NIO Runde 2 2013/2014 - Oppgaveløsninger

1 Brodering

Denne oppgaven gikk ut på å simulere hva en broderingsmaskin gjør. Det vanskelige her var å finne en måte å få annenhver rad til å skrives ut fra høyre til venstre. Det var to måter å løse dette på. Den ene var å lage en array med alle stingene i som programmet skulle skrive ut, i den rekkefølgen de skulle skrives ut - og så skrive de ut i leseorden. Kode for dette vil være noe som

```
int k, r, n;
cin >> k >> r >> n;
char sting[k*r];
int pos = 0;
for (int i=0;i<n;i++) {
  int a;
  char f;
  cin >> a >> f;
  for (int j=0;j<a;j++) {
    sting[pos] = f;
    pos++;
  }
}
for (int rad = 0; rad < r; rad++) {
  if (rad % 2 == 0) {
    //partallsrader skrives ut i normal rekkefølge
    for (int kol = 0; kol < k; kol) {
      cout << sting[rad * k + kol];
    }
  } else {
    //oddetallsrader skrives ut baklengs
    for (int kol = k - 1; kol >= 0; kol) {
      cout << sting[rad * k + kol];
    }
  }
  cout << endl;
}
```

En annen algoritme er å ha en rad-buffer som man fyller inn enten fra venstre til høyre eller høyre til venste avhengig av hvilken rad man er på. Når bufferen er fyllt opp så skrives den ut og vi får videre til neste rad. Kode for dette kunne være

```
int k, r, n;
cin >> k >> r >> n;
string buffer(k, ' ');
int rad = 0;
int kol = 0;
for (int i=0;i<n;i++) {
  int a;
  char f;
  cin >> a >> f;
  for (int j=0;j<a;j++) {
    if (rad % 2 == 0) {
      //fylle buffer fra venstre til høyre
      buffer[kol++] = f;
      if (kol == k) {
        cout << buffer << endl;
        rad++;
      }
    } else {
      //fylle buffer fra høyre til venstre
      buffer[--kol] = f;
      if (kol == 0) {
        cout << buffer << endl;
        rad++;
      }
    }
  }
}
```

Man kunne fort gå i surr med variablene når man skulle implementere en av disse løsningene, og noen av deltagerene blandet sammen `k` og `r` og fikk dermed problemer med broderier som ikke var kvadratiske.

2 Flywheel

Her skulle man finne ut hvor mye penger man kunne tjene på å kjøpe og selge elektrisitet når man vet hvordan prisutviklingen kommer til å bli. Hvis man fant ut hva strategien burde være så var implementasjonen av den temmelig enkel. Observer at hvis $p_{i+1} > p_i$ så vil det alltid lønne seg å ha et fullladet hjul når man går inn i time $i + 1$ og at hvis $p_{i+1} < p_i$ så lønner det seg alltid å ha et helt utladet hjul når man går inn i time $i+1$ (hvis $p_{i+1} = p_i$ spiller det ingen rolle hva man gjør). Ettersom man tjener penger ved at verdien av elektrisiteten man er i besittelse av endrer seg så ser vi at vi tjener $(p_{i+1} - p_i) \times 1000$ euro når vi går fra time i til $i + 1$ hvis $p_{i+1} > p_i$ og at vi hverken tjener eller taper noe dersom $p_{i+1} \leq p_i$ (ettersom vi ikke har noe elektrisitet i dette tilfellet, så blir vi ikke påvirket av prisendringen).

En ting som skapte problemer for mange deltakere var at elektrisitetprisen kunne bli negativ. Dette hadde bare betydning hvis prisen var negativ i den siste perioden i datasettet. Da ville det nemmelig lønne seg å "kjøpe" opp elektrisitet for å fullade hjulet på slutten, selv om man ikke får solgt denne elektrisiteten.

3 Byggmester

Her fikk man to mengder med punkter på en linje som representerte eksisterende hus og potensielle tomter. Så skulle man for hver av de potensielle tomtene finne ut avstanden til det huset som var nermest. Dette kunne løses trivielt med å (for hver tomt) søke gjennom alle husene og finne den som var nærmest. Ettersom hver av de T tomtene må sammenlignes med hver av de N husene så tar denne algoritmen $\Theta(N \times T)$ tid (dvs. at tiden den bruker er omtrentlig proposjonal med $N \times T$). Ettersom N kunne være opp til 200000 og T opp til 10000 så blir dette for tregt for de største datasettene.

For å få kjøretiden ned så kan man sortere koordinatene til husene. Da kan vi bruke binærsøk for å finne hvilke to hus hver tomt befinner seg mellom (og behandle spesialtilfellene der en tomt er før det første huset eller etter det siste huset separat). Kjøretiden blir da bare $\Theta(N \log N)$ for sorteringen og $\Theta(\log N)$ for hvert binærsøk, noe som gir $\Theta(N \log N + T \log N)$ totalt.

4 Julegaver

Her ønsker vi å gi to typer julegaver til alle personene i en vennskapsgraf slik at to personer som er venner aldri får samme gave. I tillegg så ønsker vi å maksimere antallet av den ene typen gaver vi gir bort. De som har sett på NIO Bootcamp har kanskje sett oppgaven *Epler og Pærer* som ligner ganske mye. Begge er eksempler av to-fargings problemet på grafer. Forskjellen mellom oppgavene er bare at i *Julegaver* så vil vi maksimere antallet noder i den ene fargen, og at grafen ikke nødvendigvis er sammenhengende. For å løse problemer så kan man fargelegge hver komponent med to farger og så gir man bøker til den fargen det var flest av i hver komponent. Dette gjentar man da for hver komponent og summerer opp tallene. Dersom en komponent ikke lar seg fargelegge med to farger så skriver man bare ut 0 og avslutter programmet med en gang.

5 Konsert

Dette var den vanskeligste oppgaven i settet, men 6 deltagere klarte alikevel å få til full pott. Tilsynelatende virker det som en slags grafsøk-oppgave, men det er *dynamisk programmering* man må bruke for å løse den.

Merk at på et gitt tidspunkt i Line sin reise så er det kun viktig hvilken konsert hun er på akkurat nå, hvor mange konserter hun har vært på tidligere, og hvor mye penger hun har brukt. Nøyaktig hvilke konserter hun har vært på er ikke viktig ettersom hun er nødt til å dra på konsertene i rekkefølge og kan dermed ikke dra på konserter som allerede har vært.

La pris$[k][m]$ være den minste kostnaden Line må ut med for å dra på m konserter, avsluttende med konsert k. pris$[k][1]$ er kun reiseutgiften fra startpunktet $(0, 0)$ til (x_k, y_k) pluss billettprisen for konsert k. Altså er pris$[k][1] = |x_k| + |y_k| + b_k$ for alle $0 \leq k \leq N-1$. For $m > 1$ så vet vi at det må ha vært en eller annen konsert c som hun var på som sin $(m-1)$'te konsert rett før hun dro på den aktuelle konserten. Altså har vi

$$\text{pris}[k][m] = \min_{0 \leq c < k} (\text{pris}[c][m-1] + |x_k - x_c| + |y_k - y_c| + b_k)$$

Når vi har beregnet alle de forskjellige verdiene pris$[k][m]$ så er det bare å finne den største m slik at det finnes en pris$[k][m]$ som er mindre enn eller lik T.

Man kunne få 40 poeng for å løse de testsettene hvor $N \leq 20$. Disse testsettene kunne løses ved å prøve ut alle de 2^N delmengene av konsertene og se om Line hadde råd til å dra på de konsertene. Dette kan f.eks. gjøres ved hjelp av en rekursiv funksjon som for hver konsert prøver ut begge alternativene (enten å gå på konserten eller ikke) og så kaller seg selv nedover helt til den har gjort en avgjørelse for alle konsertene, og husker det største antall konserter som resulterte i en kostnad på mindre enn T.

En god del av deltagere klarte å få ganske bra med poeng på denne oppgaven ved å bruke en *heurestikk*, altså en algoritme som gjør noen kloke gjettinger som ofte (men ikke alltid) fører til riktig resultat. En heurestikk som gav mye poeng var å begynne med at Line skal gå på alle konsertene. Dersom hun ikke har råd til dette så sløyfer vi den konserten som koster henne mest å gå på (tatt i betraktning billetkostnaden til konserten og hvordan den påvirker reisekostnadene hennes). Så gjentas denne prosessen helt til vi har et sett med konserter som Line har råd til å dra på.

På NIO-finalen og på IOI vil slike heurestikker typisk gi få eller ingen poeng, men for denne oppgaven så hadde vi ganske *snille* testsett som heursestiske løsninger kunne klare.