TÖL203G Tölvunarfræði 2

Lokapróf 7. maí 2019
Kennari: Páll Melsted kl. $9^{00} - 13^{30}$

Nafn og kennitala

- Öll skrifleg hjálpargögn eru heimil.
- Prófið er 110 stig, 100 stig gefa 10 í einkunn.
- Athugið að rökstyðja öll svör og að kannski er ekki best leysa dæmin í vaxandi röð.
- Prófið er **??** tölusettar síður. Á síðustu blaðsíðu er yfirlit yfir API fyrir klasa úr bókinni.
- Með því að skila inn þessu prófi staðfesti ég að þetta próf er unnið af mér og að ég hef hvorki veitt né fengið aðstoð sem er óleyfileg í vinnu við þetta próf. Ég skil að brot á reglum Háskóla Íslands geta haft í för með sér þung viðurlög, allt frá því að hafa áhrif á einkunn í viðkomandi námskeiði upp í brottrekstur úr skóla, tímabundið eða fyrir fullt og allt.

1. (35 stig) Fjölvalsspurningar. Hvert rétt svar gefur 5 stig. Aðeins er eitt rétt svar fyrir hverja spurningu. Ekki er dregið niður fyrir röng svör. **Dragið hring** utan um rétt svar.

   (a) Tölurnar 1-10 eru geymdar í tvíleitartré. Leitað er að tölunni 5 og gildi þeirra hnúta sem eru skoðaðir við leitina eru prentaðir út. Hvaða runa getur ekki verið prentuð út?

   (A) `10,9,8,6,5`
   (B) `4,10,5`
   (C) `2,7,3,8,4,5`
   (D) `1,2,10,4,8,5`

   (b) Reiknirit hefur tímaflækjuna $T(n) = n^2 + 2n$ fyrir öll n. Hver af eftirfarandi staðhæfingum er ekki rétt

   (A) Tímaflækjan er minni en 2^n fyrir n nógu stórt.
   (B) Tímaflækjan er minni en $C \cdot n^2$ fyrir eitthvert gildi á C
   (C) Tvöföldunarhlutfallið (Doubling ratio) stefnir á 2.
   (D) Hlutfallið á milli $T(n)$ og n^2 stefnir á 1

   (c) Látum G vera stefnt óhringað net (DAG), hvert af eftirfarandi reikniritum getur leyst reachability vandamálið, þ.e. gefið s finna alla hnúta sem hægt er að komast í frá s.

   (A) Depth First Search - DFS
   (B) Union-Find
   (C) Topological sort
   (D) Dijkstra

(d) Gefið er samanhangandi net G með n hnúta og allir hnútar hafa gráðuna d, þar sem $d \sim \sqrt{n}$ og jákvæðar vigtir w á leggjunum. Hver af eftirfarandi staðhæfingum um minnsta spannandi tré (MST) er rétt.

(A) Reiknirit Kruskals er hraðara reiknirit Prims á þessu neti.

(B) Reiknirit Prims er hraðara reiknirit Kruskals á þessu neti.

(C) Reiknirit Kruskals er jafnhraðvirkt reiknirit Prims á þessu neti.

(D) Það eru ekki nægar upplýsingar til að svara þessari spurningu.

(e) Tölurnar 1 til 9 eru settar á stafla og teknar af með blöndu af push og pop. Hver af eftirfarandi röðum getur komið fyrir þegar talan er prentuð út í hvert skipti sem kallað er á pop.

(A) `2 1 9 8 7 5 6 4 3`

(B) `2 1 4 3 5 9 8 6 7`

(C) `2 3 1 5 4 7 6 8 9`

(D) `1 2 9 8 6 7 5 4 3`

(f) Fylkið a er tekið frá í falli f með

```
public static String f(int N) {
  int[] a = new int[N];
  ...
}
```

seinna í fallinu er keyrt

```
a = null;
```

Hvert af eftirfarandi er á við hérna

(A) Minninu fyrir fylkið er skilað um leið og a er sett sem null

(B) Minninu fyrir fylkið er skilað þegar fallakallinu á f líkur

(C) Minninu fyrir fylkið er skilað þegar java sýndarvélin þarf á meira minni að halda

(D) Minninu gæti aldrei verið skilað á meðan java sýndarvélin keyrir.

(g) Hver af eftirfarandi reglulegum segðum fyrir tvíundarstrengi passar ekki við lýsinguna „Passar eingöngu við tvíundarstrengi með a.m.k. tvö 0, en ekki tvö 0 hlið við hlið".

(A) `1*01+0(1+0?)*`

(B) `1*01+01+(1+0)*`

(C) `(1|01)*011*0(1|10)*`

(D) `1*011*(011*)01*`

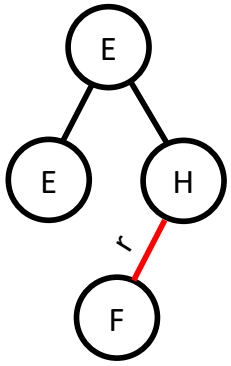
2. (5 stig) Nemandi fær í hendurnar fall sem raðar hlutum í rétta röð, keyrir á inntaki og framkvæmir tímamælingar. Í lýsingu kemur fram að „Fallið raðar fylkjum í slembiröð á tíma sem er $O(n \log n)$, aðferðin er stöðug en kastar StackOverflow villu þegar fallið er keyrt á fylki sem er raðað í vaxandi röð." Allt af þessu er rétt en nemandinn dregur þá ályktun að röðunin hljóti að vera Mergesort. Útskýrið af hverju Mergesort getur ekki kastað StackOverflow villu.

3. (8 stig) Forrit var tímamælt með eftirfarandi stærðum á inntaki N.

N	tími (sekúndur)
250	0.1
500	0.1
1000	0.2
2000	1.4
4000	10.1
8000	79.5
16000	641.0

Hvaða tímaflækju hefur forritið sem fall af N?

4. (10 stig) Eftirfarandi mynd sýnir vinstri hallandi rautt-svart tré. Lýsið því hvað gerist þegar hnútur með gildið G er settur inn í tréð með því að taka fram þau skref sem eru framkvæmd með tilliti til snúninga og breytinga á litum.



5. (12 stig) Gefið er eftirfarandi fall tekur inn streng s og tölu N.

```
public static String f(String s, int N) {
  for (int i = 1; i <= N; i *= 2) {
    s = s + s;
  }
  return r;
}
```

(a) (6 stig) Hver er tímaflækjan sem fall af N og lengdinni á strengnum s. Rökstyðjið svarið vandlega og sýnið útreikinga.

(b) (4 stig) Hvert er gildið á strengnum s eftir að eftirfarandi forritsbútur hefur verið keyrður

```
String s = "a";
f(s,4);
```

(c) (2 stig) Hver er minnisnotkunin sem fall af N?

6. (10 stig) Í tvíleitartré er eftirfarandi Node klasi notaður

```
private class Node {
  private Key key;
  private Value val;
  private Node left, right;
}
```

Skrifið fall splitTree sem tekur inn tvíleitartré T með rót x, gildi p og skiptir trénu T upp í tvö tvíleitartré T_1 og T_2 þannig að allir lyklar í T_1 eru minni en p og allir lyklar í T_2 eru stærri en eða jafnir p. Skilagildið á að vera rótin á trénu sem var klippt af. Að öðru leyti eiga T_1 og T_2 að vera eins og T.

```
private Node splitTree(Node x, Key p) {
```

7. (10 stig) Í þessu verkefni á að breyta LinearProbingHashST þannig að það noti ekki eina heldur tvær töflur. Fyrst er reynt að setja í töflu 1 (keys1,vals1), ef linear probing fer meira en k skref áfram frá fyrstu staðsetningu þá er sett í töflu 2 (keys2,vals2). Klárið að forrita get og put aðferðirnar miðað við þessa lýsingu. Þið megið gera ráð fyrir að val sé aldrei null og að aldrei þurfi að stækka töflurnar.

```
public class LinearProbingHashST<Key, Value> {
    private int n;
    private int m;
    private int k;
    private Key[] keys1, keys2;
    private Value[] vals1, vals2;

    public LinearProbingHashST(int capacity, int limit) {
       k = limit;
       m = capacity;
       n = 0;
       keys1 = (Key[])   new Object[m];
       keys2 = (Key[])   new Object[m];
       vals1 = (Value[]) new Object[m];
       vals2 = (Value[]) new Object[m];
   }

   private int hash(Key key) {
        return (key.hashCode() & 0x7fffffff) % m;
   }

   public Value get(Key key) {
     ...
   }

   public void put(Key key, Value val) {
     ...
   }
```

8. (7 stig) Skrifið fall sem tekur inn stefnt net G og telur fjölda þríhyrninga í netinu. Þríhyrningur er þrennd af hnútum $\{u, v, w\}$ þannig að allir leggirnir á milli þeirra komi fyrir í réttri röð í netinu G.

9. (8 stig) Gefið er net G með vigtir w á leggjum, hnútar s og t í netinu og jákvæð heiltala T. Lýsið reikniriti sem finnur stærstu tölu $x \geq 0$, þ.a. það er til leið p á milli s og t með lengd minni en eða jöfn T og allir leggir á leiðinni p hafa lengd $\geq x$. Hver er tímaflækjan á reikniritinu?

10. (5 stig) Þjappið strengnum "ananasbanani"með Huffman þjöppun. Athugið það er ekkert bil og engin línuending. Skrifið í svar bitastreng fyrir hvern staf og hvaða hnútar voru sameinaðir í hverju skrefi.

public class Stack<Item> implements Iterable<Item>

	Stack()	*create an empty stack*
void	push(Item item)	*add an item*
Item	pop()	*remove the most recently added item*
boolean	isEmpty()	*is the stack empty?*
int	size()	*number of items in the stack*

public class MaxPQ<Key extends Comparable<Key>>

	MaxPQ()	*create a priority queue*
	MaxPQ(int max)	*create a priority queue of initial capacity* max
	MaxPQ(Key[] a)	*create a priority queue from the keys in* a[]
void	insert(Key v)	*insert a key into the priority queue*
Key	max()	*return the largest key*
Key	delMax()	*return and remove the largest key*
boolean	isEmpty()	*is the priority queue empty?*
int	size()	*number of keys in the priority queue*

public class Graph

	Graph(int V)	*create a* V*-vertex graph with no edges*
	Graph(In in)	*read a graph from input stream* in
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(int v, int w)	*add edge* v-w *to this graph*
Iterable<Integer>	adj(int v)	*vertices adjacent to* v
String	toString()	*string representation*

public class Digraph

	Digraph(int V)	*create a* V*-vertex digraph with no edges*
	Digraph(In in)	*read a digraph from input stream* in
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(int v, int w)	*add edge* v->w *to this digraph*
Iterable<Integer>	adj(int v)	*vertices connected to* v *by edges pointing from* v
Digraph	reverse()	*reverse of this digraph*
String	toString()	*string representation*

public class Topological

	Topological(Digraph G)	*topological-sorting constructor*
boolean	isDAG()	*is* G *a DAG?*
Iterable<Integer>	order()	*vertices in topological order*

public class EdgeWeightedDigraph

	EdgeWeightedDigraph(int V)	*empty* V*-vertex digraph*
	EdgeWeightedDigraph(In in)	*construct from* in
int	V()	*number of vertices*
int	E()	*number of edges*
void	addEdge(DirectedEdge e)	*add* e *to this digraph*
Iterable<DirectedEdge>	adj(int v)	*edges pointing from* v
Iterable<DirectedEdge>	edges()	*all edges in this digraph*
String	toString()	*string representation*

public class UF

	UF(int N)	*initialize* N *sites with integer names (0 to* N-1*)*
void	union(int p, int q)	*add connection between* p *and* q
int	find(int p)	*component identifier for* p *(0 to* N-1*)*
boolean	connected(int p, int q)	*return* true *if* p *and* q *are in the same component*
int	count()	*number of components*

public class ST<Key, Value>

	ST()	*create a symbol table*
void	put(Key key, Value val)	*put key-value pair into the table (remove* key *from table if value is* null*)*
Value	get(Key key)	*value paired with* key *(*null *if* key *is absent)*
void	delete(Key key)	*remove* key *(and its value) from table*
boolean	contains(Key key)	*is there a value paired with* key*?*
boolean	isEmpty()	*is the table empty?*
int	size()	*number of key-value pairs in the table*
Iterable<Key>	keys()	*all the keys in the table*

public class Paths

	Paths(Graph G, int s)	*find paths in* G *from source* s
boolean	hasPathTo(int v)	*is there a path from* s *to* v*?*
Iterable<Integer>	pathTo(int v)	*path from* s *to* v*; null if no such path*

public class DirectedDFS

	DirectedDFS(Digraph G, int s)	*find vertices in* G *that are reachable from* s
	DirectedDFS(Digraph G, Iterable<Integer> sources)	*find vertices in* G *that are reachable from* sources
boolean	marked(int v)	*is* v *reachable?*

public class DirectedCycle

	DirectedCycle(Digraph G)	*cycle-finding constructor*
boolean	hasCycle()	*does* G *have a directed cycle?*
Iterable<Integer>	cycle()	*vertices on a cycle (if one exists)*

public class DirectedEdge

	DirectedEdge(int v, int w, double weight)	
double	weight()	*weight of this edge*
int	from()	*vertex this edge points from*
int	to()	*vertex this edge points to*
String	toString()	*string representation*

public class SP

	SP(EdgeWeightedDigraph G, int s)	*constructor*
double	distTo(int v)	*distance from* s *to* v*,* ∞ *if no path*
boolean	hasPathTo(int v)	*path from* s *to* v*?*
Iterable<DirectedEdge>	pathTo(int v)	*path from* s *to* v*,* null *if none*