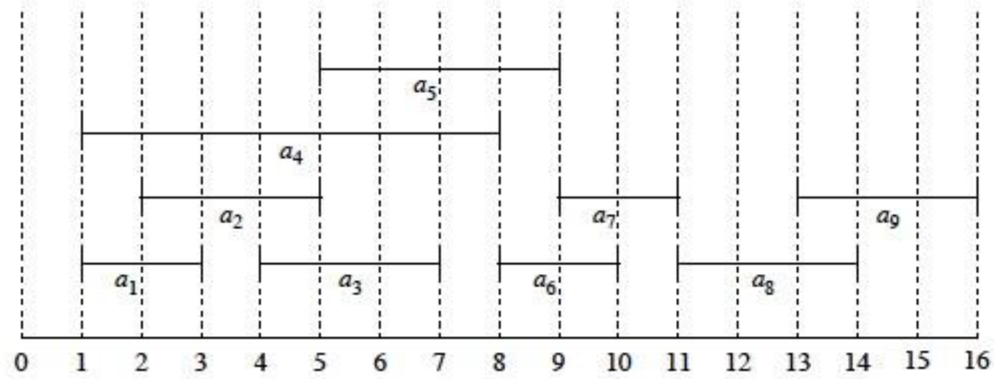
PROBLEM A: Activity Selection



The x-axis indicates time.
Activity a_i starts at a time s_i and finishes at time t_i .
For a person, the current activity must finish BEFORE another activity can start.

Find the maximum number of activities that can be done by a person ?

INPUT
The first line is the number of activities, n.
Each of the next n lines gives the start time and finish time of an activity.

OUTPUT
One number, the maximum number of activities that can be done by a person.

From Wikipedia:

> A greedy algorithm is any algorithm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum.

For many problems, *a greedy strategy does not usually produce an optimal solution,* but nonetheless a greedy heuristic may yield locally optimal solutions that *approximate a globally optimal solution in a reasonable amount of time.*

A greedy algorithm walked through state space in forward-only manner. No state is reconsidered.

- The algorithm is generally very fast, provided that the decision making at each state is efficient.
- Thus, the algorithm depends heavily on the heuristic strategy that dictates the sequence of states to be visited.

1) For activity selection problem. Sort the activities under a heuristic strategy. Then pick activity, in the sorted order, that does not overlap with the time period of the one previously picked. (NOTE: same time = overlapped)

   List out at least two strategies that you think are options that may lead to optimal solution.

2) Write the program to test each strategy.

3) As you should have at least two different programs by now, each sorts the activities based on a strategy different from the others. Conduct an experiment in order to fill in the blank below.

   The strategy that gives the always gives the best answer is to sort the activities in the (increasing/decreasing)__________________ order of _____________________

------------------- If you get the right strategy, your algorithm will complete the prove below -------------------

4) **PROOF of correctness for algorithm X** (that sorts the activities in the way answered in question 3)

Assume that the optimal set of activities is $a_1^*, a_2^*, a_3^*, \dots, a_m^*$, and so the maximum number of activities possible is m. And this list of activities is already sorted in the increasing order of finish time.

Let the output of the algorithm X be $a_1^x, a_2^x, a_3^x, \dots$. And also assume that this list of activities is sorted in the increasing order of finish time.

Then, assume further that $a_i^x = a_i^*$, for $i = 1,2, \dots, j$. And then, $a_{j+1}^x <> a_{j+1}^*$

If a_{j+1}^x finishes later than a_{j+1}^*, would algorithm X picks a_{j+1}^x for its answer ?

It would have picked ________ instead!

So, either a_{j+1}^x finishes earlier or at the same time as a_{j+1}^*.

Thus, we can replace a_{j+1}^* with a_{j+1}^x and the number of possible activities will remain to be ______.

If we repeat the same consideration on the next activity of the two lists, a_{j+2}^* and a_{j+2}^x , we will find that we can replace a_{j+2}^* with a_{j+2}^x and the number of possible activities will remain to be ______.

Is it possible that there is a $k < m$ such that a_{k+1}^x does not exist ?

Because a_k^x will finish ______________ than a_k^*, it will not conflict with a_{k+1}^*.

Therefore, at least algorithm X can at least pick _______ for a_{k+1}^x, and thus a_{k+1}^x definitely exists.

By repeating the same consideration for the next value of k, it is clear that the number of activities produced by algorithm X must be at least ________ .

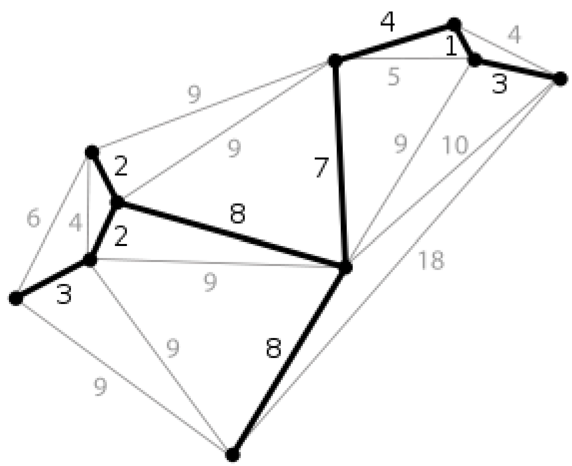
But from assumption, the maximum number of activities possible is k, therefore, the number of activities produced by algorithm X must be at most ________ .

Therefore, we can conclude that the number of activities produced by algorithm X is exactly _____ .

And thus, the algorithm X gives the maximum number of activities possible … PROVED!

A GREEDY ALGORITHM IS MOSTLY VERY EFFICIENT, BUT **IT NEEDS TO BE PROVED FOR CORRECTNESS.**

PROBLEM B: Minimum Spanning Tree



- A spanning tree T of an undirected graph G is a subgraph that is a tree which includes all of the vertices of G.
- A minimum spanning tree is a spanning tree whose sum of edge weights is as small as possible.

Example application

The standard application is to a problem like phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost.

****** MST WAS LEARNED IN DATA STRUCTURE AND ALGORITHM ******

Two famous solutions : Kruskal's (covered in this class) and Prim's

1) Review Kruskal's algorithm (from any offline or online source). For a small undirected graph, you should be able to compute by hand the total weight of an MST.

INPUT

The first line contains the number of vertices (V) and the number of edges (E), respectively

Each of the following line represents an edge. It lists out three numbers; the first two are the vertices that are connected by the edge, and the last number is the weight of the edge. (vertex ID begins at 0)

OUTPUT

One number, the total weight of a minimum spanning tree.

2) Write the program that read the input and stores all edges in a list, each as a triple (v1, v2, weight).

3) Sort the edge list in the increasing order of edge weight.

4) Study the provided “Data Structure for Disjoint Sets”. Write program that utilizes the data structure.

5) At the beginning of Kruskal’s algorithm, each vertex is associated with a dedicated set.

   - Therefore, there are V sets at the start.
   - Sequentially consider the edge according to the sorted edge list.
     - Take only the edge that connects two different sets, and then the edge causes the two sets to unite into one.
   - The operation repeats until the number of sets is reduced to 1.

6) A greedy algorithm is typically heuristic. It is accepted by being proved that it works correctly.

   PROOF:

   Let the minimum spanning tree of a graph G be T*.

   Suppose that Kruskal’s algorithm produces T.

   Assume first, that the two trees are different, and the lightest edge in T* which is not in T is e.

   Adding e to T will create a ________________ (let’s call it C)

   There must be an edge in C that is not in T*. Why ? (let’s call this edge f)

   Removing f from T will create a new spanning tree. Why ? (let’s call this new spanning tree T_1)

   T_1 will be more similar to T* than T. But is the total weight of T_1 less than T ?

   Suppose that it is, this means that e ____ f. (a comparing operation)

   If e was really so, would it be picked by Kruskal’s algorithm before f ?

   Therefore, the fact must be that e ____ f.

   Therefore, we can conclude that weight of T_1 _____ T.

   Continually transforming T_1 further in the same fashion until the resulted spanning tree becomes T*. Will the transformed spanning tree has less total weight than T ? Why ?

   But the total weight of T* is the minimum possible (by assumption). Therefore, the only possible conclusion is the total weight of T ____ the total weight of T*. **PROVED!**