Mapping Reducibility

Last Updated: August 17th, 2026

1 Introduction

The idea of reducing a computing problem, data format, or programming language to some other problem, format, or language pervades all of computer science. Oftentimes, such a reduction is called a mapping, translation, or conversion. For example, to run a computer program on a laptop, the program must be tranlated into a set of machine instructions that are native to the laptop's processor. This translation is special in the sense that the machine code has the same fumctionality as the original program. Another example, is converting a word document into a pdf document. This conversion is special because pdf document preserves the text and graphical information provided by the author of the word document.

In this lecture we study a kind of reducibility that maps instances of one computing problem to instances of another. What makes the mapping special is that it preserves the original computing problem's solution in that the problem instance that it maps to has the same solution.

Definition 1.1. Problem A is **mapping reducible** to problem B, written $A \leq_m B$, iff there exists a computable function $f : A \to B$ for which the solution to problem instance x of A is equal to the solution to problem instance $f(x)$ of B.

$x \in A$

$f : A \to B$ $\quad f(x) \in B$

name, domain, co-domain

Notes:

1. The purpose of mapping reduction $f : A \to B$ is *not* to solve an instance x of problem A. Rather it is to *translate* x into the instance $f(x)$ of B, where the solution to $f(x)$ (whatever it may be) is equal to the solution to x.

2. A special case of a mapping reduction occurs when A and B are decision problems. In this case f has the property that x is a positive instance of A iff $f(x)$ is a positive instance of B. In other words, a positive (respectively, negative) instance of A must map to a positive (respectively, negative) instance of B.

3. If we insist that the algorithm for computing f requires at most a polynomial number of steps with respect to the size of input instance x, then we say A is **polynomial-time mapping reducible** to B and write $A \leq_m^p B$.

4. The term *map* is a synonym for *function*.

2 Basic Examples

Example 2.1. We begin with a toy example by considering the two decision problems `Even` and `Odd` where an instance of either problem is an integer n. Moreover, for `Even` the problem is to decide if n is even. Problem `Odd` is similarly defined. Then we have `Even` $\leq_m$ `Odd` via function $f(n) = n + 1$. This is true since n is even if and only if $f(n) = n + 1$ is odd. Thus, the answer to n equals the answer to $f(n)$. Notice that f also provides a reduction from `Odd` to `Even`.

Case 1: n is even. n is a positive instance of Even. Then $n = 2k$ $f(n) = n+1 = 2k+1$ is odd. which is a positive instance of odd.

case 2: n is odd. n is a negative instance of Even. $n = 2k+1$ for some $k \in I$. $f(n) = n+1 = (2k+1)+1 = 2k+2$ $= 2(k+1)$ is even and is a negative instance of Odd.

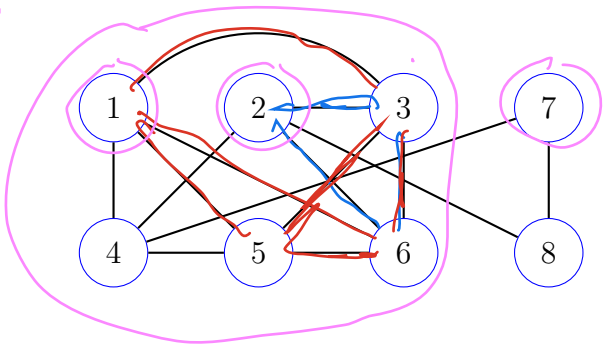
$\therefore$ f: Even $\rightarrow$ Odd is a valid mapping reduction.

Definition 2.2. A simple graph $G = (V, E)$ is an undirected graph for which i) there are no self-loops and ii) any two vertices have at most one edge that is incident with them.

Definition 2.3. Let $G = (V, E)$ be a simple graph.

1. A set of vertices $C \subseteq V$ is called a **clique** iff, for every $u, v \in C$, $(u, v) \in E$. In words, every pair of vertices that belong to C is joined by an edge.
2. A set of vertices $I \subseteq V$ is called **independent** iff, for every $u, v \in I$, $(u, v) \notin E$. In words, every pair of vertices that belongs to I is *not* joined by an edge.
3. For $k \geq 1$, a **k-clique** (respectively, **k-independent set**) is a clique (respectively, independent set) of size k.

Example 2.4. What is the largest k for which the graph below has a k-clique (respectively, k-independent set)? Provide the clique (respectively, independent set).



{7,8} is a 2-clique

{2,3,6} is a 3-clique

{1,3,5,6} is a 4-clique

{1,2,7} is an independent set

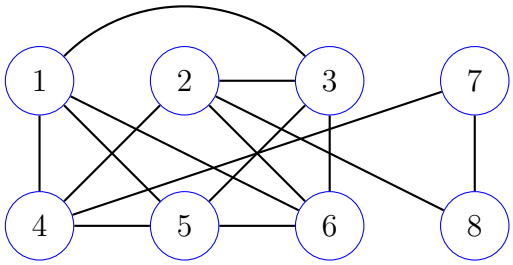
{3,8,4}

Definition 2.5. Given a simple graph $G = (V, E)$, the **complement** of G, denoted $\overline{G}$, is defined as $\overline{G} = (V, \overline{E})$, where, for all $u, v \in V$,

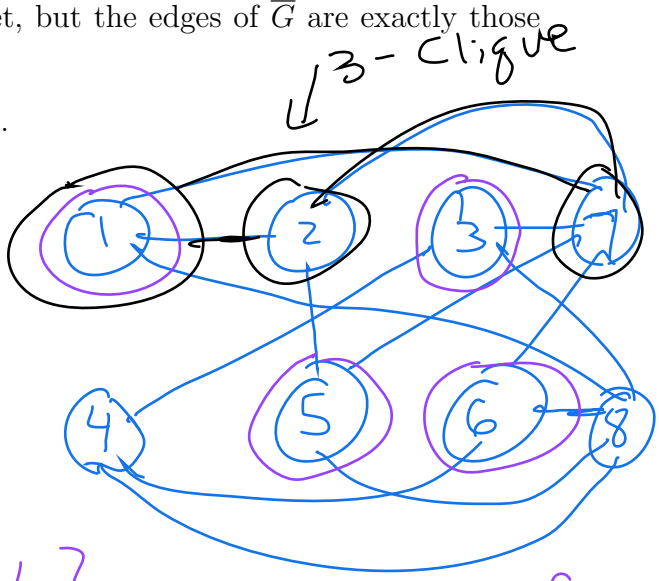
$$(u, v) \in \overline{E} \text{ iff } (u, v) \notin E.$$

In words, G and its complement $\overline{G}$ have the same vertex set, but the edges of $\overline{G}$ are exactly those edges that are *not* edges of G, and vice versa.

Example 2.6. Given the graph G below, draw $\overline{G}$ next to it.



$\overline{G}$:



$\{1, 3, 5, 6\}$ is a 4-IS for $\overline{G}$

Definition 2.7. An instance of the `Max Clique` optimization problem is a simple graph G, and the problem is to determine the size of the largest clique in G. Similarly, an instance of the `Max Independent Set` optimization problem is a simple graph G, and the problem is to determine the size of the largest independent set in G.

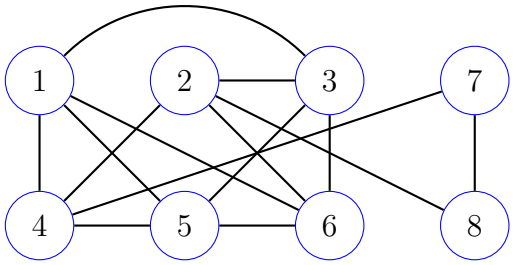
Proposition 2.8. Consider the map f : Simple Graph $\to$ Simple Graph, defined by $f(G) = \overline{G}$. Then

$\overline{\overline{G}} = G$

1. f is a valid mapping reduction from `Max Clique` to `Max Independent Set` and
2. f is a valid mapping reduction from `Max Independent Set` to `Max Clique`.

Proof of Proposition. We prove statement 1. Statement 2 is similar. Given input graph G, suppose that G has a maximum clique C of size $k \geq 1$. Then all pairs of vertices in C are adjacent. Thus, by definition of $\overline{G}$, all pairs of vertices in C must be *non-adjacent* in $\overline{G}$. Thus, C is a k-independent set for $\overline{G}$. Conversely, suppose that $\overline{G}$ has a maximum independent set I of size $l \geq 1$. Then all pairs of vertices in I are non-adjacent. Thus, by definition of $\overline{G}$, all pairs of vertices in I must be *adjacent* in G. Thus, I is an l-clique for G. Therefore, we must conclude that $k = l$ and the reduction is valid. □

Example 2.9. Verify Proposition 2.8 for the graph G in Example 2.6.



3 Embeddings

Definition 3.1. An **embedding reduction** is a kind of mapping reduction for which a problem A is map reduced to problem B, where A is a special case of B.

As an example, consider a vehicle that has an air-conditioning system that includes the ability to cool and heat different parts of the cabin, as well as control the flow of air entering and leaving the cabin. Recall our friend Sam from the Turing Reducibility lecture. Sam and his friends drove to the Mohave desert where they encountered extreme heat, lots of wind, and some very dusty roads. While driving, they reduced the problem of `cool and clean air` to the more general problem of `air conditioning`. Moreover, since `cool and clean air` is a special case of the `air conditioning` problem, the reduction simply involved realizing that `cool and clean air` is embedded into the air-conditioning system and can be activated by inputting the correct settings.

Definition 3.2. The `Subset Sum (SS)` decision problem is a pair (S, t), where S is a set of nonnegative integers, and t is a nonnegative integer. The problem is to decide if there is a subset $A \subseteq S$ whose members sum to t, i.e., a subset A for which

$$\sum_{a \in A} a = t.$$

Example 3.3. `Subset Sum` instance $(S = \{3, 7, 13, 19, 22, 26, 35, 38, 41\}, t = 102)$ is a positive instance of `SS` since $A = \{3, 7, 13, 38, 41\} \subseteq S$ and

$$3 + 7 + 13 + 38 + 41 = 102.$$

log M is the size parameter for measuring the size

$M = 2^{\log M}$ is exponential in M

$M = 2^{100}$

Definition 3.4. An instance of decision problem `Set Partition (SP)` consists of a set S of positive integers, and the problem is to decide if there are subsets $A, B \subseteq S$ for which

1. $A \cap B = \emptyset$,
2. $A \cup B = S$, and
3. $$\sum_{a \in A} a = \sum_{b \in B} b.$$

In other words, the members of A must sum to the same value as the members of B.

Example 3.5. Show that `Set Partition` instance

$$S = \{3, 14, 19, 26, 35, 37, 43, 49, 52\}$$

is a positive instance of `SP`.

$M = \sum_{s \in S} s = 278$

$\frac{M}{2} = 139$

A few moments of thought should convince you that SP is a special case of SS. Indeed given instance S of SP, if we let M denote the sum of all members of S, then we essentially are seeking a subset A whose members sum to $M/2$ since, if such A can be found, then by setting $B = S - A$, we have all three conditions met (check this!). Therefore, we may reduce SP to SS via the map

$$f(S) = (S, t = M/2), \qquad M = \sum_{s \in S} s \tag{1}$$

where M is defined as above.

Just as Sam and his friends don't care about the heating features of the air-conditioning, when solving an instance of `Set Partition` via `Subset Sum`, we don't care that `Subset Sum` can solve a wider variety of problems involving t values that may not have any relationship with S.

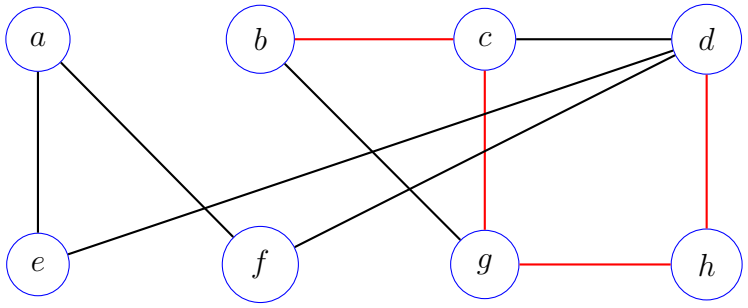
Example 3.6. Given the instance $S = \{4, 8, 17, 25, 34, 46, 53, 59\}$ of `Set Partition`, provide the instance of `SS` to which it reduces via the mapping defined in equation 1.

Solution. $f(S) = (S, \frac{M}{2}) = (S, t = 123)$

where $M = \frac{246}{2} =$

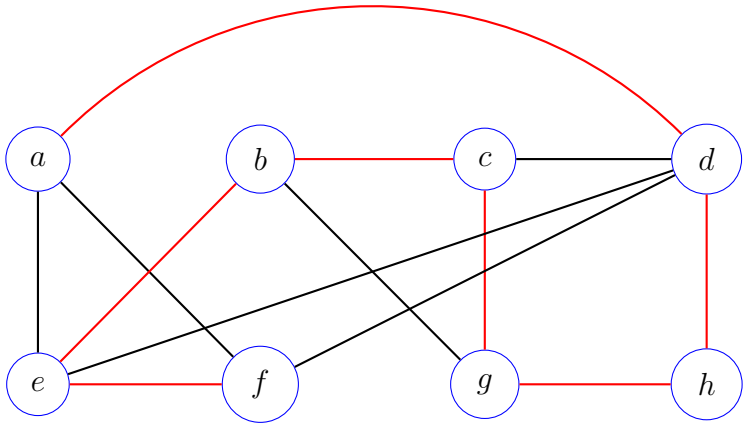
Definition 3.7. An instance of the `LPath` decision problem is a pair (G, k), where $G = (V, E)$ is a simple graph, and $k \geq 0$ is a nonnegative integer. The problem is to decide if G has a **simple path** of length k, i.e. a path that traverses k edges and visits exactly $k + 1$ different vertices.

Example 3.8. The following graph, along with $k = 4$, shows a positive instance of LPath.



Definition 3.9. An instance of the `Hamilton Path (HP)` decision problem consists of a simple graph $G = (V, E)$ and the problem is to decide if G has a **Hamilton path**, namely a path that visits every vertex in G exactly once.

Example 3.10. The following graph represents a positive instance of `HP`, with the Hamilton Path shown in red.



Once again, it is hopefully evident that `HP` is a special case of `LPath`. To see this, note that a simple graph has a Hamilton path iff it has a simple path of length $n - 1 = |V| - 1$. Therefore, we may reduce `HP` to `LPath` via the map

$$f(G) = (G, |V| - 1).$$

□

4 Contractions

As the examples in the previous section suggest, devising an embedding reduction from problem A only requires knowledge of a problem B that is more general than A and includes A as a special case. On the other hand, a **contraction reduction** is a mapping reduction for which problem B is the special case of problem A. Defining a contraction usually involves some insight and imagination. Moreover, a contraction reduction demonstrates actual computing progress in the sense that a body A of problem instances gets effectively reduced to a smaller set B which may improve the chances of efficiently solving A through the lens of B.

Example 4.1. Contractions are quite common in everyday computing. As an example, consider a legacy compiler $\mathcal{C}$ for some programming language L. Over the years L gets extended with the addition of new programming constructs that improve the ease of programming in L. We'll call this extended version Turbo-L. Rather than write a new compiler for Turbo-L, we instead provide a contraction that is capable of translating any Turbo-L program to an L program, followed by compiling the L-code with the legacy compiler. □

We now provide a contraction reduction f : `Subset Sum` $\rightarrow$ `Set Partition` from `Subset Sum` to `Set Partition`. Let (S, t) be an instance of `SS`. Let

$$M = \sum_{s \in S} s$$

denote the sum of all members of S.

Case 1. $t = M/2$. Then we seek a subset $A \subseteq S$ whose members sum to $t = M/2$, which means that the complement $B = S - A$ must also sum to $t = M/2$, since $S = A \cup B$. Thus, (S, t) is essentially a `Set Partition` problem instance, and so $f(S, t) = S' = S$.

Case 2. $t < M/2$. In this case we seek an $A \subseteq S$ whose members sum to a value that is *less than* one half of the sum of the members of S, which means that (S, t) is *not* identical to a `Set Partition` problem instance since the members of A will sum to a value that is less than the sum of $S - A$, i.e.

$$t < M - t.$$

Moreover, our strategy is to add another member J to S, which gives a new set $f(S, t) = S' = S \cup J$. What should the value of J equal? We need the equation

$$t + J = M - t$$

to be true, which indeed will be the case assuming $J = M - 2t$.

Now let's check our logic to make sure this is a valid mapping reduction.

Assume (S, t) is a positive instance of `SS`. Then there is a subset $A \subseteq S$ which sums to t. Moreover, $f(S, t) = S' = S \cup \{J\}$ is a positive instance of `SP` since both $A \cup \{J\}$ and $B = S' - (A \cup \{J\})$ sum to $M - t$. Indeed,

$$t + J = t + (M - 2t) = M - t$$

which equals the sum of the members of $S - A = S' - (A \cup \{J\})$.

Conversely, assume S' is a positive instance of `SP`. That means there are sets A' and B' that both sum to

$$\frac{\sum_{s \in S'} s}{2} = (M + J)/2 = (M + (M - 2t))/2 = M - t.$$

Without loss of generality, assume that A' is the set that contains J, then the members of $A' - \{J\}$ sum to

$$M - t - J = M - t - (M - 2t) = t.$$

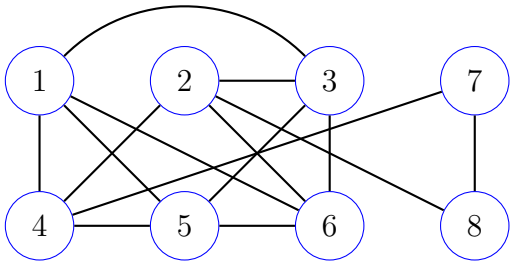
Finally, notice that the members of $A' - \{J\}$ all belong to the original set S. Therefore, S has a subset that sums to t, and (S, t) is a positive instance of `SS`.

Example 4.2. Derive the formula for J in case $t > M/2$.

Example 4.3. Use the above reduction to map the `Subset Sum` instance $(\{7, 11, 23, 25, 37, 39, 49, 73\}, t = 79)$ to and instance of `Set Partition`. Verify that the reduction is valid by showing that either both are positive instance or both are negative instances. Hint: 264

Definition 4.4. The `Vertex Cover (VC)` decision problem is the problem of deciding if a simple graph $G = (V, E)$ has a vertex cover of size $k \geq 0$, for some integer k. In other words does G have a subset C of k vertices for which every edge $e \in E$ is incident with at least one vertex in C?

Example 4.5. Show that $(G, k = 5)$ is a positive instance of `VC`, where G is shown below.



Solution.

Definition 4.6. The `Half Vertex Cover (HVC)` decision problem is the problem of deciding if a simple graph $G = (V, E)$ has a vertex cover of size equal to $|V|/2$. Note that if G has an odd number of vertices then it is a negative instance of `HVC`.

We now provide a contraction mapping $f : \texttt{VC} \to \texttt{HVC}$ from `Vertex Cover` to `Half Vertex Cover`. Let $(G = (V, E), k)$ be an instance of `VC`. Let $n = |V|$.

Case 1. $k = n/2$. In this case instance (G, k) is essentially an `HVC` instance since the problem is to determine if G has a cover of size equal to $n/2$, i.e. a half cover. Thus $f(G, k) = G$.

Case 2. $k > n/2$. In this case we may map (G, k) to an instance of `HVC` by adding J additional isolated vertices to G, where

$$k = \frac{1}{2}(n + J).$$

From this equation we see that $J = 2k - n$.

To see that this is a valid reduction, suppose that (G, k) is a positive instance of `VC`. Then $f(G, k)$ equals G with $J = 2k - n$ isolated vertices added. Since no additional edges are added, the k cover for G remains a k cover for $f(G, k)$. But it is also a half cover for $f(G, k)$ because k is now one half the number of vertices of $f(G, k)$. Thus, $f(G, k)$ is a positive instance of `HVC`.

Conversely, if $f(G, k)$ is a positive instance of `HVC`, then it has a cover of size

$$k = \frac{1}{2}(n + J)$$

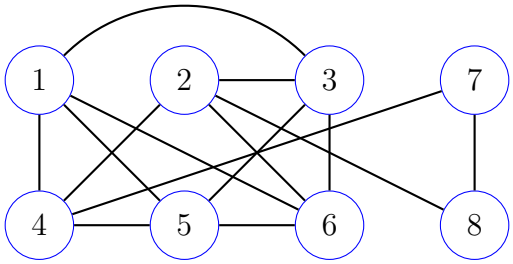
and we may assume that none of the added isolated vertices are part of the cover (since they don't cover any edges). Hence, G itself has a k cover and (G, k) is a positive instance of `VC`.

Case 3. $k < n/2$. In this case we have the opposite problem that we faced with Case 2, in that instance (G, k) involves checking if there is a cover of size *less than* half the number of vertices of G. Now when we reduce (G, k) to $f(G, k)$, we still want to add new vertices to G, but now we must add edges to these vertices in order to force the orginal k cover (assuming it exists) to take on a fraction of $1/2$ the total number of vertices in $f(G, k)$. Moreover, the ratio of added edges to vertices must exceed "1 to 2" since the original ratio of "k to $n/2$" already falls below the desired "1 to 2" ratio. One way of achieving this is to add isolated triangles. This works because a triangle has three vertices but only two are needed to cover its edges. This gives a "2 to 3" ratio which is sufficient for reaching a half cover so long as we add the appropriate number of triangles. Thus, we must add J isolated triangles so that

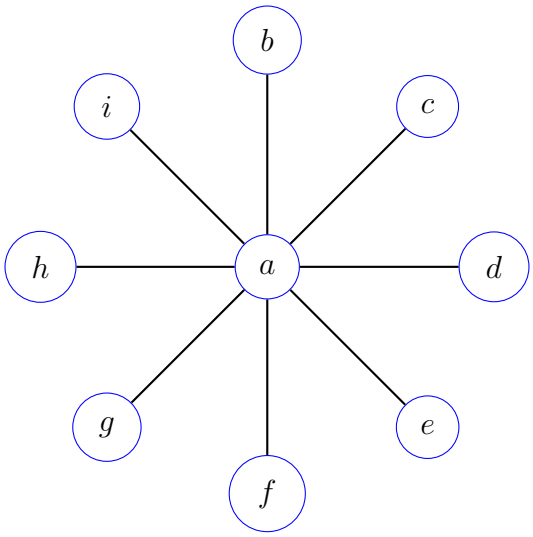
$$k + 2J = \frac{1}{2}(n + 3J).$$

Solving for J yields $J = n - 2k$. It's left as an exercise to verify that for this case (G, k) is a positive instance of `VC` iff $f(G, k)$ is a positive instance of `HVC`.

Example 4.7. Given VC instance $(G, k = 5)$, where G is shown below, draw $f(G, k)$.



Example 4.8. Given VC instance $(G, k = 1)$, where G is shown below, draw $f(G, k)$.



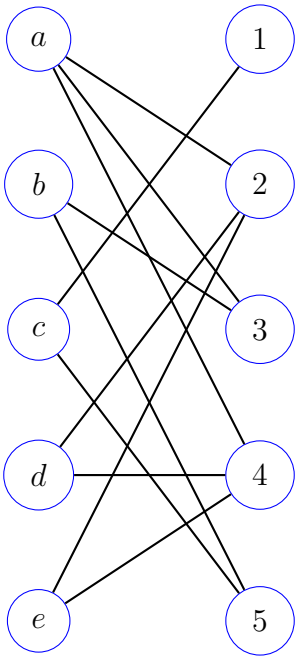


Figure 1: An example of a bipartite graph

5 Finding maximum matchings in bipartite graphs

The mapping reducibilities provided in Examples 2.1 and Proposition 2.8 involved pairs of problems that were in some sense two sides of the same coin. Our next example involves a more subtle relationship between two seemingly unrelated graph problems, namely the `Max Flow` problem and the `Maximum Bipartite Matching (MBM)` problem. We've already encountered the former Section 6 of the Turing Reducibility lecture, and now describe the latter before providing a map reduction from `MBM` to `Max Flow`.

A **bipartite** graph $G = (V_1, V_2, E)$ consists of two nonempty disjoing sets of vertices V_1 and V_2 and a set of edges for which each edge is incident with one vertex in V_1 and one vertex in V_2. Figure 1 shows an example of a bipartite graph, with the $V_1 = \{a, b, c, d, e\}$ and $V_2 = \{1, 2, 3, 4, 5\}$.

A **matching** M in the graph is a subset $M \subseteq E$ of edges of G, no two of which share a common vertex. Figure 2 shows a matchiing for the graph in Figure 1.

The following definitions prove useful when discussing matchings in a graph. Let M be a matching.

Maximum Matching M is a called a **maximum** matching iff for any other matching M' of G, $|M'| \leq |M|$.

Maximal Matching M is called **maximal** iff it is not contained in any larger matching. In other words, one cannot increase the size of M simply by adding another edge to M.

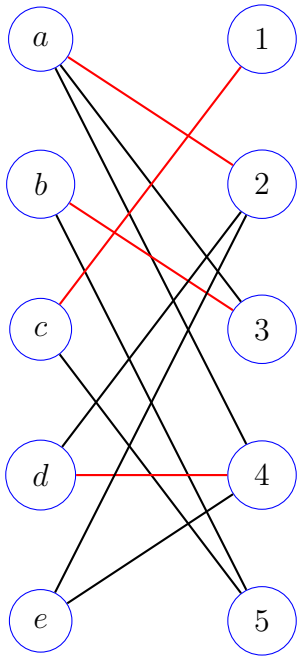


Figure 2: The red edges form a maximal matching for the bipartite graph.

Matched and Free Edges The edges of M are called **matched** edges while the edges in $E - M$ are called **free** edges.

Exposed and Covered Vertices Any vertex that is not incident with an edge in M is said to be **exposed** by M. otherwise it is **covered** by M.

Increment Matching M^* is called an **increment** of M iff $|M^*| = |M| + 1$

Notice that the matching M in Figure 2 is maximal, since no edge can be added to M to increase its size. Indeed, the only exposed vertices are e and 5, but they are not adjacent. However, M is not a maximum matching. A maximum matching is shown in Figure 3. This matching is an increment of M. Finally, `MBM` is the problem of finding the size of a maximum matching in a bipartite graph.

We now describe a map reduction from `MBM` to `Max Flow`. Let $G = (V_1, V_2, E)$ be a bipartite graph. Then

$$f(G) = G' = (V', E', c, s, t),$$

where the directed network G' is defined as follows.

Vertices $V' = V_1 \cup V_2 \cup \{s, t\}$, where s is the source vertex and t is the destination vertex

Edges i) $(s, u) \in E'$ for each vertex $u \in V_1$, ii) $(v, t) \in E'$ for each vertex $v \in V_2$, and iii) if $e = (u, v) \in E$ is an edge of G, with $u \in V_1$ and $v \in V_2$, then $e = (u, v) \in E'$ is a directed edge of G'. Note: such an edge are called an **original edge** because it also exists in G.

Capacity all edges are assigned unit (i.e. 1) capacity

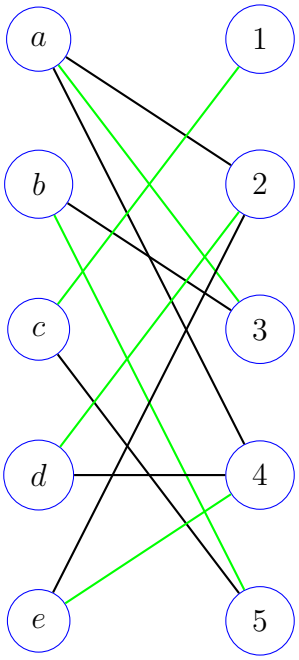


Figure 3: The green edges form a maximum matching for the bipartite graph.

Figure 4 shows $f(G)$ for the instance G of `MBM` shown in Figure 1.

Theorem 5.1. The above-described mapping from `MBM` to `Max Flow` is a mapping reduction, i.e. G has a matching of size k iff G' has a flow of size k.

Proof. Suppose M is a matching for G, with $|M| = k$. Then there exists a flow f_M on G' with the following property. For each edge $e = (u, v) \in M$,

$$f_M(s, u) = f_M(e) = f_M(v, t) = 1,$$

and $f_M(e) = 0$ for all other edges $e \in E'$. Figure 5 shows f_M for the network $G' = f(G)$, where G and M are the respective bipartite graph and matching shown in Figure 2.

To show that f_M is a flow, first note that $f_M(e) \leq 1$ and thus f_M never exceeds the unit capacity limit of e. We now show that each vertex $u \in V_1 \cup V_2$ preserves flow. Without loss of generality, assume $u \in V_1$. Case 1: u is exposed by M. Then there is no edge $e \in M$ that is incident with u. Then, by definition of f_M, there is zero flow both entering and leaving u. Case 2: u is covered by M. Then $f_M(s, u) = 1$ and there is unit flow entering u. Also, there is a unique edge $e \in M$ for which $e = (u, v)$, for some $v \in V_2$. Thus, $f_M(e) = 1$ and there is unit flow leaving u. Therefore, u conserves flow. Finally, notice that $s(f_M) = |M| = k$, the size of M.

Now assume $G' = f(G)$ has a flow f of size k, where k is a nonnegative integer. Based on the `Max-Flow` algorithm, we may assume that $f(e)$ is either 0 or 1, for every $e \in E'$. Let M denote the set of original edges e of G' for which $f(e) = 1$. Then we claim that M is a matching for G and that $|M| = k$. To see this, since $s(f) = k$, there are exactly k edges of the form (s, u) for which $f(s, u) = 1$. Then for each u for which $f(s, u) = 1$ there is a unique vertex $v_u \in V_2$ for which $f(u, v_u) = 1$. This

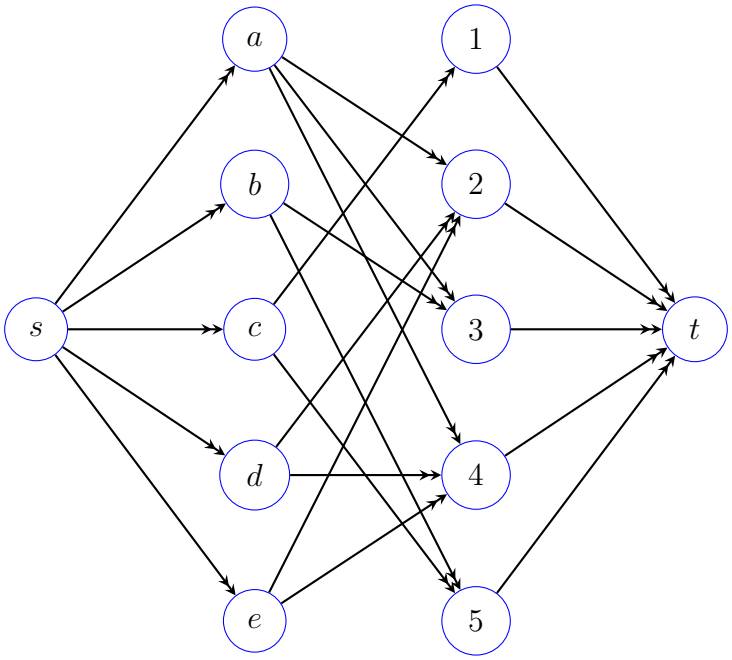


Figure 4: The directed network $G' = f(G)$, where G is the graph in Figure 1.

is because u must conserve flow and there is one unit of flow entering u. It follows then that

$$M = \{(u, v_u) | f(s, u) = 1$$

is a matching and that $|M| = k$. □

When reducing `MBM` to `Max Flow` using the mapping defined above, each residual network G'_f that results from a flow f through G' can be easily drawn using the following rule: for each $e \in E'$, if $f(e) = 1$, then e is oriented backwards in G'_f. Otherwise it remains forward oriented. Furthermore, what we earlier referred to as an augmenting path is now called an **alternating path** because, upon leaving s, it has the property of first reaching an exposed vertex in V_1, followed by alternating between forward and backward edges until it reaches an exposed vertex of V_2, and finally reaches t. Figure 6 shows the residual network G'_{f_M} for the network G' and flow f_M showed in Figure 5. Also, Figure 7 shows an alternating path (in green) for G'_{f_M}.

Finally, there is a nice way to characterize the increment matching M' that one obtains from alternating path P. Namely, $M' = M \oplus P$, where the (edge) set operation $M \oplus P$ is called the **symmetric difference** between M and P and consists of all edges that are either in M but not P, or in P but not M (here, we neither include the edge that leaves s, nor the one that enters t).

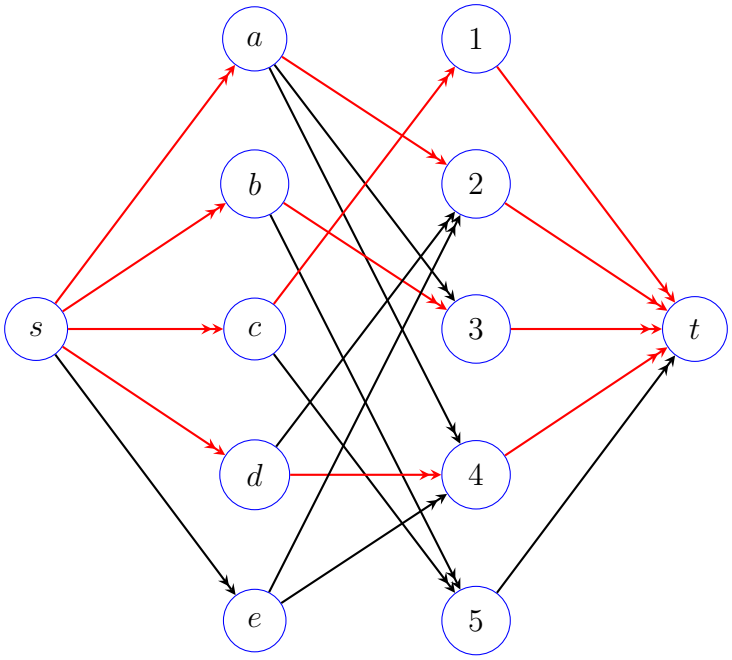


Figure 5: The network $G' = f(G)$, with flow f_M (in red) associated with matching M from Figure 2.

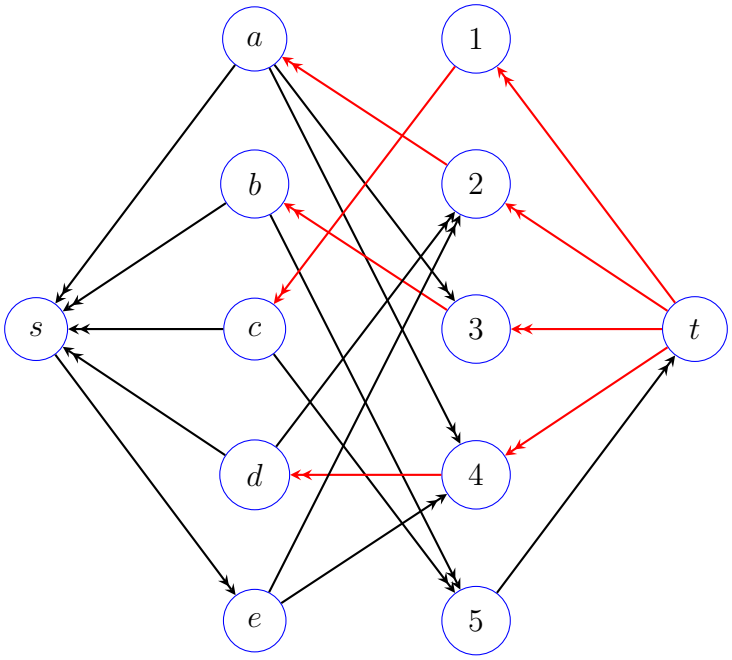


Figure 6: Residual network G'_{f_M} for graph G' and flow f_M from Figure 5.

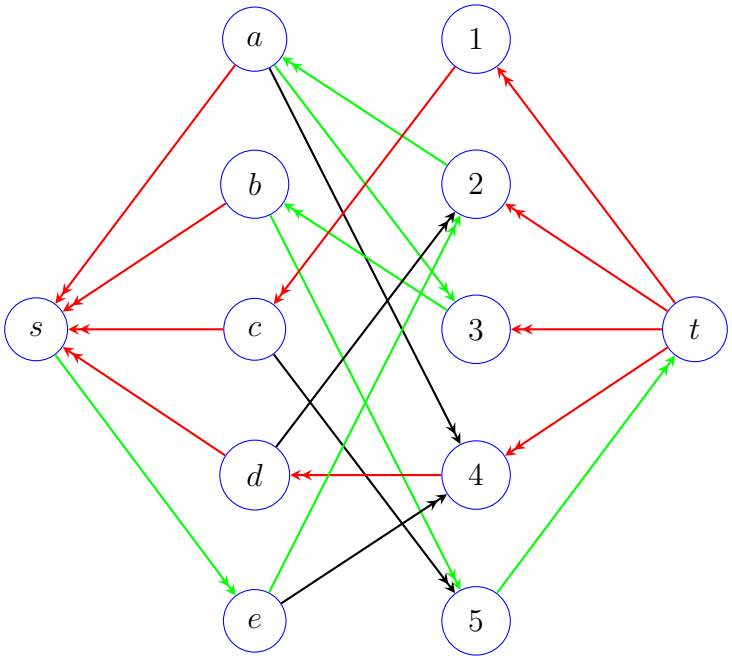


Figure 7: Alternating path (in green) for residual network G'_{f_M} of Figure 6.

Maximum Matching Algorithm

Input: bipartite graph G.

Output: a maximum matching for G.

Compute $f(G) = G' = (V', E', c, s, t)$.

Initialize matching M: $M \leftarrow \emptyset$.

While there is a vertex of G that is exposed by M

If G'_{f_M} has no alternating path, then return M.
Let P be an alternating path of G'_{f_M}.
$M \leftarrow P \oplus M$.

Return M.

The **Perfect Bipartite Matching (PBM)** decision problem is the problem of deciding if a bipartite graph G has a **perfect matching**, i.e. one that covers every vertex of G. For a bipartite graph to have such a matching, it must be the case that $|V_1| = |V_2|$. Moreover, `PBM` mapping reducible to the decsion-problem version of `Max Flow`, where a problem instance is now a network G and an integer k, and the problem is to decide if G admits a flow of size k.

6 Some Consequences of Reducibility

If we have a reduction (Turing or mapping) from problem A to problem B, what can we infer about either of the problems? Two things we may be able to learn involve the notions of solvability and complexity.

We say that, e.g., problem A is **solvable** iff there is an algorithm that takes as input any instance x of A and computes the solution to that instance. If no such algorithm exists, then we say that A is **unsolvable**. In a lecture we provide examples of several unsolvable problems and techniques for proving their unsolvability.

Now, assuming A is solvable, the complexity of solving A refers to determining lower bounds on the amount of memory and/or steps required by any algorithm that solves A. Establishing complexity bounds for a problem can seem extremely difficult if not impossible, but reducibility can nevertheless give us some insight with regards to the relative complexity of the problem.

Theorem 6.1. If $A \leq_{\mathrm{T}} B$ or $A \leq_m B$, then the following statements hold.

1. If B is solvable then so is A.
2. If A is unsolvable then so is B.

Proof. Since a mapping reducibility is a special case of Turing reducibility, we may assume $A \leq_{\mathrm{T}} B$.

For the first statement, suppose B is solvable via some algorithm $\mathcal{B}$. Let $\mathcal{R}$ denote the algorithm that Turing reduces A to B by making use of queries to a B-oracle. Note that $\mathcal{R}$ may not be a valid algorithm in the sense with which we are normally familiar. This is so because there may not be an algorithmic way to obtain the answers to the B-queries that appear in the computation. However, since B is solvable via $\mathcal{B}$, there *is* an algorithmic means for obtaining the answers, and we may replace each query of the form query(b), where b is an instance of B, with a call to algorithm $\mathcal{B}$ on input b. Therefore, $\mathcal{R}$ together with algorithm $\mathcal{B}$ that is used to answer B-queries may be combined to define an algorithm in the normal sense.

Finally, notice that the second statement is just the contrapositive of the first, an thus is also a true statement. □

Theorem 6.2. If $A \leq_m^p B$, then the following statements hold.

1. If B is solvable in $O(p(m))$ steps, for some polynomial p, where m is the size parameter for B, then A is solvable in $O(r(n))$ steps, for some polynomial $r(n)$, where n is the size parameter for A.
2. If A cannot be solved in $O(r(n))$ steps, for any polynomial $r(n)$, then B cannot be solved in $O(p(m))$ steps for any polynomial $p(m)$.

Proof. As with Theorem 6.1, the second statement is the contrapositive of the first, and so it suffices to prove the first. To this end, assume B is solvable in $O(p(m))$ steps via some algorithm $\mathcal{B}$. Since $A \leq_m^p B$, there is a map $f : A \to B$ that is computable in $O(q(n))$ steps for some polynomial $q(n)$, where n is the size parameter for A. Moreover, the solution to instance a of A equals the solution to instance $f(a)$ of B. Therefore, an algorithm for solving A first computes instance $f(a)$ via the algorithm that computes f, and then applies algorithm $\mathcal{B}$ to instance $f(a)$. Now, since $f(a)$ is computable in $O(q(n))$ steps, we may assume that the size of $f(a)$ is $m = O(q(n))$ bits, i.e. each algorithm step can construct at most $O(1)$ bits of the output. Thus, $\mathcal{B}$ runs on input $f(a)$ whose size is $m = O(q(n))$ bits which means the algorithm's running time is $O(p(q(n)))$. Thus, the total running time for solving instance a is

$$O(q(n)) + p(q(n))) = O(p(q(n))),$$

and so we may set $r(n) = q(p(n))$. □

Example 6.3. Suppose f map reduces A to B and is computable in $O(n^3)$ steps. Furthermore, suppose algorithm $\mathcal{B}$ solves an instance of B in $O(m^4)$ steps. Determine the running time of the following algorithm that solves A.

Algorithm for Solving A

Input: instance a of A.

Output: solution to a.

Compute $b = f(a)$ which is an instance of B.

Return $\mathcal{B}(b)$.

Mapping Reducibility Core Exercises

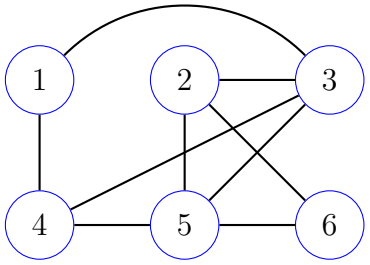
1. Recall the `Even` and `Odd` decision problems from Example 2.1. Which of the following functions provides a mapping reduction from `Even` to `Odd`?

   a. $f(n) = n^2$

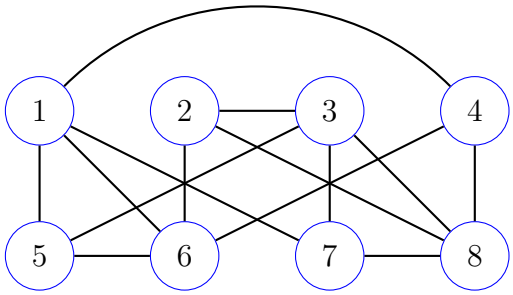
   b. $f(n) = 2n + 5$

   c. $f(n) = 3n - 7$

2. Draw the complement of the graph below.

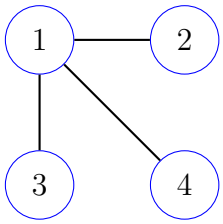


3. The simple graph $G = (V, E)$ shown below is an instance of the `Maximum Independent Set (MIS)` optimization problem. Draw $f(G)$, where f maps a graph to its complement. Verify that the largest clique in G corresponds with the largest independent set in $f(G)$. Similarly, verify that the largest independent set in G corresponds with the largest clique in $f(G)$.



4. Given the instance $S = \{16, 21, 23, 25, 38, 47, 61, 73\}$ of `Set Partition`, provide the instance of `SS` to which it reduces via the mapping $f(S) = (S, t = M/2)$. Verify that f is valid mapping reduction with respect to S.

5. The graph G shown below represents an instance of the `Hamilton Path` decision problem. Compute $f(G)$, where f is the mapping reduction from `HP` to `LPath` described in Example 3.10. Verify that the mapping is correct in the sense that the decision for G is equal to the decision for $f(G)$. Explain.



6. For each of the following instances of `Subset Sum`, show the mapping to `Set Partition` provided in lecture. In the case of both a) and b), verify that f is a valid mapping.

   a. $(\{3, 7, 11, 29, 44, 53, 66, 81\}, t = 86)$

   b. $(\{3, 7, 11, 29, 44, 53, 66, 81\}, t = 147)$

   c. $(\{3, 7, 11, 29, 44, 53, 66, 81\}, t = 177)$

7. Recall the contraction mapping f from `VC` to `HVC`. Suppose $G = (V, E)$ is a simple graph with $|V| = 135$. Then if $f(G, k = 39) = G'$, then describe the relationship between G' and G.

8. Repeat the previous exercise, but now assume $|V| = 152$ and $k = 100$.

9. The `Half Clique` decision problem is the problem of deciding if a simple graph $G = (V, E)$ has a Clique of size $|V|/2$. Provide an embedding reduction f from `Half Clique` to `Clique`.

10. Provide a contraction reduction f from `Clique` to `Half Clique`. Defend your answer, meaning prove that (G, k) is a positive instance of `Clique` iff $f(G, k)$ is a positive instance of `Half Clique`.

11. For bipartite graph $G = (U, V, E)$ we have $U = \{u_1, u_2, u_3, u_4\}$, $V = \{v_1, v_2, v_3, v_4\}$, and

$$E = \{(u_1, v_1), (u_1, v_2), (u_1, v_4), (u_2, v_1), (u_2, v_3),$$
$$(u_2, v_4), (u_3, v_1), (u_3, v_3), (u_4, v_1), (u_4, v_3)\}.$$

    a. Draw G.

    b. Does G have a (non-maximum) maximal matching M of size 1? size 2? size 3?

    c. For each `yes` answer to the previous part, draw the residual network G'_{f_M} associated with the matching, and provide an alternating path P in G'_{f_M}. Use the alternating path to find an increment of M.

12. At a school ice-cream party there are five dixie cups of ice cream that remain to be served. Each cup has a different flavor: vanilla, chocolate, cherry, rocky road, and mint and chip. There are five children who have yet to be served: Abe, Ben, Cris, Dan, and Eva. The ice-cream preferences of these children are shown below.

Child	**Vanilla**	**Chocolate**	**Cherry**	**Rocky Road**	**Mint & Chip**
Abe	X		X		X
Ben		X			
Cris		X		X	
Dan		X			X
Eva			X	X	

    In a rush to get their ice cream, Abe grabbed the cherry, Cris the chocolate, Dan the mint and chip, and Eva the rocky road. This left Ben with a (vanilla) flavor that he does not like, and which he refused to eat. Show how the maximum-matching algorithm can be used to increase the current matching (of four children to four ice creams that they prefer) to a matching of size five, in which each child will be assigned an ice cream that he or she prefers.

Solutions to Mapping Reducibility Core Exercises

1. f : `Even` $\rightarrow$ `Odd` map reduces `Even` to `Odd` iff it maps evens to odds and odds to evens (why?).

   a. $f(n) = n^2$. No, since f maps evens to evens and odds to odds. For example $f(2) = 4$.

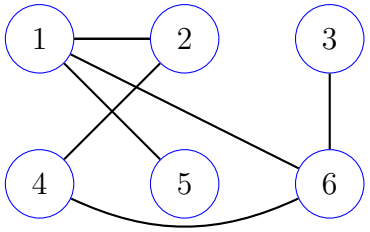
   b. $f(n) = 2n + 5$. No, since f maps evens to odds, but maps odds to odds. For example, $f(3) = 11$.

   c. $f(n) = 3n - 7$. Yes. If n is even, then $n = 2k$ for some integer k. Thus

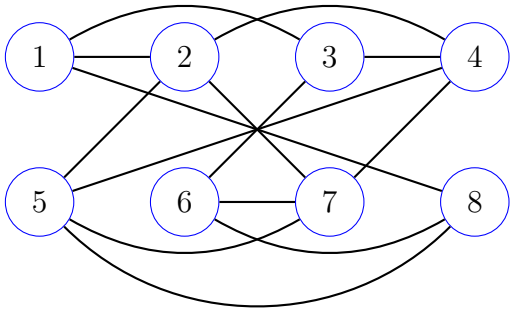
   $$3n - 7 = 3(2k) - 8 + 1 = 6k - 8 + 1 = 2(3k - 4) + 1,$$

   which is an odd number. Similarly, if n is odd, $3n - 7$ is even.

2. The complement graph is shown below.



3. The simple graph $f(G) = \overline{G} = (V, \overline{E})$ is shown below. This is a valid mapping reduction (either from `Max Clique` to `Max IS` or vice versa). To see this, notice that G has a maximum clique of size 3 (e.g. via vertices 2,3,8), while $f(G)$ has an IS of size 3 using the same vertices. Similarly, $\{2, 4, 5, 7\}$ is a max IS for G while the set is also a max Clique for $f(G)$.



4. $f(S) = (S, t = 152)$. f is valid with respect to S since S is a positive instance of `SP` via set partition $A = \{16, 25, 38, 73\}$ and $B = \{21, 23, 47, 61\}$ while $f(S)$ is a positive instance of `SS` via $A = \{16, 25, 38, 73\}$ which sums to $t = 152$.

5. $f(G) = (G, k = n - 1) = (G, k = 3)$. For this instance, the mapping is correct since G is a negative instance of `HP` which is equivalent to saying that it does not have a simple path of length 3. Furthermore G is negative for `HP` because three of G's vertices have degree 1, but a simple path of length three requires at least two vertices that have degree at least 2.

6. For each of the tree cases we have $f(S, t) = S'$, where

   a. $S' = \{3, 7, 11, 29, 44, 53, 66, 81, 122\}$

b. $S' = \{3, 7, 11, 29, 44, 53, 66, 81\}$

c. $S' = \{3, 7, 11, 29, 44, 53, 60, 66, 81\}$

We have that f is a valid reduction for (S, t) of part a, since, by trial and error, we observe that no subset of S can sum to $t = 86$. Similarly, there is no way to partition S' into sets A and B with both having the same sum. Also, f is a valid reduction for (S, t) of part b, since $(S, t = 147)$ is positive for `SS` via subset $A = \{3, 7, 11, 29, 44, 53\}$ which sums to $t = 147$ while S' is a positive instance of `SP` since it may be partitioned into $A = \{3, 7, 11, 29, 44, 53\}$ and $B = \{66, 81\}$.

7. Since $k = 39 < 135/2 = 67.5$, we have $f(G, k = 39) = G'$, where G' is the graph G with the addition of J triangles, where J satisifes

$$\frac{39 + 2J}{135 + 3J} = \frac{1}{2}$$

which implies $J = 57$. Thefore, G' is the same graph G, but with the addition of 57 distinct triangles. The addition of these triangles will make it so that G' has a half vertex cover iff G has a vertex cover of size 39.

8. Since $k = 100 > 152/2 = 76$, $f(G, k = 100) = G'$, where G' is the graph G with the addition of $2(100) - 152 = 48$ new isolated vertices.

9. Let $G = (V, E)$ be a simple graph. Then $f(G) = (G, k = |V|/2)$, i.e. G is a positive instance of `Half Clique` iff it has a clique of size $|V|/2$ iff $(G, k = |V|/2)$ is a positive instance of `Clique`.

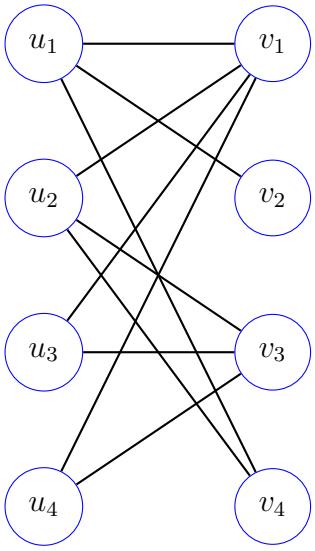
10. The contraction reduction from `Clique` to `Half Clique` is similar to the one given for the reduction from `VC` to `HVC`. Let $G = (V, E)$ be a simple graph and $k \geq 0$ be a nonnegative integer between 1 and $n = |V|$. If $k = n/2$, then $f(G, k) = G$ since (G, k) would then be an actual `Half Clique` problem instance. Now suppose $k < n/2$. Then $f(G, k) = G'$ where G' is formed by adding J additional vertices to G and placing edges between them so that they form a J-clique C_J. Futhermore, we also add an edge between each vertex in C_J and each vertex in V. Therefore, G will have a k clique iff G' has a $k + J$ clique. Moreover, this $k + J$-clique will be a half clique for G' iff

$$k + J = \frac{1}{2}(n + J) \Leftrightarrow J = n - 2k.$$

Therefore, we require that $J = n - 2k$. Finally, if $k > n/2$, then $f(G, k) = G'$ where G' is formed by adding J additional *isolated* vertices to G Therefore, G will have a k clique iff G' has a k clique. Moreover, this k-clique will be a half clique for G' iff

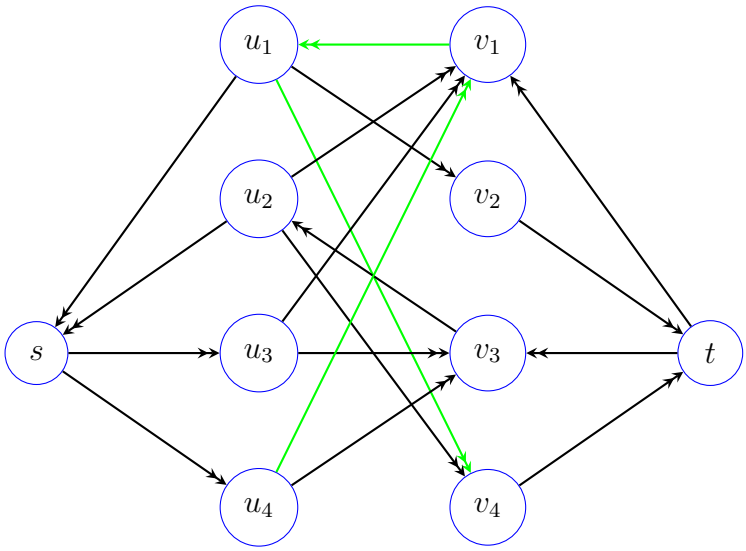
$$k = \frac{1}{2}(n + J) \Leftrightarrow J = 2k - n.$$

11. a. Below is a graph of $G = (U, V, E)$

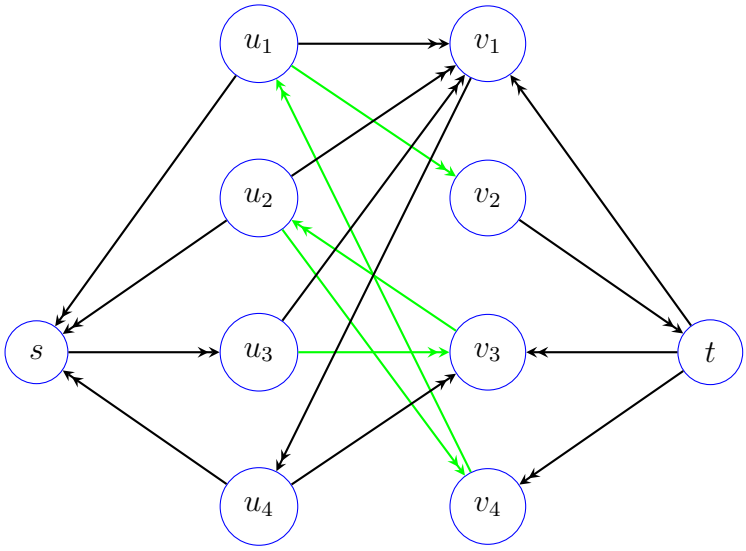


b. G does not have a size-1 maximal matching since, for any edge e, there is an edge e_2 that does not share any vertices with e. However, $M_2 = \{(u_1, v_1), (u_2, v_3)\}$ is a maximal matching of size 2, since u_1 and u_2 are the only vertices incident with v_2 and v_4. Then $P = u_4, v_1, u_1, v_4$ is an alternating path in $G'_{F_{M_2}}$, and $M_3 = P \oplus M_2 = \{(u_1, v_4), (u_4, v_1), (u_2, v_3)\}$ is a maximal matching of size 3.

c. $G'_{F_{M_2}}$ is shown below with an alternating path P drawn in green. This yields increment $M_3 = P \oplus M_2 = \{(u_1, v_4), (u_4, v_1), (u_2, v_3)\}$.



$G'_{F_{M_3}}$ is shown below with an alternating path P drawn in green. This yields increment $M_4 = P \oplus M_3 = \{(u_1, v_2), (u_2, v_4), (u_3, v_3), (u_4, v_1)\}$.

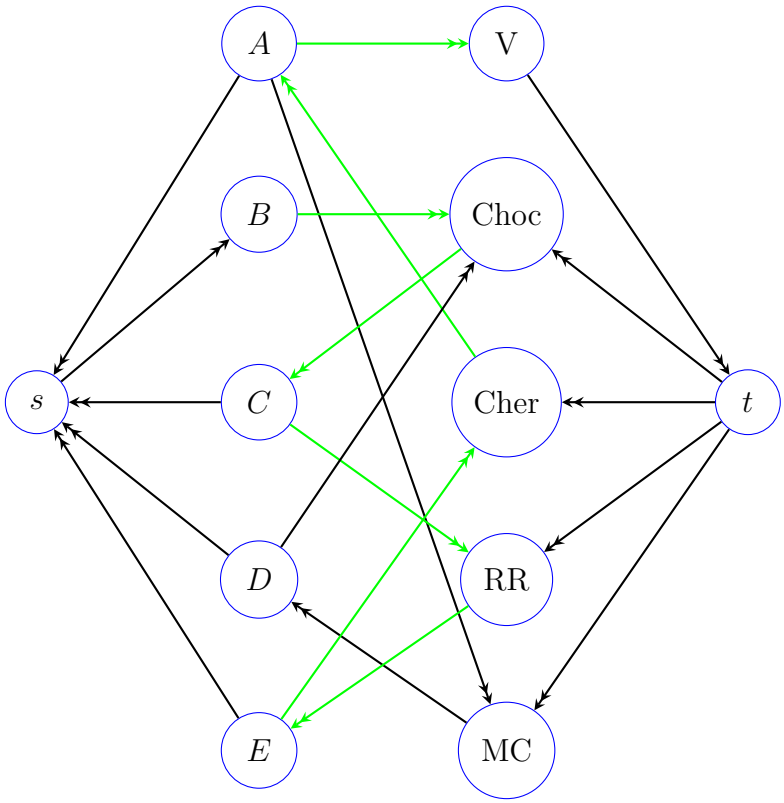


12. Let $G = (U, V, E)$ be the bipartite graph whose U set represents the set of children, and V set represents the set of ice-cream flavors. The $(u, v) \in E$ iff child u likes ice cream v. The children rushing for their ice cream resulted in the matching

$$M = \{(A, \text{Cher}), (C, \text{Choc}), (D, \text{MC}), (E, \text{RR})\}.$$

The residual network G'_{F_M} below shows an alternating path P (in green) for which

$$M^* = P \oplus M = \{(A, \text{V}), (B, \text{Choc}), (C, \text{RR}), (D, \text{MC}), (E, \text{Cher})\}.$$



Additional Exercises

A. An instance of decision problem `Set Tri-Partition (STP)` is a set of nonnegative integers S, and the problem is to decide if there exist subsets $A_1, A_2, A_3 \subseteq S$ for which i) $A_1 \cup A_2 \cup A_3 = S$, ii) $A_i \cap A_j = \emptyset$, for $i \neq j$, and iii)

$$\sum_{a \in A_1} a = \sum_{a \in A_2} a = \sum_{a \in A_3} a.$$

Show that the mapping $f : \texttt{STP} \rightarrow \texttt{SS}$ defined by $f(S) = (S, M/3)$ where

$$M = \sum_{s \in S} s,$$

is *not* a valid mapping reduction. Hint: provide a negative instance of `STP` and show that f maps it to a positive instance.

B. Recall the mapping reduction that maps a graph to its complement (and was used to show, e.g., the reducibility of `Max Clique` to `Max IS`). Show that the reduction takes a polynomial number of steps in $n = |V|$ and $m = |E|$. Do this by writing pseudocode and analyzing the number of steps taken by the code.

C. Recall the mapping reduction from `Subset Sum (SS)` to `Set Partition (SP)` described in Section 4. Show that this reduction can be computed in polynomial-time by providing a big-O expression that gives the running time for computing the reduction. Defend your answer.

Solutions to Additional Exercises

A. Consider the `STP` instance $S = \{6, 12\}$. Then $f(S) = (S, k = 18/3 = 6)$ is a positive instance of `Subset Sum` via $A = \{6\}$, but S is a *negative* instance of `STP` (why?).

B. Let $G = (V, E)$ be given. Assume that the vertex set is $V = \{1, 2, \ldots, n\}$. Since $\overline{G}$ and G both have the same vertex set, mapping f only has to produce the undirected edges (i, j) that belong to $\overline{E}$, i.e. those for which $i < j$ and $(i, j) \notin E$. This can be done as follows.

> Create hash table T and store the edges of E in T.
>
> For each $i \in V$,
>
> > For each $j \in V$
> >
> > > If $i \geq j$, then continue.
> > >
> > > If $(i, j) \notin T$, then print (i, j).

It takes $O(m)$ steps to create and populate T, and $O(n^2)$ steps to iterate through the nested `for`-loops. Moreover, we may assume that both table lookup and printing can be done in $O(1)$ steps. Therefore, the steps required to print the members of $\overline{E}$ amount to $O(m+n^2)$ steps which is a quadratic polynomial in m and n. Therefore, it is a polynomial-step mapping reduction.

C. The most costly step in the reduction from `SS` to `SP` is in the summing of all the members of S in order to compute $M = \sum_{s \in S} s$. We may assume that each member of S has not more than $\log t$ bits, since otherwise its value would exceed that of t and thus could never be part of a subset that sums to t. Thus, to compute M we must add $n = |S|$ numbers, each with $O(\log t)$ bits. This can be done in no more than $O(n^2 + \log t)$ steps which is a quadratic polynomial in $\log t$ and n. Therefore, it is a polynomial-step mapping reduction.