
# VE477 Introduction to Algorithms
# Homework 6

Taoyue Xia, 518370910087

2021/11/10

# Ex. 1 — Perfect matching in a bipartite graph

1. We denote $S_n$ to be all the permutations of $\{1, 2, \ldots, n\}$. Denote one of the permutations as $\sigma$, then the signature of $\sigma$ would be $+1$ if the interchanging of entries to meet the specific permutation can be performed in even number of times. And it will be -1 if interchanging can be performed in odd number of times. Then the product of $a_{1,\sigma_1}, a_{2,\sigma_2}, \ldots, a_{n,\sigma_n}$ can be expressed as:

$$\prod_{i=1}^{n} a_{i,\sigma_i}$$

Therefore, the final expression of calculating the determinant would be:

$$det(A) = \sum_{\sigma \in S_n} (\text{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i})$$

$\Rightarrow$ Since every vertex will be contained in one edge, it means that there will exist at most one $X_{i,j}$ in each row and column, with others all 0. Therefore, if the determinant of $A$ is identically zero, it means that there exists some row or column with all the elements being 0, so that all the product of permutations will give 0. In this sense, it tells that some node in $L$ or $R$ is not contained in edge in $G.E$. So we have got to the point that $G$ has no perfect matching.

$\Leftarrow$ If $G$ has no perfect matching, it means that at least one vertex in $L$ or $R$ are not contained in $G.E$, which will make one row or one column full of zeros. Therefore, since in all permutations, the product of all $a_{i,\sigma_i}$ is calculated, with one row or column with all the elements being 0, all the products of all permutations will give the answer 0. Therefore, by adding them together, the determinant will be identically zero. Proof done.

2. To decide whether a bipartite graph has a perfect matching, we just need to first construct an adjacency matrix $A$ of the graph, and set $X_{i,j}$ to be random positive integers. Then, if we can find some row or column full of zeros, it means that the graph does not contain a perfect matching. If we cannot find such row or column, it can be a perfect matching. The algorithm is shown below:

**Algorithm 1:** Perfect matching decision

**Input :** A bipartite graph $G = \langle V, E \rangle$, with $V = L \cup R$
**Output:** True or False

```
/* Denote L = {l_1, l_2, ..., l_n} and R = {r_1, r_2, ..., r_n}   */
/* Construct the matrix                                           */
```
$A \leftarrow$ a two-dimension array acting as a matrix;
**for** $i = 1$ *to* $n$ **do**
  **for** $j = 1$ *to* $n$ **do**
    **if** $(l_i, r_j)$ *is in* $E$ **then**
      $x \leftarrow$ a random positive integer;
      $A[i][j] \leftarrow x$;
    **else**
      $A[i][j] \leftarrow 0$;
    **end**
  **end**
**end**

```
/* Check empty row or column                                      */
```
$countRow \leftarrow 0$;
$countColumn \leftarrow 0$;
**for** $i = 1$ *to* $n$ **do**
  **for** $j = 1$ *to* $n$ **do**
    **if** $A[i][j] = 0$ **then**
      $countRow \leftarrow countRow + 1$;
    **end**
    **if** $A[j][i] = 0$ **then**
      $countColumn \leftarrow countColumn + 1$;
    **end**
  **end**
  **if** $countRow = n$ *or* $countColumn = n$ **then**
    **return** *False*;
  **end**
  $countRow,\ countColumn \leftarrow 0$;
**end**
**return** *True*;

3. Suppose that there are $n$ vertices in both $L$ and $R$, then the time complexity of the previous algorithm is $\mathcal{O}(n^2)$. Error will occur when the determinant calculated is 0, but there exists no row or column to be all zero.

4. Using Ford-Fulkerson Algorithm we can simply determine whether it has a perfect matching in $\mathcal{O}(VE)$ time. However, this strategy uses the concept of an adjacency matrix, and by calculating its determinant or checking the rows and columns, we can reach the goal. It is more like a linear algebra solution, which is more graphic.

# Ex. 2 — Critical thinking

1. Two different ways will be explained in the following two algorithms.

   - One is to use two pointers staring from the beginning node, with one moving twice as fast as the other one. When the faster pointer reaches the end or NULL, the slower one will point to the middle.

**Algorithm 2:** Faster and slower pointer

**Input :** A single linked list $l$
**Output:** The middle node $m$

```
1  f ← l.first;
2  m ← l.first;
3  n ← l.length;
4  for i = 1 to n do
5  |  if f = NULL or f.next = NULL then
6  |  |  break;
7  |  end
8  |  f ← f.next.next;
9  |  if f = NULL then
10 |  |  break;
11 |  end
12 |  m ← m.next;
13 end
14 return m;
```

   - The other one can be realized by move and stop one pointer, while making the other one move always in a loop. The algorithm is shown below:

**Algorithm 3:** Move and Stop

**Input :** A single linked list $l$
**Output:** The middle node $m$

```
1  f ← l.first;
2  m ← l.first;
3  n ← l.length;
4  move ← False;
5  for i = 1 to n do
6  |   f ← f.next;
7  |   if f = NULL then
8  |   |   break;
9  |   end
10 |   if move = True then
11 |   |   m ← m.next;
12 |   end
13 |   move ←!move;
14 end
15 return m;
```

2. To detect a loop, we just need to work like the previous one, but stop when the current node is NULL, or the fast and slow pointer point to the same node.

**Algorithm 4:** Fast and Slow

**Input :** A single linked list $l$
**Output:** True or False whether it has a cycle

```
f ← l.first;
m ← l.first;
n ← l.length;
for i = 1 to n + 1 do
    if f = NULL or f.next = NULL then
        return False;
    end
    if f = m then
        return True;
    end
    f ← f.next.next;
    m ← m.next;
end
return False;
```

This algorithm only needs $\mathcal{O}(n)$ time complexity to determine whether the list contains a cycle, since it just needs to traverse one time in the single linked list, and will find a collision of the faster and slower pointer or no collision in at most $n + 1$ loops.

# Ex. 3 — The coupon collector desillusion

1. At least $n$ boxes should be bought, if they are all attached with a different coupon.

2. From the definition, we can deduce that $X = X_1 + X_2 + \cdots + X_n$. The probability of collecting a distinctive new coupon from the obtained $j - 1$ coupons is $P_j = \frac{n-j+1}{n}$. So $X_j$ has a geometric distribution, thus its expectation can be expressed as:

$$E[X_j] = \frac{1}{P_j} = \frac{n}{n - j + 1}$$

3. From the above conclusion, we can express the expectation of $X$ as:

$$\begin{aligned} E[X] &= E[X_1 + X_2 + \cdots + X_n] \\ &= E[X_1] + E[X_2] + \cdots + E[X_n] \\ &= P_1 + P_2 + \cdots + P_n \\ &= \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} \\ &= n \cdot (1 + \frac{1}{2} + \cdots + \frac{1}{n}) \\ &\approx n \cdot \int_1^n \frac{1}{x} dx \\ &= n \log n \end{aligned}$$

Also, since $H_n = \sum_{i=1}^{n} \frac{1}{i}$ is a harmonic number, we have an equation that:

$$H_n = \log n + \gamma + \frac{1}{2n} - \sum_{k=1}^{\infty} \frac{B_{2k}}{2kn^{2k}}, \quad \text{where } B_k \text{ are Bernoulli numbers}$$

And $\gamma \approx 0.5772156649$ is the Euler–Mascheroni constant

So we will have:

$$E[X] = n \log n + n\gamma + \frac{1}{2} + \mathcal{O}(1/n)$$

Therefore, $E[X] = \Theta(n \log n)$.

4. From the previous formula, we can obviously see that when you have more distinctive coupons, the expected boxes you need to buy to get a new one will increase. Namely, for all $1 \leq i < j \leq n$, $E[X_i] < E[X_j]$. And the total boxes needed to buy for collecting all the coupons are about $n \log n$.