První část:

- **Přesně definujte jednoduchý neorientovaný graf.**
  - Uspořádaná dvojice vrcholů a hran (G=(V,E), V je množina vrcholů a E je množina hran)
- **Definujte kostru s minimálním hrdlem pro jednoduchý souvislý neorientovaný graf s ohodnocením hran. (nejprve definujte kostru a hrdlo)**
  - Kostra je takový podgraf souvislého grafu G na množině všech jeho vrcholů, který je stromem
  - Hrdlem kostry je hrana s maximální váhou
  - Kostra s minimálním hrdlem je kostra s minimální cenou hrdla
- **Co je chromatické číslo grafu?**
  - Chromatické číslo je minimální počet barev, které jsou potřebné pro vybarvení daného grafu(Č, Z, M = 3)
- **Uveďte jaký je rozdíl mezi řešením problému a verifikací řešení. Uveďte příklad NP- úplné grafové úlohy a demonstrujte na ní co je její řešení a co je verifikace.**
  - Řešení problému(P) – symetrický postup, který většinou předchází konečné tezi
  - Verifikace (NP) – je ověřování, kontrola pravdivosti výroku, hypotézy
  - Příklad NP-úplné graf. úlohy: Problém obchodního cestujícího(hamiltonova kružnice)
- **Definujte třídu NP-úplných problémů. Co je pro praxi podstatnou vlastností NP-úplných problémů?**
  - Třída NP úplných úloh tvoří v jistém smyslu ty nejtěžší úlohy z NP
  - Vlastnost: je vypočitatelná v nedeterministickým polynomiálním čase
- **Definujte orientovaný acyklický graf (DAG) a uveďte pro jakou úlohu se (například) používá.**
  - Je takový graf, který neobsahuje žádný cyklus. Používá se například jako projekt(s metodou PERT pro výpočet kritických cest)
- **Definujte isomorfismus grafu G1 = {V1, E1}, G2 = {V2, E2}. Uveďte příklad dvojice různých isomorfních a dvojice neisomorfní grafů se čtyřmi vrcholy a čtyřmi hranami.**
  - Grafy G1 a G2 jsou isomorfní právě tehdy, když existuje takové= zobrazení f: V(G) -> V(G')
- **Co je řídký graf? Jak jej efektivně reprezentujeme v počítači?**
  - Je graf, který má ostře menší počet hran než n^2, kde n je počet vrcholů. Pomocí matice (seznamem) sousedů

- **Definujte vzdálenostní metriku d_G (u, v) prostého neorientovaného grafu.**
  - Pro dva vrcholy x, y grafu G definujeme jejich vzdálenost d_G(u,v) jako délku nejkratší cesty v grafu

- **Jaký je v teorii her rozdíl mezi ryzí a smíšenou strategií?**
    - Ve hře mají oba hráči pouze jeden tah, má každý hráč tolik různých strategií, kolik má různých možností pro provedení tohoto tahu. Tyto strategie nazýváme ryzí.
    - Smíšená strategie je takové strategické rozhodnutí inteligentních hráčů, která nemůže protivník předem vykalkulovat, neboť rozhodnutí se provádí pomocí náhodného, nikoliv však libovolného mechanismu, na množině ryzích strategií
- **Definujte síť a tok sítí.**
    - Síť je uspořádaná čtveřice **f=((V,E), c, s, t),** kde **V** je množina vrcholů, **E** je množina hran, **c** je kapacita, **s** je vstupní vrchol a **t** je výstupní vrhcol
- **Definujte sled a cestu v grafu G = {V, E} z vrcholu A do vrcholu B. • Přesně definujte jednoduchý neorientovaný graf.**
    - Sled je množina vrcholů a hran, kterými sled prochází
    - Cesta je sled ve kterém se neopakují vrcholy(tudíž tím pádem i hrany)
- **Definujte co je stupeň vrcholu.**
    - Stupeň vrcholu závisí jestli je graf G orientovanýá, či neorientovaný a je to počet hran vázaných na vrchol
    - Orientovaný: je indegree(kolik hran jde do vrcholu) a outdegree(kolik z něj)
    - Neorientovaný: počet hran vázaných na vrchol
- **Co je to silně souvislá komponenta, pro jaký typ grafu má smysl?**
    - Je takový maximální podgraf orientovaného grafu, v němž pro všechny dvojice vrcholů **u,v** existuje cesta z **u** do **v** a zároveň z **v** do **u**
- **Dělení her:** sekvenční a strategické, (ne)nulový součet, (ne)kooperativní
- **Smíšená strategie:** kámen N P
- **Nezávislá množina –** je podM vrcholů všech vrcholů, kde žádné dva nejsou spojeny hranou Reziduální graf- neobsahuje zlepšující cestu
- **Min. kostra -** kostra s minimální cenou
- **Relace R** je relace ekvivalence právě tehdy, když je tranzitivní, reflexivní a symetrická.
- **Binární halda –** je kořenový strom, každý uzel má 2 potomky, všechny vrstvy stromu jsou zaplněné, kromě poslední, každý uzel je větší než jeho potomci
- **Nashovo ekvilibrium** – je stav, ve kterém se nikomu nevyplatí sám o sobě změnit svou strategii, přičemž ostatní drží tu svou (je znám jejich tah). Jedná se o sedlový bod.
- **FIND -** najdi podmnožinu ve které je prvek x
- **UNION-** sjednoť podmnožiny M(x) a M(y)
- **Matice sousednosti –** je čtvercová, zobrazuje počet hran mezi jednotlivými vrcholy
- **Incidenční matice -** představuje matici, která obsahuje informace o ohodnocením jednotlivých hranách (nemusí být čtvercová)

Druhá část

1. Dokazte, že graf je strom právě tehdy pokud pro graf platí $|V| = |E| + 1$

Definice:

- **Strom** je souvislý acyklický graf.
- |V| je počet vrcholů (vertices).
- |E| je počet hran (edges).

Důkaz:

a) Strom ⇒ |V| = |E| + 1

1. Nechť T je strom s n vrcholy a m hranami.
2. Vzhledem k tomu, že strom je souvislý a acyklický, přidání každé hrany k n-1 vrcholům vytvoří právě n-1 hran:
    - Začněme s jedním vrcholem (není žádná hrana).
    - Přidáním každého nového vrcholu přidáme právě jednu hranu, aby zůstal souvislý (aby neexistovaly žádné cykly).
3. Pokud máme n vrcholů, počet přidaných hran je n-1:
    - Počet hran m = n - 1.
4. Proto pro strom platí |V| = |E| + 1.

b) |V| = |E| + 1 ⇒ Strom

1. Nechť G je souvislý graf s n vrcholy a m hranami, kde m = n - 1.
2. Předpokládejme, že G není strom. Pak G obsahuje cyklus.
3. Pokud odstraníme libovolnou hranu z cyklu, graf G zůstane souvislý, ale počet hran se sníží o 1:
    - Počet vrcholů se nemění, zůstává n.
    - Počet hran je nyní m - 1.
4. Opakováním tohoto postupu odstraníme všechny cykly, dokud nezůstane souvislý acyklický graf:
    - Nakonec dostaneme souvislý acyklický graf s n vrcholy a n-1 hranami.
5. Tento graf je strom, což je v rozporu s naším předpokladem, že G není strom.
6. Proto G musí být strom.

2. Dokažte, že K3,3 není rovinný graf

Definice:

- **Rovinný graf** je graf, který může být nakreslen v rovině bez překrývání hran (kromě jejich koncových bodů).
- **Graf K3,3** je bipartitní úplný graf s dvěma množinami po třech vrcholech.

Důkaz:

1. Eulerova věta pro rovinné grafy: V - E + F = 2, kde V je počet vrcholů, E je počet hran a F je počet stěn.
2. Pro K3,3 máme:
    - V = 6 (3 + 3 vrcholů).
    - E = 9 (každý vrchol z první množiny je spojen s každým vrcholem z druhé množiny).
3. Předpokládejme, že K3,3 je rovinný graf.
4. Dosadíme do Eulerovy věty:
    - 6 - 9 + F = 2.
    - F = 5.
5. Počet hran v rovinném grafu je maximálně 3 * (V - 2) = 3 * 4 = 12. Pro graf bez trojúhelníků:
    - E ≤ 2 * (V - 2) = 2 * 4 = 8.
6. K3,3 má 9 hran, což je více než 8, což je maximální počet hran pro rovinný graf bez trojúhelníků.
7. Proto K3,3 není rovinný graf.

3. Vlastnosti relace ekvivalence a důkaz

Vlastnosti relace ekvivalence:

- **Reflexivní**: $\forall x \in A, x \sim x$.
- **Symetrická**: $\forall x, y \in A$, pokud $x \sim y$, pak $y \sim x$.
- **Tranzitivní**: $\forall x, y, z \in A$, pokud $x \sim y$ a $y \sim z$, pak $x \sim z$.

Důkaz, že relace je ekvivalence:

1. Reflexivita:
    - Každý vrchol má cestu k sobě samému (nulová cesta), tedy $x \sim x$.
2. Symetrie:
    - Pokud existuje cesta z x do y, pak existuje i cesta z y do x (v neorientovaném grafu).
3. Tranzitivita:
    - Pokud existuje cesta z x do y a z y do z, pak existuje cesta z x do z (spojením těchto cest).

Vlastnost, která neplatí pro orientovaný graf:

- Symetrie: V orientovaném grafu nemusí cesta z x do y implikovat cestu z y do x.

Třídy ekvivalence:

- Odpovídají souvislým komponentám grafu. Každá třída ekvivalence zahrnuje vrcholy, mezi nimiž existují cesty.
-

4. Definice stromu a důkaz jedinečné cesty

Definice:

- **Strom** je souvislý acyklický graf.

Důkaz:

1. Nechť T je strom.
2. Předpokládejme, že existují dvě různé cesty mezi vrcholy x a y.
3. Tyto dvě různé cesty tvoří cyklus, což je v rozporu s definicí stromu jako acyklického grafu.
4. Proto v stromu mezi každými dvěma vrcholy existuje právě jedna cesta.

5. Důkaz O(nlogn) pro setřídění

Důkaz:

1. Každý srovnávací třídicí algoritmus lze považovat za rozhodovací strom, kde každý vnitřní uzel představuje srovnání dvou prvků.
2. Hloubka rozhodovacího stromu pro n prvků je minimálně log2(n!), protože každá permutace prvků musí být dosažitelná.
3. log2(n!) ~ nlogn (Stirlingova aproximace).
4. Tedy jakýkoliv srovnávací třídicí algoritmus musí provést minimálně O(nlogn) srovnání.

6. Matice sousednosti a incidenční matice

Definice:

- **Matice sousednosti**: čtvercová matice, kde element A[i][j] = 1, pokud existuje hrana mezi vrcholy i a j, jinak 0.
- **Incidenční matice**: matice, kde řádky reprezentují vrcholy a sloupce hrany, element B[i][j] = 1, pokud je vrchol i incidentní s hranou j, jinak 0.

Matice pro graf:

Pro graf s vrcholy a, b, c, d:

Matice sousednosti:

```
  a b c d
a 0 1 0 1
b 1 0 1 0
c 0 1 0 1
d 1 0 1 0

Incidenční matice:

 e1 e2 e3 e4
a 1 0 1 0
b 1 1 0 0
c 0 1 0 1
d 0 0 1 1
```

Vztah:

- Diagonální matice stupňů vrcholů D: pro každý vrchol, D[i][i] je stupeň vrcholu i.
- A=D-BB^T, kde A je matice sousednosti a B incidenční matice.

Třetí část

1. Dijkstrův algoritmus

Zadání úlohy:

- **Graf**: Neorientovaný graf s ne zápornými ohodnocenými hranami.
- **Ohodnocení**: Každá hrana e=(u,v)má přidělenou váhu w(u,v)≥0
- **Počátek**: Počáteční vrchol s.

Pseudokód Dijkstrova algoritmu:

Vstupní data:

- Graf G=(V,E)G = (V, E)G=(V,E)
- Váhová funkce w : E→R+
- Počáteční vrchol s

Výstup:

- Délky nejkratších cest z vrcholu s ke všem ostatním vrcholům.
- Předchůdci jednotlivých vrcholů ve stromu nejkratších cest.

Algoritmus:

```
function Dijkstra(G, w, s):
  for each vertex v in V(G):
    dist[v] := ∞
    prev[v] := null
  dist[s] := 0
  Q := prioritní fronta všech vrcholů ve V(G) s hodnotami dist jako klíče


  while Q is not empty:
    u := extract_min(Q)
    for each neighbor v of u:
      alt := dist[u] + w(u, v)
      if alt < dist[v]:
        dist[v] := alt
        prev[v] := u
        decrease_key(Q, v, alt)
```

```
    return dist, prev
```

Prioritní fronta:

- **Definice**: Datová struktura, která umožňuje rychlý přístup k prvku s nejmenší hodnotou (klíčem) a efektivní změnu klíčů.
- **Operace potřebné pro algoritmus**:
    - `extract_min(Q)`: Odebere a vrátí prvek s nejmenší hodnotou klíče.
    - `decrease_key(Q, v, alt)`: Sníží hodnotu klíče pro prvek *v* na hodnotu *alt*

2. Silně souvislé komponenty orientovaného grafu

Definice:

- **Silně souvislé komponenty**: Podmnožina vrcholů C v orientovaném grafu, kde pro každý pár vrcholů u,v ∈ C existuje cesta z **u** do **v** a z **v** do **u**

Algoritmus Tarjan:

Vstup:

- Graf G=(V,E) reprezentovaný řídkou maticí sousednosti (seznamem sousedů).

Výstup:

- Očíslování vrcholů čísly komponent.

```
index := 0
S := empty stack
components := []

function strongconnect(v):
    v.index := index
    v.lowlink := index
    index := index + 1
    S.push(v)
    v.onStack := true

    for each neighbor w of v:
        if w.index is undefined:
            strongconnect(w)
            v.lowlink := min(v.lowlink, w.lowlink)
        else if w.onStack:
            v.lowlink := min(v.lowlink, w.index)
```

```
    if v.lowlink == v.index:
        component := []
        repeat
            w := S.pop()
            w.onStack := false
            component.append(w)
        until w == v
        components.append(component)

function tarjan(G):
    for each vertex v in V(G):
        if v.index is undefined:
            strongconnect(v)
    return components
```

3+5. Kruskalův algoritmus

Pseudokód:

Vstup:

- Graf G=(V,E) s ohodnocenými hranami.

Výstup:

- Minimální kostra grafu (MST).

```
function Kruskal(G):
    A := ∅
    foreach vertex v in V(G):
        make_set(v)
    edges := E(G)
    sort edges by weight
    foreach (u, v) in edges:
        if find_set(u) ≠ find_set(v):
            A.add((u, v))
            union(u, v)
    return A
```

Podpůrné algoritmy a datové struktury:

- Union-Find struktura:
    - `make_set(x)`: Vytvoří novou množinu obsahující prvek x.

- `find_set(x)`: Najde reprezentanta množiny, která obsahuje x.
- `union(x, y)`: Spojí dvě množiny obsahující x a y.

4+6. Primův-Jarníkův algoritmus

Pseudokód:

Vstup:

- Graf G=(V,E) s ohodnocenými hranami.
- Počáteční vrchol s.

Výstup:

- Minimální kostra grafu (MST).

```
function Prim(G, w, s):
   for each vertex u in V(G):
      key[u] := ∞
      parent[u] := null
   key[s] := 0
   Q := priority queue of all vertices in V(G) with key values as keys

   while Q is not empty:
      u := extract_min(Q)
      for each neighbor v of u:
         if v in Q and w(u, v) < key[v]:
            parent[v] := u
            key[v] := w(u, v)
            decrease_key(Q, v, w(u, v))

   return parent
```

Prioritní fronta:

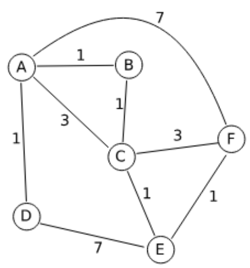
- Používá se k výběru vrcholu s nejmenším klíčem.
- Operace:
  - `extract_min(Q)`: Odebere a vrátí prvek s nejmenším klíčem.
  - `decrease_key(Q, v, new_key)`: Sníží hodnotu klíče pro vrchol v na novou hodnotu newkey

7. Ford-Fulkersonův + Edmonds-Karp algoritmus

- Opakovaně hledá zlepšující cesty v reziduálním grafu.
- Aktualizuje tok podél těchto cest.
- Reziduální kapacita:
- Rozdíl mezi kapacitou hrany a aktuálním tokem.
- Reziduální graf:
- Graf odrážející možné zlepšující cesty s aktuálním tokem.
- Zlepšující cesta:
- Cesta v reziduálním grafu umožňující zvýšení toku z s do t
- Edmonds-Karp algoritmus:
- Implementuje Ford-Fulkersona pomocí BFS pro hledání zlepšujících cest.
- Zaručuje časovou složitost O(VE^2)

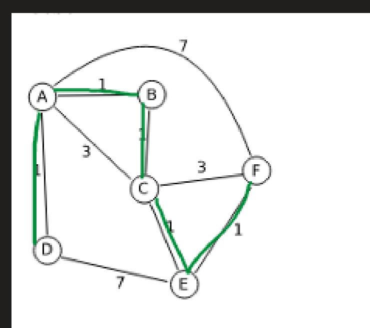
Čtvrtá část

Demonstrujte průběh Dijkstrova algoritmu při hledání nejkratších cest z vrcholu D nakresleného grafu. Průběh algoritmu zapište do tabulky, kde jeden řádek tabulky bude obsahovat zvlášť vrcholy nenavštívené, zvlášť vrcholy ve frontě a zvlášť vrcholy s definitivní nejkratší cestou. U každého vrcholu bude aktuální délka cesty. Do grafu zakreslete strom nalezených nejkratších cest.



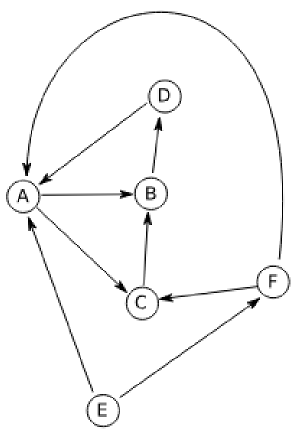
1)

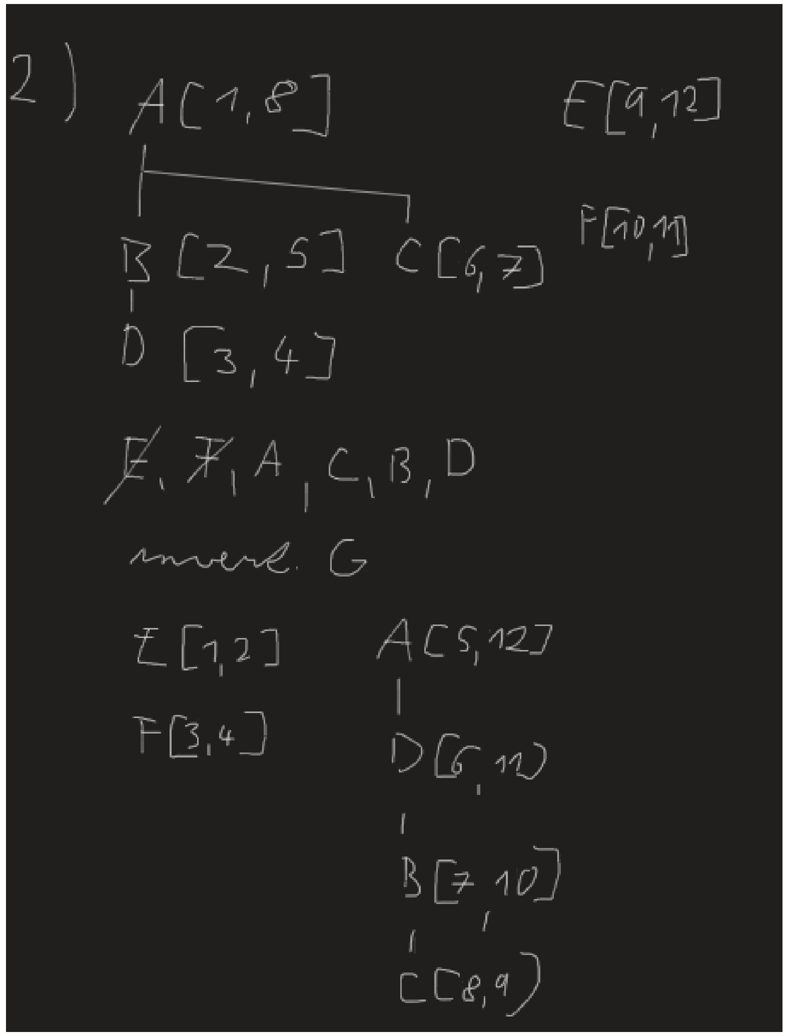
	0	1	2	3	4	5	6
A		D(1)					D(1)
B			A(2)				A(2)
C			A(4)	B(3)			B(3)
D	D(0)						D(0)
E		D(7)	D(7)	D(7)	C(5)		C(5)
F			A(8)	A(8)	C(7)	E(6)	E(6)



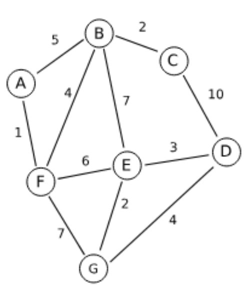
	Q	Cl	P
0	D	—	D
1)	A, E	D	A
2)	E, F, B, C	D, A	B
3)	E, F, C	D, A, B	C
4)	E, F	DABC	E
5)	F	DABCE	F
6)	—	DABCEF	—

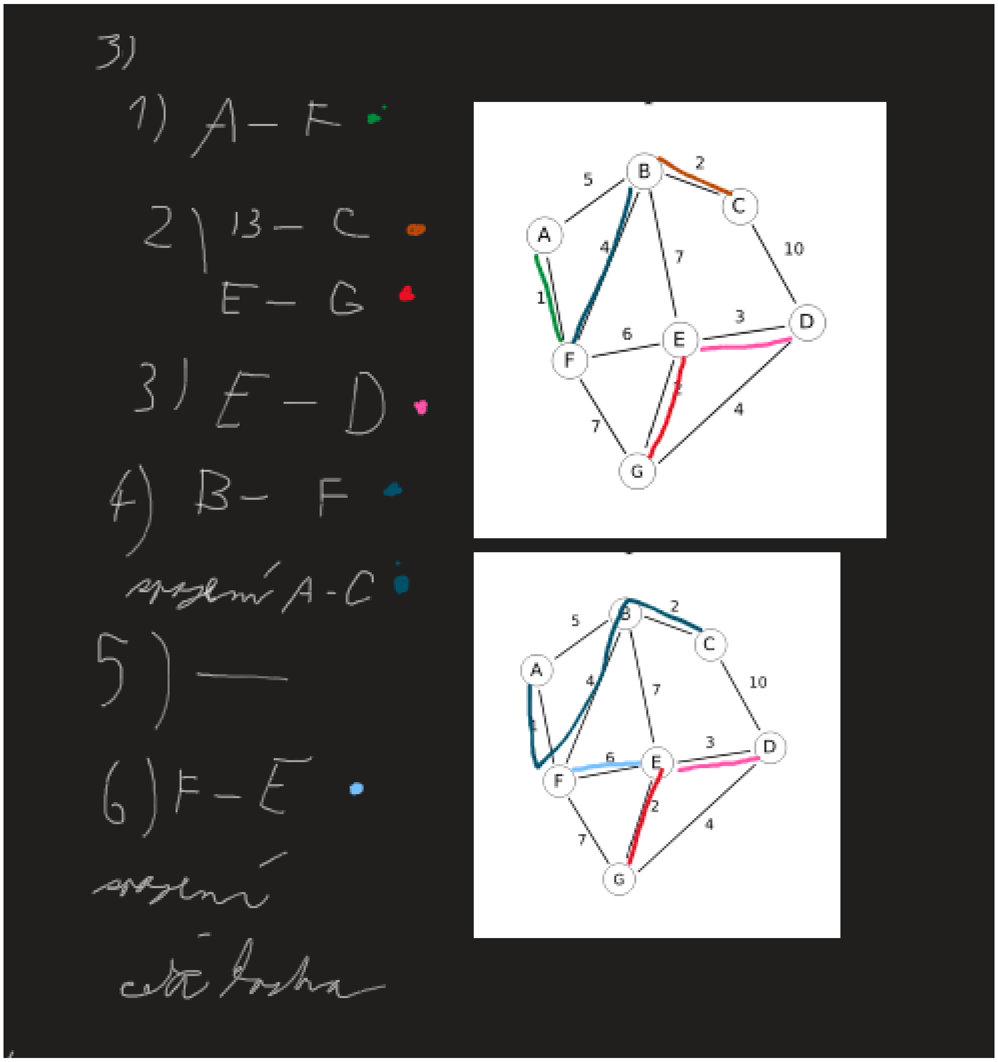
Demonstrujte průběh použitého algoritmu na grafu nakreslením lesů prohledávání a časy průchodů. Vstupní uspořádání vrcholů i uspořádání sousedů každého vrcholu je podle abecedy.



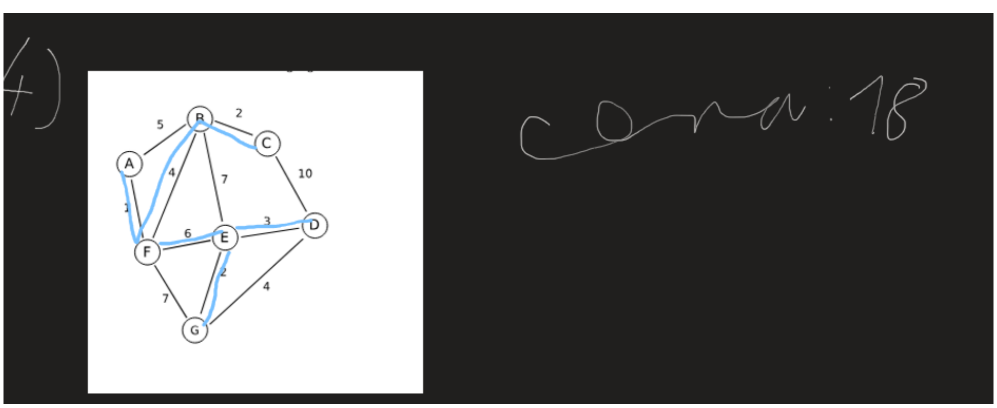


Demonstrujte průběh Kruskalova algoritmu na uvedeném grafu. Zapište pořadí přidávaných operací FIND a UNION.

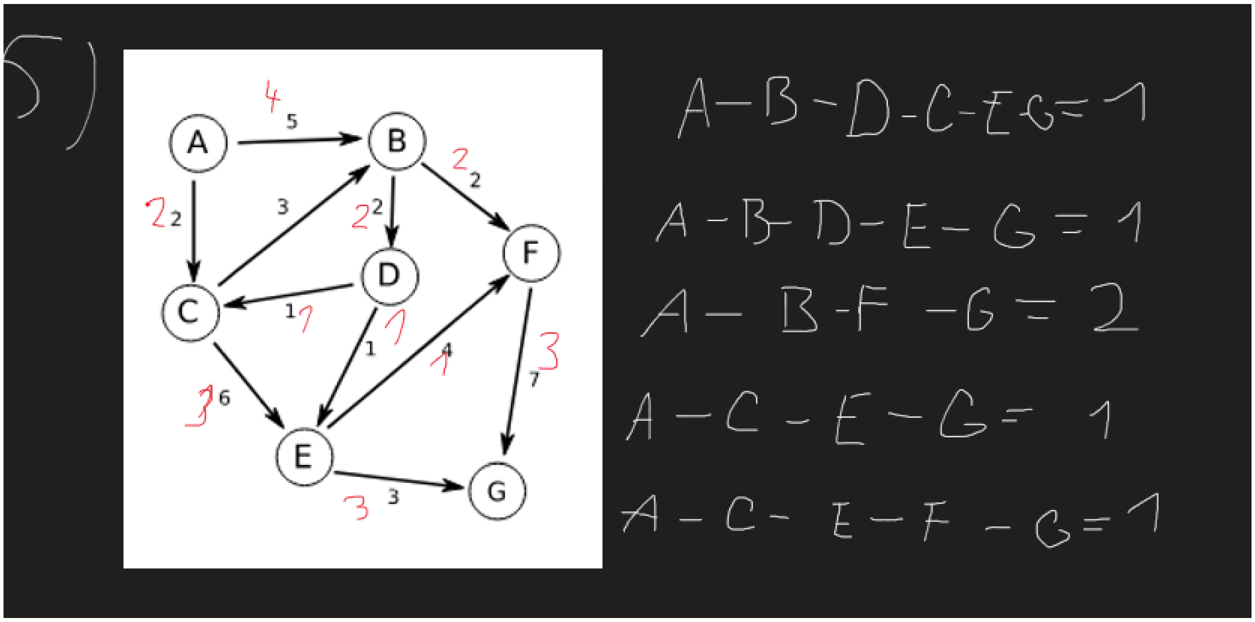




Demonstrujte průběh Primova-Jarníkova algoritmu na uvedeném grafu. Začněte ve vrcholu A. Pro každý krok algoritmu zapište hrany ve frontě a přidávanou hranu. Zakreslete výslednou kostru a určete její cenu.



Na zadané síti demonstrujte průběh algoritmu s Edmons-Karpovským hledáním zlepšující cesty. Uveďte postupné zlepšující cesty nalezené E-K algoritmem.



Příklady 5:

Definujte Nashovo ekvilibrium, tj. dvojici optimálních (smíšených) strategií, pro maticové hry a formulujte základní větu o maticovch hrách.

Nashovo ekvilibrium je dvojice strategií (jedna pro každého hráče) (s_1^*, s_2^*), kde s_1^* je optimální strategie pro hráče 1 a s_2^* je optimální strategie pro hráče 2. Tyto strategie splňují následující podmínku:

- Hráč 1 nemůže zvýšit svůj očekávaný zisk změnou své strategie, pokud hráč 2 zůstane u strategie s_2^*.
- Hráč 2 nemůže zvýšit svůj očekávaný zisk změnou své strategie, pokud hráč 1 zůstane u strategie s_1^*.

Popište datovou strukturu pro rozklad na disjunktní podmnožiny (komponenty) a její operace FIND a UNION. Uveďte implementaci těchto operací pomocí stromů. Jaké dvě strategie se používají pro výrazné zlepšení amortizované složitosti?

Union-Find (nebo také Disjoint-Set). Je to způsob organizace prvků do disjunktních (nepřekrývajících se) množin. Hlavní operace jsou FIND a UNION.

FIND (Find-Set):
Tato operace slouží k nalezení reprezentanta (kořene) dané množiny, ke které prvek patří. Implementace pomocí stromů: Každý prvek je vrchol stromu, kde kořen stromu reprezentuje množinu. Pro vyhledání reprezentanta postupujeme od daného prvku směrem ke kořeni stromu. Složitost: O(α(n)), kde α(n) je inverzní Ackermannova funkce, což je velmi pomalá rostoucí funkce.

UNION (Union-Set):
Tato operace spojuje dvě množiny do jedné. Implementace pomocí stromů: Spojujeme dva stromy tak, že nastavíme kořen jednoho stromu jako potomka kořene druhého stromu.
Složitost: O(α(n)).

Dvě strategie pro zlepšení amortizované složitosti:
Union by rank (Spojování podle ranku):
Udržujeme informaci o hloubce stromu (ranku). Při spojení dvou stromů připojíme menší strom pod větší strom.
To snižuje hloubku stromů a zlepšuje složitost operací.
Složitost: O(α(n)).

Path compression (Komprese cesty):
Při vyhledávání reprezentanta (FIND) upravíme cestu od daného prvku k reprezentantovi tak, aby byla co nejkratší (směřovala přímo k reprezentantovi).
To zkracuje cesty ve stromech a zlepšuje složitost operací.
Složitost: $O(\alpha(n))$.

Uveďte třídící algoritmus s průměrnou časovou složitostí $O(n^2)$ a algoritmus se složitostí $O(n \log n)$. Jakou časovou složitost má algoritmus průchodu grafem do hloubky v závislosti na počtu vrcholů V| a počtu h-ran |E| grafu? Jakou průměrnou časovou složitost (při náhodném obsahu pole) má následující funkce v závislosti na velikosti pole N:

Průměrná časová složitost O(n^2): Bubblesort je příkladem třídícího algoritmu s takovou složitostí. Bubblesort iterativně prochází pole a prohazuje sousední prvky, pokud nejsou ve správném pořadí. Je to jednoduchý, ale pomalý algoritmus.

Složitost O(n log n): Heapsort nebo Mergesort jsou příklady třídících algoritmů s tímto asymptotickým časem. Heapsort vytváří minimální binární haldu a postupně odebírá minimum. Mergesort rozdělí pole na menší části, setřídí je a poté je spojí.

Průchod grafem do hloubky (DFS):
Časová složitost DFS závisí na počtu vrcholů |V| a počtu hran |E| grafu. V nejhorším případě (když projdeme celý graf), je složitost $O(|V| + |E|)$. DFS prochází graf rekurzivně nebo pomocí zásobníku a prochází všechny dostupné cesty do maximální hloubky.

Funkce:

Tato funkce prochází pole pole délky N. Pokud nalezne prvek v poli roven „false“ tak jej nastaví na true a breakne loop. Pokud narazí na „true“ nastaví jej na false. Časová složitost je N.

Popište datovou strukturu minimální binarní halda. Jaké jsou její tři vlastnosti? Jak lze n prvkovou haldu uložit v n prvkovém poli? Uveďte příklad algoritmů, které používají haldu.

Minimální binární halda:
Haldová vlastnost:
Hodnota každého uzlu je menší nebo rovna hodnotě jeho rodiče. To znamená, že kořen haldy obsahuje nejmenší prvek.

Struktura:
Binární halda je úplný binární strom, kde uzly jsou uloženy zleva doprava na jednotlivých úrovních. Pokud indexujeme prvky haldy (od čísla 1), potomci každého vrcholu jsou na indexech 2i a 2i+1.

Ukládání v poli:
N-prvkovou haldu lze uložit v poli o velikosti N. Prvky pole indexujeme od 1 (nikoli od 0),

abychom zachovali vztahy mezi rodiči a potomky. Pro prvek na indexu i: jeho levý potomek je na indexu 2i a pravý potomek na indexu 2i+1.

Příklady algoritmů, které používají haldu:
Heapsort: Řadí prvky v poli pomocí min-haldy.
Dijkstrův algoritmus: Hledá nejkratší cesty v grafu.
Primův algoritmus: Najde minimální kostru grafu.