The assignment was to solve the Bridge and Torch problem by modeling it in `pddl` and finding the optimal plans for the given instances. I will summarize my solution for the domain and the instances and then I will show the found optimal plans with costs.

Domain

In the next few paragraph I will describe my model of the domain for the problem mentioned above.

Types

There are two types of objects that I decided to use, *island* and *guy*. There could also be an object *torch*, but since torch can belong to only a single guy at a time and a guy can have only a single torch, it is in my opinion better to model it as a predicate. Similar thing can be said about *bridge* and I also modeled it as a predicate taking two islands. Therefore, we have two types:

```
(:types island guy - object)
```

Functions

Next, I defined two functions, which represent the numerical values present in the problem. First of them is crossing cost of each guy. The value of this function is constant during the whole plan and is dependent on the instance. Also, we want to minimize the total cost of all the crossings in the plan, so we need a function which will hold this cost.

```
(:functions
    (crossing-cost ?g) - number
    (total-cost) - number
)
```

Predicates

Other properties of the domain are modeled as predicates.

Torches As mentioned before, we need a predicate to say, which guy(s) have a torch at the moment. The constraints for passing the torches between guys are enforced by the actions. Thus the predicate is just:

```
(has-torch ?g - guy)
```

Bridges Next we need to somehow connect the islands. I model it as a binary predicate, that there exists a bridge between two islands. In order to avoid disjunctions in the action preconditions, I opted for defining all bridges in both directions in the problem file. It makes the problem definitions longer and there is a little bit of redundancy, but I wanted to avoid potential problems with the planner.

```
(bridge ?i1 ?i2 - island)
```

Locations Another predicate is used to indicate at which island a given guy currently is located. This can be done by a binary predicate

```
(at ?g - guy ?i - island)
```

Comparing costs During modeling actions, I ran into an issue with comparing function values to determine the slower of two crossing guys. In documentation of `pddl` I found several options, including using conditional assignment combining `when` and `<` predicates. However, I was unable to force the fast-downard planner to use these predicates without throwing errors during parsing.

I could think of two options how to resolve this problem, one of those was defining the crossing costs for all pairs of guys where each pair would be assigned with the slower crossing speed. Second option was to define a predicate to compare which of the guys is faster and add it to preconditions of the action. The second option is shorter in the problem defintion but it adds a precondition to the action and it can lead to introduction of unnecessary passing of torches between guys. The first option does not add passing torches but is extremely long to define all pairs of guys in the problem file. Because passing of torches has no assigned cost and the definition is shorter, I chose the second option. Thus I defined one more predicate that says that `g1` is faster than `g2`:

```
(faster ?g1 ?g2 - guy)
```

Actions

I defined three actions in total, one of them is just to make the plan more readable and is not strictly necessary to find an optimal plan.

Passing the torch The simplest of the actions is passing a torch from one guy to another guy. The preconditions of this action are that the guys must be located at the same island and the first guy must have a torch and the second guy must not. The effect is then only changing the value of the `has-torch` predicate for both guys. As mentioned above, this action has no defined cost so we don't add anything to the `total-cost` function.

```
(:action pass-torch
    :parameters (?g1 ?g2 - guy ?i - island)

    :precondition (and
    (at ?g1 ?i)
    (at ?g2 ?i)
    (has-torch ?g1)
    (not (has-torch ?g2))
    )

    :effect (and
    (not (has-torch ?g1))
    (has-torch ?g2)
    )
)
```

Crossing the bridge The core action of the plan is crossing the bridge. The necessary preconditions are that both guys are located at the first island, there exists a bridge from the first to the second island and that the first guy has a torch. To make the plan more readable, I added a condition that the guys must be different and I added a crossing function for just a single guy but the same could be done without this condition and just one action. The only difference would be that the optimal plan would have actions where g1 would be the same as g2 which could be confusing a little bit. Also, I decided to add the before mentioned `faster` predicate to distinguish guy with slower crossing speed. In a case where the first guy would have a torch but slower crossing speed, they can just pass the torch between themselves, which does not add any cost to the plan. Or, we could add a disjunction but I wanted to avoid that as I explained before.

The effects are that we change the locations of both guys and increasing the `total-cost` function by the cost of the crossing for the second guy, which thanks to the `faster` precondition is the slower one.

```
(:action cross
    :parameters (?g1 ?g2 - guy ?i1 ?i2 - island)
    :precondition (and
        (not (= ?g1 ?g2))
        (at ?g1 ?i1)
        (at ?g2 ?i1)
        (bridge ?i1 ?i2)
        (has-torch ?g1)
        (faster ?g1 ?g2)
    )

    :effect (and
        (not (at ?g1 ?i1))
        (not (at ?g2 ?i1))
        (at ?g1 ?i2)
        (at ?g2 ?i2)
        (increase (total-cost) (crossing-cost ?g2))
    )
)
```

In the code I have another `cross` function but it is basically the same just for a single guy.

Problems

In previous section, I described the domain, now I can show the problems and solutions.

Problem 1

The simplest problem, where 4 guys are on the left island and must cross to the right island with a single torch. Thus there are 2 island and 4 guy objects:

```
(:objects
    l r - island
    a b c d - guy
)
```

The initial state is that all guys are located at the left island and the torch is with the guy *a*. Also, we must declare the bridge between the islands in both directions and crossing costs for each guy. Last, we need to add the `faster` predicate just for all pairs, where the first guy is faster than the second one.

```
(:init
    (bridge l r)
    (bridge r l)

    (has-torch a)

    (at a l)
    (at b l)
    (at c l)
    (at d l)

    (= (crossing-cost a) 1)
    (= (crossing-cost b) 2)
    (= (crossing-cost c) 5)
    (= (crossing-cost d) 8)
    (faster a b)
    (faster a c)
    (faster a d)

    (faster b c)
    (faster b d)

    (faster c d)
)
```

The goal is just that all guys are located at the right island. We don't care about the other predicates.

```
(:goal
    (and
        (at a r)
        (at b r)
        (at c r)
        (at d r)
    )
)
```

The solution for the problem is the following plan with cost 15:

```
(cross a b l r)
(pass-torch a b r)
(cross b r l)
(pass-torch b c l)
(cross c d l r)
(pass-torch c a r)
(cross a r l)
(cross a b l r)
; cost = 15 (general cost)
```

Problem 2 and 3

The definition of these problems is basically the same and they are quite longer, so I will show just the optimal plans with costs. The plan for the Problem 2 is the following with cost 16:

```
(pass-torch b e l)
(cross a b l r)
(cross e f l r)
(pass-torch e b r)
(cross a b r l)
(pass-torch b c l)
(cross c d l r)
(cross a b l r)
; cost = 16 (general cost)
```

Problem 3 has the following optimal plan with cost 33:

```
(cross a b i1 i2)
(pass-torch f c i1)
(cross c d i1 i2)
(cross a d i2 i4)
(cross a i4 i2)
(pass-torch c b i2)
```

```
(cross a c i2 i3)
(cross a i3 i2)
(cross a b i2 i1)
(pass-torch a e i1)
(cross e f i1 i2)
(cross e f i2 i4)
(pass-torch b a i1)
(cross a b i1 i2)
(cross a b i2 i3)
; cost = 33 (general cost)
```

In conlusion, the lengths of plans I found are the same as listed on the assignment webpage. To mention it again, there can be some unnecessary passing of torches due to the problems above, but they do not increase the cost of the plan, so the plans are still optimal.