Dugga i Objektorienterad modellering och design (OMD) Helsingborg – grafalgoritmer

EDAF25

23 mars 2023, 15.15 – 17.00

- Skriv ditt namn och personnummer på avsedd plats nedan på denna sida, men inte på några andra sidor
- Ha legitimation i beredskap för id-kontroll
- Skriv i första hand dina svar på frågorna på avsedd plats i detta häfte
- Skriv bara på framsidan av arken
- Behöver du mer plats kan du använda det blanka arket i slutet av häftet eller, om det inte räcker, extra tomma ark
- SKRIV TYDLIGT – om texten inte går att läsa kan du inte få några poäng.

Inlämning

- På baksidan av varje ark i detta häfte finns en anonymkod stämplad
- Om du lämnar in extra ark så skriv din anonymkod på baksidan av varje extra ark
- Om du lämnar in extra ark, eller har tagit isär arken i häftet, så använd en av häftapparaterna som finns tillgängliga för att häfta ihop arken
- Ha legitimation i beredskap om id-kontroll inte redan hunnit utföras
- Lämna svarshäftet till Roger som kommer att riva av första sidan med namn och personnummer i svarshäftet och förvara separat tills dess alla duggor är rättade

---

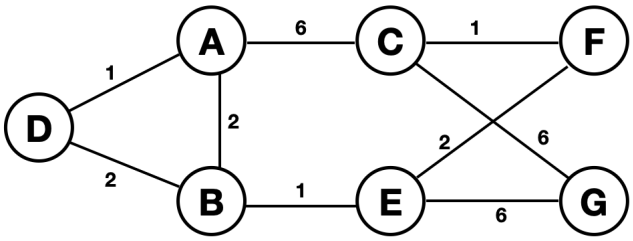
Hjälpmedel: • Java snabbreferens

Namn:

Personnummer:

Uppgift 1

Nedanstående figur visar en graf med tillhörande avstånd (kostnader) för bågarna. Vi är intresserade av att hitta det kortaste avståndet från noden **B** till var och en av de övriga noderna.



a) Grafen kan med fördel representeras med hjälp av en grannmatris. Visa hur det kan göras genom att fylla i nedanstående matris (om vi t.ex. låter värdet 0 betyda att en väg inte existerar kan du lämna dessa rutor tomma). *(1p)*

	A	B	C	D	E	F	G
A							
B							
C							
D							
E							
F							
G							

b) I kursen har vi även gått igenom ett annat sätt att representera grafen i våra program. Förklara med en eller några få meningar hur detta går till. *(1p)*

Skriv ditt svar här. Behöver du mer plats så fortsätt på sista arket i häftet.

---

c) Antag att vi väljer att använda Dijkstras algoritm för att beräkna avstånden. I vilken ordning kommer algoritmen i så fall att fastställa det slutliga avståndet till noderna? Svara med att ange noderna i den ordning algoritmen bestämmer deras avstånd från noden **B**. *(2p)*

Skriv ditt svar här.

---

d) Även Bellman-Fords algoritm kan användas för att beräkna de kortaste avstånden i grafen. Nämn en användbar egenskap som Bellman-Fords algoritm har framför Dijkstras algoritm i det generella fallet. *(1p)*

Skriv ditt svar här.

Uppgift 2

Hjälp! Roger har försökt lösa uppgiften i den sista laborationen (den om max-flöde) i kursen, men får inte riktigt sin kod att fungera. Nedanstående metod är tänkt att använda bredden-först-sökning för att hitta den kortaste (eller en av de kortaste) vägen (vägarna) från noden `source` till noden `sink` för vilken det finns en kvarvarande (positiv) kapacitet. Efter att ha matat in koden i det stora datororaklet ChatGPT och frågat var felet är har han fått veta att det finns två helt oberoende logiska fel i metoden. Eftersom Roger bara har råd med gratisversionen av ChatGPT har han dock inte fått veta exakt vad felen består i uta bara att de existerar.

- Noderna i grafen representeras som heltal med start på 0 (0..n-1).
- Parametern `residual` anger de kvarvarande kapaciteterna för bågarna mellan noderna i grafen. Om det t.ex. finns en kvarvarande kapacitet på 5 enheter från nod 1 till nod 6 så är `residual[1][6]` lika med 5.
- Parametern `path` är en vid anropet en tom lista som ska fyllas i med numren på de noder som utgör vägen från `source` till `sink`. Om t.ex. den kortaste vägen från nod 1 till nod 8 går via nod 4 så ska `path` efter anropet innehålla [1, 4, 8] och metoden returnera `true`. Om ingen väg hittas returneras `false` och `path` blir tom.
- Det är verifierat att metoden anropas på korrekt sätt med korrekt indata, men den returnerar ändå felaktiga resultat (inkorrekt innehåll i `path`).

Hjälp Roger genom att ange var i koden det är fel samt hur man kan ändra koden för att rätta till de två felen! Det räcker att du talar om vilken/vilka rader (radnummer) som ska tas bort och vilken kod som de ska ersättas med. Du behöver alltså inte skriva om hela metoden utan bara de delar som behöver ändras.

(5p)

Du hittar koden och plats för svar på nästa sida.

```
1  private static boolean bfs(int[][] residual,int source,
2                                     int sink,List<Integer> path) {
3      int numVertex = residual.length;
4      Set<Integer> visited = new HashSet<>();
5      int prev[] = new int[numVertex];
6      PriorityQueue<Integer> queue = new PriorityQueue<>();
7      path.clear();
8      visited.add(source);
9      queue.add(source);
10     boolean found = false;
11     while (!queue.isEmpty()) {
12       int current = queue.poll();
13       if (current==sink) {
14         found = true;
15         break;
16       }
17       for (int i = 0; i < numVertex; i++) {
18         if (!visited.contains(i) && residual[current][i]>0) {
19           visited.add(i);
20           prev[i] = current;
21           queue.add(i);
22         }
23       }
24     }
25     if (found) {
26       for(int i=0;i<numVertex;i++) {
27         path.add(prev[i]);
28       }
29     }
30     return found;
31 }
```

Skriv ditt svar nedan. Fortsätt på nästa sida om du behöver mer plats.

Fortsätt med dina svar här om du behöver.

Lösningsförslag

Uppgift 1

a)

	A	B	C	D	E	F	G
A		2	6	1			
B	2			2	1		
C	6					1	6
D	1	2					
E		1				2	6
F			1		2		
G			6		6		

b) Vi kan använda oss av närhetslistor. Vi har då en vektor som motsvarar noderna i grafen. På varje plats i vektorn lägger vi en länkad lista som innehåller de noder som är grannar, med tillhörande avstånd, till den nod som motsvaras av platsen i vektorn.

c) Det finns två möjliga svar: B, E, D, A, F, C, G eller B, E, A, D, F, C, G. Svar som utelämnar B godkänns också.

d) Dijkstras algoritm klarar inte grafer med negativa kostnader på bågarna. Det gör däremot Bellman-Fords algoritm – till kostnaden av en större tidskomplexitet.

Uppgift 2

För att garantera att vi hittar den kortaste vägen måste vi byta ut rad 6 mot följande:

```
List<Integer> queue = new LinkedList <>();
```

Lägger vi in noderna som ska besökas i en prioritetskö kommer vi att behandla dem i fel ordning (nummerordning i stället för efter hur långt från startnoden de är). För att dessutom returnera den funna (korrekta) vägen i rätt ordning måste vi byta ut rad 26-28 mot följande:

```
int current = sink;
do {
  path.add(0, current);
  current = prev[current];
} while (current!=source);
path.add(0,source);
```

Vektorn `prev` är en tabell som anger för varje nod vilken dess föregångare är. Den innehåller inte direkt den sökta vägen. För att konstruera den sökta vägen måste vi börja i sänkan och följa vägen baklänges genom grafen tills vi kommer till källan.

Båda felen är autentiska fel som förekom flera gånger under laborationerna i januari.