Part III
Bipartite Matching Algorithms

1 The graph matching problem (p. 38)

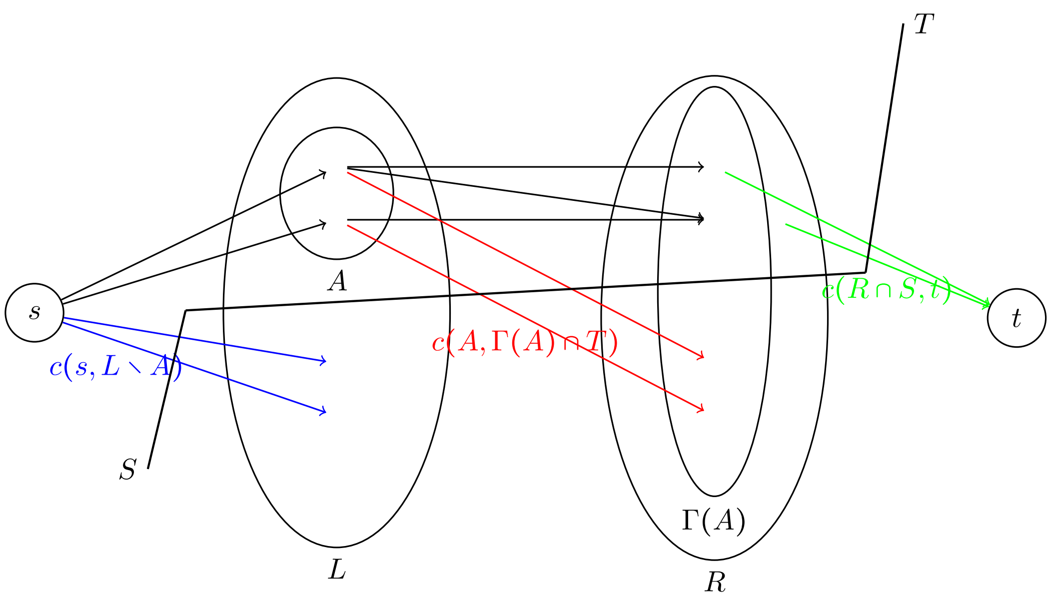
2 A network flow solution (p. 38)

2.1 Matching problems (p. 40)

1. Let $G = (V, E)$ be a bipartite graph with $|L| = |R|$. Prove Hall's theorem which states that a perfect matching $|M|$, i.e., one with $|M| = |L|$, exists if and only if for all $A \subseteq L$, $|A| \leq |\Gamma(A)|$ where $\Gamma(A)$ are all the neighbors of A in R. Make use of the min-cut max-flow theorem.

Solution: $\underline{\Rightarrow}$: Solution 1): We assume there is a perfect matching M with $|M| = |L| = |R|$. Take an arbitrary subset $A \subseteq L$ and let the set of nodes in matching with those in A be $B \subseteq \Gamma(A) \subseteq R$: $u \in A, v \in B \implies (u, v) \in M$. By definition of a matching (edges are vertex disjoint), we must have $|A| = |B| \leq |\Gamma(A)|$, which proves one direction.
Solution 2): you can show this through a simple contraposition (extra).

$\underline{\Leftarrow}$: Solution 1): We assume an **arbitrary** cut S, T in the corresponding max-flow/min-cut problem. It is possible that this is not a minimal cut. We want to prove $c(S, T) \geq |L|$.



Let $A = S \cap L$. We have (see picture above) $c(S, T) = c(s, L \setminus A) + c(A, \Gamma(A) \cap T) + c(R \cap S, t)$. Note that as capacity of every edge is 1, $c(A_1, A_2)$ equals to the number of edges from A_1 to A_2 for any disjoint $A_1, A_2 \subset V$. Therefore $c(s, L \setminus A) = |L \setminus A|$, $c(A, \Gamma(A) \cap T) \geq |\Gamma(A) \cap T|$ and $c(R \cap S, t) \geq c(\Gamma(A) \cap S, t) = |\Gamma(A) \cap S|$. Thus

$$c(S,T) \geq |L \setminus A| + |\Gamma(A) \cap T| + |\Gamma(A) \cap S| = |L \setminus A| + |\Gamma(A)| \geq |L \setminus A| + |A| = |L|.$$

The max-flow min-cut theorem states that the value of the maximum flow f^*, $|f^*| = \min_{\text{cuts } S,T} c(S,T)$. From there and $c(S,T) \geq |L|\ \forall$ cuts S, T, it follows that

$$|L| \leq \min_{\text{cuts } S,T} c(S,T) = |f^*| = |M| \leq |L|$$

We thus have $|L| = |M|$ and the proof is finished.

Solution 2): Extra: Find a different proof, without using max-flow min-cut theorem (Hint: induction on $|L|$).

2. Use the previous exercise to show that if $G(V,E)$ is a bipartite graph with $|L| = |R|$ and each node has exactly $d \geq 1$ neighbors, then there exists a matching M with $|M| = |L|$.

Solution: Solution 1): Let $A \subseteq L$ be arbitrary. We have

$$d\,|A| = c(A,R) = c(A,\Gamma(A)) \leq c(L,\Gamma(A)) = d\,|\Gamma(A)|,$$

where the capacities are measured in the corresponding flow network problem: the first and last equality are due to the fact that every vertex has d neighbours, the second equality is due to the fact that the edges between A and R are exactly those between A and $\Gamma(A)$ and the inequality is due to $A \subseteq L$. As $d \geq 1$, we therefore have $|A| \leq |\Gamma(A)|$. Using Hall's theorem finishes the proof.

Solution 2): The number of edges from A to $\Gamma(A)$ is $d|A|$. The edges from $\Gamma(A)$ to L are those from A to $\Gamma(A)$ and the edges between $L \setminus A$ and $\Gamma(A)$. Thus the number of edges from A to $\Gamma(A)$ is smaller than or equal to the number of edges from $\Gamma(A)$ to L (which is $d|\Gamma(A)|$). Thus $d|A| \leq d|\Gamma(A)|$.

Solution 3): One can also prove this by contradiction (exercise).

3. Consider a bipartite graph G with $|L| = |R| = n$ where each vertex has at least $n/2$ neighbors. Prove that a perfect matching M exists for G. ☆

Solution: Hint: what if $0 < |A| \leq n/2$? What if $|A| > n/2$?

4. Let M be a matching such that there exists no mathing M' with $M \subset M'$. Give an $O(|V| + |E|)$ algorithm to find such a matching M. Let M^* be a matching with $|M^*|$ maximized. Show that $|M^*| \leq 2|M|$ or give a counter example. ☆

5. A vertex cover of a bipartite graph $G = (V, E)$ with $V = L \cup R$ is a subset C of V such that for any $(u, v) \in E$ we have $u \in C$ or $v \in C$. Given a vertex cover C and the flow network used to find a maximum matching M in G. ☆
   (a) Show that there is a cut (S, T) with $c(S, T) = |C|$.
   (b) Conclude that the maximum flow is upper bounded by the minimum vertex cover size. Then $c(S, T) = |C|$ (draw a picture).

Part IV
Disjoint-Sets Data Structures

1 Disjoint-sets operations and the linked-list representation (p. 45)

1.1 Disjoint-sets and Linked lists: (p. 46)

1. Assume we do not append the shortest to the longer list, meaning we do not need to keep track of the length of the list. Give an example of m operations, with at most n MakeSet operations, that requires $\Theta\left(m^2\right)$ time.

Solution: One possible example is the following:

Operation	Time cost
MakeSet(1)	$n \times$ MakeSet
MakeSet(2)	
⋮	
MakeSet(n)	
Union(1,2)	1
Union(1,3)	2
⋮	⋮
Union(1,n)	$n - 1$

This sequence contains $m = 2n - 1$ operations. Each MakeSet (making a list containing 1 item) takes exactly 1 time step. The Union operations take a progressively longer time. Since we do not append the shortest to the longest, it is possible we have to modify the representative pointer of <u>each</u> of the longer list items, to point towards (the representative of) the shorter list. This longer list grows by 1 in each union.

Considering these remarks, the total time required is

$$n + \sum_{i=1}^{n-1} i = n + \frac{n(n-1)}{2} = \frac{n^2 + n}{2}$$

As $m^2 = (2n - 1)^2 = 4n^2 - 4n + 1$, these m operations require $\Theta(m^2)$ time.

2 Disjoint-sets forest (p. 46)

2.1 Disjoint-sets and forests (p. 49)

1. Suppose a tree in a disjoint-sets forest implementation contains 6 items. Draw all the possible structures of such a tree with and without path compression and explain how these trees can be constructed (by listing the operations used). ☆

2. Give an example of a sequence of operations in a disjoint-sets forest implementation such that a tree T_p becomes a child of T_r, while the longest path in T_p is larger than the longest path in T_r. ☆

   Solution: Hint: FindSet doesn't change the rank.

3. Bob and Alice play a game where Bob picks an integer $n \geq 1$ that Alice needs to guess. During each round Alice can write down as many guesses as she likes. However, if Alice writes more than n numbers in a round, Bob wins. After each round Bob indicates which of the numbers written down by Alice are smaller than n. If Alice wrote n as one of the guesses, Alice wins. Indicate how Alice can always win in $O(\log_* n)$ rounds. ☆

   Solution: Alice writes the following numbers until an upper bound on n is established:

Numbers	Round
1	1
$1, 2$	2
$1, 2, 4 = 2^2$	3
$1, 2, 4, 8, 16 = 2^4$	4
$1, 2, 4, 8, 16, 32, \ldots, 65336 = 2^{16}$	5
$1, 2, 4, 8, 16, 32, \ldots, 2^{65336}$	6
$\vdots$	$\vdots$

   If n is a power of 2 then obviously Alice wins. Otherwise, once we have an upper bound on n, as $n \in]2^m, 2^{m+1}[$ for some m, where 2^{m+1} was written down by Alice in the last round. Alice then writes all the natural numbers in that interval and wins in $O(\log_* n)$ rounds.

4. Suppose we replace the union-by-rank with a union-by-weight heuristic. In this case each set stores the number of elements in its set and when a union occurs the smaller set becomes a child of the larger set (breaking ties arbitrarily). Show that union-by-rank and union-by-weight are not equivalent. What is the worst case time complexity of a FindSet operation for the union-by-weight heuristic? ☆

Solution: Hint for the second part: How many vertices do you need to create a tree of depth k? Prove this by induction.

5. Given a graph $G = (V, E)$, when does the algorithm below return True? Should we use a linked-list or forest structure for its implementation?

```
for u ∈ V do
    MakeSet(u)
end for
Select random e' = (u', v') ∈ E; E' = E ∖ {e'}
while E' ≠ ∅ do
    for e = (u, v) ∈ E' do
        if FindSet(u) = FindSet(v) then Return False
        else
            if FindSet(u) = FindSet(u') then
                Union(FindSet(v),FindSet(v')); E' = E' ∖ e
            end if
            if FindSet(u) = FindSet(v') then
                Union(FindSet(v),FindSet(u')); E' = E' ∖ e
            end if
            if FindSet(v) = FindSet(u') then
                Union(FindSet(u),FindSet(v')); E' = E' ∖ e
            end if
            if FindSet(v) = FindSet(v') then
                Union(FindSet(u),FindSet(u')); E' = E' ∖ e
            end if
        end if
    end for
end while
Return True
```

Solution: The algorithm checks whether G is bipartite. However, if there exists edges in G not connected to the rest of the graph, the algorithm will never finish (E' will never be empty). Hence, the algorithm returns true if the graph is bipartite and there exists at most one SCC with more than one vertex. We should use linked-list: every time we execute a UNION operation, one of the sets is that of u' or v' and the other is a single vertex. Therefore, each UNION operation in the above algorithm thus takes $O(1)$ time. Every FINDSET operation also takes $O(1)$ time.

6. Suppose G is a graph with k connected components. Give a simple algorithm to determine these components. How often do we need to execute the MAKESET, FINDSET and UNION operation (as a function of k, $|V|$ and $|E|$)? ☆

Solution: *Note: you may assume that the graph is undirected.*

7. Show that any sequence of m MakeSet, FindSet and Union operations runs in $O(m)$ time if all the Union operations take place before the FindSet operations (and are executed on root elements), given that path compression and union by rank is used.

Solution: *This exercise requires knowledge of amortized cost, see chapter Fibonacci heaps.*