Algorísmia
Col·lecció d'Exàmens Resolts

Amalia Duch Jordi Petit Salvador Roura

Facultat de Matemàtiques i Estadística
Universitat Politècnica de Catalunya

Índex

I Enunciats **15**

1 Recuperació, 29 de juny de 2018 **17**
1.1 Misteri . . . 17
1.2 Suma de dos elements . . . 17
1.3 Heaps . . . 17
1.4 NP-completesa . . . 18
1.5 Problema de l'aturada . . . 18
1.6 Tall mínim . . . 18
1.7 Teoria de jocs . . . 18
1.8 Què calcula? . . . 19

2 Final, 18 de gener de 2018 **21**
2.1 Recurrències . . . 21
2.2 Els k més petits, en ordre . . . 21
2.3 Arbres AVL . . . 21
2.4 NP-completesa . . . 22
2.5 2-SAT . . . 22
2.6 Teoria de jocs . . . 22
2.7 Què fan? . . . 22
2.8 Codi estrany . . . 23

3 Parcial, 6 de novembre de 2017 **25**
3.1 Ordenació . . . 25
3.2 Cues de prioritats . . . 25
3.3 Programació dinàmica . . . 26
3.4 Multiplicació russo-francesa . . . 26
3.5 Notació asimptòtica . . . 26
3.6 Zeros i uns . . . 26

4 Recuperació, 27 de juny de 2017 **29**
4.1 Cerques i insercions . . . 29
4.2 Encara un altre codi misteriós . . . 30
4.3 I encara un altre . . . 30
4.4 Arbres AVL . . . 30
4.5 La motxilla . . . 31
4.6 Els del mig . . . 31
4.7 Collatz i Turing . . . 31
4.8 El valor d'un graf . . . 32

5 Final, 17 de gener de 2017 **33**
5.1 Hashing universal . . . 33
5.2 Recurrències . . . 33
5.3 MAX3SAT . . . 34
5.4 Flux màxim . . . 34
5.5 Teoria de jocs . . . 34
5.6 Codi espatllat . . . 34
5.7 Més codis misteriosos . . . 35

6 Parcial, 15 de novembre de 2016 **37**
6.1 Recurrència simpàtica . . . 37
6.2 Els més petits . . . 37
6.3 Mínim local . . . 37
6.4 Ordenació . . . 38
6.5 Codi misteriós . . . 38
6.6 El problema d'en Nikochan . . . 38

7 Recuperació, 7 de juliol de 2016 **39**
7.1 Recurrències . . . 39
7.2 Escacs . . . 39
7.3 Consultes en un llibre electrònic . . . 39
7.4 Codis misteriosos . . . 40
7.5 Algorisme de Kruskal . . . 41
7.6 Teoria de jocs . . . 42

8 Final, 19 de gener de 2016 **43**

9 Final, 19 de gener de 2016 **45**
9.1 Costos . . . 45
9.2 Heaps . . . 46
9.3 Cicles i camins Hamiltonians . . . 46
9.4 Inaproximabilitat . . . 46
9.5 Arbres aleatoritzats . . . 46
9.6 Components fortament connexos . . . 47
9.7 Algorisme de Prim . . . 47
9.8 Flors i colors . . . 47

10 Parcial, 3 de novembre de 2015 **49**
10.1 Pica-pica . . . 49
10.2 Maleïdes recurrències . . . 49
10.3 Element solitari . . . 50
10.4 En posició! . . . 50

10.5 Comptatge 50
10.6 Selecció 50

11 Recuperació, 26 de juny 2015 **51**
11.1 Punts propers a l'origen 51
11.2 Propietats dels algorismes d'ordenació 51
11.3 Grafs 51
11.4 Teoria de jocs 52
11.5 Arbres AVL 52
11.6 Un altre codi espifiat 52
11.7 NP-completesa 52

12 Final, 16 de gener de 2015 **53**
12.1 Recurrència 53
12.2 Arbres AVL 53
12.3 Heaps ternaris 53
12.4 NP-completesa 54
12.5 Ordenació topològica 54
12.6 Codi espifiat 54
12.7 Max-flow 55
12.8 Teoria de jocs 55

13 Parcial, 14 de novembre de 2014 **57**
13.1 Ordenació per selecció 57
13.2 Recurrència 57
13.3 Primer element repetit 57
13.4 Subvector de suma zero 58
13.5 Nombres de Fibonacci 58
13.6 Heaps 58
13.7 Ordenació topològica 58

14 Recuperació, 7 de juliol de 2014 **59**
14.1 Recurrència 59
14.2 Interseccions 59
14.3 Anagrames 59
14.4 Creació de heaps 59
14.5 Quadrat d'una matriu 60
14.6 Arbres de cerca 60
14.7 Teoria de jocs 60

15 Final, 16 de gener de 2014 **61**
15.1 Sumeu zero 61
15.2 Ordenació de vectors ordenadets 61
15.3 Arbres de cerca 61
15.4 Teoria de jocs 62
15.5 Fluxos 62
15.6 Codi mort 63

16 Parcial, 8 de novembre de 2013 **65**
16.1 Heu preparat la lliçó? . . . 65
16.2 Polinomis . . . 65
16.3 Heaps i heapsort . . . 65
16.4 Comparacions del Quicksort . . . 66
16.5 Múltiples de 66 en un conjunt . . . 66

17 Final, 15 de gener de 2013 **67**
17.1 El mateix que el del parcial . . . 67
17.2 Algorisme curiós . . . 67
17.3 Problemes petits i bonics sobre vectors . . . 68
17.4 Agència matrimonial . . . 69
17.5 Tasques i màquines . . . 69

18 Parcial, 13 de novembre de 2012 **71**
18.1 Ompliu els blancs . . . 71
18.2 Comparacions del QuickSort . . . 71
18.3 L'Euclides no hi és . . . 72
18.4 La funció misteri ataca de nou . . . 72
18.5 Nombres aborrits . . . 73
18.6 Encara més misteris? . . . 73

19 Recuperació, 29 de juny de 2012 **75**
19.1 Miscel·lània . . . 75
19.2 Algorismes d'aproximació . . . 76
19.3 Complexitat i indecidibilitat . . . 76
19.4 Programació dinàmica . . . 76
19.5 Teoria de jocs . . . 77
19.6 Codi misteriós . . . 77

20 Final, 11 de gener de 2012 **79**
20.1 Codis de Huffman . . . 79
20.2 Heaps . . . 79
20.3 Teoria de jocs . . . 79
20.4 Estructures de dades . . . 80
20.5 NP-completesa i programació dinàmica . . . 80
20.6 Algorismes d'aproximació . . . 80

21 Parcial, 27 d'octubre de 2011 **83**
21.1 Ompliu els blancs . . . 83
21.2 Costs . . . 84
21.3 Sumar un . . . 84
21.4 Heaps . . . 85
21.5 Triangles en grafs . . . 85

22 Recuperació, 5 de juliol de 2011 **87**
22.1 Cues de prioritats . . . 87
22.2 Anàlisi . . . 87
22.3 NP-completesa . . . 88
22.4 Indecidibilitat . . . 88
22.5 Aproximació . . . 88
22.6 Grafs . . . 89

22.7 Arbres . . . 89
22.8 Més arbres . . . 89

23 Final, 11 de gener de 2011 **91**
23.1 Heaps i AVLs . . . 91
23.2 Anàlisi . . . 91
23.3 NP-completesa . . . 92
23.4 Indecidibilitat . . . 92
23.5 Aproximació . . . 93
23.6 Grafs . . . 93
23.7 Llistes . . . 93
23.8 Arbres . . . 94

24 Parcial, 28 d'octubre de 2010 **95**
24.1 Ompliu els blancs . . . 95
24.2 Ordenació . . . 96
24.3 Dividir i vèncer . . . 96
24.4 Primalitat . . . 96
24.5 Producte de matrius . . . 96

II Solucions **99**

1 Recuperació, 29 de juny de 2018 **101**
1.1 Misteri . . . 101
1.2 Suma de dos elements . . . 101
1.3 Heaps . . . 101
1.4 NP-completesa . . . 102
1.5 Problema de l'aturada . . . 102
1.6 Components fortament connexos . . . 102
1.7 Tall mínim . . . 103
1.8 Teoria de jocs . . . 104
1.9 Què calcula? . . . 104

2 Final, 18 de gener de 2018 **105**
2.1 Recurrències . . . 105
2.2 Els k més petits, en ordre . . . 105
2.3 Arbres AVL . . . 105
2.4 NP-completesa . . . 106
2.5 2-SAT . . . 106
2.6 Teoria de jocs . . . 106
2.7 Què fan? . . . 106
2.8 Codi estrany . . . 107

3 Parcial, 6 de novembre de 2017 **109**
3.1 Ordenació . . . 109
3.2 Cues de prioritats . . . 109
3.3 Programació dinàmica . . . 110
3.4 Multiplicació russo-francesa . . . 110
3.5 Notació asimptòtica . . . 110
3.6 Zeros i uns . . . 110

4 Recuperació, 27 de juny de 2017 **111**
4.1 Cerques i insercions . . . 111
4.2 Encara un altre codi misteriós . . . 111
4.3 I un altre . . . 112
4.4 Arbres AVL . . . 112
4.5 La motxilla . . . 112
4.6 Els del mig . . . 113
4.7 Collatz i Turing . . . 113
4.8 El valor d'un graf . . . 113

5 Final, 17 de gener de 2017 **115**
5.1 Hashing universal . . . 115
5.2 Recurrències . . . 115
5.3 MAX3SAT . . . 115
5.4 Flux màxim . . . 116
5.5 Teoria de jocs . . . 116
5.6 Codi espatllat . . . 116
5.7 Més codis misteriosos . . . 117

6 Parcial, 15 de novembre de 2016 **119**
6.1 Recurrència simpàtica . . . 119
6.2 Els més petits . . . 119
6.3 Mínim local . . . 120
6.4 Ordenació . . . 120
6.5 Codi misteriós . . . 120
6.6 El problema d'en Nikochan . . . 120

7 Recuperació, 7 de juliol de 2016 **121**
7.1 Recurrències . . . 121
7.2 Escacs . . . 121
7.3 Consultes en un llibre electrònic . . . 122
7.4 Codis misteriosos . . . 122
7.5 Algorisme de Kruskal . . . 123
7.6 Teoria de jocs . . . 123

8 Final, 19 de gener de 2016 **125**
8.1 Costos . . . 125
8.2 Heaps . . . 126
8.3 Cicles i camins Hamiltonians . . . 127
8.4 Inaproximabilitat . . . 127
8.5 Arbres aleatoritzats . . . 128
8.6 Components fortament connexs . . . 128
8.7 Algorisme de Prim . . . 128
8.8 Flors i colors . . . 129

9 Parcial, 3 de novembre de 2015 **131**
9.1 Pica-pica . . . 131
9.2 Maleïdes recurrències . . . 131
9.3 Element solitari . . . 132
9.4 En posició! . . . 132
9.5 Comptatge . . . 132
9.6 Selecció . . . 132

10 Extraordinari, 26 de juny de 2015 **133**
10.1 Punts propers a l'origen . . . 133
10.2 Propietats dels algorismes d'ordenació . . . 133
10.3 Grafs . . . 133
10.4 Teoria de jocs . . . 134
10.5 Arbres AVL . . . 134
10.6 Un altre codi espifiat . . . 134
10.7 NP-completesa . . . 134

11 Final, 16 de gener de 2015 **135**
11.1 Recurrència . . . 135
11.2 Arbres AVL . . . 136
11.3 Heaps ternaris . . . 136
11.4 NP-completesa . . . 137
11.5 Ordenació topològica . . . 137
11.6 Codi espifiat . . . 138
11.7 Max-flow . . . 138
11.8 Teoria de jocs . . . 138

12 Parcial, 14 de novembre de 2014 **139**
12.1 Ordenació per selecció . . . 139
12.2 Recurrència . . . 140
12.3 Primer element repetit . . . 140
12.4 Subvector de suma zero . . . 141
12.5 Nombres de Fibonacci . . . 141
12.6 Heaps . . . 141
12.7 Ordenació topològica . . . 141

13 Recuperació, 7 de juliol de 2014 **143**
13.1 Recurrència . . . 143
13.2 Interseccions . . . 143
13.3 Anagrames . . . 143
13.4 Creació de heaps . . . 144
13.5 Quadrat d'una matriu . . . 144
13.6 Arbres de cerca . . . 144
13.7 Teoria de jocs . . . 144

14 Final, 16 de gener de 2014 **145**
14.1 Sumeu zero . . . 145
14.2 Ordenació de vectors ordenadets . . . 145
14.3 Arbres de cerca . . . 146
14.4 Teoria de jocs . . . 146
14.5 Fluxos . . . 146
14.6 Codi mort . . . 146

15 Parcial, 8 de novembre de 2013 **147**
15.1 Heu preparat la lliçó? . . . 147
15.2 Polinomis . . . 147
15.3 Heaps i heapsort . . . 148
15.4 Comparacions del Quicksort . . . 149
15.5 Múltiples de 66 en un conjunt . . . 149

16 Final, 15 de gener de 2013 **151**
16.1 El mateix que el del parcial . . . 151
16.2 Algorisme curiós . . . 151
16.3 Problemes petits i bonics sobre vectors . . . 152
16.4 Agència matrimonial . . . 152
16.5 Tasques i màquines . . . 153

17 Parcial, 13 de novembre de 2012 **155**
17.1 Ompliu els blancs . . . 155
17.2 Comparacions del QuickSort . . . 155
17.3 L'Euclides no hi és . . . 156
17.4 La funció misteri ataca de nou . . . 157
17.5 Nombres aborrits . . . 157
17.6 Encara més misteris? . . . 157

18 Recuperació, 29 de juny de 2012 **159**
18.1 Miscel·lània . . . 159
18.2 Algorismes d'aproximació . . . 159
18.3 Complexitat i indecidibilitat . . . 160
18.4 Programació dinàmica . . . 160
18.5 Teoria de jocs . . . 161
18.6 Codi misteriós . . . 161

19 Final, 11 de gener de 2012 **163**
19.1 Codis de Huffman . . . 163
19.2 Heaps . . . 163
19.3 Teoria de jocs . . . 164
19.4 Estructures de dades . . . 164
19.5 NP-completesa i programació dinàmica . . . 165
19.6 Algorismes d'aproximació . . . 166

20 Parcial, 27 d'octubre de 2011 **167**
20.1 Ompliu els blancs . . . 167
20.2 Costs . . . 168
20.3 Sumar un . . . 168
20.4 Heaps . . . 168
20.5 Triangles en grafs . . . 169

21 Recuperació, 5 de juliol de 2011 **171**
21.1 Cues de prioritats . . . 171
21.2 Anàlisi . . . 171
21.3 NP-completesa . . . 171
21.4 Indecidibilitat . . . 172
21.5 Aproximació . . . 172
21.6 Grafs . . . 172
21.7 Arbres . . . 172
21.8 Més arbres . . . 172

22 Final, 11 de gener de 2011 **173**
22.1 Heaps i AVLs . . . 173
22.2 Anàlisi . . . 174
22.3 NP-completesa . . . 174
22.4 Indecidibilitat . . . 175
22.5 Aproximació . . . 175
22.6 Grafs . . . 175
22.7 Llistes . . . 176
22.8 Arbres . . . 176

23 Parcial, 28 d'octubre de 2010 **179**
23.1 Ompliu els blancs . . . 179
23.2 Ordenació . . . 180
23.3 Dividir i vèncer . . . 180
23.4 Primalitat . . . 180
23.5 Producte de matrius . . . 181

Prefaci

Aquesta és una col·lecció de problemes i d'exàmens resolts per a l'assignatura **Algorísmia** del Grau de Matemàtiques de la Facultat de Matemàtiques i Estadística.

Els exàmens recollits en aquesta col·lecció són els que s'han fet en aquesta assignatura des de la seva creació. Els enunciats, amb les corresponents solucions, s'han ordenat expressament de més a menys recents.

La nostra voluntat és anar ampliant i millorant aquesta col·lecció d'exàmens resolts amb el pas del temps.

Salvador i Jordi
Barcelona, 6 de juliol de 2011.

Part I

Enunciats

1

Recuperació, 29 de juny de 2018

1.1 Misteri

Considereu el codi següent:

```
bool f(const string& a, const string& b) {
    int n = a.size();
    if (b.size() != n) return false;
    int k = 0;
    while (k < n and a[k] != b[0]) ++k;
    if (k == n) return false;
    for (int i = 0; i < n; ++i)
      if (a[(k+i)%n] != b[i]) return false;
    return true;
}
```

a) Digueu raonadament quin és el cost, en cas pitjor, de $f(a, b)$ en funció d'n.
b) Suposeu que dins d'a no hi ha cap caràcter repetit, i el mateix per a b. Què fa $f(a,b)$?
c) Expliqueu amb un exemple que passaria si tant a com b tinguessin caràcters repetits.

1.2 Suma de dos elements

Considereu un vector V amb n enters diferents i un altre enter x. Expliqueu, només amb paraules, com trobaríeu si a V hi ha almenys dos elements diferents que sumin exactament x, amb cost total $\Theta(n)$ en el cas mitjà.

1.3 Heaps

Donat el vector [4, 1, 3, 2, 16, 9, 10, 14, 8, 7], feu servir l'algorisme *heapify* per convertir-lo en un max-heap, dibuixant com es transforma a cada pas.

1.4 NP-completesa

a) Donats dos problemes NP-difícils X i Y, existeix forçosament una reducció en temps polinòmic d'X a Y?

b) Sigui X un problema NP-complet. Sigui Y un problema tal que X es redueix a Y. Podem deduir que Y és NP-complet?

c) Sigui X un problema NP-complet. Sigui Y un problema tal que Y es redueix a X. Podem deduir que Y és NP-complet?

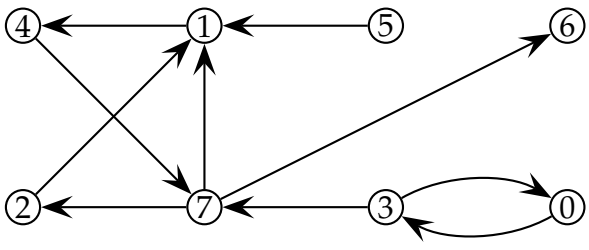
1.5 Problema de l'aturada

Considereu aquest problema: Donat un natural n, cal decidir si hi ha algun programa (en C++, per exemple) de mida n (és a dir, tal que el seu codi tingui n caràcters) que s'aturi per a totes les entrades.

És aquest problema decidible?

6. Components fortament connexos **(1'5 punts)**

Calculeu els components fortament connexos del graf següent. Per fer-ho, heu d'usar l'algorisme explicat a classe. Mostreu els diversos passos de l'algorisme amb prou detall.



1.6 Tall mínim

Considereu un graf no dirigit G, amb pesos estrictament positius a les arestes. Siguin x, y i z tres vèrtexs diferents qualssevol de G. Si el tall mínim entre x i y val 10, i el tall mínim entre y i z val 10, pot ser que el tall mínim entre x i z valgui 100? Si és possible, mostreu-ne un exemple. Altrament, expliqueu per què és impossible.

1.7 Teoria de jocs

Demostreu que la composició de dues posicions d'un joc imparcial és perdedora si i només si les dues posicions tenen el mateix nímber.

A la demostració, podeu fer servir totes les propietats dels nímbers, excepte la que diu que el nímber de la composició de dues posicions és l'"or exclusiu" dels seus nímbers.

1.8 Què calcula?

Considereu aquest codi:

```
int main() {
  int N;
  cin >> N;
  vector<int> V(N + 1, 0);
  V[0] = 1;
  for (int x = 1; x <= N; ++x)
    for (int i = x; i <= N; ++i) V[i] += V[i-x];

  for (int x = 1; x <= N; ++x) cout << V[x] << endl;
}
```

Quins són els sis primers nombres que escriu? (Suposeu que N és almenys 6.) Quin cost té aquest codi? Ignorant els sobreiximents, què calcula?

2

Final, 18 de gener de 2018

2.1 Recurrències

Considereu aquest codi per a $n \geq 1$:

```
void escriu(int n) {
  stack<int> pila;
  pila.push(n);
  while (not pila.empty()) {
    int x = pila.top();
    pila.pop();
    cout << ' ' << x;
    if (x > 1) {
      pila.push(x - 1);
      pila.push(x - 1);
} } }
```

a) Doneu un codi recursiu equivalent a l'anterior per a *escriu* (n).

b) Doneu i resoleu una recurrència que expressi el cost del vostre procediment.

2.2 Els k més petits, en ordre

Considereu un vector desordenat amb n enters diferents. Expliqueu només amb paraules com en trobaríeu els $k = n/\log n$ enters més petits, ordenats de petit a gran, amb cost total $\Theta(n)$ en el cas pitjor.

2.3 Arbres AVL

Inseriu, dibuixant l'arbre a cada pas, les claus 94, 33, 50, 76, 96 i 67 en aquest ordre en un AVL inicialment buit. Digueu per a quines claus s'ha de rebalancejar l'arbre, i amb quin tipus de rotació: senzilla o doble.

2.4 NP-completesa

Suposeu que l'entrada de tots els algorismes d'aquest problema són b bits que codifiquen un numero x entre 2^{b-1} i $2^b - 1$. És un fet conegut que comprovar si x és un nombre primer es pot fer en temps polinòmic en b. Però ningú no sap com factoritzar x com a producte de primers en temps polinòmic en b.

a) El problema de factoritzar un nombre donat x pertany a NP?

b) Si es demostrés que no existeix cap algorisme polinòmic en b per factoritzar un nombre donat x, podríem deduir P = NP o bé P $\neq$ NP?

c) Si es trobés algun algorisme polinòmic en b per factoritzar un nombre donat x, podríem deduir P = NP o bé P $\neq$ NP?

2.5 2-SAT

Considereu aquesta fórmula booleana: $(\bar{a} \vee c) \wedge (a \vee d) \wedge (\bar{b} \vee \bar{c}) \wedge (\bar{c} \vee \bar{d}) \wedge (\bar{b} \vee c)$
Fent servir l'algorisme explicat a classe, digueu si té alguna solució. Raoneu quantes.

2.6 Teoria de jocs

Recordem el joc del Nim: Hi ha diverses piles de pedres, i dos jugadors, per torns, han de treure almenys una pedra d'una pila no buida; perd qui no pot jugar. Considereu una variant on només es pot treure, o bé una pedra, o bé un nombre parell. Per exemple, d'una pila amb 6 pedres, no se'n poden treure ni 3 ni 5.

Suposeu una situació amb quatre piles amb 1, 3, 7 i 9 pedres. Digueu raonadament si treure 1 pedra de la pila amb 9 pedres és una jugada guanyadora.

2.7 Què fan?

Sigui V un un VE (vector d'enters) amb $n > 0$ enters positius en qualsevol ordre.

```
bool f(const VE& V) {
  set<int> F;
  for (int x : V) {
    if (F.find(x) != F.end()) return true;
    F.insert(x);
  }
  return false;
}

bool g(const VE& V) {
  VE G = V;
  sort(G.begin(), G.end());
  for (int i = 0; i + 1 < G.size(); ++i)
    if (G[i] == G[i+1]) return true;
  return false;
}
```

```
bool h(const VE& V) {
  int n = V.size();
  vector<VE> H(n);
  for (int x : V) {
    int i = (37*x + 23)%n;
    for (int y : H[i])
      if (y == x) return true;
    H[i].push_back(x);
  }
  return false;
}
```

Què calcula cada funció? Breument, com ho fan? Quin cost té cadascuna en el seu cas millor i en el seu cas pitjor? Quins són aquests casos?

2.8 Codi estrany

Assumiu aquestes definicions de tipus:

```
using P = pair<int, int>;
using VP = vector<P>;
using Graf = vector<VP>;
```

Suposeu que G conté un *Graf* no dirigit i connex, amb n vèrtexs entre 0 i $n-1$, i amb pesos positius a les arestes, on per a cada aresta entre dos vèrtexs x i y amb cost c, $G[x]$ conté un parell $P(y, c)$ i $G[y]$ conté un parell $P(x, c)$. Considereu el codi següent:

```
int res = 0;
vector<bool> vist(n, false);
priority_queue<P> Q;
Q.push(P(0, 0));
while (not Q.empty()) {
  P p = Q.top(); Q.pop();
  int x = p.second;
  if (not vist[x]) {
    vist[x] = true;
    res += -p.first;
    for (P aresta : G[x]) Q.push(P(-aresta.second, aresta.first));
  }
}
cout << res << endl;
```

Què calcula aquest codi? Quin cost té? Expliqueu breument el seu funcionament, incloent un exemple d'execució amb un graf petit.

3

Parcial, 6 de novembre de 2017

3.1 Ordenació

Sigui s un string amb n lletres minúscules. Sense escriure codi ni pseudo-codi, expliqueu com ordenaríeu s de la forma més eficient possible. Per exemple, si s fos "abracadabra", el resultat hauria de ser "aaaaabbcdrr".

El cost del vostre algorisme hauria de ser inferior a $O(n \log n)$. Expliqueu per què això no contradiu la fita inferior sobre els algorismes d'ordenació explicada a classe.

3.2 Cues de prioritats

Sigui V un vector amb n enters, i sigui x un natural entre 1 i $n-1$. Considereu aquest codi:

```
priority_queue <int> Q;
for (int i = 0; i < n; ++i) {
    Q.push(V[i]);
    if (i ≥ x) Q.pop();
}
while (not Q.empty()) {
    cout ≪ Q.top() ≪ endl;
    Q.pop();
}
```

a) Què escriu i en quin ordre?

b) Quin cost té, en funció d'n i x?

c) Ara, suposeu $x = \sqrt{n}$. Si només ens importa què s'escriu, i no en quin ordre, expliqueu amb paraules un algorisme alternatiu que sigui més eficient. Quin cost té?

3.3 Programació dinàmica

Sigui k un enter estrictament positiu, i sigui V un vector amb n enters diferents estrictaments positius. Per a valors de i entre 0 i n, i valors de j entre 0 i k, sigui la recurrència

$$f(i,j) = f(i-1,j) \;+\; f(i-1,j-V[i-1]) \;+\; f(i-1,j-2V[i-1])\ ,$$

amb casos base

$$f(i,j) = 0 \quad \text{per a } j < 0, \qquad f(0,j) = 0 \quad \text{per a } j > 0, \qquad f(0,0) = 1.$$

a) Expliqueu amb paraules comunes què calcula aquesta recurrència.

b) El codi corresponent, quin cost tindria en temps? Sense empitjorar el cost en temps, quin és el cost total en espai més petit que es pot aconseguir?

3.4 Multiplicació russo-francesa

Considereu el mètode següent per multiplicar dos naturals de n bits x i y:

```
int multiply(int x, int y) {
    if (y == 0) return 0;
    return 2*multiply(x, y/2) + (y%2 == 0 ? 0 : x);
}
```

a) Calculeu el cost de *multiply* en funció de n, considerant que els nombres internament són com vectors de bits. Per exemple, dividir per 2 un nombre de n bits suposa moure tots els bits una posició a la dreta, amb cost $\Theta(n)$.

b) Es pot multiplicar més eficientment? Com? Amb quin cost?

3.5 Notació asimptòtica

En els casos següents, indiqueu si $f = O(g)$, $f = \Omega(g)$, o $f = \Theta(g)$. En cada cas, cal triar-ne exactament una. No cal justificar res.

$f(n)$	$g(n)$	$f(n) = X(g(n))$
$n^{1/3}$	$n^{2/3}$	
$100 \cdot n + \log n$	$n + (\log n)^2$	
$n \cdot 2^n$	3^n	
$10 \cdot \log n$	$\log(n^2)$	
2^n	$2^{n/2}$	

3.6 Zeros i uns

Considereu el codi següent:

```
int parteix (vector<int>& v, int e, int d) {
    int p = v[e], i = e + 1;
    while (true) {
        while (v[i] ≤ p and i < d) ++i;
        while (v[d] ≥ p and d ≥ i) −−d;
        if (i ≥ d) { swap(v[e], v[d]); return d; }
```

```
            swap(v[i ], v[d ]);
}   }

void sort ( vector<int>& v, int e, int d) {
    if (e < d) {
        int p = parteix (v, e, d);
        sort (v, e, p − 1);   sort (v, p + 1, d);
}   }

void sort ( vector<int>& v) { sort(v, 0, v. size () − 1); }
```

a) Digueu quin és el cost de *sort* si tots els elements de v són 0.

b) I quin és el cost mitjà si cada $v[i]$ és 0 o 1 amb probabilitat $1/2$ independent?

4

Recuperació, 27 de juny de 2017

4.1 Cerques i insercions

Ompliu la taula següent per descriure el cost de l'operació de cerca d'una clau existent en un conjunt amb n claus implementat amb diverses estructures de dades. Per al cas mitjà, suposeu que les clau s'han inserit en ordre aleatori, i que es busca qualsevol de les claus existents amb la mateixa probabilitat. Aquí no heu de raonar res.

Cerca	Cas pitjor	Cas mitjà
Taula		
Taula ordenada		
Arbre binari de cerca		
Arbre AVL		
Taula de hash		
Perfect hashing		

Repetiu-ho per al cost d'inserir una nova clau. (L'algorisme d'inserció no pot suposar que la clau és nova, ho ha de comprovar.) Per al cas mitjà, suposeu que la nova clau té la mateixa probabilitat de ser més petita que totes les claus ja existents, o de ser més gran que la mínima actual però més petita que la segona, ..., o de ser més gran que totes les claus ja existents. Aquí tampoc no heu de raonar res.

Inserció	Cas pitjor	Cas mitjà
Taula		
Taula ordenada		
Arbre binari de cerca		
Arbre AVL		
Taula de hash		

Expliqueu per què no tindria sentit fer la pregunta sobre les insercions per al perfect hashing.

4.2 Encara un altre codi misteriós

Per resoldre aquest exercici, suposeu que els **int**'s de C++ són arbitràriament grans, és a dir, que no pateixen mai sobreiximents. També, suposeu que **int** *next*(**int** *n*); és una funció que calcula en temps constant un enter que només és funció d'*n*. Considereu aquest codi:

```
int main() {
  int n;
  cin >> n;

  set<int> S;
  S.insert(n);
  n = next(n);
  while (S.find(n) == S.end()) {
    S.insert(n);
    n = next(n);
  }

  int q = 1;
  int seg = next(n);
  while (seg != n) {
    ++q;
    seg = next(seg);
  }
  cout << q << endl;
}
```

Expliqueu breument què calcula i com ho fa. Quin cost té?

4.3 I encara un altre

Sigui G una matriu de booleans que descriu un graf no dirigit, tal que $G[x][y]$ i $G[y][x]$ són certs si i només si els vèrtexs x i y estan connectats directament amb una aresta. Considereu aquest codi:

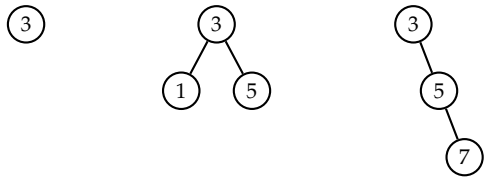
```
int n = G.size();
for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
    for (int k = 0; k < n; ++k)
      G[i][j] = G[i][j] or (G[i][k] and G[k][j]);
```

Què pretén calcular i de quina manera? Aquest codi té un error. Quin és i com s'hauria de corregir?

4.4 Arbres AVL

Donat un arbre binari de cerca, definim la seva alçada com el màxim nombre de nodes en qualsevol camí de l'arrel a les fulles. Per exemple, aquests són arbres binaris de cerca

amb alçades respectives 1, 2 i 3:



Si no existeix cap arbre binari de cerca AVL amb 12 nodes i alçada 5, expliqueu per què. Si existeix, dibuixeu-ne un i expliqueu breument quines propietats té que el fan un arbre binari de cerca AVL.

4.5 La motxilla

Recordeu el problema de la motxilla: Donats n objectes amb pesos $p_1, \ldots, p_n$ i beneficis $b_1, \ldots, b_n$ i una motxilla de capacitat C, es vol trobar un subconjunt S dels objectes que maximitzi el benefici total $\sum_{i \in S} b_i$ tot garantint que el seu pes total no excedeix la capacitat de la motxilla: $\sum_{i \in S} p_i \leq C$. Els objectes con es poden partir.

1. Demostreu que l'algorisme que ordena els objectes segons la seva relació b_i / p_i en ordre decreixent i que tria els primers que, sumats, no excedeixen la capacitat de la motxilla, no és un algorisme d'aproximació de raó constant.

2. Modifiqueu lleugerament l'algorisme anterior per tal d'aconseguir un algorisme d'aproximació de raó 2. Pista: Penseu què fer amb el primer objecte que ja no cap a la motxilla.

4.6 Els del mig

Donat un vector $v\,[0..n-1]$ d'n elements, on n és múltiple de 4, es vol saber quins són els $n/2$ elements del mig, és a dir, aquells que estarien entre les posicions $n/4$ i $3n/4-1$, ambdues incloses, si s'ordenés el vector. Sense donar codi ni pseudo-codi, descriviu amb prou detall un algorisme de cost $O(n)$ en el cas pitjor que resolgui aquest problema.

4.7 Collatz i Turing

Sigui n un natural estrictament positiu. Considereu el procés següent: Si n és parell, dividiu-lo per dos. Altrament, multipliqueu-lo per 3 i sumeu-li 1. Quan arribeu a 1, pareu. Per exemple, començant en 3, s'obté la seqüència $3, 10, 5, 16, 8, 4, 2, 1$. Collatz va conjecturar l'any 1937 que aquest procés acaba per a qualsevol n inicial, però encara no ho ha sabut demostrar ningú.

Suposeu ara que el problema de l'aturada fós decidible: És a dir, disposeu d'una funció T que, donat qualsevol programa A i qualsevol entrada x, calcula i retorna si A s'aturaria eventualment quan s'apliqués sobre l'entrada x.

Com utilitzaríeu T per saber si la conjectura de Collatz és certa o no?

4.8 El valor d'un graf

El problema del recobriment de vèrtexs (VERTEX-COVER) consisteix a, donat un graf no dirigit $G = (V, E)$ i un natural K, determinar si existeix un subconjunt $S \subseteq V$ amb $|S| \leq K$ tal que que $\forall \{u, v\} \in E$, $u \in S$ o $v \in S$. Recordeu que VERTEX-COVER és **NP**-complet.

El problema del valor d'un graf (GRAPH-VALUE) consisteix a, donat un graf no dirigit $G = (V, E)$ amb $n = |V|$ vèrtexs, una llista de n naturals $\langle a_1, \ldots, a_n \rangle$, i un natural B, determinar si existeix una permutació $\pi : \{1, \ldots, n\} \leftrightarrow V$ de forma que

$$\chi(\pi) = \sum_{\{u,v\} \in E} a_{\pi(u)} \cdot a_{\pi(v)}$$

sigui menor o igual que B.

Demostreu que GRAPH-VALUE és **NP**-complet.

5

Final, 17 de gener de 2017

5.1 Hashing universal

Sigui $\{0, 1, \ldots, 2^n - 1\}$ un univers d'elements x que es vol "hashejar" en una taula de 2^m posicions. Els elements de l'univers i les entrades de la taula es representen a través de vectors de bits de mides n i m, respectivament. Considereu la família següent de funcions de hash:

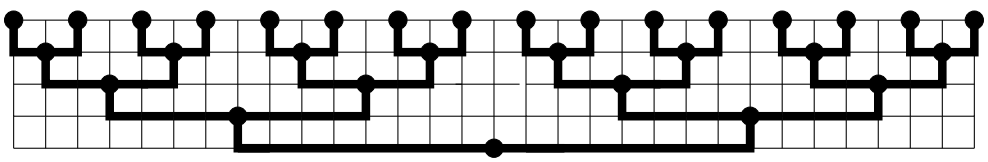
$$\mathcal{H} = \{h_M \mid M \text{ és una matriu de } 0/1 \text{ de mida } m \times n, \text{ i } h_M(x) = (M \cdot x) \bmod 2\}$$

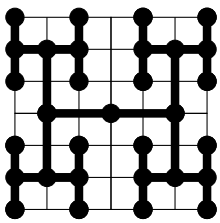
a) Definiu el concepte de família universal de funcions de hash.

b) Demostreu que $\mathcal{H}$ és una família universal de funcions de hash.

5.2 Recurrències

Tots els arbres es poden representar, sense cap intersecció entre arestes, en una graella rectangular que sigui prou gran. A continuació es mostren dues representacions d'un arbre binari complet de $2^4 = 16$ fulles (la primera té alçada 4 i amplada 30; la segona té longitud de costat 6):





a) Seguint el patró recursiu de la *primera* representació, doneu i resoleu les recurrències que expressen l'alçada $H(n)$ i l'amplada $W(n)$, respectivament, de la graella necessària per representar un arbre binari complet de 2^n fulles.

b) Seguint el patró recursiu de la *segona* representació, doneu i resoleu la recurrència que expressa la longitud $L(n)$ del costat de la graella quadrada necessària per representar un arbre binari complet de 4^n fulles.

c) Si l'objectiu és minimitzar l'àrea de la graella usada per a un arbre binari complet de 4^n fulles, expliqueu usant raonaments asimptòtics quina de les dues construccions escolliríeu. Fixeu-vos que $4^n = 2^{2n}$.

5.3 MAX3SAT

El problema MAX3SAT es defineix així: Donada un fórmula booleana ϕ en forma normal conjunctiva, i amb exactament tres literals per clàsula (3-CNF), cal trobar una assignació de valors booleans a les variables de ϕ de forma que se satisfacin el màxim nombre de clàusules possible.

a) Doneu una assignació que maximitzi el nombre de clàusules de

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_3 \vee x_4).$$

b) Existeix algun algorisme polinòmic per resoldre MAX3SAT?

c) Doneu un algorisme eficient i senzill que, donada una fórmula en 3-CNF, en retorni una assignació que satisfaci, almenys, la meitat de les clàusules.

d) Demostreu que qualsevol fórmula en 3-CNF amb n clàusules té una assignació que en satisfà, almenys, $\frac{7}{8}n$ de les clàusules.

5.4 Flux màxim

Sigui G un graf no dirigit on les arestes tenen capacitats positives. Siguin x, y i z tres vèrtexs qualssevol de G. Suposeu que el flux màxim entre x i y és 100, i que el flux màxim entre y i z és 200. Doneu raonadament la fita inferior més ajustada i la fita superior més ajustada per al flux màxim entre x i z. Al vostre raonament, us pot servir la relació entre flux màxim i tall mínim.

5.5 Teoria de jocs

A partir de la definició dels nímbers, demostreu que qualsevol posició d'un joc imparcial és perdedora si i només si el seu nímber és zero.

5.6 Codi espatllat

En aquest problema i el següent, suposeu aquestes definicions de tipus:

```
typedef vector<int> VE;
typedef vector<VE> VVE;
```

Considereu el codi següent. A tot arreu, suposeu que M és una matriu 3×3 que només conté valors -1, 0 i 1, i que *juga* val 1 o -1.

```
map<VVE, int> R;

bool guanya(const VVE& M, int juga) {
  for (int i = 0; i < 3; ++i)
    if (M[i][0] == juga and M[i][1] == juga and M[i][2] == juga) return true;
  for (int j = 0; j < 3; ++j)
    if (M[0][j] == juga and M[1][j] == juga and M[2][j] == juga) return true;
  if (M[0][0] == juga and M[1][1] == juga and M[2][2] == juga) return true;
  if (M[0][2] == juga and M[1][1] == juga and M[2][0] == juga) return true;
  return false;
}

int f(const VVE& M, int juga) {
  auto it = R.find(M);
  if (it != R.end()) return it->second;

  int res = -juga;
  for (int i = 0; i < 3; ++i)
    for (int j = 0; j < 3; ++j)
      if (M[i][j] == 0) {
        VVE C = M;
        C[i][j] = juga;
        if (guanya(C, juga)) return R[M] = juga;
        int r2 = f(C, -juga);
        if (r2 == juga) return R[M] = juga;
        else if (r2 == 0) res = 0;
      }
  return R[M] = res;
}
```

a) Què pretén calcular la funció f? Expliqueu breument el seu funcionament.

b) Aquest codi no és correcte. Per què falla? Com es podria arreglar fàcilment i sense incrementar el cost asimptòtic afegint o modificant unes poques línies?

5.7 Més codis misteriosos

Sigui G un *VVE* que conté un graf no dirigit amb $n > 0$ vèrtexs i m arestes de la forma habitual: per a tot vèrtex x, $G[x]$ conté els vèrtexs veïns de x. Considereu aquest codi:

```
bool ok(int x, int c, const VVE& G, VE& color) {
  if (color[x]) return color[x] == c;
  color[x] = c;
  for (int y : G[x])
    if (not ok(y, 3 - c, G, color)) return false;
  return true;
}

void misteri(const VVE& G) {
  VE color(G.size(), 0);
  cout << (ok(0, 1, G, color) ? "si" : "no") << endl;
}
```

Què calcula *misteri* (G)? Com funciona? Quin cost té en el cas pitjor? Quin és un cas pitjor?

6

Parcial, 15 de novembre de 2016

6.1 Recurrència simpàtica

Trobeu la solució de la recurrència $T(n) = 7T(n/7) + \Theta(n)$ i demostreu-la per inducció.

6.2 Els més petits

Donat un vector amb n elements diferents, se'n vol trobar els k elements més petits, per a $k \in \{1, \ldots, n-1\}$.

a) Expliqueu amb prou detall, i analitzeu, un algorisme el més eficient possible en el cas pitjor per resoldre aquest problema.

b) Expliqueu amb prou detall, i analitzeu, un algorisme el més eficient possible en el cas mitjà per resoldre aquest problema.

c) Expliqueu els avantatges i incovenients dels dos algorismes.

6.3 Mínim local

Sigui X un vector amb $n \geq 2$ elements diferents. Un mínim local de X és una posició i tal que $X[i-1] > X[i] < X[i+1]$. La primera posició també pot ser un mínim local, si $X[0] < X[1]$. La darrera posició també pot ser un mínim local, si $X[n-2] > X[n-1]$.

a) Demostreu que X té, almenys, un mínim local.

b) Expliqueu amb prou detall, i analitzeu, un algorisme el més eficient possible per trobar un mínim local en X.

6.4 Ordenació

Demostreu que no pot existir cap algorisme d'ordenació de propòsit general que trigui temps lineal en mitjana, suposant que les $n!$ permutacions dels elements són equiprobables. Per fer-ho, demostreu que no existeix cap algorisme d'ordenació de propòsit general que funcioni en temps lineal per, almenys, la meitat de les possibles $n!$ entrades de mida n.

6.5 Codi misteriós

Considereu aquest codi, suposant $n \geq 1$:

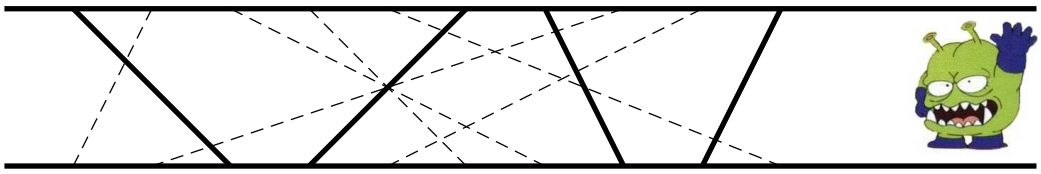
```
int n;
cin >> n;
vector<double> V(n + 1, 0);
for (int i = 1; i <= n; ++i) {
    cin >> V[i];
    V[i] += V[i-1];
}
sort(V.begin(), V.end());
double res = ∞;
for (int i = 1; i <= n; ++i) res = min(res, V[i] - V[i-1]);
cout << res << endl;
```

a) Quin cost té?

b) Si l'entrada és `5 -6 10 -8 -10 24` quina és la sortida?

c) En general, què calcula aquest codi?

6.6 El problema d'en Nikochan



El rei Nikochan vol travessar el túnel del dibuix des de la seva posició de la dreta fins a l'esquerra. Però un sistema automàtic tancarà tantes portes com pugui sense que es creuin entre si (al dibuix, quatre) per evitar que el monstre s'escapi. Cada porta i es defineix amb un parell d'enters (f_i, s_i) que indiquen la posició dels seus extrems a les parets del túnel. A partir de la informació de n portes, i assumint que no hi ha f_i's repetides ni s_i's repetides, cal trobar el màxim nombre de portes que es poden tancar al mateix temps.

Per exemple, amb les portes $(0,0)$, $(1,2)$ i $(3,3)$, totes tres es poden tancar alhora. Però amb les portes $(2,3)$, $(1,8)$ i $(5,1)$, només se'n pot tancar una (la que sigui).

Suposeu que teniu la informació de les n portes en un *vector*<*pair*<**int**, **int**>> V de mida n. Sense escriure codi ni pseudo-codi, però de forma prou clara i detallada, doneu un algorisme tan eficient com sigui possible per calcular el màxim nombre de portes que es poden tancar. Quin cost té la vostra solució? La solució esperada té cost $\Theta(n \log n)$.

7

Recuperació, 7 de juliol de 2016

7.1 Recurrències

Doneu fites asimptòtiques per a les recurrències

$$A(n) = 9A(n/4) + \mathrm{O}(n^2),$$
$$B(n) = 3B(n/3) + \Theta(\log^{3/2} n).$$

7.2 Escacs

Considereu el problema següent: Donada una posició vàlida del jocs dels escacs, en la qual toca jugar a les blanques, és aquesta posició guanyadora? És a dir, poden guanyar les blanques sempre, independentment de les jugades que facin les negres?

Expliqueu raonadament si aquest problema és o no a la classe **P**, si és o no a la classe **NP**, si és o no **NP**-complet, i si és o no indecidible.

7.3 Consultes en un llibre electrònic

Heu d'implementar un sistema per cercar paraules a les pàgines d'un llibre electrònic. Concretament, cal trobar eficientment una llista ordenada amb els números de totes les pàgines que contenen totes les paraules que l'usuari consulta.

Com a exemple petitó, suposeu que tenim un llibre amb quatre pàgines, i que cada pàgina conté els mots següents:

1 : *cada dia al mati canta el gall quiriquiqui*
2 : *la merda de la muntanya no fa pudor*
3 : *el nen canta a la gallina*
4 : *la gallina canta i el gall dorm*

Llavors, el resultat de la consulta $[el, gall, canta]$ hauria de ser $[1, 4]$, el resultat de la consulta $[canta]$ hauria de ser $[1, 3, 4]$, i el resultat de la consulta $[guinardo]$ hauria de ser la llista buida $[\,]$.

El sistema funciona en dues fases. En la primera fase (feta per l'editorial en un ordinador potent), es pre-processen d'alguna forma les pàgines del llibre. A la segona (feta en un dispositiu lector relativament lent), es llegeixen les consultes i s'escriuen les respostes. S'espera que hi hagi moltes pàgines, i totes amb moltes paraules. Es vol contestar molt ràpid cada consulta. Cadascuna de les consultes sol tenir molt poques paraules.

Fixem la nomenclatura següent:

M	=	nombre de pàgines
N	=	nombre total de paraules en totes les pàgines
n	=	nombre de paraules diferents en totes les pàgines
P	=	nombre màxim de pàgines on apareix cada paraula
k	=	nombre màxim de paraules en una consulta

Part 1: Considereu el cas on cada consulta conté una sola paraula ($k = 1$).

(a) Expliqueu quines estructures de dades i quin pre-procés usaríeu a la primera fase. Expliqueu com ho utilitzaríeu per tractar cada consulta.

(b) Quantifiqueu l'espai necessari per a les vostres estructures de dades.

(c) Quantifiqueu el temps necessari per a la primera fase de la vostra solució.

(d) Quantifiqueu el temps necessari per tractar cada consulta amb la vostra solució.

Part 2: Considereu ara el cas general ($k \geq 1$).

(e) Expliqueu com utilitzaríeu la mateixa estructura de dades de la Part 1 per processar cada consulta.

(f) Quantifiqueu el temps necessari per tractar cada consulta amb aquesta solució.

Observacions:

- Us pot ser útil descriure la vostra solució usant estructures de dades de la llibreria estàndard de C++. Però no doneu codi complet.
- Les solucions per a aquest problema que tinguin cost $\Omega(N)$ per a l'apartat (d) es consideraran insuficients i no es valoraran.

7.4 Codis misteriosos

En tots els apartats d'aquest problema, suposeu que V és un vector d'enters de mida n, amb $n \geq 2$ parell, i que $m = n/2$.

Considereu aquest codi:

```
priority_queue <int> Q;
for (int i = 0; i < m; ++i) Q.push(V[i]);
cout << Q.top() << endl;
for (int i = m; i < n; ++i) {
    Q.push(V[i]);
    cout << Q.top() << endl;
}
```

(a) Què escriu el codi si $n = 10$ i $V = [4, 8, 10, 9, 12, 2, 23, 1, 42, 7]$?

(b) Doneu una explicació d'alt nivell sobre quins valors s'escriuen amb un V qualsevol si els enters són tots diferents.

(c) Quin cost asimptòtic té el codi, en funció de n?

Considereu ara aquest codi:

```
set<int> S;
for (int i = 0; i < m; ++i) S.insert(V[i]);
cout << *S.begin() << endl;
for (int i = m; i < n; ++i) {
    S.insert(V[i]);
    S.erase(V[i-m]);
    cout << *S.begin() << endl;
}
```

(d) Què escriu aquest codi amb l'exemple de l'apartat (a)?

(e) Doneu una explicació d'alt nivell sobre quins valors s'escriuen amb un V qualsevol si els enters són tots diferents.

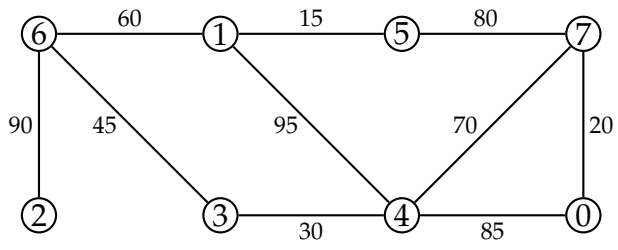
(f) Quin cost asimptòtic té el codi, en funció de n?

(g) Si els elements de V estiguessin repetits, alguns dels dos codis podria fer alguna cosa diferent a les mencionades en els apartats (b) i (e)? En els dos casos, si la resposta és afirmativa, mostreu un exemple; si la resposta és negativa, expliqueu per què.

(h) Escriviu en C++ un codi alternatiu a un dels dos codis donats, que calculi el mateix (suposant que tots els enters són diferents), però amb cost asimptòtic estrictament menor al calculat a l'apartat (c) o a l'apartat (f). Quin cost té aquest nou codi?

7.5 Algorisme de Kruskal

(a) Apliqueu l'algorisme de Kruskal al graf següent, amb prou detall perquè quedi clar què calcula i quin és el seu funcionament.



(b) Demostreu que l'algorisme de Kruskal és correcte per a qualsevol graf on els pesos de totes les arestes són diferents.

7.6 Teoria de jocs

Recordem el joc del Nim: Hi ha diverses piles de pedres, i dos jugadors, per torns, han de treure almenys una pedra d'una pila no buida; perd qui no pot jugar.

Considereu aquesta variant: per a cada pila, si té n pedres, llavors només se'n poden treure entre 1 i $\lceil n/2 \rceil$. Per exemple, si una pila té 20 pedres, se'n poden treure com a màxim 10, i si una pila en té 21, se'n poden treure com a màxim 11.

Suposeu una posició amb 9 piles, amb 1, 2, 3, 999, 4, 5, 6, 7 i 999 pedres respectivament. La posició és guanyadora o perdedora? Si és guanyadora, mostreu raonadament una jugada que guanyi.

8

Final, 19 de gener de 2016

9

Final, 19 de gener de 2016

9.1 Costos

Doneu raonadament el cost de les cinc funcions següents, suposant $n \geq 0$:

```
int f1 (int n) { // 0.25 punts
    int x = 0;
    for (int i = 0; i < n; ++i) ++x;
    return x;
}

int f2 (int n) { // 0.25 punts
    int x = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < i; ++j) ++x;
    return x;
}

int f3 (int n) { // 0.25 punts
    if (n == 0) return 0;
    return f3(n/2) + f1(n) + f3(n/2);
}

int f4 (int n) { // 0.25 punts
    if (n <= 1) return 1;
    return 2*f4(n-1) + 3*f4(n-2) + 8;
}

int f5 (int n) { // 0.5 punts
    int s = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < i*i; ++j)
            if (j%i == 0)
```

```
                for (int k = 0; k < n; ++k) ++s;
    return f1(s);
}
```

9.2 Heaps

Mostreu un max-heap amb 16 elements, tots diferents, on el cost d'esborrar el màxim del heap amb l'algorisme explicat a classe sigui el més petit possible.

9.3 Cicles i camins Hamiltonians

Considereu els dos problemes següents:

- HAM: Donat un graf G, conté un cicle Hamiltonià?
- CAM: Donat un graf G, conté un camí Hamiltonià?

Sabent que HAM $\in$ **NP-C**, demostreu que CAM $\in$ **NP-C**.

9.4 Inaproximabilitat

Considereu els dos problemes **NP**-complets següents:

- HAM: Donat un graf G, conté un cicle Hamiltonià?
- TSP: Donats un natural K i una matriu quadrada M amb n^2 naturals, existeix una permutació π de $\{1, \ldots, n\}$ tal que

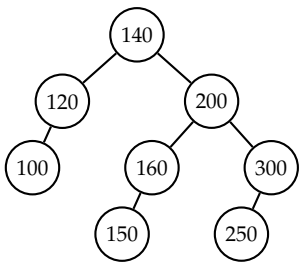
$$M[\pi_n, \pi_1] + \sum_{i=1}^{n-1} M[\pi_i, \pi_{i+1}] \leq K\,?$$

Demostreu que, si $\mathbf{P} \neq \mathbf{NP}$, llavors TSP $\notin$ **APX**.

Pista: Demostreu que, per a qualsevol $\rho \in \mathbb{Q}$, no existeix cap algorisme d'aproximació de factor ρ per a TSP (a no ser que $\mathbf{P} = \mathbf{NP}$). Utilitzeu una reducció a l'absurd utilitzant el fet que HAM és **NP**-complet.

9.5 Arbres aleatoritzats

Doneu tots els possibles arbres binaris de cerca resultat d'inserir, amb l'algorisme dels arbres aleatoritzats, un 170 en l'arbre següent:



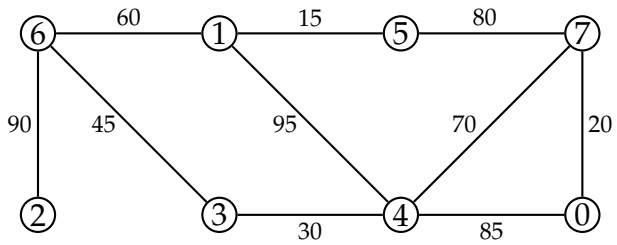
Amb quina probabilitat s'obté cadascun dels arbres?

9.6 Components fortament connexos

Considereu un graf dirigit G amb n vèrtexs, que no té arcs repetits ni arcs d'un vèrtex a ell mateix. Suposeu que G té exactament n components fortament connexos. Amb aquesta informació, digueu raonadament quin és el nombre màxim d'arcs m que pot tenir G.

9.7 Algorisme de Prim

a) Apliqueu l'algorisme de Prim al graf següent, amb prou detall perquè quedi clar què calcula i quin és el seu funcionament. Comenceu en el vèrtex 6.



b) Demostreu que l'algorisme de Prim és correcte en general. A la demostració, suposeu que tots els pesos de les arestes són diferents.

9.8 Flors i colors

Suposeu que teniu n classes de flors i n colors, i que sabeu de quins colors pot ser cada flor. Per exemple, podeu tenir $n = 3$, les flors campaneta, clavell i rosella, els colors blau, blanc i vermell, i aquestes $c = 5$ possibles combinacions: campanetes blaves, campanetes blanques, clavells blancs, clavells vermells, i roselles vermelles.

Voleu fer un ram amb el màxim nombre de flors, però sense repetir ni classes de flors ni colors. En l'exemple, podríeu crear un ram amb tres flors: una campaneta blava, un clavell blanc, i una rosella vermella. Però si els clavells no puguessin ser blancs, llavors només podríeu fer un ram amb dues flors.

Sigui f és el màxim nombre de flors que pot tenir el vostre ram. Expliqueu amb prou detall un algorisme per trobar f. Doneu una fita superior al cost temporal del vostre algorisme en funció de n, c i f.

10

Parcial, 3 de novembre de 2015

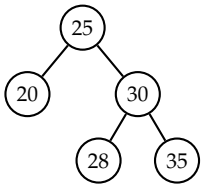
10.1 Pica-pica

1. Demostreu que per a qualssevol funcions $f, g : \mathbb{N} \to \mathbb{N}$,

   $$f(n) + g(n) = O(\max\{f(n), g(n)\}).$$

2. Suposeu que s'implementa un conjunt amb taules de dispersió, però no guardant els elements que col·lisionen en llistes, com es fa usualment, sinó en arbres AVL. Quins avantatges i desavantatges té en comparació?

3. Digueu quins són tots els possibles valors enters que provocarien una rotació doble si s'inserissin a l'arbre AVL següent. Inseriu-hi qualsevol d'aquests valors (digueu quin) i mostreu l'arbre AVL resultat.



10.2 Maleïdes recurrències

Teniu tres solucions dividir-i-vèncer per resoldre un problema de mida n:

- Dividint en 3 subproblemes de mida $n/2$, i combinant les subsolucions en cost $\Theta(n^2\sqrt{n}\,)$.
- Dividint en 4 subproblemes de mida $n/2$, i combinant les subsolucions en cost $\Theta(n^2)$.
- Dividint en 5 subproblemes de mida $n/2$, i combinant les subsolucions en cost $\Theta(n \log^2 n \log\log n)$.

Quina alternativa és la més eficient?

10.3 Element solitari

1. Considereu un vector amb n enters, amb n senar. Suposeu que cada element hi apareix exactament dos cops tret d'un element, que hi apareix exactament una vegada. Doneu un algorisme de cost linial per trobar l'element solitari.

   Per exemple, per a $[4, 7, 4, 1, 3, 3, 6, 1, 7]$, cal retornar 6.

2. Suposeu ara, a més, que el vector està ordenat. Doneu un algorisme el més eficient possible per trobar l'element solitari.

   Per exemple, per a $[1, 1, 3, 3, 4, 4, 6, 7, 7]$, cal retornar 6.

Analitzeu el cost en temps i en espai dels vostres dos algorismes.

10.4 En posició!

Donats un vector d'enters diferents v i un enter x, la funció següent examina en ordre (com a molt) les $n = v.size()$ posicions de v i retorna quina conté x, o bé retorna -1 si x no hi és.

```
int posicio (const vector<int>& v, int x) {
    for (int i = 0; i < v.size (); ++i) {
        if (v[i] == x) return i;
    }
    return -1;
}
```

Suposeu que la probabilitat que x estigui a $v[i]$ és $1/2^{i+1}$ per a $0 \leq i < n-1$, i que estigui a $v[n-1]$ és $1/2^{n-1}$. Com que aquestes probabilitats sumen 1, x sempre està present. Calculeu el cost asimptòtic de *posició*() en el cas mitjà sota aquesta distribució de probabilitats.

10.5 Comptatge

Descriviu amb prou detall (però sense escriure codi ni pseudocodi) un algorisme eficient que, donat un vector amb n enters $[x_1, \ldots, x_n]$, compti quantes posicions i, amb $1 \leq i \leq n$, compleixen la propietat

$$|\{j : 1 \leq j \leq n \text{ i } x_j \leq x_i\}| = i \ .$$

Analitzeu el cost en temps i en espai del vostre algorisme.

Per exemple, per a $[12, 14, 14, 17]$ cal retornar 3, i per a $[23, 33, 23]$ cal retornar 0.

10.6 Selecció

Ja sabem que, amb n elements diferents, l'algorisme de selecció per mediana de medianes triga temps $\Theta(n)$ en el cas pitjor quan s'utilitzen grups de 5 elements. Quin cost tindria en el cas pitjor si s'usen grups de 3 elements?

11

Recuperació, 26 de juny 2015

11.1 Punts propers a l'origen

Donats n punts a l'espai tridimensional, es vol saber quins són els k punts més propers a l'origen, amb $k = \Theta(n)$. Expliqueu (sense escriure gens de codi) un algorisme amb cost estrictament inferior a $\Theta(n \log n)$ per resoldre aquest problema.

11.2 Propietats dels algorismes d'ordenació

Ompliu (en un altre full) una taula com la següent que inclogui raonadament quines són les propietats principals d'aquests cinc algorismes per ordenar un vector de n elements:

	selection-sort	insertion-sort	merge-sort	quick-sort	heap-sort
Temps en cas pitjor					
Temps en cas millor					
Temps en mitjana					
Mida espai addicional					
És estable?					

[Una ordenació és estable si manté l'ordre relatiu que tenien originalment els elements amb claus iguals. És a dir, si dos elements x i y tenen la mateixa clau, i x està a l'esquerra de y abans d'ordenar, llavors x estarà a l'esquerra de y després d'ordenar.]

11.3 Grafs

Donat un graf no dirigit amb n vèrtexs i m arestes, totes amb pesos positius, i un vèrtex qualsevol x, cal trobar la distància més curta de x a la resta de vèrtexs. Suposeu $m = O(n)$. Quin cost en el cas pitjor tindria l'algorisme de Dijkstra sota aquestes condicions?

Suposeu ara que, a més, se sap que els pesos de les arestes són naturals entre 1 i 10. Proposeu un algorisme alternatiu al de Dijkstra que sigui asimptòticament més eficient.

11.4 Teoria de jocs

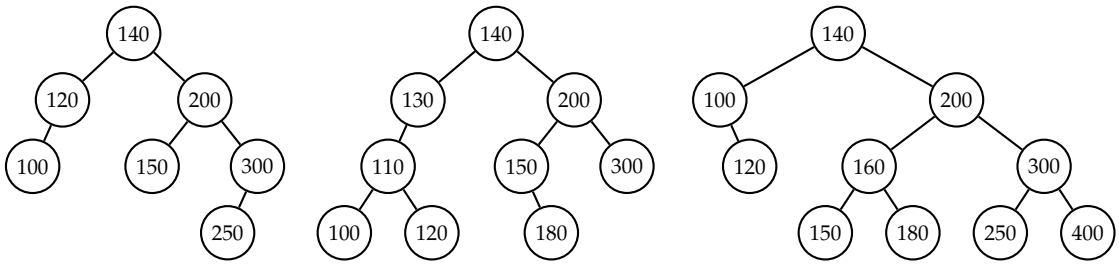
Recordem el joc del Nim: Hi ha diverses piles de pedres, i dos jugadors, per torns, han de treure almenys una pedra d'una pila no buida; perd qui no pot jugar.

Hi ha cinc piles amb el següent nombre de pedres: 12, 9, 3, 20, 16. Digueu raonadament totes les jugades guanyadores que hi ha, o si no n'hi ha cap.

11.5 Arbres AVL

a) Quin és el cost en el cas pitjor de trobar un element present en un AVL amb n nodes? I en el cas mitjà? Raoneu les dues respostes.

b) Dels arbres binaris següents, quins no són AVL? Per què?



11.6 Un altre codi espifiat

Considereu el programa següent (amb el preàmbul de les inclusions omès):

```
int main() {
    set<string> S;
    string s;
    while (cin >> s) {
        if (S.find(s) == S.end()) S.insert(s);
        else S.erase(s);
    }
    for (set<string>::iterator it = S.begin(); it <= S.end(); ++it) cout << *it << endl;
}
```

Què pretén fer aquest programa? El codi té un error. Quin? Com s'ha d'arreglar? Un cop arreglat, quin cost té aquest programa en el cas millor? I en el cas pitjor?

11.7 NP-completesa

Considereu el problema DIVISOR: Donat (en binari) un natural n, cal dir si existeix algun divisor x de n tal que $1 < x < n^{1/3}$. Ni es coneix cap algorisme polinòmic per resoldre DIVISOR, ni se sap si DIVISOR és **NP**-complet.

a) Si es demostrés $\mathbf{P} = \mathbf{NP}$, això implicaria DIVISOR $\in \mathbf{P}$, o bé DIVISOR $\notin \mathbf{P}$?

b) Si es demostrés $\mathbf{P} \neq \mathbf{NP}$, això implicaria DIVISOR $\in \mathbf{P}$, o bé DIVISOR $\notin \mathbf{P}$?

c) Si es demostrés que DIVISOR $\notin \mathbf{P}$, això implicaria $\mathbf{P} = \mathbf{NP}$, o bé $\mathbf{P} \neq \mathbf{NP}$?

d) Si es trobés un algorisme polinòmic per resoldre DIVISOR, això implicaria $\mathbf{P} = \mathbf{NP}$, o bé $\mathbf{P} \neq \mathbf{NP}$?

12

Final, 16 de gener de 2015

12.1 Recurrència

El nombre esperat de comparacions $C(n)$ fetes per l'algorisme de quicksort per ordenar un vector de n elements compleix la recurrència

$$C(n) = \sum_{i=1}^{n} \frac{1}{n} \left((n-1) + C(i-1) + C(n-i) \right) = (n-1) + \frac{2}{n} \sum_{i=1}^{n-1} C(i).$$

Demostreu per inducció que $C(n) = O(n \log n)$.

12.2 Arbres AVL

Assumiu que l'alçada d'un arbre amb un sol node és 1.

a) Quin és el cost en el cas pitjor de cercar un element en un arbre AVL amb $n2^n$ nodes?

b) Quin és l'arbre AVL resultat d'inserir les claus 1, 2, 3, 4, 5, 6 en aquest ordre en un arbre inicialment buit?

c) Quina és l'alçada màxima d'un arbre AVL amb 7 nodes? Mostreu un exemple.

12.3 Heaps ternaris

Un heap ternari és com un min-heap, però on els nodes tenen tres fills enlloc de dos.

Expliqueu com representaríeu un heap ternari en memòria (doneu la declaració de tipus *Heap3*) i com us mouríeu als fills i al pare d'un node donat.

Implementeu una funció **void** *insert* (*Heap3*& *h*, *Elem x*); que insereixi un element x en un heap ternari h i analitzeu-ne el cost.

12.4 NP-completesa

Proveu que CLICA i GRAN-CLICA es poden reduir polinòmicament entre si, en ambdós sentits:

CLICA: Donat un graf $G = (V, E)$ i donat un nombre natural k, cal determinar si G té un subgraf complet de k vèrtexs. És a dir, cal dir si existeix un subconjunt $U \subseteq V$ de k vèrtexs tal que per a tot parell de vèrtexs diferents $u, v \in U$, $\{u, v\} \in E$.

GRAN-CLICA: Donat un graf G= (V, E), cal determinar si G té un subgraf complet de $\lceil |V|/2 \rceil$ vèrtexs.

12.5 Ordenació topològica

Considereu un graf dirigit amb n vèrtexs que té exactament una ordenació topològica. Per exemple, per a $n = 1$ només tenim un possible graf, per a $n = 2$ en tenim dos, i per a $n = 3$ en tenim dotze. Exactament, quants grafs diferents hi pot haver, en funció de n?

12.6 Codi espifiat

Considereu el programa següent (amb el preàmbul de les inclusions omès):

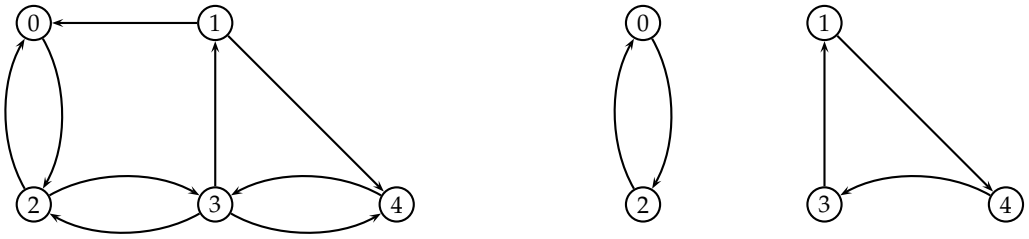
```
int repre(int x, vector<int>& pare) {
    return (pare[x] == -1 ? x : pare[x] = repre(pare[x], pare));
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> pare(n, -1);
    while (n > 1 and m--) {
        int x, y;
        cin >> x >> y;
        int rx = repre(x, pare);
        int ry = repre(y, pare);
        if (rx != ry) {
            pare[y] = x;
            --n;
        }
    }
    cout << (n == 1 ? "si" : "no") << endl;
}
```

Què pretén calcular aquest programa? El codi té un error. Quin? Com s'ha d'arreglar? Un cop arreglat, quin cost té aquest programa en el cas millor? I en el cas pitjor?

12.7 Max-flow

Donat un graf dirigit, es vol saber si es poden escollir n arcs de manera que cada vèrtex quedi inclòs en exactament un cicle de dos o més vèrtexs. Per exemple, a la dreta es pot veure una manera (única, en aquest cas) de descompondre en cicles el graf de l'esquerra:



Descriviu amb prou detall com resoldre aquest problema reduint-lo a un *max-flow*. Useu el graf de l'exemple a la vostra explicació. Pista: Construiu un nou graf amb els vèrtexs duplicats.

12.8 Teoria de jocs

Recordem el joc del Nim: Hi ha diverses piles de pedres, i dos jugadors, per torns, han de treure almenys una pedra d'una pila no buida; perd qui no pot jugar.

Algú diu que ha trobat una situació del joc on, dins de la mateixa pila, hi ha exactament dues jugades guanyadores. Si això és possible, mostreu un exemple i justifiqueu la vostra resposta. Si és impossible, expliqueu per què. La vostra resposta ha d'usar nímbers.

13

Parcial, 14 de novembre de 2014

13.1 Ordenació per selecció

Aquesta és una implementació de l'algorisme d'ordenació per selecció:

```
void sel_sort (vector<elem>& v) {
    int n = v.size ();
    for (int i = 0; i < n; ++i) {
        int m = i;
        for (int j = i + 1; j < n; ++j) {
            if (v[j] < v[m]) {
                m = j;                    (⋆)
        }   }
        swap(v[i], v[m]);
}   }
```

1. Calculeu exactament quantes assignacions d'elements es fan.
2. Calculeu exactament quantes comparacions entre elements es fan.
3. Assumint que l'entrada d'aquest codi és qualsevol de les permutacions de n elements de forma equiprobable, calculeu exactament quin és el nombre esperat de cops que s'executa la instrucció $(\star)$.

13.2 Recurrència

Demostreu que la recurrència $T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{3}{4}n\right) + O(n)$ té com a solució $T(n) = O(n)$.

13.3 Primer element repetit

Donat un vector de n naturals, s'hi vol trobar el primer element repetit. És a dir, cal trobar l'element que aparegui més d'una vegada tal que l'índex de la seva primera ocurrència sigui mínim. Si no hi ha cap element repetit cal retornar -1.

Per exemple, per a [10, 5, 3, 4, 3, 5, 6] cal retornar 5; per a [8, 6, 5, 4, 6, 120, 4, 8, 6] cal retornar 8; i per a [4, 7, 8, 2, 3] cal retornar -1.

1. Doneu un algorisme el més ràpid possible per resoldre aquest problema i analitzeu-ne el cost en temps i en espai, suposant que l'espai auxiliar està limitat a $\Theta(1)$.
2. Repetiu la pregunta anterior si la memòria addicional no està limitada.

13.4 Subvector de suma zero

Donat un vector de n enters, es vol determinar si conté alguna subseqüència no buida d'elements consecutius que sumi zero.

Per exemple, suposant que la primera posició té índex 0, per a [4, 2, -3, 1, 6] hi ha un subvector que suma zero des de l'índex 1 fins al 3; per a [4, 2, 0, 1, 6] hi ha un subvector que suma zero des de l'índex 2 fins al 2; i per a [4, 2, -4, 1, 6] no hi ha cap subvector que sumi zero.

Respongueu a les preguntes següents sense suposar res sobre els valors continguts en el vector. (Per exemple, si els nombres estan repetits, o si són petits en valor absolut, etc.)

1. Doneu un algorisme el més ràpid possible per resoldre aquest problema en el cas pitjor i analitzeu-ne el cost en temps i en espai.
2. Doneu un algorisme el més ràpid possible per resoldre aquest problema en el cas mitjà i analitzeu-ne el cost en temps i en espai.

13.5 Nombres de Fibonacci

Sigui F_n l'enèsim nombre de Fibonacci, definit com és habitual: $F_0 = 0$, $F_1 = 1$, i $F_n = F_{n-1} + F_{n-2}$ per a $n \geq 2$. Expliqueu amb prou detall però sense escriure codi com es pot calcular F_n en temps $O(\log n)$. El vostre algorisme no pot usar nombres reals.

13.6 Heaps

Considereu el heap següent:

47	41	31	37	29	13	19	17	11	23

Inseriu-hi un 43 i esborreu-ne el màxim. Feu les dues operacions independentment, és a dir, ambdues sobre el heap original. Mostreu els passos intermedis, amb una petita explicació.

13.7 Ordenació topològica

Considereu un graf dirigit amb n vèrtexs i sense cap cicle. Quin és el nombre màxim d'ordenacions topològiques que pot tenir? I el nombre mínim? Mostreu un exemple de cada.

14

Recuperació, 7 de juliol de 2014

14.1 Recurrència

Sigui $F(n) = 2F(n/2) + 2F(n/3) + \Theta(n^2)$. Demostreu que $F(n)$ és $O(n^2 \log n)$.

14.2 Interseccions

Es vol calcular la intersecció de K vectors de n elements arbitraris cadascun.

a) Doneu un algorisme el més eficient possible en el cas pitjor per resoldre aquest problema. Analitzeu el seu cost en temps i en espai.

b) Doneu un algorisme el més eficient possible en el cas mitjà per resoldre aquest problema. Analitzeu el seu cost en temps i en espai.

14.3 Anagrames

Es disposa d'una llista L de n paraules i una llista de consultes. Cada consulta és una paraula x i la seva resposta ha de ser totes les paraules de L que són anagrames de x.

Descriviu una estructura de dades per tal de respondre eficientment cada consulta. Descriviu com usar aquesta estructura de dades per respondre a una consula.

Analitzeu el cost en espai de la vostra estructura de dades. Analitzeu el cost en temps de la construcció de la vostra estructura de dades. Analitzeu el cost en temps del procés de cada consulta.

14.4 Creació de heaps

Doneu un algorisme el més eficient possible per convertir un vector de n elements arbitraris en un max-heap. Analitzeu el cost del vostre algorisme.

14.5 Quadrat d'una matriu

Per calcular el quadrat d'una matriu M de mida $n \times n$, l'algorisme obvi pren temps $O(n^3)$. Com segur que sabeu, existeixen algorismes més avançats de multiplicació de matrius que tenen un cost més baix, però no se'n coneix cap de cost $O(n^2)$. De totes formes, hom podria pensar que calcular el quadrat d'una matriu és més fàcil que fer el producte de dues matrius i, per tant, que existís algun algorisme $O(n^2)$ per calcular cuadrats de matrius.

Demostreu que si existeix un algorisme $O(n^2)$ per calcular quadrats de matrius $n \times n$, llavors també existeix un algorisme $O(n^2)$ per calcular productes de parells de matrius $n \times n$.

14.6 Arbres de cerca

Suposeu un arbre de cerca que a cada node té, a més d'una clau real x i dels punters al fill esquerra *esq* i al fill dret *dre*, un comptador m amb la mida (el nombre de nodes) del subarbre del qual x és arrel. Usant les definicions

```
struct node {
    double x;
    node* esq;
    node* dre;
    int m;
};
```

implementeu una funció **int** *posicio* (**double** y, *node** p); que retorni la posició de y dins de l'arbre apuntat per p. Per exemple, si l'arbre té 9 nodes, llavors si y n'és el mínim cal retornar 1, si n'és el màxim cal retornar 9, i si n'és la mediana cal retornar 5. Si y no es troba dins de p cal retornar -1.

La vostra funció ha de prendre temps proporcional a l'alçada de l'arbre.

14.7 Teoria de jocs

Considereu qualsevol joc que es pugui resoldre amb nímbers. Suposeu una posició del joc composta per m subjocs $S_1, S_2, \ldots, S_m$ tals que $n_1 > n_2 > \cdots > n_m$, on n_i denota el nímber del subjoc S_i. A més, suposeu que la posició total és guanyadora.

Sota aquestes hipòtesis, considereu l'afirmació "de totes les jugades guanyadores, n'hi ha almenys una dins del subjoc S_1". Si l'afirmació és certa, demostreu-la. Altrament, mostreu-ne un contraexemple.

15

Final, 16 de gener de 2014

15.1 Sumeu zero

Considereu que cadascuna de les estructures de dades següents emmagatzema n enters diferents. Per a cada poblema donat, expliqueu un algorisme el més eficient possible que el resolgui i analitzeu el seu cost en temps en el cas pitjor:

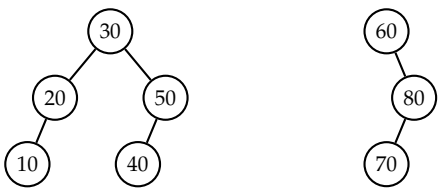
1. Donat un arbre AVL, hi ha algun element que sigui zero?
2. Donat un *max-heap*, hi ha algun element que sigui zero?
3. Donat un vector, hi ha dos elements que sumin zero?
4. Donat un vector ordenat, hi ha dos elements que sumin zero?
5. Donat un vector ordenat, hi ha dos elements que no sumin zero?
6. Donat un arbre binari de cerca, hi ha dos elements que sumin zero?
7. Donat un vector, hi ha tres elements que sumin zero?

15.2 Ordenació de vectors ordenadets

Diem que un vector de n elements es troba *ordenadet* si cada element no es troba a distància més gran o igual que $\lceil \log n \rceil$ de la seva posició un cop ordenat. Doneu un algorisme per ordenar vectors ordenadets. El vostre algorisme ha de ser el més eficient possible en el cas pitjor. Analitzeu el seu cost en temps i en espai.

15.3 Arbres de cerca

Mostreu raondament quins arbres es poden obtenir i amb quines probabilitats si es fusionen els arbres següents amb l'algorisme `join` dels arbres aleatoritzats explicat a classe:



15.4 Teoria de jocs

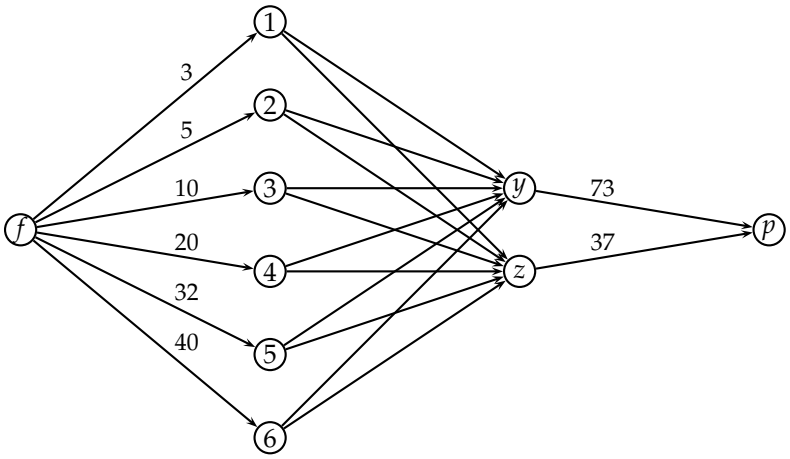
Recordem el joc del Nim: Hi ha diverses piles de pedres, i dos jugadors, per torns, han de treure almenys una pedra d'una pila no buida; perd qui no pot jugar. Considereu una variant amb una regla addicional: a cada torn, com a molt, es poden treure tres pedres.

Suposeu un joc que té nou piles amb aquestes pedres: 1, 7, 8, 21, 23, 25, 102, 884, 1000. La posició és guanyadora o perdedora? Per què? Si és guanyadora, mostreu de forma raonada una jugada que guanyi.

15.5 Fluxos

Recordem el problema *subset sum*: Donats n nombres naturals x_i i un natural y, cal decidir si hi ha algun subconjunt dels nombres que sumi exactament y. Sigui s la suma de tots els nombres, i sigui $z = s - y$. Considerem aquesta reducció a max-flow: Creem un graf amb $n + 4$ vèrtexs: un per a cada x_i, un per a y, un per a z, un per a la font i un altre per al pou. Connectem la font amb cada x_i amb un arc de capacitat x_i. Connectem y i z amb el pou amb arcs de capacitats y i z, respectivament. Finalment, connectem tots els x_i amb y i z, cadascun amb dos arcs de capacitat infinita.

Per exemple, si els x_i són 3, 5, 10, 20, 32 i 40, i $y = 73$, llavors $s = 110$ i $z = 37$, i aquest és el graf que s'obté (les capacitats ∞ no es mostren):



Ara n'hi ha prou de fer un *max-flow* sobre aquest graf per resoldre el problema original. En efecte, com que el problema té solució ($y = 3 + 10 + 20 + 40$), llavors tot el flux que arriba als vèrtexs 1, 3, 4 i 6 pot arribar fins a y, i la resta (els dels vèrtexs 2 i 5) pot arribar fins a z, de manera que el total $s = 110$ pot arribar fins al pou. Recíprocament, si el total $s = 110$ no pogués arribar fins al pou, seria perquè no existeix cap subconjunt de suma y que es pogués dirigir fins aquest vèrtex. Per tant, el problema original té solució si i només si el max-flow permet passar tot el flux s.

- Expliqueu per què aquesta reducció és incorrecta.
- Expliqueu quines conseqüències tindria que hi hagués una reducció correcta (i en temps polinòmic) de *subset sum* a *max-flow*.

15.6 Codi mort

En Genís està preparant un compilador optimitzador. Un dels objectius que té és eliminar codi mort, és a dir, codi que no pot mai ser executat independentment de l'entrada. Per exemple, en el programa següent,

```
void paritat (int n) {
    if (n%2 == 0) cout ≪ "parell";
    else if (n%2 == 1 or n%2 == −1) cout ≪ "senar";
    else cout ≪ "que estrany";                                    ⋆
}
```

la línia marcada amb un asterisc no es pot executar per a cap valor de n.

En general, doncs, en Genís vol fer un algorisme que, donat un programa p i una de les seves instruccions i, indiqui si i és codi mort o no.

Podeu donar un algorisme per resoldre el problema d'en Genís?

16

Parcial, 8 de novembre de 2013

16.1 Heu preparat la lliçó?

Definiu formalment $O(f)$, $\Omega(f)$ i $\Theta(f)$.

16.2 Polinomis

Considereu que un vector p amb n reals representa el polinomi $p(x) = \sum_{i=0}^{n-1} p[i]x^i$.

1. Descriviu un algorisme per avaluar un polinomi p en un punt real x donat. Analitzeu el cost del vostre algorisme.
2. Descriviu un algorisme de cost $\Theta(n^2)$ per multiplicar dos polinomis de grau $n-1$.
3. Mostreu com obtenir el polinomi que resulta de multiplicar dos polinomis linears $ax+b$ i $cx+d$ fent només tres productes. Pista: Un dels productes és $(a+b)(c+d)$.
4. Utilitzant la idea anterior, i suposant que n és una potència de dos, doneu un algorisme de cost $\Theta(n^{\log_2 3})$ per multiplicar dos polinomis de grau $n-1$. Justifiqueu el cost del vostre algorisme.

16.3 Heaps i heapsort

1. Implementeu una funció **bool** *is_max_heap*(**const** *vector*<**double**>& *v*); que digui si els elements de *v* formen un max-heap. Calculeu en detall el cost en temps de la vostra funció.
2. Implementeu una funció **void** *heapify* (*vector*<**double**>& *v*); que reorganitzi els elements de *v* en forma de max-heap, gastant només espai addicional $\Theta(1)$ i temps $\Theta(n)$. Calculeu formalment el cost en temps de la vostra funció.
3. Useu la funció anterior per implementar **void** *heapsort* (*vector*<**double**>& *v*); en temps $\Theta(n \log n)$ i espai $\Theta(1)$.

16.4 Comparacions del Quicksort

De les $n!$ permutacions de $\{1, \ldots, n\}$, quantes (i quines) d'elles provoquen que quicksort realitzi el nombre màxim de comparacions possibles?

16.5 Múltiples de 66 en un conjunt

Implementeu una funció **void** *elimina_multiples_de_66* (*set*<**int**>& *s*); que elimini tots els elements múltiples de 66 de *s* utilitzant espai addicional constant. Doneu el cost de la vostra funció en el cas millor i en el cas pitjor.

17

Final, 15 de gener de 2013

17.1 El mateix que el del parcial

Demostreu que el nombre esperat de comparacions per ordenar un conjunt S de n elements diferents amb QuickSort és $\Theta(n \log n)$.

Per concreció, considereu el següent algorisme aleatoritzat:

- Si $n < 2$, S ja està ordenat.
- Altrament,
  1. Es tria a l'atzar un element x de S.
  2. Tot comparant cada element y de S amb x, es construeixen els conjunts $S_1 = \{y \in S \mid y < x\}$ i $S_2 = \{y \in S \mid y > x\}$.
  3. S'ordenen S_1 i S_2 recursivament.
  4. Es retorna l'ordenació de S_1, seguida de x, seguida de l'ordenació de S_2.

17.2 Algorisme curiós

Considereu aquest algorisme: Donat un nombre natural $x \geq 1$ que inicialment té n bits, si x és parell el dividim per dos; altrament, fem que x passi a valer el seu nombre de bits (inicialment, n). Parem quan x arriba a 1.

1. En funció de n, quin cost té l'algorisme en el cas pitjor? Doneu un exemple de x amb aquest cost.
2. En funció de n, quin cost té l'algorisme en el cas millor? Doneu un exemple de x amb aquest cost.

17.3 Problemes petits i bonics sobre vectors

1. Donat un vector v amb n posicions, es vol saber si v conté tots els elements de 1 a n exactament una vegada.

   Per a cadascun dels algorismes següents, digueu si resol o no el problema i, si ho fa, doneu el seu cost en temps (CP: en el cas pitjor, CM: en el cas mitjà) i en espai (en el cas pitjor). Per respondre, ompliu una taula com la següent:

Algorisme	Funciona	Temps CP	Temps CM	Espai CP
a				
b				
c				
d				
e				
f				
g				
h		—		
i		—		
j				

   (a) Sumem tots els elements de v. Retornem cert ssi la suma és igual a $n(n+1)/2$.

   (b) Ordenem els elements de v amb l'algorisme d'ordenació per inserció, i verifiquem que la primera posició conté un 1, que la darrera conté una n, i que no hi ha dos elements consecutius repetits.

   (c) Com abans, però amb l'algorisme d'ordenació per fusió.

   (d) Com abans, però amb l'algorisme d'ordenació ràpida.

   (e) Creem un conjunt d'enters (*set*<**int**>) inicialment buit. Hi inserim cada element de v. Recorrem ordenadament el conjunt per comprovar que és igual a $\{1, \ldots, n\}$.

   (f) Creem un min-heap inicialment buit. Hi inserim cada element de v. Recorrem en pre-ordre el min-heap per comprovar que és igual a $\{1, \ldots, n\}$.

   (g) Creem una taula b de $n+1$ booleans, tots inicialitzats a fals. Per a cada element x de v, si $x < 1$ o $x > n$, retornem fals; sinó, si $b[x]$ retornem fals; sinó posem $b[x]$ a cert i continuem. Si no s'ha retornat mai fals, retornem cert.

   (h) Per a cada índex i de v, comprovem que $1 \leq v[i] \leq n$, i que, per a cada índex j de v amb $j > i$, $v[i] \neq v[j]$.

   (i) Com abans, però aquest cop utilitzem una cerca dicotòmica.

   (j) Per a cada permutació p de $[1, \ldots, n]$, si $p = v$, retornem cert. Si no s'ha retornat mai cert, retornem fals.

2. Donat un vector v amb $n-1$ posicions que només conté elements de 1 a n sense repeticions, es vol saber quin és l'element de 1 a n que falta. Proposeu un algorisme per resoldre aquest problema en temps lineal i espai constant.

3. Donat un vector v amb $n-2$ posicions que només conté elements de 1 a n sense repeticions, es vol saber quins són els dos elements de 1 a n que falten. Proposeu un algorisme per resoldre aquest problema en temps lineal i espai constant.

17.4 Agència matrimonial

En certes cultures s'admet que els homes tinguin més d'una dona. Suposeu que munteu una agència matrimonial, i que la informació de què disposeu és: el nombre d'homes h; el nombre de dones d; per a cada home, quines dones li agraden; per a cada dona, quins homes li agraden; i per a cada home, quantes dones vol com a màxim. L'objectiu es fer el màxim nombre possible de matrimonis entre persones que s'agradin l'una a l'altra.

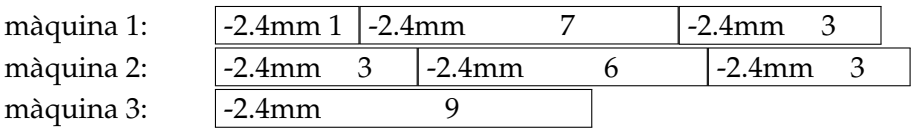
Expliqueu breument però amb prou detall com reduiríeu aquest problema a un explicat a l'assignatura. Mostreu el funcionament del vostre algorisme amb un exemple petit. Suposeu que els homes es numeren de 1 a h, i les dones de $h+1$ a $h+d$.

17.5 Tasques i màquines

Tenim m màquines idèntiques que han de processar n tasques. La tasca i-èsima triga t_i unitats de temps en ser processada. Cada màquina només pot processar una tasca en cada moment, i cada tasca s'ha de processar en una sola màquina. Estem interessats a distribuïr les n tasques a les m màquines, de forma que el temps final per completar totes les tasques sigui mínim.

La solució del problema és doncs una *assignació* de cada tasca a una màquina i a un temps d'inici, tot garantint les restriccions. El *temps de procés* d'una assigació és el temps necessari per completar totes les tasques segons aquella assignació.

Per exemple, per a tres màquines i set tasques que requereixen temps $6, 9, 3, 3, 7, 1, 3$ respectivament, el diagrama següent representa una assignació amb temps de procés 12, que es pot comprovar que és el mínim necessari.



1. Demostreu que la versió decisional d'aquest problema d'assignació de tasques és **NP**-completa.
2. Demostreu que l'estratègia següent és un algorisme d'aproximació de raó 2 per al problema d'optimització:
   (a) Ordenem de forma arbitrària les tasques.
   (b) En ordre, assignem cada tasca a la màquina que acabarà abans.

   Pista: Penseu en quin estat es troben les màquines quan comença l'última tasca.

18

Parcial, 13 de novembre de 2012

18.1 Ompliu els blancs

Ompliu els blancs de la forma més curta i precisa possible.

- Definiu formalment $O(f)$:

  ( )

- Si $T(n) = 2T(n-3) + n^2 - 2n + 1$, llavors $T(n) = \Theta($ ( ) $)$.

- Per ordenar un vector amb n elements, l'algorisme d'ordenació per inserció triga $\Theta($ ( ) $)$ passos en el cas millor i $\Theta($ ( ) $)$ passos en el cas pitjor.

- Si féssim una cerca dicotòmica partint el vector en tres trossos enlloc de dos i buscant en el terç adequat, el cost resultant seria $\Theta($ ( ) $)$.

- Suposeu que hi ha una manera de multiplicar dues matrius 3×3 amb 25 productes de reals i 50 sumes de reals. Assumint que n és una potència de 3, el cost que tindria un algorisme recursiu basat en aquest mètode per multiplicar matrius $n \times n$ seria $\Theta($ ( ) $)$.

18.2 Comparacions del QuickSort

Demostreu que el nombre esperat de comparacions per ordenar un conjunt S de n elements diferents amb QuickSort és $\Theta(n \log n)$.

Per concreció, considereu el següent algorisme aleatoritzat:

- Si $n < 2$, S ja està ordenat.

- Altrament,

1. Es tria a l'atzar un element x de S.
2. Tot comparant cada element y de S amb x, es construeixen els conjunts $S_1 = \{y \in S \mid y < x\}$ i $S_2 = \{y \in S \mid y > x\}$.
3. S'ordenen S_1 i S_2 recursivament.
4. Es retorna l'ordenació de S_1, seguida de x, seguida de l'ordenació de S_2.

18.3 L'Euclides no hi és

Probablement recordeu l'algorisme d'Euclides per calcular el màxim comú divisor (mcd) de dos nombres. En aquest problema estudiem un algorisme alternatiu i l'apliquem a nombres grans.

1. Verifiqueu les identitats següents:

$$\mathrm{mcd}(a,b) = \begin{cases} 2\mathrm{mcd}(a/2, b/2) & \text{si } a \text{ i } b \text{ parells,} \\ \mathrm{mcd}(a, b/2) & \text{si } a \text{ senar i } b \text{ parell,} \\ \mathrm{mcd}((a-b)/2, b) & \text{si } a \text{ i } b \text{ senars.} \end{cases}$$

2. Basant-vos en aquestes identitats, escriviu una funció **int** *mcd* (**int** *a*, **int** *b*) per calcular el màxim comú divisor de a i b. Analitzeu el cost del vostre algorisme en funció de a i de b.

3. Considereu que un nombre gran es representa amb un vector de booleans, amb la declaració **typedef** *vector*<**bool**> *bigint*;. Per guardar un natural x, la posició i del bigint corresponent emmagatzema l'i-èsim bit de més pes de x. Per exemple, 12 es representaria com a $[1,1,0,0]$.

   Escriviu una funció *bigint mcd* (*bigint a*, *bigint b*) per calcular el màxim comú divisor de a i b. Analitzeu el cost del vostre algorisme. Si v és un *bigint* i b és un boolèa, recordeu que les operacions v.*push_back*(b); i v.*pop_back*(); tenen cost constant.

18.4 La funció misteri ataca de nou

Considereu la funció següent:

```
int misteri (const vector<int>& v) {
    int n = v.size();
    int m = v[0];
    for (int i=1; i<n; ++i) {
        if (v[i] < m) {
            m = v[i];                    (⋆)
    }   }
    return m;
}
```

1. Digueu què calcula aquesta funció.
2. Quants cops s'executa la instrucció $(\star)$ en el cas millor i quin és aquest cas?
3. Quants cops s'executa la instrucció $(\star)$ en el cas pitjor i quin és aquest cas?
4. Considerant que v conté una permutació aleatòria de $[1,n]$, quin és el nombre esperat de cops que s'executa la instrucció $(\star)$?

18.5 Nombres aborrits

Un nombre és aborrit si només consisteix d'una xifra decimal repetida. Per exemple, 22, 666 i 88888888888888888888888888888888888888888888888888888888888888888888888888 són nombres aborrits. Doneu un algorisme de cost $\Theta(n)$ per multiplicar dos nombres aborrits de n xifres cadascun. Lògicament, considereu que guardeu cada nombre en un vector d'enters que corresponen als seus dígits.

18.6 Encara més misteris?

Què fa aquest programa? Quin cost té, tant en temps com en espai?

```
int main() {
    int n;
    cin >> n;
    set<int> S;
    S.insert (1);
    set<int>::iterator p = S.begin ();
    while (n--) {
        int x = *p;
        cout << x << endl;
        S.insert (2*x);
        S.insert (3*x);
        S.insert (5*x);
        ++p;
    }
}
```

19

Recuperació, 29 de juny de 2012

19.1 Miscel·lània

Contesteu de forma breu però precisa i raonada les preguntes següents:

a) Quin cost té aquest programa?

```
// pre: 0 ≤ e ≤ d < V.size()
void f (int e, int d, vector<int>& V) {
    if (e < d) {
        int m = (e + d)/2;
        intercanvia (V[e], V[m]);
        f(e, m, V);
        intercanvia (V[e], V[m]);
        f(e, m, V);
        intercanvia (V[m+1], V[d]);
        f(m + 1, d, V);
        intercanvia (V[m+1], V[d]);
}   }
```

b) Com escriuríeu el programa anterior perquè fes el mateix (molt) més ràpid?

c) Considereu el següent problema decisional per al joc dels escacs, en la posició inicial, encara sense cap moviment fet: "Blanques juguen. Guanyen?". Digueu si el problema és decidible o no, si és **NP** o no, i si és **P** o no.

d) Se sap que cert problema es pot resoldre amb un algorisme de cost $\Theta(n^{\log n})$, on n és la mida de l'entrada. Digueu si el problema és tractable (és a dir, si té solució de cost polinòmic), intractable, o alguna altra cosa.

e) (0'1 extra) Fa una setmana es va celebrar el centenari del naixement d'un famós informàtic. Qui?

19.2 Algorismes d'aproximació

Donat un graf no dirigit $G = (V, E)$ (amb $n = |V|$ i $m = |E|$), i un subconjunt $S \subseteq V$, $t(S)$ és el nombre d'arestes amb un extrem en S i l'altre fora de S. El problema del tall màxim (MAXCUT) consisteix a trobar un subconjunt $S \subseteq V$ tal que $t(S)$ sigui màxim. Se sap que la versió decisional de MAXCUT és un problema **NP**-complet.

Considereu l'algorisme aleatoritzat A següent: Començant amb $S = \emptyset$, per a cada vèrtex v es tira una moneda, i si curt cara, v es posa en S.

Demostreu que, en mitjana (sobre les tirades de la moneda), A és un algorisme d'aproximació de raó $\frac{1}{2}$.

19.3 Complexitat i indecidibilitat

Considereu els dos problemes següents:

- SAT: Donada una fórmula booleana sobre diverses variables, hi ha alguna assignació de valors cert o fals a les variables que faci que la fórmula sigui certa?
- ATURADA: Donat un programa (algorisme) A i una entrada x, s'atura A quan s'executa sobre l'entrada x?

Doneu una reducció en temps polinòmic de SAT a ATURADA. Sabent que SAT és **NP**-complet, podem deduir que ATURADA també ho és?

19.4 Programació dinàmica

La setmana passada va ser Sant Joan. La colla d'en Jonny, en Roy i la Steffy tenien previst passar la revetlla menjant coca, vevent cava i tirant coets. Els nostres amics van ser els responsables de comprar els coets. Entre ells va tenir lloc la conversa següent:

JONNY: He aconseguit recaptar P euros. La gent m'ha demanat que aquest any els coets facin molt de soroll, però que no en comprem cap de repetit.

ROY: Doncs en aquest catàleg hi ha N tipus de coets disponibles. Per a cada tipus hi figura el seu preu $p[i]$, el soroll que fa quan explota $s[i]$ i els grams de pólvora que conté $g[i]$.

JONNY: Perfecte! Mirem quins coets cal comprar per fer el màxim de soroll sense passar-nos del pressupost ni triar-ne dos del mateix tipus, agafem el cotxe de la Steffy i els comprem.

STEFFY: Ep, nois... Cal tenir a més en compte que en Felip Puig acaba de prohibir transportar més de G grams de pólvora en un vehicle privat!

a) Escriviu una recurrència (inclosos els casos base) per calcular el màxim soroll que es pot aconseguir triant entre els N coets de manera que en total costin P euros o menys i continguin G grams o menys de pólvora. Digueu el significat de cada paràmetre de la vostra recurrència i com es calcularia el resultat final del problema.

b) Expresseu en funció de N, P i G quin cost temporal tindria l'algorisme de programació dinàmica resultant. Raoneu d'on surt aquest cost.

Suposeu que el soroll que fan diversos coets és la suma dels sorolls individuals. Suposeu també que tots els valors són naturals. No escrigueu gens de codi. Gens!

19.5 Teoria de jocs

Demostreu que si X i Y són dos jocs imparcials, el nímber del joc resultant d'ajuntar X i Y és el "xor" del nímber de X i del nímber de Y.

19.6 Codi misteriós

Considereu el codi següent:

```
const int P = 1000003;

int modul(int x) { int r = x%P; return r < 0 ? r + P : r; }

int num(char c) { return c - 'a'; }

bool f(string s, string t) {
  int n = s.size();
  int m = t.size();
  int x = 0;
  for (int i = 0; i < n; ++i) x = modul(26*x + num(s[i]));
  int y = 0;
  for (int i = 0; i < n; ++i) y = modul(26*y + num(t[i]));
  if (y == x) return true;
  int pot = 1;
  for (int i = 0; i < n - 1; ++i) pot = modul(26*pot);
  for (int i = n; i < m; ++i) {
    y = modul(26*(y - pot*num(t[i-n])) + num(t[i]));
    if (y == x) return true;
  }
  return false;
}
```

Suposeu que tant s com t només contenen lletres minúscules, i que $n \leq m$.

Què retorna la funció f? Com funciona? Quin cost té en els casos millor i pitjor? Pot retornar **false** incorrectament? Suposant que n i m són "grans", amb quina probabilitat (aproximada) retorna **true** incorrectament?

20

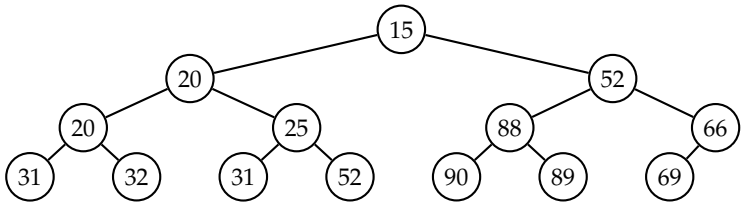
Final, 11 de gener de 2012

20.1 Codis de Huffman

Cal transmetre símbols entre 'a' i 'e', cadascun amb les probabilitats independents següents: $p_a = 0.34$, $p_b = 0.28$, $p_c = 0.18$, $p_d = 0.11$, i $p_e = 0.09$. Mostreu una possible codificació de Huffman. Expliqueu breument com l'heu obtinguda. Quants bits calen en mitjana per transmetre un missatge amb n símbols?

20.2 Heaps

a) Dibuixeu els dos heaps resultants d'afegir les claus 21 i 12 l'una després de l'altra en el heap següent:



b) Dibuixeu els dos heaps resultants d'esborrar el mínim dos cops (l'un després de l'altre) del heap anterior (abans de fer les insercions).

20.3 Teoria de jocs

Demostreu que si X i Y són dos jocs imparcials, el nímber del joc resultant d'ajuntar X i Y és el "xor" del nímber de X i del nímber de Y.

20.4 Estructures de dades

Sobre el conjunt $S = \{1, \ldots, n\}$, una relació $R \subseteq S \times S$ ve definida per un conjunt de m parells ordenats $C = \{(u_1, v_1), \ldots, (u_m, v_m)\}$, amb $u_i, v_i \in S$. Direm que u i v estan relacionats segons R si i només si $(u, v) \in C$.

a) Descriviu un algorisme i les seves estructures de dades associades per tal de saber si R és una relació simètrica en temps $O(m)$ en mitjana.

b) Descriviu un algorisme i les seves estructures de dades associades per tal de saber si R és una relació d'equivalència (reflexiva, simètrica i transitiva) en temps $O(n + m \log m)$ en el cas pitjor.

20.5 NP-completesa i programació dinàmica

Considereu els dos problemes següents (totes les entrades són nombres naturals):

- PARTICIÓ: Donats m nombres, es poden repartir en dos subconjunts disjunts que sumin el mateix?
- MOTXILLA: Tenim una motxilla que pot aguantar fins a P unitats de pes, i n objectes, cadascun amb un pes p_i i un valor v_i. Es pot escollir un subconjunt dels objectes de forma que la suma dels pesos no excedeixi P, i que la suma dels valors sigui almenys V?

a) Sabent que PARTICIÓ és un problema **NP**-complet, demostreu que MOTXILLA també ho és.

b) Utilitzant programació dinàmica, doneu una solució al problema MOTXILLA (escriviu la recurrència i el cas o casos base). Quin és el cost de la vostra solució en temps i en espai en el cas pitjor?

c) Per a la pregunta anterior, el gran professor Petit ha trobat un algorisme de cost $\Theta(nP)$. Es mereix un milió de dòlars per haver demostrat que **P** = **NP**?

20.6 Algorismes d'aproximació

Donat un graf no dirigit $G = (V, E)$ (amb $n = |V|$ i $m = |E|$), i un subconjunt $S \subseteq V$, $t(S)$ és el nombre d'arestes amb un extrem en S i l'altre fora de S.

El problema del tall màxim (MAXCUT) consisteix a trobar un subconjunt $S \subseteq V$ tal que $t(S)$ sigui màxim. Se sap que la versió decisional de MAXCUT és un problema **NP**-complet.

Considereu l'algorisme *cerca* següent:

```
S = ∅;
while (true) {
    if (hi ha algun u tal que t(S∪{u}) > t(S))   S = S ∪ {u};
    else if (hi ha algun u tal que t(S−{u}) > t(S))   S = S − {u};
    else return S;
}
```

a) Demostreu que l'algorisme *cerca* acaba.

b) Demostreu que l'algorisme *cerca* és un algorisme d'aproximació de raó 2 per al problema MAXCUT.

21

Parcial, 27 d'octubre de 2011

21.1 Ompliu els blancs

Ompliu els blancs de la forma més curta i precisa possible.

- Si $T(n) = 4T(n/2) + 3n^3 - 2n^2 + 1$, llavors $T(n) = \Theta($ $\square$).
- Si $T(n) = 3T(n-2) + n^2 - 2n + 1$, llavors $T(n) = \Theta($ $\square$).
- Qualsevol algorisme per trobar el k-èsim element més petit d'una taula de talla n necessita $\square$ passos.
- En un min-heap amb n elements, en el cas pitjor, consultar l'element mínim té cost $\square$, consultar l'element màxim té cost $\square$, inserir un element té cost $\square$ i esborrar el mínim té cost $\square$.
- Per ordenar un vector amb n elements tots iguals, l'algorisme de merge sort triga $\Theta($ $\square$) passos i l'algorisme de quick sort (amb la partició de Hoare) en triga $\Theta($ $\square$).
- Quick sort triga $\Theta($ $\square$) passos per ordenar un vector amb n elements si sempre tria com a pivot la mediana dels elements a particionar.

Dibuixeu una línia des de cadascuna de les cinc funcions en el centre al millor valor Ω de l'esquerra, i al millor valor O de la dreta.

$\Omega(1/n)$		$O(1/n)$
$\Omega(1)$		$O(1)$
$\Omega(\log\log n)$		$O(\log\log n)$
$\Omega(\log n)$		$O(\log n)$
$\Omega(\log^2 n)$		$O(\log^2 n)$
$\Omega(n/\log n)$	$1/(\log n)$	$O(n/\log n)$
$\Omega(n)$	$7n^5 - 3n + 2$	$O(n)$
$\Omega(n^{1'00001})$	$(n^2+n)/(\log^2 n + \log n)$	$O(n^{1'00001})$
$\Omega(n^2/\log^2 n)$	$2^{\log^2 n}$	$O(n^2/\log^2 n)$
$\Omega(n^2/\log n)$	3^n	$O(n^2/\log n)$
$\Omega(n^2)$		$O(n^2)$
$\Omega(n^{3/2})$		$O(n^{3/2})$
$\Omega(2^n)$		$O(2^n)$
$\Omega(5^n)$		$O(5^n)$
$\Omega(n^n)$		$O(n^n)$
$\Omega(n^{n^2})$		$O(n^{n^2})$

21.2 Costs

Considereu les funcions següents:

```
int cosa (int n) {
    int r = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < i; ++j) {
            for (int k = j; k < i + j; ++k) {
                ++r;
    }   }   }
    return r;
}

int farandula (int n) {
    if (n == 0) return 0;
    else return cosa(n) + 3 * farandula(n/3);
}
```

a) Quan triga *cosa*(n)?

b) Quan triga *farandula*(n)?

21.3 Sumar un

Per representar nombres naturals molts grans, es pot utilitzar una taula de bits. Per exemple, la taula $\langle 0, 1, 0, 1, 1\rangle$ representaria el natural 26.

Considereu l'algorisme obvi per afegir una unitat a un natural representat amb una taula de n bits.

a) Quin és el cost d'aquesta operació en el cas millor?

b) I en el cas pitjor?

c) I en el cas mitjà? (Suposeu que cadascún dels n bits és 1 amb probabilitat $\frac{1}{2}$.)

21.4 Heaps

Recordeu que les cues de prioritat de la STL estan implementades amb max-heaps.

Considereu el programa següent:

```
priority_queue <int> pq;
int x;
while (cin >> x) {
    pq.push(x);
}
while (not pq.empty()) {
    cout << pq.front () << endl;
    pq.pop ();
}
```

a) Digueu (utilitzant un màxim de dos verbs) què fa aquest algorisme.

b) Demostreu que aquest algorisme triga temps $\Theta(n \log n)$ en el cas pitjor.

c) Quin temps triga aquest algorisme en el cas millor? Quin és el cas millor?

21.5 Triangles en grafs

Un *triangle* en un graf no dirigit $G = (V, E)$ és un conjunt de tres vèrtexs distincts $u, v, w \in V$ tals que $\{u, v\} \in E$, $\{v, w\} \in E$, i $\{u, w\} \in E$.

Doneu un algorisme de temps lineal que, donat un graf no dirigit sense cicles de llargada 5 o més, indiqui si el graf conté algun triangle.

22

Recuperació, 5 de juliol de 2011

22.1 Cues de prioritats

Expliqueu (molt breument!) quines operacions principals té una cua de prioritats, amb quina estructura de dades se sol implementar i quin és el cost de les operacions amb aquesta estructura de dades.

22.2 Anàlisi

a) Analitzeu el cost de la funció següent:

```
int intrigant (int n) {
    int k = 1 + n%2;
    int x = 0;
    for (int i = 1; i ≤ n; i += k)
        ++x;
    for (int j = 1; j ≤ n; ++j)
        for (int i = 1; i*i ≤ n; ++i)
            ++x;
    for (int j = 1; j ≤ n; ++j)
        for (int i = 1; i ≤ n; i += i)
            ++x;
    return x;
}
```

b) Analitzeu el cost de la funció següent:

```
int confus (int n) {
    if (n == 0) return 2;
    else return n*n*intrigant(n) + confus(n/2);
}
```

22.3 NP-completesa

Considereu els dos problemes següents:

a) Donats n objectes amb pesos $p_1, \ldots, p_n$ i una alforja amb capacitat suficient, cal decidir si es poden repartir equitativament els objectes en les dues bosses de l'alforja.

   Recordeu que una alforja és un sac obert pel mig i tancat pels caps, els quals formen dues bosses grosses, ordinàriament quadrangulars. Aleshores, es demana si es poden repartir tots els objectes entre les dues bosses de manera que cada objecte s'ha de ficar en una bossa i les dues bosses porten exactament la mateixa càrrega.

b) Donats n objectes amb pesos $p_1, \ldots, p_n$ i una motxilla amb capacitat C, cal decidir si amb exactament tres objectes es pot carregar la motxilla completament (és a dir, si la suma dels pesos dels tres objectes és C).

Sota la hipòtesi $\mathbf{P} \neq \mathbf{NP}$, digueu raonadament, per a cadascun d'aquests problemes, si pertany o no a la classe **P**.

22.4 Indecidibilitat

El problema de l'aturada consisteix a determinar si, donat un programa i una entrada, aquest programa s'atura o no amb aquesta entrada.

El problema de l'aturada total consisteix en determinar si, donat un programa, aquest programa s'atura o no sobre totes les entrades.

Sabent que el problema de l'aturada és indecidible, què podeu dir del problema de l'aturada total?

22.5 Aproximació

El problema *Minimum makespan schedulling* (MMS) és un problema **NP**-difícil que es formula així:

> Donats els temps de procés $p_1, \ldots, p_n$ de n treballs, i m màquines idèntiques, cal trobar una assignació dels n treballs a les m màquines de forma que el temps de finalització de tots els treballs sigui mínim.

a) Sigui OPT el temps de finalització mínim. Demostreu les fites inferiors següents:

- $\max_{i \in 1..n} p_i \leq OPT$.
- $\frac{1}{m} \sum_{i \in 1..n} p_i \leq OPT$.

Considereu l'algorisme següent:

1. Ordenem els treballs arbitràriament.
2. Assignem els treballs a les màquines en aquest ordre, tot assignant el següent treball a la màquina que ha rebut menys quantitat de treball fins al moment.

b) Considereu una assignació obtinguda a través d'aquest algorisme. Sigui t un instant de temps on totes les màquines estan ocupades. Demostreu que

$$t \leq \frac{1}{m} \sum_{i \in 1..n} p_i.$$

c) Demostreu que l'algorisme donat és un algorisme d'aproximació de factor 2 per al problema del MMS.

22.6 Grafs

Considereu l'algorisme de Kruskal per trobar un arbre generador mínim d'un graf no dirigit amb pesos positius a les arestes. Suposeu que els pesos són tots diferents, i que les arestes ja venen ordenades per pes. Sabent que el graf és complet (és a dir, té n vèrtexos i $n(n-1)/2$ arestes), quin cost tindria una execució en el cas pitjor? Quina operació domina el cost i quin cost té? I en el cas millor? Mostreu un exemple de cada amb $n = 5$.

22.7 Arbres

Usant la definició

```
struct Node {
    Node* esq;
    Node* dre;
};

typedef Node* Punter;
```

feu una funció **int** *distancia_mes_curta* (*Punter p*); que, donat un punter p a l'arrel d'un arbre, en retorni la distància més curta (mesurada en nombre de nodes) fins a una fulla. Per exemple, un arbre amb un sol node té distància 1. Escriviu la funció recursivament.

Quin cost té la vostra funció?

Sense escriure codi, expliqueu com es podria calcular millor la distància, visitant només els nivells imprescindibles de l'arbre. La vostra nova solució pot ser recursiva o iterativa. Quin cost tindria aquesta nova funció en el cas millor?

22.8 Més arbres

Considereu els arbres binaris de cerca aleatoritzats, amb una modificació: Ara, a cada pas de les insercions s'usa sempre probabilitat 0.5 per decidir si el nou element es guarda en aquella posició o no. (Si s'arriba a un subarbre buit, l'element es guarda allà.)

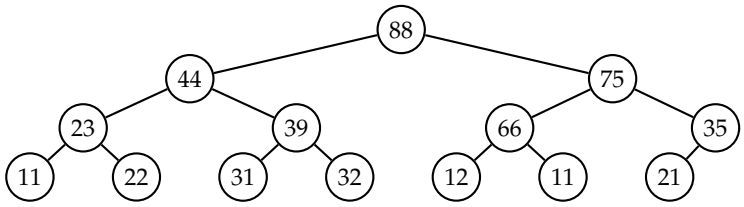
Mostreu la distribució de probabilitats dels arbres obtinguda després de la inserció de tres elements. És igual a la dels arbres aleatoris?

23

Final, 11 de gener de 2011

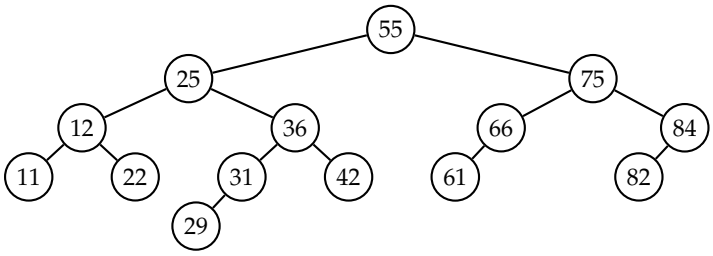
23.1 Heaps i AVLs

a) Dibuixeu els dos heaps resultants d'afegir les claus 13 i 99 l'una després de l'altra en el heap següent:



b) Dibuixeu els dos heaps resultants d'esborrar el màxim dos cops (l'un després de l'altre) del heap anterior (abans de fer les insercions).

c) Dibuixeu els tres arbres AVL resultants d'afegir les claus 35, 34 i 40 l'una després de l'altra en l'arbre AVL següent:



23.2 Anàlisi

a) Analitzeu el cost de la funció següent:

```
int misteri (int n) {
    int x = 0;
    for (int i = 1; i ≤ n; i += 2)
        ++x;
    for (int i = 1; i ≤ n; ++i)
        for (int j = 1; j ≤ i; ++j)
            for (int k = 1; k ≤ j; ++k)
                ++x;
    for (int i = n; i ≥ 1; i /= 2)
        for (int j = 1; j ≤ n; j++ )
            ++x;
    return x;
}
```

b) Analitzeu el cost de la funció següent:

```
int rebuscat (int n) {
    if (n == 0) return 42;
    else return misteri(n) + 66*rebuscat(n − 1);
}
```

23.3 NP-completesa

Considereu els dos problemes següents:

- 3-COLORABILITAT: Donat un graf, es poden pintar els seus vèrtexs usant com a molt 3 colors diferents de manera que no hi hagi dos vèrtexs veïns del mateix color?
- 2-COLORABILITAT: Donat un graf, es poden pintar els seus vèrtexs usant com a molt 2 colors diferents de manera que no hi hagi dos vèrtexs veïns del mateix color?

Sabent que 3-COLORABILITAT és un problema **NP**-complet, responeu de forma breu però raonada si cadascuna de les afirmacions següents és certa, és falsa, o si no es pot assegurar ni una cosa ni l'altra:

a) 2-COLORABILITAT es pot reduir polinòmicament a 3-COLORABILITAT.

b) 3-COLORABILITAT es pot reduir polinòmicament a 2-COLORABILITAT.

23.4 Indecidibilitat

El darrer teorema de Fermat, afirma que l'equació diofàntica $x^n + y^n = z^n$ no té cap solució entera per a $n > 2$, essent x, y i z diferents de zero.

Aquest és un dels teoremes més famosos de la història de les matemàtiques, i fins l'any 1995 no se'n va trobar una demostració. El matemàtic francès Pierre de Fermat fou el primer a proposar el teorema, però malauradament la demostració que suposadament havia realitzat no s'ha trobat mai. Fermat només va deixar escrit en un marge de la seva còpia de l'Aritmètica de Diofant el plantejament del teorema i l'afirmació que havia trobat una demostració del teorema. En les seves pròpies paraules:

> *Cubum autem in duos cubos, aut quadrato-quadratum in duos quadrato-quadratos, et generaliter nullam in infinitum ultra quadratum potestatem in duos eiusdem nominis fas est dividere cuius rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.*

Mostreu (no cal que useu un estret marge de pàgina) com reduir el problema de Fermat al problema de l'aturada.

23.5 Aproximació

Recordeu el problema de VERTEX COVER (que en versió decisional és **NP**-complet):

> Donat un graf connex $G = (V, E)$, cal determinar un subconjunt $S \subseteq V$ de talla mínima tal que tota aresta $e \in E$ contingui almenys un element de S.

Considereu l'algorisme següent:

1. Executem un recorregut en profunditat (DFS) en el graf G.
2. Retornem els vèrtexs que no són les fulles de l'arbre de DFS (incloent l'arrel).

a) Demostreu que la sortida d'aquest algorisme és realment un cobriment de vèrtexs.

b) Demostreu que l'algorisme proposat és un algorisme d'aproximació de factor 2 per al problema del VERTEX COVER.

23.6 Grafs

Sigui G un graf dirigit amb costs als arcs. Considereu l'algorisme de Dijkstra per trobar el cost mínim d'anar des d'un vèrtex x a cadascun dels altres vèrtexs. Suposeu que G té arcs amb cost negatiu, però que no té cap cicle amb cost negatiu.

a) Mostreu un exemple on l'algorisme falli.

b) Sigui $-c$ el cost negatiu més gran en valor absolut dels arcs de G. Suposeu que sumem c a totes els arcs, per fer-los no-negatius. Sigui G' el graf resultant. Justifiqueu si aplicar l'algorisme de Dijkstra a G' ens permet trobar els costs mínims de G.

Les vostres respostes han de ser prou detallades perquè sigui evident que sabeu com funciona l'algorisme de Dijkstra.

23.7 Llistes

Considereu una llista d'enters ordenada de forma estrictament creixent. Usant la definició

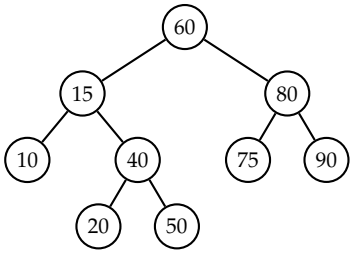
```
struct Node {
    int elem;
    Node* seguent;
};

typedef Node* Punter;
```

feu una funció *Punter esborra*(**int** *x*, *Punter p*); que, donats un enter x i una llista identificada amb un punter p al seu primer node, retorni la llista un cop se li ha esborrat el node que conté x. Si x no hi és, cal retornar la llista tal qual. Podeu escriure la funció recursivament o iterativament.

23.8 Arbres

Considereu els arbres binaris de cerca aleatoritzats. Suposeu que en un determinat moment l'arbre emmagatzemat és



Mostreu quins arbres es poden obtenir, i amb quina probabilitat, quan a l'arbre anterior se li insereix un 30 amb l'algorisme aleatoritzat.

24

Parcial, 28 d'octubre de 2010

24.1 Ompliu els blancs

Ompliu els blancs de la forma més curta i precisa possible.

- Definiu $O(f)$:

  [ ]

- El teoreme mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma $T(n) = aT(n/b) + \Theta(n^k)$ amb $b > 1$ i $k \geq 0$, llavors, fent $\alpha = \log_b a$,

$$T(n) = \begin{cases} [\quad] & \text{si } \alpha < k, \\ [\quad] & \text{si } \alpha = k, \\ [\quad] & \text{si } \alpha > k. \end{cases}$$

- Si $T(n) = 3T(n/2) + n^2 - 2n + 1$, llavors $T(n) = \Theta($ [ ]).

- Si $T(n) = 2T(n-1) + n^2 - 2n + 1$, llavors $T(n) = \Theta($ [ ]).

- Mergesort ordena n elements en temps [ ] en el cas pitjor, en temps [ ] en el cas mitjà, i en temps [ ] en el cas millor. L'espai auxiliar requerit és [ ] .

- Quicksort ordena n elements en temps [ ] en el cas pitjor, en temps [ ] en el cas mitjà, i en temps [ ] en el cas millor.

- Insertionsort ordena n elements en temps [ ] en el cas pitjor, en temps [ ] en el cas mitjà, i en temps [ ] en el cas millor.

- Per multiplicar dos nombres naturals grans eficientment podem fer servir l'algorisme de ( ) .
- Tot algorisme per trobar la mediana de n elements necessita ( ) passos.
- Un algorisme d'ordenació és estable si

  ( )

24.2 Ordenació

Escriviu en C++ un procediment de cost $O(n)$ que ordeni una taula de n enters sabent que tots ells es troben entre 0 i nK (K és algun valor constant conegut). Quin és l'espai requerit pel vostre algorisme?

Expliqueu perquè aquest algorisme no contradiu el fet de que qualsevol algorisme de propòsit general necessita $\Omega(n \log n)$ passos per ordenar n valors.

24.3 Dividir i vèncer

En una habitació fosca tenim n cargols, tots d'un ample diferent, i les seves n femelles corresponents. Volem aparellar cada cargol amb la seva femella corresponent, però a causa de la foscor no es poden comparar els cargols amb els cargols ni les femelles amb les femelles. L'única comparació possible és intentar cargolar un cargol a una femella per comprovar si és massa gran, si és massa petit o si encaixa perfectament.

Descriviu un algorisme per resoldre aquest problema en $\Theta(n \log n)$ passos en el cas mitjà.

24.4 Primalitat

Expliqueu molt breument l'algorisme clàssic que, donat un natural x, determina si x és un nombre primer o no. Quin és el cost del vostre algorisme en funció de x? Expliqueu perquè, malgrat les aparències, aquest algorisme és de cost exponencial. Podem doncs concloure que determinar si un nombre és primer és un problema intractable?

(Recordeu que es diu que un problema és intractable si no existeix cap algorisme de cost polinònic per resoldre'l.)

24.5 Producte de matrius

L'algorisme de Strassen és un algorisme de dividir-vèncer per multiplicar dues matrius $n \times n$ en $O(n^{2'81})$ passos. La idea clau és que dues matrius 2×2 es poden multiplicar usant només set multiplicacions enlloc de les vuit usuals, i que aquesta idea es pot adaptar recursivament.

a) L'algorisme de Strassen presentat a classe assumeix que n és una potència de 2. Com modificarieu l'algorisme quan n no sigui necessàriament una potència de 2 tot mantenint el seu cost asimptòtic?

b) Suposeu que trobéssim una variant de l'algorisme de Strassen basat en el fet que podem multiplicar matrius 3×3 utilitzant m multiplicacions enlloc de les 27 usuals. Com de petit hauria de ser m per tal que aquest algorisme sigui més eficient que l'algorisme de Strassen?

c) L'algorisme de Coppersmith i Winograd multiplica matrius $n \times n$ en $O(n^{2'376})$ passos. Com de petit hauria de ser m per tal de millorar aquest cost?

Part II

Solucions

1

Recuperació, 29 de juny de 2018

1.1 Misteri

a) En el cas pitjor, el codi executa dos bucles amb n iteracions cadascun, totes de cost constant. Per tant, el cost és $\Theta(n)$.
b) Comprova si b és igual que a, potser amb els caràcters moguts (cíclicament) cap a un costat.
c) Amb caràcters repetits, el codi podria fallar, dient que b no és equivalent a a quan sí que ho és. Per exemple, si a val "xxy" i b val "xyx".

1.2 Suma de dos elements

Una possibilitat de resoldre aquest problema en temps lineal es fer servir una taula de hash H. Els elements de V s'insereixen un a un a H. Cada vegada que s'insereix (amb cost mitjà $\Theta(1)$) un nou element amb valor y es comprova si a H hi ha l'element amb valor $x - y$ (en cost mitjà $\Theta(1)$). Aquest procediment es fa com a molt una vegada per cada element de V, per tant el cost mitjà total és $\Theta(n)$.

1.3 Heaps

Aquesta és l'evolució del vector:

[4, 1, 3, 2, 16, 9, 10, 14, 8, 7]

[4, 1, 3, 14, 16, 9, 10, 2, 8, 7]

[4, 1, 10, 14, 16, 9, 3, 2, 8, 7]

[4, 16, 10, 14, 7, 9, 3, 2, 8, 1]

[16, 14, 10, 8, 7, 9, 3, 2, 4, 1]

1.4 NP-completesa

a) Fals. Com que no sabem si X és a NP, no podem garantir que es pugui reduir a Y en temps polinòmic.
b) Fals. Com que no sabem si Y és a NP, no podem concloure que sigui NP-complet. De fet, tampoc s'especifica si la reducció de X a Y és en temps polinòmic.
c) Fals. No podem dir res sobre Y, excepte que es redueix a X.

1.5 Problema de l'aturada

El problema és decidible. En efecte, sigui N tal que per a tota $n \geq N$ existeix almenys un programa que para sempre, per exemple (hem omès les inclusions):

```
int main() {
  cout << "";
}
```

Sigui N la mida d'aquest programa. A dintre de la cadena "" podem posar qualsevol cadena més llarga, així que sempre hi ha algun programa que pararà per a tota $n \geq N$.

Per a cadascuna de les $n < N$, o bé hi ha algun programa de mida n que para sempre, o bé no n'hi ha cap. Considerem aquest programa:

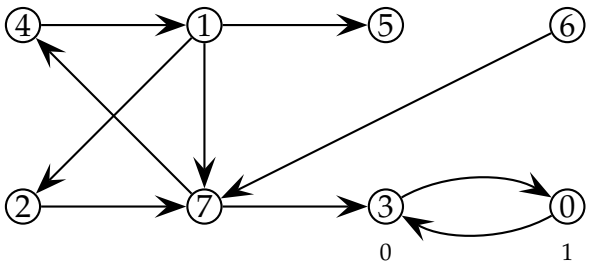
```
int n;
cin >> n;
if (n >= N) cout << "si" << endl;
else if (n == 0) cout << "no" << endl;
else if (n == 1) cout << "no" << endl;
 ...
else if (n == N - 2) cout << "no" << endl;
else cout << "no" << endl; // N - 1
```

A partir d'aquest programa en podem contruir 2^N, fent totes les combinacions de si/no (excepte per a la línia amb $n >= N$). Encara que (els humans) potser no sapiguem quina és la bona, exactament una l'és. Per tant, existeix un programa que ho decideix, i el problema és decidible.

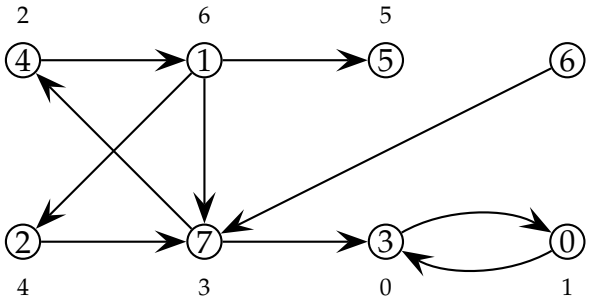
1.6 Components fortament connexos

En les explicacions que segueixen, per fixar un criteri, suposarem que els recorreguts es fan triant sempre el vèrtex més petit en tots els passos.

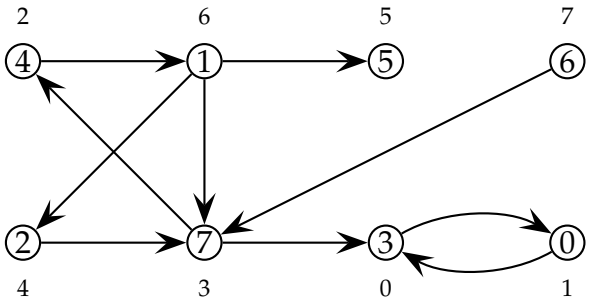
Primer, calculem el graf invers. Després, fem un recorreguts en profunditat, marcant amb comptadors de temps els vèrtexs a mesura que sortim d'ells. Primer, marquem el 3 i el 0:



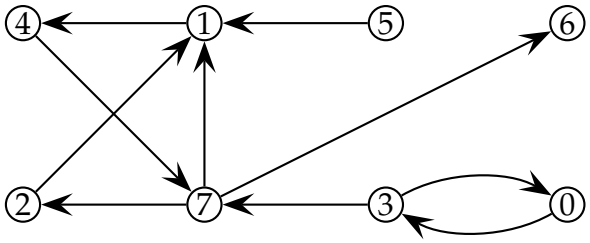
Després, a partir de l'1 marquem la resta de vèrtexs menys el 6:



Finalment, marquem el 6:



Ara tornem al graf original:

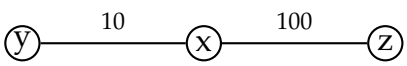


Hi fem recorreguts (en profunditat, per exemple), usant els comptadors de temps de gran a petit per triar per quin vèrtex comencem. Començant pel 6, que té el comptador de temps màxim, marcarem només el 6. Usant després l'1, marcarem l'1, el 4, el 7 i el 2. A partir del 5, el marcarem només a ell. Finalment, a partir del 0, trobarem el 0 i el 3.

Així doncs, i només amb cost linial en el nombre de vèrtexs i arestes, hem trobat els quatre components fortament connexos del graf: $\{6\}$, $\{1,2,4,7\}$, $\{5\}$ i $\{0,3\}$.

1.7 Tall mínim

Sí que és possible. Per exemple:



1.8 Teoria de jocs

Sabem que una posició és perdedora si i només si el seu nímber és 0. Anomenem X i Y les posicions, i nx i ny els seus nímbers. La demostració és per inducció en la mida conjunta d'X i Y. Si X i Y són buides, la composició també ho és. Així doncs, per al cas base es compleix que la composició és perdedora i els nímbers són iguals (en aquest cas, a 0).

Suposem que la propietat és certa fins a una mida determinada, i considerem la mida següent. Si $nx = ny$, per definició de nímber, qualsevol jugada que fem a X anirà a una posició amb nímber diferent a nx. (I similarment amb Y.) Per tant, per H.I. la composició resultat serà guanyadora. Com a conclusió, la posició actual és perdedora.

Altrament, i sense pèrdua de generalitat, suposem que $nx > ny$. Per definició de nímber, hi ha almenys una jugada a X que ens du a una posició amb nímber ny, la composició de la qual amb Y, per H.I., és perdedora. Per tant, la posició actual és guanyadora.

1.9 Què calcula?

El codi escriu 1, 2, 3, 5, 7, 11, ...

El seu cost és $\Theta(\sum_{x=1}^{N} \sum_{i=x}^{N} 1) = \Theta(N^2)$.

El codi resol un cas particular del problema de "el canvi de monedes". Aquí, estem suposant que hi ha un nombre infinit de monedes de cada tipus, i estem calculant de quantes maneres podem formar cada canvi.

Per exemple, amb $x = 4$ en tenim cinc:

$4 = 4$, $4 = 3 + 1$, $4 = 2 + 2$, $4 = 2 + 1 + 1$, $4 = 1 + 1 + 1 + 1$.

2

Final, 18 de gener de 2018

2.1 Recurrències

Una possibilitat seria aquest codi per a $n \geq 1$:

```
void escriu(int n) {
  cout << ' ' << n;
  if (n > 1) {
      escriu(n − 1);
      escriu(n − 1);
} } }
```

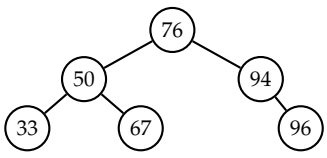
Sigui $T(n)$ el temps de la funció *escriu* () aplicada a una entrada $n \geq 1$. Llavors, tenim $T(n) = 2T(n-1) + \Theta(1)$, que és $\Theta(2^n)$ utilitzant el teorema mestre.

2.2 Els k més petits, en ordre

Una possibilitat consisteix a transformar el vector en un min–heap amb cost $\Theta(n)$ (fent servir l'algorisme "heapify") i a continuació obtenir, en ordre, els $k = n/\log n$ elements més petits del heap, cadascun amb cost $\Theta(\log n)$, amb cost total $\Theta(n)$ en el cas pitjor.

2.3 Arbres AVL

L'arbre AVL resultant d'inserir les claus 94, 33, 50, 76, 96 i 67 en aquest ordre en una arbre inicialment buit és



Per obtenir-lo es fan dues rotacions dobles en inserir les claus 50 i 67.

2.4 NP-completesa

a) El problema de factoritzar un nombre donat x pertany a NP. Efectivament, donat el nombre x i un conjunt de factors, es pot verificar en temps polinòmic en b que el nombre de factors està entre 1 i b, que cada factor es codifica en com a molt b bits, que cada factor és un nombre primer i que la multiplicació de tots els factors és x.

b) Si es demostrés que no existeix cap algorisme polinòmic en b per factoritzar un nombre donat x podríem deduir P $\neq$ NP, perquè sabem que P és a NP i hauríem trobat un problema a NP que no és a P.

c) Si es trobés algun algorisme polinòmic en b per factoritzar un nombre donat x no podríem deduir ni P $=$ NP ni P $\neq$ NP, a menys que el problema de factoritzar un nombre donat x sigui NP–complet, amb la qual cosa sí que podríem deduir P $=$ NP.

2.5 2-SAT

Primer, construïm un graf dirigit amb 8 vèrtexs (dos per a cada variable, una afirmada i l'altre negada), i dos arcs per a cada disjunció. Per exemple, $(\bar{a} \vee c)$ genera els arcs $a \to c$ i $\bar{c} \to \bar{a}$. Després, calculem en temps lineal els components fortament connexos del graf. En aquest cas, obtenim quatre "súper-nodes": Un només amb b, un altre amb a, c i $\bar{d}$, un altre amb $\bar{a}$, $\bar{c}$ i d, i un altre només amb $\bar{b}$. El primer apunta al segon i al tercer, i aquests apunten al quart.

Com que cap component conté una variable i la seva negada, la fórmula té almenys una solució. Per calcular quantes, n'hi ha prou d'observar que b implica transitivament $\bar{b}$, així que b ha de ser falsa. Respecte a les altres variables, o són certs tots els literals del segon component o ho són tots els del tercer. Per tant, hi ha exactament dues solucions: o bé $a = 1$, $b = 0$, $c = 1$ i $d = 0$, o bé $a = 0$, $b = 0$, $c = 0$ i $d = 1$.

2.6 Teoria de jocs

Primer, calculem el nímber de cada pila de pedres. Recordem que, per definició, el nímber d'una pila buida és zero, i el de les piles no buides es calcula inductivament com el mínim nímber al qual no es pot arribar amb cap jugada:

n	0	1	2	3	4	5	6	7	8
nímber(n)	0	1	2	0	1	2	3	4	5

Després de jugar, tenim quatre piles, amb 1, 3, 7 i 8 pedres, i nímbers respectius 1, 0, 4 i 5. Com que el "or exclusiu" dels nímbers és 0, la posició és perdedora, i per tant la jugada proposada és guanyadora.

2.7 Què fan?

Les tres funcions comproven si V conté algun element repetit, recorrent-lo d'esquerra a dreta. La primera usa un *set*<**int**> per guardar-hi els elements que s'han vist fins ara i poder comprovar amb cost logarítmic si l'element actual ja ha sortit. La segona ordena prèviament (una còpia de) el vector perquè els elements iguals quedin situats adjacentment. I la tercera implementa una taula de dispersió sencilla.

Per a les tres funcions, un cas millor consisteix per exemple en que tots els enters siguin iguals, i un cas pitjor en que tots siguin diferents. A més, per a $h(V)$ cal que la "funció de hash" de tots els enters apunti a la mateixa posició i.

Així doncs, per a $f(V)$, el cost en el cas millor és $\Theta(1)$ (només mirem dos elements) i el cost en el cas pitjor és $\sum_{1 \leq j \leq n} \Theta(\log j) = \Theta(n \log n)$. Tant el cost en el cas millor com en el cas pitjor de $g(V)$ és $\Theta(n \log n)$, que és el cost del *sort* () de la STL, perquè la resta del codi té com a molt cost lineal. Per a $h(V)$, el cost en el cas millor és $\Theta(n)$ (cal crear el vector H), i el cost en el cas pitjor ve dominat per $\sum_{1 \leq j \leq n} \Theta(j) = \Theta(n^2)$.

2.8 Codi estrany

El codi és una implementació senzilla de l'algorisme de Prim per trobar el cost d'un abre generador mínim d'un graf no dirigit i connex amb pesos positius a les arestes.

El codi fa un *push*() (i un *top* (), i un *pop*()) per a cada aresta del graf (dues vegades). En el cas pitjor, el cost de la majoria d'aquestes operacions serà $\Theta(\log n)$, per a un cost total $\Theta(m \log n)$, on m és el nombre d'arestes.

< Falta una explicació breu de l'algorisme amb un graf petit. >

3

Parcial, 6 de novembre de 2017

3.1 Ordenació

Una solució senzilla consisteix a declarar un vector amb 26 enters, tots inicialment a zero, i recórrer la paraula una sola vegada, comptant el nombre d'aparicions de cada lletra. Al final, només cal escriure cada lletra tantes vegades com hagi aparegut. El cost és clarament $\Theta(26 + n + n) = \Theta(n)$.

Aquest cost no contradiu la fita inferior $\Omega(n \log n)$ per als algorismes d'ordenació de propòsit general, basats en comparacions, perquè aquest algorisme usa el fet que els elements del vector pertanyen a un interval fitat i petit de valors possibles, i no ha de comparar els elements entre si.

3.2 Cues de prioritats

a) El codi escriu els x elements més petits del vector en ordre decreixent.

b) Les operacions més costoses són $Q.push(V[i])$ i $Q.pop()$, amb cost logarítmic en el cas pitjor en el nombre d'elements a la cua de prioritats Q.

   La contribució dels *push* és

   $$\sum_{i=1}^{x} \log i + \sum_{i=x+1}^{n} \log x = \Theta(x \log x) + \Theta((n-x) \log x) = \Theta(n \log x) \ .$$

   La contribució dels *pop* és similar. La contribució de la resta d'operacions, totes de cost constant, és $\Theta(n)$. El cost total és doncs $\Theta(n \log x)$.

c) Una manera alternativa d'aconseguir els x elemenst més petits del vector consisteix a seleccionar l'element k-èsim amb un mètode linial en n, ja sigui en el cas pitjor o en el mitjà. Després, el vector queda particionat directament amb els x elements més petits en una de les bandes. Quan $x = \sqrt{n}$, $\Theta(n \log x) = \Theta(n \log n)$ és pitjor que $\Theta(n)$.

3.3 Programació dinàmica

a) La recurrència calcula el nombre de maneres diferents de sumar j, si es poden usar els i primers elements del vector V, però cadascun només dues vegades com a màxim.

b) La programació dinàmica corresponent tindria cost en temps $O(n \times k)$, perquè cal calcular $f(i, j)$ per a cada possible combinació com a molt una vegada, cadascuna fàcilment en temps constant.

Com que $f(i, j)$ només depèn de valors de $i' = i - 1$, n'hi ha prou de guardar un vector de mida $k + 1$ per fer tots els càlculs, amb cost extra en espai $\Theta(k)$.

3.4 Multiplicació russo-francesa

a) A cada crida recursiva es fan les operacions següents: una divisió per dos (amb cost lineal en el nombre de bits), una comprovació de paritat (mirant l'últim dígit amb cost constant), i una multiplicació per 2 (amb cost lineal en el nombre de bits), per a un total de $\Theta(n)$ operacions a cada crida recursiva. Com que cada crida disminueix en un bit la llargada de y, el cost total es $\Theta(n^2)$.

b) Per exemple, fent servir l'algorisme de Karatsuba i Ofman, dos enters d'n bits es poden multiplicar en temps $O(n^{\log_2 3})$, el qual és un cost millor.

3.5 Notació asimptòtica

$f(n)$	$g(n)$	$f(n) = X(g(n))$
$n^{1/3}$	$n^{2/3}$	$f(n) = O(g(n))$
$100n + \log(n)$	$n + (\log(n))^2$	$f(n) = \Theta(g(n))$
$n2^n$	3^n	$f(n) = O(g(n))$
$10\log(n)$	$\log(n^2)$	$f(n) = \Theta(g(n))$
2^n	$2^{n/2}$	$f(n) = \Omega(g(n))$

3.6 Zeros i uns

a) El cost és clarament $\Theta(n^2)$, perquè l'algorisme fa n crides recursives reduint la talla del problema en 2 a cada crida: a la funció *parteix*, la i s'incrementa fins al valor d, i la d només es decrementa en 1.

b) Analitzem la primera crida a la funció *parteix*. El pivot $v[0]$ serà 0 o 1 amb igual probabilitat. Si el pivot és 0, incrementem i fins a trobar el primer 1 del vector, o fins a arribar al valor de d. En mitjana, i només s'incrementa en 2. A continuació decrementem d fins al valor $i - 1$, perquè mentre es decrementa d no es troba cap valor més petit que el pivot. Si el pivot és 1, la situació és similar.

Per tant, tant si el pivot és 0 com si és 1, *parteix* deixa per ordenar recursivament dos vectors, un dels quals amb quasi tots els elements originals, per la qual cosa el cost asimptòtic mitjà també és $\Theta(n^2)$.

4

Recuperació, 27 de juny de 2017

4.1 Cerques i insercions

Cerca	Cas pitjor	Cas mitjà
Taula	$\Theta(n)$	$\Theta(n)$
Taula ordenada	$\Theta(\log n)$	$\Theta(\log n)$
Arbre binari de cerca	$\Theta(n)$	$\Theta(\log n)$
Arbre AVL	$\Theta(\log n)$	$\Theta(\log n)$
Taula de hash	$\Theta(n)$	$\Theta(1)$
Perfect hashing	$\Theta(1)$	$\Theta(1)$

Inserció	Cas pitjor	Cas mitjà
Taula	$\Theta(n)$	$\Theta(n)$
Taula ordenada	$\Theta(n)$	$\Theta(n)$
Arbre binari de cerca	$\Theta(n)$	$\Theta(\log n)$
Arbre AVL	$\Theta(\log n)$	$\Theta(\log n)$
Taula de hash	$\Theta(n)$	$\Theta(1)$

Perfect hashing considera diccionaris o conjunts *estàtics*: no es poden inserir ni esborrar claus (totes les claus es dónen quan es construeix).

4.2 Encara un altre codi misteriós

El codi escriu la longitud del cicle al qual s'arriba aplicant repetidament la funció *next*() començant en una n inicial donada, o bé es penja si no s'arriba a cap cicle. (A la pràctica, el codi podria petar per memòria si el cicle no existís, o si es trobés després de molts passos.)

Senzillament, primer es guarden tots els nombres a un *set*<**int**> fins a trobar-ne un de repetit. Després, es tornen a generar nombres fins a retornar a l'inici del bucle, comptant quants passos cal fer.

El cost és $O(m \log m)$, on m és el nombre d'elements abans i dins del cicle.

4.3 I un altre

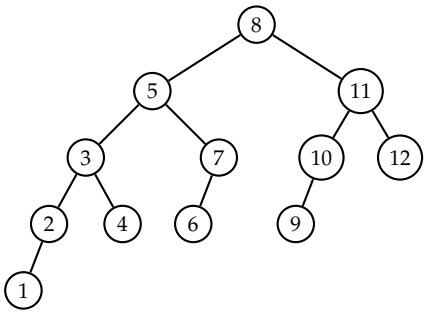
El codi és una senzilla variant de l'algorisme de Floyd-Warshall, on només es calcula l'anomenada clausura transitiva (qui està connectat amb qui), en lloc de les distàncies mínimes.

L'error és el mateix que es comet sovint amb l'algorisme de Floyd-Warshall, consistent a posar el bucle per a la k en tercer lloc, quan hauria de ser el primer de tots.

4.4 Arbres AVL

Un arbre binari és de cerca si, per a tota clau x, les claus contingudes en el seu subarbre esquerre són totes més petites que x, i les contingudes en el seu subarbre dret són totes més grans que x. Un arbre binari de cerca és AVL si la diferència d'alçades de cada parell de nodes germans és com a molt 1.

Sí que existeix algun arbre AVL amb 12 nodes i alçada 5. De fet, 12 és el mínim nombre de nodes necessaris per aconseguir aquesta alçada. Aquest n'és un:



4.5 La motxilla

1. Considerem que només hi ha dos objectes. El primer té pes 1 i benefici 2. El segon té pes C i benefici C. Llavors, l'alagorisme proposat només col·locarà el primer objecte a la motzilla (perquè la seva relació benefici per pes és 2, mentre que per l'altre és 1) i tindrà benefici 2. El segon objecte ja no cap a la motxilla. En canvi, si s'hagués posat el segon objecte, el benefici seria C. Com que C pot ser arbitràriament gran, l'algorisme proposat no pot ser un algorisme d'aproximació de raó constant.

2. Considerem un nou algorisme: Com a solució es retorna la millor d'entre la de l'algorisme proposat i la que només inclou el primer objecte que j'ha no s'ha agafat.

   Si la solució de l'algorisme proposat és subòptima, llavors ha de sobrar cert espai a la motxilla al final, diguem-ne $C - S$. Imaginem per un moment que el problema permetés agafar una fracció d'un objecte. Llavors, es podria agafar la solució de l'algorisme proposat (incloent els primers k objectes en ordre de raó benefici/pes) i afegir-hi una fracció del següent objecte (el $k+1$), de forma que afegint $b_{k+1}(C - S)/p_{k+1}$ al benefici s'igualaria (o sobrepassaria) el valor del benefici òptim OPT (pel problema sense fraccions). Per tant, o bé $\sum_{i=1}^{k} b_i \leq \frac{1}{2}\text{OPT}$ o bé $b_{k+1} \geq b_{k+1}(C - S)/p_{k+1} \geq \frac{1}{2}\text{OPT}$.

4.6 Els del mig

L'algorisme de mediana de medianes permet trobar la mediana d'un vector en temps lineal en el cas pitjor. El podem utilitzar per trobar la mediana del vector i partir-lo entre els $n/2$ element més petits i els $n/2$ element més grans. Després, podem trobar la mediana dels petits i la dels grans, per obtenir els elements del mig. Tot plegat resol el problema en cost $\Theta(n)$ en el cas pitjor.

4.7 Collatz i Turing

Podem descriure el procés de Collatz amb aquesta funció:

```
void collatz (nat n) {
    if (n ≠ 1) {
        if (n % 2 == 0) collatz (n/2);
        else collatz (3*n + 1);
    }
}
```

Per saber si el procés de Collatz s'atura per a un cert natural n, podem utilitzar la següent funció, que usa la hipotètica funció T:

```
bool collatz_atura (nat n) {
    return T( collatz , n);
}
```

Per saber si el procés de Collatz s'atura per a tot natural, es pot escriure una funció *tots* que s'atura només si per algun n, *collatz* (n) no s'atura:

```
void tots () {
    for (nat n = 1; ; ++n) {
        if (not collatz_atura (n)) return;
    }
}
```

Aleshores, la conjectura de Collatz és equivalent a **not** T(*tots* , **void**).

4.8 El valor d'un graf

Per veure que GRAPH-VALUE és **NP**, utilitzem una permutació dels vèrtexs com a testimoni i l'algorisme que en calcula la seva χ i la compara amb B. És clar que que el testimoni té mida polinòmica i que l'algorisme es pot dur a terme en temps polinòmic.

Ara veiem que VERTEX-COVER es redueix a GRAPH-VALUE: Donada una entrada per al problema del recobriment de vèrtexs (és a dir, un graf G i una fita K), creem una llista a amb K zeros i $n - K$ uns, on n és el nombre de vèrtexs de G. Establim el valor de B a zero. Llavors, el graf té un recobriment de talla K si i només si hi ha una assignació de valors de a als vèrtexs que produeix un valor zero, perquè el recobriment de vèrtexs es correspon als vèrtexs que se'ls assigna zero al problema del valor del graf.

Dels dos paràgrafs anteriors en segueix que GRAPH-VALUE és **NP**-complet.

5

Final, 17 de gener de 2017

5.1 Hashing universal

a) Un conjunt de funcions de hash H és universal si, per a tot parell de claus diferents x i y, $\mathbf{Pr}[h(x) = h(y)] \leq 1/N$, on N és el nombre de posicions de la taula de hash i h és triada a l'atzar en H.

b) Multiplicar M per x correspon a sumar (mòdul 2) les columnes de M per a les quals el bit corresponent de x és 1. Siguin x i y dues claus diferents. Sense pèrdua de generalitat, podem suposar $x_i = 0$ i $y_i = 1$ per a alguna posició i. Suposem que fixem tota M excepte la columna i-èsima. Per a tota elecció de la i-èsima columna, Mx ja és correcte. Però per a cadascuna d'aquestes 2^m possibles eleccions, hi ha un valor diferent per a My: cada cop que se'n canvia un bit, es canvia el bit corresponent a My. Per tant, la probabilitat de que Mx i My siguin iguals és $1/2^m$, perquè els m bits es trien a l'atzar.

5.2 Recurrències

a) $H(n) = H(n-1) + 1$ i $W(n) = 2W(n-1) + 2$. El teorema mestre dóna $H(n) = \Theta(n)$ i $W(n) = \Theta(2^n)$.

b) $L(n) = 2L(n-1) + 2$. El teorema mestre continua donant $L(n) = \Theta(2^n)$.

c) Siguin $A_1(n)$ i $A_2(n)$ les àrees per a un arbre complet de 4^n fulles en la primera i en la segona construcció, respectivament. Tenint en compte $4^n = 2^{2n}$, tenim $A_1(n) = H(2n)W(2n) = \Theta(2^{2n})\Theta(2n) = \Theta(2^{2n}n)$. Per altra banda, $A_2(n) = L(n)^2 = \Theta(2^{2n})$. Atès que $\lim_{n\to\infty} 2^{2n}n/2^{2n} = +\infty$, en deduïm que $A_2(n)$ és $O(A_1(n))$ però no $\Theta(A_1(n))$. Per tant, la segona construcció és asimptòticament més eficient que la primera.

5.3 MAX3SAT

a) Per exemple, l'assignació $x_1 = x_2 = x_3 = x_4 =$ **cert** satisfà ϕ i, per tant, maximitza el nombre de clàusules satisfetes.

b) És clar que MAX3SAT és **NP**, i que podem usar MAX3SAT per resoldre 3SAT, el qual és **NP**-complet. Per tant, no se sap.

c) Posem totes les variables a **cert**. Si se satisfan almenys la meitat de les clàusules, ja estem. Sinó, posem totes les variables a **fals**. Llavors s'han de satisfer més de la meitat de les clàusules (les que no se satisfeien abans).

d) Assignem **cert** o **fals** a cada variable a l'atzar. La probabilitat de no satisfer una clàusula és $1/8$. Per tant, el nombre esperat de clàusules satisfetes és $\frac{7}{8}n$. Si l'esperança té aquest valor, és que existeix alguna assignació que el té igual o més gran.

5.4 Flux màxim

El flux màxim entre dos vèrtexs coincideix amb el seu tall de cost mínim. Sabem que hi ha un tall de x i y amb cost 100. En aquest tall, z ha d'estar a la banda de y. (Si estés a la banda de x, hi hauria un tall entre y i z amb cost 100, i el flux màxim entre y i z no podria ser 200.) Com que hi ha un tall entre x i z amb cost 100, el flux màxim entre x i z és com a molt 100.

Suposem que fos més petit que 100, per exemple 50. Llavors, en el tall entre x i z amb cost 50, y no podria estar ni a la banda de x ni a la banda de z, perquè en qualsevol dels dos casos s'entraria amb contradicció amb el flux màxim conegut de 100 o de 200. Com a conclusió, el flux màxim entre x i z és exactament 100.

Cal dir que qualsevol resposta tipus "com que es pot enviar 100 de flux entre x i y, i es pot enviar 200 de flux entre y i z, llavors es pot enviar almenys 100 de flux entre x i z passant per y", tal qual, és incorrecta, perquè no considera les possibles arestes comunes entre els dos fluxos màxims (de x a y i de y a z).

5.5 Teoria de jocs

La demostració és per inducció en el nombre de jugades que queden per fer. Com a base de la inducció, el joc buit té per definició nímber 0, i la posició és perdedora. Ara, suposem que l'enunciat és cert per a tots els jocs on queden n jugades per fer, i considerem una posició qualsevol X amb $n+1$ jugades. Si X té nímber 0, per definició no és possible deixar a l'adversari cap posició amb nímber 0. Per tant, per inducció totes les posicions de l'adversari seran guanyadores, i X és perdedora. Si X té nímber estrictament positiu, per definició hi ha almenys una jugada que deixa a l'adversari en una posició amb nímber 0, que per inducció és perdedora, i per tant X és guanyadora.

5.6 Codi espatllat

a) La funció f calcula si una matriu M, corresponent a una posició del tres-en-ratlla, és guanyadora, perdedora o és empat, suposant que els dos adversaris 1 i -1 juguen de forma òptima, i que ha començat a jugar 1. El paràmetre *juga* indica a qui li toca jugar. Les caselles amb un 0 són les que encara estan lliures. Si el resultat ja és conegut, es retorna i prou. Altrament, es busca si hi ha alguna jugada guanyadora. En aquest cas no cal buscar més. Si no és així, es retorna empat (0) a no ser que totes les jugades duguin a posicions perdudes. És un exemple senzill de l'algorisme mini-max, que és una variant de programació dinàmica.

b) El codi falla quan no queden jugades a fer. En aquest cas, la funció no retorna empat com hauria de fer, sinó que retorna que el jugador perd. Per arreglar-ho, el més senzill és

afegir un booleà que marqui si s'ha trobat alguna casella lliure en el **for** del **for** sobre tots els parells i, j. Si no és el cas, cal retornar (i guardar) un 0.

5.7 Més codis misteriosos

La funció *misteri* (G) calcula si el component connex del vèrtex 0 és bicolorejable (és a dir, si és bipartit, o si no té cicles de longitud senar). És un recorregut en profunditat que intenta assignar els colors 1 i 2 alternativament a tots els vèrtexs accessibles des del 0, de manera que no hi hagi dos vèrtexs veïns del mateix color. Si en algun moment troba una contradicció, l'algorisme retorna fals. Si és incapaç de trobar-la, l'algorisme retorna cert. El cas pitjor és un graf connex i bicolorejable. En aquest cas, el cost és el d'un recorregut complet del graf, $\Theta(n+m) = \Theta(m)$, que pot arribar a ser $\Theta(n^2)$.

6

Parcial, 15 de novembre de 2016

6.1 Recurrència simpàtica

El teorema mestre mostra que la solució de $T(n) = 7T(n/7) + \Theta(n)$ és $T(n) = \Theta(n \log n)$.

Demostrem primer per inducció que $T(n) = O(n \log n)$: És suficient veure que si $T(n) \leq 7T(n/7) + cn$, llavors $T(n) \leq \beta n \log n$ per a alguna β.

Prenem doncs com a hipòtesi d'inducció que existeix alguna β per la qual, per a tota $m \leq n$, $T(m) \leq \beta m \log m$. Llavors,

$$\begin{aligned} T(n) &\leq 7T(\tfrac{n}{7}) + cn \\ &\leq 7\beta \tfrac{n}{7} \log \tfrac{n}{7} + cn \\ &\leq \beta n \log n - \beta n \log 7 + cn. \end{aligned}$$

I, per tant, $T(n) \leq \beta n \log n$ és cert quan $\beta \geq c / \log 7$.

La demostració per inducció que $T(n) = \Omega(n \log n)$ és semblant.

6.2 Els més petits

a) Aplicant l'algorisme de mediana-de-medianes per trobar el k-èsim element més petit, els elements del vector a l'esquerra de k seran els k més petits. El cost d'aquest algorisme és $\Theta(n)$ en el cas pitjor.

b) Aplicant l'algorisme de quick-select per trobar el k-èsim element més petit, els elements del vector a l'esquerra de k seran els k més petits. El cost d'aquest algorisme és $\Theta(n)$ en mitjana.

c) L'algorisme a) és més eficient en el cas pitjor, ja que b) podria té cost $\Theta(n^2)$ en el cas pitjor. En mitjana, els dos algorismes tenen el mateix cost asimptòtic, però les constants amagades a l'algorisme a) el fan més lent que b). A més, programar l'algorisme a) és més complicat que b).

6.3 Mínim local

a) És clar que X té algun mínim absolut i que aquest mínim absolut és un mínim local.

b) Es pot utilitzar una estratègia semblant a la cerca binària: Donat un vector, es mira l'element del mig. Si és un mínim local, ja està. Altrament, almenys un dels seus elements adjacents ha de ser menor que ell. Llavors, es continua recursivament en el subvector de l'element menor fins a l'extrem oposat al del mig.

   Com que cada cop es divideix per dos la talla del problema, el cost en el cas pitjor és logarítmic.

6.4 Ordenació

Considerem l'arbre de decisió T de qualsevol algorisme d'ordenació de propòsit general. Llavors, T ha de tenir, almenys, $n!$ fulles. A més, com que T és un arbre de decisió, és un arbre binari. Com que $\log_2(\frac{1}{2} \cdot n!) = \Theta(n \log n)$, no és possible que cap meitat d'aquestes $n!$ fulles es trobin a alçada $O(n)$. Per tant, per almenys la meitat de les entrades, el cost ja és més que lineal. Per tant, en mitjana també ha de ser més que lineal. Per tant, no pot existir cap algorisme d'ordenació de propòsit general que trigui temps lineal en mitjana.

6.5 Codi misteriós

El codi calcula les "sumes prefixades" del vector. Després, en troba la menor diferència en valor absolut entre dos elements eficientment, primer ordenant el vector, i després restant només els elements consecutius. Per tant,

a) El cost total és $\Theta(n \log n)$, i ve del *sort* (). Les altres parts del codi tenen cost constant o linial.

b) Les sumes prefixades són `0 -6 4 -4 -14 10`.
   Una vegada ordenades queden `-14 -6 -4 0 4 10`.
   El resultat és doncs $-4 - (-6) = 2$.

c) El codi calcula la suma d'elements més petita en valor absolut de totes les subseqüències consecutives no buides del vector. Per exemple, el 2 es correspon als elements `10 -8`.

6.6 El problema d'en Nikochan

Si s'ordenen els parells amb l'ordre per defecte, és a dir, pel *first* , amb cost $\Theta(n \log n)$, llavors cal trobar una LIS (*longest increasing subsequence*) respecte dels *second*. Com s'ha vist a classe, una LIS es pot calcular amb una programació dinàmica quadràtica, o amb un codi senzill de cost $\Theta(n \log n)$.

7

Recuperació, 7 de juliol de 2016

7.1 Recurrències

Aplicant el teorema mestre, és immediat que $A(n) = \mathrm{O}(n^2)$.

Sigui $C(n) = 3C(n/3) + \Theta(\sqrt{n}\,)$. És clar que $B(n) \leq C(n)$ per a tota n prou gran. Aplicant el teorema mestre, és immediat que $C(n) = \Theta(n)$. Per tant, $B(n) = \mathrm{O}(n)$.

7.2 Escacs

Sigui ESCACS el problema descrit. És evident que ESCACS és un problema decisional.

Malgrat ser molt gran, el nombre de posicions vàlides del joc dels escacs és finit. A més, com que les regles dels escacs impliquen que no hi ha partides amb un nombre infinit de torns, el nombre de partides, malgrat ser enormement gran, és finit també. Per tant, per a cada posició inicial es pot calcular en temps constant si és o no guanyadora. Consequentment, ESCACS es pot resoldre en temps constant.

Com que ESCACS es pot resoldre en temps constant, és evident que ESCACS pertany a la classe **P** i que ESCACS és decidible. A més, com que $\mathbf{P} \subseteq \mathbf{NP}$, també es té que ESCACS pertany a **NP**.

ESCACS és **NP**-complet o no segons l'estatus del problema $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$:

- Si $\mathbf{P} \neq \mathbf{NP}$, ESCACS no pot ser **NP**-complet, perquè si ho fós $\mathbf{P} = \mathbf{NP}$, ja que ESCACS $\in \mathbf{P}$.
- Si $\mathbf{P} = \mathbf{NP}$, tots els problemes no trivials de **P** també són **NP**-complets, per tant, ESCACS també ho seria.

7.3 Consultes en un llibre electrònic

(a) L'estructura que cal és allò que hi ha al final dels llibres: un índex. Un índex indica, per a cada paraula, a quines pàgines apareix. Es tracta doncs d'un diccionari de paraules cap a llistes ordenades de pàgines. Com que, al contrari dels llibres tradicionals, no ens cal que les paraules de l'índex estiguin ordenades alfabèticament, podem representar l'índex amb una taula de hash:

unordered_map<*string*, *vector*<**int**≫ *index*; // els vectors estan ordenats

[Una solució amb *map*<*string*, *vector*<**int**≫ és també perfectament correcta, però els costos serien diferents.]

El preprocés crea l'índex. Per fer-ho, es processen totes les paraules de cada pàgina, en ordre creixent de pàgina. Per a cada paraula x a la pàgina p, si la clau x no és al diccionari, se li afegeix, juntament amb el vector $[p]$ com a valor; si x ja és al diccionari, i el darrer element del seu vector no és p, s'afegeix p al final del vector. Això garanteix que no hi ha repeticions i que els vectors queden ordenats.

Les consultes es resolen simplement tot retornant el vector corresponent a la paraula demanada (o retornant [] si no hi és).

(b) Un *unordered_map* utilitza espai lineal respecte del nombre de claus i de valors que conté. En aquest cas, tenim n possibles paraules diferents on cadascuna d'elles pot aparèixer en, com a molt, P pàgines. Per tant, l'espai és $\mathrm{O}(nP)$.

(c) L'accés i la inserció a les taules de hash és constant en mitjana i el pre-procés en fa en ordre de N. Com que consultar el final del vector i afegir al final del vector té cost constant, es cost mitjà total del pre-procés és $\Theta(N)$.

(d) El temps de fer la cerca és $\Theta(1)$ en mitjana perquè només cal fer un accés a la taula de hash. [Es podria afegir el cost $\mathrm{O}(P)$ de la còpia del vector.]

(e) Per a cada paraula de la consulta, s'obté el seu vector de pàgines. Cal retornar doncs la intersecció de tots aquests vectors. La intersecció de dos vectors es fa amb l'algorisme de fusió, que té cost lineal respecte al nombre d'elements als dos vectors. La intersecció de molts vectors es fa incrementalment.

(f) Com que les interseccions no poden fer més que reduir el nombre d'elements dels vectors, el cost de les $k-1$ interseccions necessàries és $\mathrm{O}(kP)$ en el cas pitjor. Sobre això, cal afegir el cost $\Theta(k)$ en mitjana de fer les k cerques.

7.4 Codis misteriosos

(a) El codi escriu 12 12 23 23 42 42.

(b) En general, el codi escriu el màxim de $V[0..i]$ per a tota i entre $m-1$ i $n-1$.

(c) El cost està dominat per les n insercions, ja que la resta d'operacions tenen cost constant. El cost en el cas pitjor és $\Theta(\sum_{i=1}^{n} \log i) = \Theta(n \log n)$.

(d) El codi escriu 4 2 2 1 1 1.

(e) En general, el codi escriu el mínim de $V[i..i+m-1]$ per a tota i entre 0 i m.

(f) El cost en el cas pitjor està dominat per les n insercions, ja que la resta d'operacions tenen cost constant o, en el cas dels esborrats, el mateix cost que les insercions. Així doncs, igual que a l'apartat (c), el cost en el cas pitjor és $\Theta(n \log n)$.

(g) El primer codi no es veu afectat, perquè les cues de prioritats admeten repetits sense problemes. Els sets, però, no admeten repetits, cosa que fa variar el comportament del codi. Per exemple, amb un vector amb dos 42, el programa primer escriu 42 i després aborta.

(h) Aquest codi fa el mateix que el primer, només amb cost linial:

```
int mx = V[0];
for (int i = 1; i < m; ++i) mx = max(mx, V[i]);
cout << mx << endl;
for (int i = m; i < n; ++i) {
    mx = max(mx, V[i]);
    cout << mx << endl;
}
```

7.5 Algorisme de Kruskal

(a) L'algorisme troba un arbre generador mínim, i funciona afegint les arestes en ordre creixent de cost, sempre i quan no formin cap cicle, cosa que es pot comprovar eficientment amb un mf-set. L'algorisme pot parar quan ha afegit $n-1$ arestes.

A l'exemple donat, l'ordre en què es tracta les arestes és: 1-5, 0-7, 3-4, 3-6, 1-6, 4-7, 5-7 (s'ignora), 0-4 (s'ignora), i 2-6. L'aresta 1-4 no cal considerar-la.

(b) És fàcil veure que Kruskal genera un arbre. Per veure que és de mínim cost, sigui K l'arbre generat per Kruskal, i sigui M un arbre generador mínim. Sigui $a = u - v$ l'aresta més barata que està a un arbre però no a l'altre.

No pot ser que a estigui a M i no a K: Si fos així, voldria dir que u i v estan connectats dins de K amb arestes més barates que a (perquè Kruskal va rebutjar a en el seu moment), cosa que implicaria que M té un cicle. Per tant, a està a K però no a M.

Si afegim a a M, això forma un cicle. Si les altres arestes del cicle fossin totes més barates que a, estarien totes a K, i K tindria un cicle. Per tant, almenys una (anomenem-la b) ha de ser més cara. Però això implicaria que, afegint a i esborrant b de M, obtindríem un arbre més barat que M, contra la hipòtesi. Com a conclusió, a no pot existir, i per tant $K = M$.

7.6 Teoria de jocs

Usant la definició, es pot calcular incrementalment el nímber de piles de pedres des de 0 fins a 7. Obtenim $n(0) = 0$, $n(1) = 1$, $n(2) = 0$, $n(3) = 2$, $n(4) = 1$, $n(5) = 3$, $n(6) = 0$, i $n(7) = 4$. No cal calcular $n(999)$, perquè n'hi ha dos i es cancel·len entre si.

Ara, el nímber del joc complet és el or exclusiu de 1, 0, 2, 1, 3, 0 i 4, que és 5. Com que és diferent de 0, la posició és guanyadora. L'única jugada guanyadora que no involucra les piles amb 999 pedres consisteix a treure 3 pedres de la pila amb 7 pedres, canviant $n(7) = 4$ per $n(4) = 1$, cosa que fa anar a una posició amb or exclusiu igual a zero, i per tant perdedora.

8

Final, 19 de gener de 2016

8.1 Costos

Sigui $T_x(n)$ el temps de la funció f_x aplicada a l'entrada n per a $x \in \{1..5\}$. Llavors tenim:

- $T_1(n) = \sum_{i=1}^{n} \Theta(1) = \Theta(n)$.

- $T_2(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} \Theta(1) = \Theta(n^2)$.

- $T_3(n) = 2T_3(n/2) + \Theta(n) = \Theta(n \log n)$ utilitzant el teorema mestre.

- $T_4(n) = T_4(n-1) + T_4(n-2) + \Theta(1)$. [És la mateixa recurrència que la dels nombres de Fibonacci calculats recursivament.] Com que $T_4(n-1) \geq T_4(n-2)$, es té que $T_4(n) \leq 2T_4(n-1) + \Theta(1)$ i $T_4(n) \geq 2T_4(n-2) + \Theta(1)$. Aplicant el teorema mestre a cadascuna d'aquestes noves recurrències es troba que $T_4(n) = O(2^n)$ i $T_4(n) = \Omega(2^{n/2})$.

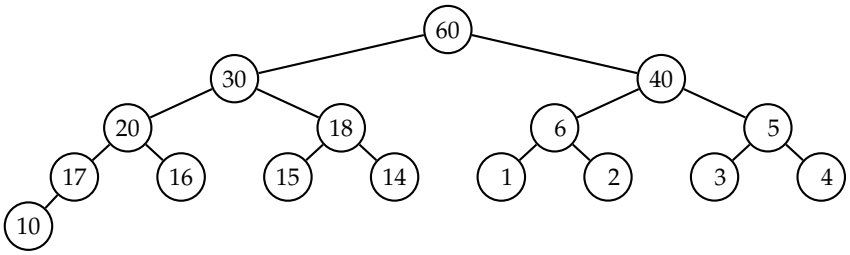
  [De fet, la solució és $T_4(n) = \Theta(\phi^n)$ on ϕ és el nombre d'or.]

- Condicionat els sumatoris per la condició del **if**, tenim

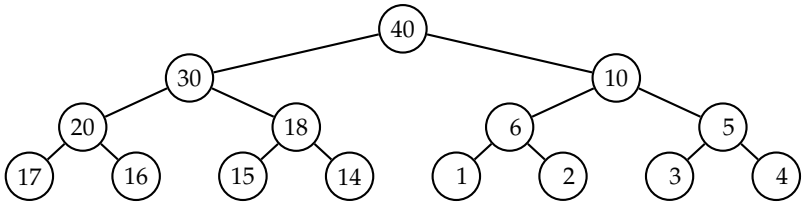
$$\begin{aligned}
T_5(n) &= \sum_{i=1}^{n} \left(\sum_{\substack{j=0 \\ j\%i=0}}^{i^2} \sum_{k=0}^{n} \Theta(1) + \sum_{\substack{j=0 \\ j\%i\neq 0}}^{i^2} \Theta(1) \right) \\
&= \sum_{i=1}^{n} \left(\sum_{\substack{j=0 \\ j\%i=0}}^{i^2} \Theta(n) + \sum_{\substack{j=0 \\ j\%i\neq 0}}^{i^2} \Theta(1) \right) \\
&= \sum_{i=1}^{n} \left(\sum_{j=0}^{i} \Theta(n) + \sum_{j=0}^{i^2-i} \Theta(1) \right) \\
&= \sum_{i=1}^{n} \left(\Theta(in) + \Theta(i^2) \right) \\
&= \sum_{i=1}^{n} \Theta(in) + \sum_{i=1}^{n} \Theta(i^2) \\
&= \Theta(n^3) + \Theta(n^3) \\
&= \Theta(n^3).
\end{aligned}$$

8.2 Heaps

Quan s'esborra l'element a l'arrel del max-heap



queda



i només s'haurà fet un intercanvi.

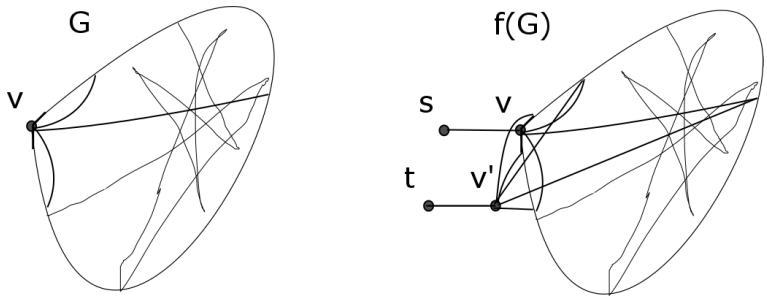
8.3 Cicles i camins Hamiltonians

Per veure que CAM $\in$ **NP-C** és suficient de veure que CAM $\in$ **NP** (1.) i que HAM es redueix a CAM en temps polinòmic (2.):

1. El testimoni de CAM pot ser una seqüència de vèrtexs. L'algorisme verificador ha de comprovar que el testimoni és una permutació dels vèrtexs del graf i que, per a cada parell de vèrtexs consecutius a la seqüència, hi ha l'aresta corresponent al graf. La talla del testimoni és lineal amb el nombre de vèrtexs al graf i és clar que es pot implementar l'algorisme verificador en temps polinòmic. Per tant, CAM $\in$ **NP**.

2. Donat un graf $G = (V, E)$, construïm un graf $f(G) = G' = (V', E')$ de la forma següent:

   Sigui v un vèrtex de G, i siguin v', s, t tres vèrtexs nous. Llavors fem $V' := V \cup \{v', s, t\}$ i $E' = E \cup \{(v', w) \mid (v, w) \in E\} \cup \{(s, v), (v', t)\}$. Esquemàticament:



   ($\Rightarrow$) Si G té un cicle Hamiltonià, el podem escriure com a $(v, u), \ell, (u', v)$ on e és una llista d'arestes encadenades entre u i u' passant per tots els vèrtexs tret de v. Llavors, $(s, v), (v, u), \ell, (u', v'), (v', t)$ és un camí Hamiltonià entre s i t en G'.

   ($\Leftarrow$) D'altra banda, si G' té un camí Hamiltonià, per força ha d'anar entre s i t perquè tenen grau 1. En aquest cas, el camí ha de ser $(s, v), (v, y), \ell', (y', v'), (v', t)$. I, llavors, G té un cicle Hamiltonià $(v, y), \ell', (y', v)$.

   Com que la reducció f es pot implementar en temps polinònic respecte la talla de G, tenim que HAM es redueix a CAM.

8.4 Inaproximabilitat

Suposem que per algun $\rho > 1$, existeix un algorisme d'aproximació A de factor ρ per a TSP. En aquest cas, veurem que podem resoldre HAM en temps polinòmic utilitzant A. Com que sabem que HAM és **NP**-complet, això no és possible sota la hipòtesi **P** $\neq$ **NP**. Per tant, TSP $\notin$ **APX** si **P** $\neq$ **NP**.

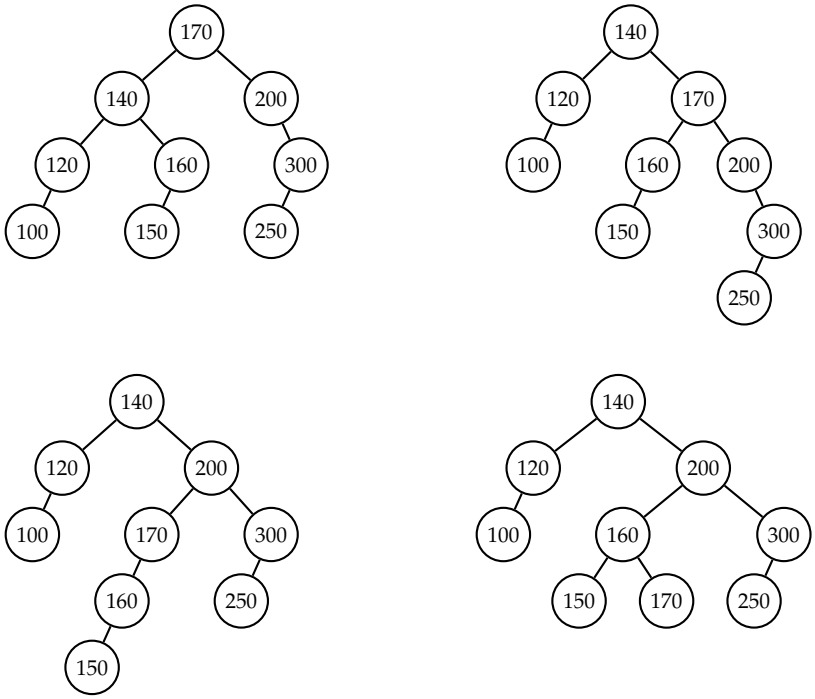
La reducció parteix d'una instància de HAM i calcula una instància de TSP. És a dir, donat un graf G amb $V = \{1..n\}$, retorna una matriu M de mida $n \times n$. Concretament, per a cada parell u, v, es fa $M[u, v] := 1$ si $\{u, v\} \in E(G)$ i $M[u, v] := \rho n + 1$ altrament.

Per tant, si G té un cicle Hamiltonià, el cost del TSP òptim en M és n, i si G no té un cicle Hamiltonià, el cost del TSP òptim en M és estrictament més gran que ρn perquè haurà d'utilitzar una de les no-arestes de G.

Així, al executar A sobre M, en el primer cas obtimdrem una solució de cost $\leq \rho n$ (perquè l'òptim és n i A és algorisme d'aproximació de factor ρ). En el segon cas, trobarem una solució de cost estrictament més gran ρn. Per tant, A pot ser usat per decidir si un graf és o no Hamiltonià en temps polinòmic.

8.5 Arbres aleatoritzats

Aplicant l'algorisme, es poden obtenir el arbres següents, amb probabilitats respectives $1/9$, $8/9 \cdot 1/6$, $8/9 \cdot 5/6 \cdot 1/3$, i $8/9 \cdot 5/6 \cdot 2/3 \cdot 1$.



8.6 Components fortament connexs

És fàcil veure que un graf dirigit amb n vèrtexs té n components fortament connexos si i només si no té cap cicle. Per tant, el graf accepta almenys una ordenació topològica. Com que cada ordenació topològica prohibeix un conjunt d'arcs (els que anirien de dreta a esquerra), i volem maximitzar aquest nombre, suposarem que només n'admet una. En aquest cas, podem tenir $(n-1)+(n-2)+\cdots+2+1 = n(n-1)/2$ arcs.

8.7 Algorisme de Prim

a) L'algorisme troba un arbre generador mínim, i funciona afegint a un subarbre format inicialment per un sol vèrtex (en el nostre cas, el 6) el vèrtex encara no afegit que estigui connectat al subarbre per l'aresta més barata. Així docs, les arestes s'afegeixen en aquest ordre: 6-3, 3-4, 6-1, 1-5, 4-7, 7-0, i 6-2.

b) És fàcil veure que Prim genera un arbre P. Per demostrar que és òptim, suposem que n'hi ha un altre de millor M, i arribarem a una contradicció. Sigui x-y una aresta que estigui a P i no a M. Aquesta aresta, quan es va escollir, era la més barata entre C, el conjunt de vèrtexs connectats al vèrtex origen, i la resta de vèrtexs R. Sense pèrdua de generalitat, suposem $x \in C$ i $y \in R$. Si afegim x-y a M, es forma un cicle. El camí antic que va de x a y, com que comença a C i acaba a R, ha de contenir almenys una aresta u-v tal que $u \in C$ i $v \in R$. Com que aquesta aresta és més cara que x-y, si l'esborrem de M i hi afegim x-y, obtenim un nou arbre generador millor que M, contra la hipòtesi.

8.8 Flors i colors

El problema proposat és exactament el de trobar el màxim aparellament possible en un graf bipartit. La solució habitual (explicada a classe) consisteix a trobar el flux màxim en el graf incrementat amb una font i un pou addicionals. L'algorisme trobarà un camí augmentatiu exactament f vegades, cadascuna de les quals triga temps $O(n + c)$ per ser un recorregut. Així doncs, una fita superior al cost és $O((n + c) \cdot f)$.

9

Parcial, 3 de novembre de 2015

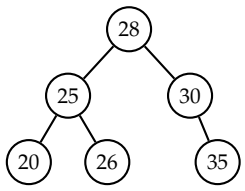
9.1 Pica-pica

1. És clar que $f(n) \leq \max\{f(n), g(n)\}$ i que $g(n) \leq \max\{f(n), g(n)\}$. Ara, sumant ambdues inequacions, $f(n) + g(n) \leq 2\max\{f(n), g(n)\} = O(\max\{f(n), g(n)\})$.

2. Una taula de dispersió amb n elements implementat amb llistes requereix temps $\Theta(n)$ per cercar/inserir/esborrar claus en el cas pitjor, i temps $\Theta(1)$ en el cas mitjà (per a un bon factor de càrrega). Anàlogament, una taula de dispersió amb AVLs requereix temps $\Theta(\log n)$ en el cas pitjor, i temps $\Theta(1)$ en el cas mitjà. Per tant, una taula de dispersió millora el cas pitjor però no afecta asimptòticament el cas mitjà.

   D'altra banda, la implementació amb AVLs és més complicada, utilitza una mica més d'espai (el doble de punters) i requereix que els elements siguin ordenables.

3. Les claus 26, 27 i 29 requereixen una rotació doble a l'arrel de l'arbre. La inserció de 26 donaria l'arbre



9.2 Maleïdes recurrències

- La recurrència és $T_1(n) = 3T_1(n/2) + \Theta(n^2\sqrt{n})$. Aplicant el teorema mestre, s'obté $T_1(n) = \Theta(n^2\sqrt{n})$.

- La recurrència és $T_2(n) = 4T_2(n/2) + \Theta(n^2)$. Aplicant el teorema mestre, s'obté $T_2(n) = \Theta(n^2 \log n)$.

- La recurrència és $T_3(n) = 5T_3(n/2) + \Theta(n^2 \log^2 n \log\log n)$, que no es pot resoldre directament. Siguin $T_4(n) = 5T_4(n/2) + \Omega(n^2)$ i $T_5(n) = 5T_5(n/2) + O(n^{2+\varepsilon})$ per a algun $\varepsilon \in (0, \frac{1}{2})$. Aplicant el teorema mestre, s'obté $T_4(n) = \Omega(n^{\log_2 5})$ i $T_5(n) = O(n^{\log_2 5})$. Però $T_3(n) = \Omega(T_4(n))$ i $T_3(n) = O(T_5(n))$, i per tant $T_3(n) = \Theta(n^{\log_2 5})$.
- Com que $\log_2 5 > 2$, la segona alternativa és la més ràpida.

9.3 Element solitari

1. Utilitzem una taula de dispersió amb n posicions. Per a cada element de l'entrada, si no és a la taula de dispersió, l'afegim; i si ja hi és, l'esborrem. Al final, l'únic element que queda a la taula de dispersió és l'element solitari.

   Cal fer unes n cerques, $n/2$ insercions i $n/2$ esborrats, cadascun de cost constant en mitjana. I, després, cal cercar l'element solitari a les n posicions. El temps total en el cas mitja és, doncs, $\Theta(n)$. L'espai és clarament $\Theta(n)$.

   [Solució una mica millor: L'element solitari és la XOR de tots els enters de l'entrada. El temps en el cas pitjor és lineal, i l'espai és constant.]

2. Fem una variació de la cerca binària: Per trobar l'element solitari en el subvector $v[i..j]$ amb $j - i + 1$ senar (i prou gran), calculem la posició central $c = (i + j)/2$, i mirem si $v[c - 1] = v[c]$. En cas afirmatiu, cal continuar cercant entre i i c, on hi ha un nombre senar d'elements i, per tant, l'element solitari. En cas contrari, cal continuar cercant entre c i j.

   El temps en el cas pitjor és logarítmic, i l'espai és constant.

9.4 En posició!

És fàcil veure que el nombre esperat de passos de la funció està afitat superiorment per $n/2^{n-1} + \sum_{i \geq 1} i/2^i = n/2^{n-1} + 2$. Per tant, el cost és $O(1)$. Com que el cost trivialment també és $\Omega(1)$, podem deduir que el cost en el cas mitjà és $\Theta(1)$.

9.5 Comptatge

Una manera de resoldre aquest problema consisteix a fer una còpia del vector original i ordenar-la. Sigui $[y_1, \ldots, y_n]$ el resultat. Llavors, cal comptar un per cada posició i tal que $x_i = y_i$, i tal que o bé $i = n$ o bé $x_i < y_{i+1}$. El cost en temps és $\Theta(n \log n)$, i l'espai addicional necessari és $\Theta(n)$.

9.6 Selecció

La recurrència per al cas pitjor és $M(n) = \Theta(n) + M(n/3) + M(2n/3)$. El primer terme es correspon al cost no recursiu de calcular $n/3$ medianes, i de particionar el vector. El segon terme, a la primera crida recursiva per calcular la mediana de $n/3$ medianes. I el tercer, a la segona crida recursiva considerant la pitjor partició possible.

La solució és $M(n) = \Theta(n \log n)$. Una possible manera de veure-ho consisteix a iterar la recurrència, i veure que la contribució de cada nivell de l'arbre de recursió al cost és linial. Finalment, n'hi ha prou d'adonar-se que aquest arbre té $\Theta(\log n)$ nivells.

10

Extraordinari, 26 de juny de 2015

10.1 Punts propers a l'origen

Podem adaptar l'algorisme de quick-select per trobar el k-èsim element d'un vector. És igual que el quick-sort, però en lloc d'ordenar les dues particions, només ordena la que cal (és a dir, la que conté l'element k-èsim). Aquest algorisme té cost $\Theta(n)$ en el cas mitjà i $\Theta(n^2)$ en el cas pitjor. La versió mediana de medianes té cost $\Theta(n)$ sempre.

10.2 Propietats dels algorismes d'ordenació

	selection-sort	insertion-sort	merge-sort	quick-sort	heap-sort
Temps en cas pitjor	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n^2)$	$\Theta(n \log n)$
Temps en cas millor	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$ $\star$
Temps en mitjana	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Mida espai addicional	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n^2)*$	$\Theta(1)$
És estable?	no	sí	sí	no	no

* L'espai de quick-sort pot ser $\Theta(n)$ en el cas pitjor, per les crides recursive que cal empilar a la pila de la recursivitat, però és fàcil rebaixar-lo a $\Theta(\log n)$.

$\star$ Quan tots els elements són iguals.

10.3 Grafs

En general, Dijkstra té cost en el cas pitjor $\Theta(n + m \log n)$. Com que aquí $m = O(n)$, tenim que el cost en el cas pitjor és $O(n \log n)$ (i no inferior a $\Theta(n)$).

Si els pesos de les arestes són naturals entre 1 i 10, per a cada arc de v a u amb cost c es poden afegir $c - 1$ vèrtexs intermedis entre v i u, i fer que tots els arcs tinguin cost unitari. Ara només cal fer un recorregut en amplada començant en x, la qual cosa té cost $O(n + 10n) = O(n)$.

10.4 Teoria de jocs

L'or exclusiu dels cinc nombres dóna 2, que com que no és 0 implica que la posició és guanyadora. Fent una mica de casuística, és fàcil veure que l'única jugada guanyadora consisteix a treure dues pedres de la pila que en té tres.

10.5 Arbres AVL

a) El cas pitjor es dóna quan l'element és en una fulla de l'arbre. Com que l'abre és AVL, el cost en el cas pitjor és $\Theta(\log n)$.

Com que la meitat dels nodes són a les fulles de l'arbre, el cost en el cas mitjà també és $\Theta(\log n)$.

b) L'arbre del mig no és AVL. El node amb un 130 està desequilibrat, ja que té alçada 2 per l'esquerra i 0 per la dreta. La resta d'arbres són arbres binaris de cerca i tenen tots els nodes equilibrats, i per tant són AVLs.

10.6 Un altre codi espifiat

El programa llegeix una seqüència de paraules i al final escriu en ordre alfabètic les que han aparegut un nombre senar de vegades. L'error és $it \leq S.end()$, que hauria de ser $it \neq S.end()$. En el cas millor totes les paraules són iguals, de manera que cada operació en el conjunt (que té com a molt un element) té cost constant. El cost és doncs $\Theta(n)$, on n és el nombre de paraules llegides. En el cas pitjor totes les paraules són diferents, de manera que cada operació en el conjunt té cost $\Theta(\log(i+1))$, on i és el nombre d'elements actual del conjunt. El cost és doncs $\sum_{i=1}^{n} \Theta(\log i) = \Theta(n \log n)$.

10.7 NP-completesa

Per començar, és evident que DIVISOR $\in$ **NP**: Donat un testimoni x, es pot comprobar en temps polinòmic que compleix els requeriments demanats.

a) Si es demostrés **P** $=$ **NP**, com que DIVISOR $\in$ **NP**, tenim que DIVISOR $\in$ **P**.

b) Si es demostrés **P** $\neq$ **NP**, això implicaria DIVISOR $\notin$ **P** només si se sabés que DIVISOR és **NP**-complet.

c) Si es demostrés que DIVISOR $\notin$ **P**, com que DIVISOR $\in$ **NP**, tenim que **P** $\neq$ **NP**.

d) Si es trobés un algorisme polinòmic per resoldre DIVISOR, això implicaria **P** $=$ **NP** només si se sabés que DIVISOR és **NP**-complet.

11

Final, 16 de gener de 2015

11.1 Recurrència

(solució d'Albert Atserias)

Demostrem un resultat més fort: $T(n) \leq 2n \ln n$ per a tot $n > 0$. Ho fem per inducció sobre n.

El cas base és per a $n = 1$. Tenim $T(1) = 0 \leq 2 \cdot 1 \cdot \ln 1 = 0$.

Per al cas inductiu ($n > 1$), assumim $T(i) \leq 2i \ln i$ per a tot $0 < i < n$. Llavors,

$$\begin{aligned} T(n) &= (n-1) + \frac{2}{n} \sum_{i=1}^{n-1} T(i) \\ &\leq (n-1) + \frac{2}{n} \sum_{i=1}^{n-1} 2i \ln i \\ &= (n-1) + \frac{4}{n} \sum_{i=1}^{n-1} i \ln i. \end{aligned}$$

Demostrem ara el següent petit lemma:

> **Lemma:** Per a tot enter $n > 0$, tenim $\sum_{i=1}^{n-1} i \ln i \leq \frac{1}{2} n^2 \ln n - \frac{1}{4} n^2 + \frac{1}{4}$.
> **Demostració:** Afitem la suma $\sum_{i=1}^{n-1} i \ln i$ superiorment per la integral $\int_1^n x \ln x \, \mathrm{d}x$. Una primitiva de $x \ln x$ és $\frac{1}{2} x^2 \ln x - \frac{1}{4} x^2$. Per tant, podem afitar la suma en qüestió per
>
> $$\left[\frac{1}{2} x^2 \ln x - \frac{1}{4} x^2 \right]_1^n = \frac{1}{2} n^2 \ln n - \frac{1}{4} n^2 + \frac{1}{4}$$
>
> tal com voliem demostrar.

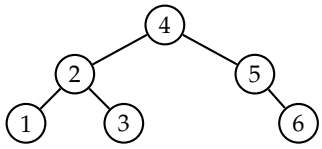
Continuem amb $T(n)$ i tenim

$$\begin{aligned} T(n) &\leq (n-1) + \frac{4}{n} \left(\frac{1}{2} n^2 \ln n - \frac{1}{4} n^2 + \frac{1}{4} \right) \\ &= (n-1) + 2n \ln n - n + \frac{1}{n} \\ &\leq 2n \ln n. \end{aligned}$$

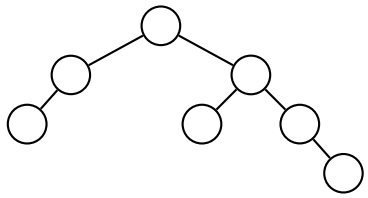
11.2 Arbres AVL

a) Recordem que el cost en el cas pitjor de cercar un element en un arbre AVL amb m nodes és $\Theta(\log m)$. Per tant, per a $m = n2^n$ nodes, el cost és $\Theta(\log(n2^n)) = \Theta(\log n + n \log 2) = \Theta(n)$.

b) L'arbre resultant és aquest bonic dibuix:



c) L'alçada és 4, amb un arbre com el següent:



11.3 Heaps ternaris

Podem representar els heaps ternaris en un vector col·locant consecutivament els seus elements de l'arrel cap a les fulles, i d'esquerra a dreta. Un node a la posició i té el seu fill k-èsim a la posició $3i + k$ per a $k \in \{1, 2, 3\}$ i el seu pare a la posició $(i - 1)/3$, en el ben entès que aquestes posicions existeixen. L'arrel és a la posició 0.

Declarant el tipus dels heaps ternaris com a vectors d'elements,

```
typedef vector<Elem> Heap3;
```

el codi de la inserció podria ser

```
void insert (Heap3& h, Elem x) {
    h.push_back(x);
    sinkup(h, h.size() - 1);
}

void sinkup (Heap3& h, int i) {
    int p = (i-1)/3;
    if (i != 0 and h[p] > h[i]) {
        swap(h[p], h[i]);
        sinkup(p);
}   }
```

Com que l'algorisme comença des d'una fulla, i es mou cap a l'arrel pujant un nivell i prenent un temps constant cada cop, el cost de l'algorisme és proporcional a l'alçada de l'arbre, que és aproximadament $\log_3 n$, és a dir, el cost és $\Theta(\log n)$ on n és la talla del heap ternari.

11.4 NP-completesa (solució d'Antoni Lozano)

La reducció de GRAN-CLICA a CLICA és senzilla: Donada una entrada $G = (V, E)$ de GRAN-CLICA, l'algorisme de reducció retorna (G, k) amb $k = |V|/2$.

La reducció de CLICA a GRAN-CLICA és més laboriosa:

Donada una entrada de CLICA $(G = (V, E), k)$, hem de retornar un graf G' tal que $G' \in$ GRAN-CLICA si i només si $(G, k) \in$ CLICA. Per descriure la reducció, definim $n = |V|$ i distingim dos casos:

1. Quan $k \geq \lceil n/2 \rceil$: Aquí n'hi ha prou a retornar un nou graf G' que és G al qual se li afegeixen $2k - n$ vèrtexs aïllats.

2. Quan $k < \lceil n/2 \rceil$: Com que un subgraf complet de k vèrtexs en G seria massa petit, definirem G' com una còpia de G més un subgraf complet de i vèrtexs connectats a tots els de la còpia de G. D'aquesta manera, un hipotètic subgraf complet de k vèrtexs a G es convertirà en una subgraf complet de $k + i$ vèrtexs a G'. Perquè $k + i$ representi la meitat dels vèrtexs de G', cal que $2(k + i) = n + i$, és a dir, $i = n - 2k$. Per tant, $G' = (V', E')$, on

   - $V' = V \cup \{u_1, \ldots, u_{n-2k}\}$, sent $u_1, \ldots, u_{n-2k}$ vèrtexs nous.
   - $E' = E \cup \{\{u_i, v\} \mid 1 \leq i \leq n - 2k \wedge v \in V\}$.

En el cas (1), és evident que la reducció és correcta. En el (2), si $(G, k) \in$ CLICA, hem construït G' de manera que pertanyi a GRAN-CLICA; en l'altre sentit, si $G' \in$ GRAN-CLICA, llavors G' té un subgraf complet amb la meitat dels seus vèrtexs, és a dir, $n - k$, dels quals almenys k han de pertànyer a la còpia de G. Per tant, $(G, k) \in$ CLICA.

És clar que ambdues reduccions es fan en temps polinòmic.

11.5 Ordenació topològica

Suposem que l'única ordenació topològica és $0, \ldots, n - 1$. Llavors els arcs mostrats en el graf de l'esquerra (per a $n = 5$) són imprescindibles ja que, altrament, hi hauria altres ordenacions possibles. Un cop fixat un ordre entre els vèrtexs, tots els arcs orientats cap a l'esquerra estan prohibits. I la resta d'arcs orientats cap a la dreta són opcionals. Per exemple, a la dreta es pot veure un dels diversos grafs amb la mateixa ordenació topològica 0, 1, 2, 3, 4.

Així doncs, el nombre d'ordenacions topològiques és $n!$ i, per a cadascuna, el nombre d'arcs opcionals és $(n - 2) + \cdots + 0 = (n - 1)(n - 2)/2$. Per tant, podem deduir que el nombre de grafs és $n! 2^{(n-1)(n-2)/2}$.

11.6 Codi espifiat

L'algorisme comprova si un graf no dirigit és connex o no. Ho fa usant un MF-set, fins que només queda una component connexa, o fins que les arestes s'acaben. L'error està a la línia

$$pare[y] = x;$$

que hauria de ser

$$pare[ry] = rx;$$

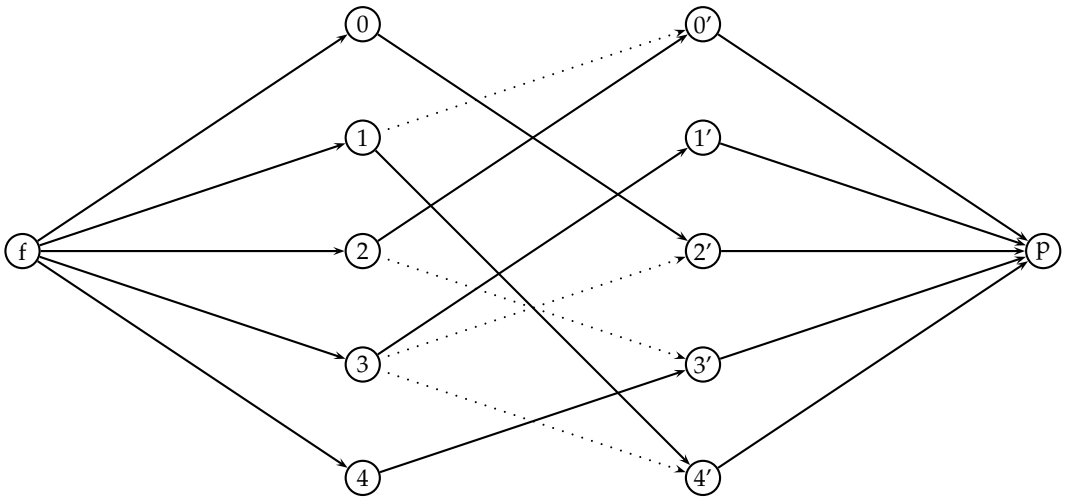
En el cas millor, les primeres $n-1$ arestes (si existeixen) connecten tot el graf. En aquest cas es fan $min(n-1, m)$ iteracions, cadascuna de cost "constant". Addicionalment, la línia

$$vector<\textbf{int}> pare(n, -1);$$

té cost $\Theta(n)$. Tot plegat, el cost en el cas millor és $\Theta(n)$. En el cas pitjor cal fer m iteracions, i el cost és $\Theta(n+m)$.

11.7 Max-flow

És fàcil veure que un graf dirigit és un conjunt de cicles si i només si tots els vèrtexs tenen exactament un arc d'entrada i un arc de sortida. Per tant, si dupliquem els vèrtexs del graf original, afegim una font i un pou, i fem que tots els arcs tinguin capacitat u, com es pot veure en l'exemple (on s'han marcat amb punts els arcs per on no acaba passant flux), llavors es poden escollir n arcs adequats si i només si el flux màxim és n.



11.8 Teoria de jocs

És impossible. Sigui n el nímber de la composició de la resta de piles, i suposem que a la pila en qüestió hi ha dues jugades guanyadores que deixen x i y pedres respectivament, amb $x \neq y$. Com que el resultat de les jugades ha de ser perdedor, tenim $x \hat{} n = y \hat{} n = 0$, cosa que implica $x = y = n$, contra la hipòtesi.

12

Parcial, 14 de novembre de 2014

12.1 Ordenació per selecció

1. Les assignacions entre elements només es fan dins dels *swap*(). Cada *swap*() realitza tres assignacions (i no dues!). Com que a cada iteració del bucle de les i s'executa un *swap*(), en total se n'executen n i, per tant, es fan $3n$ assignacions.

2. Les úniques comparacions entre elements tenen lloc dins del **if**. Per tant, se'n fan $\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = n(n-1)/2$.

3. Sigui X la variable aleatòria que compta el nombre de cops que s'executa $(\star)$. Per trobar l'esperança de X, definim $X_{i,j}$ com la variable aleatòria que val 1 si la instrucció $(\star)$ s'executa a la iteració j del bucle interior dins de la iteració i del bucle exterior, i 0 altrament. Llavors,

$$\mathbf{E}[X] = \mathbf{E}\left[\sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} X_{i,j}\right].$$

Per linearitat de l'esperança i definició d'esperança,

$$\mathbf{E}[X] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \mathbf{E}[X_{i,j}] = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \mathbf{Pr}[X_{i,j} = 1].$$

Per trobar $\mathbf{Pr}[X_{i,j} = 1]$, observem que $X_{i,j} = 1$ quan $v[j]$ és el valor més petit del subvector $v[i, \ldots, j]$. Com que els elements de i a $n-1$ sempre estan distribuïts a l'atzar i sense repeticions, la probabilitat de que l'element $v[j]$ sigui el mínim de $v[i, \ldots, j]$ és $1/(j-i+1)$. Per tant, tenint en compte que $H(k)$ és el k-èsim nombre

harmònic,

$$\begin{aligned}\mathbf{E}[X] &= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \frac{1}{j-i+1} \\ &= \sum_{i=0}^{n-1} \sum_{j=2}^{n-i} \frac{1}{j} \\ &= \sum_{i=0}^{n-1} (H(n-i)-1) \\ &= \sum_{i=0}^{n-1} H(n-i) - n \\ &= \sum_{i=1}^{n} H(i) - n.\end{aligned}$$

[A partir d'aquí, es pot veure que $\mathbf{E}[X] = \Theta(n \log n)$.]

12.2 Recurrència

[Com a referència, aquesta recurrència és la que caracteritza el cost de l'algorisme de mediana de medianes per trobar el k-èsim element ordenat d'un vector en temps lineal en el cas pitjor.]

Sigui $R(n) = R\left(\frac{1}{5}n\right) + R\left(\frac{3}{4}n\right) + \beta n$. Demostrarem per inducció que, per a tota n, $R(n) \leq \alpha n$ per a una certa constant α. D'aquí, es dedueix que $T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{3}{4}n\right) + O(n)$ és $O(n)$.

Com a hipòtesi d'inducció, suposem que, donada una n, per a tota $m < n$ tenim que $R(m) \leq \alpha m$. Llavors,

$$\begin{aligned}R(n) &= R\left(\tfrac{1}{5}n\right) + R\left(\tfrac{3}{4}n\right) + \beta n \\ &\leq \alpha\tfrac{1}{5}n + \alpha\tfrac{3}{4}n + \beta n \\ &\leq \left(\tfrac{19}{20}\alpha + \beta\right) n \\ &\leq \alpha n\end{aligned}$$

sempre que $\alpha \geq 20\beta$.

12.3 Primer element repetit

1. Per a cada element (d'esquerra a dreta), cal mirar si aquest té algun element igual a la seva dreta. El cost temporal és $\Theta(n^2)$ i l'espacial és $\Theta(1)$.

2. Utilitzem un conjunt d'enters inicialment buit. Per a cada element, si el trobem al conjunt, és que és una repetició; si no hi és, l'afegim.

   Si implementem els conjunts amb un arbre equilibrat (com ara un *set*<**int**>), en el cas pitjor cada iteració es fa en temps $O(\log n)$ i el cost total és $\Theta(n \log n)$.

   D'altra banda, si implementem els conjunts amb taules de dispersió (com ara un *unordered_set*<**int**>), en mitjana cada iteració es fa en temps $\Theta(1)$ i el cost total mitjà és $\Theta(n)$. El cas pitjor, però, és $\Theta(n^2)$.

   En ambdós casos, l'espai necessari és $\Theta(n)$.

12.4 Subvector de suma zero

Si calculem la suma prefixada del vector (és a dir, per a cada element, la suma dels elements a la seva esquerra), una subseqüència d'elements consecutius de suma zero es correspon a un element repetit a la suma prefixada.

Per tant, podem utilitzar la solució del problema anterior per resoldre el problema en $\Theta(n \log n)$ passos en el cas pitjor utilitzant un *set* o bé en $\Theta(n)$ passos en el cas mitjà utilitzant un *unordered_set*.

12.5 Nombres de Fibonacci

Sigui $M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$. Llavors tenim $\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = M \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \cdots = M^n \begin{pmatrix} 1 \\ 0 \end{pmatrix}$. Per tant, F_n és l'element de baix a l'esquerra de la matriu M^n, la qual es pot calcular amb exponenciació ràpida amb cost $\Theta(\log n)$.

12.6 Heaps

Els passos després d'inserir un 43 són:

47	41	31	37	29	13	19	17	11	23	43
47	41	31	37	43	13	19	17	11	23	29
47	43	31	37	41	13	19	17	11	23	29

Els passos després d'esborrar el màxim són:

23	41	31	37	29	13	19	17	11
41	23	31	37	29	13	19	17	11
41	37	31	23	29	13	19	17	11

12.7 Ordenació topològica

El nombre màxim d'ordenacions topològiques amb n vèrtexs és $n!$, i aquestes es poden assolir només amb un graf sense arcs.

Com que el graf no té cicles, sempre tindrà, almenys, una ordenació topològica. Un dels molts grafs que només en tenen una és $1 \to 2 \to \cdots \to n$.

13

Recuperació, 7 de juliol de 2014

13.1 Recurrència

Tenim $F(n) = 2F(n/2) + 2F(n/3) + \Theta(n^2)$. Definim $R(n) = 2R(n/2) + 2R(n/2) + \Theta(n^2)$. Pel teorema mestre, $R(n) = \Theta(n^2 \log n)$. Com que $F(n) \leq R(n)$ per a tot n prou gran, $F(n) = O(n^2 \log n)$.

13.2 Interseccions

a) Primer s'ordenen els K vectors amb l'algorisme de heap-sort. Després, es van fusionant els K vectors per anar-se quedant amb els elements comuns.

   Les K ordenacions prenen temps $\Theta(Kn \log n)$ en el cas pitjor. Amb heap-sort, el cost en espai és constant.

   La fusió de dos vectors de, com a molt, n elements pren temps $\Theta(n)$ en el cas pitjor i requereix un vector auxiliar de mida $\Theta(n)$.

   Així doncs, en total, el temps és $\Theta(Kn \log n + Kn) = \Theta(n \log n)$ i l'espai és $\Theta(n)$.

b) Primer, per a cada vector, es guarden els seus elements en un conjunt implementat amb una taula de hash. Després, per a cada element del primer vector, es mira si apareix a tots els altres conjunts.

   Com que, en mitjana, les insercions i les cerques en taules de hash prenen temps constant i se'n fan Kn de cada, el temps mitjà és $\Theta(n)$. L'espai necessari és $\Theta(Kn) = \Theta(n)$ per guardar les K taules de hash amb n elements cadascuna.

13.3 Anagrames

Dues paraules són anagrames si els seus vectors de caràcters ordenats són iguals.

Utilitzarem una estructura del tipus *unordered_map<string, vector<string>>* M que té com a claus paraules ordenades i com a informacions totes les paraules de L que són anagrames de la clau.

Per contruir l'estructura de dades, per a cada paraula s de L, s'insereix s al final de la llista de $M[o(s)]$, on $o(s)$ és s ordenada.

Per a cada consulta x, cal retornar $M[o(x)]$.

Suposant que les paraules tenen una mida constant, les seves ordenacions tenen també cost constant. Per crear M, calen n insercions, cadascuna de cost constant en mitjana. Per tant, en total, temps $\Theta(n)$ en mitjana. L'espai necessari serà el de les n paraules i la taula de hash, que és $\Theta(n)$.

Les cerques es fan en temps constant en mitjana.

[Si s'usa un *map* enlloc d'un *unordered_map* cal modificar els costos constants en mitjana per costos logarítmics en el cas pitjor.]

13.4 Creació de heaps

Vegeu problema 15.3.

13.5 Quadrat d'una matriu

Per calcular el producte de dues matrius X i Y de mida $n \times n$, es pot calcular el quadrat de

$$M = \begin{pmatrix} 0 & Y \\ X & 0 \end{pmatrix} \qquad \text{que és} \qquad M^2 = \begin{pmatrix} YX & 0 \\ 0 & XY \end{pmatrix}$$

i quedar-se amb el quadrant inferior dret.

Com que M és una matriu de mida $2n \times 2n$, el seu quadrat es podria calcular en temps $\Theta((2n)^2) = \Theta(n^2)$ i l'extracció del quadrant en temps $\Theta(n^2)$.

13.6 Arbres de cerca

```
int mida (punter p) {
    return p ? p→m : 0;
}

int posicio (double y, punter p) {
    if (not p) return −1;
    if (y == p→x) return 1 + mida(p→dre);
    if (y < p→x) return posicio(y, p→esq);
    int pos = posicio (y, p→dre);
    return pos == −1 ? −1 : mida(p→dre) + 1 + pos;
}
```

Com que cada cop es va per la dreta o per l'esquerra, el cost és proporcional a l'alçada de l'arbre.

13.7 Teoria de jocs

L'afirmació és falsa. Per exemple, si els nímbers són 5, 4 i 2, l'únic moviment guanyador que existeix segur canvia el 2 per un 1. (Potser es pot canviar el 5 per un 6, però d'això no en podem estar segurs.)

14

Final, 16 de gener de 2014

14.1 Sumeu zero

1. Cal aplicar l'operació de cerca estàndard dels AVL. El cost és $\Theta(\log n)$.
2. Es pot fer un algorisme recursiu que, començant a l'arrel, miri el signe de cada node: si és zero, ja s'ha trobat, si és negatiu ja es pot acabar, i si és positiu es continua pels seus dos fills. El cost és $\Theta(n)$ perquè el zero pot trobar-se (o no) a qualsevol posició de l'arbre.
3. S'ordena el vector i s'utilitza la solució del punt següent (4). Amb un algorisme d'ordenació $\Theta(n \log n)$ es cost és $\Theta(n \log n) + \Theta(n) = \Theta(n \log n)$.
4. Es situen dos índexos a cada extrem del vector. Mentre no es creuin ni els valors a què apunten sumin zero, si la suma és positiva es mou el de la dreta cap a l'esquerra, i sinó es mou el de l'esquerra cap a la dreta. El cost és $\Theta(n)$.
5. Només cal mirar uns pocs casos quan el vector és prou petit. El cost és $\Theta(1)$.
6. Es recorre l'arbre en inodre per deixar tots els seus elements ordenats en un vector (en temps lineal) i s'utilitza el punt 4. El cost és $\Theta(n)$.
7. S'ordena el vector i, per a cadascun dels seus elements x, es cerca si dos elements sumen $-x$ utilitzant una generalització del punt 4. El cost és llavors $\Theta(n^2)$.

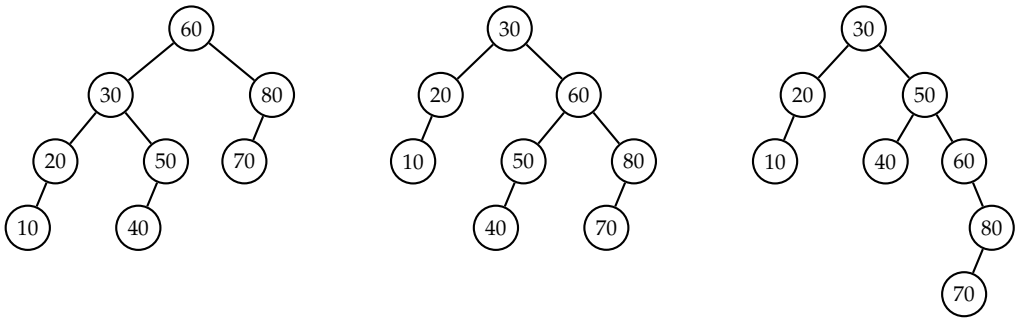
14.2 Ordenació de vectors ordenadets

Ens podem valer d'un min-heap de $2 \log n + 1$ elements. Al principi de l'algorisme, es fiquen els $2 \log n + 1$ primers elements del vector (si existeixen) en el heap. A partir d'aquell moment, a cada iteració, s'extreu el mínim del vector (que ha de ser el següent element del vector ordenat) i s'hi insereix el següent element de la taula (si en queda algun) fins que el heap quedi buit.

L'espai necessari és $\Theta(\log n)$ per emmagatzemar el heap. El temps és $O(n \log \log n)$, perquè es fan n insercions i extraccions del mínim en un heap amb $O(\log n)$ elements.

14.3 Arbres de cerca

Si l'algorisme escull l'arrel del segon arbre (60) com arrel del resultat, cosa que passa amb probabilitat $3/8$, obtenim el primer arbre. Altrament (amb probabilitat $5/8$), l'arrel del resultat serà la del primer arbre (30). En aquest cas cal fer recursivament el `join` del subarbre dret del primer arbre i del segon arbre. Amb probabilitat $(3/5) \times (5/8) = 3/8$ obtindrem el segon arbre, i amb probabilitat $(2/5) \times (5/8) = 1/4$ el tercer.



14.4 Teoria de jocs

La variant del joc presentada té les característiques necessàries per poder-lo analitzar amb nímbers. És fàcil veure per inducció que el nímber d'una pila amb n pedres és n mod 4. Així doncs, els nímbers de cada pila donada són 1, 3, 0, 1, 3, 1, 2, 0 i 0. A més, sabem que el nímber de la composició de jocs és el "xor" dels nímbers. Podem concloure que la posició de la partida té nímber 3, que no és 0, i per tant la posició és guanyadora. Una possible jugada guanyadora consisteix a treure 3 pedres de la segona pila, deixant al rival una posició amb nímber 0.

14.5 Fluxos

La reducció és incorrecta, perquè els fluxos que arriben a cada x_i es poden repartir entre els dos arcs que en surten, de manera que sempre pot arribar tot el flux s fins al pou, independentment de si la instància del problema *subset sum* que estem intentant resoldre té solució o no.

Si existís una reducció correcta (i en temps polinòmic) de *subset sum* a *max-flow*, tenint en compte que el segon pertany a **P**, llavors el primer també hi pertanyeria, i com que sabem que és **NP**-complet, tots els problemes de la classe **NP** es podrien resoldre en temps polinòmic, així que **P** seria igual a **NP**.

14.6 Codi mort

El problema d'en Genís és indecidible: Suposem que existís un algorisme per resoldre'l. Aleshores, podríem resoldre el problema de la totalitat (donat un programa p, acaba per a totes les seves entrades?) tot escribint un nou programa que sigui p al qual li hem afegit una instrucció qualsevol i al final. Si es pogués decidir que i és codi mort, es podria decidir el problema de la totalitat per a p. Però aquest problema és indecidible i, per tant, també ho és el d'en Genís.

15

Parcial, 8 de novembre de 2013

15.1 Heu preparat la lliçó?

- $O(f) = \{g : \mathrm{N} \to \mathrm{N} \mid \exists n_0 > 0, c_0 > 0 \ : \ \forall n \geq n_0 : g(n) \leq c_0 f(n)\}$.
- $\Omega(f) = \{g : \mathrm{N} \to \mathrm{N} \mid \exists n_0 > 0, c_0 > 0 \ : \ \forall n \geq n_0 : g(n) \geq c_0 f(n)\}$.
- $\Theta(f) = O(f) \cap \Omega(f)$.

15.2 Polinomis

1. Utilitzant la Regla de Horner es pot avaluar $p(x)$ en temps $\Theta(n)$:

   $$p(x) = p[0] + x(p[1] + x(p[2] + \dots))$$

2. El resultat de multiplicar $p(x) = p[0] + p[1]x + \cdots + p[n-1]x^{n-1}$ per $q(x) = q[0] + q[1]x + \cdots + q[n-1]x^{n-1}$ és $r(x) = r[0] + r[1]x + \cdots + r[2n-1]x^{2n-1}$ amb

   $$r[k] = \sum_{i=0}^{k} p[i]q[k-i]$$

   sent $p[i]$ i $q[i]$ zero quan $i \geq n$.
   El temps necessari és doncs $2\sum_{k=0}^{n-1} \Theta(k) = \Theta(n^2)$.

3. Es té

   $$(ax + b)(cx + d) = acx^2 + (a + b)(c + d)x + bd - bdx - acx$$

   i, a la banda dreta, només hi ha tres productes de reals: ac, $(a + b)(c + d)$ i bd.

4. Podem escriure els polinomis $p(x) = \sum_{i=0}^{n-1} p[i]x^i$ i $q(x) = \sum_{i=0}^{n-1} q[i]x^i$ com a

   $$p(x) = \sum_{i=0}^{n/2-1} p[i]x^i + x^{n/2} \sum_{i=0}^{n/2} p[i + n/2]x^i = b(x) + x^{n/2}a(x)$$

i

$$q(x) = \sum_{i=0}^{n/2-1} q[i]x^i + x^{n/2} \sum_{i=0}^{n/2} q[i+n/2]x^i = d(x) + x^{n/2}c(x).$$

i aplicar la idea de la pregunta anterior per obtenir

$$p(x)q(x) = acx^n + (a+b)(c+d)x^{n/2} + bd - bdx^{n/2} - acx^{n/2}$$

que estableix una forma recursiva de fer la multiplicació de polinomis de grau $n-1$ utilitzant tres multiplicacions de polinomis de grau $(n-1)/2$, a part d'operacions de suma, de restes i de multiplicació per potències de x (que tenen cost lineal).

L'algorisme resultant té cost $T(n) = 3T(n/2) + \Theta(n)$ que, aplicant el teorema mestre, resulta en $\Theta(n^{\log_2 3})$.

15.3 Heaps i heapsort

1. Cal verificar que cada node (menys l'arrel) sigui menor o igual que el seu pare:

```
bool is_max_heap (const vector<double>& v) {
    int n = v.size ();
    for (int i=1; i<n; ++i) {
        if (v[i] > v[(i-1)/2]) {
            return false;
    }   }
    return true;
}
```

El cas pitjor és quan el vector és un max-heap. En aquest cas, l'algorisme fa $n-1$ iteracions, cadascuna de cost constant. Per tant, el cost en el cas pitjor és $\Theta(n)$. El cas millor és quan a la primera iteració es veu que l'arbre no satisfà la propietat de forma. El cost en el cas millor és doncs $\Theta(1)$.

2. Es pot fer enfonsant cada node, des de les fulles de l'arbre cap a l'arrel:

```
void heapify (const vector<double>& v) {
    int n = v.size ();
    for (int i = n/2 - 1; i >= 0; i--) {
        sink(v, n, i);
}   }

void sink (vector<double>& v, int n, int i) {
    double x = v[i];
    int c = 2*i + 1;
    while (c < n) {
        if (c+1 < n and v[c] < v[c + 1]) ++c;
        if (x >= v[c]) break;
        v[i] = v[c];
        i = c;
        c = 2*i + 1;
    }
    v[i] = x;
}
```

El cost en el cas pitjor ve donat per

$$T(n) = \sum_{i=1}^{\log n} i\frac{n}{2^i}$$

perquè els $n/2$ nodes del darrer nivell s'han d'enfonsar un nivell, els $n/4$ nodes del darrer nivell s'han d'enfonsar dos nivells, ..., i l'arrel s'ha d'enfonsar $\log n$ nivells.

Per trobar $T(n)$, es pot veure que

$$T(n) - 2T(n/2) = \sum_{i=1}^{\log n} i\frac{n}{2^i} - \sum_{i=1}^{\log n - 1} i\frac{n}{2^i} = \log n$$

i, aplicar el teorema mestre per obtenir $T(n) = \Theta(n)$.

3.
```
void heapsort (vector<double>& v) {
    int n = v.size();
    make_heap(v);
    for (int i = n - 1; i >= 1; --i) {
        swap(v[0], v[i]);
        sink(v, i, 0);
}   }
```

El cost d'aquest algorisme bé donat per $T(n) = \sum_{i=1}^{n} \Theta(\log i) = \Theta(n \log n)$.

15.4 Comparacions del Quicksort

Per tal que quicksort realitzi el nombre màxim de comparacions possibles cal que cada pivot que tria sigui el màxim o el mínim del subvector a ordenar. Per tant, hi ha 2^{n-1} permutacions que provoquen el màxim nombre de comparacions.

15.5 Múltiples de 66 en un conjunt

```
void elimina_multiples_de_66 (set<int>& s) {
    auto it = s.begin();
    while (it != s.end()) {
        if (*s % 66 == 0) {
            s.erase(it++);        // C++ fa el post-increment abans de la crida
        } else {
            ++it;
}   }   }
```

Com que el cost per esborrar un element d'un set és logarítmic respecte el nombre d'elements en ell, si s conté n elements, aquest algorisme té cost $O(n \log n)$.

16

Final, 15 de gener de 2013

16.1 El mateix que el del parcial

Vegeu una possible solució a la secció 17.2.

16.2 Algorisme curiós

Per començar, suposarem com és habitual a l'assignatura que dividir té cost constant. (Altrament, els càlculs es poden adaptar.) Si x és parell, n passa a valer $n-1$. Si x és senar, n passa a valer aproximadament $\log_2 n$, que asimptòticament és molt més petit que $n-1$.

Així doncs, en el cas pitjor tenim la recurrència $P(n) = \Theta(1) + P(n-1)$, la solució de la qual òbviament és $P(n) = \Theta(n)$. El cas pitjor és $x = 2^{n-1}$.

En el cas millor, els nombres quasi sempre són senars (ja que cal passar pel 2 per arribar a 1), i tenim la recurrència aproximada $M(n) = \Theta(1) + M(\log_2 n)$, que no està coberta directament per cap "Teorema Mestre", però es pot iterar: $M(n) = \Theta(1) + M(\log_2 n) = 2\Theta(1) + M(\log_2(\log_2 n)) = 3\Theta(1) + M(\log_2(\log_2(\log_2 n)))\ldots$ fins arribar a 1, és a dir, $\Theta(\log^* n)$ vegades. Per tant, tenim $M(n) = \Theta(\log^* n)$. La successió $a_1 = 1$, $a_2 = 2$, $a_3 = 3 = 2^2 - 1$, $a_4 = 7 = 2^3 - 1$, $a_5 = 127 = 2^7 - 1$, $a_6 = 2^{127} - 1$, ...és un exemple de cas millor en el sentit invers de les crides recursives.

16.3 Problemes petits i bonics sobre vectors

1.

Algorisme	Funciona	Temps CP	Temps CM	Espai CP
a	no			
b	sí	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(1)$
c	sí	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$
d	sí	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n)$ o $\Theta(\log n)$ $\star$
e	sí	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$
f	no			
g	sí	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
h	sí	$\Theta(n^2)$	—	$\Theta(1)$
i	no			
j	sí	$\Theta(n!n)$	$\Theta(n!n)$	$\Theta(n)$

($\star$ Si s'implementa amb cura el QS, només cal espai logarítmic, però si es fa malament pot caler lineal.)

2. Podem calcular la suma S de tots els elements del vector v i, llavors, l'element que falta és el $n(n+1)/2 - S$.

3. Aquest cop calculem la suma S de tots els elements del vector v i la suma T de tots els quadrats dels elements del vector. Llavors, els elements x i y que falten han de complir

$$\begin{cases} x + y + S & = \sum_{i=1}^{n} i = n(n+1)/2 = S', \\ x^2 + y^2 + T & = \sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6 = T'. \end{cases}$$

Com que

$$x^2 + y^2 = (x+y)^2 - 2xy$$

obtenim

$$xy = ((S' - S)^2 - (T' - T))/2$$

i, per tant, x i y són les arrels de l'equació de segon grau

$$z^2 - (S' - S)z + ((S' - S)^2 - (T' - T))/2 = 0.$$

16.4 Agència matrimonial

Aquest problema es pot reduir a calcular un max-flow en un graf bipartit amb $h + d + 2$ vèrtexs. Afegim una font 0 connectada a cada home, amb capacitat de cada aresta igual al màxim nombre de dones que vol aquell home en qüestió, un pou $h + d + 1$ connectat a cada dona, amb capacitat 1 a cada aresta, i arestes amb capacitat 1 entre cada parella home-dona compatible entre si. És fàcil veure que el flux màxim d'aquest graf coincideix amb el màxim nombre de parelles que es poden fer tot respectant les restriccions.

16.5 Tasques i màquines

1. El problema decisional que hem de tractar (anomenem-lo TASQUES) és el següent:

   > Donades m màquines, n tasques de temps $t_1, \ldots, t_n$ i un natural k, existeix alguna assignació que tingui temps de procés inferior o igual a k?

   El problema és en **NP**: Podem fer servir com a testimoni d'una entrada positiva n enters entre 1 i m que indiquen a quina màquina cal executar cada tasca. Aquest testimoni té clarament talla polinòmica i és ben fàcil escriure un algorisme polinòmic per comprovar si seu temps de procés és $\leq k$.

   Per demostrar la completesa, podem reduir el problema PARTICIÓ (que sabem **NP**-complet) al problema TASQUES. Recordem que el problema PARTICIÓ és:

   > Donats q naturals $a_1, \ldots, a_q$, es poden particionar en dos subconjunts que tinguin la mateixa suma?

   A partir d'una entrada $a_1, \ldots, a_q$ de PARTICIÓ, la reducció crea una entrada per TASQUES triant $m = 2$ màquines, $n = q$ tasques, $t_i = a_i$ per a tot $i \in [1, \ldots, n]$ i $k = \lfloor S/2 \rfloor$ on $S = \sum_{i=1}^{q} a_i$. Aquest procés és clarament polinònic. Per comprovar que la reducció funciona distingim dos casos segons la paritat de S:

   En el cas que S és senar, l'entrada de PARTICIÓ és per força negativa, i és clar que l'entrada corresponent de TASQUES també.

   En el cas que S és parell, quan l'entrada de PARTICIÓ és positiva, llavors és clar que l'entrada corresponent de TASQUES també és positiva. I, del revés, quan l'entrada generades de TASQUES és positiva, llavors és clar que l'entrada original de TASQUES també és positiva.

   Nota: Una manera alternativa de resoldre el problema és adonant-se que el problema TASQUES és equivalent a BIN-PACKING$_{\leq}$ i recordar que a classe s'ha dit (però no demostrat) que aquest és **NP**-complet.

2. Sigui A el temps de procés de l'algorisme proposat i sigui OPT el temps de procés mínim.

   És clar que $OPT \geq \sum_{i=1}^{n} t_i/m$ i que $OPT \geq t_i$ per a tota i.

   D'altra banda, al moment que es posi en marxa la darrera tasca en ser processada, cal que totes les altres màquines estiguin processant altres tasques (altrament, alguna màquina hagués estat lliure anteriorment i l'algorisme li hagués assignat aquesta tasca). Sigui t_u el temps de tractament d'aquesta tasca i sigui t l'instant en què comença. Llavors, $A = t + t_u$ i $t \leq \sum_{i=1}^{n} t_i/m$.

   Com que $OPT \geq t_u$ i $OPT \geq \sum_{i=1}^{n} t_i/m$, tenim que $2OPT \geq \sum_{i=1}^{n} t_i/m + t_u \geq t + t_u = A$ i, per tant, $A/OPT \leq 2$.

   Com que l'algorisme obviament funciona en temps polinònic, queda demostrat que és un algorisme d'aproximació de raó 2.

17

Parcial, 13 de novembre de 2012

17.1 Ompliu els blancs

- Definiu formalment $O(f)$:
  $\boxed{O(f) = \{g : \mathrm{N} \to \mathrm{N} \mid \exists n_0 > 0, c_0 > 0 \ : \ \forall n \geq n_0 : g(n) < c_0 f(n)\}}$

- Si $T(n) = 2T(n-3) + n^2 - 2n + 1$, llavors $T(n) = \Theta(\ \boxed{2^{n/3}}\ )$.

- Per ordenar un vector amb n elements, l'algorisme d'ordenació per inserció triga $\Theta(\ \boxed{n}\ )$ passos en el cas millor i $\Theta(\ \boxed{n^2}\ )$ passos en el cas pitjor.

- Si féssim una cerca dicotòmica partint el vector en tres trossos enlloc de dos i buscant en el terç adequat, el cost resultant seria $\Theta(\ \boxed{\log n}\ )$.

- Suposeu que hi ha una manera de multiplicar dues matrius 3×3 amb 25 productes de reals i 50 sumes de reals. Assumint que n és una potència de 3, el cost que tindria un algorisme recursiu basat en aquest mètode per multiplicar matrius $n \times n$ seria $\Theta(\ \boxed{n^{\log_3 25}}\ )$.

17.2 Comparacions del QuickSort

Per a $1 \leq i < j \leq n$, sigui $X_{i,j}$ la variable aleatòria que val 1 si l'i-èsim element del conjunt es compara amb el j-èsim element i 0 altrament. A més, sigui $p_{i,j}$ la probabilitat que $X_{i,j}$ sigui 1. Llavors, $\mathbf{E}\left[X_{i,j}\right] = p_{i,j}$.

El nombre esperat de comparacions efectuades per Quicksort és

$$C_n = \mathbf{E}\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{i,j}\right]$$

i, per linearitat de l'esperança,

$$C_n = \mathbf{E}\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{i,j}\right] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \mathbf{E}\left[X_{i,j}\right] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} p_{i,j}.$$

Per calcular $p_{i,j}$, ens cal tenir en compte que i i j es comparen si i només si, d'entre tots els elements de i a j, es tria i o j com a pivot abans que qualsevol element entre $i+1$ i $j-1$. Com que la selecció del pivot es fa a l'atzar i hi ha dues possibles eleccions d'entre les $j-i+1$ possibles que condueixin a comparar i i j, tenim que $p_{i,j} = 2/(j-i+1)$. Aleshores,

$$C_n = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \leq 2\sum_{i=1}^{n}\sum_{k=1}^{n} \frac{1}{k} = 2nH_n = O(n\log n).$$

D'altra banda, hem vist a classe que el cas millor de Quicksort és $\Theta(n\log n)$. Com que el valor esperat ha de ser, com a mínim, tant gran com el del més petit, obtenim $C_n = \Theta(n\log n)$.

17.3 L'Euclides no hi és

1. Si els dos paràmetres són parells, segur que el seu mcd conté un factor 2 que podem eliminar de tots dos.

   Si a és parell i b és senar, el seu mcd no contindrà cap factor 2, així que podem calcular-lo en base de b i la meitat de a.

   Si a i b són senars amb $a > b$, la identitat d'Euclides $\mathrm{mcd}(a,b) = \mathrm{mcd}(a-b,b)$ i l'aplicació de l'identitat anterior (perquè $a-b$ és parell) dóna $\mathrm{mcd}(a,b) = \mathrm{mcd}((a-b)/2,b)$.

2. Cal escollir un cas base i evitar valors negatius als paràmetres:

   ```
   int mcd (int a, int b) {
       if (a == b) return a;
       if (a%2==0 and b%2==0) return 2*mcd(a/2, b/2);
       if (a%2==1 and b%2==0) return mcd(a, b/2);
       if (a > b) return mcd((a−b)/2, b);
       return mcd((b−a)/2, a);
   }
   ```

   Cada crida recursiva divideix almenys un dels dos paràmetres per la meitat. Per tant, el nombre màxim de crides recursives que es poden fer és $O(\log a + \log b)$. Com que el cost imputable a la part no recursiva és constant, el cost total és $O(\log a + \log b)$.

3. Podem rescriure l'algorisme anterior per a nombres llargs:

   ```
   bigint mcd (bigint a, bigint b) {
       if (iguals (a, b)) return a;
       if ( parell (a) and parell (b)) return mul2(mcd(div2(a), div2(b )));
       if (senar(a) and parell (b)) return mcd(a, div2(b ));
       if (major(a,b)) return mcd(div2(resta (a, b )), b);
       return mcd(div2(resta (b, a )), a);
   }
   ```

Aquí, hem utilitat funcions per les operacions aritmètiques necessàries amb significat evident i implementació directa. El seu cost és, com a molt, lineal respecte la llargada dels seus paràmetres.

Suposem que el nombre de bits totals per codificar a i b és n. Com que cada crida recursiva escurça d'un bit la talla d'un dels dos paràmetres, el nombre de crides recursives és, com a molt, n. Com que el cost de cada crida no imputable a la part recursiva és lineal, el cost total és $O(n^2)$.

17.4 La funció misteri ataca de nou

1. Aquesta funció retorna el mínim d'un vector d'enters. En el cas que el vector sigui buit comet un error accedint a un element invàlid.

2. El cas millor es dóna quan el mínim del vector es troba a la primera posició, llavors no s'executa mai.

3. El cas millor es dóna quan el vector està ordenat decreixentment. Llavors s'executa $n-1$ cops.

4. La probabilitat que l'element i-èsim sigui el més petit de i elements és $1/i$. Per tant, el nombre esperat de cops que s'executa ($\star$) sobre una permutació aleatòria és $\sum_{i=1}^{n-1} 1/i = H_{n-1} = \Theta(\log n)$.

17.5 Nombres aborrits

Siguin x i y dos nombres de n bits descomposts en $x = x_1 2^{n/2} + x_0$ i $y = y_1 2^{n/2} + y_0$ on x_1, x_0, y_1 i y_0 són nombres de $n/2$ bits. Llavors $xy = (x_1 2^{n/2} + x_0)(y_1 2^{n/2} + y_0) = x_1 y_1 2^n + x_1 y_0 2^{n/2} + x_0 y_1 2^{n/2} + x_0 y_0$.

Quan x i y són aborrits, $x_1 = x_0$ i $y_1 = y_0$, de manera que $xy = x_0 y_0 2^n + x_0 y_0 2^{n/2} + x_0 y_0 2^{n/2} + x_0 y_0$. Per tant, podem calcular el producte recursivament unicament sobre $x_0 y_0$ i obtenir que el cost de l'algorisme resultant és $T(n) = T(n/2) + \Theta(n)$ que, aplicant el teorema mestre, dóna $T(n) = \Theta(n)$.

17.6 Encara més misteris?

El programa escriu, en ordre i sense repeticions, els n primers naturals de la forma $2^\alpha 3^\beta 5^\gamma$, per a qualssevol naturals α, β i γ.

Durant la i-èsima iteració, el conjunt té entre i i $3i-2$ nombres. Per tant, el cost de la i-èsima iteració és $\Theta(\log i)$, i el cost temporal total del programa és $\sum_{i=1}^{n} \Theta(\log i) = \Theta(n \log n)$. El moment en què s'usa més memòria és al final del programa, quan el conjunt té entre n i $3n-2$ nombres. Per tant, el cost en espai és $\Theta(n)$.

18

Recuperació, 29 de juny de 2012

18.1 Miscel·lània

a) Com que el cost dels intercanvis és $\Theta(1)$, fent $n = d - e + 1$ el cost del codi segueix la recurrència

$$T(n) = 3T(n/2) + \Theta(1) \ ,$$

i del Teorema Mestre de Recurrències Divisores obtenim $T(n) = \Theta(n^{\log_2 3})$.

b) El programa no fa res!

c) El joc dels escacs o té una estratègia guanyadora o no la té. Així doncs, el problema té una solució trivial: l'algorisme que sempre diu *sí* o l'algorisme que sempre diu *no*, malgrat que ara mateix no sapiguem quin dels dos és. Per tant, el problema es pot resoldre en temps constant, per tant és a **P**i, per tant, també a **NP**.

d) No podem assegurar res, aquest problema té l'algorisme amb el cost proposat a l'enunciat, però en podria tenir un altre de cost més baix.

e) Alan Turing (23 de juny de 1912 / 7 de juny de 1954) fou un matemàtic britànic. Va treballar en camps com la informàtica teòrica, la criptoanàlisi o la intel·ligència artificial. Se'l considera el pare de la informàtica moderna. [Text i foto: Wikipedia]

18.2 Algorismes d'aproximació

Per a cada aresta $e \in E$, hi ha tres possibilitats: Els seus dos extrems són en S, cap dels seus dos extrems són en S, o un dels seus dos extrems és en S i l'altre no. Com que cada extrem és o no en S amb probabilitat $\frac{1}{2}$, les probabilitats d'aquests tres esdeveniments són $\frac{1}{4}$, $\frac{1}{4}$, i $\frac{1}{2}$ respectivament. Aixi, la probabilitat que una aresta es compti al tall és $\frac{1}{2}$.

Per linearitat de l'esperança, el tall esperat és, doncs,

$$\mathbf{E}[t(S)] = \sum_{e \in E} \mathbf{Pr}[e \in \text{ tall}] = \sum_{e \in E} \tfrac{1}{2} = \tfrac{1}{2}m.$$

D'altra banda, el tall màxim està afitat pel nombre total d'arestes: $t^*(S) \leq m$. Per tant,

$$\frac{\mathbf{E}[t(S)]}{t^*(S)} \leq \frac{1}{2}.$$

18.3 Complexitat i indecidibilitat

Per reduir SAT a ATURADA, hem de transformar les entrades de SAT a entrades d'ATURADA, és a dir, fórmules booleanes F a algorismes A i entrades x.

Per fer-ho, definim A com l'algorisme següent:

entrada: una fórmula F
per a tota assignació a de valors a les variable de F: (1)
    si $F(a)$ és cert: (2)
        atura (3)
penja (4)

La reducció és l'algorisme que, a partir de F, retorna el parell (A, x) on $x = F$.

Comprovem que realment és una reducció de temps polinòmic:

1. El cost de convertir F en (A, F) és lineal en la talla de F i, per tant, polinòmic.
2. Si F és una entrada positiva per al problema de SAT, vol dir que la fórmula F és satistactible. Per tant, té alguna assignació que satisfà F. Per tant, $A(F)$ la trobarà i s'aturarà a la línia (3). Per tant, (A, F) és una entrada positiva per al problema de l'aturada.
3. Si F és una entrada negativa per al problema de SAT, vol dir que la fórmula F no és satistactible. Per tant, no té cap assignació que la satisfà. Per tant, després de provar totes les assignacions possibles (són finites), $A(F)$ arribarà a la línia (4) on es penjarà. Per tant, (A, F) és una entrada negativa per al problema de l'aturada.

Aquesta reducció no implica que ATURADA sigui un problema **NP**-complet, només que és un problema **NP**-hard. Per tal que ATURADA fós **NP**-complet, a més caldria demostrar que ATURADA és a **NP**, cosa que és falsa, perquè sabem que ATURADA és indecidible.

18.4 Programació dinàmica

Definim la funció $sor(i, p, g)$ com el soroll màxim assolible utilitzant alguns dels i primers coets del catàleg, amb un preu total de p euros o menys i amb g grams o menys de pólvora. L'objectiu és doncs calcular $sor(N, P, G)$.

La funció *sor* es pot descriure així:

$$sor(i,p,g) = \begin{cases} 0 \\ \qquad \text{si } i = 0, \\ sor(i-1,p,g) \\ \qquad \text{si } i \neq 0 \ \wedge \ (p[i] > p \ \vee \ g[i] > g), \\ \max\{sor(i-1, p-p[i], g-g[i]) + s[i] \ , \ sor(i-1,p,g)\} \\ \qquad \text{si } i \neq 0 \ \wedge \ p[i] \leq p \ \wedge \ g[i] \leq g. \end{cases}$$

El cost de l'algorisme de programació dinàmica resultant seria $\Theta(N \cdot P \cdot G)$, perquè caldria una taula de $N \cdot P \cdot G$ valors que es podria omplir amb cost $\Theta(1)$ per casella anant dels índexos petits als grans.

18.5 Teoria de jocs

Vegeu la solució a la secció 19.3.

18.6 Codi misteriós

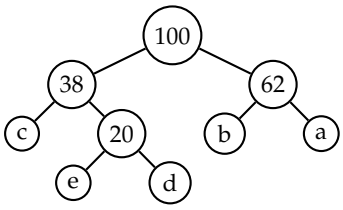
La funció *modul*(x) retorna el resultat matemàticament correcte de x mod P. Cal fer mòduls per evitar sobreiximents, tot i que això faci que la funció $f(s,\ t)$ es pugui equivocar.
La funció *num*(c) retorna el número corresponent a c entre 0 i 25. El programa fa càlculs tractant les cadenes com a nombres en base 26, aprofitant que les lletres són totes minúscules.
La funció $f(s,\ t)$ diu si el string s està dins del string t. Per fer-ho, calcula el valor de "hash" corresponent a s, i mira si hi ha alguna subcadena de t de mida n que tingui aquest mateix valor. En aquest cas, suposa que són el mateix string i retorna **true**. Si no troba cap coincidència, retorna **false**.
En el cas pitjor (quan s no està dins de t, i no hi ha cap fals positiu) la funció fa tres bucles de mida n, (un d'ells per precalcular P^{n-1}, que cal per computar eficientment el valor de "hash" de cada subcadena de t de mida n), i un bucle de mida $m - n$. Com que cada pas dels bucles té cost $\Theta(1)$, i $n \leq m$, el cost en el cas pitjor és $\Theta(m)$. En el cas millor t comença amb s, i el cost només és $\Theta(n)$.
La funció no pot retornar **false** incorrectament. Si s està dins de t, la funció ho detectarà.
La funció sí que pot retornar **true** incorrectament, si s i algun dels substrings de t de mida n són diferents però per casualitat tenen el mateix valor de "hash". La probabilitat de que això *no* passi és aproximadament $(1 - \frac{1}{P})^{m-n+1}$.

19

Final, 11 de gener de 2012

19.1 Codis de Huffman

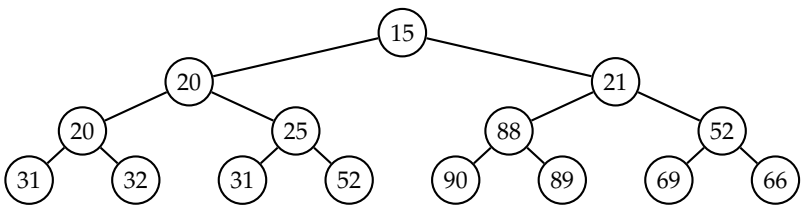
Si repetidament s'agafen els dos símbols més improbables i es posen com a fills d'un nou símbol que té probabilitat igual a la suma de les probabilitats, fins arribar a una arrel amb probabilitat 1, el resultat és aquest arbre (les fulles tenen els símbols inicials, i els altres nodes mostren les probabilitats sobre 100; suposeu que els arcs de l'esquerra tenen (arbitràriament) un 0 i els de la dreta un 1):

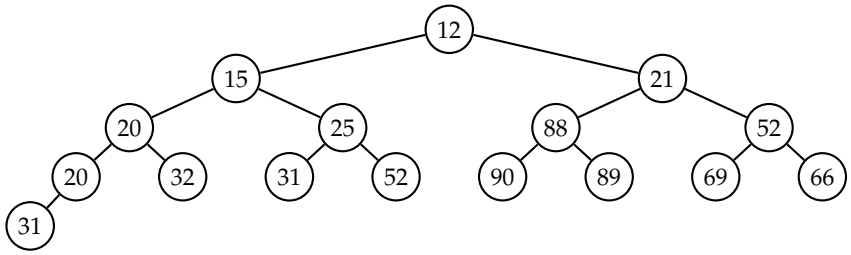


Llavors els codis serien $a = 11$, $b = 10$, $c = 00$, $d = 011$, i $e = 010$. El nombre mitjà de bits per símbol és, doncs, $0.34 \cdot 2 + 0.28 \cdot 2 + 0.18 \cdot 2 + 0.11 \cdot 3 + 0.09 \cdot 3 = 2.2$, així que calen $2.2n$ bits en mitjana per transmetre n símbols.

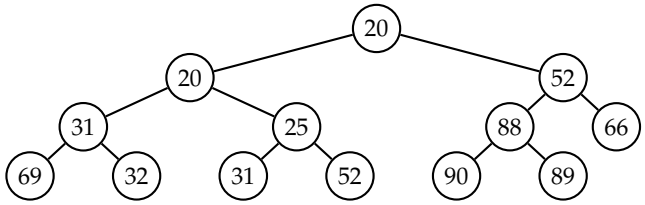
19.2 Heaps

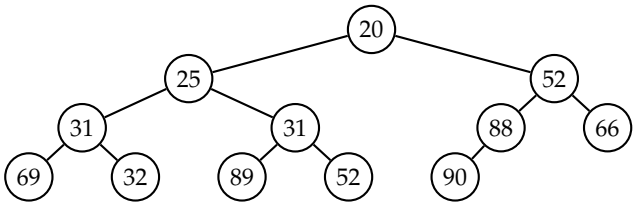
a)





b)





19.3 Teoria de jocs

Sigui x el nímber de X, y el nímber de Y, i z el nímber de la composició de X i Y. Volem demostrar que $z = x \hat{} y$. La demostració es pot fer per inducció sobre el nombre de jugades que falten. Quan no falta cap jugada, tenim $X = Y = \emptyset$, i $x = y = z = 0 = 0 \hat{} 0$. Ara, suposant que l'enunciat és cert fins a un cert nombre de jugades, anem a demostrar-ho per a jocs amb una jugada més. Cal veure:

a) El valor $x \hat{} y$ no es pot aconseguir amb cap jugada.

b) Per a tot valor $v < x \hat{} y$, v es pot aconseguir amb alguna jugada.

Considerem qualsevol moviment en (posem) Y que dugui a un Y' amb nímber y'. Per definició de nímber, $y' \neq y$. Per tant, el nímber de la composició de X i Y', que per inducció és $x \hat{} y'$, és diferent de $x \hat{} y$. Això demostra **(**a).

Per a qualsevol $v < x \hat{} y$, la diferència més a l'esquerra en la representació binària d'ambdós nombres és un bit a 0 en v que és 1 en $x \hat{} y$. Aquest bit és 0 en x i és 1 en y (o la situció simètrica). Movent a un Y' tal que y' tingui aquest bit a 0, i escollint els altres bits a la dreta convenientment (en qualsevol cas tindrem $y' < y$), sempre es pot aconseguir $x \hat{} y' = v$. Això demostra **(**b).

19.4 Estructures de dades

a) Utilitzem una taula de dispersió que enmagatzemi parells (u_i, v_i). De primer, cal recórrer C tot inserint els m parells (u_i, v_i) a la taula. Després, cal recórrer de nou C cercant per a cada parell (u_i, v_i) si el seu invers (v_i, u_i) és a la taula. La relació és simètica si i només si tots els inversos són a la taula.

El cost de cada inserció i cerca en una taula de dispersió és constant en mitjana. Com que es fan m insercions i m cerques, el cost és $\Theta(m)$ en mitjana.

b) Per saber si R és simètrica, aquest cop inserim els parells en un arbre AVL enlloc d'una taula de dispersió. Per saber si R és reflexiva, per a cada $u \in S$, cal mirar que (u, u) sigui a l'arbre AVL. Per saber si R és transitiva, construïm un graf on cada vèrtex es correspon amb cada element de S i cada aresta a cada parell de C i comprovem que cadascun dels seus components connexs és una clica.

El test per la simetria té cost $O(m \log m)$, ja que cal inserir primer i cercar després m elements en un arbre AVL que té com a molt m elements, i cada cerca i inserció tenen cost logarítmic en el cas pitjor.

El test de reflexivitat té cost $O(n + n \log m)$ que és $O(n + m \log m)$ si $n \leq m$. Si $m \leq n$, llavors segur que R no és reflexiva i només cal const constant.

El test de transitivitat es pot realitzar en temps $O(m + n)$ per construir el graf, $O(m + n)$ per etiquetar cada vèrtex amb l'identificador del seu component connex i $O(m + n)$ per comprovar que cada component és una clica.

Tot plegat és doncs $O(n + m \log m)$, tal com cal.

19.5 NP-completesa i programació dinàmica

a) Per demostrar que MOTXILLA és **NP**-complet, demotrarem que és **NP** i que PARTICIÓ es redueix en temps polinòmic a MOTXILLA.

La pertinença a **NP** queda establerta a través de testimonis que consisteixen en vectors de n bits que indiquen si cal agafar o no cada objecte. L'algorisme verificador ha de comprovar que els objectes agafats no superen el pes requerit però sí superen el valor requerit. Així, és clar que els testimonis tenen talla polinòmica i el verificador pren temps polinòmic.

La reducció tranforma els m valors $a_1, \ldots, a_m$ de PARTICIÓ en $n = m$ objectes amb $p_i = a_i$ i $v_i = a_i$ i els valors P i V s'agafen com a $P = V = \frac{1}{2}S$ on $S = \sum_{i=1}^{n} a_i$. És clar que aquesta reducció pren temps polinòmic.

Si $a_1, \ldots, a_m$ es pot particionar el dos subconjunts de mateixa suma, llavors hi ha un $A \subset [n]$ tal que $\sum_{i \in A} a_i = \sum_{i \notin A} a_i$ i, a més, $\sum_{i \in A} a_i = \sum_{i \notin A} a_i = \frac{1}{2}S$. En aquest cas, l'entrada corresponent per a MOTXILLA ha de ser positiva, perquè es poden agafar els objectes corresponents a A, amb $\sum_{i \in A} p_i = \sum_{i \in A} a_i = \frac{1}{2}S \leq P$ i $\sum_{i \in A} v_i = \sum_{i \in A} a_i = \frac{1}{2}S \geq V$.

A la inversa, si es poden agafar els objectes, cal que la suma dels seus pesos, la suma dels seus valors i $\frac{1}{2}S$ coincideixin. Per tant, agafant els a_i corresponents per una banda i els demés a_i en l'altra, la seva suma ha de ser la mateixa.

b) Definim $M(i, p)$ com al valor màxim que es pot obtenir agafant objectes fins al i inclòs amb pes menor o igual a p. Llavors,

$$M(i,p) = \begin{cases} 0 & \text{si } i = 0, \\ \max\{M(i-1, p-p_i) + v_i\,,\ M(i-1,p)\} & \text{si } i > 0 \text{ i } p_i \leq p, \\ M(i-1,p) & \text{si } i > 0 \text{ i } p_i > p. \end{cases}$$

El cost resultant d'aplicar programació dinàmica utilitzant aquesta recurrència és $\Theta(nP)$ en espai i temps. El problema decisional es respon consultant si $M(n, P) \geq V$.

c) El problema MOTXILLA és **NP**-complet i la solució anterior de cost $\Theta(nP)$ sembla tenir cost polinòmic, però en realitat no ho és, perquè el cost creix amb el valor de P, que és exponencial respecte de la talla de P. Un cas clar seria quan $P = 2^n$. Així, el gran professor Petit no ha establert que **NP** $=$ **P** i haurà de continuar vivint del seu sou retallat.

19.6 Algorismes d'aproximació

a) L'algorisme avança mentre pot incrementar el tall de S. Com que el tall comença a zero, com que creix de, com a poc, una unitat a cada iteració, i com que el tall està afitat pel nombre d'arestes del graf, l'algorisme no pot fer més de m iteracions.

b) Sigui S el subconjunt de vèrtexs retornat per l'algorisme i sigui t^* el tall màxim.

Considerem v un vèrtex de S amb grau $g(u)$. Llavors, u té $g_d(u)$ veïns dins de S i $g_f(u)$ veïns a fora de S, amb $g_d(u) + g_f(u) = g(u)$. Com que l'algorisme s'ha acabat, sabem que $g_f(u) \geq g_d(u)$ (perquè, sinó, es compliria la segona condició dels ifs) i, per tant, $g_f(u) \geq \frac{1}{2}g(u)$.

El mateix passa per a vèrtexs u fora de S, definint aquest cop $g_f(u)$ com al nombre de veïns de u dins de S.

Tenim, doncs,

$$\begin{aligned} t(S) &= \tfrac{1}{2}\left(\sum_{u\in S} g_f(u) + \sum_{u\notin S} g_f(u)\right) \\ &\geq \tfrac{1}{2}\left(\sum_{u\in S} \tfrac{1}{2}g(u) + \sum_{u\notin S} \tfrac{1}{2}g(u)\right) \\ &= \tfrac{1}{4}\sum_{u\in V} g(u) = \tfrac{1}{4}2m = \tfrac{1}{2}m. \end{aligned}$$

Com que $t^* \leq m$ i hem establert que $t(S) \geq \frac{1}{2}m$, obtenim $t^* \leq 2t(S)$.

Per tant, com que és clar que l'algorisme es pot implementar en temps polinòmic, *cerca* és un algorisme d'aproximació de raó 2.

20

Parcial, 27 d'octubre de 2011

20.1 Ompliu els blancs

Ompliu els blancs de la forma més curta i precisa possible.

- Si $T(n) = 4T(n/2) + 3n^3 - 2n^2 + 1$, llavors $T(n) = \Theta($ $\boxed{n^3}$).
- Si $T(n) = 3T(n-2) + n^2 - 2n + 1$, llavors $T(n) = \Theta($ $\boxed{3^{n/2}}$).
- Qualsevol algorisme per trobar el k-èsim element més petit d'una taula de talla n necessita $\boxed{\Omega(n)}$ passos.
- En un min-heap amb n elements, en el cas pitjor, consultar l'element mínim té cost $\boxed{\Theta(1)}$, consultar l'element màxim té cost $\boxed{\Theta(n)}$, inserir un element té cost $\boxed{\Theta(\log n)}$ i esborrar el mínim té cost $\boxed{\Theta(\log n)}$.
- Per ordenar un vector amb n elements tots iguals, l'algorisme de merge sort triga $\Theta($ $\boxed{n \log n}$) passos i l'algorisme de quick sort (amb la partició de Hoare) en triga $\Theta($ $\boxed{n \log n}$).
- Quick sort triga $\Theta($ $\boxed{n \log n}$) passos per ordenar un vector amb n elements si sempre tria com a pivot la mediana dels elements a particionar.
-

$\Omega(1/n)$	$1/(\log n)$	$O(1)$
$\Omega(n^{3/2})$	$7n^5 - 3n + 2$	$O(2^n)$
$\Omega(n^2/\log^2 n)$	$(n^2+n)/(\log^2 n + \log n)$	$O(n^2/\log^2 n)$
$\Omega(5^n)$	$2^{\log^2 n}$	$O(n^n)$
$\Omega(2^n)$	3^n	$O(5^n)$

20.2 Costs

a) El cost de *cosa*(n) és

$$T(n) = \sum_{i=0}^{n} \sum_{j=0}^{i} \sum_{k=j}^{i} \Theta(1) = \Theta\left(\tfrac{11}{6}n + \tfrac{1}{3} + \tfrac{1}{2}(n+1)^2 + \tfrac{1}{6}(n+1)^3\right) = \Theta(n^3).$$

b) El cost de *farandula*(n) ve donat per

$$R(n) = \begin{cases} \Theta(1) & \text{si } n = 0, \\ \Theta(n^3) + R(n/3) & \text{altrament.} \end{cases}$$

Utilitzant el teorema mestre, s'obté $R(n) = \Theta(n^3)$.

20.3 Sumar un

a) El cas millor es dóna quan el bit de menys pes és 0. Per sumar una unitat, només cal canviar-lo per un 1. Així, el temps no depèn de la talla del nombre i és, per tant, constant, és a dir, $\Theta(1)$.

b) El cas pitjor es dóna quan tots els bits són 1. Per sumar una unitat, cal canviar-los tots per 0. Així, el temps és lineal, és a dir, $\Theta(n)$.

c) La probabilitat de que l'algorisme se n'emporti una fins al bit i és $1/2^i$ per a $i = 1, \ldots, n$. Per tant, en mitjana triga

$$T_{\text{mig}}(n) = \Theta(1) \sum_{i=1}^{n} i/2^i = \Theta(1) \left(2 - n2^{-n} - 2^{1-n}\right) = \Theta(1).$$

Alternativament, podem establir la recurrència següent:

$$T_{\text{mig}}(n) = \tfrac{1}{2}\Theta(1) + \tfrac{1}{2}\left(\Theta(1) + T_{\text{mig}}(n-1)\right).$$

Amb el teorema mestre, s'obté $T_{\text{mig}}(n) = \Theta(1)$.

20.4 Heaps

a) L'algorisme llegeix una seqüència de n enters i l'escriu de gran a petit.

b) La cua de prioritats mai té més de n elements, per tant, les n operacions de *push*() i *front* () que s'hi apliquen sempre tenen cost $O(\log n)$ cadascuna (no Θ perquè moltes vegades no hi ha n elements!). Per tant, el cost és $O(n \log n)$.

Per veure que aquest cos és $\Theta(n \log n)$, hom pot observar que es tracta d'un algorisme d'ordenació basat en comparacions (les que fa el max-heap). Per tant, és $\Omega(n \log n)$.

Alternativament, es pot veure que, a la segona meitat de les iteracions del primer bucle i a la primera meitat de les iteracions del segon bucle, sempre hi ha, almenys, $n/2$ elements a la cua de prioritats. Per tant, les $n/2$ operacions de *push*() i *front* () que s'hi apliquen sempre tenen cost $\Omega(\log \frac{n}{2})$ cadascuna, és a dir, un total de $\Omega(\frac{n}{2} \log \frac{n}{2}) = \Omega(n \log n)$.

c) El cas millor es dóna quan tots els elements de la seqüència són iguals: en aquest cas, cada *push*() i *front* () tenen cost constant, perquè al max-heap l'element ni es sura ni s'enfonsa perquè el seu pare i els seus fills són sempre iguals que ell. Per tant, en el cas millor, el cost és $\Theta(n)$.

20.5 Triangles en grafs

Com que el graf donat no té cicles de longitud 5 o més, el problema és equivalent a detectar si el graf té algun cicle de longitud senar, cosa que és equivalent a comprovar que no sigui 2-colorejable. Per tant, un simple recorregut (que costa temps lineal en el nombre de vèrtexos i arestes) resol el problema.

21

Recuperació, 5 de juliol de 2011

21.1 Cues de prioritats

Una cua de prioritats enmagatzema un multiconjunt d'elements. Les seves operacions són:

- Crear un multiconjunt buit.
- Determinar si el multiconjunt és buit o no.
- Afegir un element al multiconjunt.
- Consultar quin és l'element més petit (gran) del multiconjunt.
- Esborrar quin és l'element més petit (gran) del multiconjunt.

Les cues de prioritat se solen implementar utilitant heaps. Utilitzant heaps, el cost de les operacions anteriors sobre una cua de prioritats amb n elements és $\Theta(\log n)$, excepte per crear i consultar si és buida, que són $\Theta(1)$.

21.2 Anàlisi

a) El primer bucle té cost $\Theta(n)$ ja que k val 1 o 2. El segon bucle té cost $\Theta(n\sqrt{n})$ perquè el bucle de les i fa $\lceil \sqrt{n} \rceil$ interacions. El tercer bucle té cost $\Theta(n \log n)$ ja que en el bucle de les i, aquesta es dobla a cada iteració. Així, la funció *intrigant* (n) té cost $\Theta(n\sqrt{n})$.

b) Podem escriure la recurrència següent per a la funció *confus*(n):

$$T(n) = \Theta(n\sqrt{n}) + T(n/2)$$

que, gràcies al teorema mestre, sabem que val $\Theta(n\sqrt{n})$.

21.3 NP-completesa

El primer problema és el problema de la PARTICIÓ, que sabem és **NP**-complet. Per tant, si $\mathbf{P} \neq \mathbf{NP}$, no pertany a **P**.

El segon problema es pot resoldre en temps $O(n^3)$ amb l'algorisme evident. Per tant, pertany a **P** amb independència de la hipòtesi.

21.4 Indecidibilitat

El problema de l'aturada total també es indecidible.

21.5 Aproximació

a) És clar que $\max_{i \in 1..n} p_i \leq OPT$ perquè no es pot trigar menys que el temps de la tasca més llarga. Igualment, $\frac{1}{m} \sum_{i \in 1..n} p_i \leq OPT$, perquè al dividir totes les tasques entre totes les màquines, pot ser que ens quedi un rest.

b)
És clar perquè l'algorisme assigna continuament tasques a les màquines mentre no s'exhaureixen.

c)
Sigui A el temps de finalització de tots els treballs. Sabem que el temps mentre totes les màquines estan treballant el temps és $\frac{1}{m} \sum_{i=1..n} p_i$ com a molt. Després, no pot haver-hi cap màquina que trigui més $\max_{i=1..n} p_i$. Per tant,

$$A \leq \frac{1}{m} \sum_{i=1..n} p_i + \max_{i=1..n} p_i \leq 2OPT.$$

21.6 Grafs

falta

21.7 Arbres

falta

21.8 Més arbres

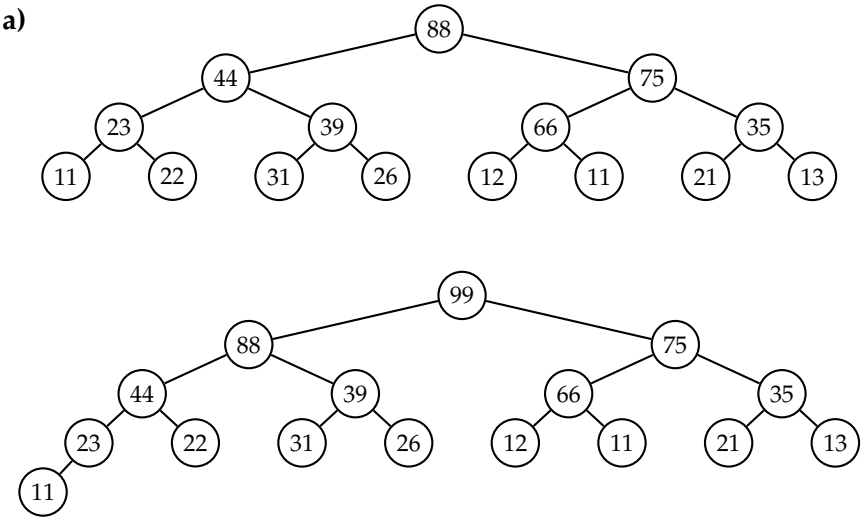
falta

22

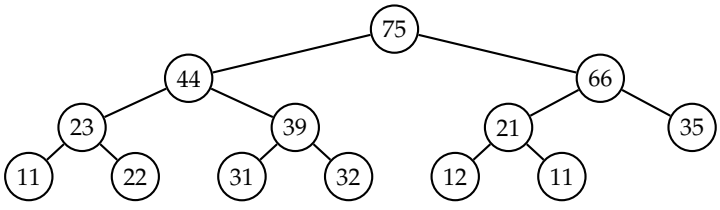
Final, 11 de gener de 2011

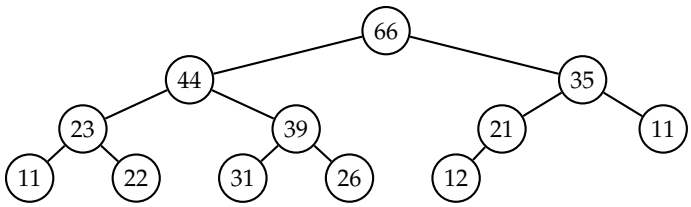
22.1 Heaps i AVLs

a)

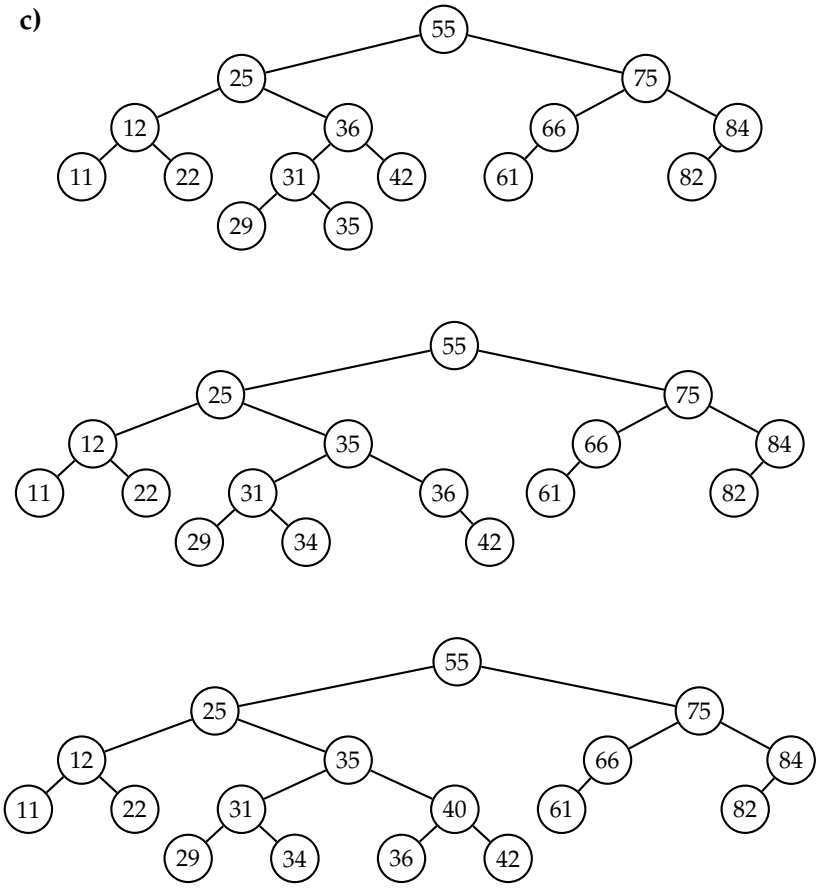


b)





c)



22.2 Anàlisi

a) El cost del primer bucle és clarament $\Theta(n)$.

El cost del segon bucle és $\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} \Theta(1)$ que és $\Theta(n^3)$.

El cost del tercer bucle és clarament $\Theta(n \log n)$.

Així, el cost total és $\Theta(n) + \Theta(n^3) + \Theta(n \log n) = \Theta(n^3)$.

b) La recurrència pel cost d'aquesta funció és $T(n) = \Theta(n^3) + T(n-1)$. Utilitzant el teorema mestre es troba que $T(n) = \Theta(n^4)$.

22.3 NP-completesa

a) 2-COLORABILITAT es pot reduir polinòmicament a 3-COLORABILITAT.

Cert: 2-COLORABILITAT pertany a la classe **P**, modificant lleugerement l'algorisme BFS (recorregut en amplada) s'obté un algorisme de temps polinòmic que decideix si donat un graf G es pot pintar amb només dos colors. Com que tot problema a **P** és també a **NP** (**P** és una subclasse de **NP**) i 3-COLORABILITAT és **NP**-complet, aleshores 2-COLORABILITATes pot reduir polinòmicament a 3-COLORABILITAT.

b) 3-COLORABILITAT es pot reduir polinòmicament a 2-COLORABILITAT.

No es pot assegurar ni una cosa ni l'altra: Si 3-COLORABILITAT es reduis polinòmicament a 2-COLORABILITAT, aleshores 3-COLORABILITAT seria a **P** i com que és **NP**-complet tindríem **P** = **NP**, igualtat de la que no es coneix la seva veritat o falsetat. Sota

la hipòtesi **P** diferent de **NP** tindríem que aquesta reducció no és possible. Sota la hipòtesi **P** = **NP** aleshores 3-COLORABILITAT es reduiria polinòmicament a 2-COLORABILITAT.

22.4 Indecidibilitat

Suposem que existís un algorisme **bool** *acaba*(*string p*, *string x*) que, donat un programa p i una extrada x, indica si el programa p s'atura o no quan s'executa sobre l'entrada x.

Ara, considerem aquest algorisme que potencialment explora totes les n, x, y, z:

```
void fermat () {
    for (int i = 3; true; ++n)
        for (int n = 3; n < i; ++n)
            for (int x = −i; x ≤ i; ++x) if (x ≠ 0)
                for (int y = −i; y ≤ i; ++y) if (y ≠ 0)
                    for (int z = −i; z ≤ i; ++z) if (z ≠ 0)
                        if (pot(x, n) + pot(y, n) == pot(z, n))
                            return;
}
```

És clar que si la conjectura de Fermat és falsa, aquest algorisme trobarà tard o d'hora un contra-exemple i acabarà en el **return**. Igualment, si la conjectura és certa, aquest algorisme es penjarà.

Així, per saber si la conjectura de Fermat és certa o no només ens caldria calcular **not** *acaba*(*fermat*, ""). De fet, l'entrada és irrellevant.

22.5 Aproximació

a) Totes les arestes amb un o dos node no-fulla estàn cobertes. Ens cal veure que no hi ha cap aresta entre dues fulles de l'arbres de DFS: Suposem que hi hagués una aresta $\{u, v\}$ entre dues fulles u i v i que u hagués estat visitat abans que v. Com que hi ha aresta entre u i v i v encara no s'ha visitat, s'ha d'anar a visitar v. Però llavors u no és fulla!

b) Trobem un aparellament maximal (*maximal matching*) M sobre l'arbre de DFS: Agafem l'arrel i un dels seus veïns i els aparallem. Els treiem de l'arbre donant lloc a més subarbres i continuem. Els subarbres amb dos o més nodes es poden continuar aparellant, els que només tenen un node, aquest es correspon amb un node fulla. Així, $|M| \geq |S|/2$ on S és el conjunt de nodes no fulla. Com que sabem que $|M| \leq OPT$, obtenim la 2-aproximació.

22.6 Grafs

a) Sigui G el graf amb vèrtexs $V = \{x, a, b, c\}$, amb els arcs

- $x \to a$, amb cost 10;
- $a \to b$, amb cost 10;
- $x \to c$, amb cost 30;
- $c \to a$, amb cost -100.

L'algorisme de Dijkstra trobaria els costs següents en aquest ordre: cost 0 per a x, cost 10 per a a, cost 20 per a b, i cost 30 per a c. Així doncs, el cost mínim real d'anar des de x fins a b, que és -60, no seria trobat per l'algorisme.

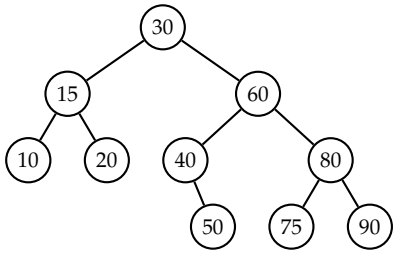
b) La modificació proposada no permet calcular els costs mínims, perquè l'increment del cost de cada camí depèn del nombre d'arcs que es travessi. Els costs obtinguts per a G' només permeten calcular fites superiors per als costs de G, perquè poden existir camins amb més passos que els millors camins trobats per a G' que amb els costs originals de G siguin menys costosos.

22.7 Llistes

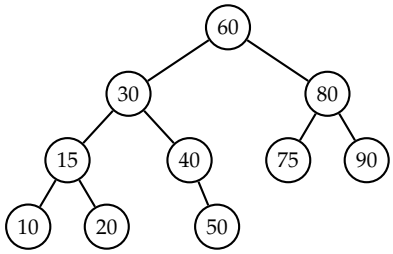
```
Punter esborra(int x, Punter p) {
    if (p == 0) return 0;
    if (p→elem > x) return p;
    if (p→elem < x) {
        p→seguent = esborra(x, p→seguent);
        return p;
    }
    punter q = p→seguent;
    delete p;
    return q;
}
```

22.8 Arbres

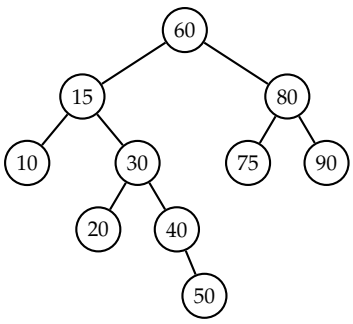
L'algorisme insereix el 30 començant a l'arrel, i a cada pas considera si es guarda en aquella posició o més avall, en funció del nombre d'elements del subarbre en curs. El possibles arbres, amb les seves probabilitats (calculades com el producte de la probabilitat de no haver inserit encara el 30, per la probabilitat de fer-ho en aquell nivell), són



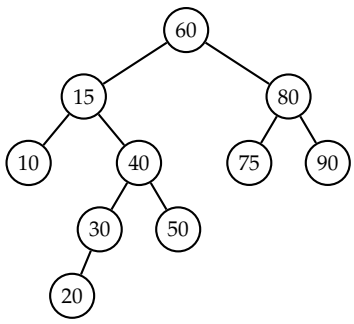
$\left(\frac{1}{10}\right)$



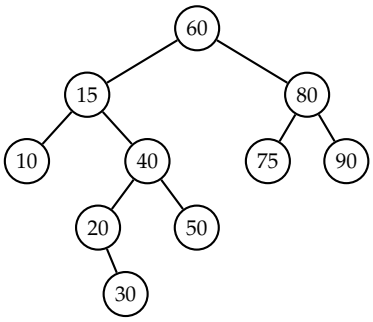
$\left(\frac{9}{10} \cdot \frac{1}{6} = \frac{3}{20}\right)$



$\left(\frac{3}{4} \cdot \frac{1}{4} = \frac{3}{16}\right)$



$\left(\frac{9}{16} \cdot \frac{1}{2} = \frac{9}{32}\right)$



$$\left(\frac{9}{32}\cdot\frac{1}{1}=\frac{9}{32}\right)$$

23

Parcial, 28 d'octubre de 2010

23.1 Ompliu els blancs

- Definiu $O(f)$:
  $\boxed{O(f) = \{g \mid \exists c_0 > 0, \exists n_0 > 0, \forall n > n_0 : g(n) \leq c_0 f(n)\}}$
- El teoreme mestre de resolució de recurrències divisores afirma que si tenim una recurrència de la forma $T(n) = aT(n/b) + \Theta(n^k)$ amb $b > 1$ i $k \geq 0$, llavors, fent $\alpha = \log_b a$,

$$T(n) = \begin{cases} \boxed{\Theta(n^k)} & \text{si } \alpha < k, \\ \boxed{\Theta(n^\alpha \log n)} & \text{si } \alpha = k, \\ \boxed{\Theta(n^\alpha)} & \text{si } \alpha > k. \end{cases}$$

- Si $T(n) = 3T(n/2) + n^2 - 2n + 1$, llavors $T(n) = \Theta($ $\boxed{n^2}$).
- Si $T(n) = 2T(n-1) + n^2 - 2n + 1$, llavors $T(n) = \Theta($ $\boxed{2^n}$).
- Mergesort ordena n elements en temps $\boxed{\Theta(n \log n)}$ en el cas pitjor, en temps $\boxed{\Theta(n \log n)}$ en el cas mitjà, i en temps $\boxed{\Theta(n \log n)}$ en el cas millor. L'espai auxiliar requerit és $\boxed{\Theta(n)}$.
- Quicksort ordena n elements en temps $\boxed{\Theta(n^2)}$ en el cas pitjor, en temps $\boxed{\Theta(n \log n)}$ en el cas mitjà, i en temps $\boxed{\Theta(n \log n)}$ en el cas millor.
- Insertionsort ordena n elements en temps $\boxed{\Theta(n^2)}$ en el cas pitjor, en temps $\boxed{\Theta(n^2)}$ en el cas mitjà, i en temps $\boxed{\Theta(n)}$ en el cas millor.

- Per multiplicar dos nombres naturals grans eficientment podem fer servir l'algorisme de [Kartabuba-Ofman] .
- Tot algorisme per trobar la mediana de n elements necessita [$\Omega(n)$] passos.
- Un algorisme d'ordenació és estable si
  [preserva l'ordre relatiu de les claus repetides] .

23.2 Ordenació

L'algorisme següent es coneix com a Counting Sort:

```
void ordena (vector<int>& v) {
    int n = v.size();
    int m = int(n*K + 1);
    vector<int> c(m, 0);
    for (int i = 0; i < n; ++i) ++c[v[i]];
    int j = 0;
    for (int i = 0; i < m; ++i) {
        for (int k = 0; k < c[i]; ++k) v[j++] = i;
    }
}
```

És clar que el cost en temps i es espai són $\Theta(nK)$. Com que K és una constant, aquests costos són lineals.

Aquest algorisme no és una algorisme de propòsit general: només serveix per ordenar enters i, a més, amb el seu valor afitat. Per això no hi ha cap contradicció amb la fita inferior d'$\Omega(n \log n)$ per ordenació. De fet, Counting Sort ni tan sols realitza compacions dels elements!

23.3 Dividir i vèncer

Si només tenim un cargol i una femella, ja hem acabat. Altrament:

Agafem un cargol qualsevol. Amb ell, classifiquem totes les femelles en femelles massa grans, massa petites i la femella que encaixa perfectament.

Amb aquesta darrera femella, classifiquem igualment tots els cargols: per un costat els massa grans, per l'altra els massa petits i deixem aparellat el que encaixa.

Un cop fet això, apliquem dos cops recursivament el mateix algorisme als dos grups de femelles i cargols.

Aquest algorisme té el mateix cost que Quicksort i és, doncs, $\Theta(n \log n)$.

23.4 Primalitat

Per saber si x és un nombre primer n'hi prou amb comprovar que no hi ha cap número entre 2 i $\sqrt{x}$ que divideixi x. Suposant que totes les operacions bàsiques entre enters tenen cost constant, el cost d'aquest algorisme és doncs $\Theta(\sqrt{x})$. Però si mesurem el seu cost en funció de la talla de x, és a dir, el seu nombre de bits, el cost és $\Omega(2^{n/2})$.

Es tracta doncs d'un algorisme exponencial, però això no vol dir que el problema de la primabilitat sigui intractable. De fet, existeixen algorismes polinònics (en funció de la talla) per saber si un nombre és primer.

23.5 Producte de matrius

a) Quan n no és una potència de 2, podem "inflar" les matrius amb zeros a baix i a la dreta amb zeros fins arribar a fer que la seva mida sigui la potència següent de 2. Llavors podem usar l'algorisme de Strassen presentat a classe i "desinflar" el resultat (treure-li les files i columnes afegides al final), que serà correcte.

Al inflar les matrius, el seu nombre d'elements no creixerà més d'un factor de 4. Per tant, el cost asimptòtic es manté.

b) Sigui $s = n^2$ el nombre d'elements a les matrius. Si descomposem les matrius en 9 submatrius i cridem recursivament m vegades, tenim la recurrència $T(s) = mT(s/9) + \Theta(s)$. Sigui $\alpha = \log_9 m$ i $k = 1$. Llavors, per a $m > 9$, es té $\alpha > k = 1$. En aquest cas, el teorema mestre ens diu que $T(s) = \Theta(s^\alpha)$. Per comparar-ho amb Strassen, ens cal veure quan $\alpha = 2'81/2$. Això es dóna per $m = 9^{1'405} = 21'9$. Per tant, ens cal que $m \leq 21$.

c) El plantejament és el mateix, ara ens cal $\alpha = 2'376/2$, i llavors ens cal que $m \leq 13$.