08.71.23/24 Tölvunarfræði 2/2a

Lokapróf **30. apríl, 2002**
Kennari: Hjálmtýr Hafsteinsson **kl. 9^{00} ? 12^{00}**

Fyrstu 5 dæmin eru fyrir alla nemendur (bæði í Tölvunarfræði 2 og 2a) Dæmi 6 er aðeins fyrir nemendur í Tölvunarfræði 2, en dæmi 7 er aðeins fyrir nemendur í Tölvunarfræði 2a (verkfræði-nema). Í báðum tilfellum gilda ***fimm bestu dæmin af sex****. Dæmin hafa jafnt vægi.*

Öll skrifleg hjálpargögn og reiknivél leyfð*.*

?? Athugið að þegar beðið er um að "Lýsa" eða "Sýna" þá er nóg að gera það í orðum og með teikningum. Ef þið eigið að skrifa C++ kóda þá er beðið um það sérstaklega.
?? Rökstyðjið öll svör og munið að það er óþarfi að skrifa upp skilgreiningar sem eru í bókinni.

1. Í viktuðu útgáfunum af Union-Find eru tvö tré sameinuð eftir fjölda hnúta í þeim, ekki eftir hæð þeirra. Það er þó hæð trjánna sem ræður versta-tilfellistíma Find() aðgerðarinnar.

Hvers vegna er hæðin ekki notuð í hraðvirkum Union-Find reikniritum? Útskýrið nákvæmlega hvert vandamálið er, t.d. með dæmum eða teikningum.

2. Gefinn er tvítengdur listi, þar sem hnútabendirinn `haus` bendir á fremsta hnútinn. Einnig eru gefnir tveir hnútabendar, `p` og `x`. Þið eigið að skrifa forritsbút í C++ sem færir hnútinn sem `p` bendir á, aftur fyrir hnútinn sem `x` bendir á, en breytir listanum ekki að öðru leyti.

a) Hér að neðan er forritsbútur sem sagt er að meðhöndli almenna tilfellið. Er það rétt eða ekki? Rökstyðjið. Lagfærið bútinn ef hann er rangur.

```
p->prev->next = p->prev;
p->next->prev = p->next;
p->next = x->next;
x->next = p;
```

b) Nefnið öll þau sértilfelli sem geta komið upp við, sem almenna útgáfan ræður ekki við. Sýnið C++ forritsbúta sem meðhöndla hvert sértilfelli.

3. Skrifið fall í C++ sem fær inn bendi á rót tvíundartrés. Fallið á að skila 1 (satt) ef tréð er **tvíleitartré** (binary search tree), en 0 (ósatt) annars.

4. Segjum að við viljum leita að tilteknu gildi í *hrúgu* (e. heap).

a) Útfærið í C++ fall sem byrjar efst í hrúgunni og leitar aðeins eins langt niður tréð og þörf er á til að annað hvort að finna stakið eða sannfærast um að það sé ekki í trénu. Versta tilfellis tími á þessari aðferð verður *O(N)*.

b) Er hægt að nota helmingunarleit inní trénu til að auka hraða leitunarinnar? Útskýrið það eða erfiðleikana við það.

5. Í *opinni hökkun* (open addressing) eru stökin sett í töfluna sjálfa. Við árekstur í hólfi *i* þarf að finna annað hólf til að setja nýja stakið í. Undirflokkur opinnar hökkunar er *línuleg hökkun* (linear probing). Í þeirri aðferð er hólf *i*+1 skoðað fyrst, síðan hólf *i*+2, *i*+3, o.s.frv. Þessi aðferð veldur hópun (clustering).

Útfærið í C++ aðra aðferð, þar sem við árekstur eru næst athuguð hólfin $i+1$, $i+4$, $i+9$, ..., almennt $(i+k^2)$ mod M, þar sem M er stærð töflunnar. Sýnið innsetningar- (`insert`) og leitunaraðferðirnar (`search`) í C++. Útskýrið einnig kosti og galla þessarar aðferðar í samanburði við *tvöfalda hökkun* (double hashing).

Aðeins fyrir nemendur í Tölvunarfræði 2:

6. *Tvíundarinnsetningarröðun* (binary insertion sort) er eins og Innsetningarröðun, nema notuð er helmingunarleit til að finna rétta staðsetningu næsta staks í raðaða hluta vektorsins.

a) Lýsið þessari röðunaraðferð nánar, t.d. með sauðakóda (ekki endilega C++).

b) Hvenær væri þessi röðunaraðferð hraðvirkari en venjuleg Innsetningarröðun? Útskýrið.

Aðeins fyrir nemendur í Tölvunarfræði 2a (verkfræðinema):

7. *a)* Er til 4ra hnúta tvíundartré sem hefur sömu *inorder* og *preorder* röð? Rökstyðjið eða sýnið dæmi um slíkt tré.

b) En 4ra hnúta tvíundartré með sömu *preorder* og *postorder* röð? Rökstyðjið eða sýnið dæmi.

c) En 4ra hnúta tvíundartré með sömu *postorder* og *levelorder* röð? Rökstyðjið eða sýnið dæmi.