# The algorithm

Iterative-Compute-Opt

M[0] = 0;

S[0] = empty; # array that holds the current optimal solution (pairs made so far) at each step

for i = 1, 2, …, : # each cell of M represents a potential pair, in increasing order by the first student of each pair’s index, then by the second student’s index. E.g. it would look like

[(1,2), (1,3), (1,4) … (5,6), (5,7) …].

set (j,k) = the pair that i represents # e.g. i=8 represents the pair (2,5) when n=6

if (vi + p(j,k,n)) > M[i-1]:

M[i] = vi + p(j,k,n)

S[i] = S[p(j,k,n)].add(i) # for solution at step i, add student i onto solution at step p(i)

else:

M[i] = M[i-1]

S[i] = S[i-1] # for solution at step i, keep the previous step’s solution

endif

endfor

return S[] as the optimal set of all pairs

**p(j,k,n)** is a function that, given there are n total students, returns the last pair before a pair of students (j,k) where both students are different from j and k. For example, p(j=3, k=5, n=6) is (2,6); p(j=5, k=6, n=6) is (3,4). If that doesn’t exist (the input is (1,x), for example, so all earlier pairs also have student 1), return 0.

**vi**is 1 if the pair (j,k) that i represents overlap by at least t, 0 otherwise

# Proof of Correctness

Observe that for an optimal solution O, pair (n-1, n) either belongs or doesn’t belong to O. If (n-1, n) ∈ O, then O *must* include an optimal solution to the problem consisting of potential pairs {(1,2) … (n-3, n-2)}. On the other hand, if n ∉ O, then O simply equals the optimal solution to the problem consisting of potential pairs {(1,2) … (n-2, n)}. We summarize this in a formula that essentially says ∀ potential pairs (j,k) (compacted into one number i so the arrays can access things), either i ∈ Oi, in which case Most\_Pairs = vi + Most\_Pairs(p(i)), or i ∉ Oi, in which case Most\_Pairs = Most\_Pairs(i-1). So

Most\_Pairs(i) = max(vi + Most\_Pairs(p(i)), Most\_Pairs(i-1))

We will now prove by **strong induction**  that the algorithm above returns the optimal answer.

## Base Case

We want to prove M[1] is the optimal/maximum pairs if there were only one student, and S[1] is empty. By definition,

M[1] = max(v1 + M[p(1)], M[1-1])

# the pseudocode essentially does a max function when using the comparison operator

= max(0 + M[0], M[0])

= max(0, 0)

= 0

S[0] = empty

S[1] = S[0]

(because M[1] is not strictly greater than M[0])

S[1] = **empty**

Which is correct because there can’t be any pairs if there’s only one student.

## Inductive Case

We need to prove M[i] = Most\_Pairs[i].

Since we’re using strong induction, we can assume that ∀ i < j, M[i] is the maximum number of pairs there can be if there were only the first i possible pairs, and that S[i] holds said pairs. Thus,

M[j] = max(vj + M[p(j)], M[j-1])

= max(vj + Most\_Pairs[p(j)], Most\_Pairs[j-1])

Which was our definition given above (before the base case is proved).

S keeps track of all the pairs that the running optimal solution consists of. S is constructed such that whichever side “wins” the max function, then that side’s set of pairs involved becomes the current running optimal solution.

# Runtime Analysis

Iterating through M is O(n2) because the formula for is . Calculating p(i) is O(1). The comparison is O(1). Setting elements of S and M is O(1). In all, it is O(n2).