Vaja 4: Dinamično programiranje

Domen Kavran

21. 11. 2023

Dinamično programiranje

- Strategija reševanja optimizacijskih problemov
- Shranjevanje vmesnih rezultatov:
  - Memoizacija (top-down): shranjujemo rešitve podproblemov
  - Tabeliranje (bottom-up): rešitve nadproblemov tvorjene iz rešitev podproblemov

Zgled: Fibonaccijeva števila - 1

- Zaporedje: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,…

- Naivni pristop

```
int fibonacci(int n) {
        if (n <= 1)
                return n;
        return fibonacci(n-1) + fibonacci(n-2)
}
```

Zgled: Fibonaccijeva števila - 2

- Dinamično programiranje z memoizacijo

```
int f[MAX];

int fib(n, f) {
        if (f[n] < 0)
                f[n] = fib(n-1) + fib(n-2);
        return f[n];
}

int fibonacci(int n) {
        for (int i = 0; i <= n; ++i)
                f[i] = -1;
        f[0] = 0;
        f[1] = 1;
        fib(n,f);
        return f[n];
}
```

Zgled: Fibonaccijeva števila - 3

- Dinamično programiranje s tabeliranjem

```
int f[MAX];

int fibonacci(int n) {
        f[0] = 0;
        f[1] = 1;
        for (int i = 2; i <= n; ++i)
                f[i] = f[i-1] + f[i-2];
        return f[n];
}
```

Vaja 4: Pirati

Opis: Pirati raziskujejo otoke v oceanu. Kapitan Jack Sparrow ima zemljevid območja, razdeljenega na mrežo velikosti $N \times M$. Na območju se nahaja K otokov – vsak izmed njih se nahaja znotraj svoje celice v mreži. Kapitan je ocenil število ur t_i, ki jih bodo potrebovali na vsakem i-tem otoku za raziskovanje, če bodo na njem pristali.

Jack Sparrow in njegovi pirati pričnejo s plovbo v zgornjem, levem kotu mreže - v celici $(1,1)$. Če so pirati trenutno na lokaciji (x, y), potem lahko v 1 uri priplujejo na lokacijo $(x, y+1)$ ali $(x, y-1)$. Če se pirati nahajajo v prvem $(y = 1)$ ali zadnjem stolpcu območja $(y = M)$, potem lahko priplujejo na lokacijo $(x+1, y)$ v 1 uri.

Pomagajte kapitanu Jacku Sparrowu ugotoviti minimalno število ur, da obišče in razišče k otokov.

Vhodni podatki:

1. vrstica: celo število N, M in K $(4 \leq N, M \leq 10^7, 1 \leq K \leq 10^4)$

K vrstic: vsaka vrstica vsebuje cela števila x, y in t, ki označujejo lokacijo otoka (x, y) znotraj mreže in potrebno število ur t, da ga pirati raziščejo $(1 \leq x \leq N, 2 \leq y \leq M-1, 1 \leq t \leq 10^5)$

Izhod: K števil, kjer k-to število predstavlja minimalno število ur, da pirati raziščejo k otokov $(k \in \mathbb{Z} : 1 \leq k \leq K)$.

Naloga

- Mreža dimenzije $N \times M$ s K otoki
- Otokov ni v prvem (y=1) ali zadnjem stolpcu (y=M)

- Dovoljeno gibanje:
  Horizontalno: (x, y+1) ali (x, y-1)
  Vertikalno: (x+1, y), če y = 1 ali y = M
- Ni potrebno raziskati otoka na lokaciji (x_j, y_j), če nam je ta na poti do otoka na lokaciji (x_i, y_i) – torej otoke je možno preskočiti

Zgled

- N=4
  M=4
  K=5
- Minimalen čas, da obiščemo?
  - 1 otok
  - 2 otoka
  - 3 otoke
  - 4 otoke
  - 5 otokov

Zgled – min. čas, da raziščemo 1 otok

- Čas:

  Premiki:
  1. (1,1) **-> (1,2) -> (1,3)** = 2 uri

  Raziskovanje:
  - otok (1,3) = 1 ura

  <u>Skupaj = 3 ure</u>

  Otoka na (1,2) nismo raziskovali!

Zgled – min. čas, da raziščemo 2 otoka

- Čas:

  Premiki:

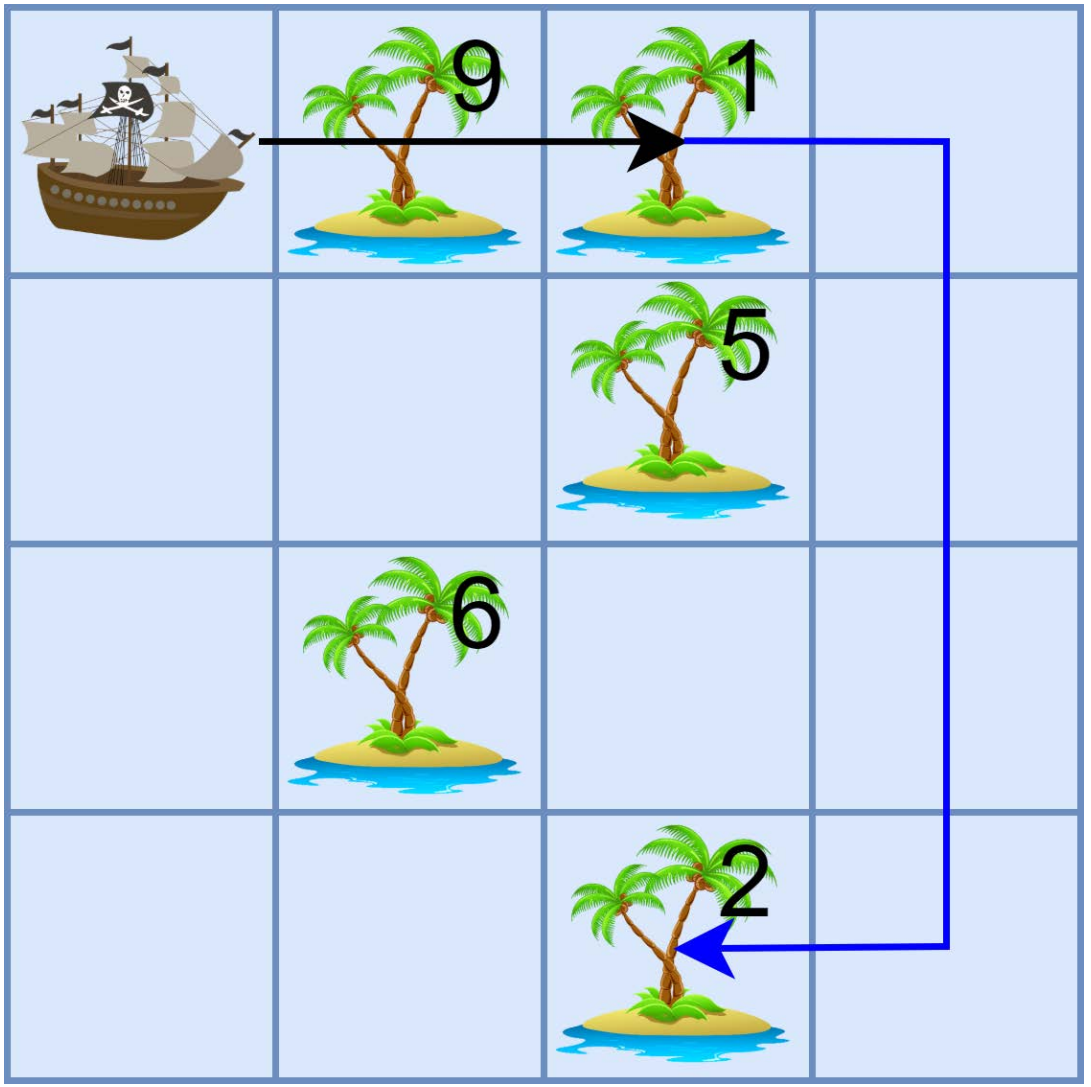
  1. (1,1) **-> (1,2) -> (1,3)** = 2 uri
  2. (1,3) -> (1,4) -> (2,4) -> (3,4) ->
     -> (4,4) -> (4,3) = 5 ur

  Raziskovanje:

  - otok (1,3) = 1 ura
  - otok (4,3) = 2 uri

  <u>Skupaj = 10 ur</u>

  Otoka na (1,2) nismo raziskovali!



Zgled – min. čas, da raziščemo 3 otoke

- Čas:

  Premiki:

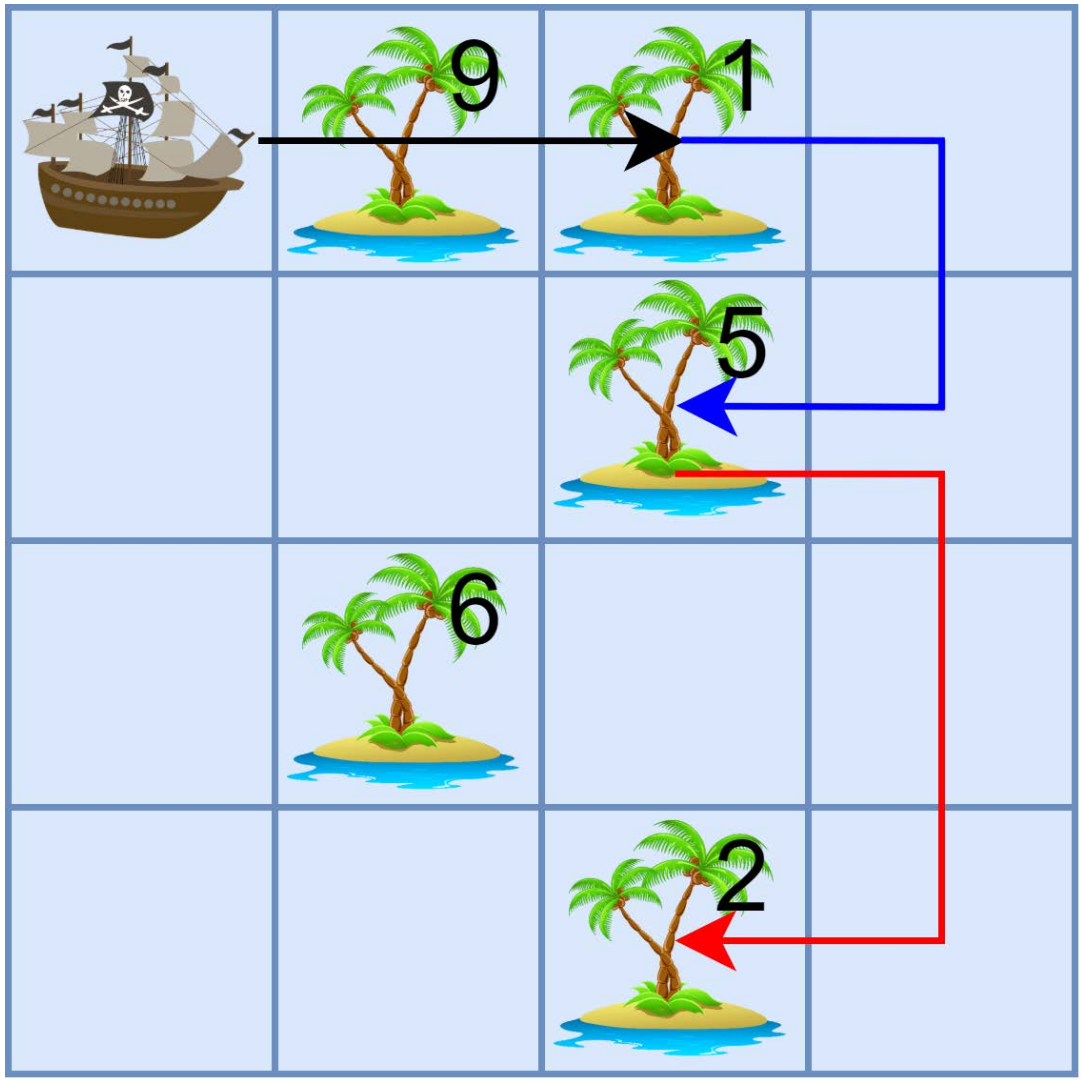
  1. (1,1) **-> (1,2) -> (1,3)** = 2 uri
  2. (1,3) -> (1,4) -> (2,4) ->
     -> (2,3) = 3 ure
  3. (2,3) -> (2,4) -> (3,4) ->
     -> (4,4) -> (4,3) = 4 ure

  Raziskovanje:

  - otok (1,3) = 1 ura
  - otok (2,3) = 5 ur
  - otok (4,3) = 2 uri

  Skupaj = 17 ur

  Otoka na (1,2) nismo raziskovali!



Zgled – min. čas, da raziščemo 4 otoke

- Čas:

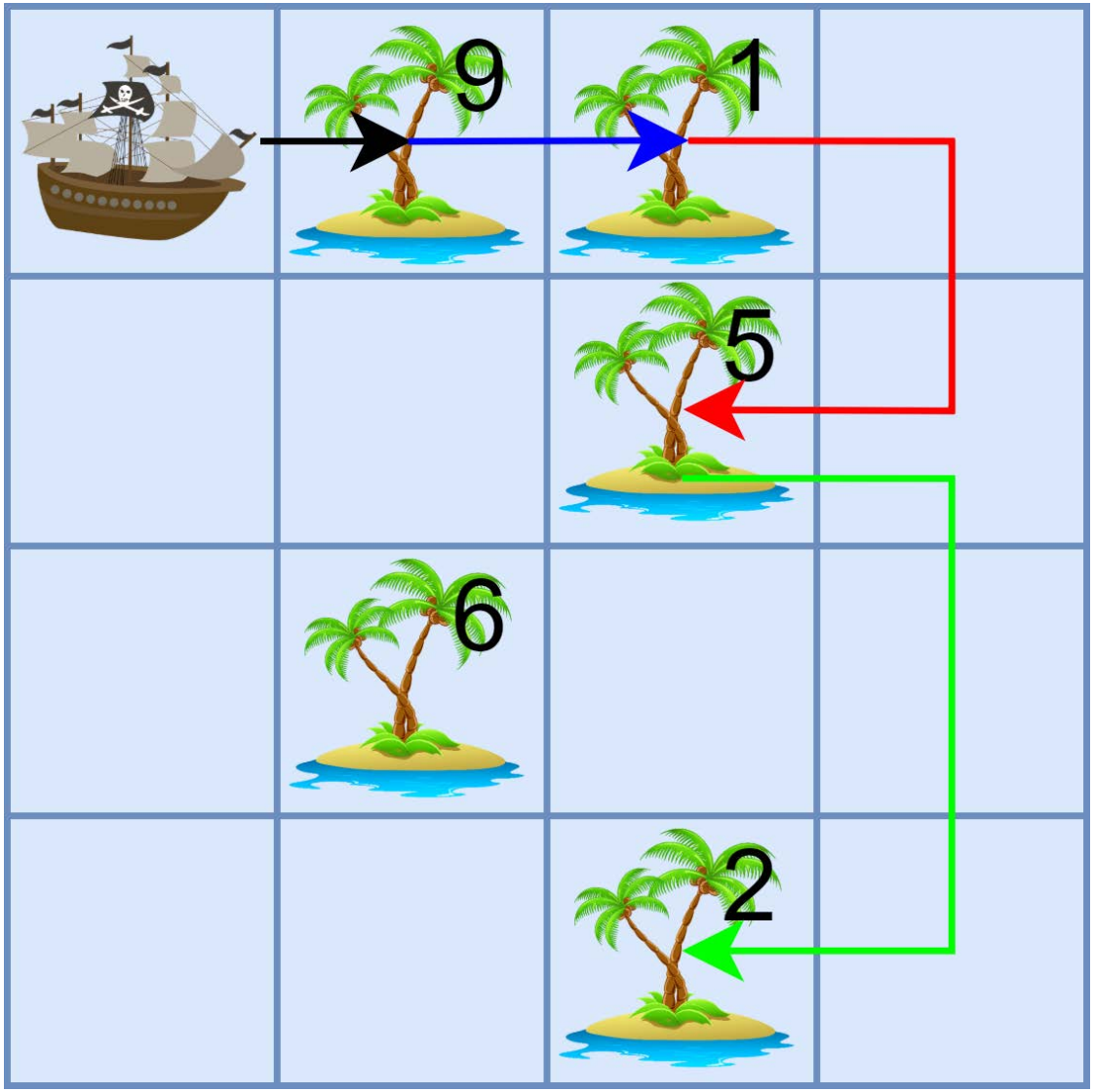
  Premiki:

  1. (1,1) **-> (1,2)** = 1 ura
  2. (1,2) -> (1,3) = 1 ura
  3. (1,3) -> (1,4) -> (2,4) ->
     -> (2,3) = 3 ure
  4. (2,3) -> (2,4) -> (3,4) -> (4,4) ->
     -> (4,3) = 4 ure

  Raziskovanje:

  - otok (1,2) = 9 ur
  - otok (1,3) = 1 ura
  - otok (2,3) = 5 ur
  - otok (4,3) = 2 uri

  <u>Skupaj = 26 ur</u>



Zgled – min. čas, da raziščemo 5 otokov

- Čas:

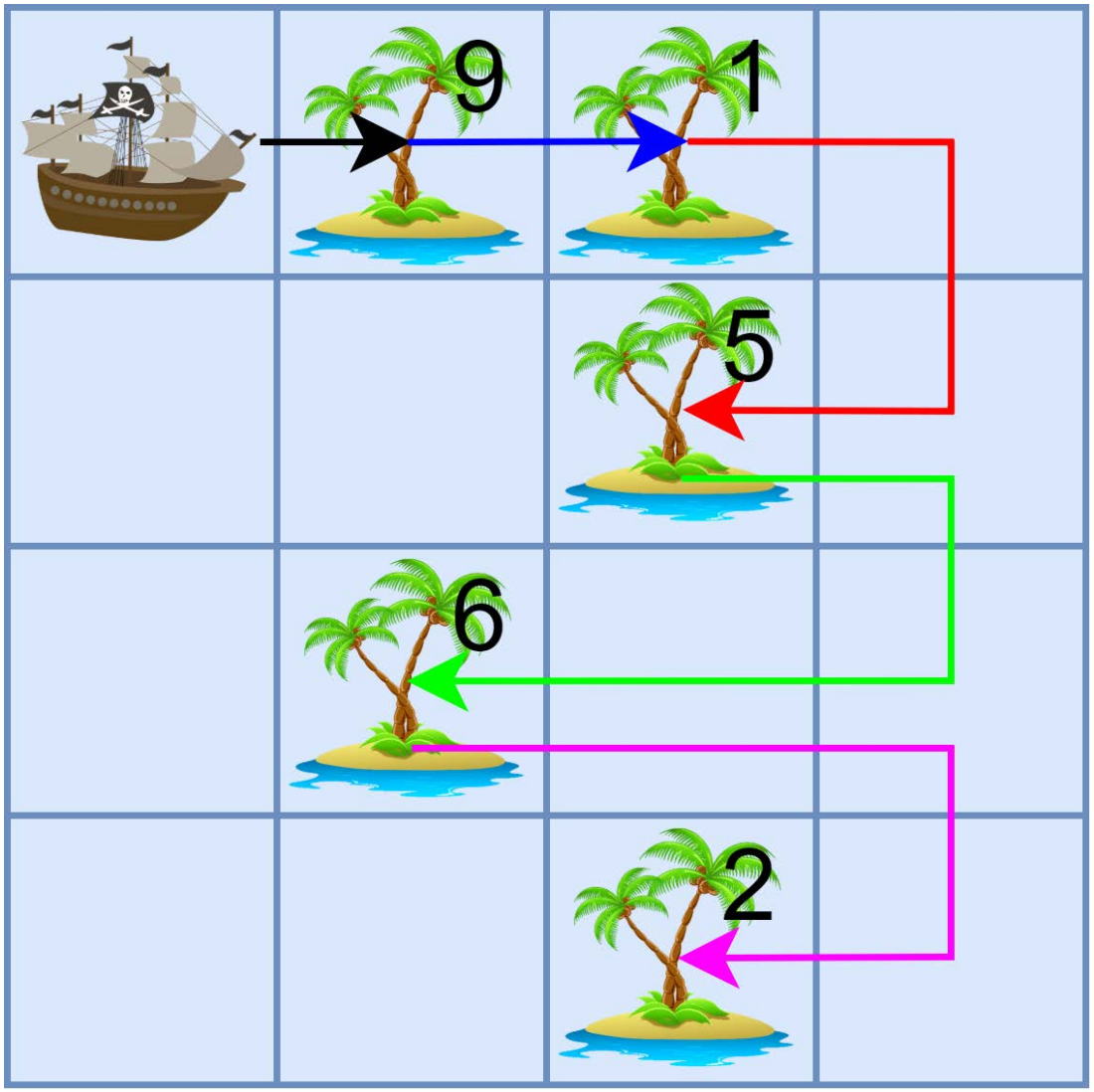
  Premiki:

  1. (1,1) **-> (1,2)** = 1 ura
  2. (1,2) -> (1,3) = 1 ura
  3. (1,3) -> (1,4) -> (2,4) ->
     -> (2,3) = 3 ure
  4. (2,3) -> (2,4) -> (3,4) -> (3,3) ->
     -> (3,2) = 4 ure
  5. (3,2) -> (3,3) -> (3,4) -> (4,4) ->
     -> (4,3) = 4 ure

  Raziskovanje:

  - otok (1,2) = 9 ur
  - otok (1,3) = 1 ura
  - otok (2,3) = 5 ur
  - otok (3,2) = 6 ur
  - otok (4,3) = 2 uri

  <u>Skupaj = 36 ur</u>



Reševanje - Tabeliranje

- Opisan je predlagan pristop s tabeliranjem

- Uporabljamo:
  - Polje rešitev RESULT (1D):
    - Velikosti K+1 (obiskali bomo od 0 do K otokov)
    - RESULT[*k*] hrani rešitev minimalnega časa za raziskovanje *k* otokov
  - Tabela stanj A (2D):
    - Velikosti [K+1, 2] (K+1 = obiščemo od 0 do K otokov,
      2 = 2 stolpca za premikanje v novo vrstico (y=1 in y=M)

- Polje RESULT vsebuje minimalen čas za raziskovanje *k* otokov, tabela stanj A pa minimalen čas za raziskovanje *k* otokov + čas za premik do stolpca y=1 in y=M + čas premikov v nove vrstice
- Vrednosti polja RESULT in tabele A so ob inicializaciji enake ∞ (inf)

Reševanje - Tabeliranje

- Ideja postopka:
  - Iteriramo skozi vrstice, ki vsebujejo otoke
  - V vsako vrstico lahko vstopimo v stolpcu y=1 ali y=M
  - 4 možnosti gibanja piratov skozi vrstico:
    - V novo vrstico vstopijo v stolpcu y=1 (pomikajo se iz leve proti desni strani (y=1,…,M)):
      - V naslednjo vrstico bodo vstopili znova v stolpcu y=1 (1. možnost gibanja)
      - V naslednjo vrstico bodo vstopili v stolpcu y=M (2. možnost gibanja)
    - V novo vrstico vstopijo v stolpcu y=M (pomikajo se iz desne proti levi strani (y=M,…,1)):
      - V naslednjo vrstico bodo vstopili v stolpcu y=1 (3. možnost gibanja)
      - V naslednjo vrstico bodo vstopili znova v stolpcu y=M (4. možnost gibanja)
  - V vrstici izračunavamo potreben čas piratov za vsa možna raziskovanja otokov (za izračune časa uporabljamo vrednosti iz tabele A, saj imajo prištet tudi čas za premike do robnih stolpcev y=1 in y=M v prejšnjih vrsticah in čas za premike v nove vrstice)
    - Posodabljamo polje RESULT z minimalnimi časi za raziskovanje *k* otokov
    - Posodabljamo tabelo A z minimalnimi časi za raziskovanje *k* otokov, katerim prištejemo še čas premikanja v y=1 ALI v y=M ter čas premikov v nove vrstice (upoštevamo zgornje 4 možnosti)
      - Za 1. in 3. možnost gibanja se minimalen čas posodablja v 1. stolpcu tabele A
      - Za 2. in 4. možnost gibanja se minimalen čas posodablja v 2. stolpcu tabele A

- Zgled reševanja* (na voljo v PDF priponki)

Nasveti

- Pri implementaciji si pomagajte z zgledom tabeliranja v PDF priponki*
- Za najtežje testne primere bo potrebno učinkovito preiskovati kombinacije premikov med otoki v vsaki vrstici
  - Npr.: 1. vrstica ima 1 otok, 2. vrstica ima 500 otokov, raziskati pa želimo 4 otoke…
  - Samostojno boste morali razviti **metodo dinamičnega programiranja** za premikanje med otoki v vrstici

    (*za lažje testne primere (1. in 2.) je dovoljeno 'brute force' preverjanje vseh kombinacij premikov med otoki v vrstici*)
- Visoka časovna zahtevnost:
  - Python bo verjetno prepočasen (veliko gnezdenih zank), bolje uporabiti C++

Zagovor vaje in kriteriji

- Primer zagona programa: ./vaja4.exe testni_primer1.txt
- Vrednost vaje: 15 točk
  - Točkovanje – delujoči testni primeri:
    - 1. testni primer: 3 točke
    - 2. testni primer: 4 točke
    - 3. testni primer: 8 točk => za izračun rešitve znotraj časovne omejitve bo potrebno dodatno implementirati **dinamično programiranje** za premikanje med otoki v vrstici
- Celotna implementacija naloge z grobo silo ALI 'hardcode'-ana rešitev dinamičnega programiranja bo ocenjena z **0 točkami**!
  - **Zgolj pri premikanju med otoki v vrstici** se lahko uporablja groba sila oz. izračuni kombinacij (delujoče zgolj za lažja testna primera 1 in 2; omenjeno na prejšnji prosojnici)
- Končni rok za oddajo: 4. december 2023 ob 6h zjutraj
- Končni rok za zagovor: 5. december 2023
- Za zagovor so koristni komentarji v izvorni kodi