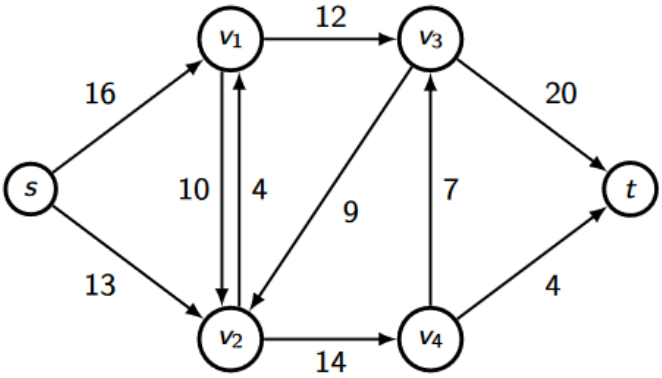
Flow Networks - Notes

Alperen Onur (z5161138)

August 2024

1 Flow Networks

A flow network $G = (V, E)$ is a directed graph in which edge $e = (u, v) \in E$ has a positive integer capacity $c(u, v) > 0$.
There are two distinguished vertices: a source s and a sink t.



A flow in G is a function $f : E \to [0, \infty), f(u, v) \geq 0$, which satisfies

1 Capacity constraint: for all edges $e = (u, v) \in E$ we require

$$f(u, v) \leq c(u, v),$$

ie) the flow through any edge does not exceed its capacity.

2 Flow conservation: for all vertices $v \in V \setminus \{s, t\}$ we require

$$\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w)$$

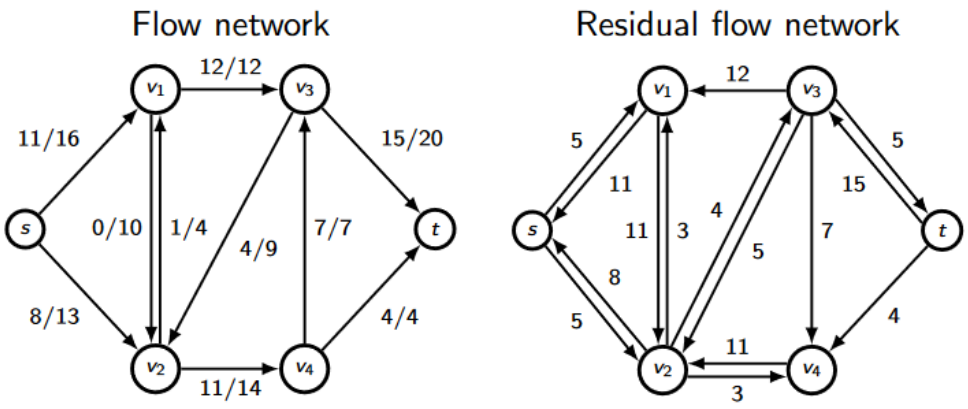
ie) the flow into any vertex (other than the source and sink) equals the flow out of that vertex

Integrality Theorem.
If all capacities are integers, then there is a flow of maximum value such that $f(u, v)$ is an integer for each edge $(u, v) \in E$.

2 Maximum flow & Residual Flow Networks.

A greedy approach to sending one unit of flow at a time along arbitrarily chosen paths does not always achieve the maximum flow. We need a way to correct our mistakes as we send flow through the graph! We can achieve this with a **residual flow network** such that we can flow in opposite directions to 'cancel out' earlier allocations.

Residual Flow Network.
Given a flow in a flow network, the residual flow network is the network made up of the left over capacities.



Suppose the original flow network has an edge from $v \to w$ with capacity c, and that f units of flow are being sent through this edge.
The residual flow network has two edges:

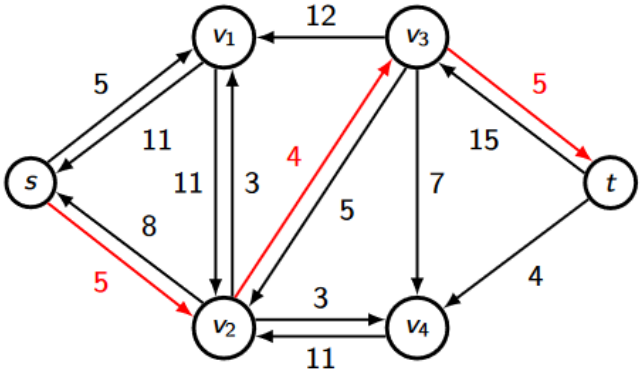
1 An edge from $v \to w$ with capacity $c - f$, and

2 and edge from w to v with capacity f.

These capacities represent the amount of additional flow that can be sent in each direction. Note that sending flow on the "virtual" edge from w to v counteracts the already assigned flow from v to w. Edges of capacity zero (when $f = 0$ or $f = c$) need not be included.

Assume an edge $v \to w$ with capacity c and f units of flow are being sent through this edge. Then a forward edge $c_1 - f_1$ allows additional units of flow and we can also send f_2 units to cancels the flow through the reverse edge.

Augmenting Paths.
An augmenting path is a path from s to t in the residual flow network.



The capacity of an augmenting path is the capacity of an augmenting path is the capacity of its "bottleneck" edge. We can now send that amount of flow along the augmenting path, recalculating the flow and the residual capacities for each edge used. Suppose we have an augmenting path of capacity f, including an edge from $v \to w$. We should:

- Cancel up to f units of flow being sent from w to v,
- Add the remainder of these f units to the flow being sent from $v \to w$,
- increase the residual capacity from $w \to v$ by f and,
- reduce the residual capacity from $v \to w$ by f.

3 Ford-Fulkerson algorithm

Keep adding flow through new augmenting paths for as long as it is possible. When there are no more augmenting paths, you have achieved the largest possible flow in the network. If all capacities are integers, then each augmenting path increases the flow through the network by at least 1 unit. However, the total flow is finite; the flow can not be larger than the sum of all capacities of all edges leaving the source. Therefore, the algorithm will terminate successfully. We need to also show that this process does indeed produce the maximum flow for a graph though. We'll use the idea of a minimum cut to prove this.

Cuts in a flow network.
A cut in a flow network is any partition of the vertices of the underlying graph into two subsets S and T such that

- $S \bigcup T = V$
- $S \bigcap T = \varnothing$
- $s \in S$ and $t \in T$

Given a flow f, the flow $f(S,T)$ through a cut $c(S,T)$ is the total flow through edges from S to T minus the total flow through edges from T to S

$$f(S,T) = \sum_{(u,v)\in E} f(u,v) : u \in S, v \in T - \sum_{(u,v)\in E} f(u,v) : u \in T, v \in S$$

Essentially, the equation is saying that edges from S to T counts its full capacity towards $c(S,T)$ but only the flow through it towards $f(S,T)$ and that an edge from T to S counts zero towards $c(S,T)$ but only counts the flow through it in the negative towards $f(S,T)$. Therefore $f(S,T) \leq c(S,T) = |f| \leq c(S,T)$ and so the value of any flow is at most the capacity of any cut.

Minimum cut.
The maximum amount of flow in a flow network is equal to the capacity of the cut of minimum capacity. We now need to show that when the Ford-Fulkerson algorithm terminates, it produces a flow equal to the capacity of an appropriately defined cut. We'll conduct a proof that shows that shows

1. All edges from S to T in a residual flow network are occupied with flow and all edges from T to S are empty.
2. If the above is true, than the flow across this cut (S,T) is precisely equal to the capacity of this cut.

Suppose an edge (u,v) from S to T has any additional capacity left. Then in the residual flow network, the path from s to u could be extended to a path from s to v. This contradicts our assumption that $v \in T$.

Suppose an edge (y,x) from T to S has any flow in it. Then in the residual flow network, the path from s to x could be extended to s to y. This contradicts our assumption that $y \in T$.

Therefore all edges from S to T are occupied with flows to the full capacity, and there is no flow from T to S, the flow across the cut (S,T) is precisely equal to the capacity of this cut. Thus, such a flow is a maximum and the corresponding cut is a minimum cut, regardless of the particular way in which the augmenting paths were chosen.

Time complexity.
The number of augmenting paths can be up the value of the max flow, denoted $|f|$. Each augmenting path is found in $O(V+E)$. In any sensible flow network, $V \leq E+1$, so we can write this as simply $O(E)$. Therefore the time complexity of the Ford-Fulkerson algorithm is $O(E|f|)$

If all capacities are all $\leq C$, then the length of the input is $O(V + E\log C)$. However the maximum flow $|f|$ can be as large as EC in general. Therefore, the time complexity $O(E|f|)$ can be exponential in the size of the input, which is unsatisfactory.

4 Edmonds-Karp Algorithm.

Improves the Ford-Fulkerson algorithm by always choosing the augmented path which consists of the fewest edges. In each step we find the next augmenting path using a breadth-first search in $O(V+E) = O(E)$ time.

The time complexity of Edmonds-Karp is $O(VE^2)$ since the number of augmenting paths is $O(VE)$ and each takes $O(E)$ time to find. Edmonds-Karp is just a specialisation of the Ford-Fulkerson algorithm and so the $O(E|f|)$ bound still applies.

5 Brooklyn 99

There are n cities, labelled $1, ..., n$ connected by m bidirectional roads. Each road connects two different cities. A criminal is currently in city 1 and wishes to get to city n on road. To catch the criminal, the police plan to prevent them criminal from reaching city n by blocking as few roads as possible. The task is the fund the smallest number of roads to block to prevent the criminal from starting at city 1 and ending at city n.

Graph Construction.
Construct the flow network containing the following:

- A source s representing the starting city and sink t representing the ending city
- Vertices v representing each city between s and t but not cities s and t
- Construct two edges between cities i and j between cities with roads that connect them to represent the bidirectional roads.

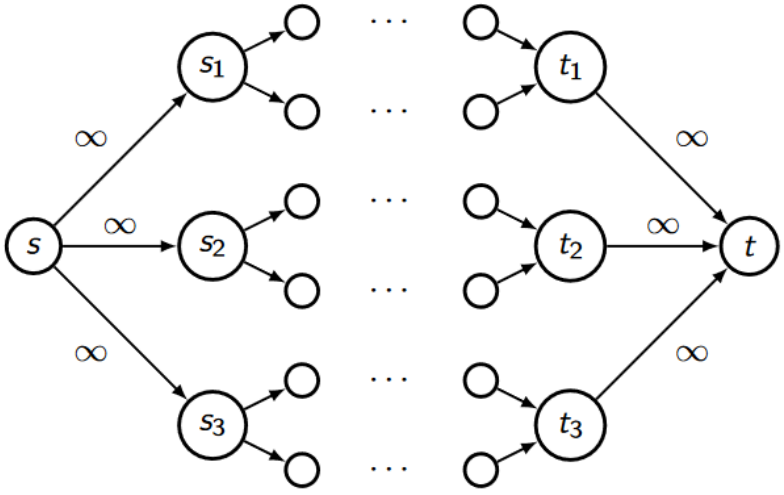
Algorithm.
Run an Edmonds-Karp algorithm to find the minimum cut of the flow network. This is because the minimum cut corresponds to the least amount of roads blocked off by the police supported by our graph construction (an example response should be more detailed).

Time Complexity.
Edmonds-karp can be bounded for this graph since m is the total number of roads and the max flow can not be greater than the amount of roads amount of roads that exist (it would be non-sensible to travel between cities over and over, and the max flow algorithm ensures this doesn't happen). Therefore we have $O(nm + n)$ edges and our final time complexity is $O(nm^2)$.

6 Multiple sources and sinks

Flow networks with sources and sinks are reducible to networks with a single source and single sink by adding a "super-source" and "super-sink" and connecting them to all sources and sinks respectively.

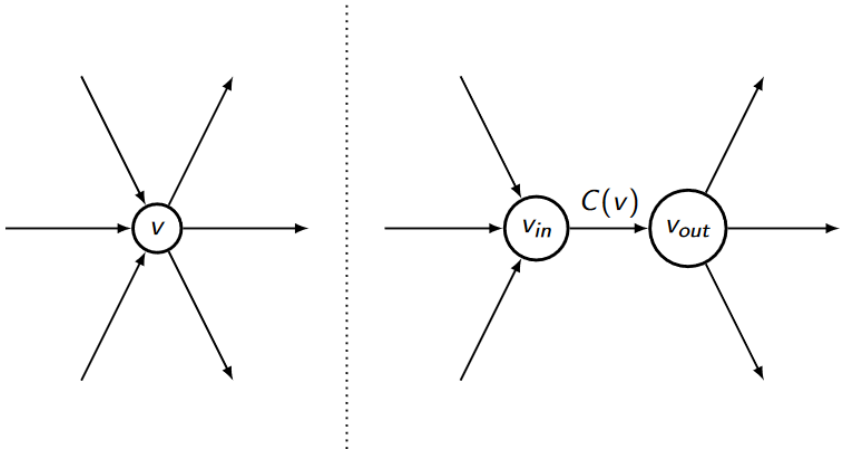
Infinite capacity edges.
Infinity capacity edges are used if there isn't a constraint on how much flow can originate at each of the sources or be collected at each of the sinks. Infinite capacity edges will never be saturated; there will always be more forward flow to send in the residual graph.



Sometimes not only the edges but also the vertices v_i of the flow might have capacities $C(v_i)$, which limit the total flow throughout coming to the vertex (and leaving the vertex).

$$\sum_{e(u,v)\in E} f(u,v) = \sum_{e(v,w)\in E} f(v,w) \leq C(v)$$

Suppose vertex v has capacity $C(v)$. Then split v into two vertices v_{in} and v_{out}. Attach all of $v's$ incoming edges to v_{in} and its outgoing edhes from v_{out}. Connect v_{in} and v_{out} with an edge $e^* = (v_{in}, v_{out})$ of capacity $C(v)$.

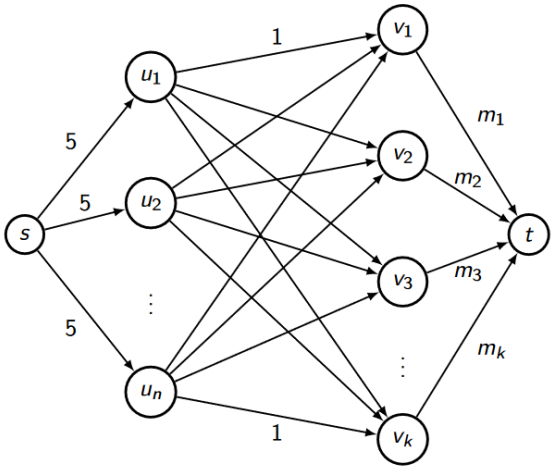


7 Movie Rental

Suppose you have a movie rental agency. At the moment you have k movies in stock, with m_i copies of movie i. There are n customers, who have each specified which subset of the k movies they are willing to see. However, no customer can rent out more than 5 movies at a time. Design an algorithm which runs in $O(n^2k)$ time and dispatches the largest possible number of movies.

Graph Construction.
Construct a flow network with:

- source s and sink t,
- a vertex U_i for each customer i and a vertex v_j for each movie j,
- for each i, an edge from s to u_i with capacity 5,
- for each customer i, for each movie j that they are willing to see, an edge from u_i to v_j with capacity 1.
- for each edge j, an edge from v_j to t with capacity m_j.



Since the customer-movie edges are of capacity 1, we interpret a flow of 1 along these edges as assigning a movie j to customer i. We also ensure that each customer receives only movies they want to see by these same edges.

By **flow conservation**, the amount of flow sent along the edge from s to u_i is equal to the total flow sent from u_i to all movie vertices v_j, so it represents the number of movies received by customer i. Again, the capacity constraint ensures that this not exceed 5 as required and the movie stock m_j is also represented. Therefore the max flow corresponds to a valid allocation of movies to customers.

Algorithm.
To maximise the movies dispatched, we find the max flow using the Edmonds-Karp algorithm. There are $n+k+2$ vertices and up to $nk+n+k$ edges, so the time complexity is $O((n+k+2)(nk+n+k)^2)$. Since the value of any flow is constrained by the total capacity s (the number of customers $\times 5$), we can achieve a tighter bound of $O(E|f|) = O(n(nk+n+k)) = O(n^2k)$

8 Assigning Schools

You are the mayor of your local town. In this town, there are m schools and n students requiring a school. Each student has a proximity range of schools that they can apply to ans, since you value your education, you require that every student is assigned a school. However, you want to assign students to a school such that the schools aren't too overloaded. The load of a school is the number of students assigned to the school. Design an $O(mn^2 \log n)$ algorithm that finds a feasible allocation of students and schools while minimising the maximum load of any particular school, or return that such an allocation is not possible.

Graph Construction.
Construct a flow next with:

- source vertex s and sink vertex t
- a vertex x_i for each student
- a vertex y_j for each school
- for each student i, an edge (s, x_i) with capacity 1, because every student can only be matched to 1 school
- for each school j, an edge (y_j, t) with k capacity, restricting the k students the school j can enroll
- for each school j that a student i can apply to, an edge (x_i, y_j) with capacity 1 because this is 1 assignment

Algorithm.
We can use a binary search on a range $[1, n]$ where n is the case where every student is allocated to 1 school. The search of this range of an element b in the search allows us to apply an Edmonds-Karp $O(E|f|)$ algorithm on the graph with $k = b$ to check whether that this k value is possible allocation. If the allocation is possible, search to the left and if it's not possible then search to the right. If no allocation is possible then there will be no solution where $k = n$.

We can do this because if the capacity of a school is k and is valid, then increasing capacity will be valid. If the capacity of a school is k and k is invalid, then decreasing the capacity will also be invalid. This is a monotonic behaviour and so we can apply the binary search.

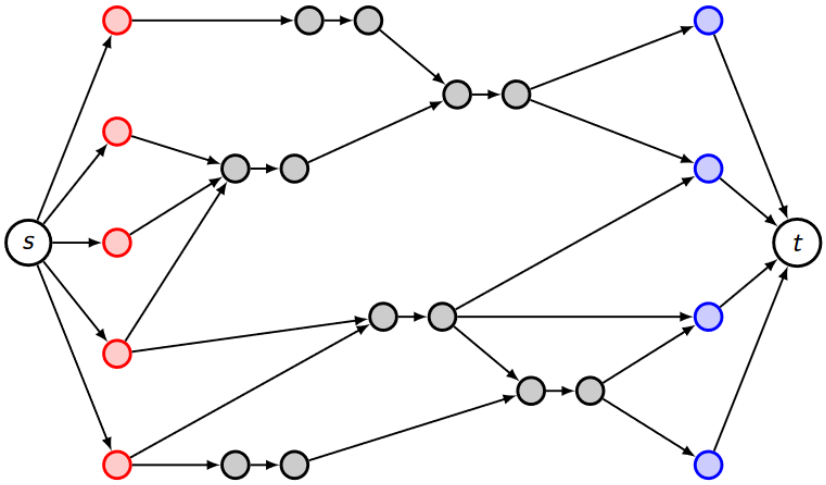
Time Complexity.
Binary searching the range in $O(\log n)$ time and run Edmonds-Karp in $O(mn^2)$ using the tighter bound of Edmonds-Karp with $mn + m + n$ edges and n flow. Final time complexity is $O(mn^2 \log n)$.

9 Vertex-Disjoint Path

You are given a directed graph G with n vertices and m edges. Of these vertices, r are painted red, b are painted blue, and the remaining $n - r - b$ are black. Red vertices have only outgoing edges and blue vertices have incoming edges. Design an algorithm which runs in $O(n(n+m))$ time and determines the largest possible number of vertex-disjoint paths in this graph, each of which starts at a red vertex and finishes at a blue vertex. To be clear, a vertex-disjoint path is a unique path that starts at a red vertex i and ends at a blue vertex j that passes through a black vertex x. Two paths that pass through the same vertex x are not considered different paths.

Graph Construction.
Construct a flow network with:

- super-source s joined to each red vertex,
- each blue vertex joined to super-sink t,
- for each black vertex, two vertices v_{in} and v_{out} joined by an edge,
- each edge of the original graph, with edges from a black vertex drawn from the corresponding out-vertex, and edges to a black vertex drawn to the corresponding in-vertex.
- all edges have capacity 1



A flow through an edge is interpreted as one of the red-blue paths. Flow conservation ensures that each unit of flow travels from s to a red vertex, then to zero or more black vertices, before arriving at a blue vertex and terminating at t. By the capacity constraint, each red vertex receives at most one unit of flow from s, so flow conservation ensures that at most edge from that vertex is flowed. Similarly, each blue vertex is used in at most one path also. Likewise, each in-vertex and out-vertex pair contributes to at most one path, so the path are indeed vertex-disjoint. Therefore, any flow in this graph corresponds to a selection of vertex-disjoint red-blue paths in the original graphs.

Algorithm.
To maximise the number of paths, we find the maximum flow using Edmonds-Karp algorithm. There are at most $2n$ vertices and exactly $n+m$ edges, so the time complexity is $O(n(n+m)^2)$. Since the value of any flow is constrained by the total capacities from s and to t, which in this case are r and b respectively, we can achieve a tighter bound of $O(E|f|) = O((n+m)min(r,b)) = O(n(n+m))$.

10 Bipartite Graphs

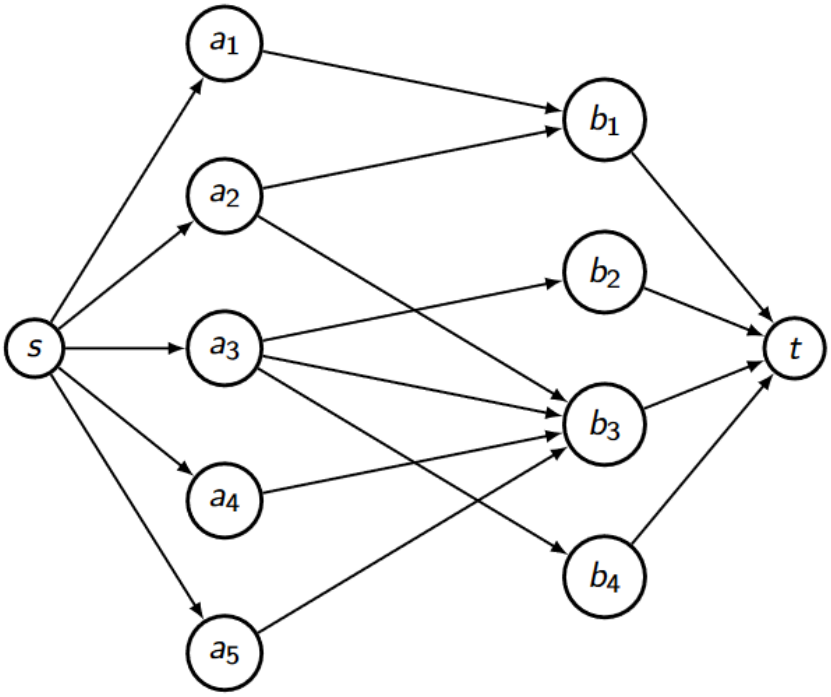
A graph $G = (V, E)$ is said to be bipartite if its vertices can be divided into two disjoint sets A and B such that every edge $e \in E$ has one end in the set A and the other in the set B.

Matchings.
A matching in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that each vertex of the graph belongs to at most one edge in M. A maximum matching in G is a matching containing the largest possible number of edges.

Maximum Bipartite Matching
Given a bipartite graph G, find the size in a maximum matching.

Graph Construction.

- Create two new vertices, s and t (the source and sink).
- Construct an edge from s to each vertex in A, and from each vertex in B to t.
- Orient the existing edges from A to B Assign capacity 1 to all edges.



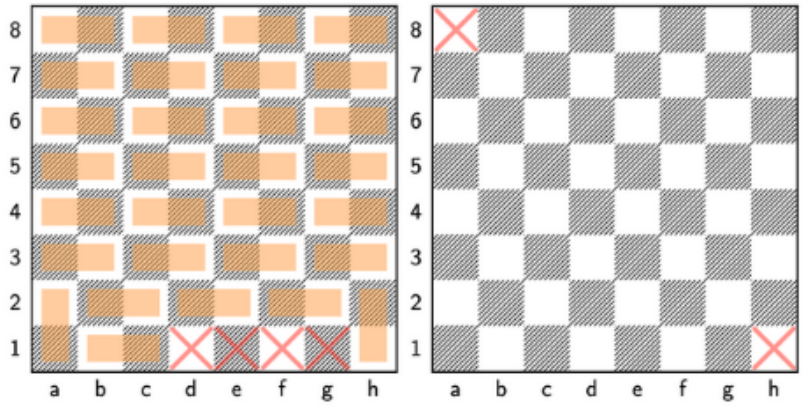
Recall that for each edge $e = (v, w)$ of the flow networks, with capacity c and carrying f units of flow, we record two edges in the residual graph:

- an edge from $v \to w$ with capacity $c - f$
- an edge from $w \to v$ with capacity f

Since all capacities in the flow network are 1, we need to only denote the direction of the edge in the residual graph! As always, the residual graph allows us to correct mistakes and increase the total flow.

11 Chess board

You were gifted a supple of dominoes and an $n \times n$ chessboard for Christmas. Unfortunate, some of the squares on the chessboard have been removed. A domino covers two adjacent squares and the squares alternate in colour. Your goal is to find a way to cover the chessboard with dominoes, or return that it is not possible. For example, the left board can be tiled and the second board cannot.



We observe that this problem can be modelled by matchings in a bipartite graph since we can partition the vertices into black and white squares and dominoes must cover adjacent black and white squares. We also know that black and white squares will never be next to each.

Graph Construction.
Construct the following flow network graph:

- A source vertex s and sink vertex t
- Vertices b on one side representing black squares and vertices w on the other side representing white squares. Connect these to the source and sink respectively.
- Construct edges from b_i to white vertices w representing a black square i that is adjacent to set of white vertices. Omit constructing the edges to removed squares of the chessboard.

Algorithm.
Run the Edmonds-Karp algorithm with $n^2 - k$ vertices where k is the amount of removed squares. This means we need exactly $\frac{(n^2-k)}{2}$ dominoes. If our max flow is not this, then a valid tiling can not exist. There is a bound of $\frac{n^2}{2}$ vertices and so we apply the $O(E|f|)$ bound to achieve a time complexity of $O(n^2 \times n^2) = O(n^4)$ (in the case of no missing tiles).