HOMEWORK 1

Exercise 1.1

For this problem we start by defining the optimal solution (even if we don’t know it yet).

In order to define whether a word is palindrome or not, we define OPT(i,j).

Hence, we adopt a Dynamic Programming approach building a 2-dimensional array which is filled at each iteration with the memoization technique.

**Longest Palindrome Substring(W,n)**

P is another 2-dimensional array in which we store the substrings collected

N is the length of the word

**Initialize** M[1..n,1..n] and P[1..n,1..n]

**Palindrome**(1,n,P)

**Return** P[1][n]

**Palindrome(i,j,P)**

i,j are 2 indexes for the matrix

**If** M[i][j] **is empty** **then**

**If** i>j **then** **return** 0 //if this situation happens we “let win the other element into //the max function” because the substring is over

**If** i=j **then**

M[i][j] = 1

P[i][j] = w[i] to w[j]

**Else If** w[i] == w[j] **then**

M[i][j] = j-i+1

P[i][j] = w[i] to w[j]

**Else**

M[i][j] = max{ Palindrome(i,j-1,P), Palindrome(i+1,j,P)}

P[i][j] = w[i] to w[j]

**Return** M[i][j]

Giving as input i = 1 and j = n you will get the longest palindrome substring in at P[1][n].

We have to fill a nxn table with operations that cost O(1) each. Thus, the total computation time is O(n2)

1.2

Longest Palindrome Substring()

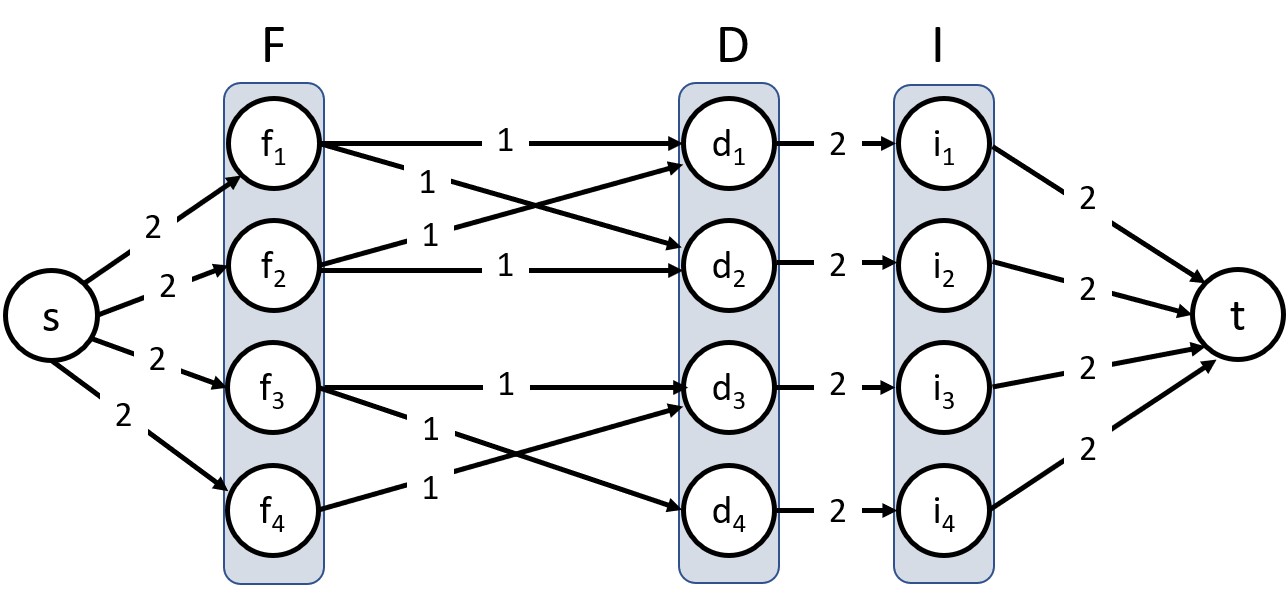
Exercise 2

In order to model the problem, we developed a graph based on the bipartite matching problem, which can be solved as a maximum flow problem. Therefore we decided to design the following set of nodes F, D, I U {s,t} = V. F represents all the founders, I represents all the investors, D is a particular subset of nodes which helped us to model the following constraints:

・Let ik be an investor, he can have at most fi and another fj seated close to him;

・Let fk be a founder, he can have at most ii and another ij seated close to him.

To show an example, given the list P ⊆ I×F of good pair (i1,f1), (i1,f2), (i2,f1), (i2,f2), (i3,f3), (i3,f4), (i4,f3), this is the resulting flow network N.



The numbers on each edge represents the maximum capacity, which helped us to model the fact of having at least three people seated at one table respecting the above constraints. Exploring this network with a maximum flow algorithm will solve the problem of respecting as much good pairings as possible in finding a seating arrangement.

PROOF OF CORRECTNESS **(TODO)**

Def: Given an undirected graph *G = (V, E)* a subset of edges *M* ⊆ *E* is a matching if each node of *F* and *D* appears in at most two edges in *M*.

We want to prove that the max cardinality of a matching in *G* is equal to a max flow in *G’* (the directed graph designed as shown above)

Proof (⇐)

・Given a max matching *M* of cardinality *k*.

・Consider flow *f* that sends 1 unit along each of *k* paths (because 1 is the bottleneck).

・*f* is a flow, and has value *k*.

Proof (⇒)

・Let *f* be the max flow in *G’* of value *k*

・From the *Integrality Theorem* we know that *k* is integral and ∀ *e* ∈ *M* we can either have *f*(*e*) = 0 or 1

・Consider *M* as the set of edges from *F* to *D*, with *f*(*e*) = 1

1. Each node in *F* and *D* participates in at most 2 edges of *M*
2. |*M*| = *k*: consider the cut (*s*∪*F*, *D*∪*I*∪*t*)