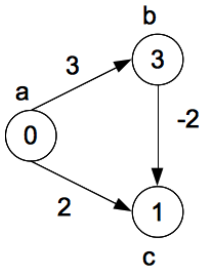
# CS360 Assignment 10

24.3-2 Give a simple example of a directed graph with negative weight edges for which Dijkstra's algorithm produces incorrect answers. Why doesn't the proof of Theorem 24.6 go through when negative weight edges are allowed?

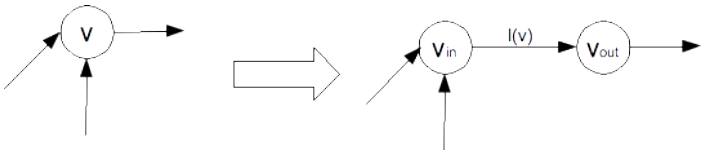
Consider the simple graph




Clearly running Dijkstra's algorithm starting at vertex $a$ would give $d =< 0, 3, 2 >, \pi =< /, a, a >$. However the *shortest* path to vertex $c$ would be $< a, b, c >$ with $c.d = 1$. The problem occurs in the assumption that $\delta(s, y) \leq \delta(s, u)$ which ensures that the $v.d = \delta(u, v)$ when the vertex is DEQUEUED. Note Bellman-Ford does not have this condition as all edges are repeatedly checked.

26.1-7 Suppose that, in addition to edge capacities, a flow network has *vertex capacities*. That is each vertex $v$ has a limit $l(v)$ on how much flow can pass through $v$. Show how to transform a flow network $G(V, E)$ with vertex capacities into an equivalent flow network $G'(V', E')$ without vertex capacities, such that a maximum flow in $G'$ has the same value as a maximum flow in $G$. How many vertices and edges does $G'$ have?

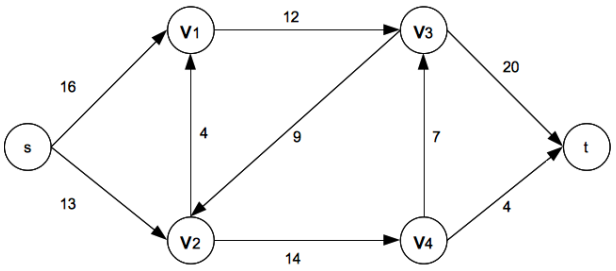
In order to limit the flow through a vertex based on its *vertex capacity*, we can simply convert each vertex $v$ into a pair of vertices $v_{in}$, $v_{out}$ with a *single* edge between them with capacity equal to the vertex capacity. Then any incoming edges are connected to vertex $v_{in}$ and any outgoing edges are connected to vertex $v_{out}$ as shown in the figure below



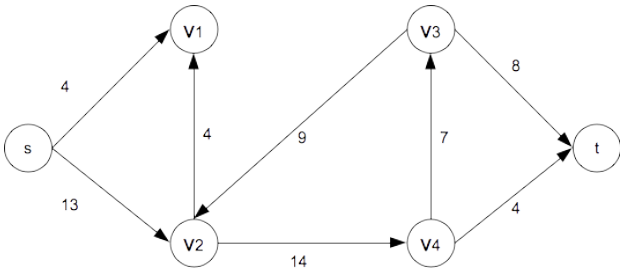

Thus since all vertices except the source and sink will be converted into two vertices, $|V'| = 2(|V| - 2) = 2|V| - 4$. A new edge will be created for each vertex (except the source and sink) giving, $|E'| = |E| + (|V| - 2)$.

26.2-3 Show the execution of the Edmonds-Karp algorithm on the flow network of Figure 26.1(a).

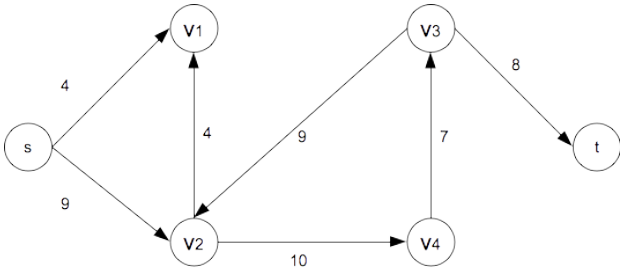
The original network is



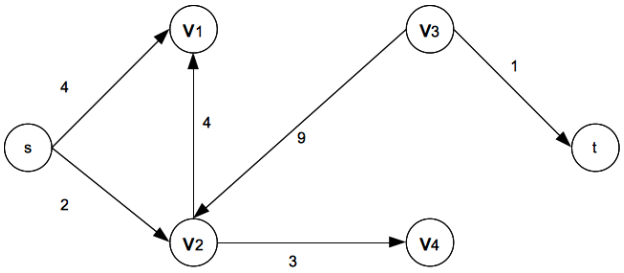

Using Edmonds-Karp, i.e. BFS to find the augmenting path, would give (assuming we take vertices in numerical order) $p_1 =< s, v_1, v_3, t >$ which has 3 edges and $c_p = 12$. Thus the flow $|f| = 12$ and the residual network (showing only residual capacities) is



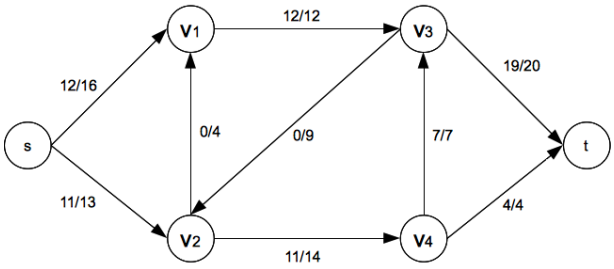

The next path found with BFS would be $p_2 =< s, v_2, v_4, t >$ which has 3 edges and $c_p = 4$. Thus the flow $|f| = 16$ and the residual network is




The next path found with BFS would be $p_3 =< s, v_2, v_4, v_3, t >$ which has 4 edges and $c_p = 7$. Thus the flow $|f| = 23$ and the residual network is




At this point BFS finds no other paths from $s$ to $t$ and thus the algorithm terminates with $|f^*| = 23$. The final flow network is




34-2 *Bonnie and Clyde.* Bonnie and Clyde have just robbed a bank and have a bag of money containing $n$ items they wish to divide up. Which of the following problems have polynomial time solutions and which are $NP$-complete.

a. The bag contains $n$ coins of which there are only *two* denominations, say $x$ and $y$, which they wish to divide evenly.

This problem is in $P$ since *brute force* is a polynomial time solution. Assume there are $a$ coins of type $x$ and $b$ coins of type $y$ - hence the total stolen amount is $M = ax + by$ and we wish to determine if there is a division such that $a_1x + b_1y = a_2x + b_2y = M/2$ where $a = a_1 + a_2$ and $b = b_1 + b_2$. Since the *number* of coins are integers, we can simply examine all pairs $(a_i, b_j)$ where $0 \leq a_i \leq a$ and $0 \leq b_j \leq b$ checking if $a_ix + b_jy = M/2$. Clearly this will take $O(ab) = O(n^2)$ worst case time which is polynomial (but not necessarily the *most efficient* way to determine the answer .) Note we only need to show *a* polynomial time solution exists even if it is *inefficient*.

b. The bag contains $n$ coins of which there are an *arbitrary* number of denominations such that each denomination is an integer power of 2, i.e. 2, 4, 8, etc., which they wish to divide evenly.

This problem is also in $P$ as can be shown by a *greedy algorithm*. First sort the coins in decreasing order (polynomial). Then give the largest coin to Bonnie. Give Clyde an equal amount *using the largest coins possible* until his amount matches Bonnie's. Note this can *always* be done since each coin amount can be

made from smaller coin amounts. Hence either Clyde's amount can be made equal to Bonnie's or he takes all the remaining coins yet still has less than Bonnie (in which case there is no equal division). If there is an equal division, then repeat the process giving the largest remaining coin to Bonnie and determining if Clyde can obtain an equal amount from the remaining coins. Since each coin is distributed only once, the division takes $O(n)$ time which again is polynomial. Thus the problem can be solved using a combination of polynomial time algorithms.

c. The bag contains $n$ checks (*amazingly* made out to "Bonnie or Clyde") which they wish to divide equally.

This version of the problem is $NP$-complete. It can be shown that the *set-partition* problem is $NP$-complete. The set-partition problem states that given a set of numbers, can they be partitioned into two sets such that the sum of the values in both sets is equal. Thus we can cast Bonnie and Clyde's problem directly into a set-partition problem since the checks are numbers and we wish to find an *equal* split.

d. The bag contains $n$ checks (again *amazingly* made out to "Bonnie or Clyde") which they wish to divide such that the difference in the split is no larger than 100 dollars.

This problem *appears* to be easier than part (c) since now we do not need an *exact* split, but rather have some slop to find a partition with. **Unfortunately** even with this extention, this variant is still $NP$-complete. To show this, we will again reduce set-partition to this Bonnie and Clyde problem. Assume we are given a set of values and a polynomial time algorithm to solve the sloppy Bonnie and Clyde problem. If we multiply each value in the set by 1000 (which can clearly be done in polynomial time) and let those become the checks, we can then run the polynomial Bonnie and Clyde algorithm. The result of this algorithm will decide whether or not a partition exists such that the difference is no more than \$100. If the answer is *yes*, then there must be a equal partition for the original values since if there were not an equal partition of the original values then there would be a difference of at least \$1 which would be a difference of \$1000 in the transformed values (and then the Bonnie and Clyde algorithm would not have returned *yes*). Conversely, if there is an equal partition of the original values, then there will still be an equal partition of the transformed values. Therefore set-partition is solvable in polynomial time *if and only if* the Bonnie and Clyde variant is solvable in polynomial time. Thus the variant is also $NP$-complete.