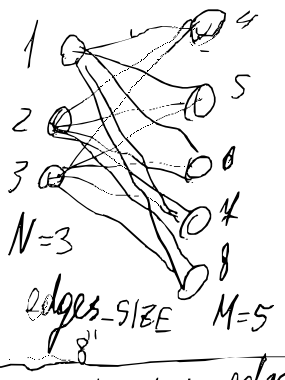
N, M INT ← INPUT

STORAGE[N] INT ← INPUT

MARKETS[M] INT ← INPUT

COSTS[N][M] INT ← INPUT



edges[N+M][INT] VECTOR<UNORDERED_MAP<INT, edge>>

ALL POSSIBLE NODES (i) — ALL POSSIBLE NODES FROM i (j)

struct edge { FLOW INT, COST INT, GOAL INT, IS_ACTIVE BOOL = FALSE }

FOR (INT i=0, j=0, IS_STORAGE=1; i<N && j<M;) {
  edges[i][j].FLOW = MIN(STORAGE[i], MARKETS[j]);
  IF (STORAGE[i] <= MARKETS[j]) {
    MARKETS[j] -= STORAGE[i];
    i++
  } else {
    STORAGE[i] -= MARKETS[j];
    j++;
  }
}

IS_ACTIVE = TRUE

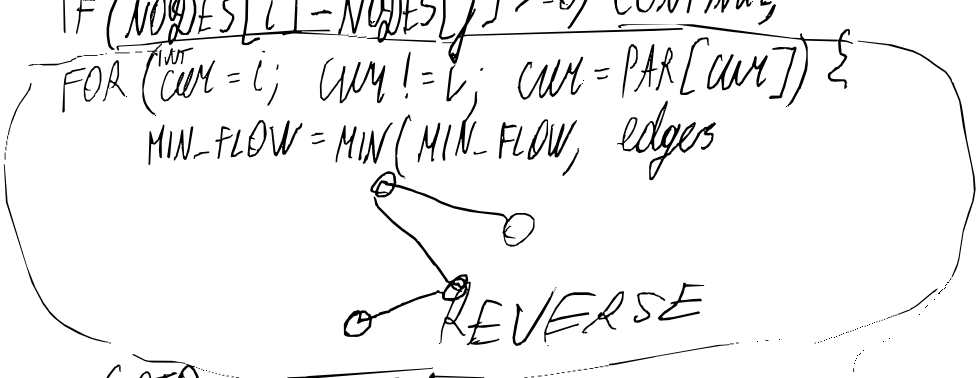
1ST STEP:
MAXFLOW

2ND STEP:
BUILD A TREE

```
INT NODES[N+M]{0}//VECTOR WILL BE EASIER
QUEUE<PAIR<INT, INT>> VISIT_ORDER;
VISIT_ORDER.push({0,0});
UNORDERED_SET<INT> VISITED;
WHILE (!VISIT_ORDER.EMPTY()) {
    PAIR P = VISTORDER.TOP;
    INT current = P.FIRST, PARENT = P.SECOND;
    VISIT_ORDER.POP();
    IF (VISITED.count(current)) continue;
    VISITED.INSERT(current);
    IF (current != PARENT) {
        edge* curredge = edges[current][parent];
        int sum = curredge.cost;
        IF (curredge.goal == current) sum = -sum;
        NODES[current] = sum + NODES[PARENT];
    }
    FOR (CONST PAIR &P: edges[current]) {
        IF (visited.count(P.FIRST)) continue;
        IF (!P.SECOND.IS_ACTIVE) continue;
        VISIT_ORDER.PUSH ({P.FIRST, current});
    }
}
```

3RD STEP:
FIND CYCLE (-) ∞

```
FOR (INT i=0; i<N; i++){
  FOR (INT j=N; j<N+M; j++) {
    IF (edges[i][j].IS_ACTIVE) CONTINUE;
    IF (NODES[i]-NODES[j]>=0) CONTINUE;
    FOR (INT cur=i; cur!=i; cur=PAR[cur]) {
      MIN_FLOW = MIN(MIN_FLOW, edges
```




```
    GOTO 2ND STEP;
  }
}
```

OPTIMAL COST IS FOUND (!)