Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer

3. December 2018

Datenstrukturen & Algorithmen Blatt P13 HS 18

Please remember the rules of honest conduct:

- Programming exercises are to be solved alone
- Do not copy code from any source
- Do not show your code to others

You can submit your codes to the Judge until the exam. No bonus points are given for these exercises. Questions concerning the assignment will be discussed as usual in the forum.

Exercise P13.1 *Frequency Assignment.*

The city of Algo wants to auction the usage rights of its radio spectrum to $m \geq 1$ telecommunication companies, numbered from 1 to m. The radio spectrum is divided into contiguous, non-overlapping, and indivisible *channels* that are 10MHz wide. In particular, the j-th channel allows to use the frequencies between $j \cdot 10$ MHz and $(j+1) \cdot 10$ MHz.

The i-th telecommunication company will pay $p_{i,j} \geq 0$ *Flops* (the currency of the city of Algo) for the right to use the j-th channel. Moreover, the i-th company will also pay an extra amount of $h_{i,j} \geq 0$ Flops for the right to use the contiguous 30 MHz-wide block starting at $j \cdot 10$ MHz and ending at $(j+3) \cdot 10$ MHz (i.e., the company will pay $p_{i,j} + p_{i,j+1} + p_{i,j+2} + h_{i,j}$ if it is assigned the channels j, $j+1$, and $j+2$).

The assignable range of frequencies is between 10 MHz and $(N+1) \cdot 10$ MHz (i.e., there are exactly $N \in \mathbb{N}^+$ channels, numbered from from 1 to N), and each channel can be assigned to at most one company (some channels may be left unassigned). Moreover, as prescribed by the laws of Algo, no single company can own a contiguous block of frequencies that is wider than 30 MHz.

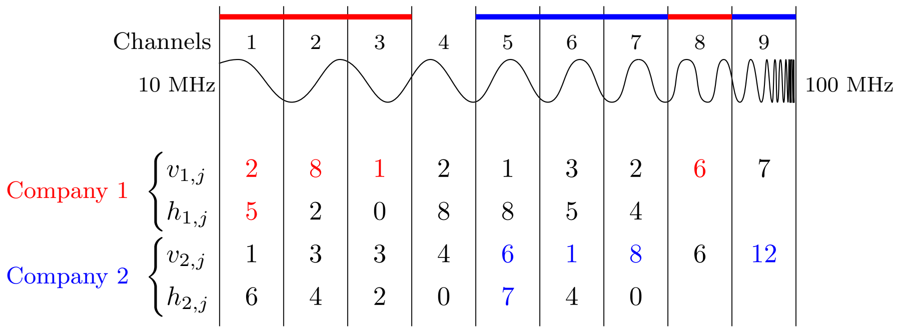
You are given the numbers N, m, $p_{i,j}$, and $h_{i,j}$ ($\forall i = 1, \dots, m$, $\forall j = 1, \dots N$) and your task is to find an assignment of the channels to the companies that abides the law and maximizes the amount of Flops earned by the city.

Input The input consists of a set of instances, or *test-cases*, of the previous problem. The first line of the input contains the number T of test-cases. The first line of each test-case contains the integers N and m. The next m pairs of two lines contain the values v and h for the m companies. In particular, the $(2 \cdot i)$-th line of the test case contains the values $v_{i,j}$ $\forall j = 1, \dots, N$ while the $(2 \cdot i + 1)$-th line contains the values $h_{i,j}$ $\forall j = 1, \dots, N-2$.

Output The output consists of T lines, each containing a single integer. The i-th line is the answer to the i-th test-case, i.e., it contains the maximum amount of Flops that can be earned by the city of Algo.

Grading This task awards 0 points. Your algorithm should require $O(N \cdot m)$ time (with reasonable hidden constants). Solutions with a time complexity of $O(N \cdot m^2)$ (and reasonable hidden constants) will not pass all test cases. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD8H13P1`. The enrollment password is "`asymptotic`".

Example



		1	2	3	4	5	6	7	8	9
Company 1	$v_{1,j}$	2	8	1	2	1	3	2	6	7
	$h_{1,j}$	5	2	0	8	8	5	4		
Company 2	$v_{2,j}$	1	3	3	4	6	1	8	6	12
	$h_{2,j}$	6	4	2	0	7	4	0		

Figure 1: An instance with $N = 9$ and $m = 2$. An optimal assignment of channels to companies is highlighted in red and blue. This assignment collects 56 Flops from the companies.

Input (the first test-case corresponds to Figure 1):

```
2
9 2
2 8 1 2 1 3 2 6 7
5 2 0 8 8 5 4
1 3 3 4 6 1 8 6 12
6 4 2 0 7 4 0
8 1
3 2 3 6 8 4 1 6
9 1 1 5 3 4
```

Output:

```
56
35
```

Notes

For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD8H13P1.Frequency_Assignment.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class.

Exercise P13.2 *Jungle.*

Surrounded by dense jungle and wild ravines, you nervously look at your map. A storm is approaching and you need to leave the area as soon as possible …

On your map, the jungle is divided into a square grid $n \times n$. For each square with coordinates $0 \leq i, j < n$, your map shows the time $t_{i,j}$ it takes you to move into this square from one of its neighbors. (For simplicity, only *entering* the square costs time and the direction does not change that.) Your starting square is identified by having $t_{i_0,j_0} = 0$ and there is exactly one such square; the other squares satisfy $1 \leq t_i \leq 10\,000$. The size of the map is $1 \leq n \leq 500$.

From any square you can move in four directions to an adjacent square. When you arrive at a square adjacent to the edge of the map (at any of the four edges), you may immediately leave the jungle and your journey ends. The time that the entire journey takes is the sum of the required times of the squares you visited.

Your task is to find the *minimal time* needed to leave the jungle from the starting square. The number of squares visited is irrelevant, just the time matters. Note that there may be several fastest paths of equal time but we care only about their time.

Example

Below are two examples of jungle maps with $n = 8$ and $n = 12$ and a shortest path in marked gray. The time needed to leave the first jungle is 10, and in the second case 236.

9	1	1	9	1	1	9	1
9	1	9	9	9	9	1	9
9	1	1	9	1	1	1	9
9	1	1	9	1	1	9	9
9	1	9	1	1	0	9	9
9	1	1	1	9	1	9	9
9	1	9	1	9	9	9	1
9	9	1	9	1	9	1	42

65	81	23	82	98	41	68	48	85	96	17	22
71	10	34	65	29	39	28	21	50	66	14	86
41	71	44	79	13	78	66	56	12	13	36	29
85	50	99	87	10	94	79	31	14	20	65	91
35	98	41	27	55	23	26	18	63	41	77	50
53	21	83	47	71	10	91	38	60	41	19	68
74	93	58	70	43	24	0	92	96	38	33	93
86	26	51	63	64	44	13	91	57	39	27	63
73	39	48	27	85	50	72	72	70	67	52	35
34	14	81	95	61	80	68	65	17	96	11	31
49	72	22	96	40	99	94	38	15	37	74	50
97	58	80	35	77	72	71	41	78	27	19	38

Input The input consists of several cases. The first line of the file contains the number of cases to follow.

Each case starts with n on a separate line. The next n lines contain n numbers each. i-th line contains the integers $t_{i,j}$ for $j = 0, \ldots, n-1$, separated by spaces.

Output For every test case, write the time required to leave the jungle on a separate line.

Example

Input (for the examples above)

```
2
8
9 1 1 9 1 1 9 1
9 1 9 9 9 9 1 9
```

```
9 1 1 9 1 1 1 9
9 1 1 9 1 1 9 9
9 1 9 1 1 0 9 9
9 1 1 1 9 1 9 9
9 1 9 1 9 9 9 1
9 9 1 9 1 9 1 42
12
65 81 23 82 98 41 68 48 85 96 17 22
71 10 34 65 29 39 28 21 50 66 14 86
41 71 44 79 13 78 66 56 12 13 36 29
85 50 99 87 10 94 79 31 14 20 65 91
35 98 41 27 55 23 26 18 63 41 77 50
53 21 83 47 71 10 91 38 60 41 19 68
74 93 58 70 43 24 0 92 96 38 33 93
86 26 51 63 64 44 13 91 57 39 27 63
73 39 48 27 85 50 72 72 70 67 52 35
34 14 81 95 61 80 68 65 17 96 11 31
49 72 22 96 40 99 94 38 15 37 74 50
97 58 80 35 77 72 71 41 78 27 19 38
```

Output

```
10
236
```

Grading This task awards 0 points. The program should run in time $\mathcal{O}(n^2 \log n)$ to pass all test cases. If you want to use a heap, use `java.util.TreeSet<>` or `java.util.PriorityQueue<>`[1] or implement your own. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD8H13P2`, enroll password is "`asymptotic`".

Notes

For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD8H13P2.Jungle.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}`, `java.util.{TreeSet<>, PriorityQueue<>}` and `java.util.Scanner` class.

[1]For these library structures, you will need to create a `class` for the heap elements and either make the class `Comparable` or make a `Comparator`. Also note that `PriorityQueue` has slow removal of elements other than the queue head, and you should not modify the order of the elements in the structures without removing and reinserting them. If unsure about the programming or timing details, implement your own heap specific to your needs.

Exercise P13.3 *Flea Market.*

You have the bad habit of leaving items on the floor of your basement, which is now completely full. You decide to clean it up by selling some of the items in the flea market of the city of Algo. There are n items in your basement and the i-th item occupies a surface of s_i m^2, weighs $w_i \geq 1$ Grahams (the weight unit of the city of Algo), and can be sold for a price of p_i Flops (the currency of the city of Algo). You want to free an area of least S m^2 from your basement but you can only carry at most W Grahams to the flea market.

Your task is to design an algorithm that computes the maximum amount of Flops that you can earn from the sale (subject to the above conditions).

Input The input consists of a set of instances, or *test-cases*, of the previous problem. The first line of the input contains the number T of test-cases. The first line of each test-case contains the three positive integers n, S and W. The next n lines each describe one item: the i-th line contains the three integers s_i, w_i and p_i.

Output The output consists of T lines, each containing a single integer. The i-th line is the answer to the i-th test-case, i.e., it contains the total value V of the items to sell at the flea market. More precisely, $V = \max_{X \in \mathcal{I}} \sum_{i \in X} p_i$ where $\mathcal{I}$ contains all the sets of items that have a total area of at least S and a total weight of at most W, i.e., $\mathcal{I} = \{X \subseteq \{1, \dots, n\} : \sum_{i \in X} s_i \geq S \text{ and } \sum_{i \in X} w_i \leq W\}$.

Grading This task awards no bonus points. Your algorithm should require $O(n \cdot S \cdot W)$ time (with reasonable hidden constants). Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD8H13P3`. The enrollment password is “`asymptotic`”.

Example

Input:

```
1
6 10 12
1 4 10
3 5 8
7 10 5
5 2 3
1 1 1
3 4 2
```

Output:

```
22
```

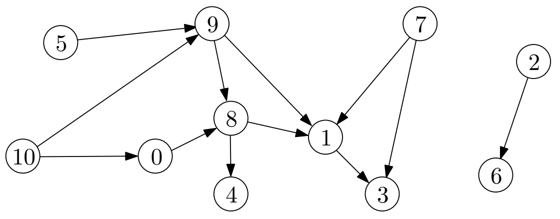
Notes For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD8H13P3.Flea_Market.zip` The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones `java.io.{InputStream, OutputStream}` and `java.util.Scanner` class.

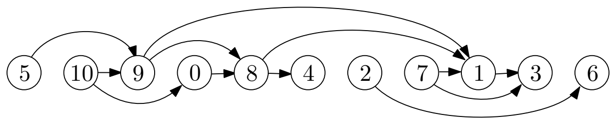
Exercise P13.4 *Longest path in a directed acyclic graph.*

Given a directed acyclic graph (also called a *DAG*), your task is to compute the longest directed path in the graph. The n vertices of the graph are numbers $0, 1, \ldots, n-1$ and there are m directed edges (also called *arcs*), each from vertex s_i to vertex t_i for $i = 1, \ldots, m$. A *directed path* of length l is a sequence of l distinct[2] vertices $p_1, \ldots, p_l$ such that there is a directed edge from p_i to p_{i+1} for every $i = 1, \ldots, l-1$. The graph being *acyclic* means that there is no directed cycle in the graph, or formally there is no directed path from p_1 to p_l such that there is also an edge from p_l to p_1 (closing the directed cycle).

For finding the longest path in a graph, no efficient algorithms are known that would work for *every* graph[3]. But in the case of DAGs, simple and efficient solutions exist and we suggest the following:

First, find a *topological ordering* on the vertices (an ordering $v_1, \ldots, v_n$ of the vertices such that edges from v_i only go to v_j with $j > i$) as seen in the lecture. Then use a dynamic programming or a similar approach on that ordering to find the longest directed path (we leave the details to you).





Example Consider a graph with $n = 11$, $m = 12$ and the following edges (from-to): 7→3, 1→3, 7→1, 2→6, 5→9, 10→9, 10→0, 0→8, 9→8, 9→1, 8→4, 8→1. The graph and one of its topological orderings are drawn above. The longest directed path has length 5 and there are 3 longest paths: 5-9-8-1-3, 10-9-8-1-3 and 10-0-8-1-3 (there may be many more, and we only care about the length).

Input The input consists of several cases. The first line of the file contains the number of cases to follow.

Each case consists of two lines: The first line contains the integers $1 \leq n \leq 10\,000$ and $0 \leq m \leq 10\,000$, separated by a space. The second line contains $2m$ integers $s_1, t_1, s_2, t_2, \ldots, s_m, t_m$, the start and end points of the directed edges, all $s_i, t_i \in \{1, \ldots, m\}$, separated by spaces.

The graphs contain no loops (e.g. edges from v to the same vertex v) or multiple parallel edges from u to v. Also note that if there is an edge from u to v, there may be no edge from v to u, as that would break acyclicity. The input graph may or may not be connected, as you can see in an example below. Also note that there may be *undirected* cycles (cycles when you ignore the direction of the edges), also illustrated in the example. The edges may be listed in any order.

Output For every test case, write the length of a longest directed path on a separate line.

[2]In directed acyclic graph, every directed path is a simple path, as repeating a vertex would imply a directed cycle.
[3]Look up "Hamiltonian path problem" and "NP-completeness" on Wikipedia if you want to know more.

Example

Input (for the example above and an empty graph)

```
2
11 12
7 3 1 3 7 1 2 6 5 9 10 9 10 0 0 8 9 8 9 1 8 4 8 1
5 0
```

Output

```
5
1
```

Grading This task awards no bonus points. The program should run in time $\mathcal{O}(m+n)$. Submit your `Main.java` at `https://judge.inf.ethz.ch/team/websubmit.php?cid=25012&problem=AD8H13P4`, enroll password is "`asymptotic`".

Notes For this exercise we provide an archive containing a program template available at `https://www.cadmo.ethz.ch/education/lectures/HS18/DA/uebungen/AD8H13P4.Longest_Path.zip`. After loading, the graph is represented in three ways for your use: A list of start- and end-vertices s_i and t_i for the m edges. For every vertex v a list of out-neighbors (vertices with direct edge *from* v). For every vertex v a list of in-neighbors (vertices with a direct edge *to* v). Note that adjacency matrix is not well-suited for this task.

The archive also contains additional test cases (which differ from the ones used for grading). Importing any additional Java class is **not allowed** (with the exception of the already imported ones and `java.util.{Scanner, List, ListIterator, ArrayList}`, as well as `java.io.{InputStream, OutputStream}` class.

Extra If you can solve the problem above and find the length l of a longest path, how would you count all the directed longest paths in time $\mathcal{O}(m+n)$? (You may assume the final number fits into an `int`.) This is an extra question for no points, but see if you can solve it.