$$
\begin{aligned}
& l(x_4)+l(y_2)-w(x_4,y_2), l(x_4)+l(y_4)-w(x_4,y_4)\Big\} \\
= \quad & \min\Big\{2+0-0, 2+0-0, 3+0-1, 3+0-0, 4+0-0, 4+0-0\Big\} \\
= \quad & \min\Big\{2,2,2,3,4,4\Big\} \\
= \quad & 2.
\end{aligned}
$$

The minimum is achieved for three pairs, so the three edges (x_1,y_2), (x_1,y_4), and (x_2,y_2) are added to the equality graph after the relabeling of the vertices. Now, we have $S=\{x_1,x_2,x_4\}$, $T=\{y_1,y_3\}$, and $N_l(S)=\{y_1,y_2,y_3,y_4\}$. We first pick $y_2 \in N_l(S)\setminus T$ which is matched to x_3, so we update $S=\{x_1,x_2,x_3,x_4\}$ and $T=\{y_1,y_2,y_3\}$. Since both (x_1,y_2) and (x_2,y_2) are added, y_2 can be viewed as the child of either x_1 or x_2 in the alternating tree. Finally, we choose $y_4 \in N_l(S)\setminus T$, and notice that y_4 is free. Tracing the alternating tree gives the augmenting path x_2,y_1,x_1,y_4, and the matching augments to $\{(x_1,y_4),(x_2,y_1),(x_3,y_2),(x_4,y_3)\}$. This is a perfect matching in G_l, and corresponds to a maximum matching in G of total weight $0+4+6+4=14$.

Let us now analyze the performance of the Hungarian algorithm on a general bipartite graph with $|X|=|Y|=n$. First, let me argue that the algorithm terminates. That is, I want to show that so long as M is not perfect, we can eventually discover an augmenting path. If $N_l(S)\neq T$, we can grow the alternating tree, and the process cannot proceed indefinitely because there are only finitely many vertices. If we reach the state $N_l(S)=T$, we can improve the labeling to introduce new edges in G_l enlarging the neighborhood $N_l(S)$. Since Y contains free vertices, (at least) one such vertex must eventually end up in $N_l(S)$.

I now show that each augmentation of the matching can finish in $O(n^2)$ time. For each $y \in Y\setminus T$, define the quantity

$$\delta_y = \min\Big\{l(x)+l(y)-w(x,y) \mid x \in S\Big\}.$$

All δ_y can be initially computed in $O(n^2)$ time.

In each step of growing the alternating tree, we add a new vertex to S. The δ_y value for each $y \in Y\setminus T$ can be updated in $O(1)$ time. Since $|Y\setminus T| \leqslant n$, and we can add at most n vertices to S, the total effort associated with growing the alternating tree is $O(n^2)$.

Let us now look at the relabeling stage. Since each relabeling introduces at least one new vertex of Y to $N_l(S)$, at most n relabeling stages are necessary. During each relabeling, one can compute $\alpha_l = \min\{\delta_y \mid y \in Y\setminus T\}$ in $O(n)$ time. After the relabeling, we subtract α_l from δ_y for all $y \in Y\setminus T$, again in $O(n)$ time. Therefore each relabeling can be done in $O(n)$ time.

It follows that each augmentation of M (by an edge) can be completed in $O(n^2)$ time. We start with the empty matching, and eventually arrive at a perfect matching having n edges, so the overall running time of the Hungarian algorithm is $O(n^3)=O(|V|^3)$.

21.6.4 Stable matching

Let $G=(V,E)$ be a complete bipartite graph with n vertices in each of the parts M and W. There are $n!$ perfect matchings possible for this graph, because any permutation of the vertices in W can be matched with the vertices of M.

Suppose that M represents a set of n men, and W a set of n women. They want to get engaged and marry. Each man has an order of preferences for the women in W. Likewise, each woman has an order of preferences for the men in W. A perfect matching N in G prescribes a set of n engagements for the men and the women. Take a pair $(m,w) \in E \setminus M$. Let (m,w') and (m',w) be chosen in N, where $m' \neq m$ and $w' \neq w$. The pair (m,w) is called *unstable* if *both* the following conditions hold.

(1) m prefers w to w'.
(2) w prefers m to m'.

A perfect matching N is called a *stable matching* if there are no unstable pairs for this matching. The problem of finding a stable matching is called the *stable matching problem* or the *stable marriage problem*. In what follows, I explain the *Gale–Shapley algorithm* to solve this problem.

The basic idea is that so long as a man is not engaged, he keeps on proposing to the women in the order of his preferences. A woman not yet engaged always accepts a proposal. However, if a proposal is made by m to a woman w already engaged to m', w checks which one of m and m' has higher preference for her. If m has higher preference, she changes her engagement with m' to a new engagement with m (so m' becomes not engaged again). On the other hand, if m' has higher preference than m, w rejects the proposal of m (so m continues to remain not engaged). The process stops when all men and women are engaged.

The engagements made during the running of the algorithm are provisional. Women accept proposals from men of higher preferences. Eventually, when all men and women are engaged, we obtain the final matching. The algorithm is more explicitly stated now.

```
Start with the empty matching N.
For each man, set the next-to-propose woman to the woman of his highest preference.
So long as N contains less than n edges, repeat:
    Take a man m not engaged (a vertex of M not matched by N).
    Let w be the next-to-propose woman for m.
    If w is not engaged (not matched by N), add the pair (m,w) to N.
    else do the following:
        Let (m',w) ∈ N currently.
        If m has higher preference than m' to w, replace (m',w) by (m,w) in N,
        else keep N as it is.
    Update the next-to-propose woman for m to his next preference.
Return N.
```

Before the analysis of the Gale–Shapley algorithm, let me furnish an example illustrating the working of the algorithm. Take $n = 4$. The men are named A,B,C,D, and the women E,F,G,H. To start with, the orders of preferences of the men and women are given. In order that the comparison of m with m' can be made by w in constant time, the preference lists for women should be maintained as illustrated in the third table below.

	1	2	3	4
A	E	H	F	G
B	H	G	E	F
C	E	H	F	G
D	H	F	E	G

Men's choices

	1	2	3	4
E	D	C	B	A
F	C	B	A	D
G	B	D	C	A
H	B	C	A	D

Women's choices

	A	B	C	D
E	4	3	2	1
F	3	2	1	4
G	4	1	3	2
H	3	1	2	4

Women's choices

We assume that the turns of the men come in the round-robin fashion. If so, the algorithm works as given in the following table. The matching N is shown only when it changes. Notice how A ends up with his last choice. G also gets her last preference. Feel sorry for A and G. E and F can improve upon their initial provisional engagements. H luckily gets her first choice at the beginning, and rejects all other proposals.

Man's action	Woman's reaction	Current matching (N)
–	–	$\emptyset$
A proposes to E	E accepts A	$\{(A,E)\}$
B proposes to H	H accepts B	$\{(A,E),(B,H)\}$
C proposes to E	E replaces A by C	$\{(B,H),(C,E)\}$
D proposes to H	H rejects D	
A proposes to H	H rejects A	
B is already engaged		
C is already engaged		
D proposes to F	F accepts D	$\{(B,H),(C,E),(D,F)\}$
A proposes to F	F replaces D by A	$\{(A,F),(B,H),(C,E)\}$
B is already engaged		
C is already engaged		
D proposes to E	E replaces C by D	$\{(A,F),(B,H),(D,E)\}$
A is already engaged		
B is already engaged		
C proposes to H	H rejects C	
D is already engaged		
A is already engaged		
B is already engaged		
C proposes to F	F replaces A by C	$\{(B,H),(C,F),(D,E)\}$
D is already engaged		
A proposes to G	G accepts A	$\{(A,G),(B,H),(C,F),(D,E)\}$

Let us now look at the correctness of the algorithm. First, observe that if a woman is already engaged, she continues to remain so, but she can improve her partner. On the contrary, a man having a provisional engagement may become not engaged in future. We first need to argue that no man runs out of options for proposing. If that happens for m, this means that he has proposed to all of the n women, and has been either rejected or replaced by each of them. Since n women cannot be engaged to $n-1$ men (m is excluded), there must exist a woman w not engaged. But m has proposed to all of the n women and, in particular, to w. Since w is not engaged, she must have accepted m, a contradiction. This argument also establishes that the size of the matching eventually reaches n, that is, the matching N produced by the Gale–Shapley algorithm is a perfect matching.

Next, we need to show that N is a stable matching. Choose any pair $(m,w) \in E \setminus M$. Let N contain the pairs (m,w') and (m',w) (N is a perfect matching). I show that at least one of the conditions (1) and (2) for an unstable pair does not hold for (m,w). Consider the two cases.

Case 1: m has never proposed to w.

Since m is eventually engaged with w', he must have proposed to w' at some point of time. Since men propose in the order of their preferences, w' has a higher preference to m than w. So Condition (1) does not hold.

Case 2: m has proposed to w.

If the decision of w is to reject m, she must be already engaged to a man of higher preference. That partner of w may or may not be m'. If not, she eventually improves her partner further to m'. On the other hand, if m is provisionally engaged to w, the engagement of w eventually changes to m' (in one or more replace steps), because she gets men of higher preferences (than m) later. In either case, Condition (2) does not hold.

To sum up, the Gale–Shapley algorithm terminates after producing a perfect matching which is stable. Each proposal by a man can be handled in $O(1)$ time. There can be at most n proposals from each of the n men. Therefore the running time of the algorithm is $O(n^2)$.

21.7 Huffman codes

Data compression is an important and useful tool that finds immense applications in many areas including archiving and multimedia systems. Let us here plan to compress text data only. A text string is a finite sequence of symbols. It is customary to use binary encoding of the symbols occurring in strings. For example, ASCII is a 7-bit encoding of characters (Roman letters, numerals, and punctuation and control symbols).

In many cases, different symbols come in a string with different frequencies. For example, in an English text, the letter e usually comes with the highest frequency, whereas the letters q and z come with very low frequencies. In order to compress English text, it is, therefore, useful to have a short code for e and longer codes for q and z. Since e occurs more often than q or z, the overall bit length of the encoded message shrinks under this strategy.

An encoded message needs to be decoded unambiguously. A fixed-length encoding is always unambiguous. However, if we use variable-length codes for different symbols, we may run into trouble. As an example, let a, e, q and z have the codes 001, 10, 1101001 and 101101, respectively. Then, an encoded string, that starts with 101101001, may be decoded as 10 followed by 1101001 implying eq or as 101101 followed by 001 implying za.

In order to avoid this difficulty, we may design the encoding in such a way that no prefix of the code of a symbol equals the code for another symbol. An encoding satisfying this property is called a *prefix code*. Fixed-length encoding evidently yields prefix codes. There exist other schemes too that enjoy this property. Our task is to arrive at a prefix code for which the overall length of a string is minimum, that is, we achieve the maximum amount of compression.

Binary trees can be used to represent a prefix code. The leaves of the tree correspond to the symbols being encoded. In order to obtain the code for a symbol, we follow the unique path from the root to the leaf corresponding to that symbol. We start with the empty string. Whenever we follow a left link, we append a 0, and whenever we follow a right link, we append a 1. Eventually, we reach the desired leaf and declare the string, obtained by the traversal, as the code of the symbol. For example, see Figure 88(f) and (g). Let us calculate the code for b. The unique path from the root to the leaf labeled b consists of moving right first and subsequently moving left twice. Thus, b has the code 100.

A binary tree T representing a prefix code is called a *prefix tree*. Let $a_1, a_2, \ldots, a_n$ denote all the leaves of the tree. Each leaf a_i stands for a symbol denoted also as a_i. Let w_i denote the relative