Problema 1. Algorismes voraços (2 punts)

Avui, en Jonny, en Roy i l'Steffy han anat a esquiar amb la seva colla per oblidar-se dels exàmens. Falta poc perquè es faci fosc, i volen saber si tenen temps d'agafar un telecadira per pujar al cim de la muntanya i llavors baixar tots alhora per una excitant pista negra.

JONNY: Ostres, aquí tenen un telecadira de dues places! Quina andròmina! I a sobre diu que el pes màxim per cadira es de k quilos. Quina cutrada!

ROY: Vinga, distribuïm-nos en grups de com a molt dues persones de manera que ocupem el mínim nombre de cadires. Jonny, recull els pesos de la gent de la colla.

JONNY: Aquí els tens: $p_1, \ldots, p_n$ (és la mateixa llista de pesos de quan vam anar a fer ràfting). Però escolta, aquest problema no és idèntic al del BIN-PACKING, on cada cadira és un contenidor de capacitat k?

ROY: És veritat. I no es coneix cap algorisme polinòmic per BIN-PACKING. Quan tinguem solucionat el problema ja serà negra nit...

STEFFY: Nois, no us precipiteu. El nostre problema té la restricció addicional que cada cadira només admet dues persones o menys. Amb aquesta restricció el problema es pot solucionar ràpidament.

Descriviu (sense donar codi ni pseudocodi però amb un mínim de justificació) un algorisme voraç per resoldre el problema del telecadira biplaça. Digueu quin és el cost de la vostra solució.

Problema 2. Cerca exhaustiva (2 punts)

Donat un graf no dirigit $G = (V, E)$ i un subconjunt $S \subset V$ dels seus vèrtexs, el tall de S és el nombre d'arestes que tenen un extrem en S i un altre fora de S, és a dir,

$$\text{tall}(S) = |\{\ \ \{u, v\} \in E\ :\ u \in S \wedge v \notin S\ \ \}|.$$

El problema del tall màxim (MAXCUT) consisteix en, donat un graf G, determinar el seu tall màxim. Se sap que la versió decisional de MAXCUT és **NP**-completa.

Completeu la classe següent en C++ tot escrivint un algorisme de *backtracking* que determini el tall màxim d'un graf. Analitzeu el cost de la vostra solució.

Observeu que els grafs (no dirigits) es troben implementats amb matrius d'adjacència. Per tant, `G[u][v]` és un booleà indicant si `u` i `v` es troben connectats en `G`, i val el mateix que `G[v][u]`.

Problema 3. NP-completesa (1 punt)

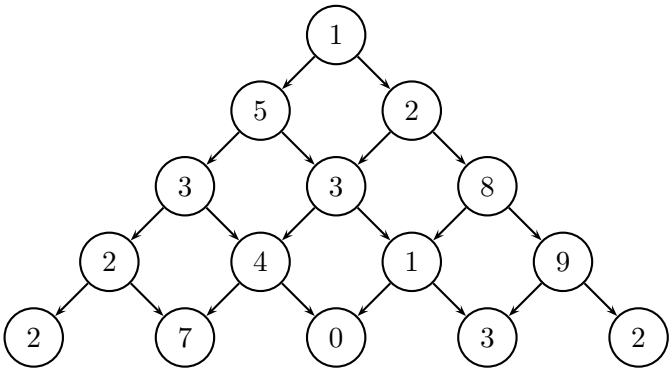
Se sap que un cert problema L és **NP**, però no es coneix cap algorisme que resolgui L en temps polinòmic. Contesteu de forma breu però raonada les preguntes següents:

a) Si es trobés un algorisme polinòmic que resolgués L, això implicaria $\mathbf{P} = \mathbf{NP}$?

b) Si es demostrés que no existeix cap algorisme polinòmic que resolgui L, això implicaria $\mathbf{P} \neq \mathbf{NP}$?

Problema 4. Programació dinàmica (2 punts)

És ben conegut que els ratolins mostren una gran intel·ligència quan els científics els tanquen dins de laberints amb habitacions, passadissos i formatges. Considereu laberints triangulars com el de la figura següent:



El ratolí comença a la posició superior i descriu un camí de dalt cap abaix, podent anar a cada pas cap a baix a l'esquerra o bé cap a baix a la dreta (el ratolí mai pot tornar cap amunt). Cada cop que passa per una habitació del laberint, pot menjar tantes unitats de formatge com indica el número en aquella habitació.

Per exemple, si al laberint de la figura anterior el ratolí descriu el camí ⟨esquerra, esquerra, esquerra, esquerra⟩ llavors podrà menjar 13 unitats de formatge. En canvi, si el ratolí descriu el camí ⟨dreta, dreta, dreta, esquerra⟩ llavors podrà menjar 23 unitats de formatge. Fixeu-vos que no hi ha cap camí pel qual el ratolí pugui afartar-se més.

Representem un laberint amb n nivells amb una matriu M amb n files i n columnes, utilitzant només la meitat inferior que defineix la diagonal que va de l'element $(1,1)$ a l'element (n,n). Els elements al triangle superior de la matriu no es poden consultar. Per exemple, per al laberint anterior,

$$M = \begin{bmatrix} 1 & & & & \\ 5 & 2 & & & \\ 3 & 3 & 8 & & \\ 2 & 4 & 1 & 9 & \\ 2 & 7 & 0 & 3 & 2 \end{bmatrix}.$$

Per saber quina és la quantitat màxima de formatge que pot menjar-se un ratolí en un laberint donat, definim la funció millor(i,j) com la suma dels números del millor camí que comença al j-èsim element de la fila i. L'objectiu és doncs calcular $\mathit{millor}(1,1)$.

a) Escriviu una recurrència per a $\mathit{millor}(i,j)$.

b) Digueu de forma justificada quin cost en temps i en espai tindria l'algorisme de programació dinàmica resultant (no heu d'escriure l'algorisme, només heu de donar el seu cost).

Problema 5. Estructures de dades (3 punts)

Donada una taula $T[0 \,..\, n-1]$ amb n enters i un índex i (amb $0 \leq i < n$), la i-èsima suma parcial de T és $\sum_{0 \leq j \leq i} T[j]$. Considereu un TAD que permeti calcular sumes parcials de taules. En particular, les operacions considerades són:

- `create(`n`)`: Crea una taula $T[0 \,..\, n-1]$ amb tots els elements inicialitzats a 0.
- `set(`i`,`x`)`: Assigna l'enter x a la posició i de la taula (amb $0 \leq i < n$).

- `get(`i`)`: Retorna el valor de la posició i de la taula (amb $0 \leq i < n$).
- `sum(`i`)`: Retorna la i-èsima suma parcial de la taula (amb $0 \leq i < n$).

Implementeu en C++ aquest TAD seguint la interfície de la classe que es dóna a continuació, fent que les operacions `set` i `sum` tinguin cost $\mathrm{O}(\log n)$, que `get` tingui cost constant, i que `create` tingui cost $\mathrm{O}(n)$. No podeu usar punters.

Dibuixeu l'estat de la vostra estructura de dades quan es té una taula amb els valors $[12, 41, 31, 12, -10, 13, -21, 12]$.

Observacions sobre la solució:

- Les solucions proposades no són úniques.
- Els textos entre claudàtors proporcionen aclariments o informació addicional que no es demanava a l'examen.

Solució 1

Ordenem primer les persones per ordre creixent de pes: $p_1 \leq p_2 \leq \cdots \leq p_n$.

Considerem ara la persona amb major pes (p_n). Aquesta persona per força ha de pujar en una cadira. La qüestió és si algú pot acompanyar-la.

Si la persona més lleugera (pes p_1) no pot acampanyar-la (perquè $p_1 + p_n > k$), per força la persona amb major pes haurà d'anar sola. Ara podem continuar amb la resta de les $n - 1$ persones.

Altrament, ens interessa compartir la cadira amb la persona que tingui el pes més gran possible sense excedir $k - p_n$. Aquestes dues persones compartiràn cadira. Ara podem continuar amb la resta de les $n - 2$ persones.

Es pot implementar l'algorisme en temps $\mathrm{O}(n \log n)$.

[Hi ha una solució semblant consistint en aparellar el més pesat amb el més lleuger, si es pot. De fet, aparellar el més pesat amb qualsevol altre amb qui es pugui també és correcte.]

[Errors freqüents: No justificar perquè funciona l'algorisme proposat.]

Solució 2

a) No. Això voldria dir que L és a la classe $\mathbf{P}$ i prou. [Si L fós $\mathbf{NP}$-complet, llavors sí que $\mathbf{P} = \mathbf{NP}$.]

b) Sí: Per una banda sabem que $L \in \mathbf{NP}$. Per altra banda, sabem que $L \notin \mathbf{P}$. Com que $\mathbf{P} \subseteq \mathbf{NP}$, tenim $\mathbf{P} \neq \mathbf{NP}$.

[Errors freqüents: Encertar la resposta donant justificacions incorrectes.]

Solució 4

a)

$$\text{millor}(i,j) = \begin{cases} M[i,j] + \max\{\ \text{millor}(i+1,j)\ ,\ \text{millor}(i+1,j+1)\ \} & \text{si } i \leq n, \\ 0 & \text{si } i > n. \end{cases}$$

b) Es pot calcular $\text{millor}(i,j)$ de baix cap a dalt, amb cost espacial i temporal $\Theta(n^2)$. [Es pot reduir fàcilment l'espai a $\Theta(n)$ guardant només les dues darres files de la matriu.]

Solució 3

```
class MaxCut {

private:

    graf G;                  // el graf
    int n;                   // nombre de vèrtexs de G
    int tm;                  // tall màxim
    vector<bool> S;          // indica, per a cada vèrtex, si és o no a S

    int tall () {            // Retorna el tall de S
        int t = 0;
        for (int u=0; u<n; ++u) {
            for (int v=u+1; v<n; ++v) {
                if (G[u][v] and S[u]!=S[v]) {
                    ++t;
        }   }   }
        return t;
    }

    int backtracking (int u) {
        if (u==n) {
            return tall();
        } else {
            S[u] = false;  int t0 = backtracking(u+1);
            S[u] = true;   int t1 = backtracking(u+1);
            return max(t0,t1);
    }   }

public:

    MaxCut (graf G) {
        this->G = G;
        n = G.rows();
        S = vector<bool>(n);
        tm = backtracking (0);
    }

    int tall_maxim () { return tm; }
};
```

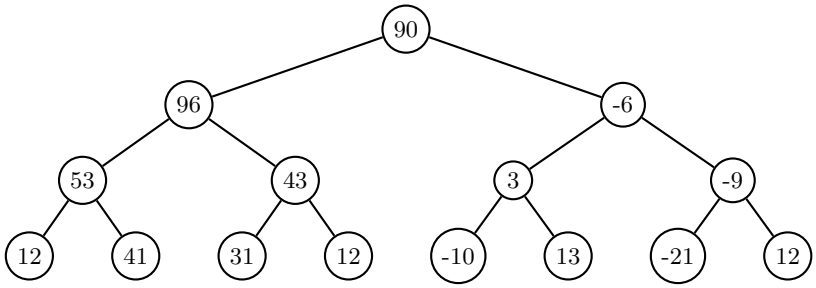
El cost és $\Theta(2^n n^2)$ perquè hi ha 2^n solucions possibles i per a cadascuna es calcula el tall amb cost $\Theta(n^2)$.

[Errors freqüents:

- Deixar-se de calcular el cost de la solució proposada.
- No explorar bé l'espai de cerca: Cada vèrtex pot estar o no estar en S. Es tracta, doncs, de generar el conjunt de totes les parts de V, no pas el conjunt de totes les permutacions dels vèrtexs.
- No sortir corrent de l'examen per anar cobrar un milió de dolars quan s'ha demostrat que $\mathbf{P} = \mathbf{NP}$ trobat un algorisme polinòmic per un problema **NP**-complet.]

Solució 5

Implementem l'estructura de dades descrita a la solució de l'examen parcial:




```
class SumesParcials {

    vector<int> t;              // taula que descriu l'arbre
    int m;                      // següent potència de 2 de n

public:

    SumesParcials (int n) {
        for (m=1; m<n; m*=2);
        t = vector<int>(2*m,0);
    }

    void set (int i, int x) {
        int j = m + i;
        int d = x - t[j];
        while (j != 0) {
            t[j] += d;
            j /= 2;
    }   }

    int get (int i) {
        return t[m+i];
    }

    int sum (int i) {
        int j = m+i;
        int s = t[j];
        while (j != 0) {
            if (j%2 != 0) s += t[j-1];
            j /= 2;
        }
        return s;
    }
};
```

[Errors freqüents: No haver-se llegit la solució de l'examen parcial. Donar solucions de temps lineal. Usar punters.]