Solució de l'Examen Final EDA 12/06/2023

Proposta de solució al problema 1

(a) Pel cas $\Theta(\log n)$ considerarem la crida recursiva

return $f(v,e,e) + f(v,e,(e+d)/2)$;

En aquest cas, observem que la crida a f de l'esquerra es calcularà amb temps $\Theta(1)$, mentre que la de la dreta és una crida recursiva on la distància entre e i d s'ha dividit per 2. Per tant, la recurrència que descriu el cost d'aquesta funció és $C(n) = C(n/2) + \Theta(1)$. Si apliquem el teorema mestre per recurrències divisores del tipus $C(n) = a \cdot C(n/b) + \Theta(n^k)$, podem identificar $a = 1$, $b = 2$, $k = 0$ i calcular $\alpha = \log_2(1) = 0$. Com que $k = \alpha$, sabem que la solució és $C(n) = \Theta(n^k \log n) = \Theta(\log n)$.

Pel cas $\Theta(n)$ considerarem la crida recursiva

return $f(v,e,(e+d)/2) + f(v,(e+d)/2,d)$;

En aquest cas, en ambdues crides recursives la distància entre e i d s'ha dividit per 2. Per tant, la recurrència que descriu el cost d'aquesta funció és $C(n) = 2C(n/2) + \Theta(1)$. Si apliquem el teorema mestre per recurrències divisores del tipus $C(n) = a \cdot C(n/b) + \Theta(n^k)$, podem identificar $a = 2$, $b = 2$, $k = 0$ i calcular $\alpha = \log_2(2) = 1$. Com que $\alpha > k$, sabem que la solució és $C(n) = \Theta(n^\alpha) = \Theta(n)$.

(b) Per analitzar el cos de la crida a *cerca* en cas pitjor, considerarem el cas en que s'efectuen les tres crides a funció. Notem que en totes tres, la distància entre els dos últims paràmetres és una tercera part de la distància original. Per analitzar el cost de la crida a *cerca*2 veiem que és una funció no recursiva que, en cas pitjor, travessa els $n/3$ elements que hi ha entre e i d, fent un treball constant per a cadascun d'ells. Per tant el cost d'aquesta crida és $\Theta(n/3) = \Theta(n)$.

Per analitzar la crida a *cerca*3, hem de considerar que és una funció recursiva. Fixem-nos que totes les operacions que s'hi fan són de cost constant, però efectuem, en cas pitjor, una crida recursiva on la distància entre els paràmetres s'ha dividit per 2. Per tant, la la recurrència que descriu el cost d'aquesta funció és $C(n) = C(n/2) + \Theta(1)$. Si apliquem el teorema mestre per recurrències divisores del tipus $C(n) = a \cdot C(n/b) + \Theta(n^k)$, podem identificar $a = 1$, $b = 2$, $k = 0$ i calcular $\alpha = \log_2(1) = 0$. Com que $k = \alpha$, sabem que la solució és $C(n) = \Theta(n^k \log n) = \Theta(\log n)$. Com que, en la crida a *cerca*3 la distància entre els darrers paràmetres és $n/3$, el cost és $\Theta(\log(n/3)) = \Theta(\log n)$.

Finalment, hem de considerar la crida recursiva a *cerca*, on altra vegada, la distància entre els darrers paràmetres s'ha dividit per 3. Per tant, la recurrència que descriu el cost total de la crida a *cerca* és: $C(n) = C(n/3) + \Theta(\log n) + \Theta(n) = C(n/3) + \Theta(n)$. Si apliquem el teorema mestre per recurrències divisores del tipus $C(n) = a \cdot C(n/b) + \Theta(n^k)$, podem identificar $a = 1$, $b = 3$, $k = 1$ i calcular $\alpha = \log_3(1) = 0$. Com que $k > \alpha$, sabem que la solució és $C(n) = \Theta(n^k) = \Theta(n)$.

Proposta de solució al problema 2

Una possible solució és:

```
void to_heap (Node* n, vector<int>& h) {
  if (not n) return;
  to_heap(n→right,h);
  h.push_back(n→key);
  to_heap(n→left,h);
}

vector<int> to_heap (Node* n) {
  vector<int> h(1);
  to_heap(n,h);
  return h;
}
```

Proposta de solució al problema 3

a)
```
bool es_torneig (const Graf& G) {
    int n = G.size();
    for (int u = 0; u < n; ++u) {
        if (G[u][u]) return false;
        for (int v = u+1; v < n; ++v) {
            if (G[u][v] == G[v][u]) return false;
    }   }
    return true;
}
```

b) Base: És clar que un graf torneig amb un vèrtex o dos vèrtexs té un camí Hamiltonià. Inducció: Suposem que tot graf torneig amb n vèrtexs té un camí Hamiltonià. Considerem un graf torneig G amb $n+1$ vèrtexs. Sigui u un vèrtex qualsevol de G. Llavors $G-u$ és un graf torneig de n vèrtexs i, per hipòtesi d'inducció, té un camí Hamiltonià $v_1, v_2, \ldots, v_n$. Si (u, v_1) és un arc de G, llavors $u, v_1, v_2, \ldots, v_n$ és un camí Hamiltonià de G. Si no, (v_1, u) ha de ser un arc de G (perquè és torneig). Si (u, v_2) també és un arc de G, llavors $v_1, u, v_2, \ldots, v_n$ és un camí Hamiltonià de G. Si no, (v_2, u) ha de ser un arc de G (perquè és torneig). Continuant així fins a considerar v_n, arribem a que si (u, v_n) no és un arc de G llavors (v_n, u) ho ha de ser (perquè és torneig), i llavors $v_1, \ldots, v_n, u$ és un camí Hamiltonià de G.

c) Una possible solució és la següent:

```
list <int> cami (const Graf& G) {
    int n = G.size();
    list <int> L;
    if (n == 0) return L;
    if (n == 1) return {0};
    if (G[0][1]) L = {0, 1}; else L = {1, 0};
```

```
    for (int u = 2; u < n; ++u) insereix (L, u, G);
    return L;
}

void insereix ( list <int>& L, int u, const Graf& G) {
    auto it1 = L.begin ();
    if (G[u][*it1 ]) { L.push_front (u); return; }
    auto it2 = it1 ; ++it2;
    while (it2 ≠ L.end()) {
        if (G[*it1 ][u] and G[u][*it2 ]) { L. insert (it2 , u); return; }
        ++it1; ++it2;
    }
    L.push_back(u);
}
```

Proposta de solució al problema 4

(a) **No sabem** si aquesta afirmació és certa o falsa. Si fos certa, podríem demostrar que $P = NP$, que és un problema obert. En efecte, ja sabem que $P \subseteq NP$. Per veure l'altra inclusió, prenem un problema $C \in NP$ qualsevol i vegem que pertany a P. Com que B és NP-difícil i $C \in NP$, podem reduir C cap a B, i com que B es pot reduir cap a A, composant les dues reduccions sabem reduir C cap a A. L'algorisme que primer redueix C cap a A i a continuació resol A (en temps polinòmic) és un algorisme polinòmic pel problema C. Per tant $C \in P$.

Si fos falsa, aleshores per a qualsevol parell de problemes $A \in P$ i $B \in NP$-difícil, no podríem reduir B cap a A. Si prenem B que sigui NP-complet (en particular NP-difícil), aleshores tindríem un problema $B \in NP$ que no es pot reduir a $A \in P$. Com tots els problemes de P es poden reduir entre ells, això implicaria que B no pertany a P i, per tant, que les classes P i NP no són iguals, i hauríem resolt un problema obert.

(b) Aquesta afirmació és **certa**. Considerem A: determinar si un vector està ordenat i B: trobar un cicle hamiltonià en un graf. Com que $A \in P$, també pertany a NP. I com que B és NP-complet (en particular NP-difícil), segur que existeix una reducció d'A cap a B (per la definició de problema NP-difícil).

(c) Aquesta afirmació és **certa**. Sigui A el problema del 3-colorejat de grafs i B el problema de trobar un cicle Hamiltonià en un graf. Són tots dos problemes NP-complets (i per tant, també NP-difícils). Com que sabem que tots els problemes NP-complets es poden reduir entre ells, l'afirmació és certa.