1 Matching in General Graphs

For the most part, we've discussed matching restricted to bipartite graphs. We're going to generalize it now to general graphs. First define the function $o(G)$ as the number of **odd connected components** in G. Odd connected components have an odd number of vertices. In order for a graph to have a perfect matching, we'll use what could be loosely be considered as a generalization of Hall's Condition.

Tutte's Theorem states that a graph G with a perfect match satisfies the inequality $\forall S \subseteq V(G) : o(G - S) \leq |S|$. Formally, a graph $G = (V, E)$ has a perfect matching if and only if for every possible vertex set $S \subseteq V(G)$, the subgraph induced by $V - S$ has at most $|S|$ connected components with an odd number of vertices.

We can prove this theorem using a similar approach as we did with Hall's.

2 Edmond's Blossom Algorithm

As Berge's Theorem stated, a matching M on graph M is a maximum match if and only if there are no M-augmenting paths in G. As this condition is not restricted to bipartite graphs, we can again use the idea to create a maximum match on a non-bipartite graph, or graph with odd cycles. The general idea is to again use a modified BFS from an unsaturated vertex to identify an M-augmenting path. The outer loop of the algorithm is the same, where we iteratively increase the size of a match M through finding successive M-augmenting paths of increasing length.

```
procedure MatchGraph(Graph G)
    M ← ∅                                   ▷ Match M initially empty
    do
        P ← BlossomAlg(G, M)                ▷ New augmenting path found with M, G
        M ← MΔP                             ▷ Symmetric difference between M, P
    while P ≠ ∅
    return M
```

However, the method in which we find them is not quite so simple. The main idea is to start the modified BFS from an unsaturated vertex v and follow successive unmatched then matched edges creating a tree of M-alternating paths from v. For each new edge examined from vertex u that was previously placed in the BFS queue Q, there are 4 possibilities for the co-adjacent vertex w:

1. w is an unsaturated vertex: Then we have found an augmenting path from v to w. We return the path.

2. w is already matched and not contained in the BFS tree T so far: Then we add w and its matched neighbor x to T and put x into the BFS queue Q_n.

3. w is already contained in T and we have detected a cycle of odd length via a non-tree edge: We identify the cycle as the the shortest path from w to u and perform a *contraction* on the entire cycle.

4. w is already contained in T and we have detected a cycle of even length via a non-tree edge: We do nothing in this instance. Even cycles can't help us.

A contracted odd cycle in T is referred to as a **blossom**, hence the name of the algorithm, **Edmonds' Blossom Algorithm**. An odd cycle can be traversed in either direction from v, which allows us to explore any adjacent unsaturated edge. Note in item 2 above: to enforce T as being M-alternating, we initially only place in Q_n the second vertex x of a newly discovered match (w, x) for further exploration. By being able to reach w via x by traversing an odd cycle containing them both, we break this restriction and can now also explore edges from w.

```
procedure BlossomAlg(Graph G(V, E) and matching M(V_M, E_M))
    G' ← G
    for all v ∈ V do
        marked(v) ← false
    for all v ∈ V : v ∉ V_M, marked(v) = false do
        marked(v) ← true
        Q ← v, Q_n ← ∅
        T ← v                                              ▷ Modified BFS tree
        while Q ≠ ∅ do
            for all u ∈ Q do
                for all w ∈ N'(u) do
                    if w ∉ V_M then                        ▷ w is unsaturated
                        return shortestPath(T, u, v) +(u, w)
                    else if marked(w) = false then         ▷ Not in T yet
                        x ← y ∈ N(w) : (y, w) ∈ E_M        ▷ Vertex w's match
                        marked(w), marked(x) ← true
                        T ← T + (u, w) + (w, x)
                        Q_n ← x
                    else if abs( level(T, u) − level(T, w)) mod 2 = even then
                            ▷ Even level difference on non-tree edge means blossom found
                        B ← shortestPath(T, w, u) + (w, u)
                        G' ← G' · B                        ▷ Contraction of blossom
                        T ← T · B
                        b ← contracted blossom vertex
                        marked(b) ← true
                        Q_n ← b
    return ∅
```