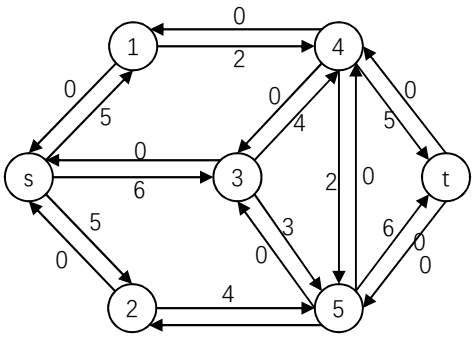
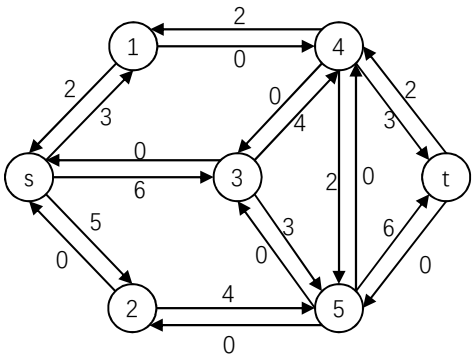
Step 0



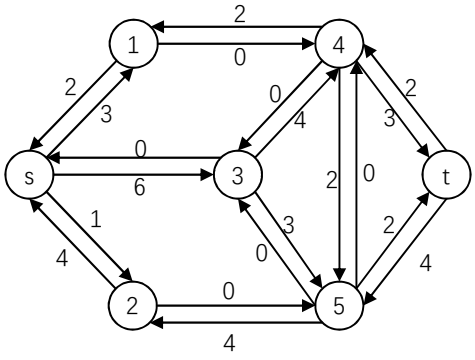
Step 1

S-1-4-t flow = 2



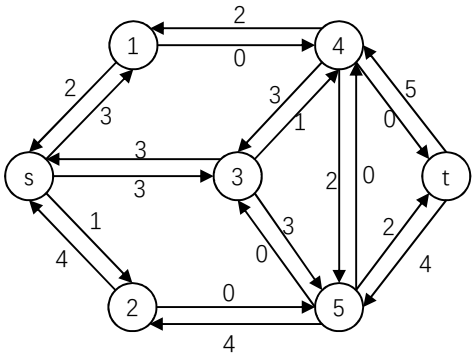
Step 2

S-2-5-t flow = 4



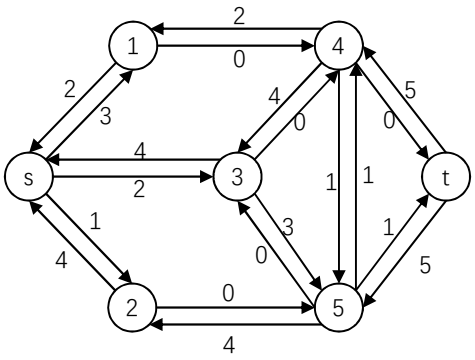
Step 3

S-3-4-t flow = 3



Step 4

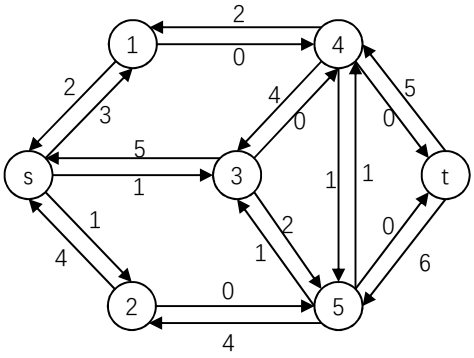
S-3-4-5-t flow = 1



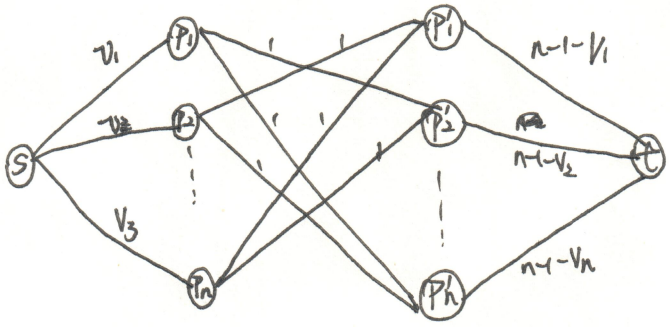
Step 5

S-3-5-t flow = 1

Max flow is 11



P2 define a graph:



Here: S denotes the source node, t denotes the end node.

$P_1, P_2, \cdots P_n$ denote the n players from $1 \to n$

$P_1', P_2', \cdots P_n'$ also denote the players from $1 \to n$, $P_1 = P_1'$, $P_2 = P_2' \cdots$

The edge capacity of $S \to P_i = v_i$, $P_i \to P_j$ $(i \neq j) = 1$

and $P_i' \to t = n-1-v_i$,

↳ means that i^{th} player can lose up to $n-1-v_i$ games.

the algorithm

① define a ~~network~~ directed ~~graph~~ graph shown above

② find the max flow of this graph by Ford-Fulkerson algorithm

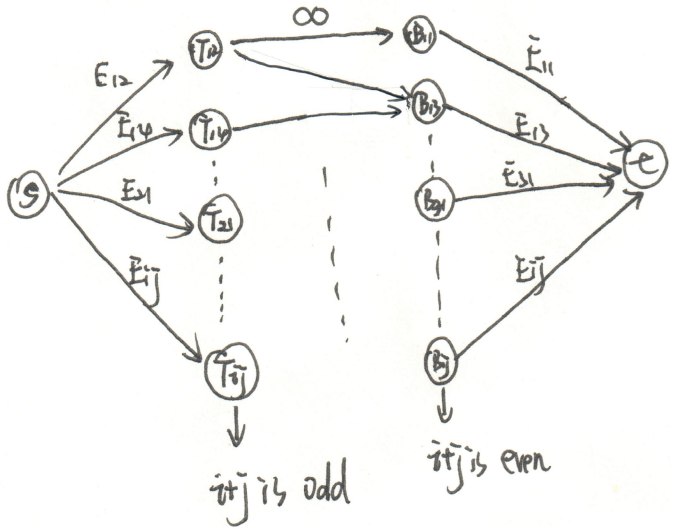
③ ~~the~~ check whether the max flow equals to $\sum_{i=1}^{n} v_i$

if equal, then this vector $(v_1, v_2, \cdots v_n)$ is a possible result

if not ~~- - - - - - -~~, isn't ~~- - - - -~~

P3 Since maximizes the sum of selected elements is equal to minimize the sum of un-selected elements, therefore, here we use min cut to find the minimum value of the sum of un-selected elements.

The graph is defined below:



E_{ij} means the value of element in matrix (i,j)

The relationship of T to B is: if elements in T and B are adjacent elements then connect these elements from T to B, the capacity of this connection is ∞.

Then, we use Ford-Fulkerson algorithm, to find the max flow, then find the min cut. Since the T-B connection has ∞ capacity, the cut must happen in $S \to T$ and $B \to t$. The connections that are cut down is the elements that we do not selected. Such as. if cut down the $B_{11} \to t$, then we'll not select element (1,1), if cut down $s \to T_{21}$, then we'll not select element (2,1).

This algorithm can satisfy the disjoint rule because if $B_{ij} \to t$ preserves, then all $s \to T_{ij}$, where T_{ij} connect to B_{ij}, will be cut.

P4. (a) 1) generate the residual graph by the ~~selection~~ maximum flow before it runs $O(|E|)$ times.

2) try to find an augment path contains the edge e by Ford-Fulkerson

it's worth noting that we only need to use this algorithm once, $O(|E|+|V|)$

3) if there contains a augment path, then the maximum flow will add 1

else, ~~selection~~ remains

4) $O(E)+O(|E|+|V|) = O(|E|+|V|)$

(b) 1) generenerate a residual graph like (a), 1), cost $O(|E|)$ time

2) suppose that the edge e : $x \to y$, x,y are nodes.

find two paths : $s \to x$ and $y \to t$ by BFS, whose flow will both large than 0 (≥ 1)

decrease the flow in these two paths by 1, to satisfy the decrease in edge e. this process cost $O(|E|+|V|)$

3) from $s \to t$, try to find an augment path by Ford-Fulkerson

similar to (a) 2), here we only need to use this algorithm once, $O(|E|+|V|)$

4) If contains an augment path, then the maximum flow remains

else ~~selection~~ will minus 1

5) $O(|E|)+O(|E|+|V|)+O(|E|+|V|) = O(|E|+|V|)$

P5 Here we generate a directed graph below:

① each ~~node~~ element in this table (except (1,1) and (n,m)) is divided into 2 part : output part and input part . the (1,1) only has the output part and (n,m) only has input part.

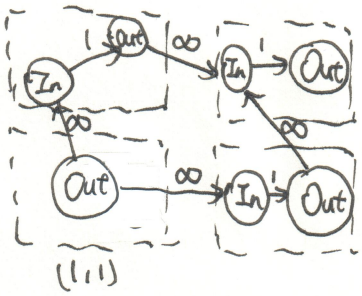
The obstacles don't have output and input part

② build a directed graph follow the rule of robot movement,

the capacity of input → output is 1 , the output → input is ∞ .

Every ~~node~~ element receive flow from input nodes and output flow from output nodes.

Ex.



③ for this graph, we use Ford-Fulkerson algorithm to find the min-cut (or max-flow)

It will have 3 cases

1) there does not have an augment path → add 0 obstacle

2) the min-cut is 1. → add one obstacle at the cut

3) . . — -- .. 2 → . . two - - - — - - -

P6. 1. The algorithm is.

A= {A[X_i]} , m=1

for i=1 : n

  for j = 1 : m

    if A_j - last term < X_i

      A_j . add (X_i)

    endif

  if all A_j - last term ≥ X_i

    A.add (X_i)

  endif

  end

end

Example

2 4 5 2 3

↓ ↓ ↓ ↓ ↓

A, 2   24   245   {245, 2}   {245, 23}

24: ↓ here we add a term in A_1

{245, 2}: ↓ here we add a term in A

check the max length in A , L=max length in A

time complexity of the worst case is $O(n^2)$

2. Similar to 1, we use:

X = [X_1, X_2 ... X_n]

i=1 , count =0

while X is not empty

A=[X_i] , m=1

search all other elements by order, if met elements larger than the last term of A, add it to A immediately, m=m+1

End searching if m=L or search finished.

then, if m=L, count = count+1, delete all the elements corresponding to A in the X

else, delete the first element in X

end

output the count