Problem Module lv3

General info

- FIRE group number 75
- module number 3
- By
    - Oskar Wallgren, 960107-2292, IT, oskarwallgren@icloud.com
    - Hugo Cliffordson, 970917-5799, IT, cliffords.contact@gmail.com
- "We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions.
- Number of hours spent for each one of you
    - Both 14h
- Number of hours spent in supervision for this module
    - 4h

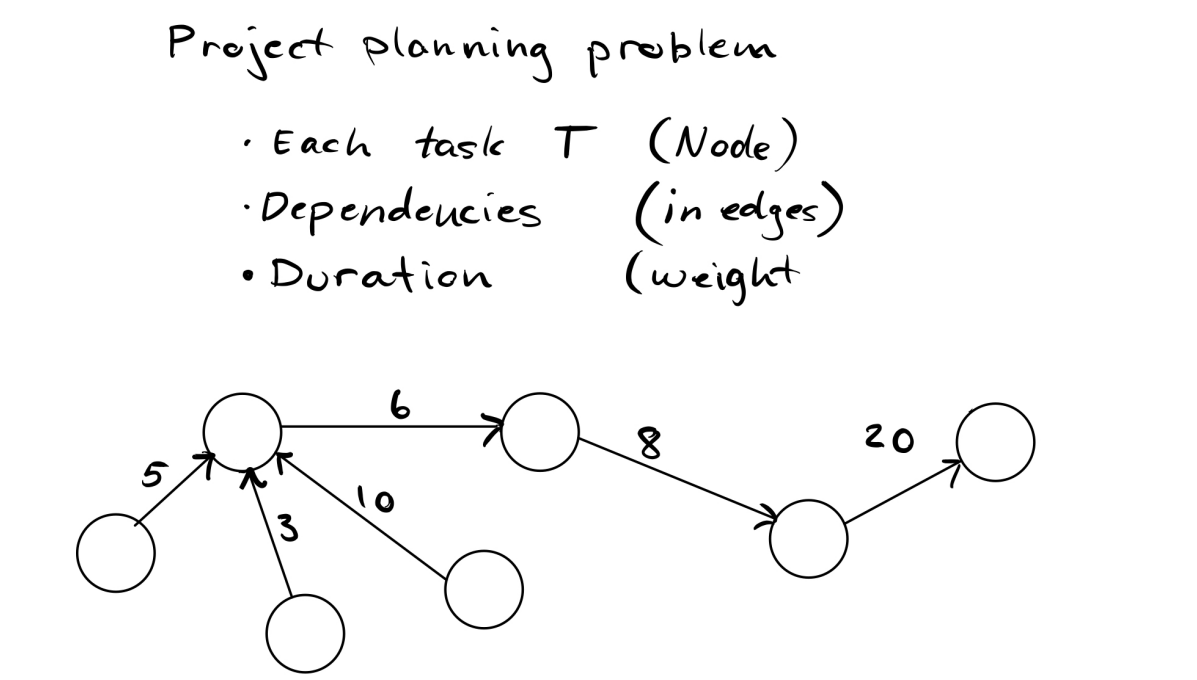
1. Using Basic Discrete Structures

- Set: unordered collection of items.
  When we have several persons signed up for a trip.
- Sequence: sequentially ordered collection of items.
  Movies sorted based on genre where one movie can appear under more than one genre
- Tree: branching structure with a root.
  When we have a file system on a computer with a root and several folders connected as children
- Graph: Structure with vertices and edges
  Social relations with individuals as vertices and their connection as edges
- Directed graph: graph where links are directed (usually represented with arrows)
  The predator-prey example where the predator has a directed edge to its prey, that prey has another edge directed to its prey and so on.
- Weighted graph: graph with a single number for each edge (can be undirected or directed)
  The tram network in Gothenburg with the stations as vertices and connecting rails as edges.

2. Project Planning Problem

We know:

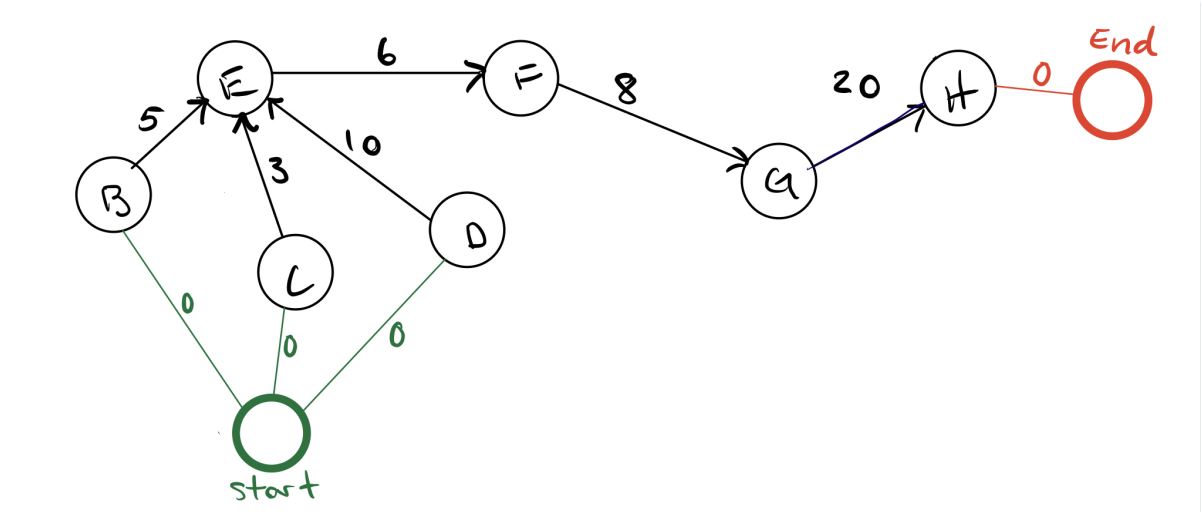
- the duration: how long the task T will take; and
- its dependencies: which tasks must be finished before T can be started.



.

a)

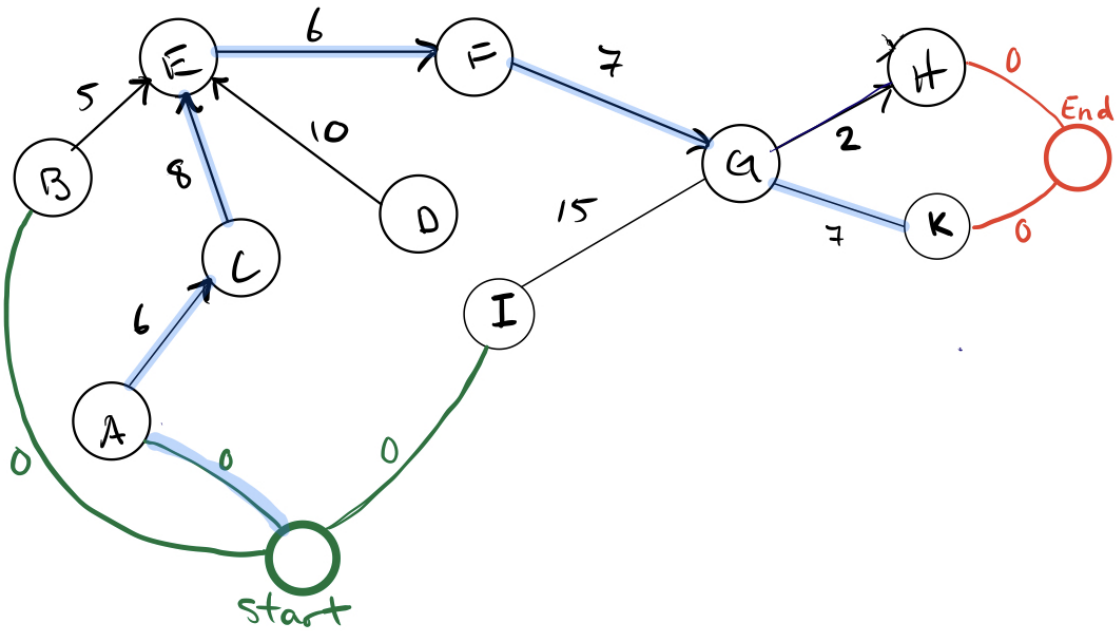
We can use a directed graph to model this problem. The first thing we did was to model a graph as the one in the figure above. In this graph the nodes are tasks, the weight of the edges is the time the task takes and the amount of in edges are constraints. The constraints describe that tasks has to be completed before moving on to the next task. Here we ran into our first problem. We have no clear representation of where we start and where we end. This problem exists because we might have a graph with many nodes that have zero *in edges* or zero *out edges*. We solved this by adding a starting node (task) that connects to all the nodes that have zero in edges with an edge that has zero weight and also an ending node (task) that connects to all the nodes with that has zero out edges with an edge of weight zero.

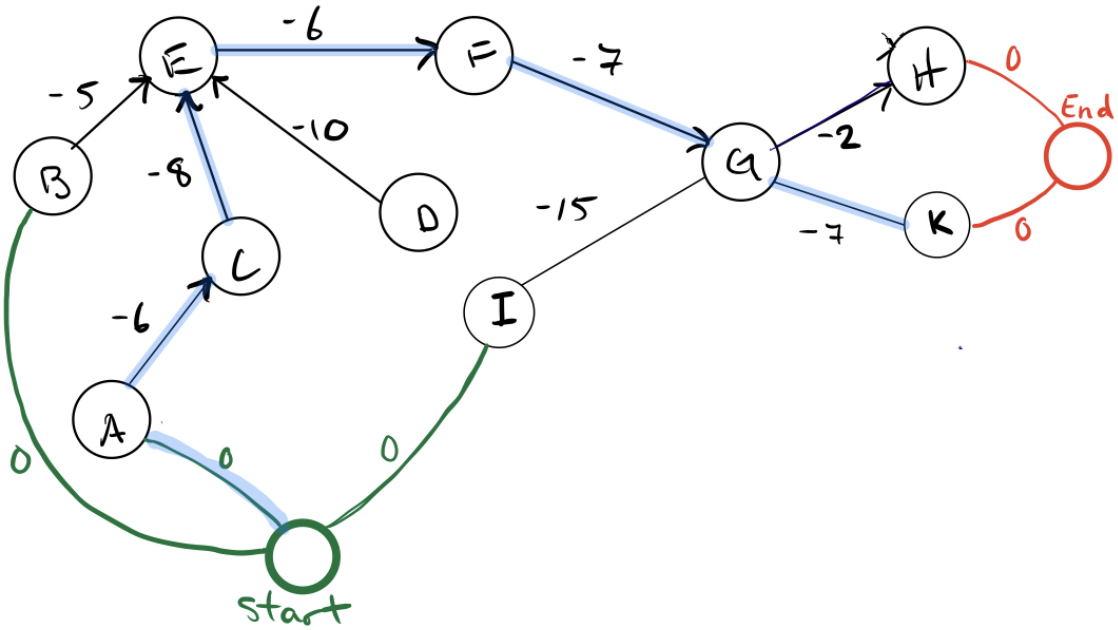


b)

Assuming that we got a program that provides a genereic solution to the shortest path problem in a graph, then we can use this to solve the planning problem. We noticed that the shortest path would not yield the right answer. Since we were supposed to use the shortest path, we know that we in some way have to manipulate the graph in order to find the solution.

We came to the conclution that if we negate all the edge values, we got the right answer in many of the graphs we tried.





Similar to $a)$ we add a start and an end node to represent starting and finishing the project. Then we negate all edges to the cost. Starting from the *start* node, finding the shortest path to end *end* node and negating the result we can see that we get the right answer.

This solution can be justified by considering that we are looking for the longest path. This is because we cannot start a task before we finish a task that it is dependent on. If the task is dependant on many tasks, the time before it can start is equal to the longest time it depends on. We can also be sure that the shortest path of values that are negated will always be the longest path. This justification is enough for us to be satisfied with the model.

c)

Since we know that we can find the shortest path with linear programming, we can assume can that it also can be used together with the logic from b).

The shortest path can be modeled as a linear programming problem because it can follow the typical expression of a linear programing optimization. We can:

- minimize an objective function.
- subject it to some constraints
- with the variables to be determined ≥ 0

If we have a directed graph with n nodes. The shortest path will be the minimum "length/weight" of the edges/arcs needed to get from the start node to the end node. Assuming one unit of flow enters at start and leaves at the end node.

Variables

$X_{(from,to)}$

This variable describes an edge with start in the node *from* and end in the node *to* as a binary value of being either 0 or 1. The 0 meaning that we do not need it in our shortest path and the 1 that we need the edge in the shortest path.

$E_{(from,to)}$

This variable describes the value (weight) of the edge that we need to add to our shortest path.

Our objective function is represented by

$$min \sum_{i=1}^{n} X_{(from,to)} E_{(from,to)}$$

It is reasonable that this is our objective function if you consider that X is one in all iterations of the sum, then we will get the sum of all the edges in the graph. If then only the edges in the shortest path has the X value of 1, then we will get the shortest path. To do this we add the following constraints.

$$X \geq 0$$

The start node is going to have zero in edges and exactly one out edge, while the end node is going to have exactly one in edge and zero out. We also need a constraint saying that the number of in edges has to be equal to to number of out edges for all other nodes in the shortest path.

When we use this together with negating all the edge values and adding a *start* and *end* node as described in *b)*. Then we will calculate the longest path by a linear programming model. This tells us that we can use a linear programming solver to find the minimum time.

d)

We interpret not knowing the exact duration means we only have an approximation of the time. An approximation of the time could be described by an interval. Let's say for example, we know a task approximately has a duration of 4 days, but it could take both three and five days to complete.

We apply this to the task solving the "worst case" scenario. Meaning we use the highest duration of time in the interval. The downside of modeling the problem like this is that we won't get the minimal time to finish the project most of the time. But we know that we won't run into any trouble starting a task before a task it's dependent on is finished.

A way to avoid this is to always use the lowest value in the interval. If we do this, we know that we will get the minimum time required every time we get it right. But the downside of this model is that we might start a task before a task it is dependent on is finished.

Another way to is to just input the approximated value in the shortest path model that we described. This will give you the shortest amount of time based on estimated values. If you then update the model continously while working on the tasks, you could run the calculation multiple times to get a better description of the actual time it will take to complete the project.

3. Conference Problem

Using shortest path

Reasonably there are several ways to solve this problem, one way we feel confident will give us the correct answer is using *shortest path*.

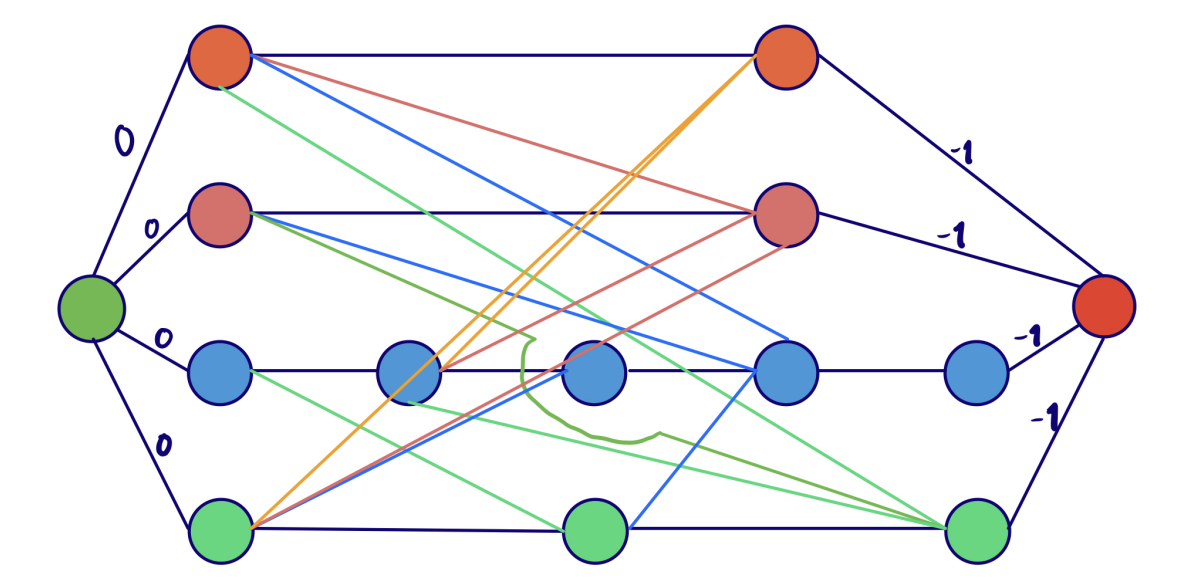
If we have all the lectures as vertices and edges with a value of 1 representing that one lecture is attended. We can use shortest path to find the "longest path" and therefore the maximum number of lectures we can attend. To do this we have to model the data with edges that represent the possible lectures to attend after finishing another lecture. We also add a *start* and *end* node that will represent starting and end of the day. These will be the nodes we want to find the shortest path between.

Let's show this with an example.

![](<Screenshot 2019-09-17 at 14.33.08.png>)

If we model this with a graph where all edges have a value of -1 and where there's an edge between all possible ways to walk between lectures we get this graph. We get this (except

for a few edges that we choose to exclude from the figure, for better readability):



The green starting node to the left represents a start and the red to the far right represent the end. If we take the shortest path between these and negates the result we are going to get the longest path and therefore the maximum number of lectures that is possible to attend. This can be justified by analizing the longest path, here this will be the path with the largest number of edges.

Using set packing

In addition to shortest path, we believe that this problem can be solved using set packing. We model the problem according to the definition of set packing linked in the problem module. The objects our case will be the lectures. For better readability we only use the part after the coffee break.

We denote the lectures from left to right $A \rightarrow D$ and with subsets $A_{1-2}, B_{1-2}, C_1, D_{1-2}$. With the value for each object as 1. Meaning "one attended lecture". Now we have to subject the object to some constraints, in order to maximize the value of objects chosen. In order to do this, we have to list all the subsets containing objects that are not possible to conbine. The subsets and lectures that are not possible to combine are going to be:

$$\{A_1, B_1, C_1, D_1\},$$
$$\{A_1, D_2\},$$
$$\{B_1, A_2, D_2\},$$
$$\{A_2, B_2, C_1, D_2\},$$
$$\{B_2, D_2\}$$

Let's say we want to attend A_1. If we look at the subsets, we can see that the only object possible to combine with A_1 is B_2 and A_2. This will return the value 2. Because we choose one of the two subsets. Meaning you have attended two lectures.

Let's list all the possible combinations:

$$\{A_1, A_2\}, \{A_1, B_2\}, \{B_1\}, \{C_1\}, \{D_1, A_2\}, \{D_1, B_2\}, \{D_1, B_2\}, \{D_1, D_2\}$$

Here we can see that the largest subsets contain two objects. As all objects have a value of 1 , the set packing algoritm will return 2.

4. Subset-sum and Partition Problem

a) Solve Partition with Subset-sum

Let's say we have a set A with n integers $a_1, ..., a_n$. Assuming we can find the sum of this set $\sum_{i=1}^{n} a_i$ we can use this to solve *partition* as true or false. We assume that with partition we have to use all elements in the set.

We know this can only be possible if $\sum_{i=1}^{n} a_i$ is even, because partition of an odd sum will return false. Subset_sum has two parameters, a set and an integer S.

$$\mathrm{SubsetSum}(\mathrm{set}, \mathrm{S})$$

If we input the set A and its sum divided by two $\frac{\sum_{i=1}^{n} a_i}{2}$ and use it as S. Subset_Sum will tell us if there is a subset of A with the sum half of the entire sum. If there is a sum like this, we know that the rest of the elements will have the same sum. This can be justified by

$$\frac{\sum_{i=1}^{n} a_i}{2} - \sum_{i=1}^{n} a_i = -\frac{\sum_{i=1}^{n} a_i}{2}$$

Let's show this with an example:

$$\mathrm{We\ have\ the\ set:}\ A = \{1, 3, 5, 7, 8, 2\}$$
$$\sum_{i=1}^{6} a_1 = 26$$

We now insert the sum divided by two and the set in the Subset*sum function. We ask the function if there is a subset of A with the sum $\frac{26}{2} = 13$. We find that for example $\{1, 5, 8\}$ yields the sum 13 and that the rest $\{3, 8, 2\}$ also yields the sum 13 Therefore _partition* will return *true*.

Reflection module 3

I. WEEKLY MEETING AND FOLLOW-UP LECTURE

a) Did you have your compulsory weekly meeting with a supervisor?

Yes, we discussed the module with Robin Adams and refined our answer on the first problem in this module. We also changed how we phrased one part of the project planning problem when we explained how we used the shortest path as a linear model.

b) Did both of you attend the compulsory follow-up lecture? If you already talked to us about this, please explain.

We attended the follow-up lecture.

c) If you were asked to talk to a supervisor about the main submission, who did you talk to?

We were not asked to talk to a supervisor.

II. WHAT DID YOU EXPERIENCE AND LEARN?

We improved our skills reasoning and discussing real life situations and model them in terms of discreet data structures.

In the project planning problem, we also learnt how to model a real life scenario in terms of discrete data structures. After modeling the problem with a directed graph, we improved our skills in using a solution to a diffrent problem to find a solution to our problem. We used the shortest path solver to find the longest path by negating the values, which gave us the answer to the question. We improved these skills by remodelling inorder to take uncertainty (when time to complete a task is uncertain) into account.

For the conference problem we used the same idea we learned in the project planning problen and again used the shortest path to find the longest path. This could be used when we represented each lecture with a node and connected possible ways to walk between them with an edge valued 1. Finding the longest path between these yielded the maximum number of lectures. We also found it reasonable to use *set packing* because it defines objects that are not possible to combine, in this case lectures overlapping.

In the subset_sum and partition problem we improved our skills breaking down the problem in smaller parts and started with small examples trying to find the solution. Trying a few different examples, we quickly found a model using the subset_sum to solve the partition problem. We were unable to solve the second part of the problem trying to model the partition solver to find a subset_sum.

III. (HOW WELL DID YOU SOLVE THE PROBLEMS?)

Other than not being able to solve the second part of problem 4 we believe we solved the problems very good. We used our modeling skills acquired so far in the course and found well developed models for each problem. We constantly tried to disprove out theories, making sure that our solutions could be justified by scenarios. We spent lots of time trying diffrent creative approaches to the problems, often arriving at more than one solution.

Assessment: very good.