Data Structures

linked or wraparound array

FreeM <- M — FreeM is a queue (m in queue means m is free, not in queue means not free)
FreeW <- W — FreeW[w] ← 1 for all w
Invitations <- {} — Next[m] ← 1 for all m (m's current pos of their pref list) #invitations from m so far +1
Tentative <- {} — MatchM[m] ← NIL for all m; MatchW[w] ← NIL for all w — $\Theta(n)$

While there is an m in FreeM s.t. there is a w s.t. (m,w) not in Invitations — check if queue empty $O(1)$

finding m s.t. m ∈ FreeM $O(n^2)$ if FreeM is a hash set :(
$O(n)$ if FreeM is an array of 0/1 :(
but don't need set ops – queue is enough!

choose such an m — remove head of queue $O(1)$

let w be m's highest ranked s.t. (m,w) not in Invitations — w ← PrefM[m][Next[m]]; Next[m] ← Next[m]+1 — $O(1)$

add (m,w) to Invitations — $O(1)$ (linked list)

if w in FreeW then — if FreeW[w] = 1 — $O(1)$

remove w from FreeW — FreeW[w] ← 0

remove m from FreeM

add (m,w) to Tentative ← MatchM[m] ← w; MatchW[w] ← m

else

find m' s.t. (m', w) in Tentative — m' ← MatchW[w]

if w prefers m to m' — still need to do this in $\Theta(1)$ time

remove m from FreeM — already removed — $O(1)$

add m' to FreeM — add to back $O(1)$

remove (m', w) from Tentative — MatchM[m'] ← NIL

add(m, w) to Tentative

return Tentative — else add m to back $O(1)$

Invitations

find next invitation to make – $O(n^2)$ if search list of Invitations :(

(3, 1), (4, 5), (2, 1)
(3, 6), (1, 1), (1, 6)

	1	2	3	4	5	6
1	1	0	0	0	0	1
2	–	.	.			
3	1	.	.	.		

1: 1 6 4 3 2 5

need to know where machinists are on their pref lists – array of current pos in list

size of Tentative = # of iterations of main loop

Tentative ⊆ M×W, which is of size n^2

So main loop terminates after at most n^2 iterations.

Work before loop is $\Theta(n^2)$ [including initializing data structure for preferences]

Work in loop is $\Theta(1)$

So total time is $\Theta(n^2)$

Invariants

Loop Invariant: Something true at beginning of every iteration of loop

right before condition is tested

Loop Invariant Thm:

statement about variables and # iterations of loop

For predicate P, if

a) true before 1st iteration

b) if true before one iteration and guard then true at beginning of next iteration

condition on loop

then P is loop invariant

essentially induction on # iterations of loop

Also want

loop must terminate

the fact invariant true at end of loop means algorithm did what it was supposed to

```
SelectionSort(A)
  for i = 0 to n-2
    find min among A[i],...,A[n-1]
    swap min with A[i]
```

$A[i] = i + 3?)$

3 2 1 5 4

1 | 2 3 5 4

1 2 | 3 5 4

1 2 3 | 5 4

1 2 3 4 | 5

INV: $A[0] \le A[1] \le \dots \le A[i-1]$ (1st i elements are in order)

and $A[i-1] \le A[i], \dots, A[n-1]$

and A has original values but in possibly a different order

loop terminates when $i = n-1$

when $i = n-1$, inv says

$A[0] \le \dots \le A[n-2]$ are sorted and $A[n-1] \ge A[n-2]$

so $A[0] \le A[1] \le \dots \le A[n-2] \le A[n-1]$ (whole array is in order)

(still need to prove that the claimed invariant <u>is</u> the invariant)

```
InsertionSort(A)
  for i = 1 to n-1
    insert A[i] into correct location among A[0],...,A[i-1]
```

INV: $A[0] \le \dots \le A[i-1]$ (1st i in order)

and 1st i elts of A now are the same as original 1st i elts (in possibly different order)

3 | 2 1 5 4

2 3 | 1 5 4

1 2 3 | 5 4

and 1^{st} i elts of A now are
the same as original 1^{st} i elts
(in possibly different order)
and remaining elts are same as before
(and in same order)

2 3 | 1 5 4

i

1 2 3 | 5 4

1 2 3 5 | 4

i