Central European Programming Contest 2009
příprava na soutěž

Jan Stoklasa

stoklja5 at fel.cvut.cz

CEPC 2009

- Wroclaw, 6.-8.11.2009
- Odjezd v pátek 6.11. ve 14:30 od budovy FEL v Dejvicích
- Návrat na stejné místo v neděli 8.11. v noci
- Prosím buďte tam v pátek dřív a uložte si pro jistotu můj mobil
- POZOR: Váš tým MUSÍ mít 3 členy

Účastníci

- 71 týmů
  - Čechy
  - Slovensko
  - Polsko
  - Rakousko
  - Maďarsko
- Dobrá zpráva: favoriti z Ruska soutěží v jiném regionu

- Adam Mickiewicz University
- AGH University of Science and Technology
- Budapest University of Technology and Economics
- Charles University in Prague
- Comenius University
- **Czech Technical University in Prague**
- Eötvős Loránd University
- Jagiellonian University in Krakow
- Masaryk University
- Matej Bel University in Banská Bystrica
- Nicolaus Copernicus University
- Opole Technical University
- Pavol Jozef Šafárik University in Košice
- Polish Japanese Institute of Information Technology
- Poznan University of Technology
- Silesian University of Technology
- Slovak University of Technology in Bratislava
- Szczecin University of Technology
- University of Debrecen
- University of Ljubljana
- University of Maribor
- University of Primorska
- University of Szeged
- University of Warsaw
- University of West Bohemia in Pilsen
- University of Wroclaw
- University of Zagreb
- University of Žilina
- VŠB - Technical University of Ostrava
- Warsaw School of Computer Science
- Warsaw University of Technology
- Wroclaw University of Technology
- Wyższa Szkoła Biznesu - National-Louis University

Naše zkratka: CTU (Czech Technical University)

Výběr z pravidel

- Váš tým MUSÍ mít 3 členy
- Podobný soutěžní systém jako na CTU Open
  - 5 hodin
  - rozhoduje počet vyřešených úloh
  - druhé kritérium je čas
  - 20 minut penalizace za nesprávné řešení
- Programovací jazyky: C,C++,Pascal
  - počítejte s tím že Java nebude k dispozici

Výběr z pravidel

- Tým si smí vzít k počítači knihy a výpisy programů
  - nic v elektronické podobě, žádná CD, USB
- Do soutěžní místnosti se nesmí nosit mobily, PDA, vlastní notebooky, kalkulačky…
- Přečtěte si kompletní pravidla na http://cepc09.ii.uni.wroc.pl
  - General rules, Technical rules

Soutěž

- Vždy začínejte od nejjednodušších úloh
  - pokud si nejste jistí které to jsou, sledujte průběžné výsledky
- Výsledky runu
  - compile time error
  - run-time error
  - time limit exceeded
  - wrong answer
  - contest rule violation

Kostra úlohy

```
#include <stdio.h>
#include <stdlib.h>

// Na prvni radce vstupu je pocet testovacich pripadu (number of cases)
// Testovaci pripad: cislo N, newline, posloupnost N cisel oddelenych mezerou
// Vystup: setridene posloupnosti cisel

int cmp(const void *v1, const void *v2)
{
    return (*(int *)v1 - *(int *)v2);
}

int main()
{
  int Cases=0;
  scanf("%d",&Cases);
  for(int c=0;c<Cases;c++)
  {
     int N=0;
     scanf("%d",&N);
     int* pole=(int*)malloc(N*sizeof(int));

     for(int i=0;i<N;i++)
        scanf("%d",&pole[i]);
     qsort(pole, N, sizeof(pole[0]), cmp);

     for(int i=0;i<N;i++)
        printf("%d ",pole[i]);
     printf("\n");
     free(pole);
  }
  return 0;
}
```

Tenhle kód mějte „v ruce“

Vstupy čtěte funkcí scanf, neřešte mezery nebo nuly na konci pokud to není nutné

Pamatujte si funkci qsort a definici cmp

Úlohy se snažte řešit v poli (1D,2D), netrapte se s pointery a dynamickými strukturami

Pokud je v úloze zadaná maximální velikost pole, použijte statické pole

malloc – velikost pole vždy násobte velikostí typu (N*sizeof(int))

Z funkce main vracejte 0

Typické druhy úloh a užitečné algoritmy

- Hledání půlením intervalu
- Hledání nejkratších cest
- Prohledávání grafu/stavového prostoru
- Topologické třídění
- Dynamické programování
- Úlohy s přirozenými čísly
- Geometrické úlohy

Hledání půlením intervalu

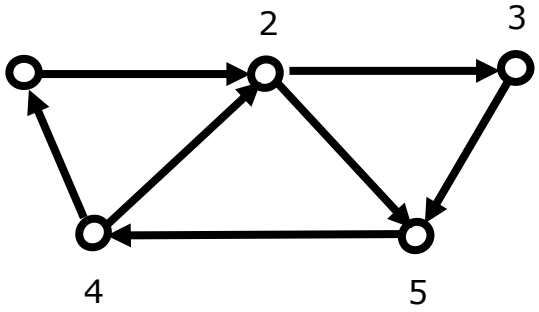
- Binary search
- Může být v úloze „schovaný“
- Pozor, častá „chyba o jedničku“ při implementaci (použijte vzor)

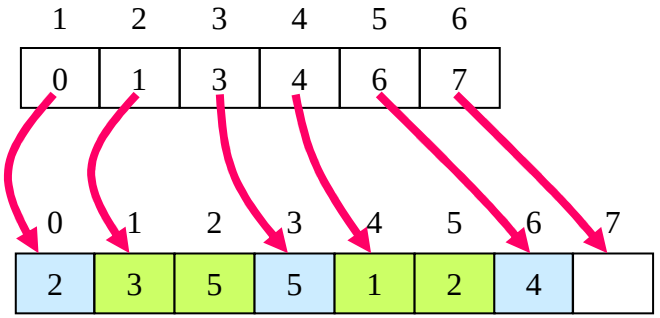
Příklad: A Careful Approach

- světové finále ACM ICPC 2009
- Zkusíme všechny možnosti
- 2-8 letadel, 8!=40320
- Časový interval 0-86400 sekund
- 40320*86400=3483648000 možností?
  Time limit exceeded
- Lepší by bylo $40320*\log_2(86400)=661198$

Implementace grafu v poli

- vhodné pro graf který se v průběhu výpočtu nemění
- neorientovaný graf – obě orientace hrany
- setřídění hran po přečtení (moc) nezhorší složitost





Hledání nejkratších cest

- **Dijkstrův algoritmus**
- Graf s **kladným** ohodnocením hran (vzdálenost, cena)
- Vzdálenost z jednoho vrcholu grafu do všech ostatních
- Složitost
  - O(log(počet vrcholů)*počet_hran)

    O(log(počet_vrcholů)*počet_vrcholů^**2**)

Hledání nejkratších cest

- **Floyd-Warshallův algoritmus**
- Graf s **kladným** ohodnocením hran
- Vzdálenosti mezi všemi dvojicemi vrcholů
- Složitost
  - O(počet_vrcholů^**3**)
- jednodušší na naprogramování ale úlohy jsou obvykle nastavené tak aby nestíhal
- POZOR při implementaci: vnější cyklus přes k

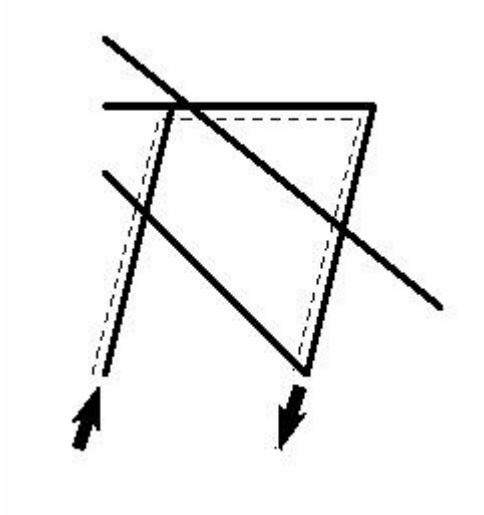
Příklad: Vlez mi na záda

- Je zadána mapa s vzdálenostmi mezi jednotlivými body (graf s ohodnocenými hranami)
- Dva závodníci (A a B) běží ze startu do cíle tak že jeden nese druhého na zádech a po každém úseku se musejí vystřídat
- Známe rychlost A když nese B a rychlost B když nese A

- (autorem této úlohy je kolega Doc. Josef Kolář)

Příklad: Robotic rails

- CTU Open 2009

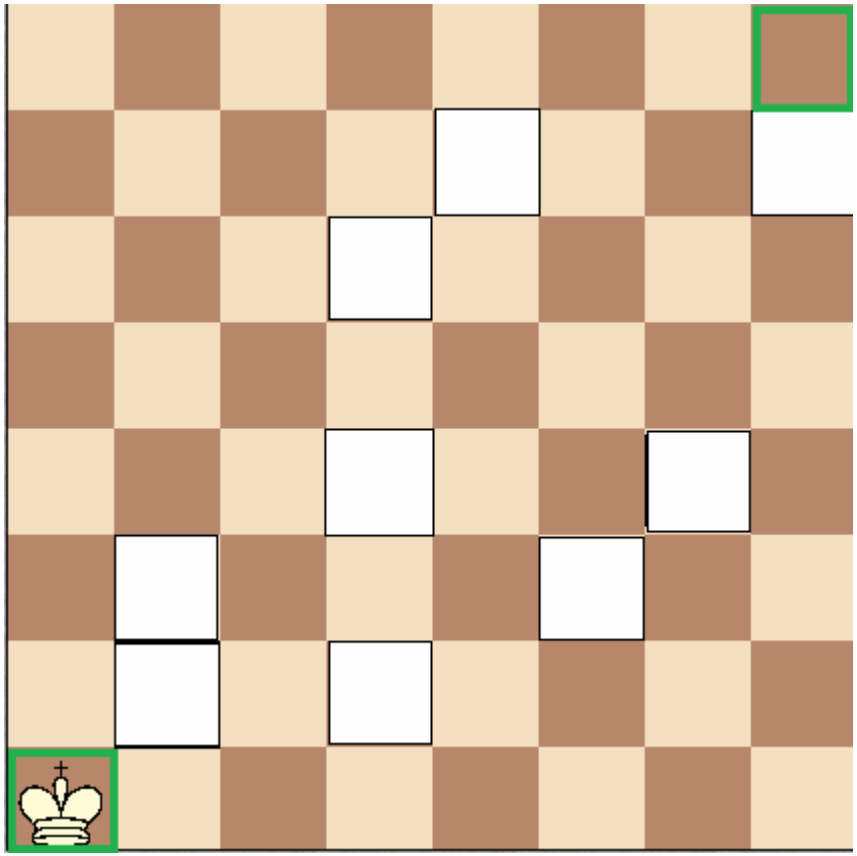


- Dijkstrův algoritmus + bereme v úvahu natočení robota

Prohledávání grafu

- Graf bez ohodnocení hran (1 hrana=1 krok)
- Prohledávání do šířky (Breadth-First Search)
  - rychlý: O(počet_vrcholů+počet_hran)
  - vrcholy dáváme do fronty
  - najde nejkratší cestu
  - fronta zabírá paměť
- Prohledávání do hloubky (Depth-First Search)
  - O(počet_vrcholů+počet_hran)
  - vrcholy dáváme na zásobník (rekurze)
  - najde cestu ale ne nutně tu nejkratší
  - menší paměťové nároky

Příklad: král na děravé šachovnici



Kolika tahy se král nejrychleji dostane z a1 na h8?

Příklad: Hra Frogger

- Nordic Collegiate Contest 2003

- Kolika skoky se žabka nejrychleji dostane na druhou stranu?

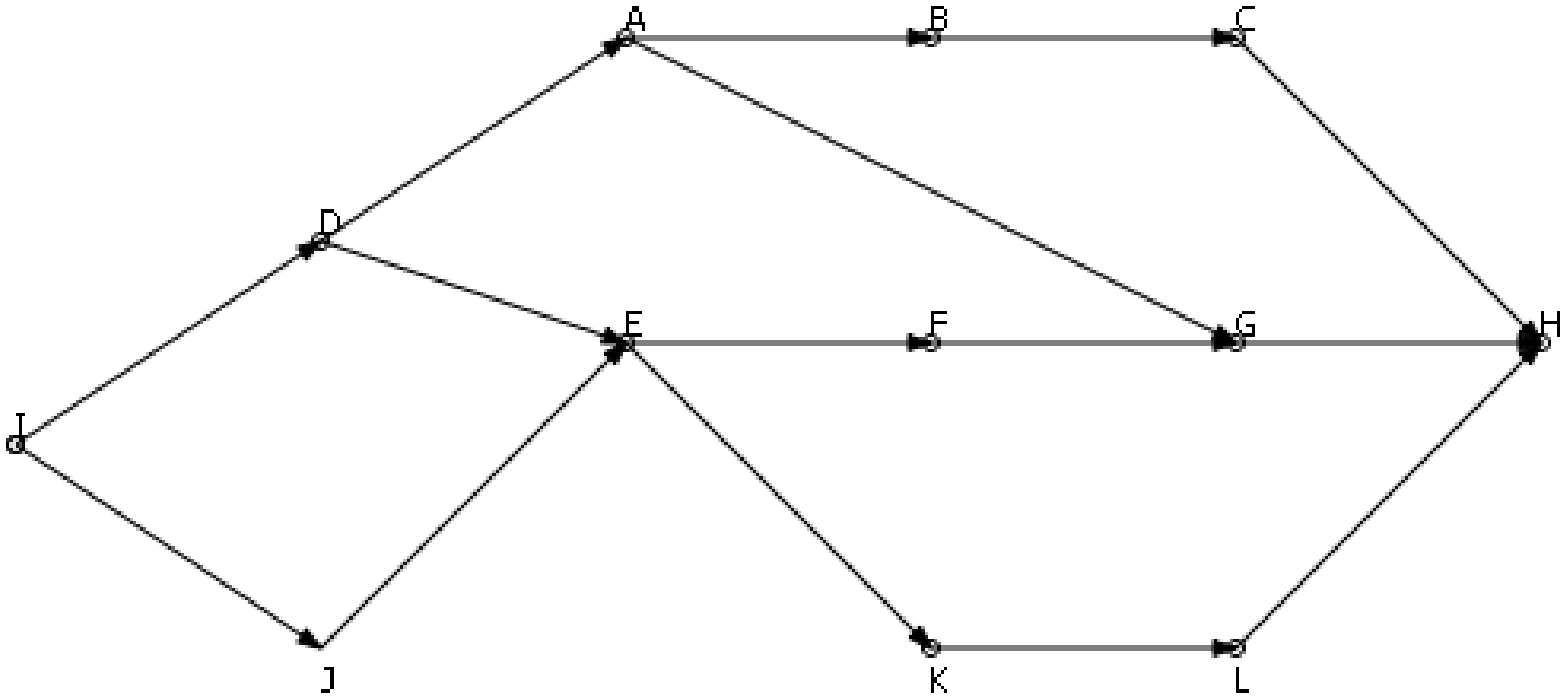
Topologické třídění

- Zní to vědecky ale je to jednoduchý algoritmus.
- Motivace: naplánujte úkoly tak aby byla dodržena jejich návaznost (než si dám kávu, musím si jí uvařit, než si uvařím kávu, musím zapnout vodu)

Algoritmus (orientovaný graf):

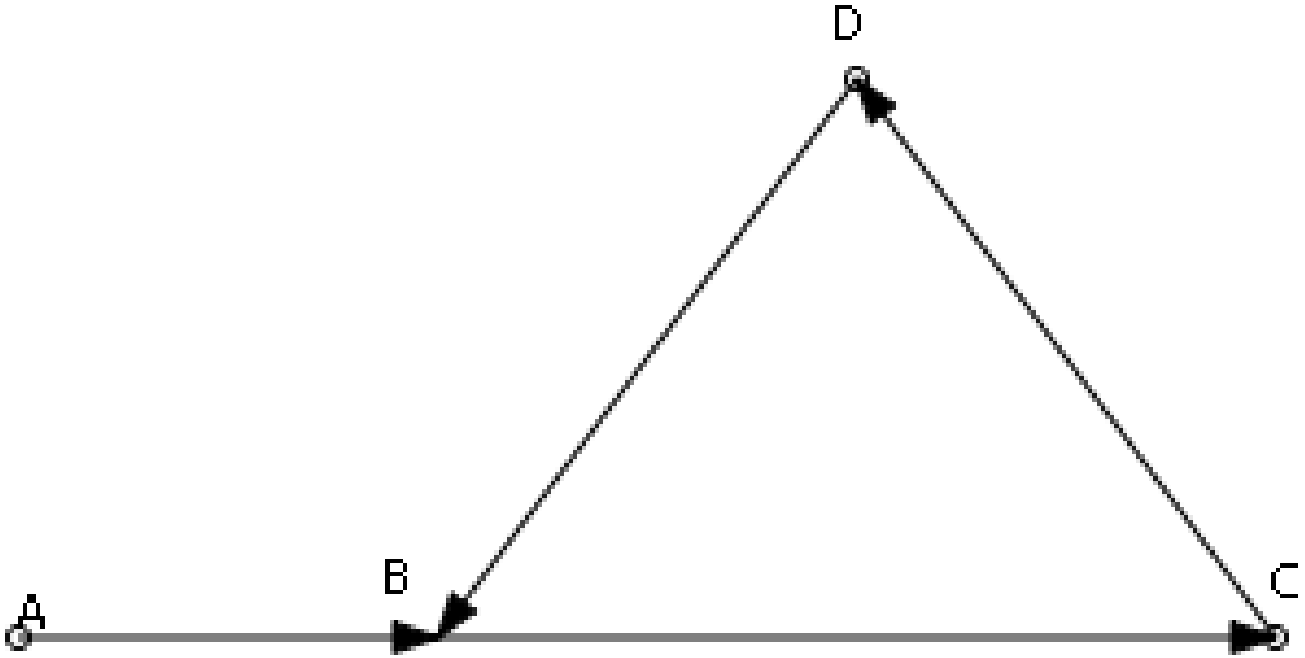
```
while(graf není prázdný) {
    najdi vrchol který nemá předchůdce
    pokud neexistuje, error "V grafu je cyklus"
    odstraň tento vrchol z grafu
}
```

Topologické třídění



I,D,J,A,E,B,C,F,G,K,L,H

Topologické třídění



NELZE – V grafu je cyklus B,C,D

Příklad: Letter Lies

- CTU Open Contest 2009
- Topologické třídění řeší návaznost vět

Dynamické programování

- Řešení většího zadání se dá odvodit z řešení menších zadání
- Počítám šikovně tak abych menší zadání počítal jen jednou

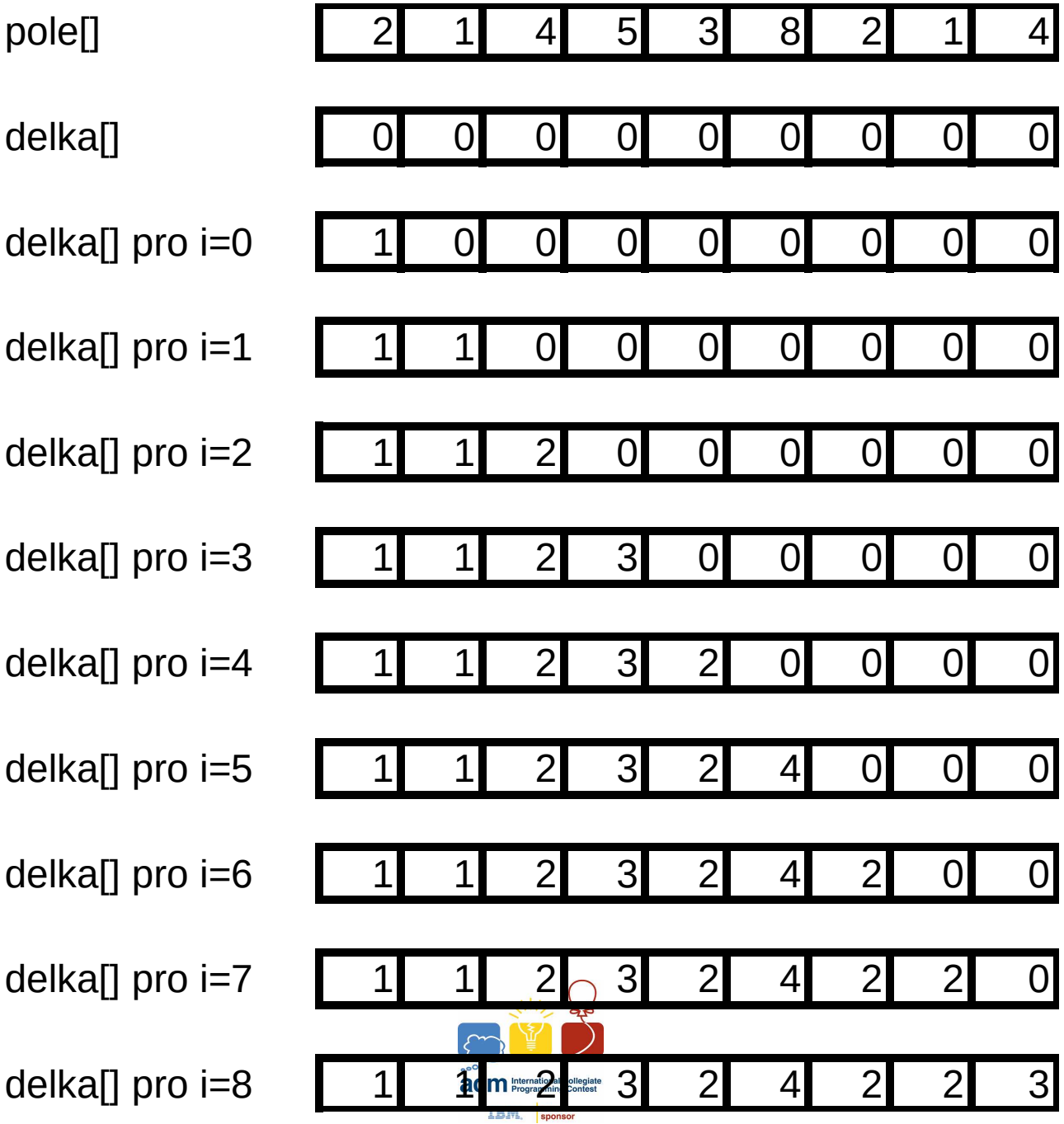
- Nejjednodušší příklad: Fibonacciho čísla
- fib(x) = fib(x-1)+fib(x-2) // exponenciální
- pamatuji si dříve spočítané hodnoty, nebo to rovnou spočítám v cyklu od nejmenšího

Příklad: Nejdelší neklesající podposloupnost

- Posloupnost kladných čísel: 2,1,4,5,3,8,2,1,4
- Zajímají nás její neklesající podposloupnosti
  - 2,3,8
  - 2,3,4
  - 2,4,5,8
- a hledáme tu nejdelší z nich
- Počet všech podposloupností – exponenciální

Příklad: Nejdelší neklesající podposloupnost

```
// Dynamické programování: řešení v O(n^2) v 1D poli
// delka[i] obsahuje délku nejdelší neklesající podposloupnosti která končí prvkem i
for(int i=0;i<N;i++)
{
    int max=0;
    for(int j=0;j<i;j++)
    {
        if(pole[j]<=pole[i] && delka[j]>max)
            max=delka[j];
    }
    delka[i]=max+1;
}
```


pole[] 2 1 4 5 3 8 2 1 4
delka[] 0 0 0 0 0 0 0 0 0
delka[] pro i=0 1 0 0 0 0 0 0 0 0
delka[] pro i=1 1 1 0 0 0 0 0 0 0
delka[] pro i=2 1 1 2 0 0 0 0 0 0
delka[] pro i=3 1 1 2 3 0 0 0 0 0
delka[] pro i=4 1 1 2 3 2 0 0 0 0
delka[] pro i=5 1 1 2 3 2 4 0 0 0
delka[] pro i=6 1 1 2 3 2 4 2 0 0
delka[] pro i=7 1 1 2 3 2 4 2 2 0
delka[] pro i=8 1 1 2 3 2 4 2 2 3

Příklad: Nejdelší společná podposloupnost

- Jsou dány dvě posloupnosti
  - **B**,D,**C**,A,**B**,**A**
  - A,**B**,**C**,**B**,D,**A**,B
- Hledáme nejdelší společnou podposloupnost
  - **B,C,B,A**
- Algoritmus LCS (Longest common subsequence)
- Algoritmická klasika - základ unixové utility diff

Příklad: Nejdelší společná podposloupnost

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Obrázek 7.12: Nejdelší podposloupnost

Zdroj: skripta Teoretická informatika, Doc. Josef Kolář

Příklad: Nejdelší společná podposloupnost

Algoritmus 7.12 Nejdelší společná podposloupnost

```
NSP(A, B)
1   m := d(A); n := d(B)                          Určení délek posloupností
2   for i := 1 to m do s[i, 0] := 0               a inicializace
3   for j := 1 to n do s[0, j] := 0               okrajů pole s.
4   for i := 1 to m do                            Hlavní cyklus
5       for j := 1 to n do                        běží po řádcích:
6           if  a_i = b_j                         je-li společný konec,
7               then s[i, j] := s[i − 1, j − 1] + 1   použije se.
8                    t[i, j] := " ↖ "
9               else if s[i − 1, j] ≥ s[i, j − 1]     Jinak se bere
10                   then s[i, j] := s[i − 1, j]       delší ze dvou
11                        t[i, j] := " ↑ "             uvažovaných možností.
12                   else s[i, j] := s[i, j − 1]
13                        t[i, j] := " ← "
14  return s, t
```

Zdroj: skripta Teoretická informatika, Doc. Josef Kolář

Úlohy s přirozenými čísly

- Typický druh úlohy: velmi dlouhý interval, vybrat z něj jen některá čísla
- Většinou to nejde udělat jednoduše

```
for(int i=0;i<INT_MAX;i++)
    otestuj(i);
```

- Obvykle je v tom nějaký trik

Příklad: Kvadratický ciferný součet

- Vezmeme číslo v desítkovém zápise, každou číslici umocníme na druhou a sečteme, operaci opakujeme
- Některá čísla skončí v 1, některá ne
- 5555 -> 100 -> 1
- 147 -> 72 -> 53 -> 34 -> 25 -> 29 -> 85 -> **89** -> 145 -> 42 -> 20 -> 4 -> 16 -> 37 -> 58 -> **89** (cyklus)

Příklad: Kvadratický ciferný součet

- Úkolem je určit počet čísel v daném intervalu které skončí v 1
- Triky:
  - na pořadí číslic nezáleží, po jednom kroku se 1234 a 4321 dostanou na stejnou hodnotu
  - velká čísla se touto operací rychle zmenšují (123456789->285), předpočítám pro malá čísla a když se dostanu do čísla o kterém už vím jestli končí v 1 už nemusím pokračovat

Příklad: Yaptcha

- CEPC 2008

$$S_n = \sum_{k=1}^{n} \left[\frac{(3k+6)!+1}{3k+7} - \left[\frac{(3k+6)!}{3k+7} \right] \right]$$

- (1 <= n <= 10^6) a to celé miliónkrát
- Není reálné spočítat to ve for cyklu
- n > 1 je prvočíslo právě když (n-1)! = -1 (mod n)
- Stačí si předpočítat počet k pro které je 3*k+7 prvočíslo

Prvočíselné triky

- Jak rychle zjistit jestli je číslo x prvočíslo?
- Naivní postup: zkouším najít dělitele x od 2 do odmocniny x
- Vylepšení: mám připravený seznam prvočísel od 2 do řekněme 2000 a napřed zkouším prvočísla
  - viz yaptcha.java

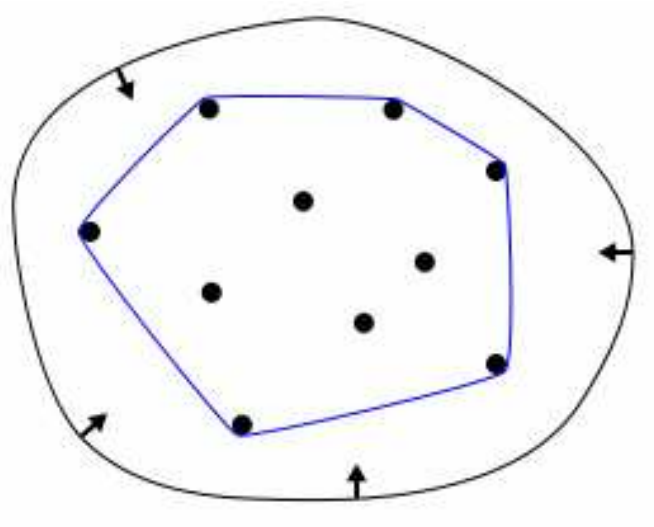
Prvočíselné triky

- n > 1 je prvočíslo právě když (n-1)! = -1 (mod n)
- Wilsonova věta

- Další užitečný trik na prvočísla je malá Fermatova věta
- p prvočíslo => a^(p-1)=1(mod p)
- (opačná implikace neplatí)

Geometrické úlohy

- Může se hodit konvexní obálka

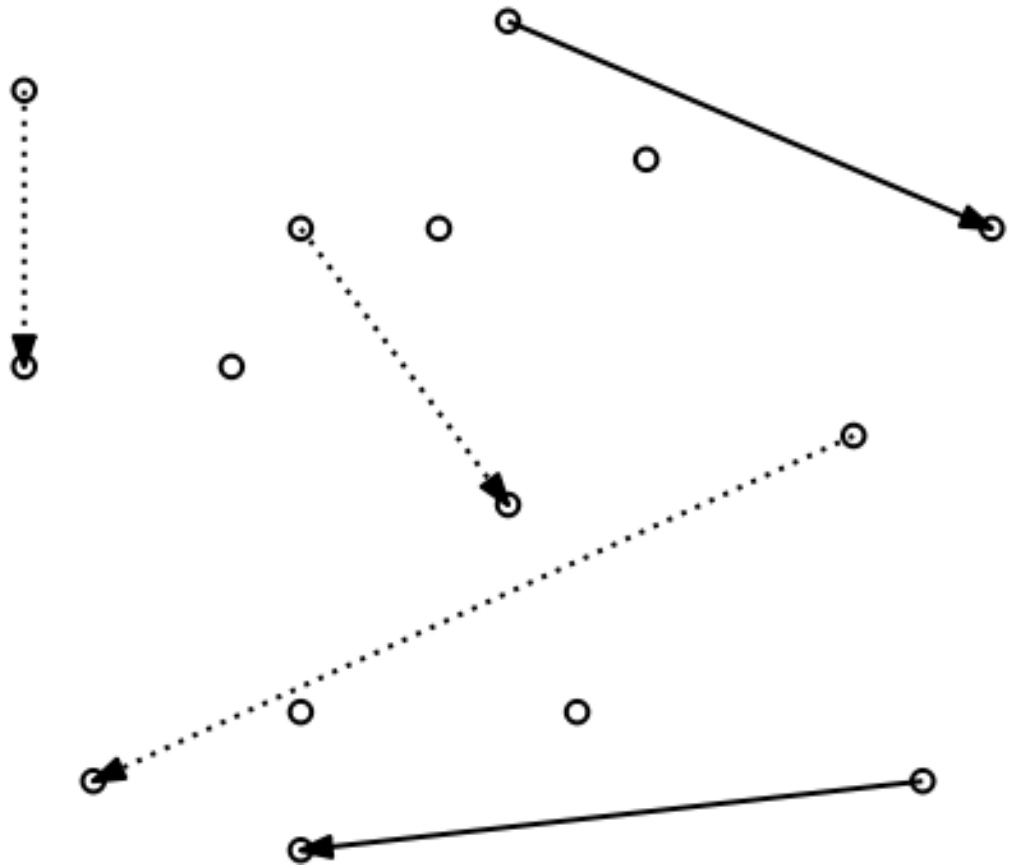


Zdroj: wikipedia.com

Konvexní obálka

- Snadno zapamatovatelný algoritmus
  - zkouším všechny dvojice bodů, spojím je orientovaným vektorem
  - zjišťuju jestli všechny ostatní body leží vpravo od přímky definované tímto vektorem
  - O(n^3)
  - existují i rychlejší algoritmy ale tenhle se jednoduše pamatuje

Konvexní obálka



A to je všechno…

- Good Luck!