The Potential of Answer Classes in Large-scale Written Computer-Science Exams

Dominic Lohr[1], Marc Berges[2], Michael Kohlhase[3], Florian Rabe[4]

Abstract: Students' answers to tasks provide a valuable source of information in teaching as they result from applying cognitive processes to a learning content addressed in the task. Due to steadily increasing course sizes, analyzing student answers is frequently the only means of obtaining evidence about student performance. However, in many cases, resources are limited, and when evaluating exams, the focus is solely on identifying correct or incorrect answers. This overlooks the value of analyzing incorrect answers, which can help improve teaching strategies or identify misconceptions to be addressed in the next cohort.

In teacher training for secondary education, *assessment guidelines* are mandatory for every exam, including anticipated errors and misconceptions. We applied this concept to a university exam with 462 students and 41 tasks. For each task, the instructors developed *answer classes* – classes of expected responses, to which student answers were mapped during the exam correction process. The experiment resulted in a shift in mindset among the tutors and instructors responsible for the course: after initially having great reservations about whether the significant additional effort would yield an appropriate benefit, the procedure was subsequently found to be extremely valuable.

Keywords: computer science education; task analysis; answer classes

1 Introduction

A reliable and fair assessment of exam assignments is a basic requirement for professional teaching. Especially in large-scale written exams, resources are limited, and high-quality assessment of submissions is challenging for educators. Inconsistencies in correction are possible when multiple people correct the same assignment. The feedback provided to students is very basic at best and often only provided in the form of scoring.

To support the correction process to make it fairer and more objective, in secondary education and higher education, marking schemes respectively rubrics [AP11] are mandatory when creating assignments – especially exams. The advantage is reduced required corrections and more objective task evaluation options. In addition to the model solution, rubrics contain alternative solutions and a list of possible incorrect answers. This led us to the concept of

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didaktik der Informatik dominic.lohr@fau.de
[2] Friedrich-Alexander-Universität Erlangen-Nürnberg, Didaktik der Informatik marc.berges@fau.de
[3] Friedrich-Alexander-Universität Erlangen-Nürnberg, Wissensrepräsentation & -verarbeitung michael.kohlhase@fau.de
[4] Friedrich-Alexander-Universität Erlangen-Nürnberg, Wissensrepräsentation & -verarbeitung florian.rabe@fau.de

answer classes – groups of answers that share the same underlying idea. The intelligent clustering of possible solutions into answer classes holds several great potentials for modern teaching – especially when dealing with so many learners that individual face-to-face support is impossible.

To investigate the potential of answer classes in large-scale written computer science exams, we developed a process model that we evaluated in the first iteration of an exam for an Artificial Intelligence major program that 462 students took. Following the process model, 192 answer classes were developed for 41 tasks. In the sequel, we present a detailed analysis of five tasks aimed at identifying the specific (possible) causes of incorrect answers and feeding those back into next year's teaching process. The results demonstrate the efficacy of answer classes in creating rubrics for large courses and highlight their significant potential for enhancing instructional materials, improving the fairness of correction, and developing adaptive feedback.

A completely unexpected consequence was that the experiment resulted in a paradigm shift among the tutors and instructors responsible for the course, who – in a university setting – were not accustomed to rubric practices in secondary education. Despite initial reservations regarding the additional effort required, the procedure was ultimately deemed valuable and worthy of undertaking in every future exam.

2 Related Work

Classifying answers in examinations or tasks, in general, is an essential but tedious part of correcting and rating written exams. It is no surprise that the advent of computers has led to efforts to automate that.

Auto-grading systems and feedback generation often refer to a set of rules defined by the author of the task. For instance, this is often done in programming using static and dynamic code analysis, as seen in the system JACK, which uses a query language in XML to specify test cases and code structures relevant for grading [St16]. JACK handles diverse tasks like UML modeling and fill-in-the-gap assignments. For complex programming tasks, many e-assessment systems detect errors by executing test cases and analyzing source code [Ih10, SS22]. Automated grading matches human performance [GDG14], with fine-grained feedback being effective [Fa14]. In contrast, Course Master by Higgins et al. integrates marking schemes, separating exercises and grading logic as Java classes, enabling parametrized tasks, marks (grades), and feedback [HST02]. Lawrence et al. explore performance-based grading, crucial for e-learning and remote exams [LFU23].

Answer Clustering/Classification A different approach to classifying student answers is conducted by Zehetmeier et al. [Ze15]. They analyzed student submissions to identify and cluster errors, and quantify significant groups to uncover error causes. The researchers also observed the students while programming and interviewed them about their errors at

crucial stages and used the results to create teaching materials and tasks to prevent similar misconceptions. Nabil et al.'s EvalSeer uses machine learning for classifying answers and proposes syntax fixes with an appropriate accuracy [Na21]. While the focus of many studies is on programming tasks due to their complexity, there are other complex tasks in computer science, such as marking up UML database diagrams [FUL22].

3 Theoretical Background

Answer classes are one of the four components of the Y-Model [Lo23] – a model for formalizing tasks in Computer Science (see Figure 1). The model forms a basis for a competence-oriented integration of tasks in adaptive educational systems.

The concept of **answer classes** enables a selection process of tasks based on prior knowledge and competency goals, as well as depending on answers given by students from previous tasks. In addition, specific error patterns, such as those proposed by Zehetmeier et al. [Ze15], or Berges et al. [Be16], can be integrated into answer classes and, for instance, infer possible causes of errors by referring to a corresponding learner model. In addition, answer classes can act as keys for specifying feedback or hints on how to proceed. We review the essentials for the discussion.

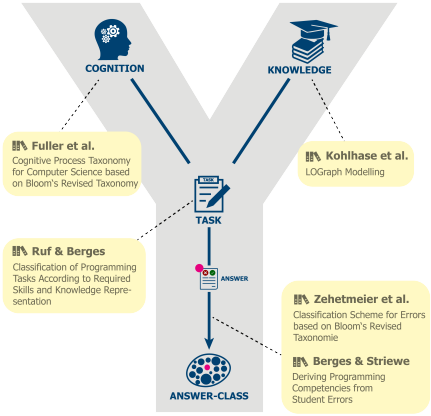


Fig. 1: The Y-model

Answer classes (ACs) can be understood as a set-theoretic propositional form. Each answer class contains an identifier AC_x (unique to the task) and a (precise) description (*ACD*) of the criteria when a given answer is assigned to this AC. Each answer R to a task can be assigned to one (or more) *AC*s. For the *AC* to be reliably annotated by any educator, the *ACD* must be unambiguous (free of interpretation) and objectively observable. *AC*s are notably not free of intersections, and answers can be a member of several different *AC*s.

A general distinction is made between context-independent (applicable to any task) answer classes and context-dependent (related to a specific type of task or subject area). Examples of the former are provided by $\mathrm{AC}_1 = \{R \mid R \text{ is empty}\}$, which contains all responses that have not been processed, and $\mathrm{AC}_2 = \{R \mid R \text{ is crossed out}\}$, which contains answers that have been written but crossed out. In terms of scoring, these two answer classes have the same effect, but in the course of a didactic evaluation, it makes a difference whether a task was not worked on or was attempted. The criteria of task-specific *AC*s depend on the knowledge dimension or task.

4 The Potential of answer classes

Developing and implementing answer classes requires extra work for educators, but there are also possible benefits. To evaluate the potential of ACs, we first define criteria based on the benefits of marking schemes or rubrics from different sources (e.g., [WS07], [RA10]):

ST *Save time* Conventional correction process, involving marginal notes and closing comments, can be time-consuming. *AC*s alleviate the need for these annotations, potentially making the correction process more efficient.

BF *Better feedback* In most cases, the lack of time during corrections leads to the omission of valuable feedback. However, developing feedback for each answer class must only be done once (or for their combinations) and can be effortlessly provided as a table.

MOC *More objective correction* Multiple educators correct the same task in large courses, leading to disparate results due to varying personal interpretations. Answer classes offer a set of precisely defined, objectively observable criteria, enhancing the reliability of the correction process.

BPE *Better pre-estimation* Addressing potential student errors in advance helps to ensure that exam tasks are appropriate and error-free.

IIT *Incentives to improve teaching* The annotation of answers with ACs allows for a higher quality evaluation of exam results, providing valuable insights for teaching improvement. Prevalent error classes can indicate particular areas in the teaching material that must be revised to avoid misconceptions.

Of course, all of these benefits invite and profit from automation. As a prerequisite, we have devised a process model to systematically develop and apply the concept of ACs to evaluate their effectiveness. The model and the results from an initial pilot are presented in this paper.

5 Methodology

To investigate the potential categories stated in section 4, we devised an iterative process model to develop and apply the answer classes concept systematically (Figure 2).

Upon creation of the exam, the instructors were introduced to the concept of answer classes as detailed in Section 3. To begin with, a preliminary set of task-specific answer classes was created based on prior experience with similar tasks (Step 1).

Subsequently, we discussed the initial answer classes for each task in a plenary session (Step 2). For this purpose, the instructors presented the tasks and the first draft of the set of answer classes to the group. During this session, all stakeholders checked that the answer classes were objectively observable and free of subjectivity or ambiguity.

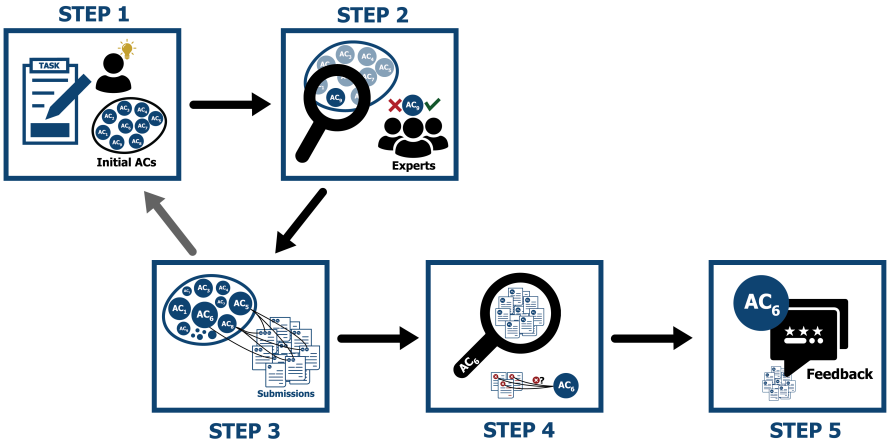


Fig. 2: Iterative Process of AC-Creation

Considering that the exam was administered as a paper-based test, we developed an **AC mapping form** to facilitate the mapping process of students' submissions to corresponding answer classes (Step 3). The list contained – besides the students' IDs – a table with all the identifiers of the ACs for each task.

Based on a descriptive analysis of the results in Step 3, we selected ACs with frequent occurrences and further scrutinized them in Step 4. Here we analyzed sample exams of the respective AC and identified possible underlying factors that led the student to that answer.

The outcomes of Step 4 informed the development of feedback for each answer class. The development of AC identifiers for automated annotation of submissions with their respective AC can be incorporated in this step but was left as future work.

If exam task types are reused in later exams, this process is iterative: the revised set of answer classes after Step 3 serves as the initial set in Step 1 of the subsequent iteration.

6 Results

The procedures outlined in section 5 were initially tested in an exam for the course *Artificial Intelligence 1*. Of the 502 registered students, 462 participated in the written exam (90 minutes, conducted in presence). The majority of the students were pursuing their master's degree in Data Science (225), Artificial Intelligence (118), and Computer Science (69). The exam consisted of 12 problems, encompassing 41 distinct tasks. Steps 1 to 3 were done for all 41 tasks. A total of four common and 192 initial answer classes were developed. Since steps 4 and 5 were more time-intensive, we selected one problem (containing five tasks) and

executed these steps. The selected problem focused on searching in a directed graph with edge costs and node heuristics. The concrete objective was to apply five different search algorithms (e.g., A^*-Search, Greedy-Search, ...) starting from an already expanded node and to give the subsequent four visited nodes (for a full problem description, see Figure 4 in Appendix). This section describes the application of steps 1-5 to this problem.

Table 1 shows the initial set of ACs. AC_1 to AC_4 are the task-unspecific classes that are the same for all tasks. Since the tasks differ only in the search algorithm to be applied, the initially conceived task-specific answer classes AC_5 to AC_8 were also the same.

ID	Answer Class
AC_1	$\{R \mid R$ is empty$\}$
AC_2	$\{R \mid R$ is crossed out$\}$
AC_3	$\{R \mid R$ is fully correct$\}$
AC_4	$\{R \mid$ no suitable AC known$\}$
AC_5	$\{R \mid$ the first node of R is correct$\}$
AC_6	$\{R \mid$ the second node of R is correct $\}$
AC_7	$\{R \mid$ the third node of R is correct$\}$
AC_8	$\{R \mid$ the fourth node of R is correct$\}$

Tab. 1: List of Initial ACs for Problem 2.1 (Step 2)

It turned out that these initial ACs were not very helpful for further analysis of the answers because the information was limited. They were useful only in correction since it was possible to determine the exact score from the combination of ACs assigned to an answer. However, the feedback that could be generated from this was only very simple. For this reason, in addition to the AC-mapping process (Step 3), various answers were recorded in a table via frequency analysis to find an improved set of new answer classes for these tasks.

ST 1 (depth-first)			ST 2 (breadth-first)			ST 3 (uniform-cost)			ST 4 (greedy)			ST 5 (A*-search)		
AK	Answer	#	AK	Answer	#	AK	Answer	#	AK	Answer	#	AK	Answer	#
A3	BDHI	406	A3	BCDE	418	A3	CFBE	301	A3	BEDH	229	A3	CFGE	135
	BDHE	18		BCED	18		CFBG	21		BECG	89		CFGI	52
	BDEC	7		BECD	9		CFGB	14		CFEC	18		CFEC	34
	BDEH	6		BCDF	3		CBFE	12		CFHI	16		CFIG	29
	BDHF	3		ABCD	2		CFHI	12		BDHI	12		CFEI	20
	ABDH	2		10,8,5,9	1		CFEG	10		BECF	10		CFGB	19
	BDEH	2		BCD	1		CBEF	9		BEFI	9		CFGF	15
	BEDH	2		BCEF	1		CEFB	5		BEDC	8		CFEG	13
	CFEH	2		BCGF	1		CFEC	5		BCEG	6		CFHI	13
	10,8,7,4	1		BECF	1		CBEG	4	A10	CFEH	5		CBFE	12
	...	1		...	1		...	4		...	4		...	10

Fig. 3: Frequency analysis of given answers to the tasks

The basic assumption of this study was that (incorrect) answers given identically by a large number of students could represent an identical or similar error pattern and thus provide justification for further investigation. Figure 3 shows the results of the frequency-analysis. In addition to the group of correct answers (first line, AC_3), we found many other relevant clusters of answers that represent potential ACs. Limited resources were available for further evaluation, so we studied only the largest five groups per task in a group session with experienced educators to identify possible causes of the errors.

The groups for which objectively observable criteria could be found were included as new answer classes (see Table 2 for an excerpt of these). For example, although the task description clearly states that the start node should not be included in the list of visited nodes, some of the submissions showed it as the first node visited (AC_{10}). Also, some of the nodes already visited were re-added to the list of visited nodes (AC_{11}).

While these examples are probably based on a careless reading of the task description, we also found ACs that indicate deeper technical errors. Most ACs that emerged in the second iteration were much more focused on the causes of the answers given and made it possible to develop valuable feedback. In task 4 (greedy search), there was a significantly large number of submissions (89 out of 462) that had the same error pattern: the detailed analysis revealed that this error pattern occurs when the greedy search algorithm is misapplied, namely by always selecting a neighbor of the last selected node instead of the overall cheapest node (AC_{30}). For the largest group of errors in A^*-search (52 of 462), the experts conjectured the cause to be the swapping of the edge direction between nodes *G* and *I*. This could be due to the rather small arrowheads in the task description, which may, in particular, pose a problem for students with impaired vision.

However, which misconceptions or knowledge gaps are underlying the respective AC cannot be determined with certainty without information from/about the learner. Further research would be necessary, for instance, with the help of interviews with the students.

Task	ID	Answer Class
ST1-5	AC_{10}	start node included in the sequence of visited nodes
ST1-5	AC_{11}	visited node included multiple times (circles/loops)
ST1-5	AC_{12}	use of reverse alphabetical order to break ties
ST1, ST2	AC_{21}	algorithm applied to heuristic instead of costs
ST3, ST4	AC_{22}	algorithm applied to costs instead of heuristic
ST2	AC_{31}	selection of nodes that lie in a horizontal line
ST2	AC_{32}	selection of nodes that lie in a vertical line
ST4	AC_{30}	greedy search considering only neighbors of previous node
ST5	AC_{30}	A* search with reversed edge direction at nodes *G* and *I*

Tab. 2: Some answer classes newly found during the analysis

7 Discussion

Applying the steps described in section 5 required additional time. Whether the use of ACs saves time (see **ST** in section 4) cannot be answered based on the results in this paper. In the long term, however, we assume that time savings can be substantial, especially in shared tasks and re-used settings. Even in task variants – e.g., by "changing numbers," ACs can often be transferred (at least in spirit). In particular, if predominant error patterns are documented, and suitable feedback has already been developed, the correction effort (and providing feedback) is thus reduced to the annotation with ACs. In our example of

problem 2.1, many of the ACs found are general to tasks involving the application of search algorithms to graphs (or trees).

While it is impossible to tell without further knowledge of the learner whether an answer ended up in a particular AC due to a misunderstanding or just sloppy work, it is possible to determine possible causes of errors and address them specifically in subsequent cohorts. Referring to AC_{30} in task 4, we discussed in plenary what the reason could be that so many students incorrectly applied the greedy search algorithm. The instructor assumes that one prominent example given in the lecture was misread in a way that supported the false reading of the algorithm. In the example, a city network was used, which was traversed step by step. This can create the misconception to only pay attention to nodes directly connected to the previously visited node because, when physically traveling, it is usually not optimal to travel back to an already visited city to try out an alternative route. Also, the possible cause of AC_{30} in task 5 made the educators decide to scale the arrows larger in the next exam.

The use of answer classes can induce an improvement in feedback (**BF**), especially in cases where limited correction time disincentivizes writing the same (helpful) feedback into the answer sheets over and over again. Just marking the ACs and thus referring to a general feedback table for the exam can allow significant practical improvements of **BF**. Even if it is impossible to perform steps 4 and 5 for every answer class in large exams, it can still be used to handle the largest groups of answers.

The question of whether ACs result in a more objective correction (**MOC**) can be answered clearly from our point of view. Due to predefined, objectively observable criteria, interpretations or subjective decisions are less likely, even when multiple humans are correcting an exam. If ACs are linked to numerical values, their combinations correspond to the task's scoring. It has also been found that corrections can be made much more fairly without additional effort. For instance, previously, students who started with the wrong starting node, but performed the algorithm correctly, did not get any points in the exam because all the mentioned nodes did not match the sample solution. However, these submissions are all in the same ACs (wrong starting node, correct algorithm). Thus, it is possible to realize a point deduction for the wrong start node but to consider consequential errors.

The annotation of answers with ACs enables a higher quality evaluation of the exam results, which are of great use for the further development of teaching. Particularly pronounced error classes can indicate weaknesses in the teaching material that trigger misconceptions and may even suggest ways to heal them. Moreover, using suitable tasks in subsequent years may help verify whether the suspected causes and remedies were effective. Incorrect or ambiguous tasks could be identified and corrected in advance (steps 1 and 2) (**BPE**), and potential for improvement could be identified after analyzing the submissions (step 3). Thus, applying the concept of answer classes stimulated incentives to improve teaching **IIT**, as the instructors developed new instructional materials that targeted uncovered problems and revised existing materials to avoid potential misconceptions in future courses.

8 Conclusion and Future Work

We have picked up the concept of answer classes from the Y-Model [Lo23] and evaluated it in practice in a large-scale written exam. We have fleshed out a practical procedure for implementing answer classes in a written presence exam with manual paper-based correction. The results of this experiment are very encouraging for the quality of education by implementing answer classes. To quote the instructor of the course (who is a co-author of this paper):

> *I was very skeptical whether this concept from secondary education could carry over to the university setting and whether the effort of AC annotation would be sustainable in a correction process that keeps my entire research group busy full-time for a week. Frankly, I only agreed to this to make my didactics colleague happy. [. . .] But the result of the experiment has utterly convinced me of the approach, even in a paper-based setting. Seeing the discussions among the graders induced by developing and annotating ACs and the content awareness in the grading process made the added effort worthwhile. [. . .], and that is before we have taken a closer look at the data that this experiment generated.*

Encouraged by this, we are now preparing to digitize the process presented in section 5 in a browser-based application that presents scanned exam papers and supports AC assignment, AC-based grading, AC-based feedback, and eventually exam review to also reap the scalability benefits conjectured in section 4 and discussed in section 8.

Developing and annotating ACs is a time-consuming process, and creating them individually in all situations may not be feasible. Therefore, future efforts should investigate the generalizability of answer classes, e.g., for similar and variant tasks, and whether any global answer classes can be used across all types of tasks.

We have not yet studied AC-based feedback generation and communication in detail. However, in the past, students received no feedback on their exam answers except from verbal discussions during exam review sessions. Therefore, the limited experience from the experiment reported in this paper shows the potential ACs to give standardized feedback that vastly improves the status quo. Future research must clarify whether AC-based feedback can fully replace individual feedback or whether the inclusion of, e.g., a learner model is necessary to automate learner-specific feedback.

Finally, the automation of AC assignment is out of scope for this paper and was left to future research. In single/multiple-choice tasks like problem 2.1 focused on above, the respective (combinations of) choices directly correspond to ACs, so automation is trivial. But about half of the exam tasks were more open-ended. Already for fill-in-the-blanks tasks, we have to pre-meditate or collect specific ACs, but given those, grading automation should be relatively easy to implement in general. For open-answer tasks, the problem of automating grading seems AI-hard – i.e., if we can solve that, we can also solve the general AI problem. In situations where we have a lot more data – with ACs and human classifications as a gold standard – supervised deep learning methods might be successful in the future.

Bibliography

[AP11] Ahmed, Ayesha; Pollitt, Alastair: Improving marking quality through a taxonomy of mark schemes. Assessment in Education: Principles, Policy & Practice, 18(3):259–278, 2011.

[Be16] Berges, Marc; Striewe, Michael; Shah, Philipp; Goedicke, Michael; Hubwieser, Peter: Towards Deriving Programming Competencies from Student Errors. In: 4th International Conference on Learning and Teaching in Computing and Engineering (LaTiCE). IEEE Press, Los Alamitos, pp. 19–23, 2016.

[Fa14] Falkner, Nickolas; Vivian, Rebecca; Piper, David; Falkner, Katrina: Increasing the effectiveness of automated assessment by increasing marking granularity and feedback units. In (Dougherty, John; Nagel, Kris; Decker, Adrienne; Eiselt, Kurt, eds): SIGCSE '14. ACM, New York, pp. 9–14, 2014.

[FUL22] Foss, Sarah; Urazova, Tatiana; Lawrence, Ramon: Automatic Generation and Marking of UML Database Design Diagrams. In (Merkle, Larry; Doyle, Maureen; Sheard, Judithe; Soh, Leen-Kiat; Dorn, Brian, eds): Proceedings of the 53rd ACM Technical Symposium on Computer Science Education. ACM, New York, NY, USA, pp. 626–632, 2022.

[HST02] Higgins, Colin; Symeonidis, Pavlos; Tsintsifas, Athanasios: The marking system for CourseMaster. SIGCSE Bull, 34(3):46–50, 2002.

[Ih10] Ihantola, Petri; Ahoniemi, Tuukka; Karavirta, Ville; Seppälä, Otto: Review of recent systems for automatic assessment of programming assignments. In (Schulte, Carsten; Suhonen, Jarkko, eds): Proceedings fo the 10th Koli Calling International Conference on Computing Education Research. ACM Press, New York, pp. 86–93, 2010.

[LFU23] Lawrence, Ramon; Foss, Sarah; Urazova, Tatiana: Evaluation of Submission Limits and Regression Penalties to Improve Student Behavior with Automatic Assessment Systems. ACM Transactions on Computing Education, 2023.

[Lo23] Lohr, Dominic; Berges, Marc; Kohlhase, Michael; Müller, Dennis; Rapp, Max: The Y Model - Formalization of Computer-Science Tasks in the Context of Adaptive Learning Systems. Submitted to GECON 2023, 2023.

[Na21] Nabil, Ramez; Mohamed, Nour Eldeen; Mahdy, Ahmed; Nader, Khaled; Essam, Shereen; Eliwa, Essam: EvalSeer: An Intelligent Gamified System for Programming Assignments Assessment. In (Bahaa-Eldin, Ayman, ed.): 2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC). IEEE, Piscataway, NJ, pp. 235–242, 2021.

[RA10] Reddy, Y. Malini; Andrade, Heidi: A review of rubric use in higher education. Assessment & Evaluation in Higher Education, 35(4):435–448, 2010.

[SS22] Strickroth, Sven; Striewe, Michael: Building a Corpus of Task-Based Grading and Feedback Systems for Learning and Teaching Programming. International Journal of Engineering Pedagogy (iJEP), 12(5):26–41, 2022.

[St16] Striewe, Michael: An architecture for modular grading and feedback generation for complex exercises. Science of Computer Programming, 129:35–47, November 2016.

[WS07] Wolf, Kenneth; Stevens, Ellen: The role of rubrics in advancing and assessing student learning. The Journal of Effective Teaching, 7:3–14, 2007.

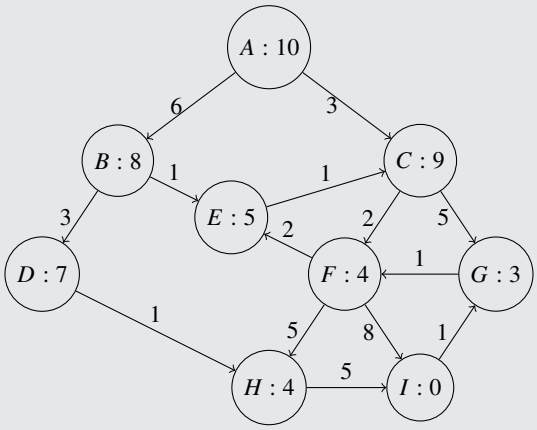
[Ze15] Zehetmeier, Daniela; Böttcher, Axel; Brüggemann-Klein, Anne; Thurner, Veronika: Development of a Classification Scheme for Errors Observed in the Process of Computer Programming Education. In: HEAd'15. Conference on Higher Education Advances. Editorial Universitat Politècnica de València, pp. 475–484, 2015.

9 Appendix

Task

Problem 2.1 (Search Algorithms) 8 pt

Consider the following graph:



Every node is labeled with $n : h(n)$ where n is the identifier of the node and $h(n)$ is the heuristic for estimating the cost from n to a goal node. Every edge is labeled with its actual cost.

Assume you have already expanded the node A. List the next four nodes (i.e., excluding A) that will be expanded using.

1. depth-first search 1 pt.
2. breadth-first search 1 pt.
3. uniform-cost search 2 pt.
4. greedy search 2 pt.
5. A*-search 2 pt.

If there is a tie, break it using alphabetical order.

Fig. 4: The analyzed task consisting of 5 subtasks