Homework set 2 Lorenzo Esposito M63001417

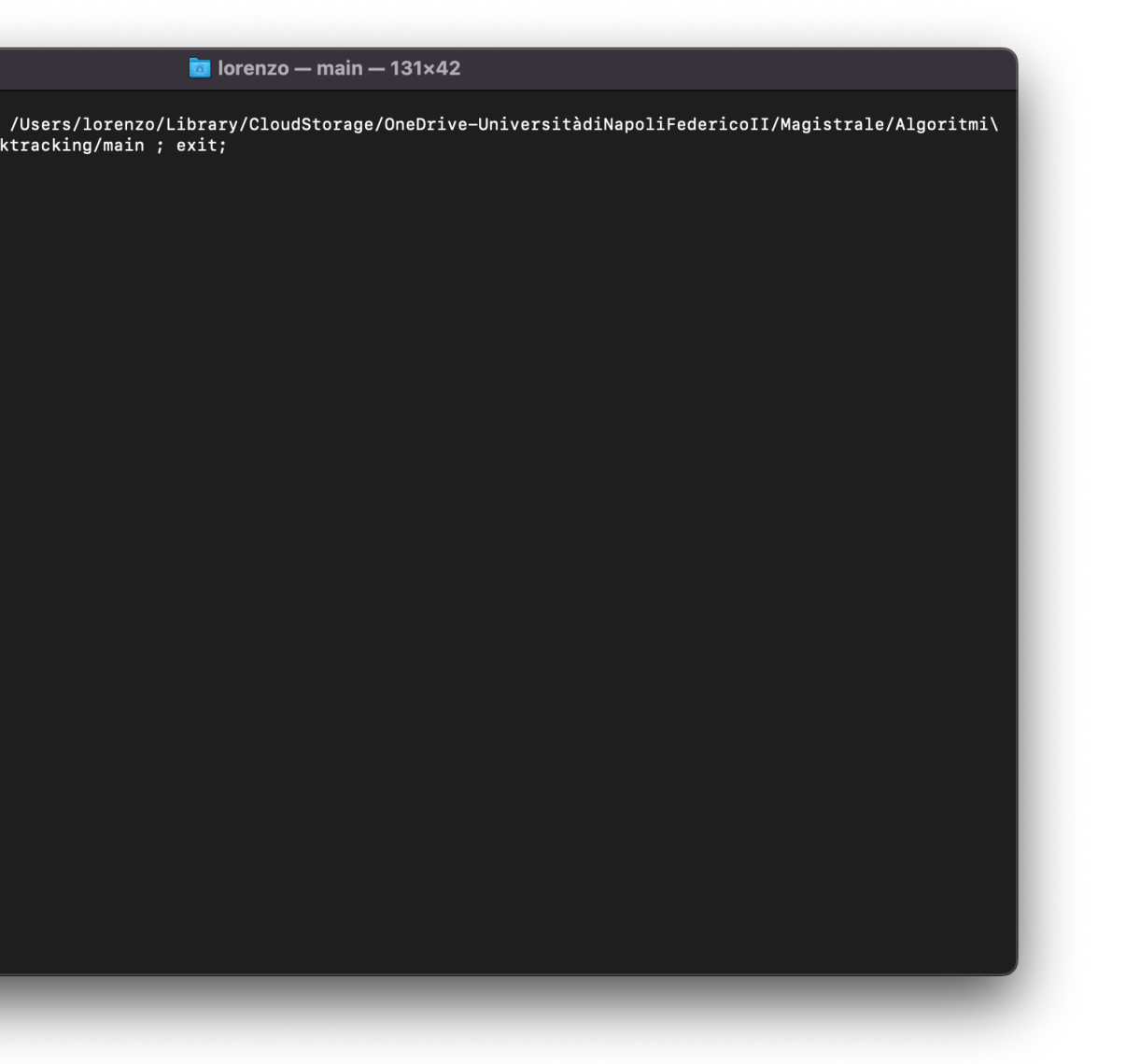
venerdì 25 novembre 2022 14:00

Problema 2.1

Si consideri il seguente problema di minimo percorso vincolato. Sia data una matrice rettangolare di 1 e 0, in cui 1 indica una cella che è possibile percorrere e 0 una cella che non è possibile percorrere. Inoltre, si assuma che non è possibile percorrere neanche le quattro celle adiacenti (sinistra, destra, sopra e sotto) ad una cella che contiene uno 0.
Calcolare la lunghezza del più breve percorso possibile da qualsiasi cella nella prima colonna a qualsiasi cella nell'ultima colonna della matrice. L'obiettivo è evitare le celle contrassegnate con 0 e le loro quattro celle adiacenti (sinistra, destra, sopra e sotto).
Le mosse possibili da una determinata cella sono lo spostamento di una cella adiacente a sinistra, destra, sopra o sotto (non sono consentiti spostamenti in diagonale).

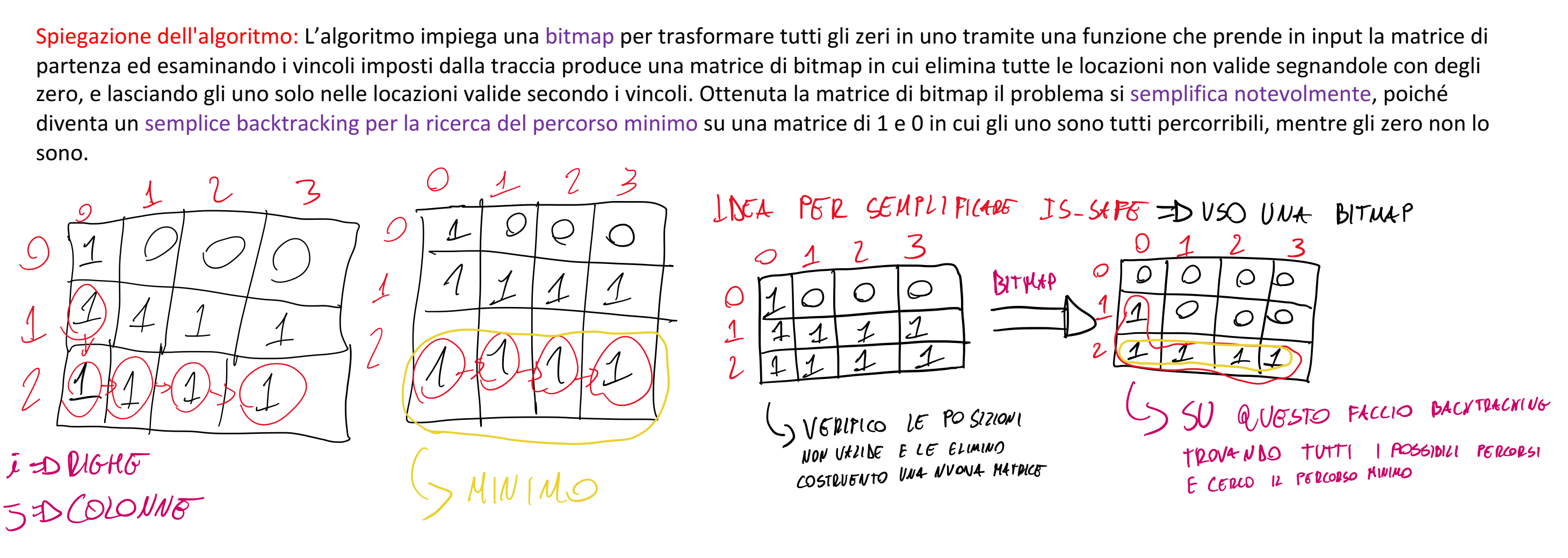
Si alleghi al PDF un file editabile riportante l'implementazione in un linguaggio a scelta, corredato da almeno tre casi di test con il corrispondente output atteso

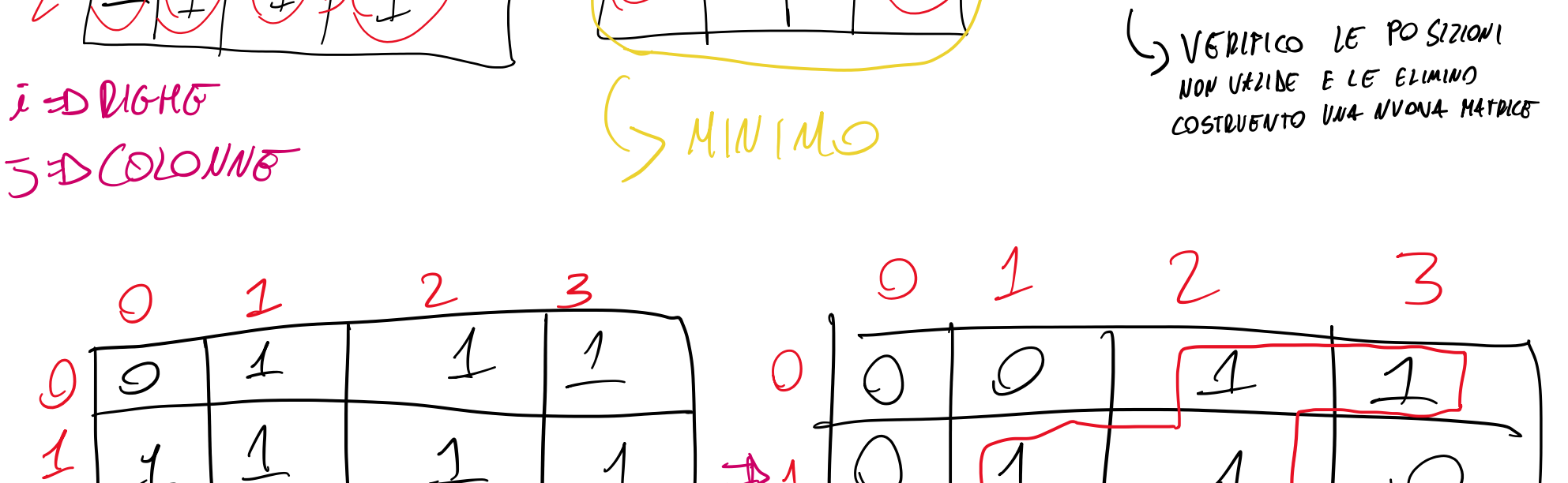
```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;
void print_matrix(vector<vector<bool>> mat)
{
    int i, j;
    for (i = 0; i < mat.size(); i++)
    {
        for (j = 0; j < mat[0].size(); j++)
        {
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
}
void bitmap(vector<vector<bool>> &mat, vector<vector<bool>> &bit,int i, int j)
{
    if(mat[i][j]==false){
        if(i-1>=0){ //SOPRA
            bit[i-1][j]=false;
        }
        if(i+1<=mat.size()-1){ //Sotto
            bit[i+1][j]=false;
        }
        if(j-1>=0){ //sinistra
            bit[i][j-1]=false;
        }
        if(j+1<=mat[0].size()-1){ //destra
            bit[i][j+1]=false;
        }
        bit[i][j]=false;
    }
}
bool is_safe(vector<vector<bool>>mat,int u,int k ){
    if (k > mat[0].size() - 1 || u > mat.size() - 1||u < 0 || k < 0)
        return false;
    if(mat[u][k]==false)
    return false;
    return true;
}

void algorithm(int i, int j, vector<vector<bool>> &mat, int lenght, int &min_lenght)
{
    // condizione di terminazione
    //cout<<"la j è "<<j<<" la size è "<<mat[0].size()-1<<endl;

      //  cout<<"sono nella condizione di terminazione"<<endl<<lenght<<endl<<min_lenght<<endl;
        if ( j==mat[0].size()-1)
        {
            if(lenght<min_lenght){
            min_lenght = lenght;
            //cout<<min_lenght<<endl;

            }
            return;
        }
           //pongo la posizione attuale così non effettua il rollback finchè non termine l'algoritmo
            mat[i][j] = false;
for (int k = 0; k < 4; k++)
    {
        if (k == 0)
        {
            if (is_safe(mat,i,j+1))
            {
                //cout << "(DESTRA)sono in "<<i<<" , "<< j<< endl;
                algorithm(i, j + 1, mat, lenght+1 , min_lenght);
            }
            // destra
        }
        if (k == 1)
        {
            if (is_safe(mat,i+1,j))
            {
                //cout << "(BASSO)sono in "<<i<<" , "<< j<< endl;
                algorithm(i + 1, j, mat, lenght+1, min_lenght);
            }
            // basso
        }
        if (k == 2)
        {
            if (is_safe( mat,i,j-1))
            {
                //cout << "(SINISTRA)sono in "<<i<<" , "<< j<< endl;
                algorithm(i, j - 1, mat, lenght+1 , min_lenght);
            }
            // sinistra
        }
        if (k == 3)
        {
            if (is_safe( mat,i-1,j))
            {
                //cout << "(ALTO)sono in "<<i<<" , "<< j<< endl;
                algorithm(i - 1, j, mat, lenght+1 , min_lenght);
            }
            // alto
        }
    }
    //rimetto true per per
        mat[i][j] = true;
    return;
    }

//CASO IN CUI IL CAMMINO MINIMO é UN SOTTO INSIEME DI UN CAMMINO PIù GRANDE
void testcase1(vector<vector<bool>> &mat, vector<vector<bool>> &bit)
{
    mat[0][0]=true;
    for (int i = 1; i < mat.size(); i++)
        for (int j = 0; j < mat[0].size(); j++)
            mat[i][j] = true;
    cout << "LA MATRICE DI PARTENZA è" << endl;
    print_matrix(mat);
    cout << "LA MATRICE DI BITMAP è" << endl;
    for(int i=0;i<bit.size();i++)
    for(int j=0;j<bit[0].size();j++)
    bitmap(mat,bit,i,j);
    print_matrix(bit);
    int min=INT_MAX;
    int l=1;
    for(int u=0;u<bit.size();u++){
    if(bit[u][0]==true){
            algorithm(u,0,bit,l,min);
    }
    }
 if(min==INT_MAX){
        cout<<"IL PROBLEMA NON HA SOLUZIONE"<<endl;
    }
    else{
        cout<<min<<endl;
    }
}
//CASO SENZA SOLUZIONE
void testcase2(vector<vector<bool>> &mat, vector<vector<bool>> &bit){
    mat[0][0] = true;
    mat[0][1] = true;
    mat[0][2] = true;
    mat[0][3] = true;
    mat[1][0] = true;
    mat[1][1] = false;
    mat[1][2] = true;
    mat[1][3] = true;
    mat[2][0] = true;
    mat[2][1] = true;
    mat[2][2] = true;
    mat[2][3] = false;
     cout << "LA MATRICE DI PARTENZA è" << endl;
    print_matrix(mat);
    cout << "LA MATRICE DI BITMAP è" << endl;
    for(int i=0;i<bit.size();i++)
    for(int j=0;j<bit[0].size();j++)
    bitmap(mat,bit,i,j);
    print_matrix(bit);
    int min=INT_MAX;
    int l=1;
    for(int u=0;u<bit.size();u++){
    if(bit[u][0]==true){
            algorithm(u,0,bit,l,min);
    }
    }
    if(min==INT_MAX){
        cout<<"IL PROBLEMA NON HA SOLUZIONE"<<endl;
    }
    else{
        cout<<min<<endl;
    }
}
//CASO IN CUI IL CAMMINO MINIMO é UN SOTTO INSIEME DI UN CAMMINO PIù GRANDE
//con matrice quadrata
void testcase3(vector<vector<bool>> &mat, vector<vector<bool>> &bit){
  mat[0][1]=false;
     cout << "LA MATRICE DI PARTENZA è" << endl;
    print_matrix(mat);
    cout << "LA MATRICE DI BITMAP è" << endl;
    for(int i=0;i<bit.size();i++)
    for(int j=0;j<bit[0].size();j++)
    bitmap(mat,bit,i,j);
    print_matrix(bit);
    int min=INT_MAX;
    int l=1;
    for(int u=0;u<bit.size();u++){
    if(bit[u][0]==true){
            algorithm(u,0,bit,l,min);
    }
    }
 if(min==INT_MAX){
        cout<<"IL PROBLEMA NON HA SOLUZIONE"<<endl;
    }
    else{
        cout<<min<<endl;
    }
}
void testcase4(vector<vector<bool>> &mat, vector<vector<bool>> &bit){
  mat[0][0]=false;
  mat[2][3]=false;
     cout << "LA MATRICE DI PARTENZA è" << endl;
    print_matrix(mat);
    cout << "LA MATRICE DI BITMAP è" << endl;
    for(int i=0;i<bit.size();i++)
    for(int j=0;j<bit[0].size();j++)
    bitmap(mat,bit,i,j);
    print_matrix(bit);
    int min=INT_MAX;
    int l=1;
    for(int u=0;u<bit.size();u++){
    if(bit[u][0]==true){
            algorithm(u,0,bit,l,min);
    }
    }
 if(min==INT_MAX){
        cout<<"IL PROBLEMA NON HA SOLUZIONE"<<endl;
    }
    else{
        cout<<min<<endl;
    }
}
int main()
{
    vector<vector<bool>> mat(3, vector<bool>(4));
    vector<vector<bool>> bit(3, vector<bool>(4,true));
    //primo caso di test
    testcase1(mat, bit);
    //secondo caso di test
    testcase2(mat,bit);
    //cambio i vettori per usare matrici più grandi
    vector<vector<bool>>mat2(4,vector<bool>(4,true));
    vector<vector<bool>> bit2(4, vector<bool>(4,true));
    //terzo caso di test
    testcase3(mat2,bit2);
    //cambio i vettori
    vector<vector<bool>>mat3(3,vector<bool>(4,true));
    vector<vector<bool>> bit3(3, vector<bool>(4,true));
    //quarto caso di test
    testcase4(mat3,bit3);
    return 0;
}
```


lorenzo — main — 131×42

```
tracking/main ; exit;
lorenzo@MacBook-Air-di-Lorenzo ~ % /Users/lorenzo/Library/CloudStorage/OneDrive-UniversitàdiNapoliFedericoII/Magistrale/Algoritmi\
e\ strutture\ dati/homework\ 2/backtracking/main ; exit;
LA MATRICE DI PARTENZA è
1 0 0 0
1 1 1 1
1 1 1 1
LA MATRICE DI BITMAP è
0 0 0 0
1 0 0 0
1 1 1 1
4
LA MATRICE DI PARTENZA è
1 1 1 1
1 0 1 1
1 1 1 0
LA MATRICE DI BITMAP è
0 0 0 0
0 0 0 0
1 0 0 0
IL PROBLEMA NON HA SOLUZIONE
LA MATRICE DI PARTENZA è
1 0 1 1
1 1 1 1
1 1 1 1
1 1 1 1
LA MATRICE DI BITMAP è
0 0 0 1
1 0 1 1
1 1 1 1
1 1 1 1
4
LA MATRICE DI PARTENZA è
0 1 1 1
1 1 1 1
1 1 1 0
LA MATRICE DI BITMAP è
0 0 1 1
0 1 1 0
1 1 0 0
6
```

Spiegazione dell'algoritmo: L'algoritmo impiega una bitmap per trasformare tutti gli zeri in uno tramite una funzione che prende in input la matrice di partenza ed esaminando i vincoli imposti dalla traccia produce una matrice di bitmap in cui elimina tutte le locazioni non valide segnandole con degli zero, e lasciando gli uno solo nelle locazioni valide secondo i vincoli. Ottenuta la matrice di bitmap il problema si semplifica notevolmente, poiché diventa un semplice backtracking per la ricerca del percorso minimo su una matrice di 1 e 0 in cui gli uno sono tutti percorribili, mentre gli zero non lo sono.





Analisi della complessità: La tecnica del backtracking è nota per essere un approccio tipicamente brute force, quindi il dispendio computazionale per forza di cose non potrà essere inferiore ad una complessità esponenziale. In questo caso la complessità è O(m*4^n).

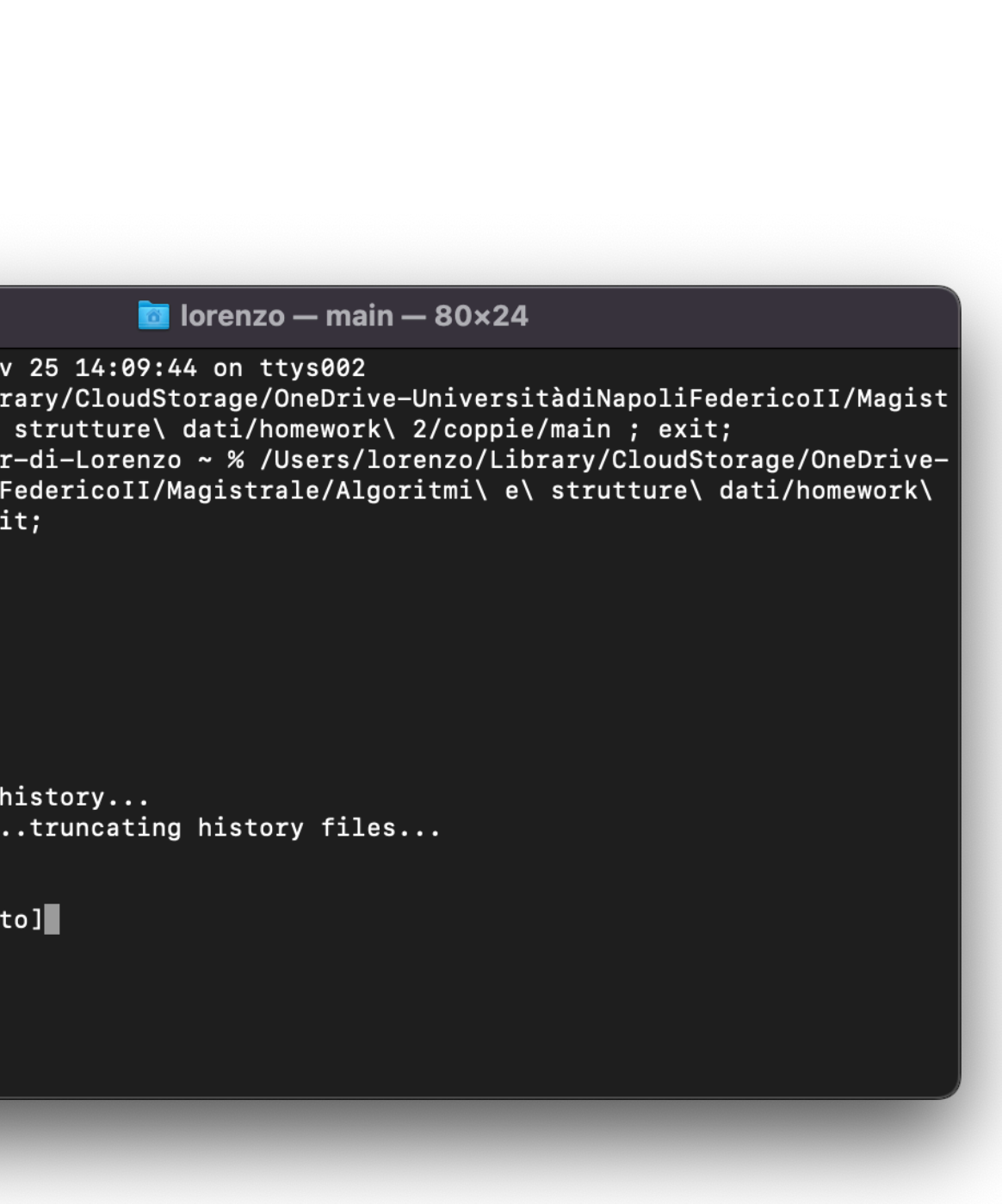
Problema 2.2

Sia dato un insieme di n elementi, e si supponga che ogni elemento possa essere accoppiato con qualche altro elemento oppure può essere non accoppiato ("single"). Ogni elemento può essere accoppiato solo una volta. Si implementi un algoritmo per scoprire il numero totale di modi in cui gli n elementi possono restare single o possono essere accoppiati. Si utilizzi la programmazione dinamica per risolvere il problema.

Ad esempio:
Input : n = 3
Output : 4

Spiegazione:
{1}, {2}, {3} : tutti gli elementi sono single
{1}, {2, 3} : 2 e 3 sono accoppiati ma 1 è single.
{1, 2}, {3} : 1 e 2 sono accoppiati, ma 3 è single.
{1, 3}, {2} : 1 e 3 sono accoppiati, ma 2 è single.
Le coppie {a, b} e {b, a} sono considerate come un'unica coppia.

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
//APPROCCIO BOTTOM-UP
int algorithm(int num){
    //struttura dati
    int DP[num+1];
    for(int i=0;i<num+1;i++)
    if(i<=2){
        //caso base
        DP[i]=i;
    }
    else
    DP[i]=DP[i-1]+(i-1)*DP[i-2];
    return DP[num];
}
void testcase1(){
   cout<<algorithm(3)<<endl;
}
void testcase2(){
   cout<<algorithm(4)<<endl;
}
void testcase3(){
   cout<<algorithm(1)<<endl;
}
int main(){
cout<<"AVVIO TESTCASE 1"<<endl;
testcase1();
//output previsto 4
cout<<"AVVIO TESTCASE 2"<<endl;
testcase2();
//output previsto 10
cout<<"AVVIO TESTCASE 3"<<endl;
//output previsto 1
testcase3();
//caso base

return 0;
}
```


lorenzo — main — 80×24

```
Last login: Fri Nov 25 14:09:44 on ttys002
/Users/lorenzo/Library/CloudStorage/OneDrive-UniversitàdiNapoliFedericoII/Magist
rale/Algoritmi\ e\ strutture\ dati/homework\ 2/coppie/main ; exit;
lorenzo@MacBook-Air-di-Lorenzo ~ % /Users/lorenzo/Library/CloudStorage/OneDrive-
UniversitàdiNapoliFedericoII/Magistrale/Algoritmi\ e\ strutture\ dati/homework\
2/coppie/main ; exit;
AVVIO TESTCASE 1
4
AVVIO TESTCASE 2
10
AVVIO TESTCASE 3
1

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Processo completato]
```

Spiegazione dell'algoritmo: L' approccio impiegato per la risoluzione dell'algoritmo è un approccio bottom-up, ovvero ho risolto prima i sottoproblemi necessari per risolvere i sottoproblemi superiori mediante la seguente formula f(n)=f(n-1)+(n-1)*f(n-2) e salvo ogni risoluzione di un sottoproblema nella struttura dati DP e applico ciclicamente la formula fino all'n-esimo problema del quale devo ottenere la soluzione. In questo caso non ho situazioni di deadlock ed è possibili usare l'approccio bottom-up, altrimenti sarei stato obbligato ad un approccio top-down (ricorsione+memoization).

GUESSING: 2 POSSIBILI SCELTE "SINGLE" OPPURE "PAIR"

↳ 2^n POSSIBILI COMBINAZIONI (STRADA BRUTE FORCE)

NUMERO TOTALE DI MODI

SOTTOSTRUTTURA OTTIMA ⇒ $f(n) = \underbrace{f(n-1)}_{\text{SINGLE}} + \underbrace{(n-1)f(n-2)}_{\text{PAIR}}$

CASO BASE ⇒ $n <= 2$ ⇒ IN QUESTO CASO $f(n)=n$

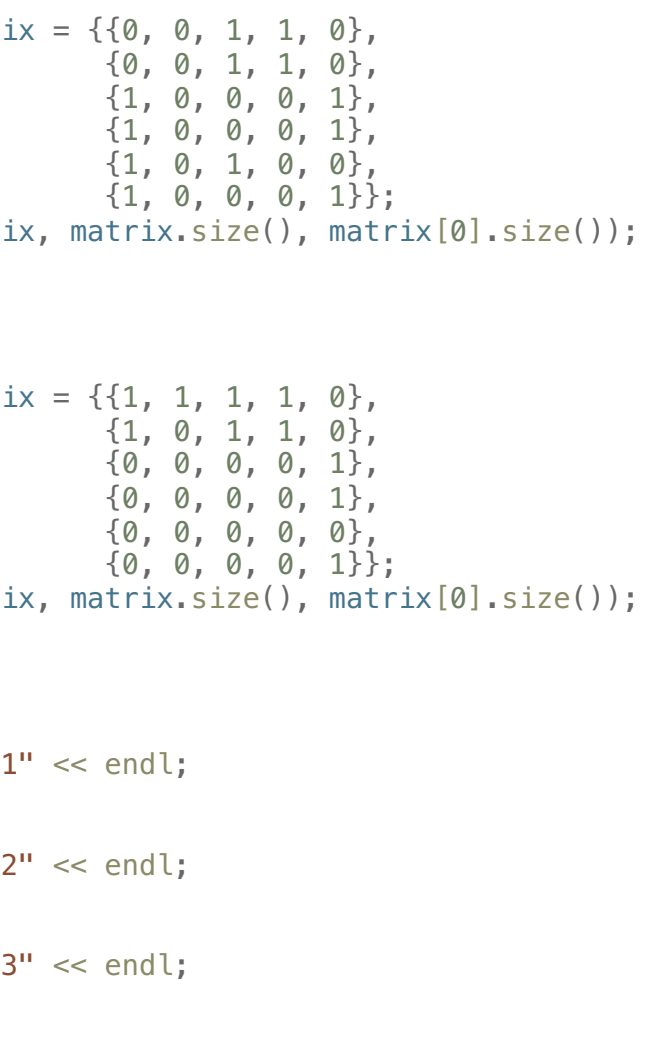
Analisi della complessità: Tipicamente l'approccio con programmazione dinamica rende polinomiali problemi che solitamente hanno complessità esponenziale o fattoriale, in questo caso la complessità computazionale è O(n) poiché abbiamo un singolo ciclo for .

Problema 2.3

Si consideri una matrice di 0 ed 1, in cui "1" indica "posizione occupata" e "0" indica "posizione libera". Si scriva un algoritmo per determinare la sottomatrice massima libera (ossia che contiene tutti 0). L'algoritmo deve riportare il numero di 0 di tale sottomatrice.

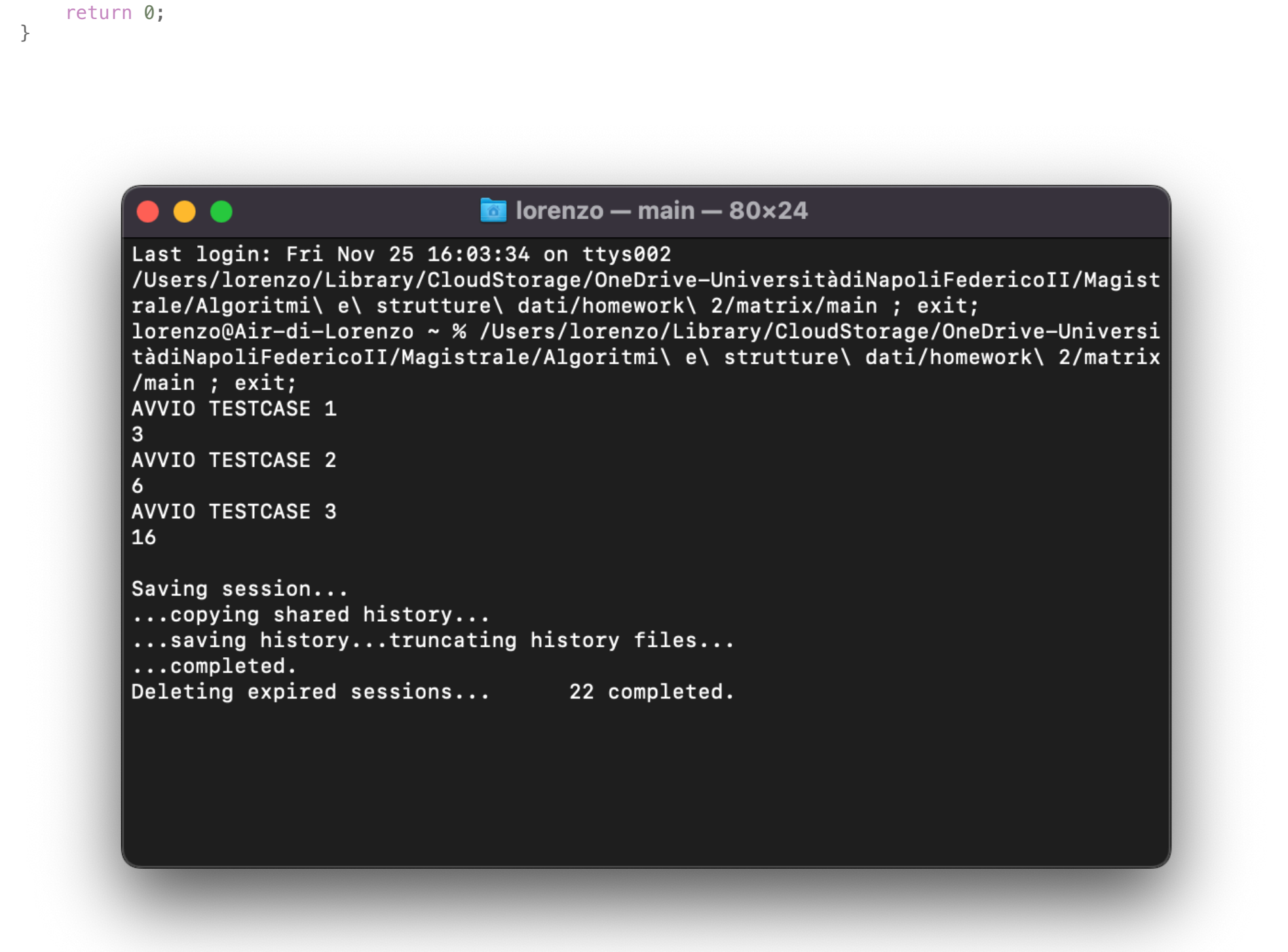
Si alleghi al PDF un file editabile riportante l'implementazione in un linguaggio a scelta, corredato da almeno tre casi di test con il corrispondente output atteso

```cpp
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
int Solution(vector<int> &histo)
{
    stack<int> st;
    // uso uno stack di appoggio per memorizzare l'area dei rettagoli più grandi
    int maxA = 0;
    int n = histo.size();
    for (int i = 0; i <= n; i++)
    {
        while (!st.empty() && (i == n || histo[st.top()] >= histo[i]))
        {
            int height = histo[st.top()];
            st.pop();
            int width;
            if (st.empty())
                width = i;
            else
                width = i - st.top() - 1;
            // il massimo è il massimo tra il massimo attuale e l'area del rettagolo
            maxA = max(maxA, width * height);
        }
        st.push(i);
    }
    return maxA;
}
int algorithm(vector<vector<int>> &mat, int m, int n)
{
    int maxArea = 0;
    vector<int> height(n, 0);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (mat[i][j] == 0)
                // se trovo uno zero aumento l'altezza
                height[j]++;
            else
                // se non trovo uno zero l'altezza viene resettata a zero
                height[j] = 0;
        }
        // calcolo l'area ottenendo il massimo tra il massimo attuale e l'area appena calcolata
        int area = Solution(height);
        maxArea = max(area, maxArea);
    }
    return maxArea;
    // ritorno l'area della sottomatrice più grande contenente tutti zeri
}
void testcase1()
{
    vector<vector<int>> matrix = {{0, 1, 0, 1},
                                  {0, 1, 0, 0},
                                  {0, 1, 1, 1},
                                  {1, 0, 1, 1},
                                  {0, 1, 1, 1}};
    int max = algorithm(matrix, matrix.size(), matrix[0].size());
    cout << max << endl;
}
void testcase2()
{
    vector<vector<int>> matrix = {{0, 0, 1, 1, 0},
                                  {0, 0, 1, 1, 0},
                                  {1, 0, 0, 0, 1},
                                  {1, 0, 0, 0, 1},
                                  {1, 0, 1, 0, 0},
                                  {1, 0, 0, 0, 1}};
    int max = algorithm(matrix, matrix.size(), matrix[0].size());
    cout << max << endl;
}
void testcase3()
{
    vector<vector<int>> matrix = {{1, 1, 1, 1, 0},
                                  {1, 0, 1, 1, 0},
                                  {0, 0, 0, 0, 1},
                                  {0, 0, 0, 0, 1},
                                  {0, 0, 0, 0, 0},
                                  {0, 0, 0, 0, 1}};
    int max = algorithm(matrix, matrix.size(), matrix[0].size());
    cout << max << endl;
}
int main()
{
    cout << "AVVIO TESTCASE 1" << endl;
    testcase1();
    // output atteso 3
    cout << "AVVIO TESTCASE 2" << endl;
    testcase2();
    // output atteso 6
    cout << "AVVIO TESTCASE 3" << endl;
    testcase3();
    // output atteso 16
    return 0;
}
```


lorenzo — main — 80×24

```
Last login: Fri Nov 25 16:03:34 on ttys002
/Users/lorenzo/Library/CloudStorage/OneDrive-UniversitàdiNapoliFedericoII/Magist
rale/Algoritmi\ e\ strutture\ dati/homework\ 2/matrix/main ; exit;
lorenzo@Air-di-Lorenzo ~ % /Users/lorenzo/Library/CloudStorage/OneDrive-Universi
tàdiNapoliFedericoII/Magistrale/Algoritmi\ e\ strutture\ dati/homework\ 2/matrix
/main ; exit;
AVVIO TESTCASE 1
3
AVVIO TESTCASE 2
6
AVVIO TESTCASE 3
16

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...      22 completed.
```

Spiegazione dell'algoritmo: L'algoritmo sopra presentato è basato su un' intuizione, l'algoritmo ragiona partendo dal fatto di conoscere la dimensione della matrice di partenza, ciò ci permette di scorrere la matrice con un doppio for grazie al quale incrementiamo le dimensioni del nostro rettangolo di zeri man mano che ne troviamo di nuovi, quando si trova un elemento diverso da zero si resetta l'altezza del nostro rettangolo e ripartiamo alla ricerca di nuove sottomatrici di zeri, finché non troviamo quella massima mediante il calcolo dell'area delle sottomatrici di zeri trovate. Per tenere traccia dei massimi relativi durante la ricerca impieghiamo un stack di appoggio. Una volta trovato il massimo assoluto lo stampiamo.



NOTA: SI SUPPONE UNA SOTTOMATRICE ANCHE UN VETTORE OPPURE UN SINGOLO ELEMENTO

Analisi della complessità: Per quanto riguarda l'analisi della complessità in questo problema questa corrisponde a O(m*n^2) che nel caso in cui m=n quindi nel caso della matrice quadrata diventa O(n^3).