Maximum Flow

CMSC 27230: Honors Theory of Algorithms

February 13-17, 2023

Corresponding section(s) of Kleinberg-Tardos: 7.1, 7.2, 7.5

1 Overview

The maximum flow problem is a very important problem which is solvable in polynomial time yet as far as we know does not have a greedy algorithm, a divide and conquer algorithm, or an algorithm using dynamic programming. Instead, the following idea is used. Given a solution which is not optimal, we can find a way to improve it. By iteratively improving our solution, we will eventually reach an optimal solution (assuming that this process terminates).

2 The Ford-Fulkerson Algorithm for Maximum Flow

Problem 2.1 (Max Flow)**.** *The maximum flow problem is as follows: Given a directed graph $G = (V, E)$ and a capacity c_e for each edge e, what is the maximum possible flow from s to t?*

We can state this problem more precisely with the following definition:

Definition 2.2. *A flow from s to t on G is a function $f : E(G) \to \mathbb{R}$ such that*

1. *Defining $f_{in}(v)$ to be $f_{in}(v) = \sum_{e=(u,v)\in E(G)} f(e)$ and $f_{out}(v) = \sum_{e=(v,w)\in E(G)} f(e)$, for all $v \in V(G) \setminus \{s, t\}$, $f_{in}(v) = f_{out}(v)$ (flow in = flow out)*
2. *For all $e \in E(G)$, $f(e) \in [0, c_e]$ (no capacity is exceeded)*

We define the value of the flow f to be $f_{out}(s) - f_{in}(s) = f_{in}(t) - f_{out}(t)$ (we will generally have that $f_{in}(s) = f_{out}(t) = 0$ in which case the value of the flow is $f_{out}(s) = f_{in}(t)$).

With this definition, our question can be rephrased as follows. What is the maximum value of a flow f from s to t on G?

Algorithm 2.3 (Ford-Fulkerson Algorithm)**.** *In the Ford-Fulkerson algorithm, we start with zero flow and then iteratively do the following:*

1. *Find a way to route flow from s to t in the residual graph, which describes the remaining capacities for the edges.*

2. *Add this flow to the current flow, increasing the value of the flow from s to t. Update the residual graph accordingly.*

The Ford-Fulkerson algorithm terminates when it is impossible to route any more flow from s to t in the residual graph. As we will show, when this happens we have found both a flow of maximum value and a minimum cut certifying that our flow is maximal.

To state this more precisely, we need a few more definitions:

Definition 2.4. *Given a directed graph G and a flow f, we define the residual graph G_f to be the graph obtained when we apply the following transformation to G. For each edge $e \in E(G)$, we replace the edge $e = (u, v)$ with two edges $e^{\rightarrow} = (u, v)$ and $e^{\leftarrow} = (v, u)$ where $e^{\rightarrow}$ has capacity $c_e - f(e)$ and $e^{\leftarrow}$ has capacity $f(e)$. We think of $e^{\rightarrow}$ as a forward edge representing the amount of flow which can still be routed through e and we think of $e^{\leftarrow}$ as a backwards edge representing the possibiliy of undoing some of the flow which we are currently routing through e.*

Definition 2.5. *Given a graph G and a flow f on G, we define a flow path P from u to v on the residual graph G_f to be a sequence of edges $e'_1, \ldots, e'_l$ such that*

1. *There is a sequence of vertices $v_0, v_1, \ldots, v_{l-1}, v_l$ such that $v_0 = u$, $v_l = v$, and for all $i \in [l]$, e'_i goes from v_{i-1} to v_i. In other words, $e'_1, \ldots, e'_l$ give a path from u to v.*
2. *For all $i \in [l]$, either $e = (v_{i-1}, v_i) \in E(G)$, $e'_i = e^{\rightarrow}$, and $f(e) < c_e$ or $e = (v_i, v_{i-1}) \in E(G)$, $e'_i = e^{\leftarrow}$, and $f(e) > 0$*

Note: The length of this path l can be arbitrary.

We define the capacity of P to be

$$c_P = \min\{\min_{i\in[l]:e=(v_{i-1},v_i)\in E(G),e'_i=e^{\rightarrow}}\{c_e - f(e)\}, \min_{i\in[l]:e=(v_i,v_{i-1})\in E(G),e'_i=e^{\leftarrow}}\{f(e)\}\}$$

In other words, the capacity c_P of P is the maximum amount of flow which can be routed along P, which is the minimum among the capacities of the forward edges $e^{\rightarrow}$ which P goes forward through and the capacities of backwards edges $e^{\leftarrow}$ which P goes backwards through.

Definition 2.6. *Given a flow path P from s to t, we define the flow f_P so that*

1. *For all $i \in [l]$ such that $e = (v_{i-1}, v_i) \in E(G)$ and $e'_i = e^{\rightarrow}$, $f_P(e) = c_P$*
2. *For all $i \in [l]$ such that $e = (v_i, v_{i-1}) \in E(G)$ and $e'_i = e^{\leftarrow}$, $f_P(e) = -c_P$.*
3. *For all other edges, $f_P(e) = 0$.*

In other words, f_P is the flow corresponding to routing c_P units of flow along P.

Proposition 2.7. *Given a flow f on G from s to t, if P is a flow path from s to t on the residual graph G_f then $f' = f + f_P$ is a flow on G from s to t and $value(f') = value(f) + c_P$.*

Proof. We make the following observations:

1. For all $v \in V(G) \setminus \{s, t\}$, $f_{in}(v) = f_{out}(v)$ and $(f_P)_{in}(v) = (f_P)_{out}(v)$ so $f'_{in}(v) = f'_{out}(v)$.

2. $value(f') = f'_{out}(s) - f'_{in}(s) = (f_P)_{out}(s) + f_{out}(s) - f_{in}(s) = c_P + value(f)$.

3. For all $i \in [l]$ such that $e = (v_{i-1}, v_i) \in E(G)$ and $e'_i = e^{\rightarrow}$, $f_P(e) \leq c_e - f(e)$ so $f'(e) = f(e) + f_P(e) \leq c_e$.

4. For all $i \in [l]$ such that $e = (v_i, v_{i-1}) \in E(G)$ and $e'_i = e^{\leftarrow}$, $f_P(e) = -c_P \geq -f(e)$ so $f'(e) = f_P(e) + f(e) \geq 0$.

□

Algorithm 2.8 (Ford-Fulkerson restated)**.** *In the Ford-Fulkerson algorithm, we do the following:*

Stored data: We keep track of a flow f and the residual graph G_f.

Initialization: We start with $f = 0$ and $G_f = G$

Iterative step: At each step, we find a flow path P from s to t in G_f. We then take the flow $f' = f + f_P$ and take the residual graph $G_{f'}$. If there is no flow path P from s to t on G_f then we terminate as f is a maximum flow from s to t.

3 Analyzing Ford-Fulkerson

We now analyze the Ford-Fulkerson algorithm.

Theorem 3.1. *Given a directed graph G where all of the capacity ranges $[a_e, b_e]$ have integer values (i.e. $a_e, b_e \in \mathbb{N} \cup \{0\}$), if F is the value of the maximum flow from s to t in G then the Ford-Fulkerson algorithm finds a maximum flow from s to t and terminates in time $O(F|E(G)|)$.*

Warning: If the capacities are arbitrary real numbers then there is no guarantee that the Ford-Fulkerson algorithm will terminate.

Proof. A key idea for analyzing the Ford-Fulkerson algorithm is to look at cuts C separating s from t.

Definition 3.2. *We define a cut separating s and t to be a partition $C = (L, R)$ of the vertices $V(G)$ such that $s \in L$ and $t \in R$. We define the capacity of a cut $C = (L, R)$ to be*

$$capacity(C) = \sum_{e=(u,v) \in E(G): u \in L, v \in R} c_e$$

As we now show, $capacity(C)$ is the largest amount of flow which can cross C.

Definition 3.3. *If f is a flow from s to t on G and C is a cut separating s and t then we define $f_{across}(C)$ to be*

$$f_{across(C)} = \sum_{(u,v) \in E(G): u \in L, v \in R} f(e) - \sum_{(u,v) \in E(G): v \in L, u \in R} f(e)$$

Lemma 3.4. *If f is a flow from s to t on G and C is a cut separating s and t then $f_{across}(C) = f_{out}(s) - f_{in}(s)$*

Proof. Consider the expression $\sum_{v \in L} f_{out(v)} - f_{in}(v)$. On the one hand, $f_{out(v)} = f_{in}(v)$ for all $v \in V(G) \setminus \{s, t\}$ so $\sum_{v \in L} f_{out(v)} - f_{in}(v) = f_{out(s)} - f_{in}(s) = value(f)$ On the other hand, looking at the number of times $f(e)$ appears in this expression for each edge $e \in E(G)$ we have that

$$\begin{aligned}\sum_{v \in L} f_{out(v)} - f_{in}(v) &= \sum_{v \in L} \left(\sum_{e=(v,w) \in E(G)} f(e) - \sum_{e=(u,v) \in E(G)} f(e) \right) \\ &= \sum_{e=(u,v) \in E(G)} (1_{u \in L} - 1_{v \in L}) = \sum_{(u,v) \in E(G): u \in L, v \in R} f(e) - \sum_{(u,v) \in E(G): v \in L, u \in R} f(e) \\ &= f_{across}(g)\end{aligned}$$

where $1_{v \in L}$ is 1 if $v \in L$ and is 0 if $v \in R$. Thus, $f_{across}(C) = f_{out}(s) - f_{in}(s)$, as needed. □

Corollary 3.5. *If f is a flow from s to t on G and C is a cut separating s and t then $value(f) \leq capacity(C)$*

Proof. Since we have that $f(e) \in [a_e, b_e]$ for every edge $e \in E(G)$,

$$\begin{aligned}value(f) = f_{across(C)} &= \sum_{(u,v) \in E(G): u \in L, v \in R} f(e) - \sum_{(u,v) \in E(G): v \in L, u \in R} f(e) \\ &\leq \sum_{e=(u,v) \in E(G): u \in L, v \in R} c_e = capacity(C)\end{aligned}$$

□

This corollary shows that every cut C separating s and t gives an upper bound on the maximum value of a flow from s to t. To prove that when Ford-Fulkerson terminates, it gives a maximum flow f, we show that there is a cut C such that $value(f) = capacity(C)$. Since we know that for any flow f', $value(f') \leq value(C)$, this certifies that f is a maximum flow.

To do this, we take $C = (L, R)$ where $L = \{v : \text{ there is a flow path } P \text{ from } s \text{ to } v \text{ in } G_f\}$ and $R = V(G) \setminus L$. We now make the following observations:

1. We cannot have $t \in L$ as otherwise there would be a flow path P from s to t in G_f and the Ford-Fulkerson algorithm would not have terminated.

2. For any edge $e = (u, v) \in E(G)$ where $u \in L$ and $v \in R$, we must have that $f(e) = c_e$. To see this, assume that $u \in L$, $v \in R$, and $f(e) < c_e$. Observe that since $u \in L$ there is a flow path P from s to u in G_f. Since $f(e) < c_e$, we can add the edge $e^{\rightarrow} = (u, v)$ to P and this gives us a flow path P' from s to v in G_f. But then by definition $v \in L$, which is a contradiction.

3. For any edge $e = (v, u) \in E(G)$ where $u \in L$ and $v \in R$, $f(e) = 0$. To see this, assume that $u \in L$, $v \in R$, and $f(e) > 0$. Observe that since $u \in L$ there is a flow path P from s to u in G_f. Since $f(e) > 0$, we can add the edge $e^{\leftarrow} = (u, v)$ to P and this gives us a flow path P' from s to v in G_f. But then by definition $v \in L$, which is a contradiction.

Putting everything together, for this cut $C = (L, R)$,

$$value(f) = f_{across}(C) = \sum_{e=(u,v)\in E(G):u\in L,v\in R} c_e - \sum_{e=(u,v)\in E(G):v\in L,u\in R} 0 = capacity(C)$$

Thus, if the Ford-Fulkerson algorithm terminates then it gives a maximum flow f. To see that the Ford-Fulkerson algorithm must terminate if all of the capacities are integers, note that each time a flow path is found in G_f, this increases the value of the flow by 1. Thus, if F is the value of the maximum flow from s to t, there can be at most F iterations, each of which takes $O(|E(G)|)$ time for a total runtime of $O(F|E(G)|)$. □

Corollary 3.6 (Max flow/Min Cut Theorem for Integer Capacities)**.** *If G has integer capacities then the maximum flow from s to t is equal to the minimum capacity of a cut C separating s and t.*

Remark 3.7. *The max flow/min cut theorem is true for non-integer capacities as well, though proving it requires either modifying this argument or using a different argument such as linear programming duality.*

4 Maximum Matching in Bipartite Graphs

Problem 4.1. *In the maximum matching problem on bipartite graphs, we are given a bipartite graph G with vertices $A \cup B$ and edges $E(G) \subseteq \{\{u, v\} : u \in A, v \in B\}$. We are then asked to find the largest matching M of G, i.e. a set $M \subseteq E(G)$ of edges such that no two edges in M are incident to the same vertex.*

We can reduce maximum matching on bipartite graphs to max flow as follows. Given G, we create a graph G' where

1. $V(G') = V(G) \cup \{s, t\}$ (we add a source and a sink to G)
2. $E(G') = \{(u, v) : \{u, v\} \in E(G), u \in A, v \in B\} \cup \{(s, u) : u \in A\} \cup \{(v, t) : v \in A\}$ (we make each edge in G point from A to B, we add edges from the source s to each vertex $u \in A$, and we add edges from each vertex $v \in B$ to the sink.)
3. Every edge $e \in E(G')$ has capacity 1 (i.e. its capacity range is $[0, 1]$)

Proposition 4.2. *There is a bijection between integer valued flows $f : E(G') \to \{0, 1\}$ from s to t on G' with value k and matchings M in G with k edges.*

Proof. We take the following maps between integer valued flows $f : E(G') \to \{0, 1\}$ from s to t on G' with value k and matchings M in G with k edges.

1. Given a matching $M = \{\{u_i, v_i\} : i \in [k]\}$, we take the flow f where $f(e) = 1$ for the edges $\{(s, u_i) : i \in [k]\} \cup \{(u_i, v_i) : i \in [k]\} \cup \{(v_i, t) : i \in [k]\}$ and $f(e) = 0$ for all other edges in $E(G')$.

2. Given an integer valued flow $f : E(G') \to \{0, 1\}$ from s to t on G' with value k, let $M = \{\{u, v\} : u \in A, v \in B, f((u, v)) = 1\}$. Note that no two edges in M can be incident with the same vertex $u \in A$ as this would require routing at least two units of flow from s to u and this exceeds the capacity of the edge from s to u. Similarly, no two edges in M can be incident with the same vertex $v \in B$. Finally, note that if we take $C = (A \cup \{s\}, B \cup \{t\})$ then $|M| = f_{across}(C) = value(f) = k$ so M is a matching of size k, as needed.

□

It is not hard to verify that these maps are inverses of each other and thus give a bijection between integer valued flows $f : E(G') \to \{0, 1\}$ from s to t on G' with value k and matchings M in G with k edges. Thus, by solving max flow on G', we can solve the maximum matching problem on our original input graph G.

In fact, the reduction from maximum matching on bipartite graphs to max flow allows us to easily prove König's Theorem.

Definition 4.3. *We say that a set of vertices $V \subseteq V(G)$ is a vertex cover if for every edge $(u, v) \in E(G)$, either $u \in V$ or $v \in V$.*

Theorem 4.4 (König's Theorem)**.** *If G is a bipartite graph then the maximum size of a matching in G is equal to the minimum size of a vertex cover V of G.*

Proof. We first show that the maximum size of a matching M in G is at most the minimum size of a vertex cover V of G. To see this, observe that each vertex in V can be used in at most one edge in M, which immediately implies that $|M| \leq |V|$.

On the other hand, letting k be the maximum size of a matching in G and letting G' be the result of applying the above reduction to G, the maximum flow from s to t in G' has value k. By the max flow/min cut theorem, there exists a minimum cut $C = (L, R)$ separating s and t in G' which cuts k edges. Choose C so that $|A \cap R| + |B \cap L|$ is maximized. We now make the following claims

1. There is no edge $(u, v) \in E(G)$ such that $u \in A \cap L$ and $v \in B \cap R$.
2. $(A \cap R) \cup (B \cap L)$ is a vertex cover of G
3. $C = (L, R)$ cuts exactly $|(A \cap R) \cup (B \cap L)|$ edges

To see the first claim, assume that we have an edge $e = (u, v) \in E(G)$ such that $u \in A \cap L$ and $v \in B \cap R$. If so consider $C' = (A \setminus \{u\}, B \cup \{u\}$. $capacity(C') \leq capacity(C)$ because C' no longer cuts the edge $e = (u, v)$ and the only additional edge C' cuts is (s, u). Since we chose C to be a minimum cut, we must have that $capacity(C') = capacity(C) = k$. However, C' has a larger value of $|A \cap R| + |B \cap L|$, which contradicts our choice of C.

The second claim immediately follows from the first claim as for any edge $e = (u, v) \in E(G)$, either $u \in A \cap R$ or $v \in B \cap L$.

To see the third claim, observe that the set of edges cut by C is

$$\{(s, u) : u \in A \cap R\} \cup \{(u, v) : u \in A \cap L, v \in B \cap R\} \cup \{(v, t) : v \in B \cap L\}$$

Since $\{(u, v) : u \in A \cap L, v \in B \cap R\} = \emptyset$, C cuts exactly $|(A \cap R) \cup (B \cap L)|$ edges so $|(A \cap R) \cup (B \cap L)| = k$. Thus, $(A \cap R) \cup (B \cap L)$ is a vertex cover of G of size k. □

We now discuss how the Ford-Fulkerson algorithm applies to the maximum matching problem on bipartite graphs. In particular, a flow path from s to t corresponds to an augmenting path.

Definition 4.5. *Given a graph G and a matching M on G, we say that a path $P = v_0 \to v_1 \to \ldots \to v_l$ in G is an augmenting path if*

1. *$\forall i \in [l], (v_{i-1}, v_i) \in E(G)$ (all edges in the path are in G).*
2. *l is odd.*
3. *$\forall i \in [l] : i$ is even., $(v_{i-1}, v_i) \in M$ and $\forall i \in [l] : i$ is odd., $(v_{i-1}, v_i) \notin M$ (P alternates between edges which are not in M and edges which are in M)*
4. *v_0 and v_l are unmatched in M.*

Proposition 4.6. *If there is an augmenting path then we can increase the size of M by 1 by replacing the edges $\{(v_{i-1}, v_i) : i \in [l], i$ is even$\}$ with the edges $\{(v_{i-1}, v_i) : i \in [l], i$ is odd$\}$.*

Lemma 4.7. *If M is not a maximum matching of G then there exists an augmenting path.*

Proof. If M is not a maximum matching of G, let M' be a larger matching. Consider the multigraph G' with vertices $V(G)$ and edges $M \cup M'$.

Observe that since every vertex in G' has degree 0, 1, or 2, G' consists of cycles and paths. Letting O be the set of vertices of G' which have degree 1, for each vertex $u \in O$ there must be a vertex $v \in O$ such that u and v are the endpoints of a path in G. Now observe that since $|M'| > |M|$, there are more vertices in O which are matched in M' but not in M than there are vertices in O which are matched in M but not in M'. Thus, there must be a pair of vertices which are matched in M' but not in M and which are the endpoints of a path P in G'. This path P is an augmenting path, as needed. □

Remark 4.8. *Note that this lemma holds for all graphs, not just bipartite graphs. This is very useful because it means that if we currently have a matching, instead of searching for a better matching from scratch, we just need to find an augmenting path. Jack Edmond's blossom algorithm takes advantage of this to solve maximum matching in polynomial time on all graphs, not just bipartite graphs.*

5 Menger's Theorem

In this section, we consider the maximum number of disjoint paths from s to t. There are two variations we can consider.

1. What is the maximum number of edge-disjoint paths from s to t?
2. What is the maximum number of vertex-disjoint paths from s to t?

We can reduce the first variant to max flow by simply making each edge have capacity 1. If we have k edge-disjoint paths from s to t then routing one unit of flow through each path gives a flow from s to t with value k. To show that if there is a flow from s to t of value k then there are k edge-disjoint paths from s to t, we observe that since all capacities are integers, there is an integer-valued max flow. We then use the following lemma.

Lemma 5.1. *Let G be a directed graph where all edges have capacity 1. If there is an integer-valued flow f with value k then there are k edge-disjoint paths from s to t.*

Proof. We can find a walk from s to t as follows.

1. We start at s.
2. Whenever we are at a vertex $v \in V(G) \setminus t$, we choose an arbitrary edge $e \in E(G)$ with flow 1 that we have not yet chosen and travel along this edge.
3. We stop when we reach t.

To see why this works, observe that since $f_{out}(s) > f_{in}(s)$ and $f_{in}(v) = f_{out}(v)$ for all $v \in V(G) \setminus \{s, t\}$, whenever we reach a vertex $v \in V(G) \setminus \{t\}$ there must be a way to leave v so we can keep on going until we reach t.

Once we find a walk from s to t, we delete all the edges from this walk. After deleting the edges from this walk, we have a flow from s to t whose value is reduced by 1, so we can repeat this process until we have k edge-disjoint walks from s to t. Deleting cycles from these walks gives us k edge-disjoint paths from s to t, as needed. □

We now consider the problem of finding the maximum number of vertex-disjoint paths from s to t. This problem is a bit trickier but we can again reduce it to max flow by using a trick.

1. We split each vertex $v \in V(G) \setminus \{s, t\}$ into two vertices v_{in} and v_{out}. We then create an edge from v_{in} to v_{out} with capacity 1
2. For each edge $e = (u, v) \in E(G)$, we replace it with an edge from u_{out} to v_{in} with capacity 1 (where we take $s_{out} = s$ and $t_{in} = t$).

To see why this reduction is correct, we make the following observations. Let G' be the resulting graph after we apply this reduction to G.

1. Each path $P = s \to v_1 \to \ldots \to v_{l-1} \to t$ becomes a path

   $$P' = s \to (v_1)_{in} \to (v_1)_{out} \to (v_2)_{in} \to (v_2)_{out} \to \ldots \to (v_{l-1})_{in} \to (v_{l-1})_{out} \to t$$

   in G'. Thus, if we have k vertex-disjoint paths from s to t in G then we will have k vertex-disjoint paths from s to t in G'. Routing one unit of flow through each path gives a flow of value k, as needed.

2. If there is a flow of value k from s to t in G' then since all of the capacities are integers, there must be an integer-valued flow with value k. By Lemma 5.1, there are k edge-disjoint paths from s to t in G'. Now observe that whenever a path P' enters a vertex v_{in}, it must immediately continue to v_{out}. This implies that any path P' from s to t must be of the form

   $$P' = s \to (v_1)_{in} \to (v_1)_{out} \to (v_2)_{in} \to (v_2)_{out} \to \ldots \to (v_{l-1})_{in} \to (v_{l-1})_{out} \to t$$

   for some vertices $v_1, \ldots, v_{l-1} \in V(G) \setminus \{s, t\}$ so this path corresponds to a path $P = s \to v_1 \to \ldots \to v_{l-1} \to t$ from s to t in G. Moreover, since the edge from v_{in} to v_{out} has capacity 1, we cannot have two edge-disjoint paths from s to t in G' visit v_{in} or v_{out}. This implies that these k paths from s to t in G' correspond to k vertex-disjoint paths from s to t in G, as needed.

We now state and prove Menger's Theorem.

Definition 5.2. *We say that S is a vertex separator separating s and t if $S \subseteq V(G) \setminus \{s, t\}$ and every path from s to t has a vertex in S.*

Theorem 5.3 (Menger's Theorem)**.** *The number of vertex disjoint paths from s to t is equal to the minimum size of a vertex separator S separating s and t*

Proof. It is not hard to show that if S is a vertex separator separating s and t then there are at most $|S|$ vertex disjoint paths from s to t as each vertex in S can only be used by one path.

For the other direction, let k be the maximum number of vertex-disjoint paths from s to t in G. Applying the above reduction and letting G' be the resulting graph, the maximum flow from s to t in G' has value k. By the max flow/min cut theorem, the minimum capacity of a cut $C = (L, R)$ of G' where $s \in L$ and $t \in R$ is k. Let C be one such cut.

We now observe that we can choose C so that only edges of the form (v_{in}, v_{out}) are cut. To see this, observe that if C cuts some other edge (u_{out}, v_{in}) (i.e. $u_{out} \in L$ and $v_{in} \in R$) then we may as well shift v_{in} to L.

We now take $S = \{v : v_{in} \in L, v_{out} \in R\}$. We have that $|S| = capacity(C) = k$. To show that S is a vertex separator separating s and t, observe that for any path P from s to t, C must cut some edge in the corresponding path P' and this edge must be of the form (v_{in}, v_{out}) where $v \in V(P)$. This implies that $v \in S$ so P contains a vertex in S, as needed.

□