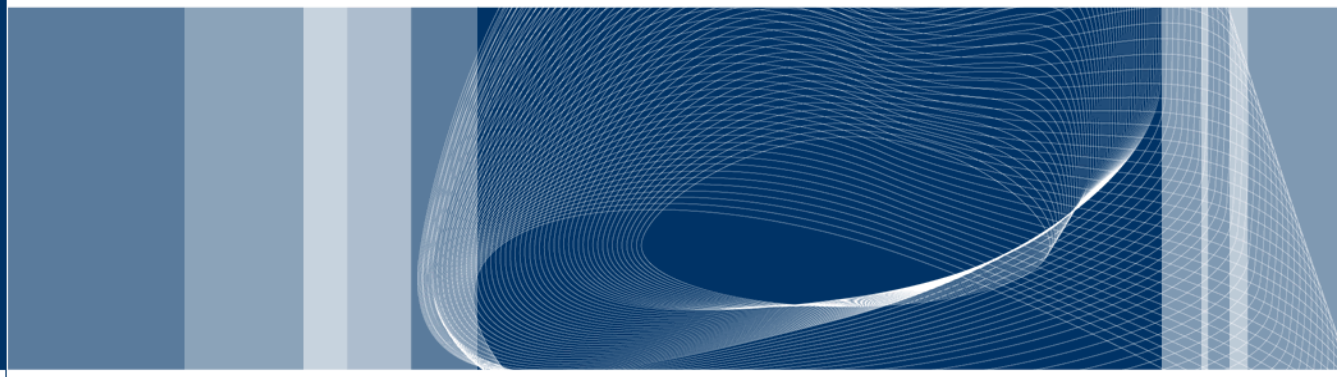
POLITECNICO DI MILANO



Esercizi Vari

Algoritmi, Strutture Dati e Calcolo Parallelo

Daniele Loiacono

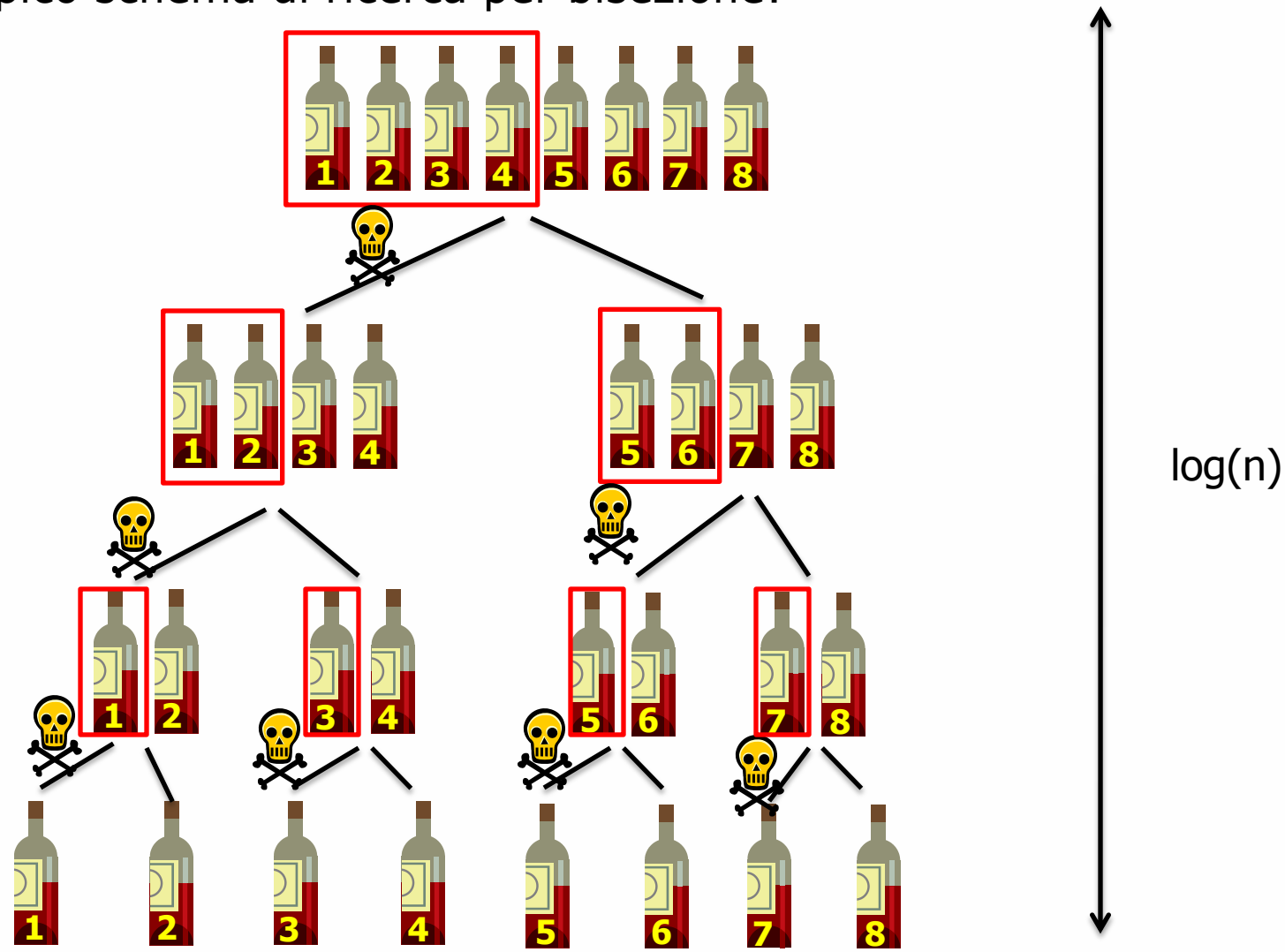
Ricerca e Complessità

Esercizio 1

- Un re malvagio possiede n bottiglie di un vino pregiato e le sue guardie hanno appena catturato una spia che stava tentando di avvelenare il vino del re. Per fortuna le guardie hanno catturato la spia dopo che era riuscita ad avvelenare solo una bottiglia. Le guardie però, non sono riuscite ad individuare la bottiglia. Il veleno che usa la spia è molto potente, una goccia diluita un miliardo di volte potrebbe ancora uccidere. Il veleno funziona lentamente, affinché sopraggiunga la morte è necessario un mese intero. Progettare uno schema che determini con precisione quale delle bottiglie sia stata avvelenata impiegando un numero di assaggiatori O(log(n)) e un tempo non superiore a O(log(n)) mesi.

Esercizio 1 - Soluzione

- Tipico schema di ricerca per bisezione:



Esercizio 1bis

- Quanto impiega la soluzione data in precedenza per individuare la bottiglia avvelenata?
- È possibile introdurre uno schema che usi log(n) assaggiatori e consenta di trovare la bottiglia avvelenata in un mese al massimo?
- In caso affermativo, descrivere tale schema. In caso negativo, giustificare la risposta.

Hashing

Esercizio 1

- Si consideri una tabella di hash con 13 slot (numerati da 0 a 12)
- Alla tabella si accede con hashing doppio:
  - h1(k) = k mod 13
  - h2(k) = Rev(k+1) mod 11
- Dove Rev è una funzione che restituisce un numero ottenuto invertendo le cifre di k (e.g. Rev(37) = 73, Rev(7) = 7).
- Si mostri la tabella dopo l'inserimento delle chiavi: 2, 8, 31, 20, 19, 18, 53, 27

Esercizio 1 – Soluzione

- H1(2) = 2

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			2										

Esercizio 1 – Soluzione

- H1(8) = 8

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			2						8				

Esercizio 1 – Soluzione

- H1(31) = 5

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			2			31			8				

Esercizio 1 – Soluzione

- H1(20) = 7

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			2			31		20	8				

Esercizio 1 – Soluzione

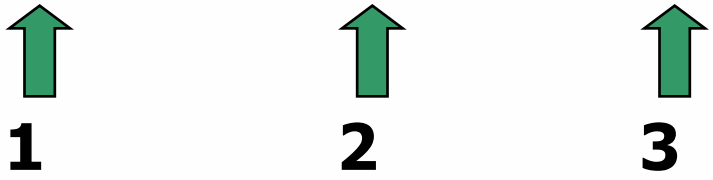
- H1(19) = 6

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			2			31	19	20	8				

Esercizio 1 – Soluzione

- H1(18) = 5
- Rev (19) = 91 → H2 (18) = 3

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key			2			31	19	20	8			18	



Esercizio 1 – Soluzione

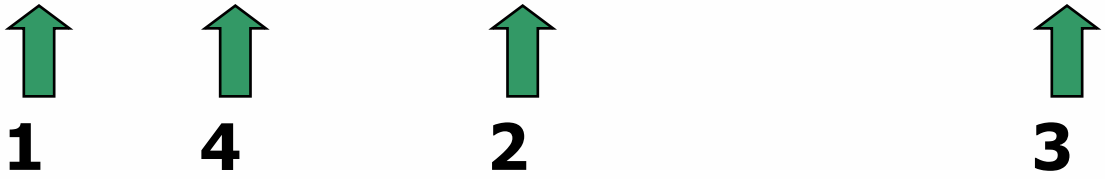
- H1(53) = 1

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key		53	2			31	19	20	8			18	

Esercizio 1 – Soluzione

- H1(27) = 1
- Rev (28) = 82 → H2(27) = 5

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key		53	2	27		31	19	20	8			18	



Esercizio 1 – Soluzione

- Stato finale:

Pos	0	1	2	3	4	5	6	7	8	9	10	11	12
Key		53	2	27		31	19	20	8			18	

Esercizio 4

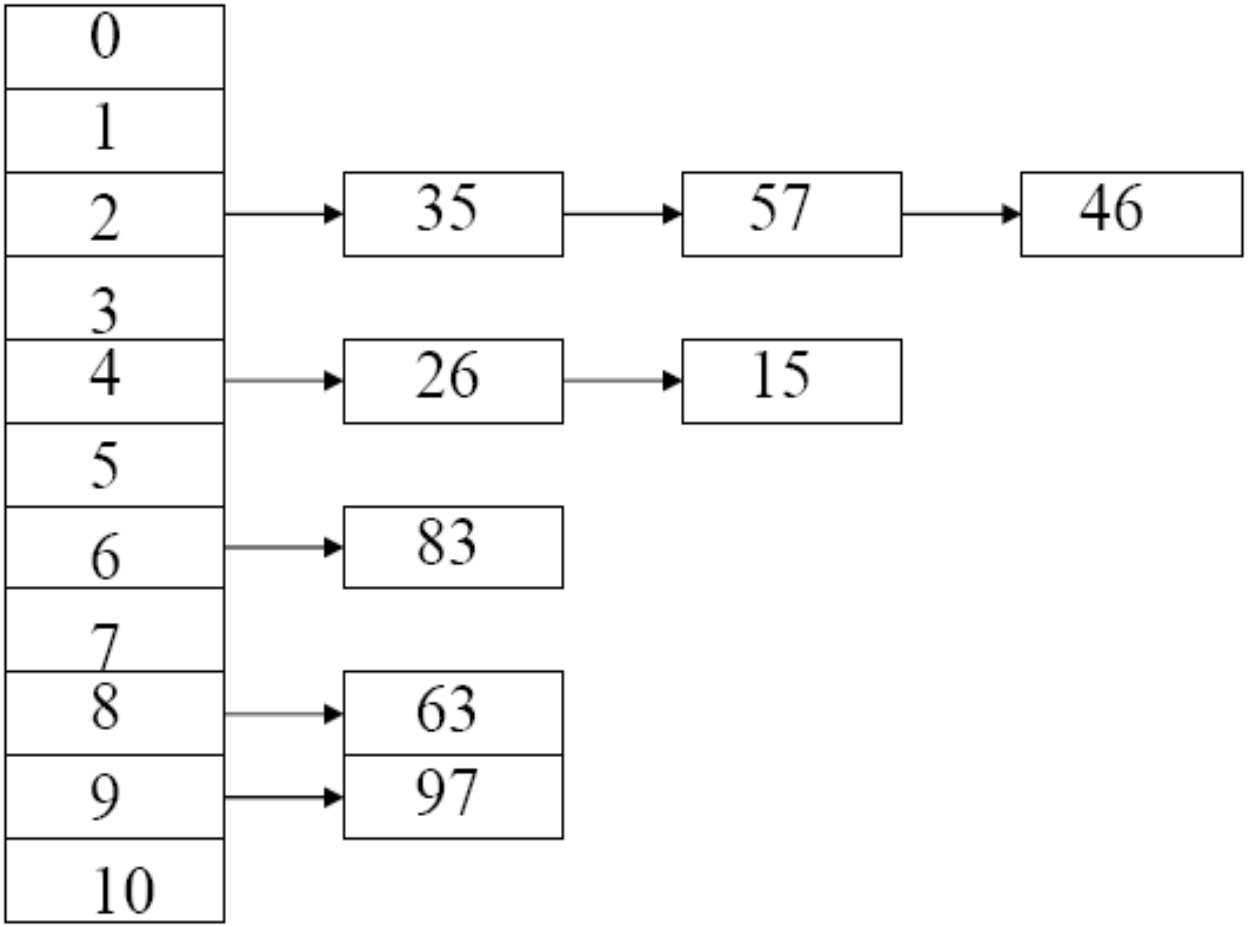
- Data una tabella hash di lunghezza m=11, si supponga di dover inserire (in ordine) le chiavi: 35, 83, 57, 26, 15, 63, 97 ,46, con la funzione di hash h(k) = k mod m. Si illustrino i risultati dell'inserimento usando:
  - separate chaining
  - linear probing
  - quadratic probing ($h(k,i)=(h(k)+i^2)\mathrm{mod}(m)$)
  - double hashing con $h2(K)=1+(k\cdot \mathrm{mod}(m-1))$

Esercizio 4 – Soluzione

- h(35)=35mod11=2
- h(83)=83mod11=6
- h(57)=57mod11=2
- h(26)=26mod11=4
- h(15)=15mod11=4
- h(63)=63mod11=8
- h(97)=97mod11=9
- h(46)=46mod11=2

Esercizio 4 – Soluzione

- Separate Chaining



h(35)=2
h(83)=6
h(57)=2
h(26)=4
h(15)=4
h(63)=8
h(97)=9
h(46)=2

Esercizio 4 – Soluzione

- Linear Probing

0	1	2	3	4	5	6	7	8	9	10
		35	57	26	15	83	46	63	97	

h(57)=2 -> la slot 2 è occupata $h_1(57)=3$

h(15)=4 -> la slot 4 è occupata $h_1(15)=5$

h(46)=2 -> la slot 2 è occupata

$h_1(46)=3$ -> la slot 3 è occupata

$h_2(46)=4$ -> la slot 4 è occupata

$h_3(46)=5$ -> la slot 5 è occupata

$h_4(46)=6$ -> la slot 6 è occupata $h_5(46)=7$

h(35)=2
h(83)=6
h(57)=2
h(26)=4
h(15)=4
h(63)=8
h(97)=9
h(46)=2

Esercizio 4 – Soluzione

- Quadratic probing

0	1	2	3	4	5	6	7	8	9	10
46		35	57	26	15	83		63	97	

h(57)=2 -> la slot 2 è occupata $h_1(57)=3$

h(15)=4 -> la slot 4 è occupata $h_1(15)=5$

h(46)=2 -> la slot 2 è occupata

$h_1(46)=3$ -> la slot 3 è occupata

$h_2(46)=6$ -> la slot 6 è occupata $h_3(46)=0$

h(35)=2
h(83)=6
h(57)=2
h(26)=4
h(15)=4
h(63)=8
h(97)=9
h(46)=2

Esercizio 4 – Soluzione

- Double Hashing

0	1	2	3	4	5	6	7	8	9	10
	46	35		26	15	83		63	97	57

h(57)=2 -> la slot 2 è occupata $h_1(57)=2+1*8=10$

h(15)=4 -> la slot 4 è occupata $h_1(15)=4+1*6=10$

-> la slot 10 è occupata $h_2(15)=4+2*6=5$

h(46)=2 -> la slot 2 è occupata $h_1(46)=2+1*7=9$

-> la slot 9 è occupata $h_2(46)=2+2*7=5$

-> la slot 5 è occupata $h_3(46)=2+3*7=1$

h(35)=2
h(83)=6
h(57)=2
h(26)=4
h(15)=4
h(63)=8
h(97)=9
h(46)=2

Alberi

Esercizio 1

- Due alberi binari A e B si dicono isomorfi se scambiando i sottoalberi sinistri e destri di alcuni nodi di A si può ottenere un albero identico a B.

- Data la seguente classe C++ per implementare gli albero binari, di cui sono specificati solo i metodi rilevanti per l'esercizio,

```
public BinNode {
   public:
        int value;       // The node's value
        BinNode *left;   // Pointer to left child
        BinNode *right; // Pointer to right child
   …
}
```

Esercizio 1 - Soluzione


```
//Input: Two treeNodes A and B
//Output: Returns true if the trees rooted at A and B are
   isomorphic
bool isom(BinNode *A, BinNode *B)
{
    if (A == null && B == null) return true;
    if ((A == null && B !=null) || (A != null && B == null))
      return false;
    if (A->value != B->value ) return false;
    else
      return
      (isom(A->left,B->left) && isom(A->right,B->right)) ||
      (isom(A->left,B->right) && isom(A->right,B->left));
}
```

Esercizio 2

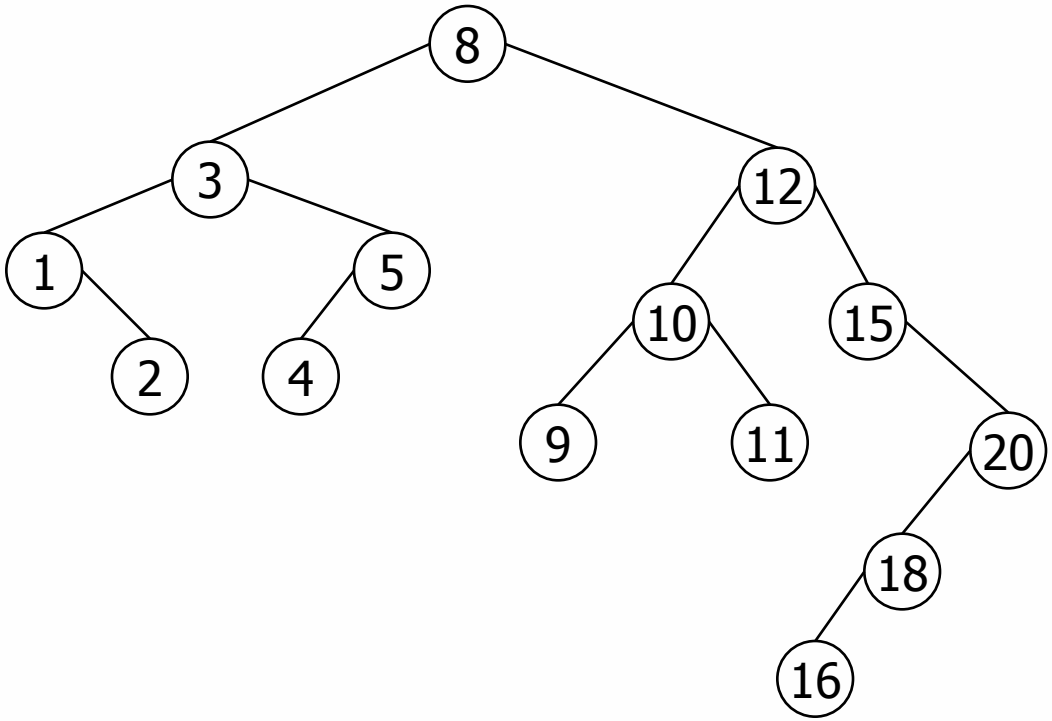
- Costruire un albero di ricerca binario che contenga i seguenti elementi (inserire gli elementi nell'albero nell'ordine seguente):

  8 3 1 5 12 4 10 15 2 9 11 20 18 16

- Come si modifica l'albero se vengono cancellati gli elementi (nell'ordine): 12 e 5?

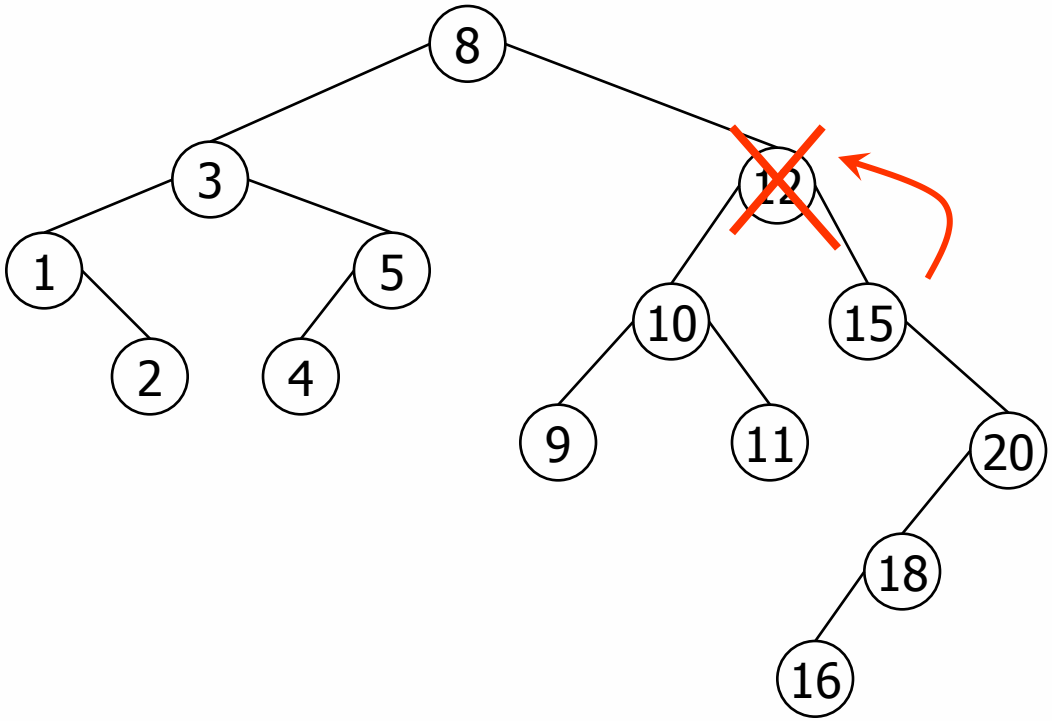
Esercizio 2 - Soluzione

- Inserimento: 8 3 1 5 12 4 10 15 2 9 11 20 18 16



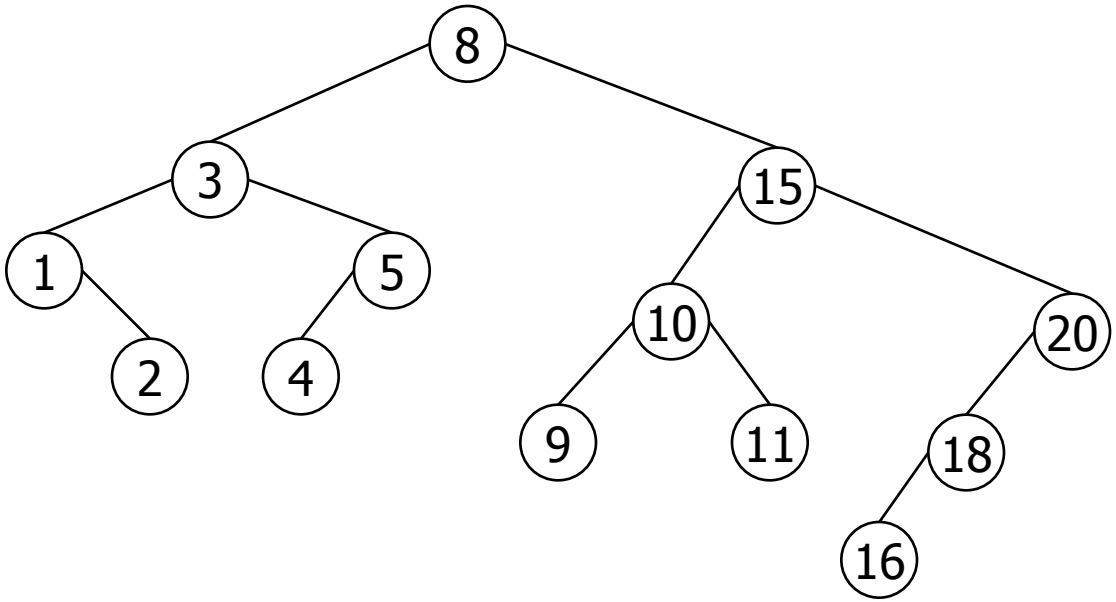
Esercizio 2 - Soluzione

- Inserimento: 8 3 1 5 12 4 10 15 2 9 11 20 18 16
- Rimozione: 12



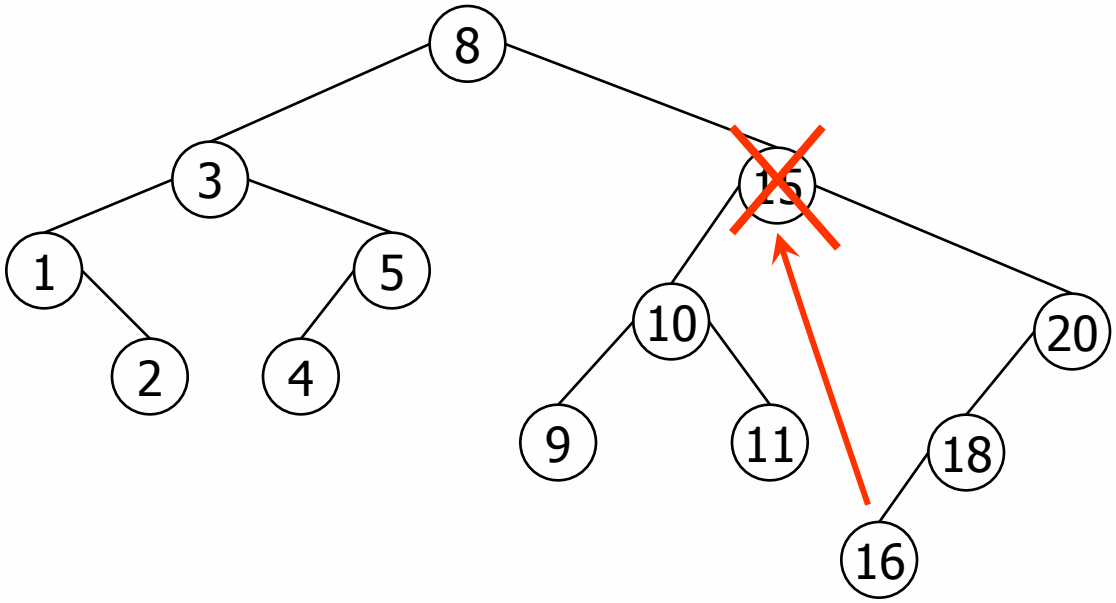
Esercizio 2 - Soluzione

- Inserimento: 8 3 1 5 12 4 10 15 2 9 11 20 18 16
- Rimozione: 12



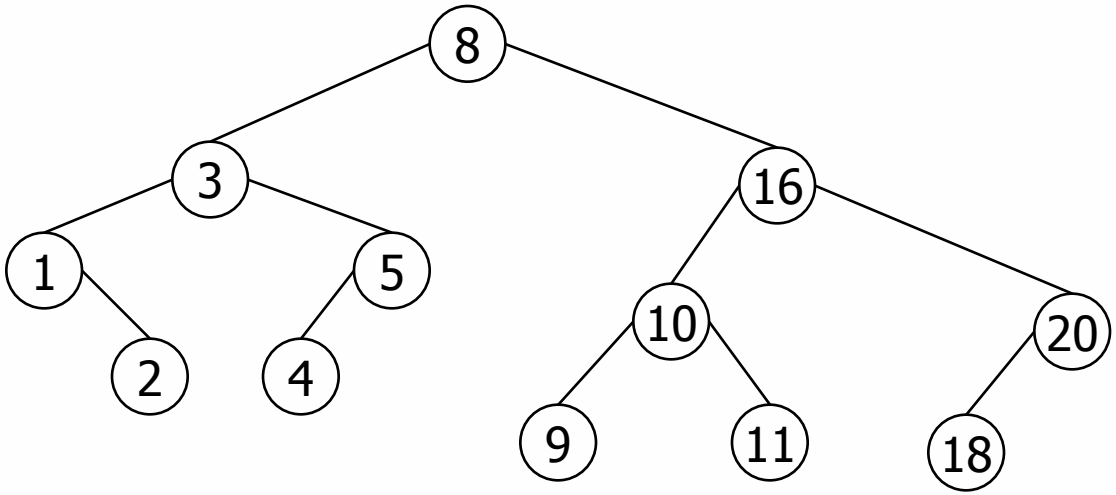
Esercizio 2 - Soluzione

- Inserimento: 8 3 1 5 12 4 10 15 2 9 11 20 18 16
- Rimozione: 12 e 15



Esercizio 2 - Soluzione

- Inserimento: 8 3 1 5 12 4 10 15 2 9 11 20 18 16
- Rimozione: 12 e 15



Grafi

Esercizio 1

- Il quadrato di un grafo orientato G = (V,E) è il grafo G2 = (V,E2) tale che (v,w) $\in$ E2 se e solo se $\exists$u : (v, u) $\in$ E AND (u,w) $\in$ E. In altre parole, se esiste un percorso di due archi fra i nodi u e w.
- Scrivere un algoritmo che, dato un grafo G rappresentato con matrice d'adiacenza, restituisce il grafo G2

Esercizio 1 - Soluzione


```
void quadrato (matrice &A, matrice &A2){
   for (int i = 0; i<N;i++)
        for (int j = 0; j<N;j++){
              bool found = false;
              for (int k = 0; k<N;k++){
                    if (A[i][k] && A[k][j] && i!=k && j!=k){
                          found = true;
                          break;
                    }
              }
              if (found)
                    A2[i][j]=1
              else
                    A2[i][j]=0;
        }
}
```

Esercizio 2

- Data una rappresentazione con matrice di adiacenza, mostrare un algoritmo che opera in tempo $\Theta(V)$ in grado di determinare se un grafo orientato contiene un pozzo universale: ovvero un nodo con grado uscente uguale a zero e grado entrante uguale a $|V|-1$.

Esercizio 2 - Soluzione

- Cerchiamo di scartare velocemente i vertici che non sono pozzi:
  - Dati due vertici i, j, se A[i, j] = 1, allora il vertice i non è un pozzo universale (c'è un arco uscente da i);
  - se A[i, j] = 0, allora j non è un pozzo universale (manca arco entrante in j). Si noti inoltre che puè esistere un solo pozzo universale.
- La soluzione può avere pertanto questa struttura:
  - Partiamo dalla prima riga: i = 1.
  - Cerchiamo il minore indice j tale j > i e A[i, j] = 1.
  - Se tale vertice non esiste, i non ha archi uscenti ed è l'unico candidato per essere un pozzo universale
  - Se invece tale j esiste, ne possiamo dedurre che tutti i vertici h tali che 1 h < j non possono essere pozzi universali, perchè manca un arco da i. Quindi ci spostiamo nella riga i := j, e torniamo al passo 1.
- Si noti che un possibile candidato viene trovato sempre; al limite è dato dall'ultima riga. A quel punto, si verifica che sia effettivamente un pozzo universale, guardando la riga i (deve contenere tutti 0) e la colonna i (deve contenere tutti 1, tranne A[i, i]).
- Il costo dell'algoritmo è pari a 3|V |.

Esercizio 3

- Dimostrare che in ogni grafo diretto aciclico esiste almeno un vertice che non ha archi entranti.

Esercizio 3 – Soluzione

- Supponiamo per assurdo che esista un grafo aciclico in cui tutti i vertici hanno almeno un arco entrante.
- Partendo da un vertice qualunque, potremmo quindi continuare a percorrere archi all'indietro quante volte vogliamo. Osserviamo che, poiché il grafo è aciclico, il vertice raggiunto ad ogni passo deve essere necessariamente diverso da quelli visitati precedentemente. Ne consegue che, prima o poi, dovremo necessariamente incontrare un vertice già visitato, il che contraddirebbe l'ipotesi che il grafo sia aciclico.

Threads

Esercizio 1

- Si utilizzino i thread per modellizzare la movimentazione di un conto bancario.
- Si implementino i seguenti thread:
  - Un thread *timer* che scandisce il tempo a partire da 1/1/2010 (si ipotizzi per semplicità che tutti i messi abbiano 31 giorni). Per ogni giorno il thread aggiorna gli interessi applicando al capitale corrente un tasso pari al 2% annuo. Gli interessi vengono sommati al capitale l'ultimo giorno dell'anno.
  - Un thread *stipendio* che incrementa il capitale di 1650 euro il giorno 27 ogni 2 mesi.
  - Un thread *mutuo* che decrementa il capitale di 800 euro il giorno 10 di ogni mese.
- Si implementino i thread in modo da consentire un accesso esclusivo alla variabile capitale.

Esercizio 1 - Soluzione


```
int anno,giorno,mese;
float interessi,float capitale;
pthread_mutex_t mutex_capitale;
void *timer(void *arg);
void *mutuo(void *arg);
void *stipendio(void *arg);
int main(int argc, char *argv[])
{
    pthread_t t[3];
    capitale=300; interessi=0;
    giorno=1; mese=1; anno=2010;

    printf("In main: creating thread timer\n");
    pthread_create(&t[0], NULL, timer, NULL);
    printf("In main: creating thread mutuo\n");
    pthread_create(&t[1], NULL, mutuo, NULL);
    printf("In main: creating thread mutuo\n");
    pthread_create(&t[2], NULL, stipendio, NULL);
    pthread_exit(NULL);
}
```

Esercizio 1 - Soluzione


```
void *timer(void *arg){
    printf("Hello World! I am timer\n");
    while (true){
        pthread_mutex_lock(&mutex_capitale);
        interessi += capitale * TASSO/365.0;
        giorno++;
        if (giorno>31) { giorno = 1; mese++; }
        if (mese>12) {
                mese=1;anno++;
                capitale+=interessi; interessi=0;
        }
        pthread_mutex_unlock(&mutex_capitale);
        usleep(10000);
   }
    pthread_exit(NULL);
}
```

Esercizio 1 - Soluzione

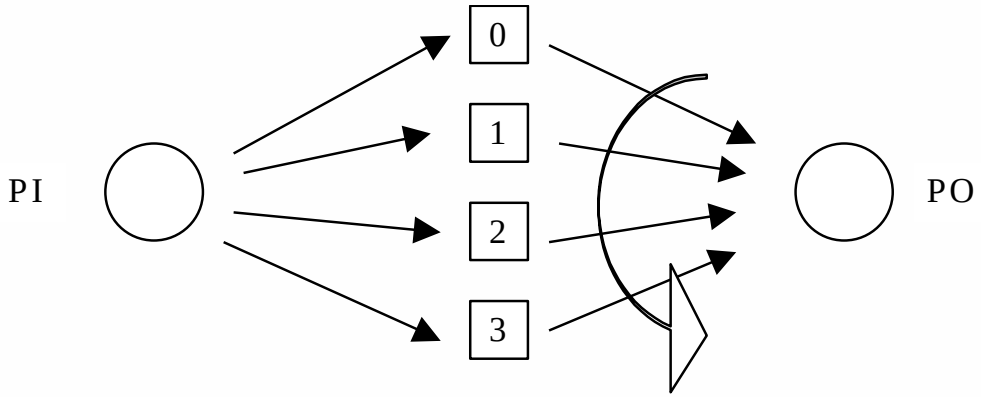

```
void *mutuo(void *arg){
    printf("Hello World! I am mutuo\n");
    while (true){
        bool done;
        while (giorno!=10) done=false;
        if (!done){
                pthread_mutex_lock(&mutex_capitale);
                capitale -=  MUTUO;
                pthread_mutex_unlock(&mutex_capitale);
                done = true;
        }
    }
    pthread_exit(NULL);
}
```

Esercizio 1 - Soluzione


```
void *stipendio(void *arg){
    printf("Hello World! I am mutuo\n");
    while (true){
        bool done;
        while (giorno!=27 || mese%2!=0) done=false;
        if (!done)
        {
                pthread_mutex_lock(&mutex_capitale);
                capitale +=  STIPENDIO;
                pthread_mutex_unlock(&mutex_capitale);
                done = true;
        }
    }
    pthread_exit(NULL);
}
```

Esercizio 2

- Il thread PI esegue ripetutamente le seguenti operazioni:
  - legge da tastiera una coppia di valori <i, ch>, dove i è un numero tra 0 e 3, ch un carattere
  - inserisce il carattere ch nel buffer i (ognuno dei quattro buffer contiene al più un carattere)
- Il thread PO considera a turno in modo circolare i quattro buffer e preleva il carattere in esso contenuto, scrivendo uscita la coppia di valori <i, ch> se ha appena prelevato il carattere ch dal buffer i
- Implementare i thread in C++ in modo che l'accesso a ognuno dei buffer sia in mutua esclusione (PI rimane bloccato se il buffer a cui accede è pieno, PO se è vuoto)



Esercizio 2 - Soluzione


```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

char buffer[SIZE];
bool empty [SIZE];
pthread_mutex_t mutex_buffer[SIZE];

void *pi_main(void *arg);
void *po_main(void *arg);
int main(int argc, char *argv[]){
    pthread_t pi,po;
    for (int i = 0; i<SIZE; i++)
          empty[i] = true;
    printf("In main: creating thread PI\n");
    pthread_create(&pi, NULL, pi_main, NULL);
    pthread_create(&po, NULL, po_main, NULL);
    pthread_exit(NULL);
}
```

Esercizio 2 – Soluzione (2)


```
void *pi_main(void *arg){
   printf("Hello World! I am PI\n");
   while (true) {
         char ch;
         int i;
         scanf("%d",&i);
         scanf("%c",&ch);
         if (i>=0  && i < SIZE)    {
                  while (empty[i]==false);
                  pthread_mutex_lock(&mutex_buffer[i]);
                  printf("In PI: writing  %c in buffer %d\n",ch,i);
                  empty[i] = false;
                  buffer[i] = ch;
                  pthread_mutex_unlock(&mutex_buffer[i]);
         }
    }
   pthread_exit(NULL);
}
```

Esercizio 2 – Soluzione (3)


```
void *po_main(void *arg)
{
   printf("Hello World! I am PO\n");
   while (true)
   {
        for (int i=0; i<SIZE; i++)
        {
          while (empty[i]==true);
          pthread_mutex_lock(&mutex_buffer[i]);
          empty[i] = true;
          printf("In PO: buffer[%d] = %c\n",i,buffer[i]);
          pthread_mutex_unlock(&mutex_buffer[i]);
        }
   }
   pthread_exit(NULL);
}
```

Esercizio 2 – Domande

- Data la seguente sequenza di valori letta da PI, scrivere la sequenza scritta in corrispondenza da PO.
  <1, c> <0, b> <2, m> <0, f> <1, h> <3, n>

R. *<0, b> <1, c> <2, m> <3, n> <0, f> <1, h>*

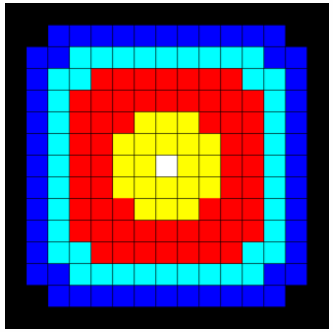
- Descrivere brevemente in quali casi si può verificare una situazione di *deadlock* tra PI e PO. Illustrare con un semplice esempio.

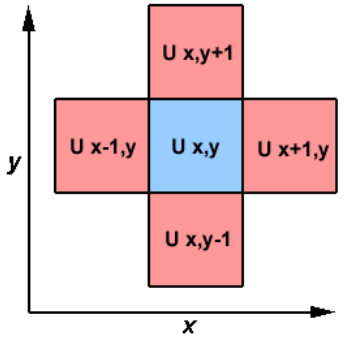
R. *Deadlock: <1,a> <1,b>*

Esercizio 3

- Proporre una soluzione basata su pthreads per parellizzare su una macchina con M processori il calcolo di una semplice equazione di calore su una matrice NxN:

$$U_{x,y} = U_{x,y} + C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy}) + C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})$$



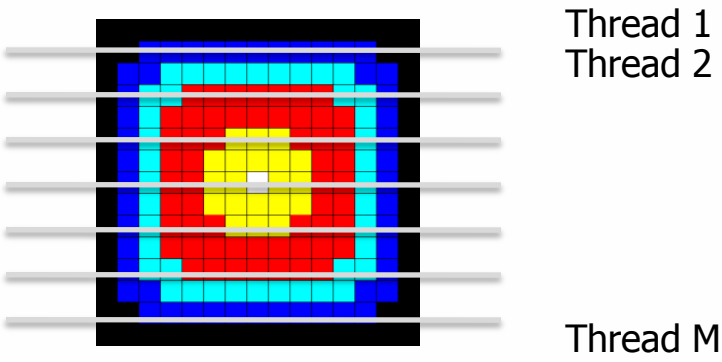


- La soluzione seriale del problema può essere schematizzata come:

```
for (x=1; x < N-1; x++)
   for (y=1; y < N-1; y++)
        u[x][y] = u[x][y] + cx*(u[x+1][y] + u[x+1][y]
        - 2* u[x][y]) + cy*(u[x][y+1] + u[x][y-1]
        - 2* u[x][y]
```

Esercizio 3 – Soluzione

- Si può scomporre la matrice in blocchi ed utilizzare ciascun thread per aggiornare un diverso blocco della matrice



- Ciascun thread può svolgere i propri calcoli in maniera totalmente indipendente.

Esercizio 3 – Soluzione (codice)


```
float U[N][N];
float U2[N][N];
float cx=0.1; float cy=0.2;
void *update_row(void *arg);

int main(int argc, char *argv[]){
    pthread_t jobs[M];
    /* INIT MATRIX U */
    for (int i=0; i<M; i++){
          printf("In main: creating thread %d\n",i);
          pthread_create(&jobs[i], NULL, update_row , (void *) i);}
    for (int i=0; i<M; i++){
          printf("In main: waiting thread %d\n",i);
          void *status;
          pthread_join(jobs[i], &status);}
    /* PRINT RESULT MATRIX U2 */
    pthread_exit(NULL);
}
```

Esercizio 3 – Soluzione (codice)


```
void *update_row(void *arg)
{
    int tid = (long) arg;
    int start=tid*(N-2)/M + 1;
    int end=(tid+1)*(N-2)/M;
    printf("Updating rows from %d to %d\n",start,end);
    for (int x=start; x<=end; x++)
   for (int y=1; y<N-1; y++)
        U2[x][y] = U[x][y] + cx*(U[x+1][y] + U[x-1][y]
        - 2* U[x][y]) + cy*(U[x][y+1] + U[x][y-1]
        - 2* U[x][y]);
    pthread_exit(NULL);
}
```

Esercizio 3bis

- Estendere l'esercizio precedente affinchè vengano svolte K iterazioni di aggiornamento della matrici
- Problemi
  - Sincronizzazione fra i threads tra un'iterazione e quella successiva
  - Gestione efficiente della matrice risultato

Esercizio 3bis - Soluzione


```
float *U,*U2;
float cx,cy;
typedef struct { float *src; float *dest; int tid;} t_arg;
void *update_row(void *arg);
int main(int argc, char *argv[])
{
    pthread_t jobs[M];
    U = (float*) malloc(N*N*sizeof(float));
    U2 = (float*) malloc(N*N*sizeof(float));
    /*INIT MATRIX U*/
    t_arg ta[M];
    for (int k=0; k<STEPS; k++){
          for (int i=0; i<M; i++){
                ta[i].src = (k%2==0)?U:U2;
                ta[i].dest = (k%2==0)?U2:U;
                ta[i].tid = i;
                pthread_create(&jobs[i], NULL, update_row , (void *) &ta[i]);}
          for (int i=0; i<M; i++) pthread_join(jobs[i], &status);
          /*PRINT MATRIX U oppure U2*/
    }
    pthread_exit(NULL);
}
```

Esercizio 3bis - Soluzione


```
void *update_row(void *arg) {
    t_arg *ta = (t_arg *) arg;
    int tid = ta->tid;
    float *uu = ta->dest;
    float *u = ta->src;
    int start=tid*(N-2)/M + 1;
    int end=(tid+1)*(N-2)/M;
    printf("Thread%d: Updating rows from %d to %d\n",tid,start,end);
    for (int x=start; x<=end; x++)
    for (int y=1; y<N-1; y++)
         uu[x*N+y] = u[x*N+y] + cx*(u[(x+1)*N+y] + u[(x-1)*N+y]
         - 2* u[x*N+y]) + cy*(u[x*N+y+1] + u[x*N+y-1]
         - 2* u[x*N+y]);
    pthread_exit(NULL);
}
```