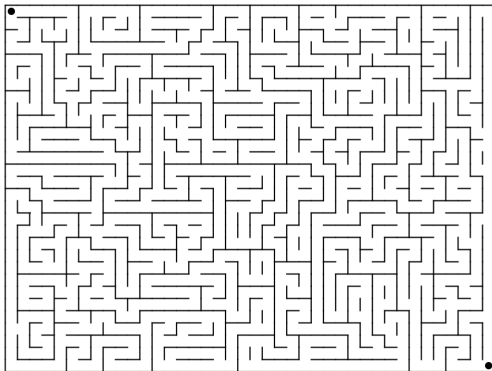


Grafové algoritmy

Obsah

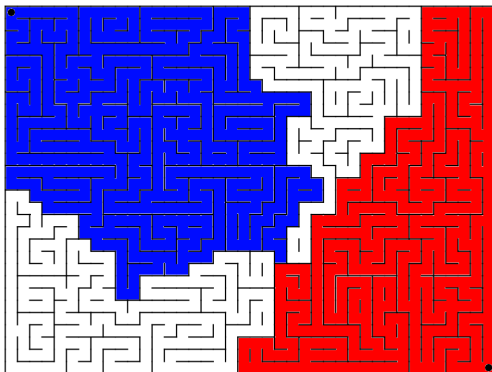
- 1 Grafy
 - Pojem grafu
 - Příklady grafů
 - Reprezentace grafu
- 2 Procházení grafu
 - Procházení do šířky
 - Procházení do hloubky
 - Aplikace
 - Příklady
- 3 Složitější algoritmy
 - Nejkratší vzdálenosti
 - Kostra grafu
 - Další
- 4 Shrnutí

Motivační příklad I

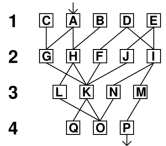
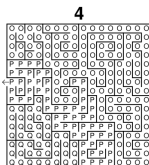
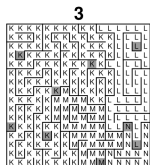
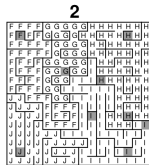
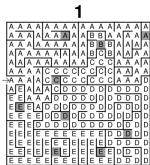


Kde prokopat zeď, aby se bludiště stalo průchodným?

Motivační příklad I řešení



Motivační příklad II řešení



Graf

Základní pojmy:

- uzly (vrcholy)
- hrany: orientované, neorientované, vážené
- stupeň vrcholu
- cesta v grafu, dosažitelnost, cyklus
- souvislost, komponenty
- strom, klika (úplný graf)

Použití grafů

- dopravní síť (silniční, vlaková, letecká, ...)
- elektrická síť
- Internet
- sociální síť
- plánování: závislosti mezi úlohami
- stavový prostor logické úlohy

Potravní řetězce

uzly: zvířata, hrany: pokud jedno žere druhé

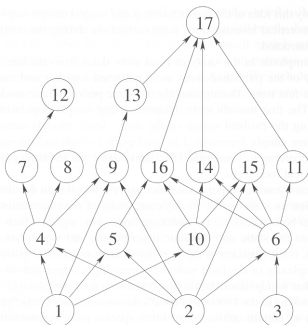
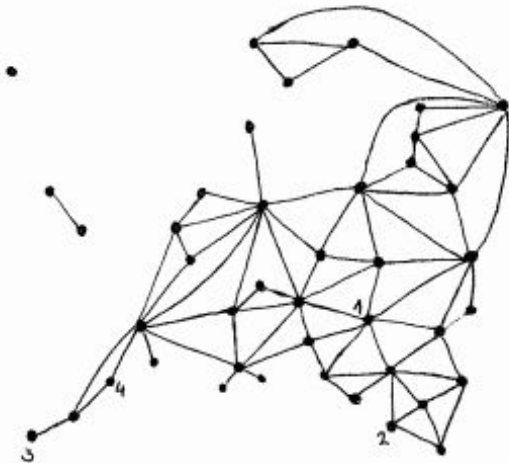


Figure 10.1: Narragansett Bay food web. 1=flagellates, diatoms; 2=particulate detritus; 3=macroalgae, eelgrass; 4=*Acartia*, other copepods; 5=sponges, clams; 6=benthic macrofauna; 7=ctenophores; 8=meroplankton, fish larvae; 9=pacific menhaden; 10=bivalves; 11=crabs, lobsters; 12=butterfish; 13=striped bass, bluefish, mackerel; 14=demersal species; 15=starfish; 16=flounder; 17=man. (After Yodzis 1989. *Introduction to theoretical ecology*, Harper and Row, New York.)

Co je to?



Logická úloha: Misionáři a kanibalové

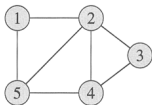


- 3 misionáři, 3 kanibalové
- řeka, 1 loďka (max 2 lidé)
- víc kanibalů jak misionářů na jednom místě \Rightarrow problém
- (jen jeden misionář a jeden kanibal umí pádlovat)
- zkuste:
 - najít řešení
 - „najít“ v tom graf

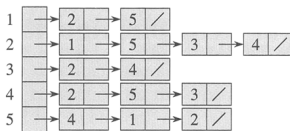
Reprezentace grafu v počítači

- seznamy sousedů
 - pro každý vrchol seznam jeho sousedů
 - vhodné pro řádké matice
- matice souslednosti
 - binární matice
 - pro každou dvojici vrcholů: sousedí (1) / nesousedí (0)
 - vhodné pro malé nebo husté matice

Reprezentace grafu: příklad neorientovaný graf



(a)



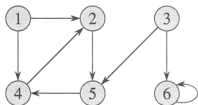
(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

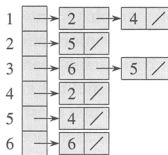
(c)

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

Reprezentace grafu: příklad orientovaný graf



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

Procházení grafu

procházení grafu = základní operace nad grafy

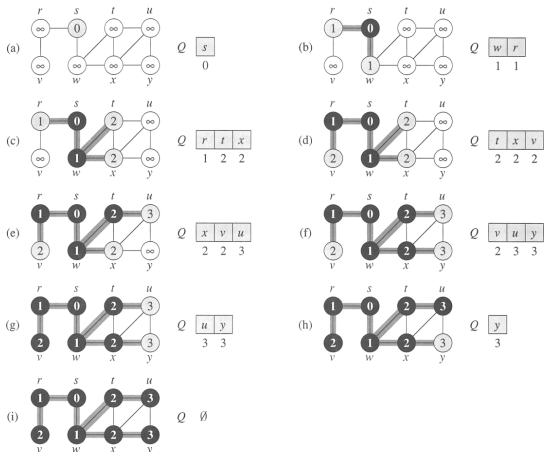
- procházení do šířky
- procházení do hloubky

Procházení do šířky

BFS = breadth-first search

- „povodeň“ z počátečního vrcholu
- realizace pomocí fronty
- pro aktuální vrchol:
 - projdu všechny sousedy
 - pokud soused nebyl navštíven, dám jej do fronty

Ilustrace 2



Pseudokód

Algorithm 9.4 BFS

Input: A directed or undirected graph $G = (V, E)$.

Output: Numbering of the vertices in breadth-first search order.

1. $bf_n \leftarrow 0$
2. **for** each vertex $v \in V$
3. mark v *unvisited*
4. **end for**
5. **for** each vertex $v \in V$
6. **if** v is marked *unvisited* **then** $bf_n(v)$
7. **end for**

Procedure $bf_n(v)$

1. $Q \leftarrow \{v\}$
2. mark v *visited*
3. **while** $Q \neq \{\}$
4. $v \leftarrow Pop(Q)$
5. $bf_n \leftarrow bf_n + 1$
6. **for** each edge $(v, w) \in E$
7. **if** w is marked *unvisited* **then**
8. Push(w, Q)
9. mark w *visited*
10. **end if**
11. **end for**
12. **end while**

Procházení do šířky – poznámky

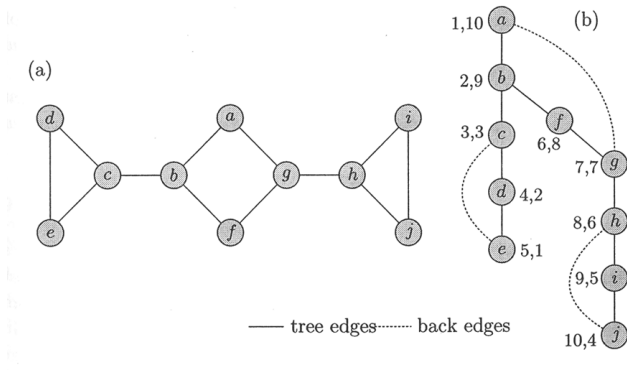
- procházím v pořadí podle vzdálenosti od zdroje
- jednoduché napočítat vzdálenosti (počet kroků)
- strom předchůdců – rekonstrukce nejkratších cest

Procházení do hloubky

DFS = depth-first search

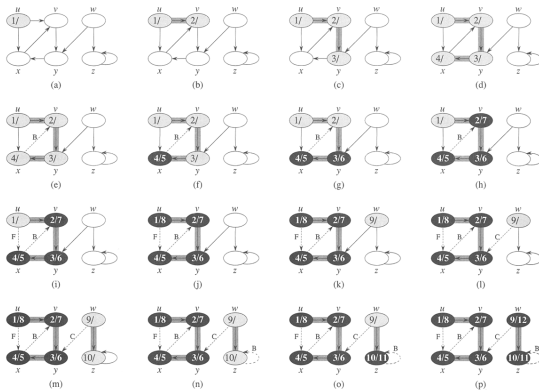
- „zavrtávání“ se do hloubky
- realizace pomocí rekurze (zásobníku)
- pokud najdu nenavštíveného souseda, začnu prohledávat z něho
- ostatní sousedy obsloužím až později

Ilustrace



M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

Ilustrace 2



Pseudokód

Algorithm 9.1 DFS

Input: A (directed or undirected) graph $G = (V, E)$.

Output: Preordering and postordering of the vertices in the corresponding depth-first search tree.

1. $predfn \leftarrow 0$; $postdfn \leftarrow 0$
2. **for** each vertex $v \in V$
3. mark v *unvisited*
4. **end for**
5. **for** each vertex $v \in V$
6. **if** v is marked *unvisited* **then** $dfs(v)$
7. **end for**

Procedure $dfs(v)$

1. mark v *visited*
2. $predfn \leftarrow predfn + 1$
3. **for** each edge $(v, w) \in E$
4. **if** w is marked *unvisited* **then** $dfs(w)$
5. **end for**
6. $postdfn \leftarrow postdfn + 1$

Procházení do hloubky – poznámky

- jednoduché na implementaci pomocí rekurze
- užitečné vlastnosti využitelné pro aplikace, např.
 - detekce cyklů
 - (silné) komponenty souvislosti
 - topologické třídění

Aplikace prohledávání

- komponenty spojitosti
- hledání nejkratších cest
- detekce cyklů
- bipartita

Komponenty souvislosti

- komponenta souvislosti = maximální množina vrcholů $U \subseteq V$, každé dva vrcholy z U vzájemně dosažitelné
- detekce pomocí prohledávání (do šířky, do hloubky)

Hledání nejkratších cest

- pokud všechny hrany mají váhu 1
- hledání nejkratších cest pomocí BFS
- jednoduchá úprava

Detekce cyklu

- cyklus (kružnice) – cesta z vrcholu sama do sebe
- jak poznat, zda graf obsahuje cyklus?

Detekce cyklu

- cyklus (kružnice) – cesta z vrcholu sama do sebe
- jak poznat, zda graf obsahuje cyklus?
- neorientovaný – spočítat počet hran a vrcholů
- orientovaný – pomocí DFS
- kontrola, zda je soused aktuálního vrcholu na zásobníku

Bipartitní graf

- bipartitní graf
 - existuje rozdělení množiny vrcholů na V_1 , V_2
 - hrany vedou pouze mezi V_1 a V_2 , nikoliv v rámci množin
- jak poznat, zda je graf bipartitní?

Bipartitní graf

- bipartitní graf
 - existuje rozdělení množiny vrcholů na V_1 , V_2
 - hrany vedou pouze mezi V_1 a V_2 , nikoliv v rámci množin
- jak poznat, zda je graf bipartitní?
- pomocí BFS (či DFS) – průběžně přiřazují do množiny 1/2 a kontrolují

Použití procházení grafu

- mnoho problémů lze řešit jednoduše pomocí aplikace procházení grafu
- klíčové správně „pojmenovat“ graf

Příklad: robot v bludišti

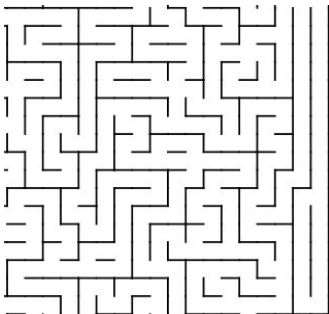
- čtverečkové bludiště
- robot s operacemi: krok, rotace vlevo, rotace vpravo
- jak se dostat na co nejméně operací z jednoho místa do druhého

Logická úloha: Misionáři a kanibalové



- 3 misionáři, 3 kanibalové
- řeka, 1 loďka (max 2 lidé)
- víc kanibalů jak misionářů na jednom místě \Rightarrow problém
- (jen jeden misionář a jeden kanibal umí pádlovat)
- zkuste:
 - najít řešení
 - „najít“ v tom graf

Příklad: generování bludiště



- bludiště \sim graf
- generování bludiště pomocí randomizovaného DFS – prokopávání zdí

Složitější algoritmy nad grafy

- nejkratší vzdálenosti
- kostra grafu
- eulerovská cesta – domečkologie
- hamiltonovská cesta – problém obchodního cestujícího
- toky v sítích

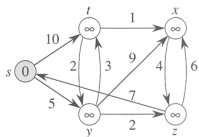
Nejkratší vzdálenosti

více různých problémů

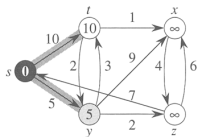
- váhy hran:
 - konstantní (1)
 - přirozená čísla
 - celá čísla
- odkud kam:
 - z jednoho vrcholu do druhého (SSSP = single source shortest path)
 - mezi všemi dvojicemi vrcholů (APSP = all pairs shortest path)

Dijkstrův algoritmus

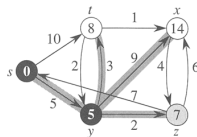
SSSP s kladnými hranami



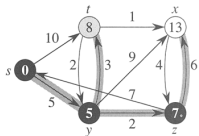
(a)



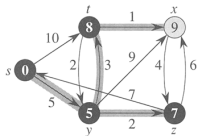
(b)



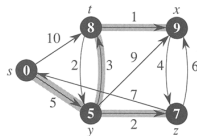
(c)



(d)



(e)



(f)

Dijkstrův algoritmus

SSSP s kladnými hranami

- opakuj:
 - 1 vyber nezpracovaný vrchol s nejmenší vzdáleností od startu
 - 2 zpracuj vrchol: aktualizuj vzdálenost sousedů
- efektivní implementace: prioritní fronta

APSP s kladnými hranami

- naivně:
 - spustit SSSP z každého vrcholu
 - neefektivní
- Floyd-Warshallův algoritmus:
 - nejkratší vzdálenosti vedoucí přes vrchol 1
 - nejkratší vzdálenosti vedoucí přes vrcholy 1, 2
 - nejkratší vzdálenosti vedoucí přes vrcholy 1, 2, 3
 - ...

Floyd-Warshallův algoritmus

APSP s kladnými hranami

Algorithm 7.3 FLOYD

Input: An $n \times n$ matrix $l[1..n, 1..n]$ such that $l[i, j]$ is the length of the edge (i, j) in a directed graph $G = (\{1, 2, \dots, n\}, E)$.

Output: A matrix D with $D[i, j] =$ the distance from i to j .

1. $D \leftarrow l$ {copy the input matrix l into D }
2. **for** $k \leftarrow 1$ to n
3. **for** $i \leftarrow 1$ to n
4. **for** $j \leftarrow 1$ to n
5. $D[i, j] = \min\{D[i, j], D[i, k] + D[k, j]\}$
6. **end for**
7. **end for**
8. **end for**

Celočíselné hrany

- mohou být i záporné váhy hran, problémy:
 - nelze jednoduše najít ideální pořadí počítání vzdáleností
 - záporný cyklus
- obecný přístup:
 - opakovaná „relaxace“ hran
 - detekce záporných cyklů

Kostra grafu

- kostra grafu = minimální množina hran, tak že graf je spojitý
- ceny hran → nejlevnější kostra grafu
- aplikace: např. elektrická síť
- historie: prof. Borůvka

Kruskalův algoritmus

- seřadit hrany podle ceny
- postupně procházet hrany: vytvoří hrana cyklus?
 - ano → zahodit
 - ne → použít
- datová struktura: union-find

Kruskalův algoritmus

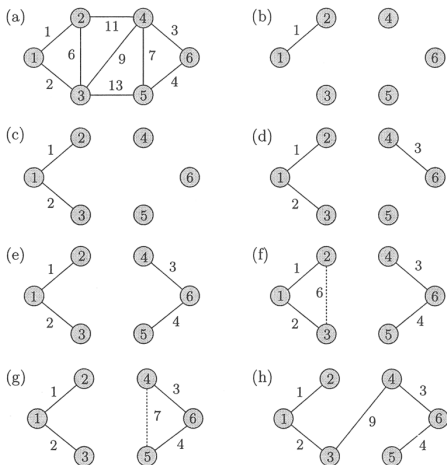
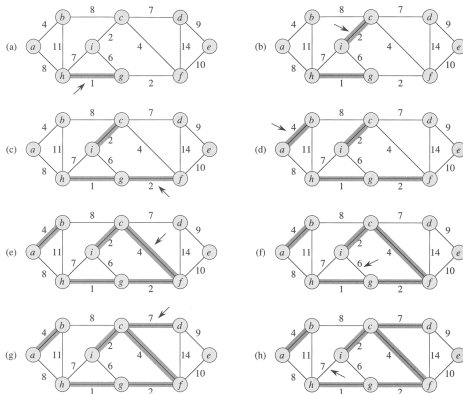


Fig. 8.4 An Example of Kruskal Algorithm.

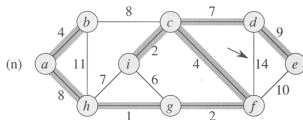
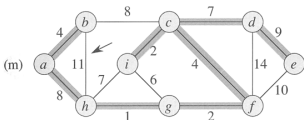
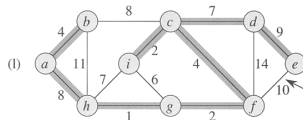
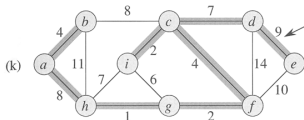
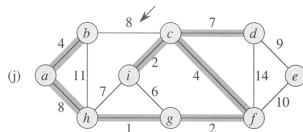
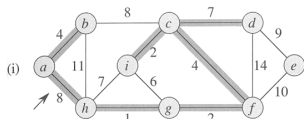
Kostra grafu

Kruskalův algoritmus



T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

Kruskalův algoritmus



Kruskalův algoritmus

Algorithm 8.3 KRUSKAL

Input: A weighted connected undirected graph $G = (V, E)$ with n vertices.

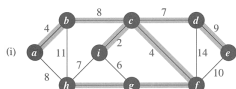
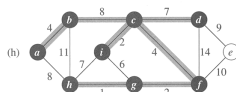
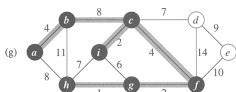
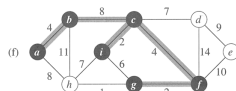
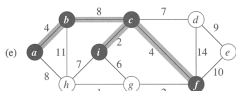
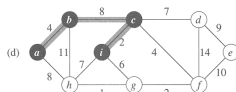
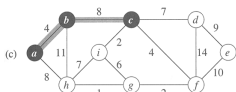
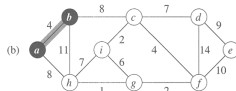
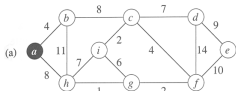
Output: The set of edges T of a minimum cost spanning tree for G .

1. Sort the edges in E by nondecreasing weight.
2. **for** each vertex $v \in V$
3. MAKESET($\{v\}$)
4. **end for**
5. $T = \{\}$
6. **while** $|T| < n - 1$
7. Let (x, y) be the next edge in E .
8. **if** FIND(x) \neq FIND(y) **then**
9. Add (x, y) to T
10. UNION(x, y)
11. **end if**
12. **end while**

Primův algoritmus

- začít od nejlevnější hrany
- postupně „přilepujeme“ další sousedící hrany
- datová struktura: prioritní fronta

Primův algoritmus



Primův algoritmus

Algorithm 8.4 PRIM

Input: A weighted connected undirected graph $G = (V, E)$, where $V = \{1, 2, \dots, n\}$.

Output: The set of edges T of a minimum cost spanning tree for G .

1. $T \leftarrow \{\}; \quad X \leftarrow \{1\}; \quad Y \leftarrow V - \{1\}$
2. **for** $y \leftarrow 2$ **to** n
3. **if** y adjacent to 1 **then**
4. $N[y] \leftarrow 1$
5. $C[y] \leftarrow c[1, y]$
6. **else** $C[y] \leftarrow \infty$
7. **end if**
8. **end for**
9. **for** $j \leftarrow 1$ **to** $n - 1$ {find $n - 1$ edges}
10. Let $y \in Y$ be such that $C[y]$ is minimum
11. $T \leftarrow T \cup \{(y, N[y])\}$ {add edge $(y, N[y])$ to T }
12. $X \leftarrow X \cup \{y\}$ {add vertex y to X }
13. $Y \leftarrow Y - \{y\}$ {delete vertex y from Y }
14. **for** each vertex $w \in Y$ that is adjacent to y
15. **if** $c[y, w] < C[w]$ **then**
16. $N[w] \leftarrow y$
17. $C[w] \leftarrow c[y, w]$
18. **end if**
19. **end for**
20. **end for**

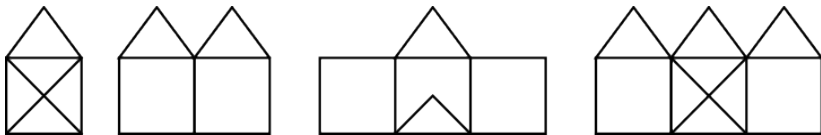
Nečekaná aplikace: generování bludiště

- další způsob jak generovat bludiště
- budujeme „kostru“ – bouráme zdi
- Kruskal, Prim – každý trochu jiná bludiště

Toky v sítích

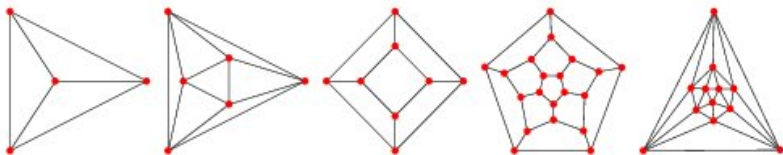
- tok v síti: např. voda, elektřina, energie (potravní řetězec)
- uzly: zdroj, dřez
- hrany: maximální kapacita
- podmínky:
 - zachování kapacity
 - konzistence uzlů: co přiteče, to odeče
- problém: hledání maximálního toku
- řešení: postupné „fecování“ toku

Eulerovský cyklus



- eulerovský cyklus = navštívit všechny **hrany** právě jednou
- problém 1: existuje eulerovský cyklus?
- problém 2: vypsát cyklus
- oboje jednoduše řešitelné

Hamiltonovský cyklus



- hamiltonovský cyklus = navštívit všechny **uzly** právě jednou
- problém obchodního cestujícího – vážené hrany
- obtížné (NP-úplné), hrubá síla, heuristiky

Shrnutí

- graf, reprezentace
- prohledávání do hloubky, do šířky
- aplikace prohledávání
- složitější algoritmy: cesty, kostry, toky, cykly, ...