Zadanie 5 - pomidorki

dokumentacja końcowa

Igor Kaźmierczak 293118 Bartłomiej Olber 300237

21 stycznia 2021

Spis treści

AAL Dokumentacja

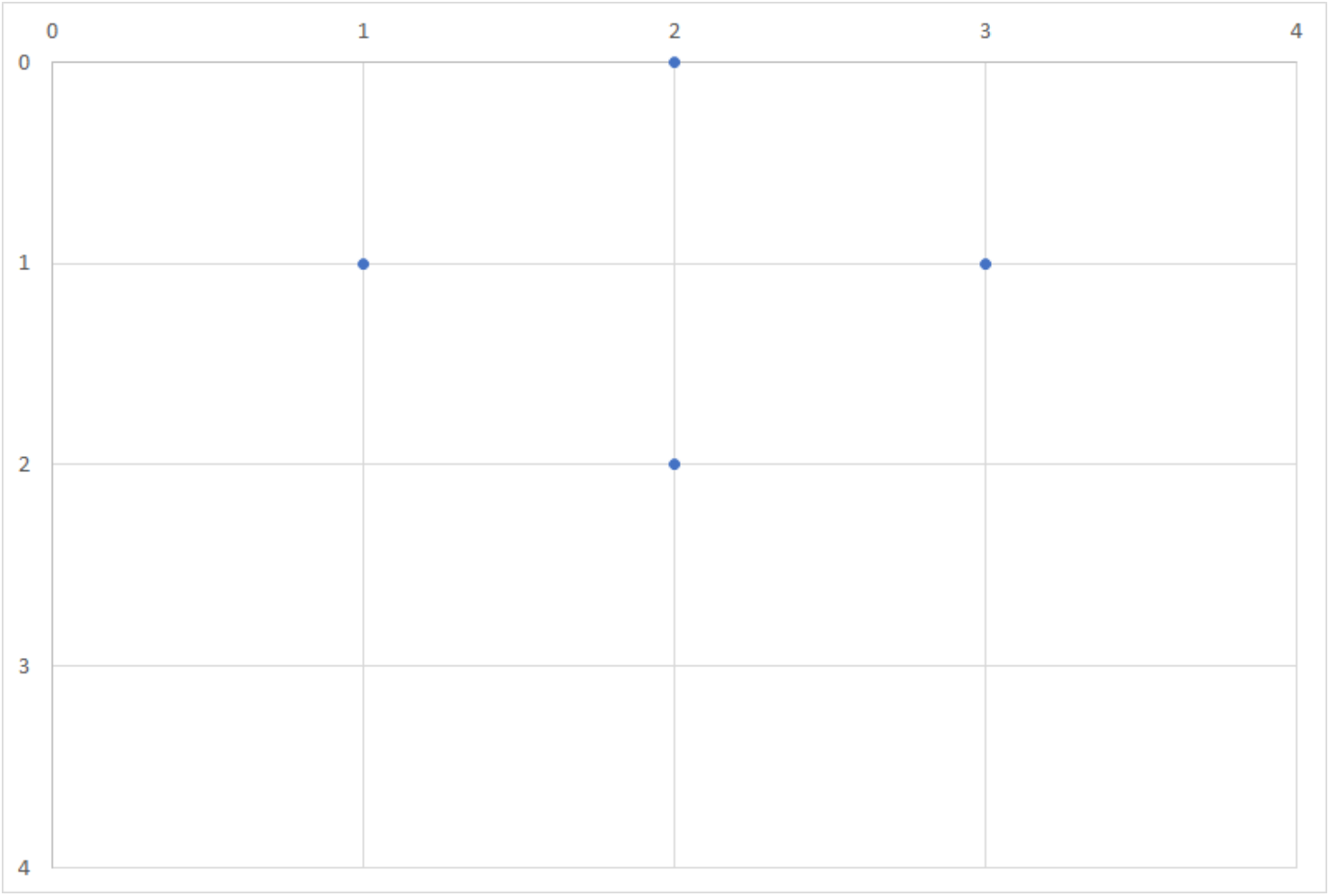
Treść zadania

Rolnik musi zabezpieczyc swoja plantacje pomidorków przed inwazja stonki pomidorozernej poprzez przykrycie krzaków pomidorków płachta przezroczystego plastiku. Niestety - plantacja jest wieksza niz posiadana przez rolnika płachta, zatem musi on wybrac takie jej położenie, w którym mozliwie duza liczba krzaków bedzie przykryta. Plantacja ma postac kwadratu o bokach 60000 na 60000 metrów. Rolnik dysponuje płachta o wymiarach a $\times$ b, gdzie a, b $<$ 10000. Na plantacji rosnie k $<$ 15000 krzaków pomidorków o znanych rolnikowi współrzednych. Zaproponuj algorytm, który musi dla ustalonego zbioru współrzednych krzaków pomidorków zaproponuje optymalne położenie płachty plastiku (przy czym przyjmujemy, iz krzak znajdujacy sie na brzegu płachty jest takze przez nia przykryty).

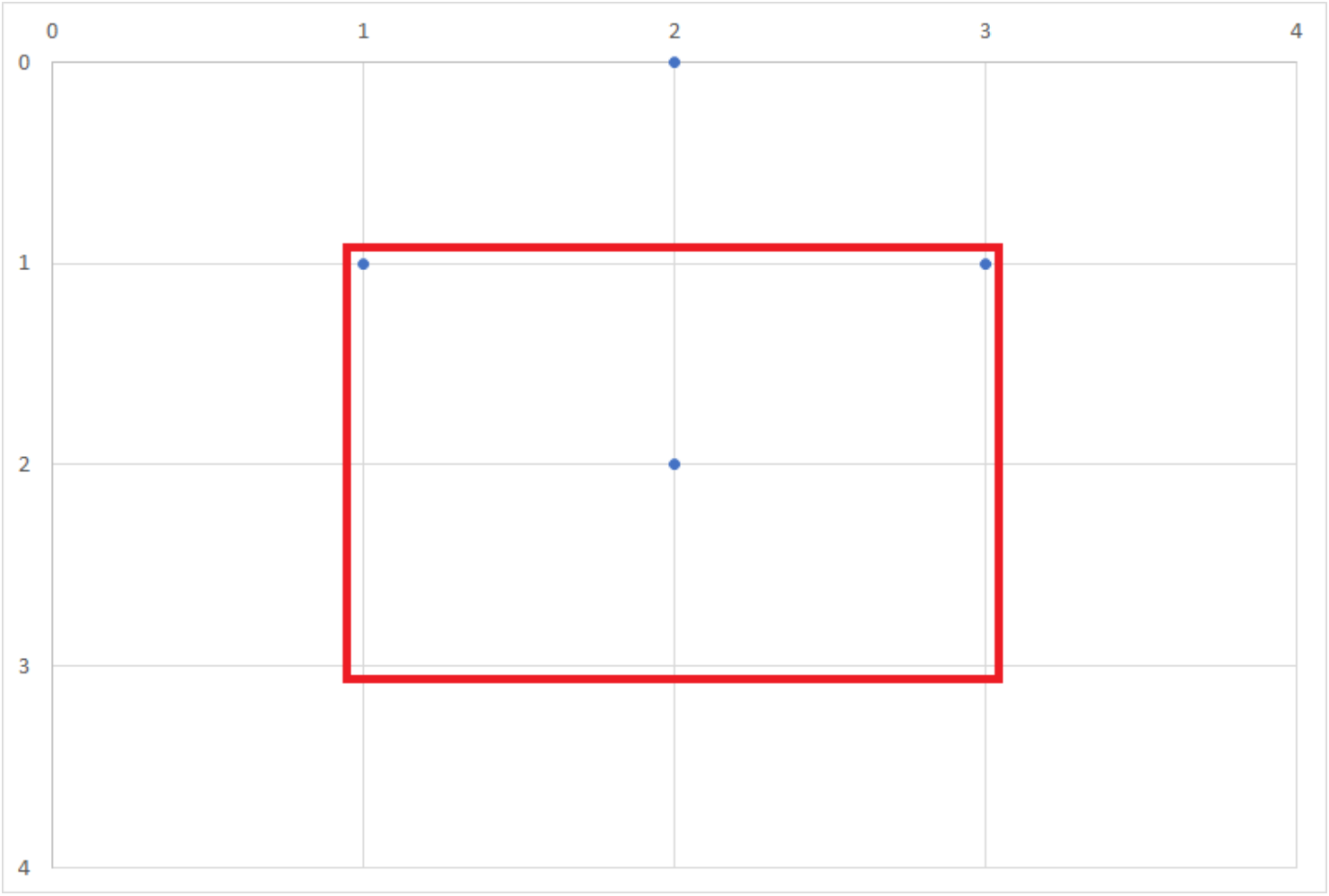
Opis problemu

Zadanie, choć może się wydawać proste a jego treść zdegenerowana do "znalezenia najlepszego prostokąta w kwadracie", okazało się być dosyć wymagające, szczególnie pod względem złożoności czasowej i pamięciowej. Pierwsze rozwiązanie jakie przychodziło do głowy to sprawdzenie każdego możliwego punktu (każdej współrzędnej) pola. Szybko okazało się, że pomysł ten jest całkowicie niedopuszczalny - wszystkich współrzędnych na polu jest n^2 co dla $n = 60000$ daje liczbę 3600000000. Wartość ta nie mieści się chociażby w podstawowym typie `int`. O wiele lepszym pomysłem okazało się przechowywanie informacji o polu w postaci współrzędnych tych punktów na polu, na których znajdują się krzaki pomidorów. Dla tej reprezentacji pola będziemy potrzebować maksymalnie $k = 15000$ par współrzędnych.

Koncepcja, jaka się rodzi mając taką reprezentację położenia to sprawdzenie dla każdego z krzaków ile pomidorów byłoby przykrytych przez płachtę przyłożoną (rozpoczynającą się) w miejscu rozpatrywanego krzaka. Niestety pomysł nie jest prawidłowy gdyż nie jest prawdą, że najlepszy punkt zaczepienia płachty będzie się zawsze znajodwał w miejscu występowania jednego z krzaków pomidorów, dobrze pokazuje to poniższy przykład (punkty przecięcia linii symbolizują miejsca w których mogą występować krzaki)

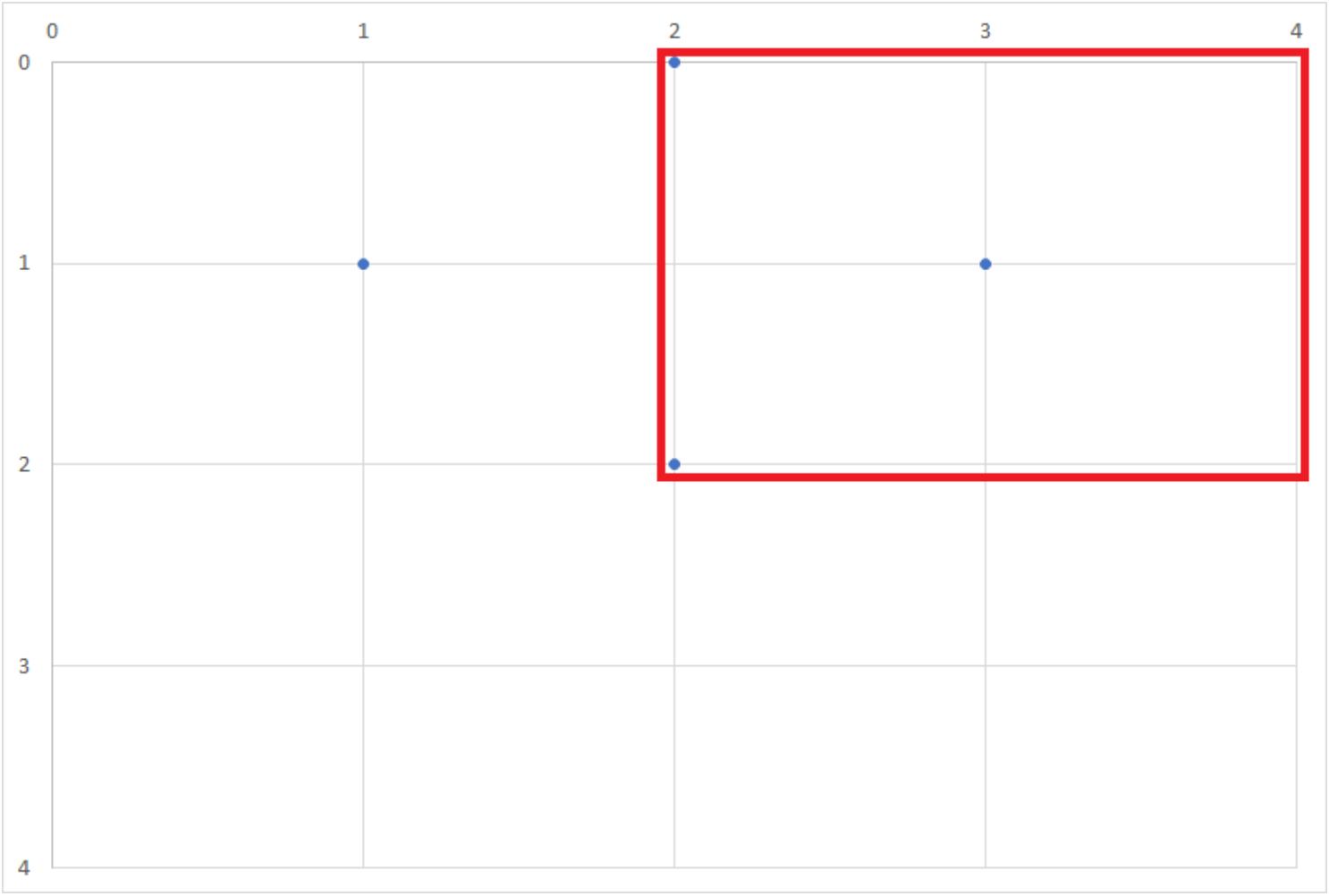


Mamy pole o rozmiarach 5×5 i płachtę o wymiarach 3×3

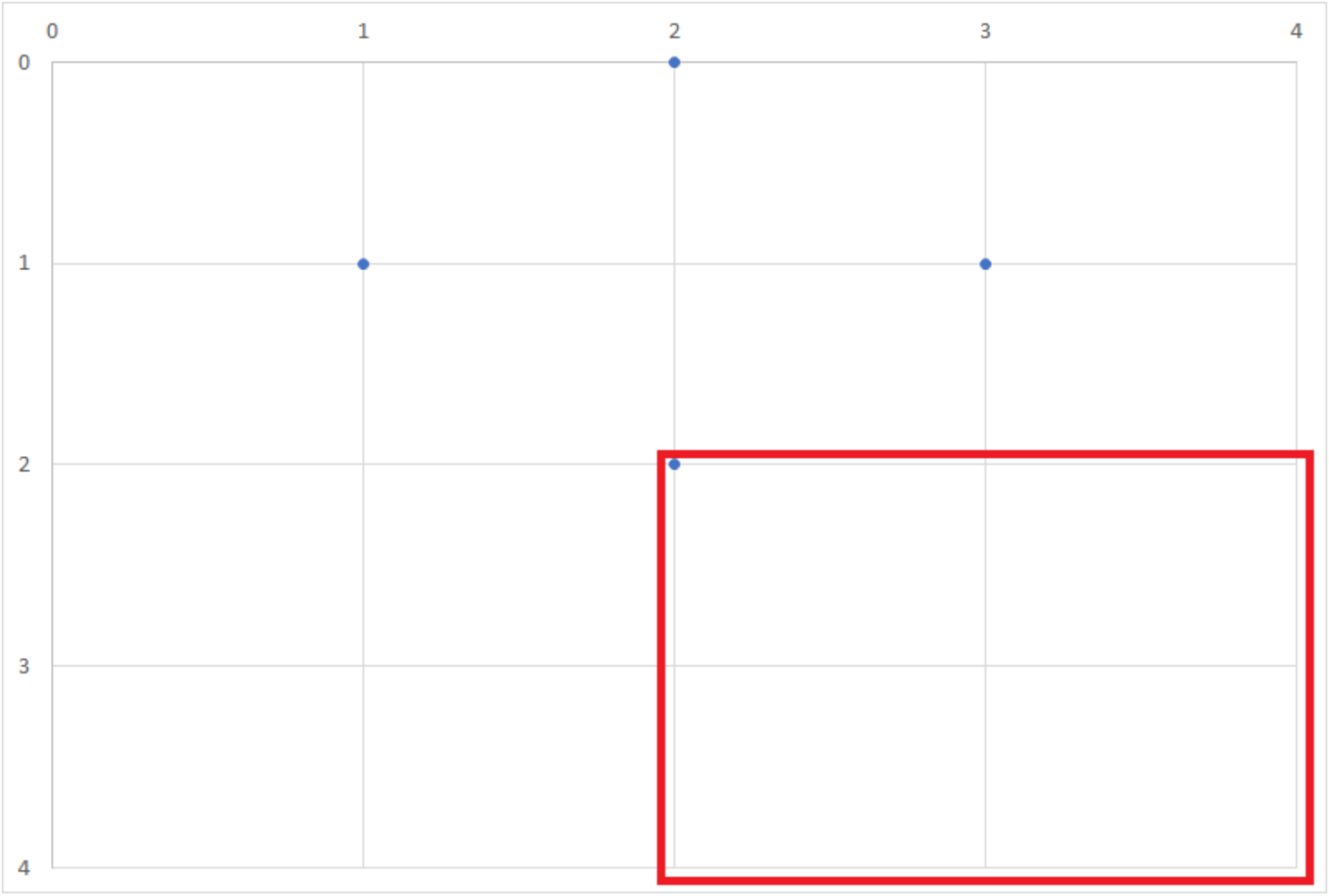


Zaczepiamy płachtę kolejno na każdych współrzędnych, zajmowanych przez krzak pomidorów.

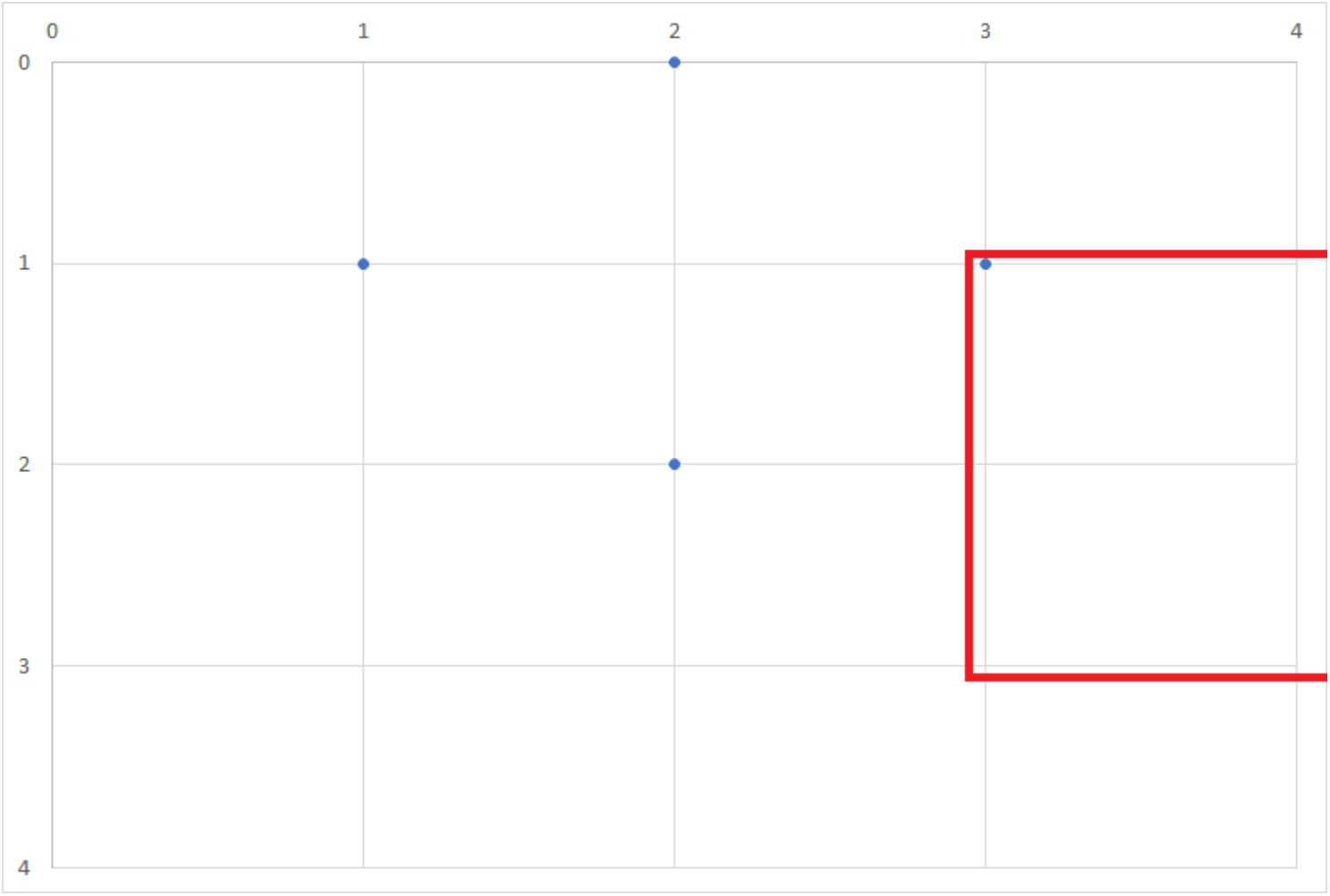
W tym przypadku liczba przykrytych pomidorów = 3



liczba przykrytych pomidorów = 3

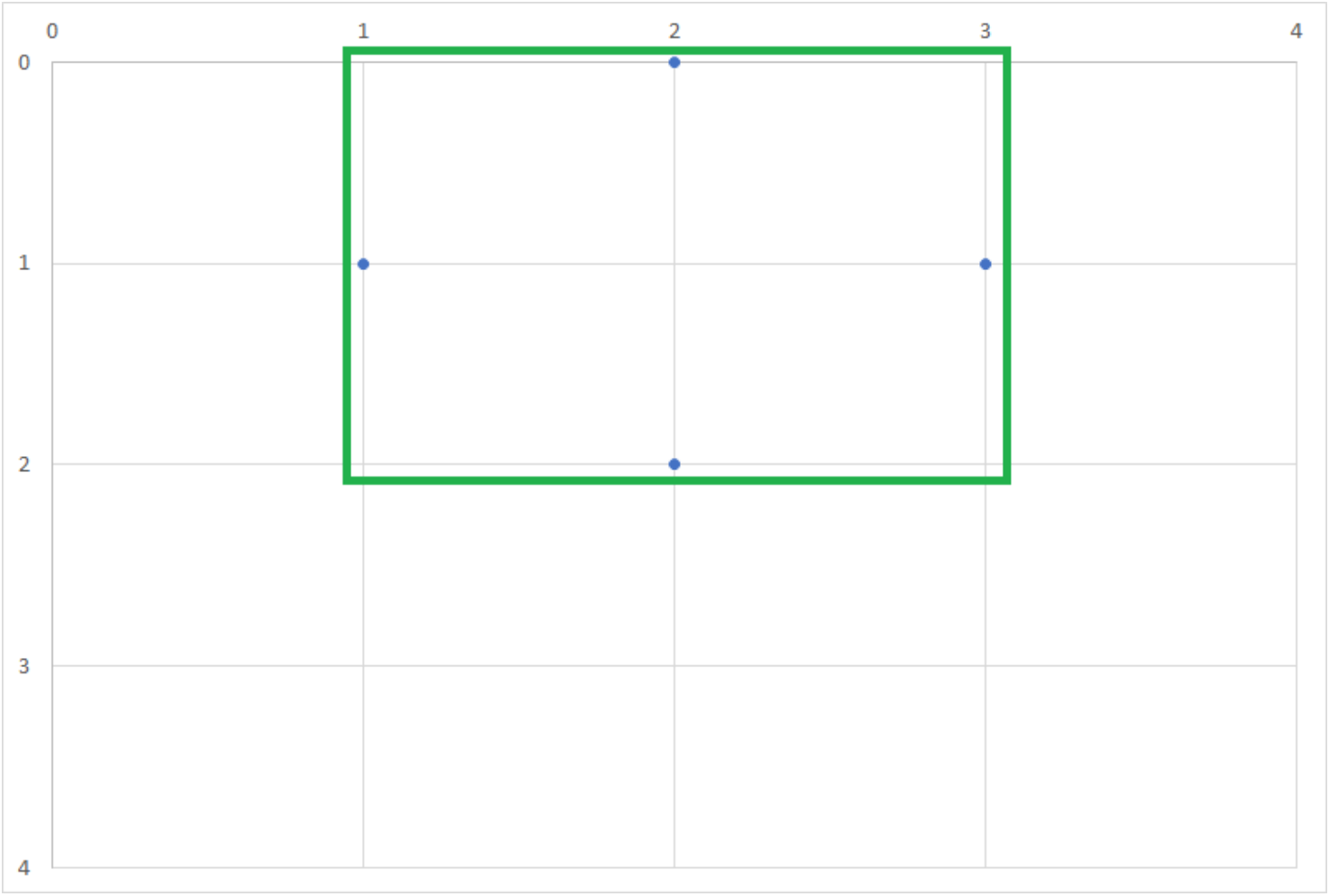


liczba przykrytych pomidorów = 1



liczba przykrytych pomidorów = 1

Dodatkowo trzeba rozpatrzeć oddzielnie sytuacje, w których płachta wychodzi poza pole.



liczba przykrytych pomidorów = 4 Najlepsze położenie płachty jest otrzymywane kiedy nie jest ona zaczepiona na żadnym z krzaków. Niestety biorąc pod uwagę tylko punkty zaczepenia będące punktami występowania krzaka, nie otrzymamy optymalnego wyniku.

Wykorzystany algorytm

Optymalny algorytm, wykorzystujący przedstawioną wyżej reprezentację położenia krzaków wygląda następująco:

1. Utwórz strukturę mapy, kojarzącą wartość całkowitą z wektorem liczb całkowitych (zmienna `potentially_covered`).
2. Uzupełnij mapę `potentially_covered` przy użyciu współrzędnych krzaków: pierwsza współrzędna (x-owa) krzaku jest kluczem a druga współrzędna (y-owa) jest wartością dodawaną do odpowiadającego kluczowi wektora.
3. Stwórz pomocnicze zmienne `pair<uint32_t,uint32_t> best_coords;` oraz`uint32_t maxsum = 0;`, przechowują one dla każdej iteracji algorytmu najlepsze dotychczasowe wyniki.
4. Dla każdego z krzaków (dla każdej pary współrzędnych `(a,b)`):

   4.1. Wyznacz odcinek zaczynający się w $(max(a-(w-1),0),b)$ a kończący się w $min(a+(w-1),n-1),b)$.

   4.2. Stwórz wektor całkowitych liczb nieujemnych: `vector<uint32_t> pointsdown;` a następnie dostosuj jego wielkość do rozmiaru problemu: `pointsdown.resize(field.max_coord());`.

   4.3. Dla każdego punktu na odcinku (dla każdej kolumny przechodzącej przez odcinek) sprawdź, używając mapy `potenitally_covered`, ile krzaków na polu jest odległych o maksymalnie `h` jednostek (ile krzaków o zadanej współrzędnej `x` ma współrzędną `y` w przedziale $(b, min(b+h, n-1))$).

   4.4. Wiedząc ile krzaków przypada na każdy punk odcinka, wpisz tę wartość pod odpowiednim indeksem w wektorze `pointsdown`.

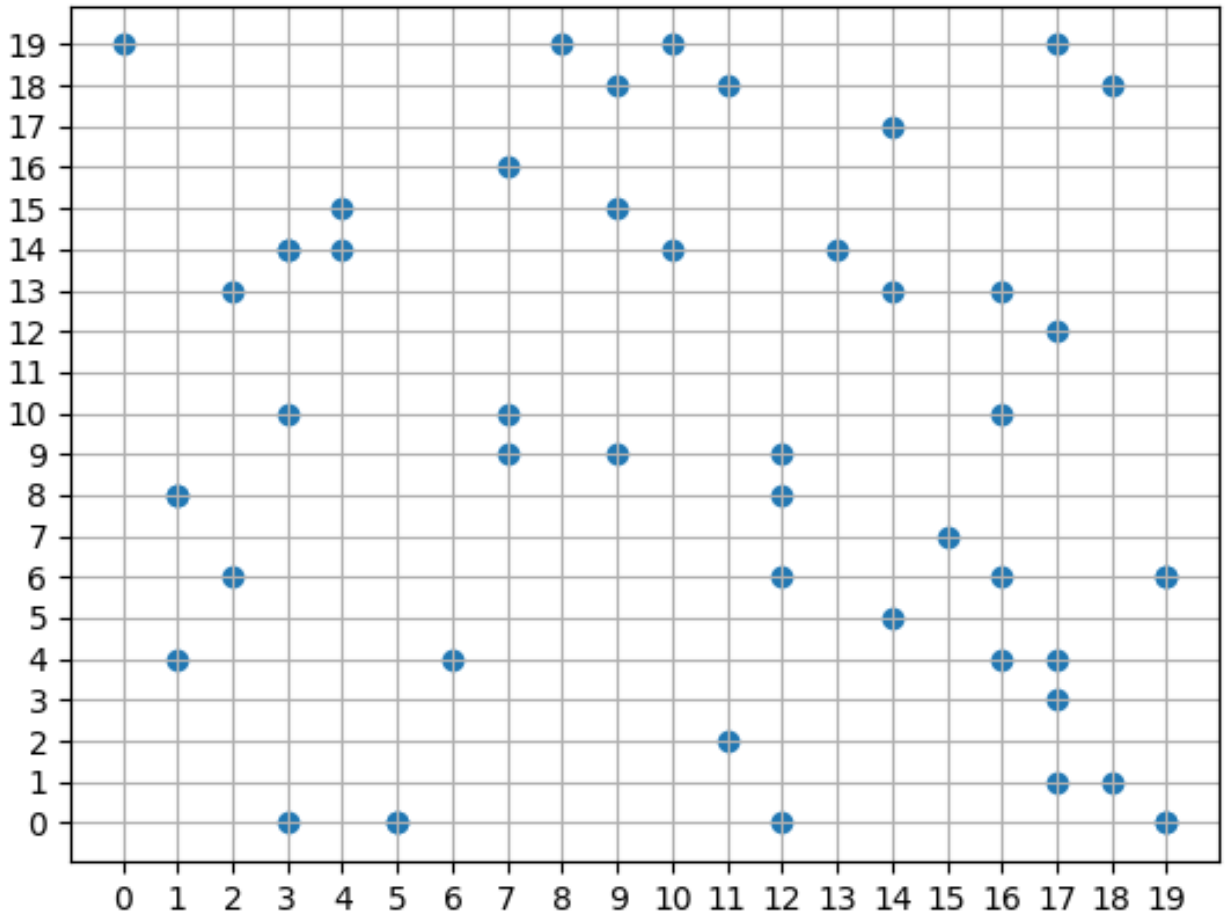
4.5. Znając wartości wektora `pointsdown`, sprawdź sumy, jakie daje ułożenie odcinka o długości `w` na wszystkich możliwych rozmieszczeniach wewnątrz odcinka z puntku 4.1.

4.5.1 Jeżeli badana suma jest większa od dotychczas największej to ustaw obecną sumę i ustawienie jako najlepsze.

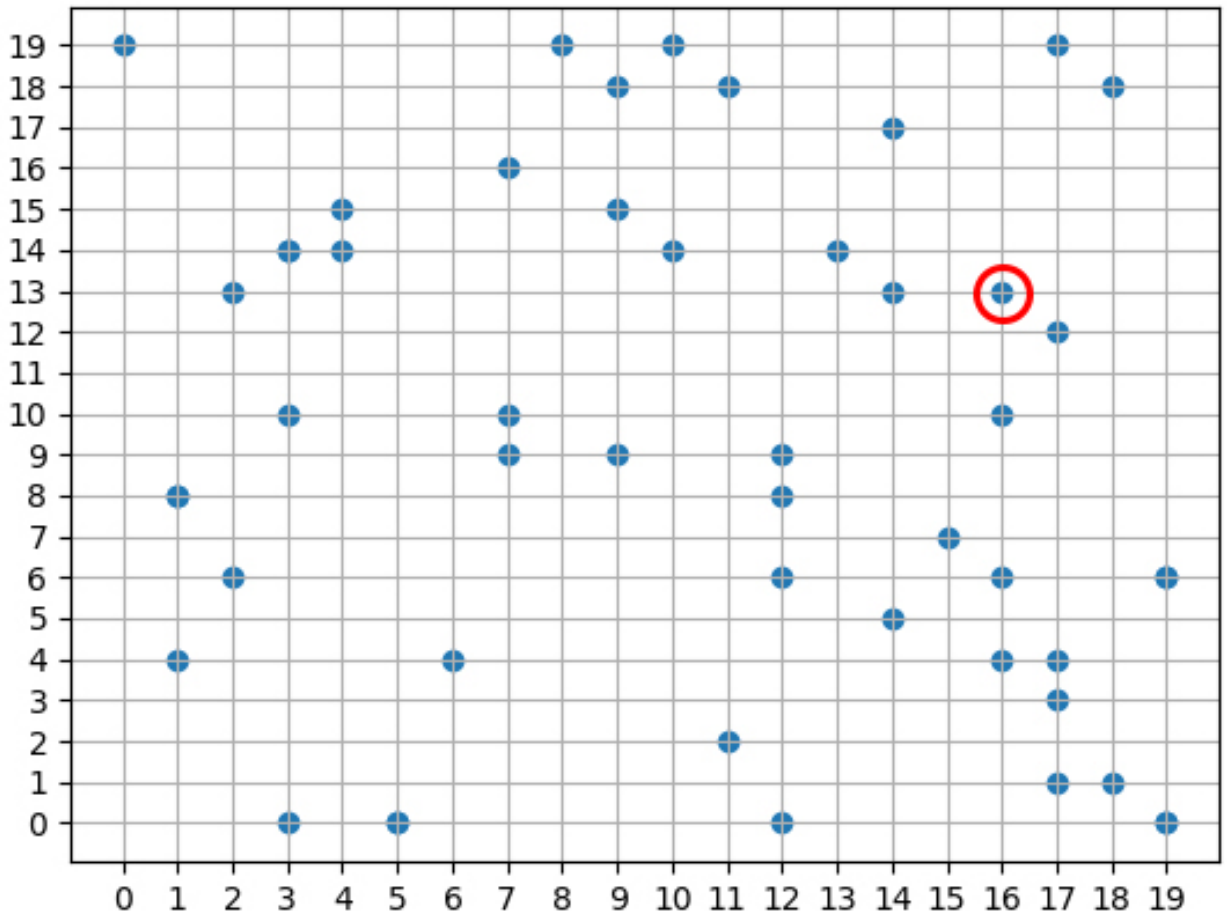
5. Jeżeli `h`!=`w` to powtórz uruchom algorytm jeszcze raz, tym razem z zamienionymi wartościami `h` oraz `w`.

Wizualizacja działania algorytmu:

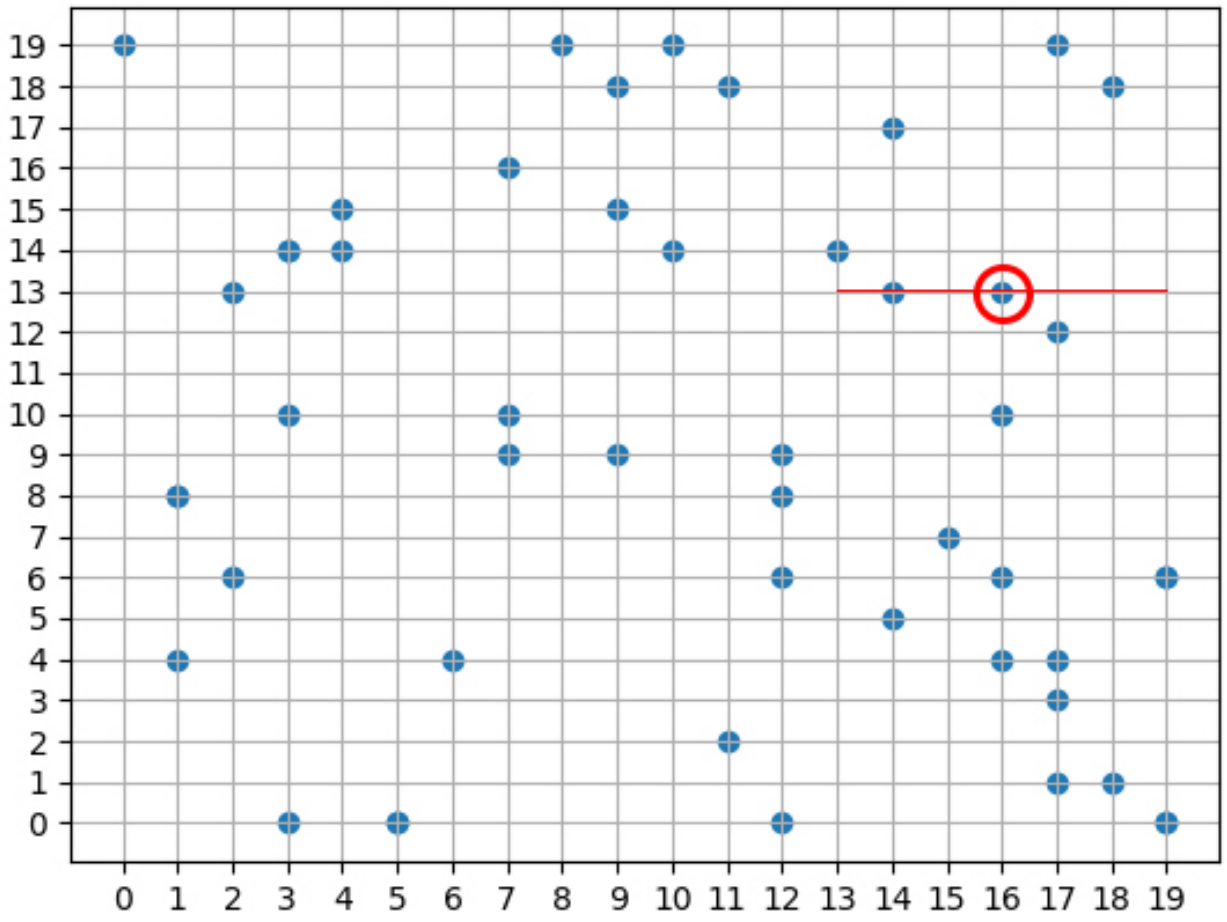
Obrazki przedstawione niżej różnią się od faktycznego stanu rzeczy: generowane przez generator pole ma odwrotnie zindeksowane wiersze (na górze wartość 0).



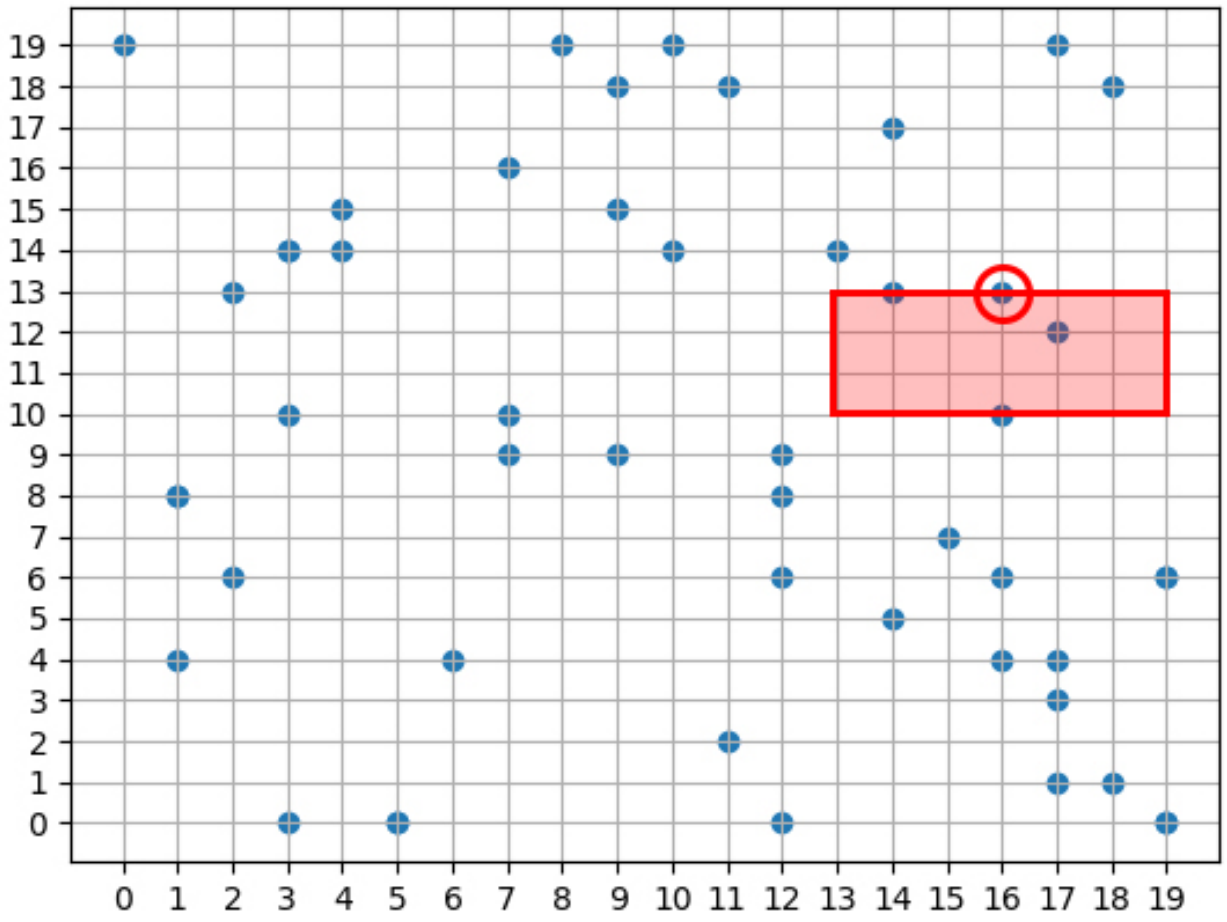
1. Wygenerowane pole 20×20, przyjęte rozmiary płachty to $4 \times 3 (h \times w)$



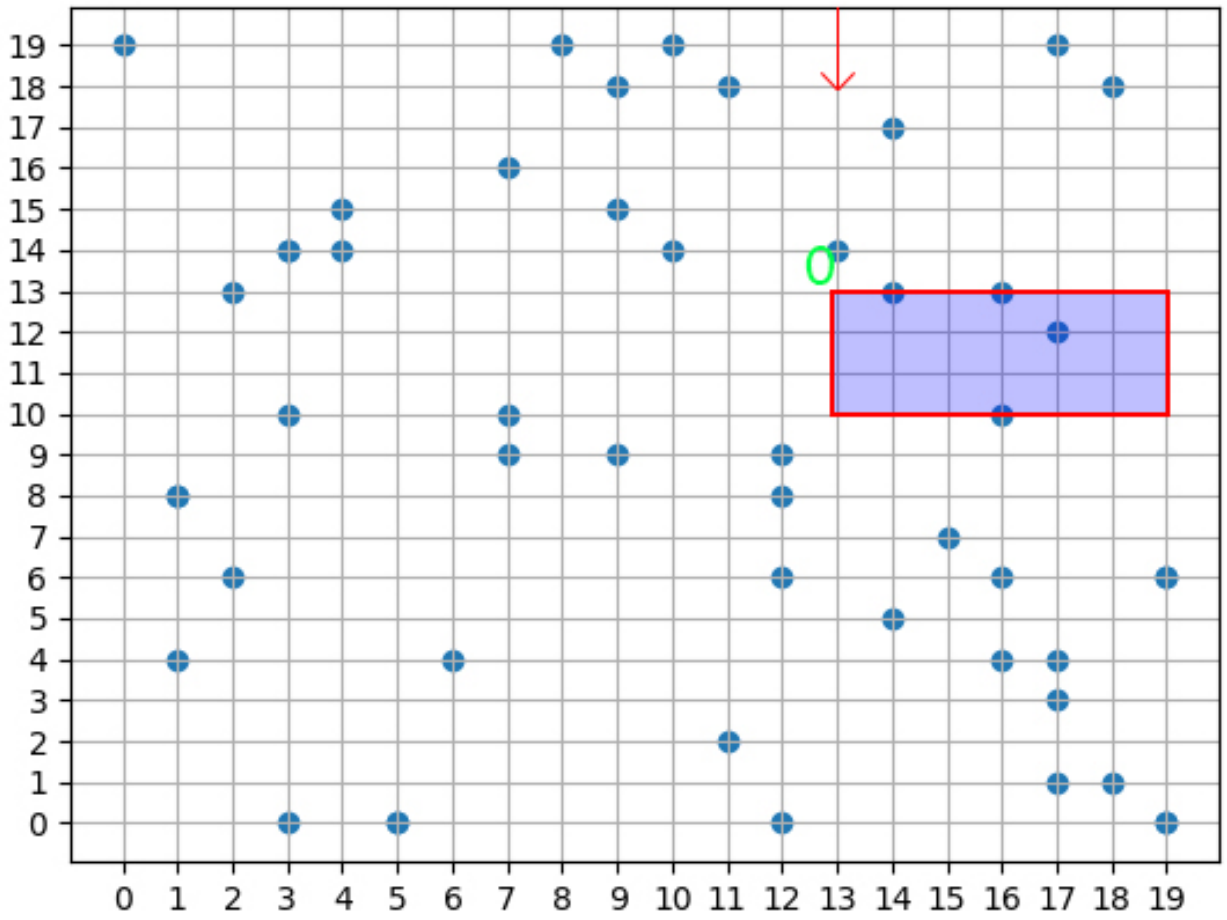
2. Przyjmujemy, że pierwszym badanym punktem jest krzak o współrzędnych $(16, 13)$



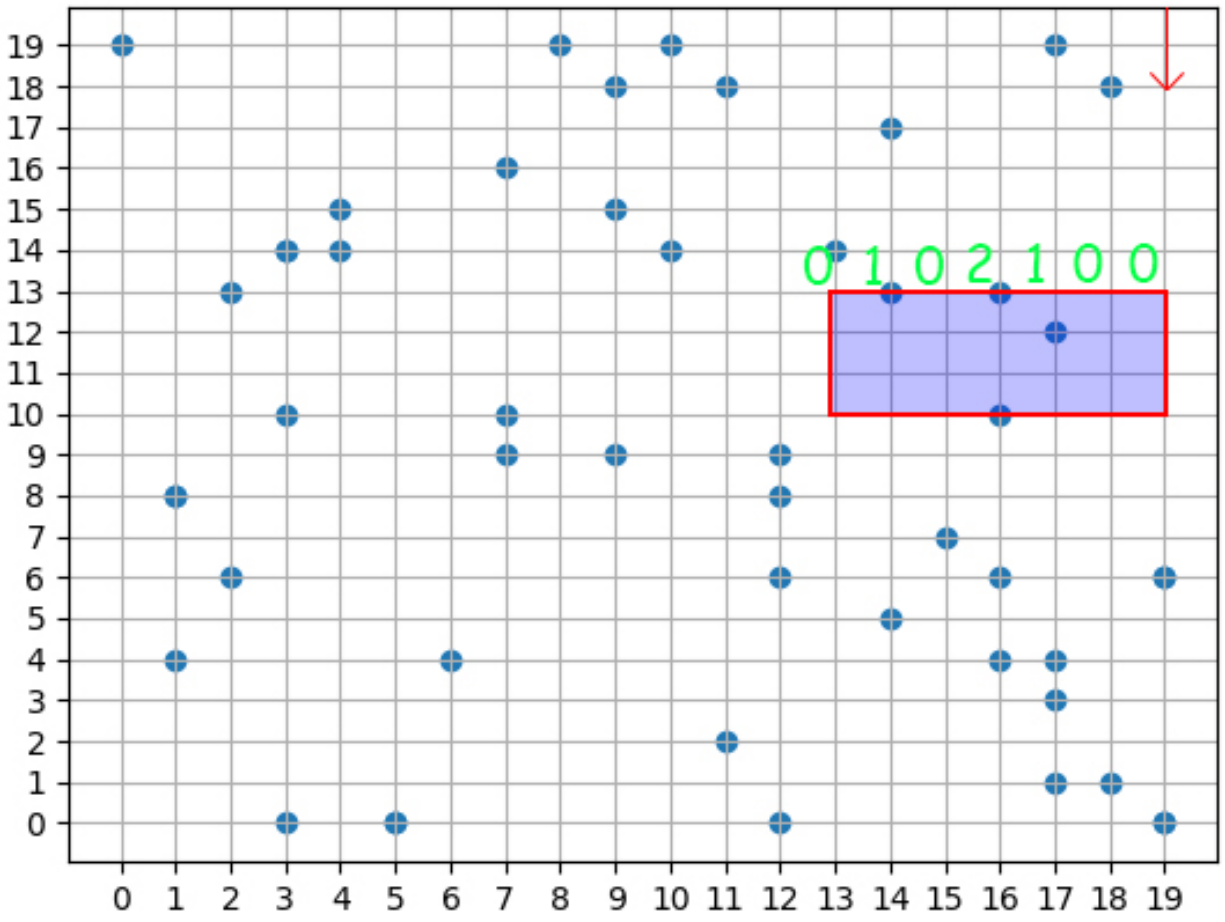
3. Rozciągany jest odcinek o długości $2 * (w - 1)$ o środku w badanym punkcie, jeżeli okazało by się, że nie da się utworzyć takiego odcinka ze względu na przekroczenie granicy pola, to zatrzymujemy się na owej granicy.



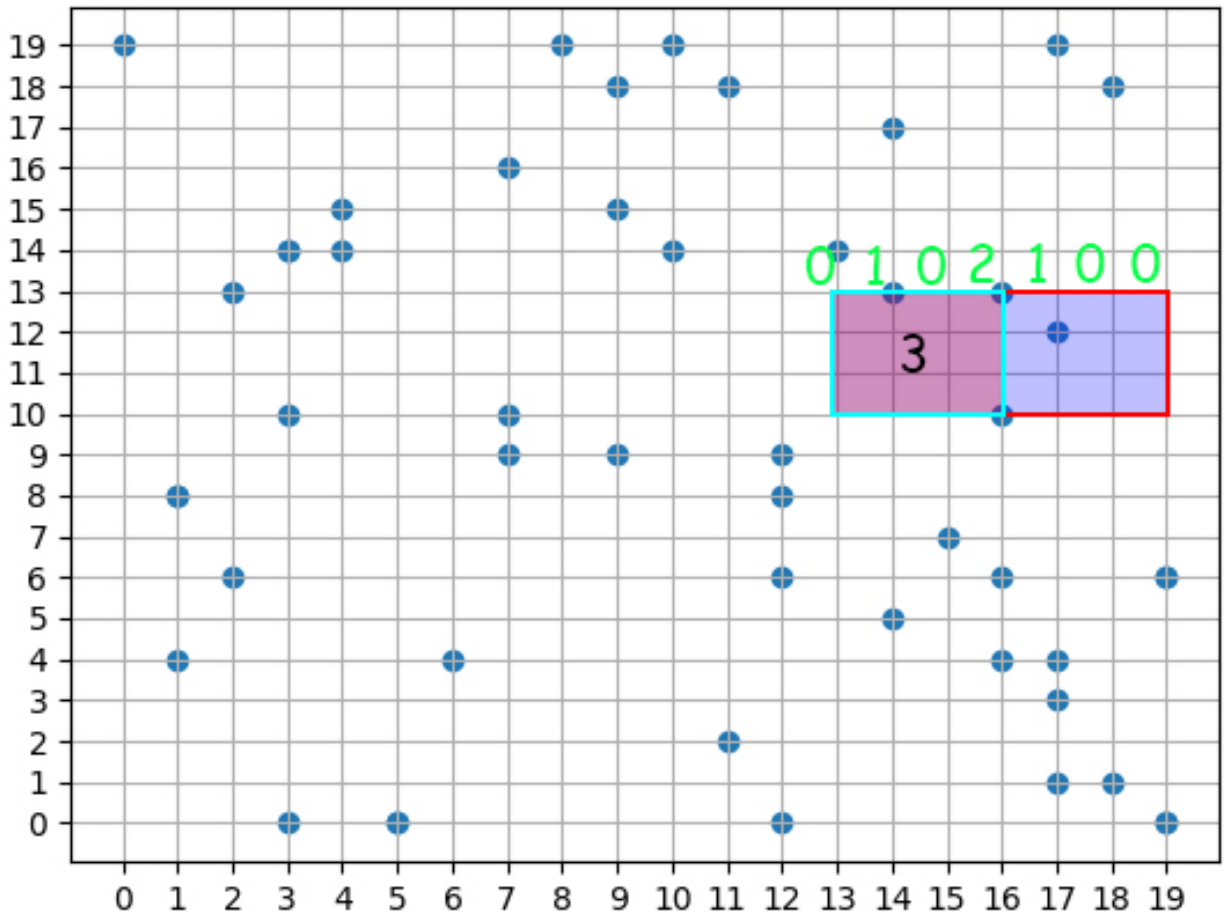
4. Przechodzimy o `h` jednostek w dół, tworząc prostokąt. To właśnie krzaki wewnątrz tego prostokąta będziemy brali pod uwagę.



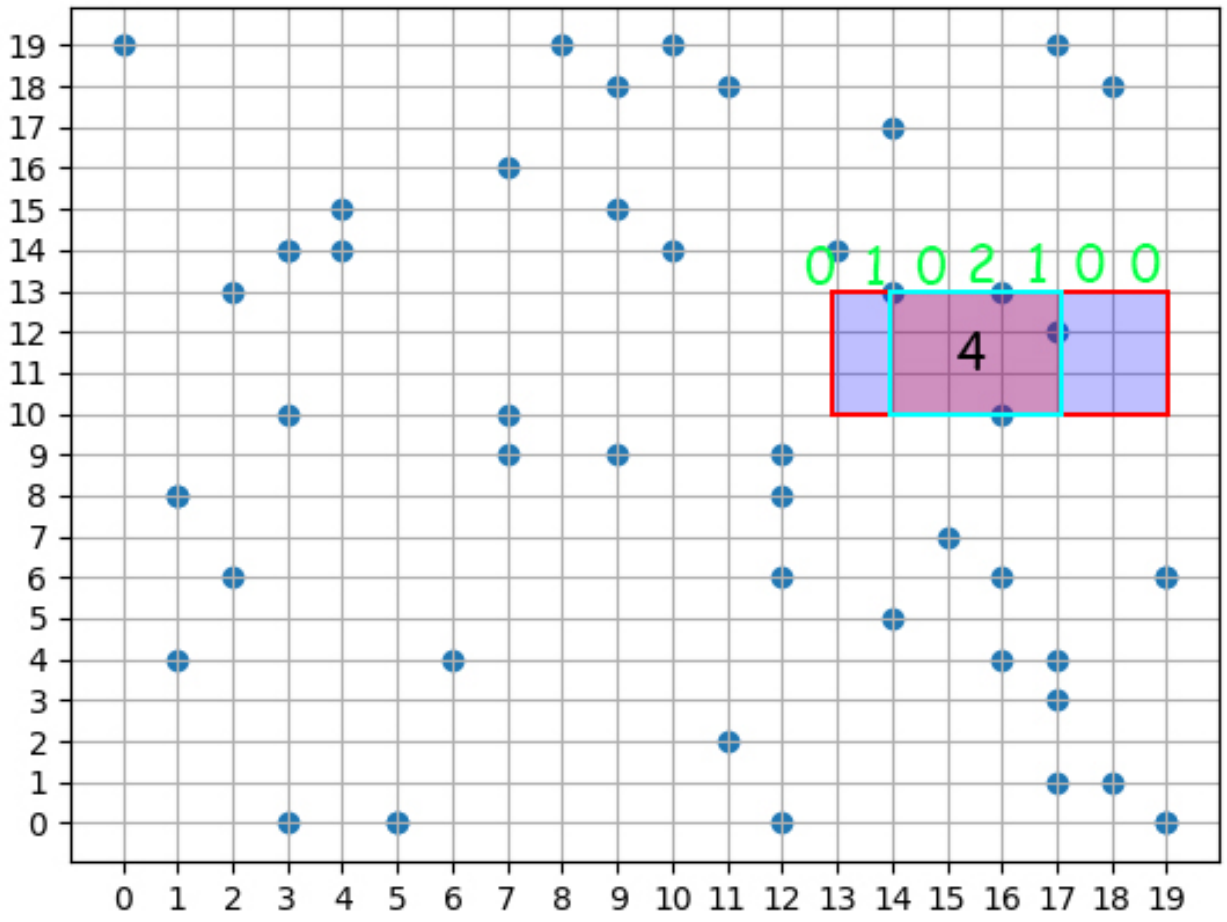
5. Przechodzimy po każdej kolumnie, w jakiej zawiera się nasz roboczy prostokąt i dla każdej z nich (kolumn) zapamiętujemy ile z krzaków o danej współrzędnej na polu zawierało się wewnątrz naszego prostokąta. Na obrazku powyżej badana jest kolumna `13`, znajduje się w niej jeden krzak, jednakże nie zawiera się on w prostokącie, dlatego nie bierzemy go pod uwagę. Właśnie w tym miejscu jest używana mapa `potentially_covered` - sprawdzamy wszystkie elementy wektora skojarzonego z daną współrzędną `x`.



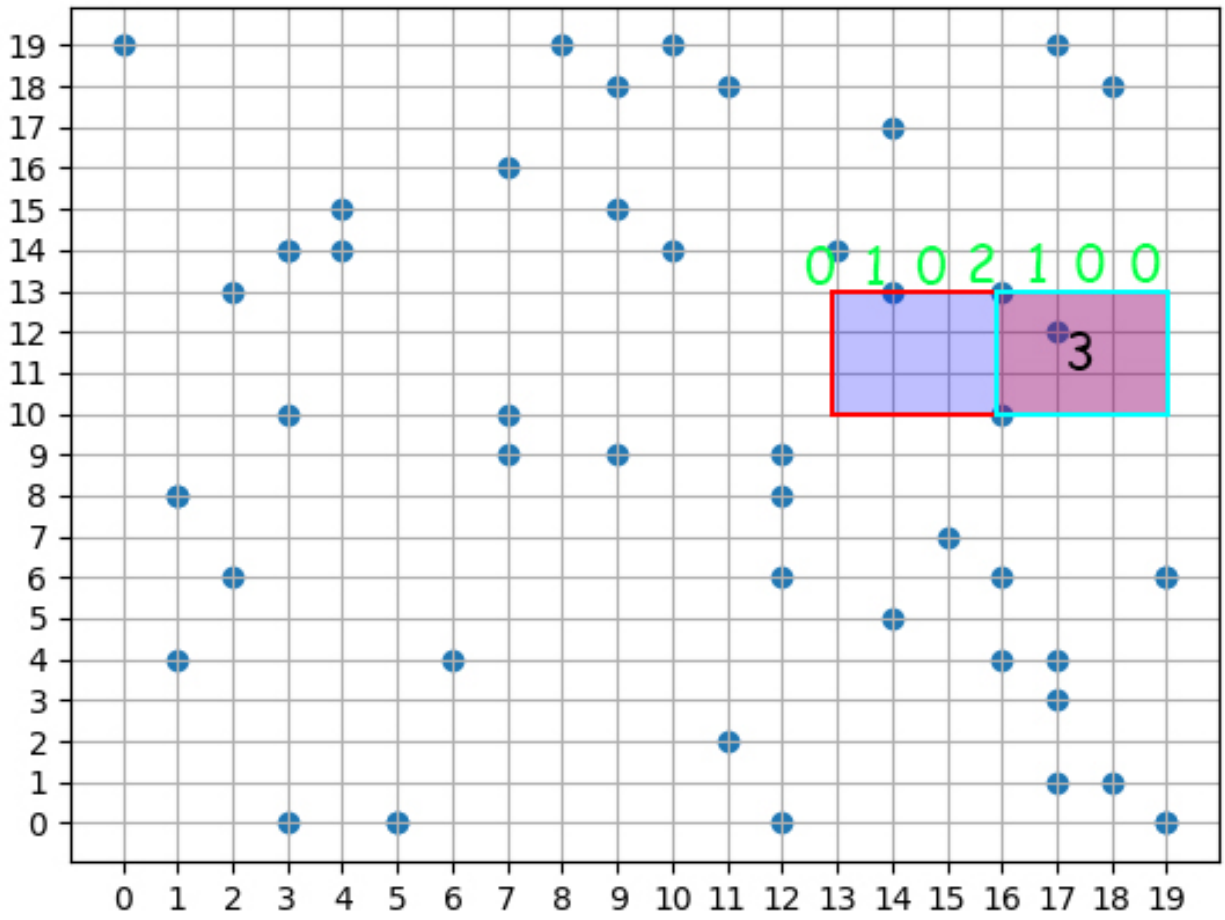
6. Sprawdzanie jest wykonywane dla każdej kolumny, przechodzącej przez wyznaczony prostokąt.



7. Po wyznaczeniu liczb krzaków w każdej z kolumn, przystępujemy do dopasowania płachty. Ustawiamy ją na pierwszej możliwej pozycji i sprawdzamy ile krzaków ona przykryje poprzez zsumowanie wcześniej wyznaczonych wyliczeń dla przykrytych kolumn.



8. Następuje przesunięcie płachty i ponowne policzenie ile krzaków może potencjalnie zostać przykrytych. Jeżeli liczba przykrytych krzaków dla danego ustawienia płachty jest większa od dotychczas najlepszej (największej) wartości to obecne ustawienie zostaje zapisane jako ustawienie najlepsze.



9. Płachta jest przesuwana aż do końca wyznaczonego przez nas prostokąta

Powyższe czynności są powtarzane dla każdego z krzaków ### Analiza złożoności algorytmu *złożoność* = 1 - przejście po każdym z krzaków - *złożoność* $*$ k - dla każdego z krzaków analizowanie jego otoczenia odległego o maksymalnie $(w-1)$ jednostek na prawo i na lewo - *złożoność* $*$ $max(h, w)$ (w przypadku gdy `h!=w` algorytm uruchamiany jest drugi raz z tymi wartościami zamienionymi miejscami - płachta jest obracana o 90°) - sprawdzenie dla każdego punktu z otoczenia krzaku ile innych krzaków znajduje się "w zasięgu" - *złożoność* $*$ k (dla `k` pomidorków na polu o wymiarach n^2 na jedną z `n` kolumn przypada średnio k/n krzaków pomidorów - asymptotycznie jest to zależność od `k`, musimy zawsze przejrzeć wszystkie punkty w badanej kolumnie)

Ostateczne szacunki: *złożoność* $= k^2 * max(h, w)$

Algorytm naiwny

W ramach projektu zaimplementowany został również algorytm naiwny, polegający na przejściu po każdej ze współrzędnych pola i sprawdzeniu jakie krzaki byłby przykryte, gdyby płachta zaczynała się na danej współrzędnej. Algorytm ten ma złożoność $n^2 * k$ co czyni go o wiele mniej wydajnym od zaproponowanego przez nas algorytmu. Jest on używany w celu porównania działań obu rozwiązań - do jego testowania został dodany tryb `-m4`.

Wykorzystane struktury

Pole pomidorów

Informacje o położeniu jednego z krzaków pomidorów są reprezentowane przez parę liczb nieujemnych:

```
typedef std::pair<uint32_t,uint32_t> tomato;
```

Całe pole jest reprezentowane przez wektor położeń wszystkich krzaków na nim umieszczonych:

```
std::vector<tomato> field;
```

Algorytm

Algorytm do analizy każdego z krzaków wykorzystuje mapę z biblioteki standardowej:

```
map<uint32_t,vector<uint32_t>> potentially_covered;
```

Wszędzie indziej wykorzystywane są typy podstawowe języka C++, z wyjątkami w postaci `std::vector`, który jest wykorzystywany jak tablica.

Tryby uruchomienia

Program przewiduje 4 tryby uruchomienia: - m1 - tryb pobierania danych ze standardowego wejścia - m2 - tryb generacji i rozwiazania problemu - m3 - tryb przeprowadzenia procesu testowania - m4 - tryb porównawczy z algorytmem naiwym

Więcej informacji o trybach uruchomienia znajduje się w pliku `readme.md`

Wynik uruchomienia w trybie testowym (`-m3`)

Wynik działania programu, uruchomionego w trybie `-m3` z następującymi ustawieniami: - początkowa liczbka krzaków pomidorów = 100 - początkowa wysokość płachty = 20 - początkowa szerokośc płachty = 30 - liczba rozmiarów wielkości problemu = 30 - liczba uruchomień algorytmu dla każdej z wielkości problemu = 10 - tryb zwiększania za każdym razem wszystkich składowych rozmiaru: liczby krzaków, szerokości oraz wysokości płachty

Przykład uruchomienia programu z powyższymi atrybutami:

```
AAL_pomidorki.exe -m3 -k100 -h20 -w30 -r30 -c10 -step20 -hwi > wyniki.txt
```

Wygenerowana tabela:

size	t(size)[ms]	q(size)
0.3	32.6304	1.73661
0.72	54.8475	1.21626
1.372	89.6314	1.04306
2.304	146.035	1.01199
3.564	213.893	0.95821
5.2	308.521	0.947291
7.26	437.852	0.962924
9.792	583.881	0.952039
12.844	769.142	0.95611
16.464	993.759	0.963712
20.7	1261.7	0.973166
25.6	1564.85	0.975966
31.212	1912.59	0.97837
37.584	2321.79	0.986328
44.764	2782.75	0.992537
52.8	3306.99	1
61.74	3860.3	0.998288
71.632	4523.98	1.00836
82.524	5292.38	1.02394
94.464	6056.75	1.02371
107.5	6870.57	1.02044
121.68	7809.12	1.02467
137.052	8826.63	1.02828
153.664	10036.4	1.04282
171.564	12247.4	1.13977

size	t(size)[ms]	q(size)
190.8	14001.2	1.17163
211.42	15316.9	1.15671
233.472	16729.9	1.14409
257.004	18556.4	1.1528
282.064	18741.7	1.06087

Wartość funkcji `q()` oscyluje w okolicach wartości 1.0 - sugeruje to, że założone przez nas oszacowanie złożoności algorytmu w postaci $k^2 max(h, w)$ jest prawidłowe. Czas działania wykonania programu z powyższymi parametrami, generującego pokazaną tabelkę, wynosił **37** minut na komputerze z procesorem o taktowaniu 3.8 GHz.