	**Międzyuniwersyteckie Centrum Informatyzacji** http://muci.edu.pl	**USOS 5.1.2**

Uniwersytecki System Obsługi Studiów

Sprawdziany w USOSweb

Algorytmy wystawiania ocen
i
Statystyki

Maciej Strzelczyk
Uniwersytet Warszawski

12 marca 2011

Spis treści

1. Wstęp

Witam w drugim podręczniku poświęconym modułowi *Sprawdziany* w USOSweb. Są w nim opisane dwie przydatne funkcje, których obecność została zasygnalizowana w pierwszej części. Mowa o automatycznym wystawianiu ocen oraz wyświetlaniu statystyk.

Wyobraźmy sobie, że pewien profesor prowadzi przedmiot, na który zapisanych jest 600 osób. Po egzaminie, każdemu ze swoich pomocników dał kilkadziesiąt prac do sprawdzenia. Dzięki modułowi *Sprawdziany*, każdy z tych pomocników wpisał już samodzielnie do USOSweb wyniki studentów, których prace sprawdzał. Nie mógł jednak przy okazji wystawić ocen, bo wykładowca nie zadecydował jeszcze, jakie będą progi punktowe. Prowadzący nie jest pewien jak studentom poszedł ten egzamin, dlatego przed podjęciem decyzji chciałby zapoznać się z wynikami. Tutaj bardzo pomocna okazuje się funkcja statystyk, jaką oferuje moduł *Sprawdziany*. Przy pomocy kilku kliknięć profesor może wygenerować na ekranie komputera wykres pokazujący rozkład wyników. Wciskając kilka klawiszy sprawdzi szybko ile osób osiągnęło więcej niż podana liczba punktów. Kiedy po zorientowaniu się w wynikach egzaminator ustali już progi punktowe na poszczególne oceny, staje przed monotonnym zadaniem wystawienia tych ocen setkom studentów. Na szczęście jednak wystarczy, że skorzysta z możliwości automatycznego wystawiania ocen, a moduł *Sprawdziany* w okamgnieniu wykona tę pracę za niego!

UWAGA! Moduł *Sprawdziany* został napisany tak, aby działał z najnowszymi wersjami wiodących przeglądarek internetowych (Firefox, Opera i Internet Explorer). Jeśli coś nie działa zgodnie z oczekiwaniami, to należy sprawdzić czy używa się najnowszej wersji przeglądarki.

2. Wstęp do algorytmów

Aby używać funkcji automatycznego wystawiania ocen, trzeba poznać podstawy algorytmiki. Sposób wystawiania ocen definiuje się w module *Sprawdziany* w postaci prostego algorytmu. W tym rozdziale omówię z czego składa się taki algorytm.

2.1. Co to algorytm?

Zgodnie z definicją zaczerpniętą z Wikipedii „***Algorytm*** *– w matematyce oraz informatyce – to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego rodzaju zadań.*”. Można by więc powiedzieć że jest to forma przepisu na rozwiązanie jakiegoś problemu, bądź zrobienie czegoś. Przyglądając się tej definicji od początku widzimy, że algorytm musi być skończony. Nic dziwnego, w końcu co nam po sposobie, którego wykonanie nigdy się nie skończy. Kolejną cechą jest uporządkowanie, które mówi nam w jakiej kolejności należy wykonywać czynności składowe. Te czynności muszą być jasno zdefiniowane, co za tym idzie, muszą być jednoznaczne.

W naszym wypadku problemem do rozwiązania jest wystawienie studentowi oceny. Natomiast sposób rozwiązania tego problemu będzie ułożony z prostych operacji, które wykona dla nas moduł *Sprawdziany*.

2.2. Zmienne

Podstawą działania algorytmów są zmienne. Każda zmienna ma jakąś nazwę oraz wartość. W jednym algorytmie nie mogą wystąpić dwie zmienne o tej samej nazwie. W

nazwie mogą występować litery alfabetu (bez polskich znaków), cyfry oraz kilka innych znaków jak na przykład podkreślenie _. W nazwach nie wolno używać spacji! Dla przykładu możemy mieć w algorytmie zmienną o nazwie suma_punktow, której wartością będzie 15. W informatyce często pojawia się konieczność zdefiniowania jaki typ wartości może przyjąć zmienna (liczby całkowite, ułamki, wartości logiczne czy napisy), jednak my teraz nie musimy się tym przejmować, ponieważ moduł *Sprawdziany* sam rozpozna z czym ma do czynienia. Zmienne będą mogły przyjmować wartości liczbowe, wartości logiczne (prawda i fałsz) oraz specjalne wartości reprezentujące oceny. Jest jeszcze specjalna wartość null reprezentująca brak wartości. Same zmienne nie byłyby wiele warte bez operacji jakie można na nich wykonywać. Ogólnie rzecz biorąc na zmiennych można działać jak na zwykłych liczbach, należy jednak pamiętać, że zgodnie z nazwą mogą one zmieniać swoją wartość. Zmienna zaczyna istnieć gdy tylko wpiszemy w treść algorytmu jej nazwę w jakimś działaniu, jej wartość początkowa to null.

Uwaga: Operacje, w których występuje zmienna o wartości null (czyli bez wartości) czasem postrzegają wartość null jako zero, a czasem jakoś inaczej. Dlatego, aby uniknąć błędów najlepiej unikać zmiennych o tej wartości.

Poniżej znajduje się tabela operacji jakie można wykonywać na zmiennych o wartościach liczbowych. W przykładach tych będę operował na zmiennych a, b i c.

Operacja	**Opis**
a = 1;	Przypisanie zmiennej a wartości 1
a = b;	Przypisanie zmiennej a wartości, którą ma zmienna b.
a = a + 1;	Przypisanie zmiennej a wartości o 1 większej niż miała do tej pory. Z matematycznego punktu widzenia jest to bzdura, z informatycznego w pełni poprawna operacja.
a = b + c;	Przypisanie zmiennej a wartości sumy b i c.
a = b – c;	Przypisanie zmiennej a wartości różnicy b i c.
a = b * c;	Przypisanie zmiennej a wartości iloczynu b i c.
a = b / c;	Przypisanie zmiennej a wartości ilorazu b i c. Dzielenie przez zero przerywa działanie algorytmu.

Dozwolone jest także łączenie operacji arytmetycznych w jednym wierszu, na przykład

```
delta = b*b + 4*a*c;
```

jest poprawną instrukcją algorytmu. Moduł wykonując takie bardziej złożone działania zachowuje kolejność działań zgodną z tą, której uczyli nas w podstawówce, czyli najpierw mnoży i dzieli, a później dopiero dodaje i odejmuje. Oczywiście jeśli chcemy wykonać działania w innej kolejności, to możemy użyć nawiasów, tak jak tutaj:

```
delta2 = b*(b + 4*a)*c;
```

W wyniku takiej instrukcji moduł *Sprawdziany* najpierw wykona mnożenie 4*a i doda do wyniku b, a następnie pomnoży wynik działania w nawiasie przez b i c. Należy też zauważyć, że delta2 jest poprawną nazwą zmiennej. W nazwach zmiennych wolno używać cyfr, ważne jest tylko, aby nie były one pierwszym znakiem w nazwie.

UWAGA: Po każdej instrukcji przypisania należy umieścić średnik!

Oprócz liczb, zmienne mogą przyjmować także wartości logiczne, takie wartości mogą też pojawić się w wyniku operacji na dwóch zmiennych liczbowych, na przykład jako wynik porównania. Na tym poziomie te wartości mogą się wydać jeszcze mało przydatne, ale są one często niezbędne do poprawnego wykonania algorytmu. Operacje porównania traktują null jak zero. Z wartościami logicznymi wiążą się następujące operacje:

Operacje na liczbach	**Opis**
a > b	Sprawdzenie czy a jest większe od b.
a >= b	Sprawdzenie czy a jest większe od lub równe b.
a < b	Sprawdzenie czy a jest mniejsze od b.
a <= b	Sprawdzenie czy a jest mniejsze od lub równe b.
a == b	Sprawdzenie czy a jest równe b.
a != b	Sprawdzenie czy a jest różne od b.
a === b	Sztywne sprawdzenie czy a jest równe b. Ta operacja różni się od operacji == tym, jak pojmuje wartość null. Dla operacji == null jest równy 0 i tak jest traktowany, natomiast dla operacji === null nie jest równy zero. Dlatego możemy sprawdzić czy zmienna jest === null bez obaw o sytuację gdy zmienna jest równa zero.
a !== b	Sztywne sprawdzenie czy a jest różne od b.
is_null(a)	Sprawdzenie czy zmienna a ma wartość null. Najwygodniejszy sposób na sprawdzenie tego warunku.

Operacje na wartościach logicznych:

Operacja	**Opis**
a = true;	Przypisanie zmiennej a wartości „prawda”
a = false;	Przypisanie zmiennej a wartości „fałsz”
a and b	Wynikiem operacji jest „prawda” jeśli zarówno a, jak i b mają wartość „prawda”.
a && b	j.w.
a or b	Wynikiem operacji jest „prawda” jeśli przynajmniej jedna ze zmiennych ma wartość „prawda”.
a || b	j.w.
not a	Operacja zaprzeczenia. Wynikiem jest „prawda” jeśli wartością a był „fałsz”
!a	j.w.

Operacje logiczne także można łączyć w dłuższe operacje i używać w nich nawiasów. Dla zwiększenia czytelności wręcz zalecane jest używanie nawiasów, ponieważ kolejności w działaniach logicznych nie są tak oczywiste jak w przypadku działań na liczbach. Najpierw wykonywane są operacje negacji (not, !), następnie operacje alternatywy (or, ||) i koniunkcji (and, &&), a na końcu operacje porównania dwóch liczb.

Poprawne jest takie skomplikowane wyrażenie logiczne:

```
((a > b) or (c > b)) and !(d)
```

Występują w nim trzy zmienne o wartościach liczbowych (a, b i c) oraz jedna o wartości logicznej (d).

Proszę się nie martwić jeśli to wszystko wydaje się obecnie niezrozumiałe. W dalszej części podręcznika pojawi się kilka przykładów, którymi mam nadzieję rozwieję ewentualne niejasności.

2.3. Instrukcje warunkowe

W poprzednim podrozdziale pojawiły się zmienne o wartościach logicznych i operacje, których wynikami były wartości logiczne. Są one potrzebne do obsłużenia instrukcji warunkowych. Te instrukcje działają tak, że sprawdzają jakiś warunek i jeśli jest on spełniony, to wykonują jakiś kawałek algorytmu, a jeśli nie jest, to wykonany zostaje inny kawałek. Dzięki temu możemy na przykład sprawdzić, która z dwóch zmiennych ma większą wartość, a następnie wykonać odpowiednie działanie w zależności od tego, czego się dowiedzieliśmy. Tutaj może drobny przykład, jak to mniej więcej wygląda:

Czy (a > b)?

Jeśli tak, to oznajmij światu, że a jest większe.

W przeciwnym przypadku oznajmij światu, że a nie jest większe.

Ogólny schemat instrukcji warunkowej, zapisanej w sposób zrozumiały dla modułu *Sprawdziany*, wygląda tak:

```
if (operacja, której wynikiem jest wartość logiczna)
{ [Ciąg instrukcji do wykonania jeśli operacja dała
   wynik „prawda”] }
elseif (warunek drugi) {[Ciąg instrukcji do wykonania
   jeśli warunek przy if nie został spełniony, ale
   spełniony jest warunek drugi]}
else { [Ciąg instrukcji do wykonania jeśli operacją dała
   wynik „fałsz”] }
```

Przykład sprawdzający, która z trzech liczb a, b i c jest największa:

```
if ((a >= b) and (a >= c))
{ najwieksza = a; }
elseif ((b >= a) and (b >= c)) {najwieksza = b; }
else {najwieksza = c; }
```

2.4. Określenie wyniku

Na końcu każdego algorytmu ważne jest, aby otrzymać wynik. W naszym przypadku wynikiem jest ocena jaką dostanie student. Do określenia wyniku działania algorytmu służy instrukcja return. Po niej musi wystąpić ocena lub null. W przypadku wyniku null, algorytm nie wystawi oceny. Po napotkaniu instrukcji return algorytm przerywa swoje działanie.

Oto jakie wartości ocen można podać po instrukcji return:

Wartość	**Opis**
[NK]	Student nieklasyfikowany
[ZAL]	Zaliczono
[NZAL]	Nie zaliczono
[ZWOL-LEK]	Zwolniony przez lekarza (WF)
[2]	Ocena 2
[3]	Ocena 3
[3,5] [3.5]	Ocena 3+
[4]	Ocena 4
[4,5] [4.5]	Ocena 4+
[5]	Ocena 5
[5!]	Ocena 5!

Ważne jest, aby po return występowała ocena zgodna z typem oceny do jakiej algorytm będzie przypisany. Niedopuszczalne jest, aby algorytm wystawił [ZAL], podczas gdy ocena jest typu standardowego (czyli od 2 do 5!).

W razie użycia skali ocen innej niż standardowo używane, na przykład ocen ECTS, wystarczy podczas wybierania typu oceny (podczas dodawania jej do modułu sprawdziany) zanotować jej możliwe wartości. Następnie można ich użyć w algorytmie po otoczeniu ich nawiasami kwadratowymi.

AM	Skala ocen amerykańska	F, E, D, C, B, A	wybierz
ECTS	Ocena ECTS	FX/F, E, D, C, B, A, A!	wybierz
SOCR	Skala ocen w programie wymiany	3, 4, 5, 6, 7, 8, 9, 10	wybierz
STD	Skala ocen standardowa	NK, 2, 3, 3,5, 4, 4,5, 5, 5!	wybierz
ZAL	Zaliczenie	NK, NZAL, ZAL	wybierz
ZAL-STD	Zaliczenie lub ocena w skali standardowej	NK, NZAL, ZAL, 2, 3, 3,5, 4, 4,5, 5, 5!	wybierz
ZAL-WF	Zaliczenie zajęć wychowania fizycznego	ZWOL-LEK, NZAL, ZAL	wybierz

Przykładowe zakończenie algorytmu przy pomocy instrukcji return (zawsze ze średnikiem po ocenie lub po wartości null):

```
return [3];
```

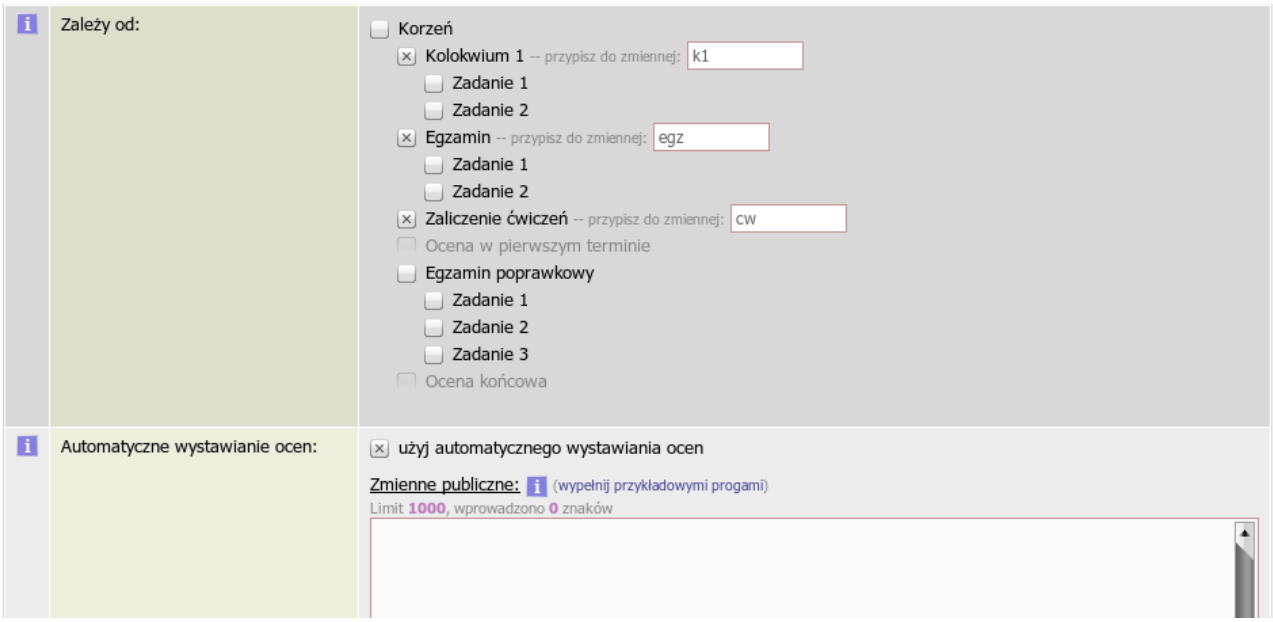
2.5. Komentarze

Można poprawić zrozumiałość algorytmu umieszczając w jego treści komentarze. Moduł *Sprawdziany* czytając algorytm, po napotkaniu znaku # uznaje całą resztę czytanego wiersza za komentarz i go ignoruje. Przy pomocy takich komentarzy będę objaśniał krok po kroku co dzieję się w każdym wierszu algorytmu.

3. Pisanie algorytmu

Pierwszym krokiem do napisania algorytmu wystawiania ocen jest oczywiście podjęcie decyzji jak ma wyglądać zaliczenie przedmiotu i według jakich zasad będziemy wystawiać oceny. Często ocena końcowa będzie zależała nie tylko od wyniku egzaminu, ale też od liczby punktów zdobytych w ciągu semestru.

Kiedy już ustalimy od czego zależeć będzie ocena i wprowadzimy do modułu *Sprawdziany* węzły odpowiadające tym zależnościom, możemy przejść do zdefiniowania oceny w module. Po dodaniu oceny do struktury zasad rozliczania, należy wejść w edycję jej właściwości (węzeł → Właściwości). Tam należy zaznaczyć opcję „użyj automatycznego wystawiania ocen". Powinny się wówczas pojawić dwa pola do wpisywania tekstu. Zanim jednak zajmiemy się tymi polami, należy zaznaczyć węzły, od których zależeć będzie ocena.



Warto zwrócić uwagę na pola obok zaznaczonych węzłów. Pojawiają się one tylko przy zaznaczonej opcji „użyj automatycznego wystawiania ocen" i pozwalają nam powiedzieć do jakich zmiennych chcemy przypisać wartości tych węzłów. Warto pamiętać, że wartością folderu (czyli np. sprawdzianu) jest suma punktów zdobytych przez studenta w zadaniach należących do tego folderu. Zgodnie z obrazkiem powyżej, przypisujemy zmiennej **k1** liczbę punktów zdobytych przez studenta na kolokwium, zmiennej **egz** przypisujemy liczbę punktów zdobytych na egzaminie, a zmiennej **cw** przypisujemy ocenę ([ZAL]/[NZAL]) oznaczającą zaliczenie ćwiczeń.

W tym wypadku będziemy omawiać ocenę wystawianą w pierwszym terminie, będzie ona zależała od liczby punktów zdobytych przez studenta na kolokwium (max 20 punktów) i egzaminie (max 50 punktów), konieczne też będzie zaliczenie ćwiczeń. Bez zaliczenia ćwiczeń student nie powinien być dopuszczony do egzaminu w pierwszym terminie, a więc sprawimy, że algorytm nie wystawi mu oceny.

3.1. Zmienne publiczne

W pierwszej kolejności zdefiniujemy zmienne publiczne. Można to zrobić w pierwszym polu tekstowym pod listą węzłów, od których zależy ocena. Zmienne publiczne to najczęściej progi punktowe, choć nic nie stoi na przeszkodzie aby wykorzystać je do czegoś innego. Są one dostępne później w algorytmie. Przykładowe zdefiniowanie progów punktowych może wyglądać tak:

```
na50 = 63; # minimalna liczba punktów na ocenę 5
na45 = 54; # minimalna liczba punktów na ocenę 4,5
na40 = 46; # minimalna liczba punktów na ocenę 4
na35 = 37; # minimalna liczba punktów na ocenę 3,5
na30 = 28; # minimalna liczba punktów na ocenę 3
```

Dzięki tym przypisaniom będziemy mieli w treści algorytmu progi punktowe w postaci zmiennych. Zmienna na30 będzie miała wartość 28, na35 będzie miała wartość 37 itd. W ten sposób zapisaliśmy, że trzeba mieć z egzaminu i kolokwium w sumie 28 punktów na ocenę 3, 37 punktów na 3+, 46 punktów na 4, 54 punkty na 4+ i 63 punkty na 5. Maksymalna liczba punktów jaką może uzyskać student to 70, więc jak widać nasze progi są dość łagodne. Jeśli przewidujemy, że progi punktowe lub suma ocen, jakie można zdobyć ulegną zmianie, możemy ustalić progi jako procent możliwych do zdobycia punktów. Oto jak tego dokonać:

```
max_punktow = 70; #ustalam ile punktów może zdobyć student
na50 = max_punktow*0.90; #ustalam próg na ocenę 5 na 90%
na45 = max_punktow*0.77; #ustalam próg na ocenę 4+ na 77%
na40 = max_punktow*0.65; #ustalam próg na ocenę 4 na 65%
na35 = max_punktow*0.52; #ustalam próg na ocenę 3+ na 52%
na30 = max_punktow*0.40; #ustalam próg na ocenę 3 na 40%
```

UWAGA! Zmienne publiczne są widoczne także dla studentów. Można zatem w ten sposób przekazać studentom informacje o zastosowanych progach.

3.2. Treść algorytmu

Kiedy mamy już ustalone progi punktowe dla ocen, pora przejść do poinstruowania modułu *Sprawdziany*, jak ma tych progów użyć. System wykona wpisany przez nas algorytm osobno dla każdego studenta. W przykładzie obecnie omawianym, warunkiem otrzymania oceny z egzaminu jest otrzymanie oceny ZAL z ćwiczeń. W takim razie pierwszą rzeczą, jaką zrobimy w algorytmie jest sprawdzenie, czy rozpatrywany student zaliczył ćwiczenia. Z pomocą przyjdzie nam instrukcja warunkowa if. Porównamy w niej ocenę otrzymaną przez studenta z ćwiczeń (którą na początku rozdziału przypisaliśmy na zmienną o nazwie cw) z wartością [ZAL] oznaczającą zaliczenie. Jeśli ocena, którą otrzymał student różni się od [ZAL] oznacza to, że otrzymał on [NZAL] lub [NK]. Możliwe też, że nie dostał żadnej oceny, ale w takim wypadku uznajemy, że nie zaliczył ćwiczeń. Oto jak będzie wyglądała instrukcja sprawdzająca, czy student zaliczył ćwiczenia:

```
if (cw !== [ZAL]) {
   return null;
}
```

W pierwszym wierszu kodu sprawdzam, czy zmienna cw jest różna on [ZAL]. Jeśli tak jest, to wykonana zostaje instrukcja return null; która oznacza zakończenie wykonywania algorytmu i ustalenie jego wyniku na null, czyli praktycznie brak wyniku. W efekcie wszyscy studenci, którzy nie mają z ćwiczeń wystawionej w module oceny ZAL, nie otrzymają oceny z pierwszego terminu.

Przejdźmy teraz do wystawiania konkretnych ocen. Na początek będziemy musieli ustalić ile punktów otrzymał student z egzaminu i kolokwium. Zakładając, że oba sprawdziany są już sprawdzone, a ich wyniki wprowadzone do modułu *Sprawdziany*, możemy dodać je do siebie i zapisać jako jedną zmienną.

```
suma = egz + k1;#suma oznacza sumę punktów z egzaminu i kolokwium
```

Następne co musimy zrobić, to sprawdzić po kolei, na którą ocenę zasłużył student. Zrobimy to przy pomocy serii instrukcji warunkowych.

```
if (suma >= na50) { return [5]; }
if (suma >= na45) { return [4,5]; }
if (suma >= na40) { return [4]; }
if (suma >= na35) { return [3,5]; }
if (suma >= na30) { return [3]; }
return [2]; #student nie spełnił wymagań na żadną pozytywną ocenę
```

W podanym kodzie sprawdzamy w kolejnych wierszach, czy student ma wystarczająco dużo punktów, aby uzyskać daną ocenę. Jeśli ma mniej punktów niż potrzeba na piątkę, to sprawdzamy, czy ma dość punktów na 4+ itd. Jeśli któryś z warunków będzie spełniony, to algorytm zakończy swoje działanie wystawiając odpowiednią ocenę. Jeśli żaden z warunków nie zostanie spełniony, będzie to oznaczało, że suma punktów uzyskanych przez studenta jest mniejsza niż tego wymagamy na ocenę 3, więc algorytm kończy działanie wystawiając ocenę 2. Zamienienie 4 ostatnich if na elseif nie zmieniłoby działania algorytmu, obie wersje są poprawne.

W całości zawartość pola Algorytm będzie wyglądała tak:

```
if (cw !== [ZAL]) {
   return null;
}
suma = egz + k1;#suma oznacza sumę punktów z egzaminu i kolokwium
if (suma >= na50) { return [5]; }
if (suma >= na45) { return [4,5]; }
if (suma >= na40) { return [4]; }
if (suma >= na35) { return [3,5]; }
if (suma >= na30) { return [3]; }
return [2]; #student nie spełnił wymagań na żadną pozytywną ocenę
```

Modyfikując ten przykład można z łatwością uzyskać algorytm, który będzie realizował najpospolitszy scenariusz wystawiania ocen.

3.3. Inne przykłady

Załóżmy, że w przykładzie z poprzednich rozdziałów chcielibyśmy dodać warunek, że do egzaminu przystąpić może jedynie osoba, która uzyskała z kolokwium co najmniej 10 punków. Możemy to zapisać w algorytmie na dwa sposoby. Pierwszy polega na dopisaniu jeszcze jednej instrukcji warunkowej na samym początku algorytmu, wtedy cały kod wyglądałby tak:

```
if (k1 < 10) { return null; } # warunek na punkty z kolokwium
if (cw !== [ZAL]) { return null; }

suma = egz + k1;
if (suma >= na50) { return [5]; }
if (suma >= na45) { return [4,5]; }
if (suma >= na40) { return [4]; }
if (suma >= na35) { return [3,5]; }
if (suma >= na30) { return [3]; }
return [2];
```

Możemy zrobić to też krócej rozbudowując warunek w już istniejącej instrukcji warunkowej. Wtedy nasz kod wyglądałby następująco:

```
 if ((cw !== [ZAL]) or (k1 < 10)) { return null; }

suma = egz + k1;
if (suma >= na50) { return [5]; }
if (suma >= na45) { return [4,5]; }
if (suma >= na40) { return [4]; }
if (suma >= na35) { return [3,5]; }
if (suma >= na30) { return [3]; }
return [2];
```

Teraz warunek w pierwszym wierszu możemy rozumieć następująco: Jeśli ocena z ćwiczeń nie jest równa ZAL lub liczba punktów zdobytych na kolokwium jest mniejsza niż 10, to zakończ działanie nie wystawiając oceny. Ważne jest by pamiętać o nawiasach wokół poszczególnych wyrażeń logicznych. Dzięki nim będziemy mieli pewność, że moduł *Sprawdziany* wykona je i zinterpretuje zgodnie z naszą wolą.

Bardziej skomplikowanym przypadkiem mogą być następujące zasady zaliczania: studenci mieli w trakcie semestru do wykonania trudne zadanie. Ci, którym się udało je wykonać otrzymają z egzaminu ocenę o połowę wyższą (o ile normalnie dostaliby ocenę pozytywną). Czyli Ci, którzy dostaliby z egzaminu 2, wciąż będą mieli 2, ale Ci, którzy dostaliby 3, dostaną 3+ o ile wykonali tamto zadanie. W naszym przykładzie zmienna **zad** będzie przechowywała informację, czy studentowi udało się wykonać zadanie (1 punkt jeśli tak, 0 punktów jeśli nie). Na zmiennej **egz** będziemy mieć przypisane punkty otrzymane na egzaminie. Oto jak może wyglądać algorytm wystawiający ocenę według podanego opisu.

```
if (egz >= na50) {
   if (zad == 1) { return [5!]; }
   else { return [5]; }
if (egz >= na45) {
   if (zad == 1) { return [5]; }
   else { return [4,5]; }
if (egz >= na40) {
   if (zad == 1) { return [4,5]; }
   else { return [4]; }
if (egz >= na35) {
   if (zad == 1) { return [4]; }
   else { return [3,5]; }
if (egz >= na30) {
   if (zad == 1) { return [3,5]; }
   else { return [3]; }
return [2];
```

Innym, również poprawnym zapisem podanych zasad zaliczania jest też poniższy algorytm:

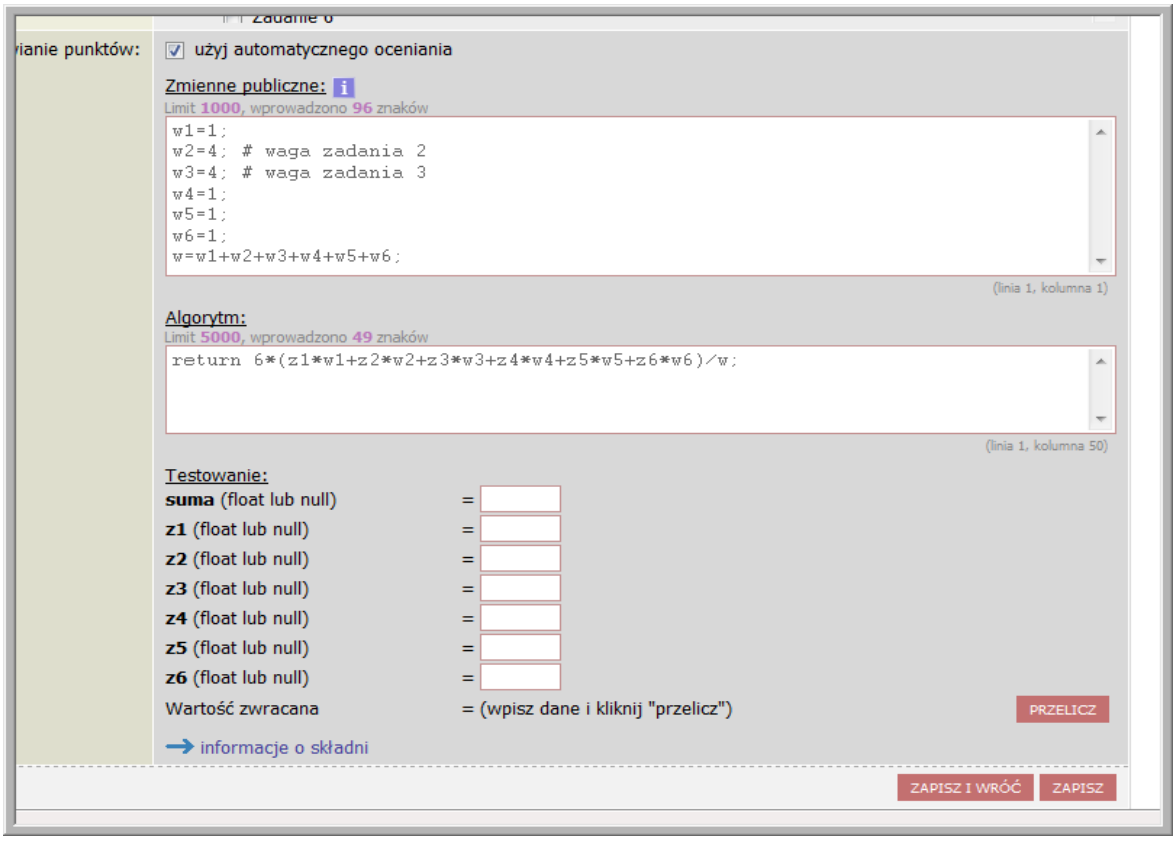
```
if (zad == 1) {
   if (suma >= na50) { return [5!]; }
   if (suma >= na45) { return [5]; }
   if (suma >= na40) { return [4,5]; }
   if (suma >= na35) { return [4]; }
   if (suma >= na30) { return [3,5]; }
} else {
   if (suma >= na50) { return [5]; }
   if (suma >= na45) { return [4,5]; }
   if (suma >= na40) { return [4]; }
   if (suma >= na35) { return [3,5]; }
   if (suma >= na30) { return [3]; }
}
return [2];
```

4. Automatyczne ocenianie zadań

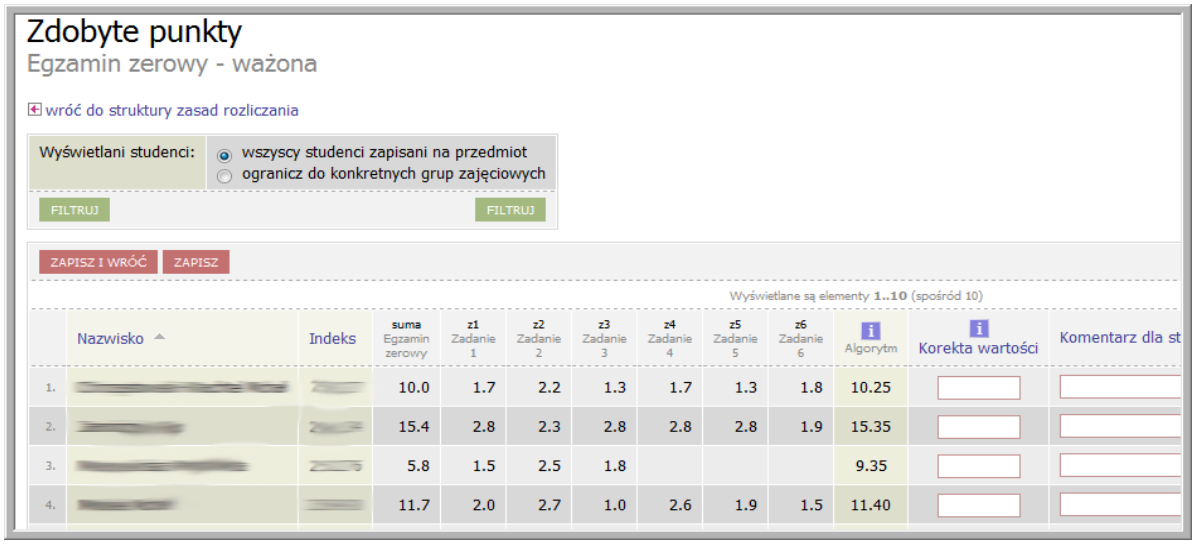
Możliwość automatycznego oceniania dotyczy nie tylko węzłów z ocenami, lecz również węzłów zadań. Jeśli algorytm oceniania jest skomplikowany, to można go podzielić na kilka logicznych części, a wynik każdej z nich zaprezentować w oddzielnym węźle.

Każdy z takich „pośrednich" wyników można odrębnie odsłonić studentom, a także można oglądać statystyki takich wyników.

We właściwościach węzła typu zadanie są dostępne pola takie jak w ocenach. Można z ich pomocą zdefiniować algorytm, podobnie jak w ocenach, ale przekazujący wartości liczbowe.

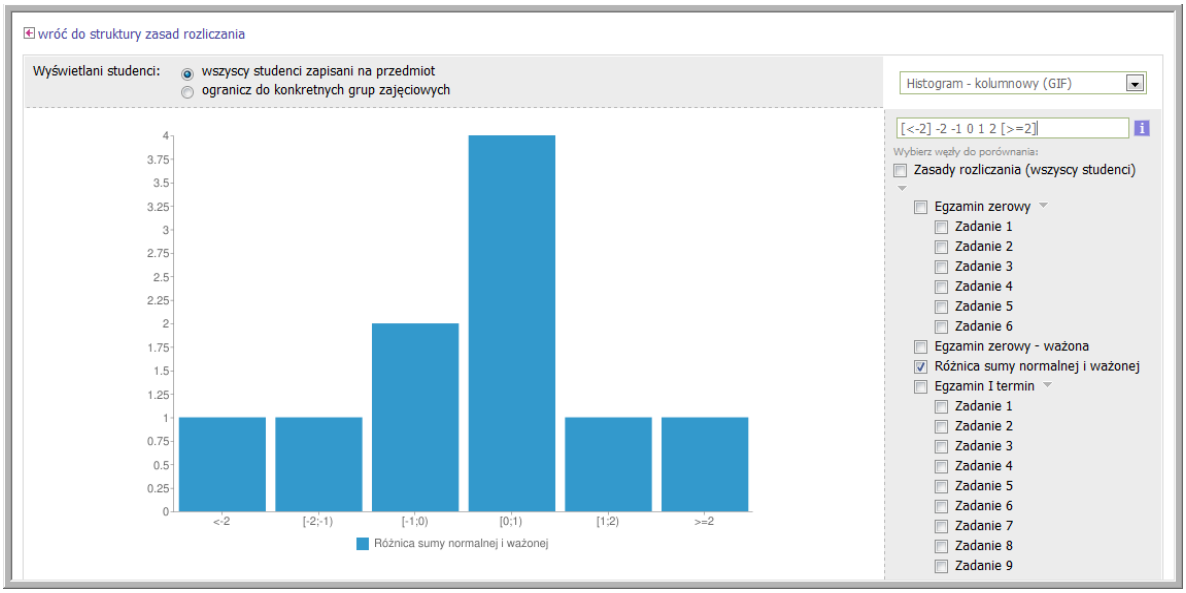


Podobnie, jak w przypadku ocen, wartości dostarczone przez algorytm można nadpisywać - na tym ekranie widać też, że suma dostarczona przez algorytm jest inna niż suma zwykła.

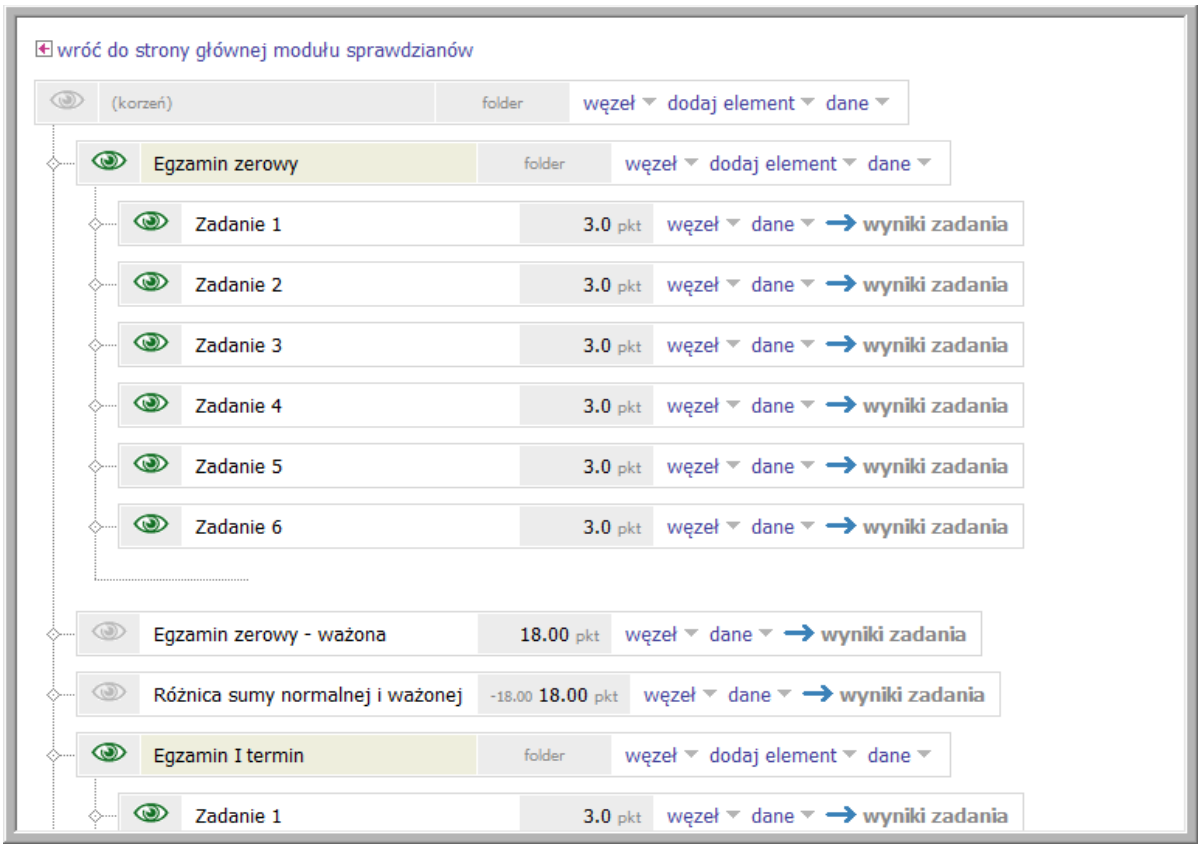


Utworzone w ten sposób węzły można wykorzystać w innych węzłach z algorytmami. Czyli np. przy wyliczaniu oceny można brać średnie ważone, zamiast zwykłych sum.

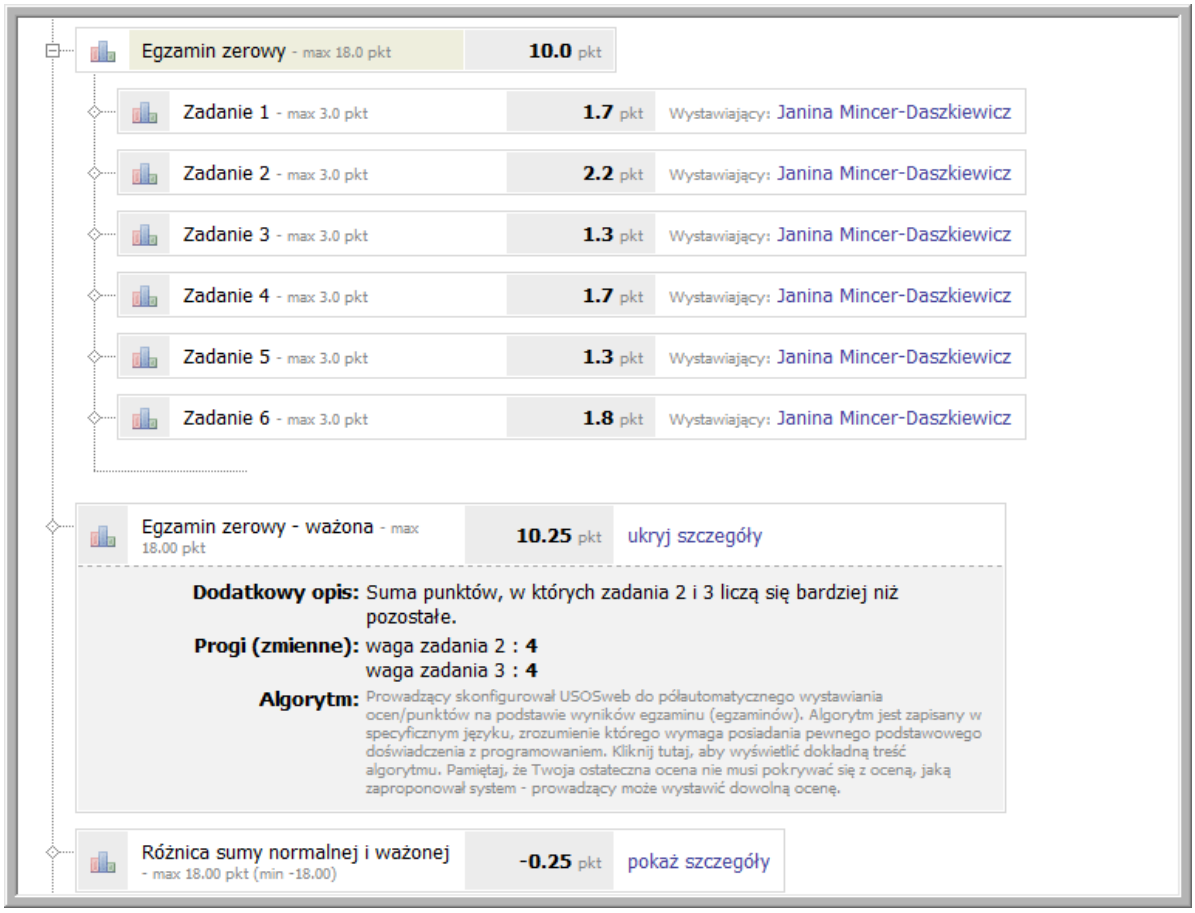
Można również wykorzystać te węzły w statystykach, czyli można zrobić np. wykres dla średnich ważonych lub dla innych PROSTYCH matematycznych wzorów.



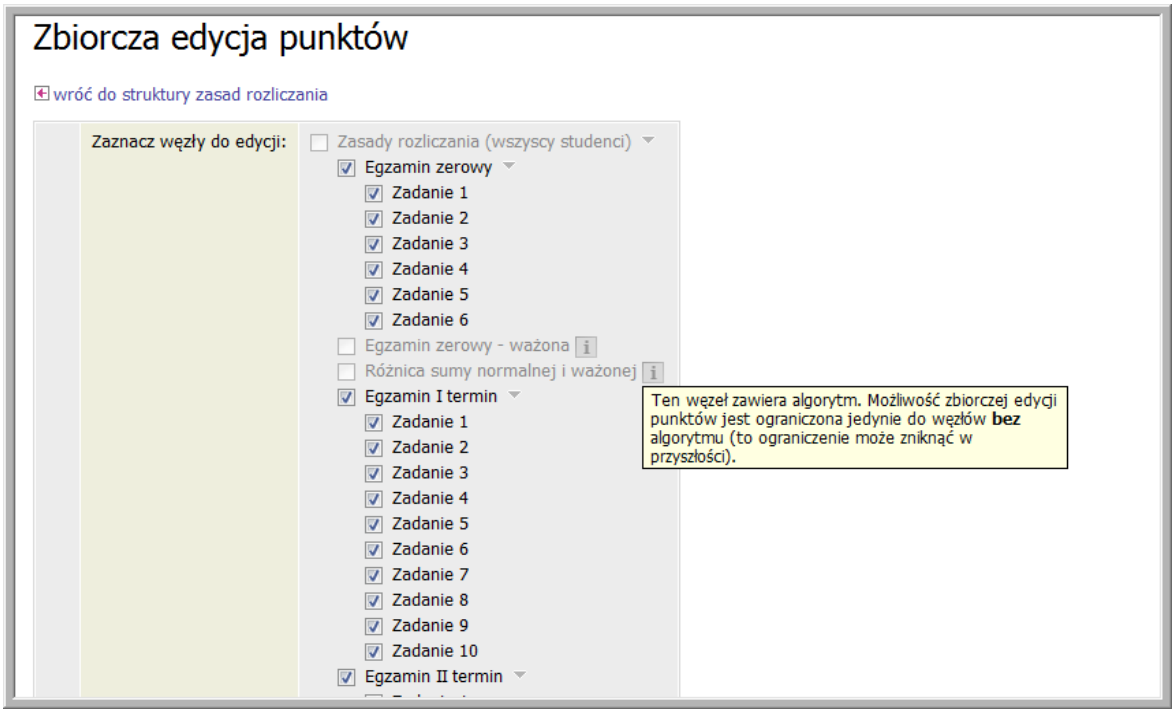
Węzły można ukrywać i pokazywać studentom.



Studenci widzą wyniki takich zadań oraz opisy i algorytmy, podobnie jak w ocenach:

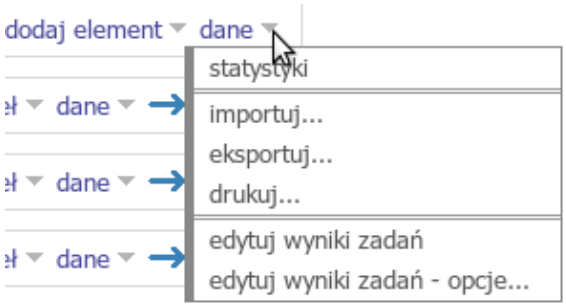


Przy edycji zbiorczej węzłów z algorytmem NIE można edytować (podobnie jak nie można edytować ocen).

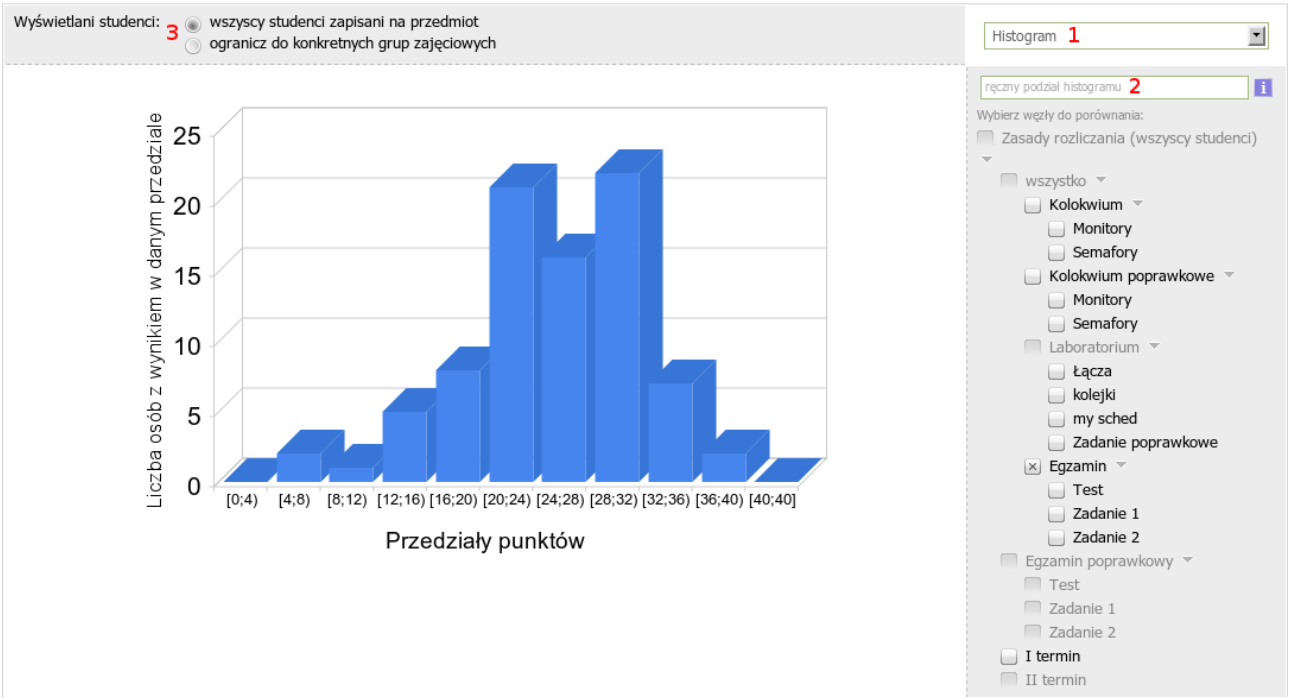


5. Statystyka w module *Sprawdziany*

Moduł *Sprawdziany* oferuje bardzo wygodne narzędzie do analizowania wyników studentów, jakim jest funkcja statystyk (do statystyk mają dostęp również studenci). Możemy jej użyć wchodząc do widoku struktury wybranego zestawu zasad, a następnie przy dowolnym węźle klikając w odnośnik „dane” i wybierając z listy opcję „statystyki”.



Dostaniemy się wtedy na stronę, na której możemy wyświetlać różnego rodzaju wykresy dotyczące osiągnięć studentów. Powinna ona wyglądać mniej więcej tak:



Na tej stronie możemy wybrać jakiego rodzaju wykres i zarazem statystykę chcemy zobaczyć (1), pełną listę możliwych wyborów omówię za chwilę. Możemy też ręcznie wprowadzić podział statystyki (2), ta opcja działa jedynie w przypadku wyświetlania histogramów. Istnieje również możliwość ograniczenia badanych wyników do studentów konkretnych grup (3), ta opcja przydaje się, kiedy chcemy sprawdzić jak poszło studentom z naszej grupy ćwiczeniowej. Przy pomocy listy pod polem podziału statystyki, znajduje się lista wszystkich węzłów z zasad rozliczania. Dzięki niej możemy na jednym wykresie wyświetlić na przykład wyniki z egzaminu w pierwszym i w drugim terminie. Zachęcam do samodzielnego pobawienia się narzędziem statystyk i eksperymentowania, przy jego pomocy wyniki są jedynie odczytywane, więc nie ma obawy, że coś zepsujemy.

5.1. Rodzaje wykresów

Generator statystyk daje nam do wyboru sześć różnych rodzajów wykresów. Każdy z nich dostępny jest w kilku odmianach. Poniżej przedstawiam ich listę wraz z krótkim opisem.

• Histogramy

Histogramy są podstawowym rodzajem wykresów oferowanych przez moduł *Sprawdziany*. Przedstawiają one wykres, w którym przedziałom punktowym jest przypisana liczba osób, których wynik mieści się w tym przedziale.

Histogram kolumnowy jest domyślnym rodzajem wykresu. Pod każdą kolumną jest przedstawiony przedział punktowy, którego dana kolumna dotyczy. Wysokości kolumn odpowiadają liczbie osób. Histogram kolumnowy dostępny jest w domyślnej, ładnie wyglądającej formie wykresu 3D oraz w formacie GIF. Zaletą tego drugiego formatu jest możliwość kliknięcia na wykres prawym przyciskiem myszy i wybranie opcji „Zapisz obraz jako...” w celu zapisania wykresu na dysku komputera.

Histogramy powierzchniowe i liniowe są bardzo do siebie podobne. W obu przypadkach zamiast ze słupkami, mamy do czynienia z kropkami umieszczonymi na różnych wysokościach, połączonymi ze sobą linią. W histogramie powierzchniowym, obszar poniżej linii i kropek jest zapełniony jakimś kolorem.

Histogram – tabelka. Przy tej opcji moduł wygeneruje dla nas tabelkę, w wierszach której znajdować się będą przedziały punktowe i dane o tym, ilu studentów znajduje się w danym przedziale.

Ręczny podział histogramu. Wykresy w postaci histogramu pozwalają nam zadecydować, jakie przedziały punktowe mają zostać zaprezentowane. Wyboru tego dokonujemy przy pomocy pola (2) znajdującego się nad listą węzłów. Jeśli wpiszemy tam liczby oddzielone spacjami, np. „5 10 15 20”, to na histogramie pojawi się 5 następujących przedziałów: [0; 5) [5; 10) [10; 15) [15; 20) [20; max]. Zapis [10; 15) oznacza następujący zbiór punktów {10, 11, 12, 13, 14}. Czyli ogólniej mówiąc liczba przy nawiasie kwadratowym należy do zbioru, a liczba przy nawiasie okrągłym już nie.

Jeśli chcemy komuś pokazać później nasz diagram, to możemy chcieć skorzystać z możliwości wyświetlenia pod kolumnami etykiet zamiast przedziałów punktowych. Jeśli chcemy zrobić wykres pokazujący ilu studentów zaliczyło egzamin (próg punktowy niech będzie równy 20), to możemy wpisać: „[nie zdali] 20 [zdali]”. Taka formułka sprawi, że otrzymamy wykres o dwóch kolumnach etykietowanych odpowiednio napisami „nie zdali” i „zdali”. Możemy oczywiście umieścić więcej etykietowanych kolumn w następujący sposób: „[etykieta_1] próg_1 [etykieta_2] próg_2 [etykieta_3] próg_3 … próg_n [etykieta_n+1]”, gdzie próg_x to liczba, a etykiety to napisy.

Przy pomocy polecenia „iter x”, gdzie x jest jakąś liczbą możemy sprawdzić, że każda kolumna będzie przedstawiała przedział punktowy wielkości x. Na przykład wpisanie „iter 1” sprawi, że każda kolumna będzie odpowiadała jednej wartości.

• Liczba osób

Wykresy liczby osób pozwalają nam porównać liczby osób, które otrzymały wyniki z różnych sprawdzianów/zadań. Wystarczy, że zaznaczymy na przykład egzamin i egzamin

poprawkowy, a ujrzymy wykres porównujący liczbę osób, które brały udział w tych egzaminach. Wykres jest dostępny w postaci kolumnowej 3D oraz w formacie GIF, który da się łatwo zapisać na dysku komputera. Wykres kołowy przedstawia proporcje liczby osób piszących. Jeśli pola są mniej więcej równej wielkości, oznacza to, że mniej więcej tyle samo osób napisało oba egzaminy. Możemy sobie również zażyczyć od modułu ładnej i przejrzystej tabelki, w której zostaną umieszczone odpowiednie dane.

• Liczba osób z grup

Ten rodzaj wykresów działa najlepiej, kiedy ograniczymy wyświetlanych studentów do kilku wybranych grup (3). Przy pomocy tej grupy statystyk będziemy mogli zorientować się ile osób z jakich grup otrzymało wyniki z jakiegoś zdania/sprawdzianu. Jak zwykle mamy do wyboru wersję kolumnową 3D i w formacie GIF. Oprócz nich do wyboru mamy tylko tabelkę.

• % osób z grup

Ta rodzina wykresów jest bardzo zbliżona do poprzedniej, jedyną różnicą jest to, że zamiast porównywać liczbę osób jakie dostały wyniki, porównywany jest ich odsetek,to znaczy ile procent osób z danej grupy uzyskało wyniki z danego sprawdzianu/zadania.

• średni wynik w %

Przy pomocy tej grupy wykresów możemy łatwo porównać średnie wyniki studentów w dwóch różnych sprawdzianach/egzaminach. Jeśli zaznaczymy np. egzamin i egzamin poprawkowy, to dostaniemy dwie kolumny, każda będzie przedstawiała średni wynik z jednego z egzaminów. Do wyboru mamy wersję kolumnową 3D i GIF oraz tabelkę.

• średni wynik w % z grup

Dzięki tym wykresom porównamy osiągnięcia studentów z wybranych grup. Możemy zaznaczyć węzeł przedstawiający egzamin, a nad polem wykresu wybrać tylko kilka grup ćwiczeniowych. Wtedy wygenerowany zostanie wykres porównujący średnie osiągnięcia wybranych grup. Wynik przedstawiony zostanie w postaci kolumn 3D, GIF lub jako tabela.

Możliwość przejrzenia wyników w tak wielu postaciach jest bardzo pomocna przy ocenianiu, czy egzamin był łatwy, czy nie. Dzięki powyższym funkcjom można łatwo porównać, która osoba prowadząca ćwiczenia najlepiej przygotowała studentów do egzaminu. Należy pamiętać, że studenci również mają dostęp do tej części modułu *Sprawdziany*, tak więc mogą się sami przekonać jak wyglądają ich wyniki na tle pozostałych.