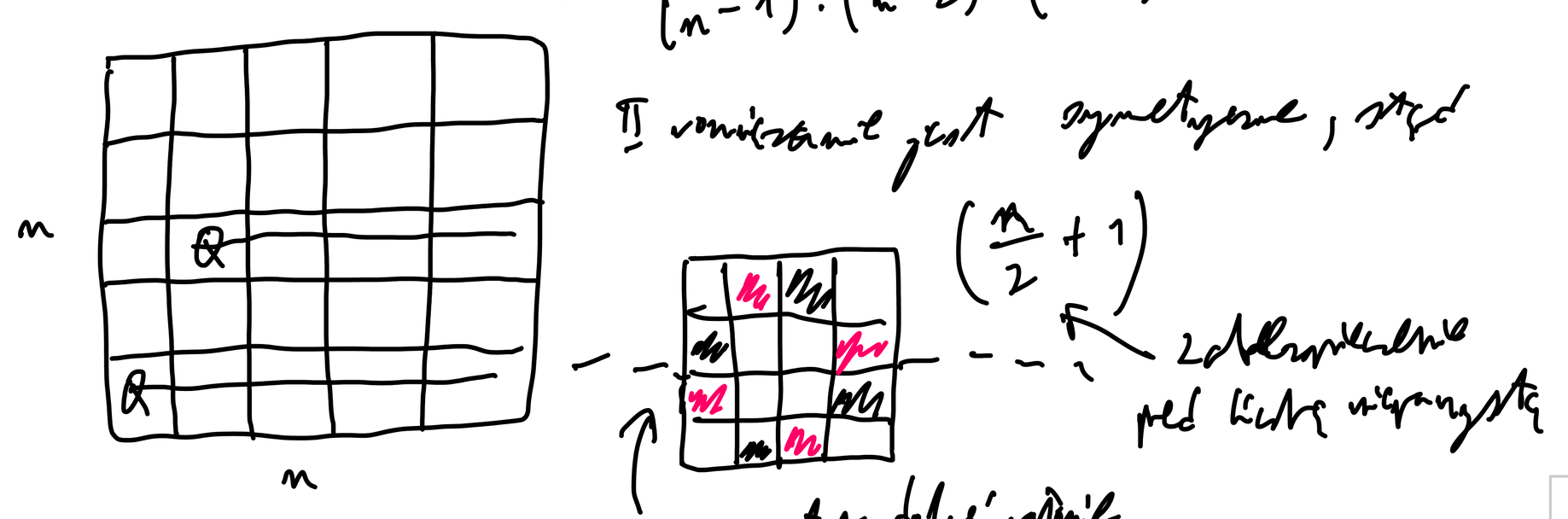
Lista 9

Thursday, 15 December 2022 05:19

2.1

tylko 1 ustawienie → $(n-1)\cdot(n-2)\cdot(n-3)\cdot\ldots\cdot 2\cdot n$ ← na ostatniej kolumnie będzie n kombinacji, jeśli tam ustawić



I ustawienie jest symetryczne, stąd $\left(\frac{n}{2}+1\right)$ ← zaokrąglenie pod liczbę nieparzystą

I hetman jest w dolnej połowie, mamy $\binom{n^2}{n}$ rozmieszczeń

(x, y) – sprawdzana pozycja

(a, b) – pozycja hetmana

ten sam wiersz + przekątna (kolumny nie badam, bo ją zmieniamy)

y == b ?

ten sam skos, gdy

$x+y=a+b \iff x-y=a-b$

a ≠ x

* lewa przekątna, współrzędne sumy jest te same
* prawa przekątna, współrzędne te same różnice

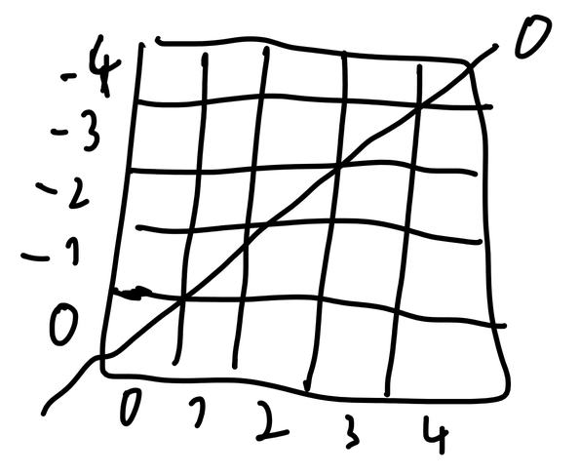
```
def isfree(x, y):
# isfree(x,y) = 1 wtw (x,y) nie jest atakowane
# przez hetmany z kolumn 0,1,2,...,x-1
    for i in range(x):
        if b[i]-i==y-x or b[i]+i==y+x or b[i]==y:
            return 0
    return 1
```

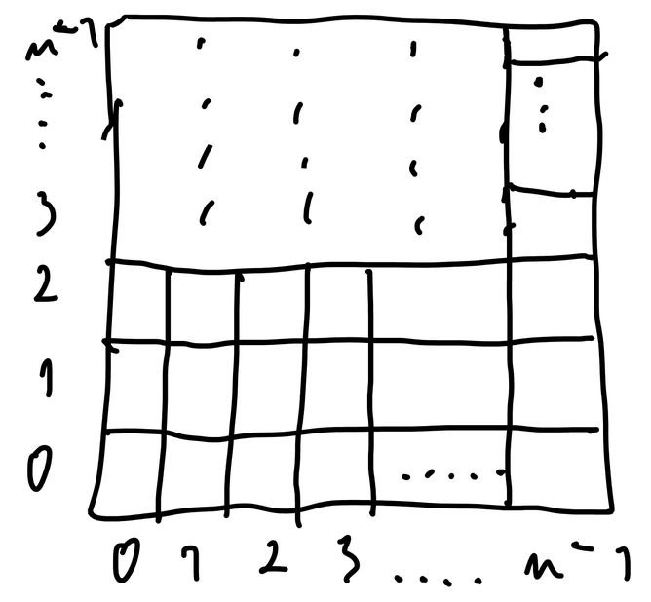
isfree(n-1, y) — KOLUMNA (ostatnia); WIERSZ SIĘ DECYDUJE

· for wykona się n razy

* ile jest skosów? $2(n-1)$

ile jest rzędów? = n

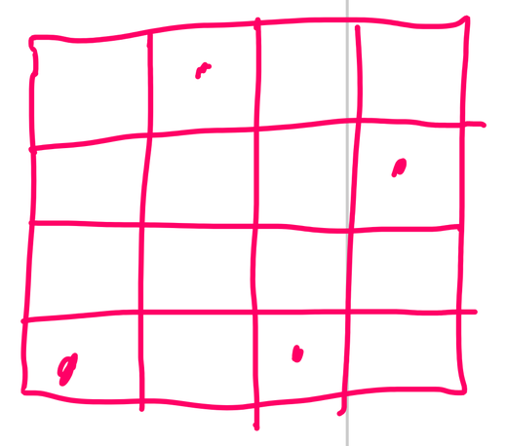




2.3

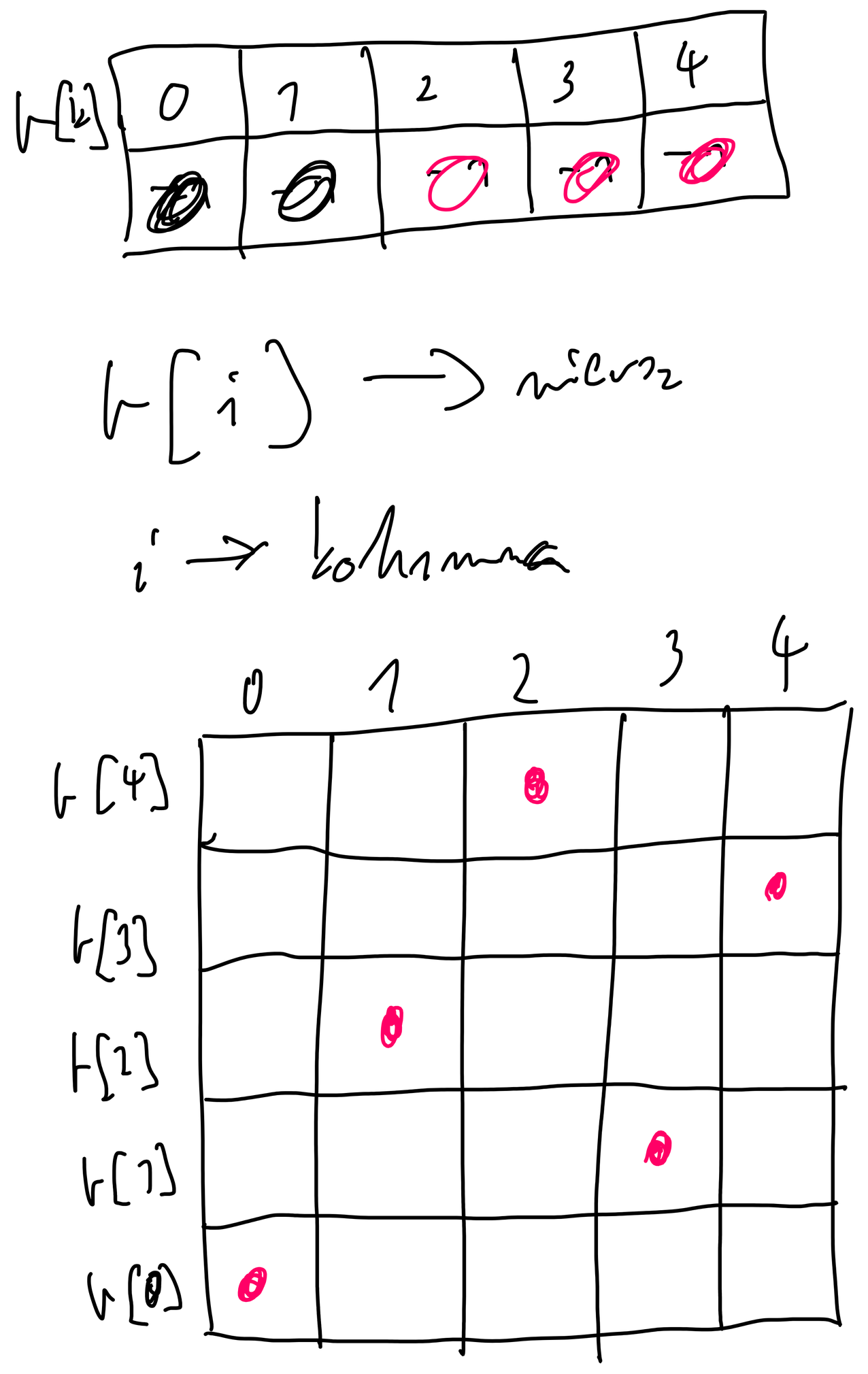
```
def poprawne(a, n):
    for i in range(n):
        for j in range(n):
            if a[i]-j-i == a[j]
            or a[i]+j-i == a[i]
            or a[i] == a[j]
                return 0
    return 1
```

nie jest ok



b[i] → wiersz

i → kolumna



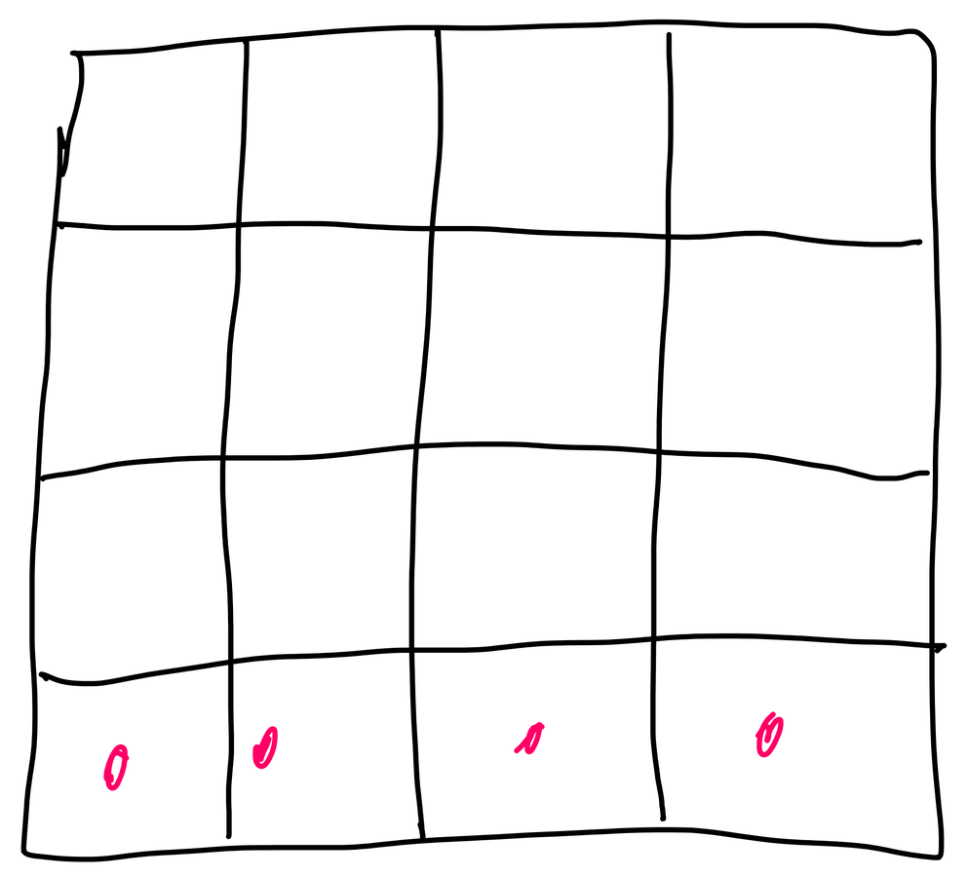
kiedy hetmany są na tej samej pozycji? (x, y); (a, b)

– kolumna: y = b

– wiersz: x = a

– skos ↗: x – y = a – b

– skos ↘: x + y = a + b



nie jest optymalne, bo nie sprawdza całego czasu kroków, idzie do końca z krokami

Instytut Informatyki UWr

Wstęp do informatyki

Lista 9

1. [1] Przyjmijmy, że algorytm przeszukiwania z nawrotami z wykładu **sprawdza** jakieś ustawienie n hetmanów, jeśli wywoływana jest funkcja isfree(n – 1, y)[1] dla $0 \le y < n$, czyli **sprawdzamy** możliwość „dostawienia" n-tego hetmana do ustawionych już poprawnie $n-1$ hetmanów. Uzasadnij, że dla $n \ge 4$ algorytm sprawdza nie więcej niż $\left(\frac{n}{2}+1\right)\cdot n!$ różnych ustawień n hetmanów na szachownicy.
Istotna uwaga: W swoim rozwiązaniu przyjmij, że dla każdego $n \ge 4$ istnieje rozwiązanie problemu hetmanów, tzn. można ustawić n hetmanów na planszy $n\times n$ tak, aby żadne dwa nie atakowały się nawzajem.

2. [1] Napisz funkcję, która dla zadanej wartości n wyznaczy **liczbę** możliwych ustawień n hetmanów na szachownicy o rozmiarze $n\times n$, tak aby hetmany nie atakowały się nawzajem. **Wskazówka**: możesz wykorzystać algorytm z wykładu, znajdujący dowolne ustawienie hetmanów (wystarczy go lekko zmodyfikować):

```
int queens()
{  int k;
   b[0]=0;
   k=1;
   while (k<n && k>=0)
   {  do
        { b[k]++; }
      while (b[k]<n && !isfree(k,b[k]));
      if (b[k]<n) k++;
      else {b[k]=-1; k--;}
   }
   return k;
}
```

(notes: b[0] – wartość; k=1 – kolumna; b[k]++ – zwiększam wiersz; if (b[k]<n) k++ – do następnej kolumny; else – kolumna do tyłu; if k==n: ilość++; k--; [po znalezieniu odpowiedniego miejsca] sprawdzamy 2 więc innego rozwiązania)

3. [1] Rozważmy następujący algorytm znajdujący ustawienie n hetmanów na szachownicy $n\times n$

```
int hetmany(int n, int k, int a[])
{
  if (k==n) return poprawne(a,n);
  for(int i=0; i<n; i++)  {
    a[k]=i;
    if (hetmany(n,k+1,a)) return 1;
  }
  return 0;
}
```

(notes: lista, hetmany; if)

[1] Zwróć uwagę, że wartość $n-1$ dla pierwszego argumentu isfree oznacza, że zliczamy tylko wywołania isfree dla pozycji w ostatniej kolumnie planszy!

```
def hetmany(n, k, a):
  if k==n: return poprawne(a,n)
  for i in range(n):
    a[k]=i
    if (hetmany(n,k+1,a)): return 1
  return 0
```

gdzie a[i] wskazuje wiersz hetmana umieszczonego w kolumnie i, a funkcja poprawne zwraca wartość 1, jeśli ustawienie hetmanów opisane w tablicy a[0..n-1] jest poprawne (tzn. żadne dwa hetmany nie atakują się nawzajem) oraz zwraca 0 w przeciwnym przypadku. Twoje zadanie:

$a[n] = \{-1, -1, -1, \ldots -1\}$

a. Podaj z jakimi parametrami należy wywołać funkcję hetmany, aby rozwiązać problem hetmanów dla szachownicy $n \times n$? hetmany(n, 0, a)
b. Napisz treść funkcji poprawne.
c. Porównaj działanie funkcji hetmany z tego zadania z działaniem metody opartej o przeszukiwanie z nawrotami. Która metoda sprawdza więcej ustawień hetmanów? Dlaczego?

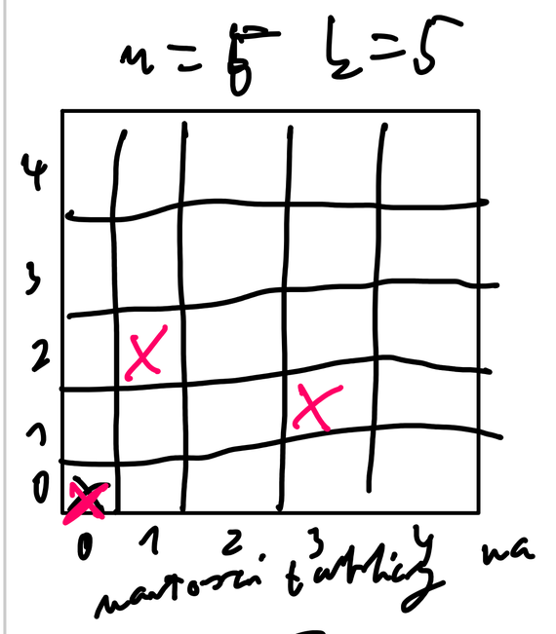
4. [1] Zapoznaj się z problemem skoczka szachowego. Następnie opracuj i przedstaw **ideę** rozwiązania tego problemu metodą przeszukiwania z nawrotami. Precyzyjnie omów struktury danych, których będziesz używać w swoim rozwiązaniu i sposób wykorzystania tych struktur.

5. [2] Kwadratem magicznym rozmiaru $n\times n$ nazywamy planszę $n\times n$ złożoną z liczb całkowitych $1,\ldots,n^2$, gdzie każda z tych liczb występuje dokładnie jeden raz oraz suma w każdym wierszu, kolumnie i na przekątnych są równe. Napisz funkcję, która dla zadanego n znajduje kwadrat magiczny rozmiaru $n\times n$ (o ile istnieje).
Uwaga. Zastosuj przeszukiwanie z nawrotami.

6. [1] Ścieżka trawersująca kwadratową planszę rozmiaru $n \times n$ powinna spełniać następujące warunki:
   a) Ścieżka zaczyna się w polu [0, 0], kończy w polu [n – 1, n – 1].
   b) Dla $i < n - 1$: z pola [i, j] możemy przejść do pola [i+1, j].
   c) Dla $j < n - 1$: z pola [i, j] możemy przejść do pola [i, j+1].
W dwuwymiarowej tablicy a[n][n] zapisane są koszty odwiedzania poszczególnych pól na planszy – koszt odwiedzenia pola (i, j) jest równy a[i][j]. *Koszt ścieżki* jest równy sumie kosztów pól, przez które ta ścieżka przechodzi.
Twoje zadanie
Napisz algorytm realizujący poniższą specyfikację. Oszacuj asymptotycznie złożoność czasową swojego rozwiązania, odpowiedź uzasadnij!
Specyfikacja
Wejście:
n – liczba naturalna dodatnia
$a[n][n]$ – tablica liczb całkowitych rozmiaru $n \times n$
Wyjście:
Najmniejszy koszt ścieżki *trawersującej* planszę rozmiaru $n \times n$ z kosztami odwiedzania pól opisanymi w tablicy a.

Please wait while OneNote loads this printout...

n = 5, k = 5



wartości tablicy na konkretnym indeksie to numer wiersza

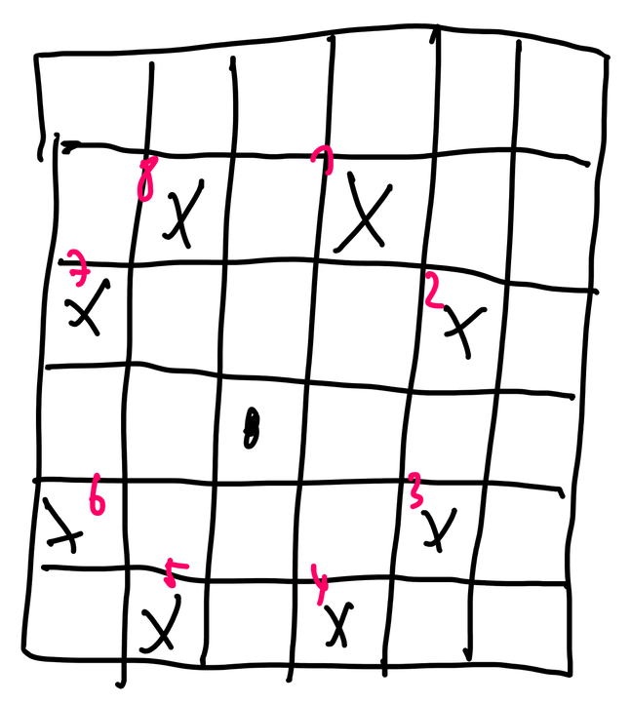
b = [-1, -1, -1, -1, -1] ← indeksy to numery kolumn (0 1 2 3 4)

b[n] pole będące wartością w kolumnie n

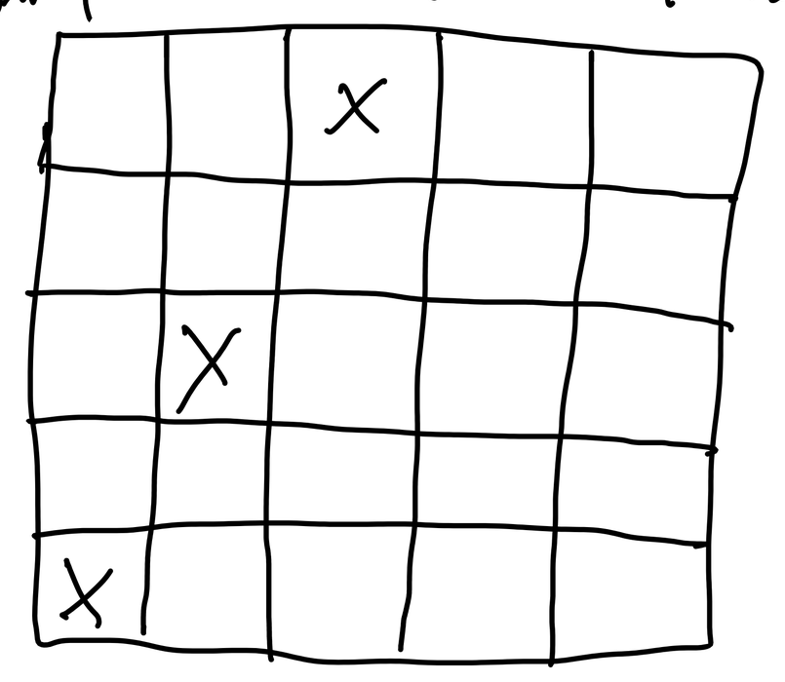
2.4

* ruchy:
1 – i–2; j+1
2 – i–1; j+2
3 – i+1; j+2
4 – i+2; j+1
5 – i+2; j–1
6 – i+1; j–2
7 – i–1; j–2
8 – i–2; j–1

· zaczynam od [0, 0]
· sprawdzam na które pozycje mogę wejść
· wybieram pierwszą możliwą, idę tam
· zapisuję pole gdzie byłem do listy "MARTWYCH PÓL"
· zapisuję obecny ruch do tablicy ruchów (wartości 1–8)
· powtarzam procedurę dopóki nie mogę dalej skakać, a liczba "MARTWYCH PÓL" nie jest równa n^2



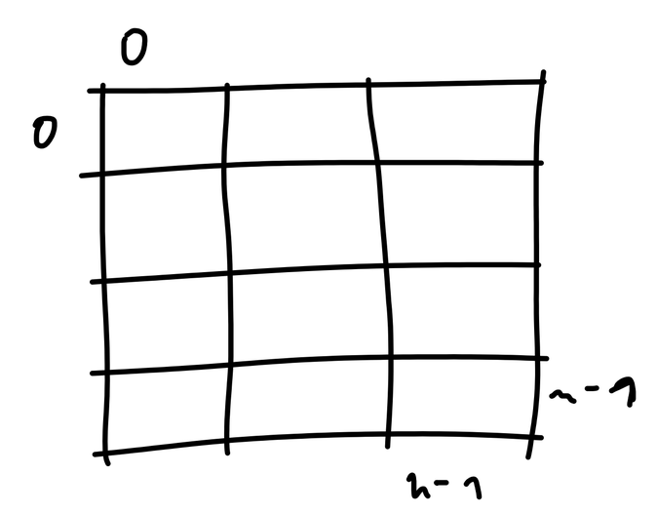
* gdy taka sytuacja się wydarzy cofam się po ruchach i wymazuję z listy "MARTWYCH PÓL" odpowiednie pola, robię to tak długo aż ruch na który się cofnąłem umożliwi mi pójście na inną niż poprzednio pozycję, wtedy skaczę i dalej powtarzam



* STRUKTURY DANYCH
· tablica pozycji[n][n] ("MARTWE POLA") – to można zastąpić przez stos wywołań rekurencyjnych
· tablica historii ruchów [n^2]
· tablica możliwych ruchów [8]

PROBLEM SKOCZKA SZACHOWEGO
– jak odwiedzić wszystkie pola na szachownicy, będąc na każdym tylko raz
– rozwiązanie istnieje dopiero dla planszy o liczbie pól większej niż 4, dla 3,4, do środkowego kwadratu nie da się wskoczyć

2.6



Każdy krok w tablicy to wybór min + if:
· 4 ify
· 3 porównania by znaleźć wartość minimum
· w najgorszym przypadku program wykona się i·j razy, bo ścieżka będzie wyglądać tak

$j = n-1$, $i = n-1$

$i(3+3) + j\cdot(3+3) =$
$= 6i + 6j = O(i\cdot j) =$
$= O(n^2)$