2976. Minimum Cost to Convert String I

You are given two **0-indexed** strings `source` and `target`, both of length `n` and consisting of **lowercase** English letters. You are also given two **0-indexed** character arrays `original` and `changed`, and an integer array `cost`, where `cost[i]` represents the cost of changing the character `original[i]` to the character `changed[i]`.

You start with the string `source`. In one operation, you can pick a character `x` from the string and change it to the character `y` at a cost of `z` **if** there exists **any** index `j` such that `cost[j] == z`, `original[j] == x`, and `changed[j] == y`.

Return *the* ***minimum*** *cost to convert the string* `source` *to the string* `target` *using* ***any*** *number of operations. If it is impossible to convert* `source` *to* `target`, *return* `-1`.

Note that there may exist indices `i`, `j` such that `original[j] == original[i]` and `changed[j] == changed[i]`.

Example 1:

```
Input: source = "abcd", target = "acbe", original = ["a","b","c","c","e","d"], changed = ["b","c","b","e","b","e"], cost = [2,5,5,1,2,20]
Output: 28
Explanation: To convert the string "abcd" to string "acbe":
- Change value at index 1 from 'b' to 'c' at a cost of 5.
- Change value at index 2 from 'c' to 'e' at a cost of 1.
- Change value at index 2 from 'e' to 'b' at a cost of 2.
- Change value at index 3 from 'd' to 'e' at a cost of 20.
The total cost incurred is 5 + 1 + 2 + 20 = 28.
It can be shown that this is the minimum possible cost.
```

Example 2:

```
Input: source = "aaaa", target = "bbbb", original = ["a","c"], changed = ["c","b"], cost = [1,2]
Output: 12
Explanation: To change the character 'a' to 'b' change the character 'a' to 'c' at a cost of 1, followed by changing the character 'c' to 'b' at a cost of 2, for a total cost of 1 + 2 = 3. To change all occurrences of 'a' to 'b', a total cost of 3 * 4 = 12 is incurred.
```

Example 3:

```
Input: source = "abcd", target = "abce", original = ["a"], changed = ["e"], cost = [10000]
Output: -1
Explanation: It is impossible to convert source to target because the value at index 3 cannot be changed from 'd' to 'e'.
```

Constraints:

- `1 <= source.length == target.length <= 10`^{5}
- `source`, `target` consist of lowercase English letters.
- `1 <= cost.length == original.length == changed.length <= 2000`
- `original[i]`, `changed[i]` are lowercase English letters.
- `1 <= cost[i] <= 10`^{6}
- `original[i] != changed[i]`

2976. Minimum Cost to Convert String I

```
/*
   run over all letters + Dijkstra (TLE)
   Time complexity: O(nmlog26)=O(nm)
   Extra space complexity: O(26*2m+m+26+26)
*/
typedef std::vector<bool> vb;
typedef std::vector<long long> vi;
typedef std::pair<long long, long long> ii;
typedef std::vector<ii> vii;
typedef std::vector<vii> vvii;
class Solution {
   public:
      vvii graph;
   public:
      void build_graph(int m,std::string& source, std::string& target, vector<char>& original,
std::vector<char>& changed, std::vector<int>& cost){
         graph.resize(26);
         for(int i=0;i<m;++i){
            int u=original[i]-'a';
            int v=changed[i]-'a';
            int w=cost[i];
            graph[u].push_back({v,w});
         }
      }
```

```
void dijkstra(int start,int end,vi& distances,vb& visited){
    distances[start]=0;
    std::priority_queue<ii,vii,std::greater<ii>> q;
    q.push({0,start});
    while(!q.empty()){
        auto [cur_dis,u]=q.top();
        q.pop();
        if(u==end) break;
        if(visited[u]) continue;
        visited[u]=true;
        for(auto& [v,w]: graph[u]){
            if(distances[v]>distances[u]+w){
                distances[v]=distances[u]+w;
                q.push({distances[v],v});
            }
        }
    }
}
```

```
 long long minimumCost(std::string source, std::string target, vector<char>& original,
std::vector<char>& changed, std::vector<int>& cost) {
        int n=source.size();
        int m=original.size();

        build_graph(m,source,target,original,changed,cost);

        long long  ans=0;
        for(int i=0;i<n;++i){
           int start=source[i]-'a';
           int end=target[i]-'a';
           if(start==end) continue;
           vi distances(26,LLONG_MAX);
           vb visited(26,false);

           dijkstra(start,end,distances,visited);

           if(distances[end]!=LLONG_MAX) ans+=distances[end];
           else return -1;
        }

        return ans;
    }
};
```

2976. Minimum Cost to Convert String I

```
/*
   Just 26 letters(to avoid re-computation)+Dijkstra
   Time complexity: O(m+n)
   Extra space complexity: O(2*26+26m+26+m)
*/
typedef std::vector<bool> vb;
typedef std::vector<long long> vi;
typedef std::vector<vi> vvi;
typedef std::pair<long long, long long> ii;
typedef std::vector<ii> vii;
typedef std::vector<vii> vvii;
class Solution {
   public:
      vvii graph;
   public:
      void build_graph(int m,std::string& source, std::string& target, vector<char>& original,
std::vector<char>& changed, std::vector<int>& cost){
         graph.resize(26);
         for(int i=0;i<m;++i){
            int u=original[i]-'a';
            int v=changed[i]-'a';
            int w=cost[i];
            graph[u].push_back({v,w});
         }
      }
```

```
void dijkstra(int start,vi& distances,vb& visited){
        distances[start]=0;

        std::priority_queue<ii,vii,std::greater<ii>> q;

        q.push({0,start});

        while(!q.empty()){
           auto [cur_dis,u]=q.top();
           q.pop();

           if(visited[u]) continue;

           visited[u]=true;

           for(auto& [v,w]: graph[u]){
              if(distances[v]>distances[u]+w){
                 distances[v]=distances[u]+w;
                 q.push({distances[v],v});
              }
           }
        }
     }
```

```
long long minimumCost(std::string source, std::string target, vector<char>& original,
std::vector<char>& changed, std::vector<int>& cost) {
        int n=source.size();
        int m=original.size();

        build_graph(m,source,target,original,changed,cost);

        vvi distances(26,vi(26,LLONG_MAX));
        for(int u=0;u<26;++u) {
           vb visited(26,false);
           dijkstra(u,distances[u],visited);
        }

        long long  ans=0;
        for(int i=0;i<n;++i){
           int start=source[i]-'a';
           int end=target[i]-'a';

           if(distances[start][end]!=LLONG_MAX) ans+=distances[start][end];
           else return -1;
        }

        return ans;
    }
};
```