# 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;
}
```

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;
}
```

[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. Podaj z jakimi parametrami należy wywołać funkcję `hetmany`, aby rozwiązać problem hetmanów dla szachownicy $n \times n$?
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 $a$.

**Przykład**

Rozważmy $n = 3$ oraz następującą tablicę $a$:

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | **10** | **9** | **31** |
| 1 | **21** | **7** | **8** |
| 2 | **13** | **14** | **10** |

Koszt ścieżki przechodzącej przez pola [0, 0], [1, 0], [2, 0], [2, 1] i [2, 2] jest równy 10 + 21 + 13 + 14 + 10 = 68.

Najmniejszy koszt ścieżki przechodzącej przez planszę jest równy 10 + 9 + 7 + 8 + 10 = 44.

**Uwaga.** Akceptowane są m.in. rozwiązania wykorzystujące przeszukiwanie z nawrotami lub programowanie dynamiczne. Zastanów się, czy potrafisz zaimplementować obie metody. Która z nich zagwarantuje lepszą złożoność czasową?

**Zadania dodatkowe, nieobowiązkowe (nie wliczają się do puli punktów do zdobycia na ćwiczeniach, punktacja została podana tylko jako informacja o trudności zadań wg wykładowcy)**

7. [1] Napisz funkcję hetmany, która znajduje ustawienie $n$ hetmanów na szachownicy rozmiaru $n \times n$, tak aby hetmany nie atakowały się nawzajem. Twoja funkcja powinna stosować przeszukiwanie z nawrotami i ustawienie hetmanów reprezentować w tablicy **dwu-wymiarowej** (jeden element tablicy odpowiada jednemu polu szachownicy).
   **Wskazówka**: wystarczy odpowiednio zmodyfikować rozwiązanie z wykładu.
8. [1] Przedstaw pełną implementację rozwiązania problemu skoczka szachowego.
9. [2] Napisz funkcję minhetmany, która znajduje minimalną liczbę hetmanów atakujących wszystkie pola szachownicy $n \times n$. (Dokładniej, Twoja funkcja powinna zwrócić taką liczbę $k$, że przy pewnym ustawieniu $k$ hetmanów atakowane są wszystkie pola szachownicy $n \times n$ i nie jest możliwe ustawienie $k - 1$ hetmanów tak, aby wszystkie pola szachownicy $n \times n$ były atakowane.)