Offline algorithm by Reingold and Westbrook 1996 on list update problem

Keypoints –

1. Modification of Manasse 1988 algo which runs in O(m*(n!)^2)) and Space O(n!) n is length of list and m is number of requests
2. Time complexity of this algo is O(m*(n-1)!*2^n) and Space O(n!)
3. Operations : access , insert and delete
   Static model : Let’s consider only access request sequence

List update problem and Offline algorithm

- **Access cost –** Access is done by starting from start of list and searching till the point you find requested item.

  Cost of access = position of element in list
  #cost is not same as time complexity

- **Exchange cost –** two items in the list can be transposed side by side for lower access cost in future.
  Exchange cost = no. of exchanges required to convert the list to other list

- **Total cost = ∑ (access cost + exchange cost)**

- Offline algorithm is an algo. that will give total cost for any request sequence given.

Common Terminologies

- Initial list is $\mathbf{L_0}$ and sequence of requests
  ð = < r_1 , r_2 , …,…………., r_m >

- **Free exchanges -** Immediately ***after*** item is accessed that item can be moved forward in the list by repeated exchanges. This is called as free exchanges. They won't cost us anything.

- **Paid exchanges –** List can also be rearranged by paid exchanges . For each exchange 1 unit cost. Paid exchanges may also be done ***before*** the first request arrives.

- *If request sequence is of access requests only then free exchanges are not at all necessary for optimum cost calculation .*

- *Given request sequence :* $\partial = < r_1 , r_2 , \ldots, \ldots\ldots\ldots\ldots, r_m >$
  service sequence: $ß = < E_1 , E_2 , \ldots, \ldots\ldots\ldots\ldots, E_m >$

  where $\boldsymbol{E_i}$ is sequence of exchanges to be performed between serving $\boldsymbol{r_{i-1}}$ and $\boldsymbol{r_i}$.

  Before first request is processed $\boldsymbol{E_1}$ is employed on $\boldsymbol{L_0}$ to form new list $\boldsymbol{L_1}$,
  before second request is processed $\mathbf{E_2}$ is employed on $\mathbf{L_1}$ to form new list $\boldsymbol{L_2}$
  and so on.

- ***Net cost*** is defined as = ***cost*** $\boldsymbol{(L_0 , \partial , ß)}$
  when ß is employed service sequence.

- The minimum cost to service **∂** is denoted ***opt (∂).***

- There are several service sequences that achieve the optimum cost. An optimum offline algorithm takes ∂ as input and computes opt(∂) and a service sequence that realizes this cost.

•

- ***Theorem 1*** :

  For any sequence of accesses ∂, there exists a service sequence containing no free exchanges that achieves the minimum cost to service ∂ in the standard model.

  Proof:

  Let's consider service sequence with both paid and free exchanges allowed then we will transform that service sequence in only paid exchanges with the same cost.

  Free exchange : initial position at l → then free exchanges to move to location m (m < l) .
  Cost is only access cost = l

  Paid exchange : Initial paid exchanges to move to location m and then access cost .
  Exchanges = l-m and access cost = m
  Net cost for this request = l;

- Hence, all free exchange requests can be turned to paid exchange requests with same cost.

- ***Inversion table:***
  The inversion table of a permutation L is a sequence (a_1, a_2, a_3, ……., a_n) where a_i is the number of elements less than i and to its right in L .
  Example :
  Let**,** **L_0 = < 1, 2, 3 , 4 , 5 >** and
  L_1 = < 2, 3,1,5,4> then inversion table is **< 0, 1, 1, 0, 1 >**

- Inversion table can be used to identify every permutation of L uniquely and can be encoded by $n(L) = {}^{n}\sum_{j=2} a_j(j-1)!$

  This will identify every permutation with unique code.

- Minimum number of exchanges of adjacent elements required to convert **L_1** to **L_2** is equal to $| inv\{L_1, L_2\} |$ and is addition of COMPONENTS in inversion table.

- ***Every permutation of L can be derived from L by exchange method of Johnson – Trotter algorithm***

- ***Motivation of Algorithm :***
  Between each pair of accesses there are, n! ways to rearrange the list.
  We show that in computing opt(∂) it is possible to restrict our attention to at most 2^n of these rearrangements.

- ***Subset Transfer:***

  *It is sequence of **paid exchanges made just prior** to the access(x) that moves a **subset** of items **preceding x** to to just behind x in such a way that **relative order** in subset is not changed.*

  Ex. - <2,4,***1***,5,3>
  Let ***access(1)*** is called then lists that can form by subset transfer are:

  <2,4,1,5,3> **<2,1,4,5,3>** **<4,1,2,5,3>** **<1,2,4,5,3>**

  position of 1 is at k = 3
  then number of subset transfer lists formed are = 2^{k-1}

  subsets(2,4) = null , [2], [4] , [2,4]

- **Theorem 2 :**

  *Let ∂ = < r_1 , r_2 , …,…………….., r_m > be request sequence . Then there is an optimal service sequence **ß** in which any rearrangement is subset transfer rearrangement.*

  Proof

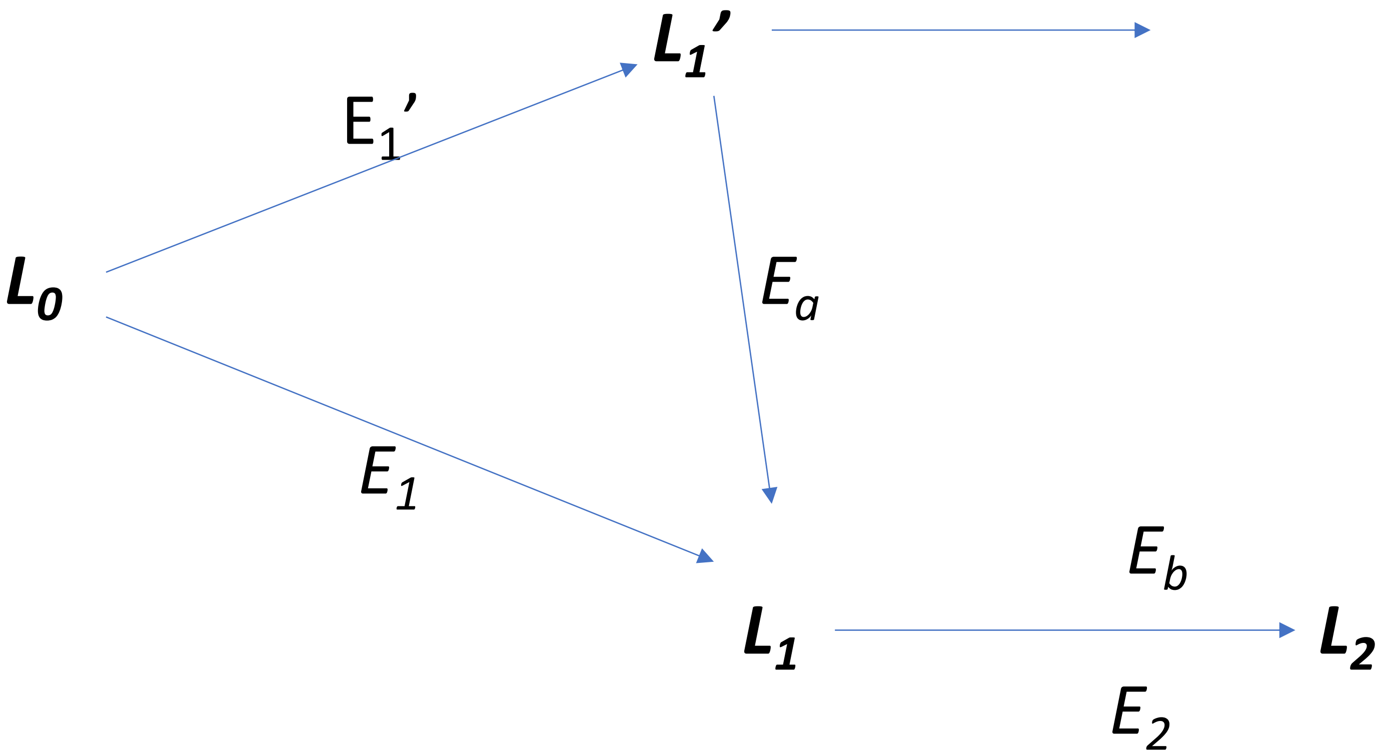
  by induction over number of requests:
  For m = 1: trivially true as list L is also subset transfer of itself.
  Let it be true till (m-1) requests.
  We need to prove for m^{th} request.

  Let ß = < E_1 , E_2 , …,…………., E_m > be arbitrary service sequence which has only paid exchanges …..(From Theorem 1)

  We will show that **ß** can be converted to **ß'** which has only subset transfer exchanges and
  cost (L_0 , ∂ , ß') ≤ cost (L_0, ∂ , ß)
  i.e an alternate path to reach same place with less cost and that path has only subset transfers .

Let L_1' be list formed by E_1' . Inductive hypotheses on *L_1'* there exists (m-1) length optimal subset transfer service sequence from *L_1'*.
First request r_1 is access(x). L_1 be list after after applying E_1 to L_0.

•



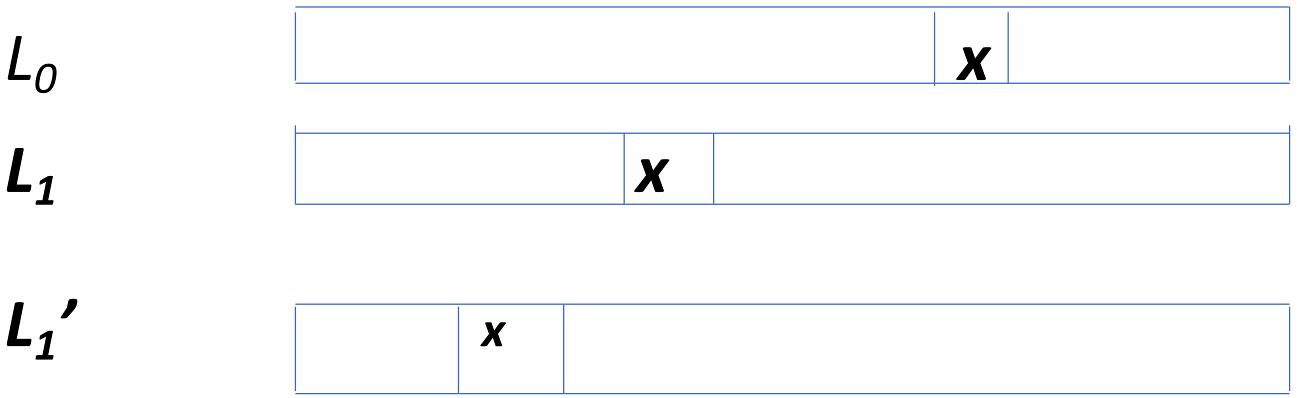
$E_1 – E_2$ *is normal earlier path from* ***L_0 - L_1 – L_2***

E_1' *has only subset transfers*

E_a *is inversion table for* ***L_1' – L_1***

$E_b = E_2$

- *Let **R** be the items that are in front of x in both L_0 and L_1 ;*



- *let **S** be the items that are in front of x in L_0 but behind x in L_1*

- *Let **T** be the items that are behind x in L_0 but in front of x in L_1*

- *Number of items in front of x in L_1 = r + t*
  Let, p_0 is number of exchanges in E_1
  ***access cost of x** = p_0 + r + t + 1*

- *For subset transfer L_0 to L_1' : no elements from back would move forward. Hence , T = null set*
  Let, p_1 is number of exchanges in E_1'
  access(x) from L_1'= r . Let, p_2 be number of exchanges of E_a
  Total cost from other path = $p_1 + r + 1 + p_2$

Show, $p_0 \geq p_1 + p_2$

inversion between L_0 *and* L_1' *must involve an element of* ***S*** *and either* ***x*** *or an element of* ***R****.*
i.e of this form ***({S},x)*** *or* ***({S},{R})***

e.g – (1,2,3,4,5) is L_0 and
(2,3,1,4,5) is L'_1.
L_1 is subset transfer of L_0 over 3.

R= {2} and S = {1}
Inversions involved are (2,1) followed by (3,1)

- *No inversion between L_1' and L_1 can be of this form. L_1' is formed by subset transfer and we need to convert it to L_1.*
- *Hence , proved that 2 inversions are partition between L_0 and L_1.*
  Hence, new path from L_1' is the shortest path from L_0 to L_1

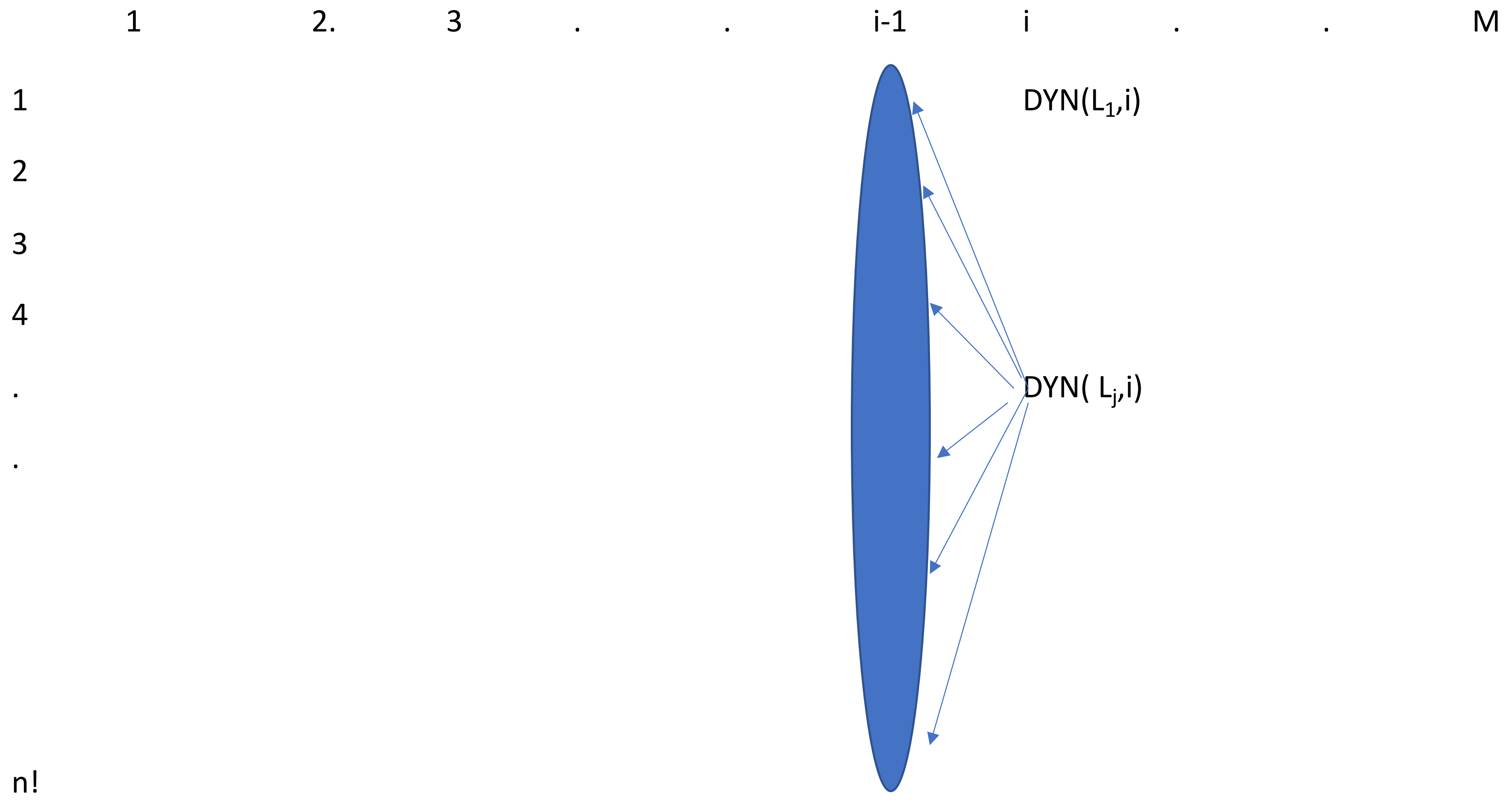
Hence , proved that $p_0 \geq p_1 + p_2$.

- **Implementation Of Optimum algorithm**

Let DYN(L , i) be optimum cost to service first i requests and after serving i requests we end with L starting from L_0.
POS(r , L) be position of r in L
MOV(L_1 , L_2) is inversion cost of L_1 to L_2

DYN(L , 0) = MOV(L_0 , L) *# no requests are served yet*

- ***DYN(L , i) = $\min_{L'}$ { DYN(L' , i-1) + POS(r_i , L') + MOV(L' , L) }***


1
2.
3
.
.
i-1
i
.
.
M
1
2
3
4
.
.
n!
DYN(L1,i)
DYN(Lj,i)

- *By using Theorem 2 , we can consider only those Lists in the previous column whose subset transfer leads to L_j*

  Method to know number of such lists:

- *Consider a row of k-1 girls followed by one Supervisor and n-k boys further.*

G_1. G_2 G_3 G_4. G_{k-1} S B_1. B_2. B_3. B_4. B_{n-k-1}

B1 will go and socialize with all girls this can be done in $\binom{k}{1}$ *ways.*

B2 will now go and socialize with girls but he can't cross B1 ------- $\binom{k+1}{2}$ *ways.*

And so on.

Sum of all these ways = $\binom{k}{1} + \binom{k+1}{2} + \binom{k+2}{3} + \qquad + \binom{n-1}{n-k} + 1$

Euler's identity : will add to $\binom{n}{k}$

- If our access element is at position k then we need to consider $\binom{n}{k}$ *elements of previous column*
-
- *No. of lists with access element at position k = (n-1)!*

- *Total lookups for i^{th} column =* $\binom{n}{1}(n-1)! + \binom{n}{2}(n-1)! + \binom{n}{3}(n-1)! + \ldots\ldots$

$$\ldots\ldots\ldots \binom{n}{n}(n-1)!$$

$$= (2^n - 1)\,(n-1)!$$

$$= O(2^n * (n-1)!)$$

For all columns $O(m * 2^n * (n-1)!)$