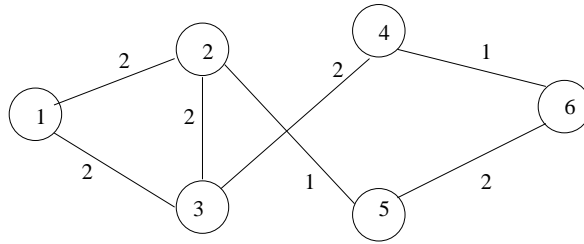


## SEGON PARCIAL, 23 de Juny de 2006

### Problema 1

[1 punt]

Sigui  $G = (V, E)$  el graf de la següent figura:



Dibuixeu tots els arbres d'expansió mínims de  $G$ .

### Problema 2 - Dijkstra

[2 punts]

Sigui  $G = (V, E)$  un graf dirigit i etiquetat amb pesos positius i  $vini$  i  $vfinal$  dos vèrtexs qualsevol del graf. Introduïu les modificacions que calguin a l'algorisme de Dijkstra que trobareu a continuació per tal que:

1. Calculi el pes del camí de pes mínim des de  $vini$  a  $vfinal$
2. Compti el número de camins diferents amb pes mínim entre els dos vèrtexs. Dos camins són diferents si difereixen com a mínim en una aresta.

Cal que indiqueu com es fa la crida inicial a Dijkstra i que justifiqueu les modificacions introduïdes, les estructures de dades auxiliars, etc.

```

class Dijkstra {

private:
    struct aresta{
        int u;          // vèrtex d'origen
        int v;          // vèrtex destí
        double p;       // pes de l'aresta
        aresta(int x, int y, double pp):u(x), v(y), p(pp){}
    };
    vector<double> d;

public:
    Dijkstra (wgraph& G, int u) {

        // inicialització
        int n = G.size();
        vector<boolean> S(n,false);
        d = vector<double>(n,infin);
        d[u] = 0;
        priority_queue<aresta> CP;
        CP.push(aresta(-1,u,0));

        // bucle principal
        while (not CP.empty()) {
            aresta a = CP.top(); CP.pop();
            int v = a.v;
            if (not S[v]) {
                S[v] = true;
                forall_adj(vw,G[v]) {
                    int w = vw->v;
                    if (not S[w]) {
                        if (d[w] > d[v]+vw->p) {
                            d[w] = d[v]+vw->p;
                            CP.push(aresta(v,w,d[w]));
                        }
                    }
                }
            }
        }
    };
};

```

### Problema 3 - La grip aviària (Cerca Exhaustiva)

[2 punts]

La grip aviària és una malaltia contagiosa en animals causada per virus que normalment afecten aus i, menys habitualment, porcs. Entre aus de granja es contagia ràpidament i té una mortalitat que pot arribar al 100%, sovint en només 48 h.

En un laboratori de referència s'està estudiant el cep més perillós del virus entre una mostra d'aus salvatges recollides a les nostres costes. El laboratori disposa de  $n^2$  gàbies col·locades formant un quadrat  $n \times n$ .

Es vol *col·locar el màxim nombre d'aus a les gàbies del laboratori*, però s'han de tenir en compte les restriccions següents:

- Algunes gàbies estan espatllades i no es poden utilitzar.
- Cada gàbia només pot contenir una au, com a molt.
- Quan es col·loca una au en una gàbia, les gàbies del voltant han de quedar buides (per tal d'evitar contagis).

Per solucionar aquest problema podríem fer cerca exhaustiva, tot modelitzant les gàbies com un taulell  $n \times n$  (vegeu el dibuix).

1	AU	×		■	
2	×	×			
3		■			
4					
5	■				■
	1	2	3	4	5

■ : gàbia espatlada  
 × : gàbia que s'ha de deixar lliure per evitar contagis

Figura 1: Les gàbies del laboratori vistes com un taulell

Es demana:

- Completeu la proposta d'algorisme de backtracking que trobareu a la plana següent per tal que calculi quin és el nombre màxim d'aus que es poden estudiar al laboratori.

```


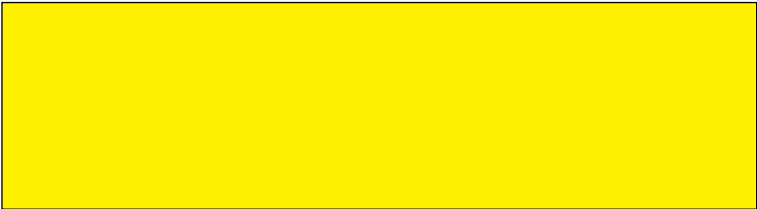
#include "ada.hh"

class Laboratori {
    int n; // nombre de files i columnes
    matrix<int> G; // les gàbies del laboratori
    int maxim; // màxim nombre d'aus que es poden engabiar

    static const int ESPATLLADA = -1;
    static const int LLIURE = 0;
    static const int OCUPADA = 1;

    bool dintre_laboratori (int x, int y) {
        return (x>=0 and x<n and y>=0 and y<n);
    }

    bool perill_contagi (int x, int y) {
        // Determina si la cassella (x,y) està en perill de contagi
        return ((dintre_laboratori(x-1,y-1) and G[x-1][y-1]==OCUPADA) or
                (dintre_laboratori(x-1,y) and G[x-1][y]==OCUPADA) or
                (dintre_laboratori(x-1,y+1) and G[x-1][y+1]==OCUPADA) or
                (dintre_laboratori(x,y-1) and G[x][y-1]==OCUPADA));
    }

    void recursiu (int cas, int nombre_aus) {
        if (cas==n*n) {
            
        } else {
            
        }
    }
}

public:

    GripAviaria (int n) {
        this->n = n;
        G = matrix<int>(n,n,LLIURE);
        maxim = 0;
    }

    void inhabilita (int x, int y) {
        if (dintre_laboratori(x,y)) G[x][y]=ESPATLLADA;
    }

    int configura () {
        recursiu(0,0);
        return maxim;
    }
};

// PROGRAMA PRINCIPAL

int main () {
    int n, x, y;
    cin >> n;
    Laboratori lab(n);
    while(cin >> x) { // Llegeix la informació de gàbies espatllades
        cin >> y;
        lab.inhabilita(x,y);
    }
    cout << lab.configura() << endl;
}

```

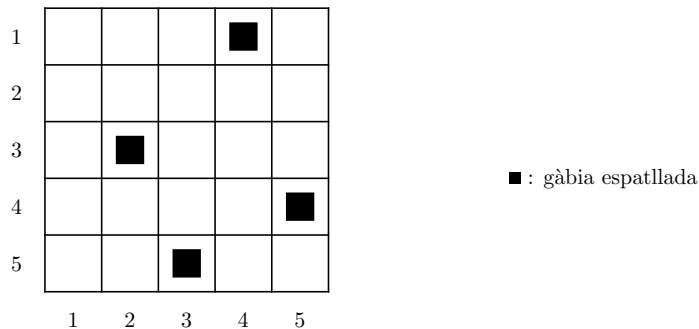
- b) Doneu la versió decisional del problema i demostreu que és a NP.

### Problema 4 - La grip aviària (Programació Dinàmica) [3 punts]

Recordeu que al laboratori de referència del problema anterior es troben engabiades aus salvatges per estudiar-ne la grip aviària. El laboratori disposa de  $n^2$  gàbies col·locades formant un quadrat  $n \times n$ , a on algunes gàbies es troben espatllades (vegeu el dibuix).

Amb les aus a estudiar ja col·locades per dur a terme els estudis pertinents, el becari del laboratori ha d'anar a una gàbia  $(x, y)$  determinada i, si està ocupada, prendre un seguit de mostres de l'au que hi viu.

Degut a la distribució del laboratori, el recorregut del becari ha de començar a la gàbia  $(1, 1)$ . Per avançar pel laboratori des d'una determinada gàbia  $(i, j)$ , el becari només pot accedir a la següent gàbia de la dreta o a la següent gàbia de sota.



Tenint en compte que passar per una gàbia no espatllada té un cost constant  $c_1$  i que passar per una gàbia espatllada té un cost constant  $c_2$  (amb  $1 \leq c_1 < c_2$ ), es planteja el problema següent: *calcular el cost del camí més barat que pot conduir al becari des de la gàbia  $(1, 1)$  a la gàbia  $(x, y)$  donada*. Es demana:

- Expliqueu per què té sentit aplicar l'esquema de Programació Dinàmica per resoldre aquest problema.
- Doneu una recurrència que el resolgui, explicant clarament què descriu el terme de la recurrència. Quin terme d'aquesta recurrència calcula l'objectiu del problema?
- Descriviu l'estructura de dades auxiliar que guardarà els resultats dels subproblemes ja resolts i expliqueu amb detall com s'omple l'estructura.
- Justifiqueu quin és el cost, espacial i temporal, de la vostra solució.

### Problema 5 - La grip aviària (Algorisme Voraç) [2 punts]

Al laboratori de referència dels problemes anteriors, les aus salvatges contagiades del virus de la grip aviària arriben a una velocitat vertiginosa. El laboratori disposa d'un algorisme voraç per col·locar ràpidament les aus dins les  $n \times n$  gàbies seguint les restriccions següents:

- Algunes gàbies no es poden utilitzar perquè estan espatllades.

- Cada gàbia només pot contenir una au, coma a molt.
- Quan es col·loca una au en una gàbia, les gàbies del voltant han de quedar buides.
- La primera au ha d'anar a una posició  $(x, y)$  especificada pels científics, la qual mai està espatllada.

L'algorisme voraç que fa servir el laboratori consisteix a recórrer fila per fila el taulell de gàbies, començant i acabant a la posició  $(x, y)$  on s'ha posat la primera au. Per cada posició considerada, es posarà una au dins la gàbia d'aquella posició sempre que les restriccions anteriors ho permetin.

Per exemple, suposeu que les gàbies del laboratori descriuen un taulell  $5 \times 5$  com el del dibuix, i que totes les gàbies de la diagonal estan espatllades. Si la primera au ha d'anar a la gàbia  $(1, 4)$ , l'algorisme voraç faria el següent:

1. Acabaria de recórrer la fila 1 a partir de la posició 5, i trobaria que no pot col·locar cap au en la gàbia  $(1, 5)$ .
2. Passaria a explorar la fila 2, i trobaria que a la columna 1 pot posar una au. En explorar la resta de columnes d'aquesta fila, trobaria que no pot posar més aus.
3. De la mateixa manera, l'algorisme exploraria les files 3, 4 i 5. Finalment, acabaria d'explorar la fila 1, des de la columna 1 fins a la columna 3 (ja que a la columna 4 s'hi troba la primera au col·locada).

Al final, l'algorisme hauria col·locat aus a les gàbies:  $(1, 4)$ ,  $(2, 1)$ ,  $(3, 4)$ ,  $(4, 1)$  i  $(5, 3)$ .

1	■	×	×	AU <sub>1</sub>	×
2	AU <sub>2</sub>	■	×	×	×
3	×	×	■	AU <sub>3</sub>	×
4	AU <sub>4</sub>	×	×	■	×
5	×	×	AU <sub>5</sub>	×	■
	1	2	3	4	5

Es demana:

- a) Justifiqueu (raonadament) si l'algorisme voraç del laboratori és òptim, és a dir, si sempre col·loca el màxim nombre d'aus dintre de les gàbies disponibles.
- b) Proposeu un algorisme voraç diferent a l'anterior i justifiqueu perquè és voraç.

## SOLUCIONS

### Problema 1 (al final de tot)

#### Problema 2

L'algorisme de Dijkstra calcula el pes del camí de pes mínim des de  $u$  a la resta de vèrtexs del graf. En el nostre cas només volem que calculi el pes del camí de pes mínim des de  $vini$  a  $vfinal$  i per això modifiquem la capçalera afegint un nou vèrtex, el  $vfinal$ , i fent que  $u = vini$ . La crida inicial serà: `Dijkstra(G, vini, vfinal)`. Al codi hem de controlar en quin moment arribem a tractar el vèrtex  $vfinal$ . El booleà `fin` ens serveix per determinar en quin moment aquest vèrtex surt de la cua de prioritat el que vol dir que el seu valor de `d[vfinal]` ja conté el pes del camí de pes mínim des de  $vini$  a  $vfinal$  i, per tant, podem aturar el bucle. També hem de prendre nota del número de camins de pes mínim que arriben a un vèrtex qualsevol (en particular ens interessa la informació del vèrtex  $vfinal$ ). El vector `NCM` porta un comptador per cada vèrtex visitat. Per un vèrtex  $u$  qualsevol si la seva `d[u]` canvia (només pot reduir-se) és perquè el camí de pes mínim passa per el vèrtex, per exemple  $w$ , que acabem de marcar a `vist` (`S[w]` s'acaba de posar a cert) i, en aquest cas, el número de camins de pes `d[u]` que arriben a  $u$  és exactament el mateix que els que arriben a  $w$  perquè tots passen per  $w$ . L'altre cas és quan s'arriba a  $u$ , passant per  $w$ , amb un pes idèntic al que ja tenia  $u$ ,

llavors el número de camins amb pes  $d[u]$  és els que ja tenia  $u$  més tots aquells que arriben a  $w$ .

```
class Dijkstra {

private:
    struct aresta{
        int u;           // vèrtex d'origen
        int v;           // vèrtex destí
        double p;        // pes de l'aresta
        aresta(int x, int y, double pp):u(x), v(y), p(pp){}

        vector<double> d;

    public:

        Dijkstra (wgraph& G, int u, int vf) { // hem afegit un paràmetre

            // inicialització
            int n = G.size();
            vector<boolean> S(n,false);
            vector<int>      NCM(n,0); // vector amb el número de camins de pes
                                     // mínim que arriben als vèrtexs des de u
            NCM[u]=1; // hi ha un camí de pes mínim des de u a ell mateix.
                     // u és el vèrtex inicial, vini, i vf es el vèrtex final.
            d = vector<double>(n,infin);
            d[u] = 0;
            priority_queue<aresta> CP;
            CP.push(aresta(-1,u,0));

            // bucle principal
            boolean fin=false; // per aturar-nos en tractar el vèrtex final.
            while ((not CP.empty())&&(not fin)) {
                aresta a = CP.top(); CP.pop();
                int v = a.v;
                if (not S[v]) {
                    S[v] = true;
                    if (v==vf) {fin=true;} // acabem si el vèrtex és el final.
                }
            }
            // tractament habitual
            forall_adj(vw,G[v]) {
                int w = vw->v;
                if (not S[w]) {
                    if (d[w] > d[v]+vw->p) {
                        d[w] = d[v]+vw->p;
                        NCM[w] = NCM[v]; // el número de camins que arriben
                        // a w amb pes d[w] és el mateix que els que arriben a v
                        CP.push(aresta(v,w,d[w]));}
                    else if (d[w] == d[v]+vw->p) {
                        NCM[w] = NCM[w] + NCM[v];}
                } } } } } }

};
```

## Problema 3

a)

```
void recursiu (int cas, int nb_au) {
    if (cas==n*n) {
        if (nb_au>maxim) maxim = nb_au;
    } else {
        int i = cas/n, j = cas%n;
        if (G[i][j]==ESPATLLADA or perill_contagi(i,j)) {
            recursiu(cas+1,nb_au);
        } else {
            G[i][j] = OCUPADA;
            recursiu(cas+1,nb_au+1);
            G[i][j] = LLIURE;
            recursiu(cas+1,nb_au);
        }
    }
}
```

- b) Sigui  $T$  un taulell  $n \times n$  que representa les gàbies i té marcades les que no estan disponibles.

**Versió funcional:** Donat  $T$ , trobar el màxim nombre d'aus que s'hi poden col·locar tenint en compte les restriccions que (1) les gàbies marcades a  $T$  no es poden utilitzar perquè estan espatllades, (2) cada gàbia només pot contenir una au, i (3) quan es col·loca una au en una gàbia, les gàbies del voltant han de quedar buides.

**Versió decisional:** Donat  $T$  i un enter  $k$ , decidir si es poden col·locar al menys  $k$  aus a  $T$ , tot respectant les restriccions que (1) les gàbies marcades a  $T$  no es poden utilitzar perquè estan espatllades, (2) cada gàbia només pot contenir una au, i (3) quan es col·loca una au en una gàbia, les gàbies del voltant han de quedar buides.

Observeu que, fent servir la versió decisional del problema i variant el valor de  $k$  (per exemple com indica una cerca dicotòmica), podem resoldre la versió funcional del problema.

Aquest problema és a NP perquè hi ha un testimoni de mida polinòmica (un taulell amb aus col·locades a certes caselles) i un algorisme verificador que rep el taulell  $T$  (amb les gàbies espatllades), la  $k$  i el taulell  $TE$  (testimoni) i en temps polinòmic el verificador torna cert si  $TE$  conté  $k$  o més aus col·locades tot respectant les restriccions de  $T$  i de la manera de disposar les aus (torna fals quan falla alguna de les condicions). Quan aquest verificador torna cert podem assegurar que la versió decisional del problema té solució i que el testimoni  $TE$  és una de elles.

## Problema 4

- a) Perquè el problema es pot dividir en subproblemes i compleix els dos criteris necessaris per aplicar programació dinàmica: la propietat de *subestructura òptima* (solucions òptimes a subproblemes es poden utilitzar per trobar solucions òptimes al problema complet), i *subproblemes repetits o solapats* (el mateix subproblema s'ha de resoldre diverses vegades en intentar resoldre el problema complet).
- b) Sigui  $C(i, j)$  el cost del camí més barat per arribar des de la gàbia  $(1, 1)$ , fins la gàbia  $(i, j)$  (només amb moviments cap a la dreta i cap a baix). Segon això, l'objectiu del problema és calcular  $C(x, y)$ .

Aleshores,  $C(i, j)$  es calcula segons la següent recurrència:

$$C(i, j) = \begin{cases} \chi(i, j) & \text{si } i = j = 1 \\ \chi(i, j) + C(i, j - 1) & \text{si } i = 1, 1 < j \leq n \\ \chi(i, j) + C(i - 1, j) & \text{si } j = 1, 1 < i \leq n \\ \chi(i, j) + \min \{C(i - 1, j), C(i, j - 1)\} & \text{altrament} \end{cases}$$



on  $\chi(i, j)$  és una variable indicadora del fet que la gàbia  $(i, j)$  està espatllada o no,<sup>1</sup> és a dir:

$$\chi(i, j) = \begin{cases} c_1 & \text{si la gàbia } (i, j) \text{ no està espatllada} \\ c_2 & \text{si la gàbia } (i, j) \text{ està espatllada} \end{cases}$$

En la recurrència, el cas base correspon al cost de passar per la gàbia  $(1, 1)$ . El primer cas recursiu correspon a emplenar la primera “fila” del laboratori. El segon cas recursiu correspon a emplenar la primera “columna” del laboratori.

- c) Es va considerar correcta com a resposta una matriu  $\mathbb{C}$  de talla  $x \times y$  (o de talla  $n \times n$ ), on  $C[i][j] = C(i, j)$  tot i que amb un vector de  $y$  posicions hi ha prou ja que només interessa la darrera fila calculada de la matriu anterior. La següent fila es calcula sobreescrivint la fila  $i$  de dreta a esquerra. Per omplir la matriu, s’han de respectar les dependències entre valors que descriu la recurrència. Una forma d’emplenar la matriu respectant aquestes restriccions seria emplenant-la per files (però n’hi ha d’altres).
- d) El cost temporal és  $\Theta(xy)$  o  $O(n^2)$  (quan el becari ha d’anar a la gàbia  $(n, n)$ ). I com que amb una fila de mida  $y$  n’hi ha prou, el cost espacial és  $\Theta(y)$  o  $O(n)$ .

## Problema 5

- a) L’algorisme proposat no és òptim, ja que al taulell donat és possible posar més de 5 aus (vegeu el dibuix). Aleshores, hem trobat un contraexemple.

1	■	AU <sub>2</sub>	×	AU <sub>1</sub>	×
2	×	■	×	×	×
3	AU <sub>3</sub>	×	■	×	AU <sub>4</sub>
4	×	×	×	■	×
5	AU <sub>5</sub>	×	AU <sub>6</sub>	×	■
	1	2	3	4	5

■ : gàbia no disponible

× : gàbia que s’ha de deixar lliure per evitar contagis

- b) L’algorisme voraç que proposem consisteix a posar aus a totes les gàbies on sigui possible recorrent el taulell començant per les cantonades (posicions  $(1, 1)$ ,  $(1, n)$ ,  $(n, n)$ ,  $(n, 1)$ ), seguint pels costats (fila 1, columna  $n$ , fila  $n$ , columna 1) i repetint el mateix procediment al taulell  $[2 \dots n - 1] \times [2 \dots n - 1]$ . D’aquesta manera deixem lliure el màxim nombre de gàbies per continuar posant aus, ja que escollim primer aquelles que invaliden el mínim nombre de gàbies possible.

<sup>1</sup>També es podria prescindir d’aquesta variable i expressar tots els casos de la recurrència (els casos quan  $\chi(i, j) = c_1$  i els casos quan  $\chi(i, j) = c_2$ ) per separat.

## Problema 1

Hi ha vuit MST, tots de pes 8.

