Chapter D1: Maximum Matchings

A matching in a graph is a collection of edges such that no pair of edges share a common node. A matching with the most edges is known as a maximum matching.

This question comes up in scheduling problems. Suppose we have a school and we list all the classes occurring at a particular time as one set of nodes, and all the venues available at that time as another set of nodes, and join two nodes if the corresponding venue is suitable for the corresponding class. The result is a graph. If there is a matching that uses all the classes, then a schedule for that time is possible.

D1.1 Trees

An algorithm for maximum matching in trees is the following. A leaf-edge is an edge whose one end has no other neighbor. The greedy algorithm is to repeatedly take any leaf-edge. (Recall that a forest is a collection of trees.)

```
TreeMatch(F:forest)
M ← []
while F nonempty do {
  select any leaf-edge e
  M ← M + [e]
  F ← F − both ends of e
}
```

Example. In the following picture, a maximum matching can be obtained by pairing BE, then CG, then AD.

```
Example. In the following picture, a maximum matching can be obtained by pairing BE, then CG, then AD.
```

Why does the greedy algorithm work? Well, assume e is a leaf edge and consider any maximum matching N. Suppose N does not contain e. Then if we add e to N, only one node now has two edges incident with it. So we can delete one of the edges of N and
attain a maximum matching containing $e$. And so on. Hence at each stage, the partial
matching $M$ is promising.

One can implement this algorithm using a postorder traversal: every time one visits a
node, if there is a child available then one pairs the node with one of its children.

D1.2 Maximum Matching in General Graphs

We now consider the maximum matching problem in a general graph. The first polynomial-
time algorithm for this problem was given by Edmonds. We give some of the ideas. We
complete the algorithm for the case that the graph is “bipartite”, defined later.

Let $M$ be a matching. A path is called alternating (with respect to $M$) if the edges
alternate being in $M$ and out of $M$. A path is called augmenting if it is alternating
and neither end-node is incident with an edge of $M$.

Example. Here’s a graph, a matching $\{BC, EF\}$, and an augmenting path
is $ACBEGF$.

\[
\begin{array}{c}
A \\
B \\
C \\
D \\
E \\
F \\
G \\
H
\end{array}
\]

We also need the concept of the symmetric difference of two sets: this consists of
those elements in precisely one of the two sets. That is, the symmetric difference of $E$
and $F$ is given by

$$E \triangle F = (E - F) \cup (F - E).$$

Before reading further, explain to yourself what the symmetric difference of two match-
ing can look like. The following result is due to Berge.

Lemma. A matching is maximum if and only if there exists no augmenting path.

Proof. We show: (1) if there is an augmenting path, then one can increase the matching;
and (2) if the matching is not largest, then there is an augmenting path.

(1) Suppose $M$ is a matching and there is an augmenting path $P$. Then let $M' = M \triangle P$.
In other words, delete from $M$ those edges lying in $P$ and add back in those edges in $P$
which didn’t lie in $M$. The result is a matching. The matching $M'$ has one more edge
than $M$. For example, consider the following picture of $P$: [Diagram]
(2) Suppose $M$ is a matching and there is a larger matching $N$. Consider the graph $H = M \Delta N$. Each node of $H$ is incident with at most two edges, and if it is incident with two edges then one edge is from $M$ and one is from $M$. This means that $H$ consists of cycles and paths. Furthermore, a cycle in $H$ must have an even number of edges: they alternate between $M$ and $N$.

Now, since there are more edges in $N$ than in $M$, in $H$ there must be more edges of $N$ than of $M$. In an even cycle, the edges of $M$ and $N$ alternate; so they are equinumerous. Hence there must be a path $P$ in $H$ in which $N$ is in the majority. But think about the path $P$: it must be augmenting with respect to $M$! 

This lemma gives an iterative algorithm for finding a maximum matching in a graph.

```
MaximumMatching (G:graph)
M ← []
repeat
    find augmenting path P
    if found then M ← M \triangle P
until path not found
```

There’s only the “small” problem of finding an augmenting path! We only discuss one important special case.

### D1.3 Maximum Matching in Bipartite Graphs

A graph is **bipartite** if the nodes can be colored with two colors such that nodes connected by an edge receive different colors. Equivalently, the nodes can be partitioned into two sets $X$ and $Y$ such that every edge has one end in $X$ and one end in $Y$. The graph of the scheduling problem described above is bipartite: the classes form one of the sets of nodes and the venues form the other set of nodes.

The problem we face is to find an augmenting path in a bipartite graph. Let $\hat{G}$ be the directed graph that results when one orients all edges of $M$ from $Y$ to $X$ and orients all other edges from $X$ to $Y$. Note that every possible path (respecting the orientations) in $\hat{G}$ must alternate between edges in $M$ and edges not in $M$. We then use a BFS:
FindAugmentingPath (bipartite graph $G$)

for each node $x$ in $X$ not incident with edge of $M$ do {
    perform BFS from $x$
    if reach node $y$ in $Y$ not incident with edge of $M$
        then stop: path is found
}

Now, every augmenting path has by definition an odd number of edges and so must have one end in $X$ and one end in $Y$. Thus this process cannot miss an augmenting path. Furthermore, by the nature of BFS, every node we find in the BFS search is connected back to $x$ by a path that does not repeat nodes.

Now, there is trivial speed up: one can do the BFS for all such $x$ simultaneously. (Equivalently, one can add a node $s$ that is joined to all the “free” vertices of $X$ and a node $t$ that is joined to all the “free” vertices of $Y$ and then do a BFS from $s$ to try to get to $t$.)

Putting all this together, there result is an $O(na)$ time algorithm for maximum matching in bipartite graphs, where $a$ is the number of edges. This running time can be improved by noting that maximum matching in bipartite graphs is a special case of the network flow problem on bipartite graphs; see section on “network flows” (to be added one day . . .

Exercises

1. Show that if a graph is bipartite then it cannot contain a cycle with an odd number of node. (The converse is also true.)

2. A tree is always bipartite. Construct a linear-time algorithm for finding a 2-coloring of a tree.

3. Illustrate the steps of the maximum matching algorithm on the following graph.

![Graph Diagram]