}}$ iteration before v.

If the update to u occurred in the k^{th} iteration, the induction hypothesis is applicable. If it did not, we examine u and $u.\texttt{parent}$ and apply the same reasoning. The absence of updates in the k^{th} iteration implies a similar lack of updates in the $(k+1)^{\text{th}}$ iteration. Thus, it is impossible that none of the vertices in the sequence $(v, \texttt{v.parent}, \dots)$ were updated in the k^{th} iteration. Consequently, we find a vertex w updated in the k^{th} iteration that is present in the sequence $(v, \texttt{v.parent}, \dots)$. By the induction hypothesis, the first $k+1$ terms of the sequence $(w, w.\texttt{parent}, \dots)$ are not `None`. Thus, the first $k+2$ terms of the sequence $(v, \texttt{v.parent}, \dots)$ are not `None` , as w follows v. Consequently, through induction, we establish the validity of the statement for all $k \geq 1$.

(c) In part (b), we demonstrated that when `v.dist` is updated during the k^{th} iteration, the initial $k+1$ vertices within the sequence $(v, \texttt{v.parent}, \dots)$ cannot be `None`. Using the proof provided in **15**, it follows that these vertices define a path from some vertex u to v.

Consequently, if there exists a vertex v that gets updated during the final iteration, we establish the existence of a path consisting of $|V| + 1$ vertices. Since the graph contains $|V|$ distinct vertices, such a path must contain a cycle. Let us denote the vertices present in this cycle as $u_1, \dots, u_r, u_1$, in that order. Since this cycle is part of the sequence $(v, \texttt{v.parent}, \dots)$, and `v.dist` receives an update during the last iteration, it logically follows that the sum of the edge lengths within the cycle results in a reduction of `v.dist` at the $(|V| - 1)^{\text{th}}$ iteration.

The path obtained by removing the vertices $u_2, \dots, u_r, u_1$ from the sequence $(v, \texttt{v.parent}, \dots)$ must have been previously discovered in an iteration before the final one. Consequently, the decrease in the value of `v.dist` during the last iteration can only occur if the edge lengths within the cycle sum to a negative value. Therefore, the presence of a vertex being updated in the last iteration indicates the existence of a cycle within the graph, the edge lengths of which sum to a negative value.

(d) Let's consider the vertices forming the cycle as $u_1, \dots, u_r, u_1$ in that particular order, and focus on the $(|V| - 1)^{\text{th}}$ iteration. For convenience, we can define $u_0 = u_r$. Our assertion is that there exists a vertex $u_j, 1 \leq j \leq r$, within the cycle, such that $u_j.\texttt{dist} > u_{j-1}.\texttt{dist} + \ell(u_{j-1}, u_j)$.

Suppose for a moment that this is not the case. If so, for every $1 \leq i \leq r$, we'd have $u_i.\texttt{dist} \leq u_{i-1}.\texttt{dist} + \ell(u_{i-1}, u_i)$. Summing both sides of this inequality across $1 \leq i \leq r$ and taking into account that

$$\sum_{i=0}^{r-1} u_i.\texttt{dist} = \sum_{i=1}^{r} u_{i-1}.\texttt{dist} = \sum_{i=1}^{r} u_i.\texttt{dist}$$

we deduce that the sum of the edge lengths within the cycle is non-negative, which leads to a contradiction.

As a result, at the conclusion of the $(|V|-1)^{\text{th}}$ iteration, we can conclude that there exists a value of $j, 1 \leq j \leq r$, for which $u_j.\texttt{dist} > u_{j-1}.\texttt{dist} + \ell(u_{j-1}, u_j)$. In the $|V|^{\text{th}}$ iteration, if $u_j.\texttt{dist}$ is updated before u_{j-1} is encountered, our objective is achieved. Alternatively, it must be updated when u_{j-1} is encountered. Consequently, we can assert that there is indeed a vertex updated during the last iteration, successfully concluding our argument. ■

§ Problem 18

Problem. Consider the following algorithm for finding a minimum weight spanning tree in a connected undirected graph. Initially, let T consist of an arbitrary vertex v and no edges. Then, repeatedly add to T the minimum weighted edge with exactly one vertex in T. (This algorithm, similar to Dijkstra's algorithm, is called Prim's algorithm.)

(a) Based on the blue and red rules discussed in class, show that the algorithm is correct.

(b) Describe an implementation of the algorithm that runs in time $O((m+n)\log n)$. (Assume that you have an implementation of a heap that supports find-min and delete-min in $O(\log s)$ steps, for a heap of s elements, and can build a heap on n elements in $O(n)$ steps.)

Solution. **(a)** Prim's algorithm is written as following :

```
1  X = {} ( edges picked so far)
2  repeat until |X| = |V| − 1:
3      pick a set S ⊂ V for which X has no edges between S and V \ S.
4      e ∈ E be the minimum-weight edege between S and V \ S
5      X = X ∪ {e}
```

Correctness of Algorithm. Since the graph is finite the algorithm will surely terminate. We will prove the correctness by induction. For the base case consider $T_0 = \{v\}$, which is minimum spanning tree of one vertex. Take the cut $\{v\} \sqcup V \setminus \{v\}$, take the minimum weight edge across this cut and colour it blue. If the edge is e_0 it's not hard to see $T_1 = \{v\} \cup e_0$ is minimum spanning tree of 2 vertex.

For the hypothesis part, let's assume T_n be the tree at n-th iteration and let V_n be the vertex set, then T_n is the minimum spanning tree of the spanning subgraph of V_n. Also consider that the edges of T_n is coloured blue. Let's consider the cut of graph $V_n \sqcup (V \setminus V_n)$. Let e be the minimum weight edge between V_n and $V \setminus V_n$, by rule $\{e\}$ must be coloured blue. If $\tilde{e}$ was an edge two vertices are from V_n, $T_n \cup \tilde{e}$ must contain in an cycle, color it red (as it has maximum weight among the edges in the cycle). In this case we still have T_n is the minimum weight spanning tree with n vertices.

If there is an edge $\tilde{e}$ across V_n and $V \setminus V_n$, which must added to T_n such that $T_n \cup \tilde{e}$ is a minimum weight spanning tree on $n+1$ vertices. By blue rule we can say $((T_n \cup \{\tilde{e}\}) \setminus \{\tilde{e}\}) \cup \{e\}$ is the minimum spanning tree on $n+1$ edges. (This proof was similar to the proof of **Blue rule theorem**, which says if any graph G starting with uncolored edges and blue rule, red rue are applied in an arbitrary order, then the final set of edges with blue colour forms a minimum weight spanning tree.) Thus by induction we can say the algorithm is correct. ■

(b) An implementation of the above algorithm is given as following:

```
1   # Input: (G, W) a graph G = (V, E) with W being the set of weights
2   # output: Minimum weight spanning tree T.
3  prim (G, W):
4    for all u ∈ V:
5      cost(u)= ∞
6      prev(u) = NULL
7   arbitrarily choose v ∈ V
8     set cost(v)=0
9   H = makeHeap (V) # accordiing to the heam implementation mentioned
10  if H ≠ ∅ :
```

```
s = deletemin(H)
 for all (s,t) ∈ E :
   cost (t) > w(s,t)
     cost(t) = w(s,t)
     prev(t) = s
   decreasekey(H,t)
```

This code is creating a heap to store the vertices according to the cost function defined by

$$cost(v) = \min_{u \in T} w(u, v), v \notin V$$

At some k-th iteration step it picks a vertex v with minimum cost and an edge which has minimum weight, one vertex of the edge is v. Which means the edge has exactly one vertex in T and it has minimum weight. This code is working according to Prim's algorithm.

TIME COMPLEXITY. If the graph has n vertices and m edges the heap implementation takes $\sim \log n$ time to perform `decreasekey`, `deletemin` and it takes $\sim n$ steps to build the heap. So the total time complexity is given by (this is similar to the case of Dijkstra's algorithm),

$$\begin{aligned} &|V| \times \texttt{deletemin} + (|V| + |E|) \times \texttt{insert} \\ &= n \times \log n + (m + n) \times \log(m + n) \\ &\sim O((m + n) \log n) \end{aligned}$$

§ Problem 19

Problem. Let $(\mathcal{S}, \mathcal{I})$ be a matroid ($\mathcal{S}$ is finite).

(a) Suppose $A, B \in \mathcal{I}$, such that $|A| < |B|$. Show that there is an element $x \in B \backslash A$, such that $A \cup \{x\}$ is independent.

(b) Let I be a maximal independent and let $x \notin I$. Show that there is a unique cycle c contained in $I \cup \{x\}$. (A cycle is a minimal dependent set.)

Solution. **(a)** Let there do not exist any $\{x\} \in B \setminus A$, such that $A \cup \{x\}$ is independent. So for any $C \subseteq B \setminus A$, $A \cup C$ is not independent. This means A is maximal independent subset of $A \cup B$. By definition of **Matroid**, we can say all maximal independent sets of $A \cup B \subseteq \mathcal{S}$ must have same cardinality. Let, M be the maximal independent set of $\mathcal{S}$ containing B, then

$$|M| = |A| \implies |B| \leq |A|$$

This is a contradiction. Thus there is an element $x \in B \backslash A$, such that $A \cup \{x\}$ is independent.

(b) Let, I be a maximal independent set and $x \notin I$, then $I \cup \{x\}$ is dependent set, so there must exist a cycle $c \subseteq I \cup \{x\}$. If there is two distinct cycle c_1 and c_2 contained in $I \cup \{x\}$, clearly the cycles can't be contained in I, so $x \in c_1 \cap c_2$. Let $e \in c_1 \setminus c_2$, see that, $c_1 \setminus \{e\}$ is independent set by minimality of of cycles. Note that, $|c_1| < |I| + 1$ and hence $|c_1 \setminus \{e\}| < |I|$. Py part a we can say there is an element in $I \setminus (c_1 \setminus \{e\})$ such that it's unioun with $c_1 \setminus \{e\}$ is an independent element. By this process we can get an maximal independent element $X \subset I \cup \{x\}$ containing $c_1 \setminus \{e\}$. By the matroid property we can say $|I| = |X|$ as both are maximal independent set of $I \cap x$. From the construction we can see $e \notin X$ and $x \notin I$ thus,

$$X \setminus \{x\} = I \setminus \{e\} \Rightarrow X = (I \cup \{x\}) \setminus \{e\}$$

Since $c_2 \subset I \cup \{x\}$ and $e \notin c_2$ we can say, $c_2 \in (I \cup \{x\}) \setminus \{e\}$ which is independent set. We know subset of any independent set is also independent, thus c_2 is independent set. This is a contradiction. So there is a unique cycle c contained in $I \cup \{x\}$. ■