① یک list از n عدد به نام P داریم. یک کپی از این لیست تهیه می‌کنیم، P و لیست P' را سورت می‌کنیم. حال کافی است الگوریتم Longest Common subsequence را روی این دو لیست اجرا کنیم. طولانی‌ترین زیررشته مشترک بین این دو لیست جواب مسئله است. LCS به دست آمده از P به صورت صعودی سورت شده است و به طور واضح طولانی‌ترین طول را دارد.

LCS اوردر زمانی ← از بخاطر هر دو لیست $O(n \times n) = O(n^2)$: $O(m \times n)$ طول n دارند ⟹ $O(n^2)$

اوردر زمانی سورت کردن ← عمومی‌ترین الگوریتم سورت : $O(n \lg n)$

بدون داشتن هیچ شرطی روی مسئله

$O(n \lg n)$ است

اگر بخواهیم به صورت صعودی این سورت بکنیم ابتدا عناصر متوالی P را حذف می‌کنیم که این از اوردر زمانی خطی است که باز زمان کل مسئله از $O(n^2)$ خواهد بود.

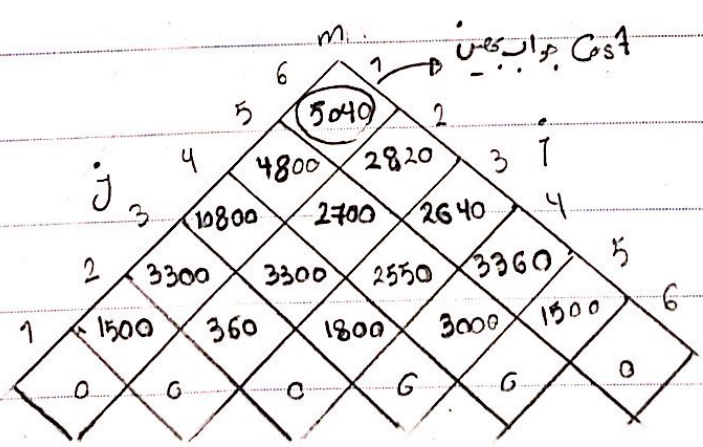
---

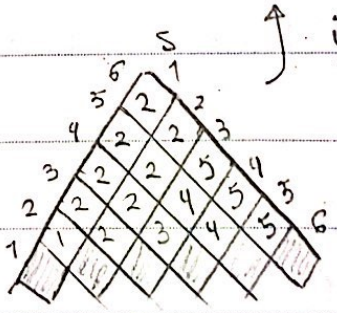
②

M	A_1	A_2	A_3	A_4	A_5	A_6
D	50×10	10×3	3×12	12×50	50×5	5×6

$((A_1A_2)((A_3A_4)A_5)A_6)$

جواب بهین ↑





محاسبات:

$m[1,2] = m[1,1] + m[2,2] + P_0P_1P_2 = 1500$

$m[2,3] = m[2,2] + m[3,3] + P_1P_2P_3 = 360$

$m[3,4] = m[3,3] + m[4,4] + P_2P_3P_4 = 1800$

ادامه صفحه بعد ←

$$m[4,5] = m[4,4] + m[5,5] + P_3 P_4 P_5 = 3000$$

$$m[5,6] = m[5,5] + m[6,6] + P_4 P_5 P_6 = 1500$$

$$m[1,3] = \min \begin{cases} m[1,2] + m[3,3] + P_0 P_2 P_3 = 3300 \\ m[1,1] + m[2,3] + P_0 P_1 P_3 = 6360 \end{cases} = 3300$$

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 = 28300 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 = 10800 \\ m[1,3] + m[4,4] = P_0 P_3 P_4 = 33300 \end{cases}$$

⋮

---

(3) ابتدا یک جواب بازگشتی برای مسئله می‌یابیم سپس یک الگوریتم dp ارائه داده و optimal substructure آن را بررسی می‌کنیم.

$$C[i] = \begin{cases} 0 & i = 0 \\ \min\left(\{ C[j] + t_i(x_i - x_j) \mid x_i - x_j \leq 100 \} \right) & 0 < i < n+1 \\ \qquad 0 \leq j < i & \end{cases}$$

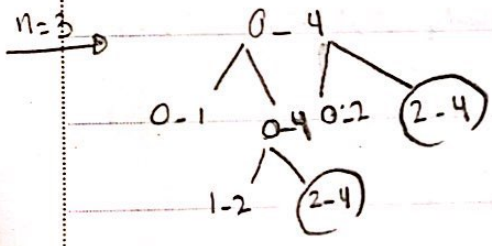
$t[0], t[n+1]$ را صفری می‌گذاریم زیرا نقاط X_0 و X_{n+1} در اصل مبدا و مقصد ما هستند.

```
STATION(X,T)
  let C[0,...,n+1], S[0,...,n+1], T'[0,...,n+1] be new arrays
  n = X.length + 2
  T'[0] = T'[n+1] = 0
  for i=1 up to n:
      T'[i] = T[i]
  C[0] = 0
```

ادامه صفحه‌ی بعد ←

```
for i=0 up to n+1
    C[i]=∞
    for j=0 up to i:
        if X[i]-X[j]≤100:
            q=C[j]+T[i](X[i]-X[j])
        if q<C[i]
            C[i]=q
            S[i]=j
```

- در الگوریتم بازگشتی به وضوح دچار Overlap می‌شویم زیرا در حقیقت آن برای n های به اندازه‌ای بزرگ زیردولت‌های یکسان به تعدادی تکرار می‌شوند که الگوریتم دچار زمانی از مرتبه نمایی می‌شود.



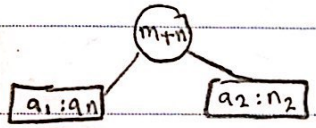
- قطعا optimal substructure داریم. گذشته باشیم با استفاده از متد cut and paste ایجادی کنیم. فرض کنیم جزا ایستگاه دارم باشیم و بخواهیم C[i] را حساب کنیم و برای آن که min آن را بیابیم از جمع کردن زمان ایستگاه‌های قبلی با زمان این ایستگاه استفاده می‌کنیم. حال از C[j] ای که انتخاب می‌کنیم min باشد پس قطعا j ای وجود دارد که C[j] آن min باشد پس این با این که min، C[i] باشد در تناقض پس j را انتخاب می‌کنیم.

(4) فرض کنیم n=36 باشد. طبق الگوریتم اول 1، 25 سنتی برمی داریم. 11 = 25-36 حال با این 11 سکه‌ی 1 سنتی برداریم که در مجموع می شود 12 سکه. ولی بهترین حالت زمانی است که 3 سکه‌ی 10 سنتی، 6 سکه‌ی 1 سنتی انتخاب می کنیم که در مجموع 7 سکه. پس الگوریتم Greedy ارائه شده خوب نیست.

---

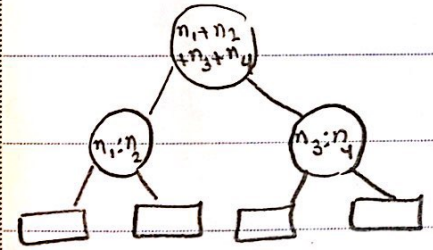
(5) نیازی به استفاده از الگوریتم هافمن نیست زیرا طول دقت هایی تولید شده با استفاده از این الگوریتم حالتی که Fixed-length را پوشش برابر 8 می شود.

زیرا: می دانیم مجموع تمام احتمال های کاراکترها به صورت تکی می باشد پس در هر مرحله از تشکیل دقت در مرحله‌ی اول 128 دقت به صورت زیر خواهد بود:



زیرا در هر دو نمایی را که جمع می کنیم به آخر صف می رود.

در مرحله‌ی بعد هم بازهم این اتفاق می افتد:



از آنجایی که هر بار تعداد دقت ها نصف می شود

داریم: $depth(T) = \lg(256) = 8$ ☐

(6) برای ساده سازی چند فرض می کنیم. کل مسیر را خطی به طول L در نظر می گیریم.
و فرض می کنیم هر طول روز به اندازه ی d واحد از L می توانیم پیش برویم (بدون در نظر گرفتن شرایط آب و هوا، زمین و ...). فرض کنیم نقاطی که در آن ها می توانیم توقف کنیم در فاصله ی $x_{11}, x_{21}, \dots, x_{n1}$ از نقطه ی شروع قرار دارند. و همچنین فرض می کنیم حساب کتاب ما برای بررسی رسیدن به نقطه ی توقف بعدی قبل از تاریکی هوست است.

یک مجموعه از نقاط توقف را قابل قبول می گوییم اگر فاصله ی هر دو نقطه ی مجاور در آن مجموعه حداکثر d باشد، اولین نقطه در فاصله ی حداکثر d از نقطه ی شروع است و آخری در فاصله ی حداکثر d از نقطه ی پایان است. پس یک مجموعه از نقاط قابل است اگر بتوانیم در این نقاط توقف کنیم و همچنان بتوانیم به آخر راه برسیم. پس فرض می کنیم مجموعه ی تمام نقاط توقف که یک مجموعه ی n نقطه ای است یک مجموعه ی قابل قبول است چون اگر نباشد هیچ راهی برای رسیدن به نقطه ی پایان وجود ندارد.

یک نگرانی که در رابطه با این الگوریتم وجود دارد این است که آیا توقف زود هنگام در یک روز به هنگام سازی موقعیت های توقف در روزهای بعدی تاثیر می گذارد یا خیر. آیا ممکن است که راه حل متفاوتی وجود داشته باشد که از الگوریتم Greedy عقب بیافتد اما در ادامه از آن جلو بزند؟ خیر نه ممکن است و نه می خواهیم حرکت ماکسیمم در طول روز را توصیه می کند. پس با استفاده از تکنیک "stay in ahead" نشان می دهیم که الگوریتم ما چون در هر روز از بقیه راه حل ها جلوتر است هیچ راه حلی نمی تواند روز بعد از آن پیشی بگیرد.

مجموعه ی $R = \{x_{p_1}, \dots, x_{p_k}\}$ را مجموعه ی نقاط توقفی در نظر می گیریم که الگوریتم Greedy حاصل می کند.
با برهان خلف فرض می کنیم که یک مجموعه ی Valid از نقاط توقف کمتر وجود دارد. $S = \{x_{q_1}, \dots, x_{q_m}\}$
که $m < k$ است.

برای رسیدن به تناقض نشان می دهیم که نقطه توقف حاصل شده از الگوریتم Greedy در هر روز ن دورتر از نقطه توقف حاصل شده با استفاده از راه حل تناوبی است.

برای هر $m=1,2,\dots,k$ ن داریم: $x_{p_j} \geq x_{q_j}$

با استقرا روی ن اثبات را کامل می کنیم: اگر $j=1$ باشد به طور مستقیم از الگوریتم Greedy پیروی می کند: "در طول روز به اندازه ی ممکن قبل از توقف حرکت کنید."

حال اگر ن در نظر بگیریم و فرض کنیم که ادعای ما برای $j-1$ برقرار است. پس داریم:

$$x_{q_j} - x_{q_{j-1}} < d$$

وچون S یک مجموعه ی Valid از نقاط توقف است، پس:

$$x_{q_j} - x_{p_{j-1}} \leq x_{q_j} - x_{q_{j-1}}$$

چون $x_{p_{j-1}} \geq x_{q_{j-1}}$ است پس دوتا ناوی بالا ناوی زیر را نتیجه می دهند:

$$x_{q_j} - x_{p_{j-1}} \leq d.$$

این به این معنا است که ما می توانیم در طول روز از نقطه ی $x_{p_{j-1}}$ تا x_{q_j} حرکت کنیم.

و از آنجایی که موقعیت نقطه ی x_{p_j} در جایی که در نهایت توقف می کنیم می تواند از نقطه ی x_{q_j} بیشتر باشد به تناقض می رسیم.

اثبات بالا نشان می دهد که $x_{q_m} < x_{p_m}$ حال اگر $m < k$ باشد پس داریم $x_{p_m} < L-d$ در غیر اینصورت هیچ گاه نیاز به توقف در نقطه ی $x_{p_{m+1}}$ را نداریم. ترکیب این دو ناوی می دهد:

$$x_{q_m} < L-d$$

که این با Valid بودن مجموعه نقاط توقف S در تناقض است. پس نمی تواند $m < k$ باشد پس الگوریتم Greedy یک مجموعه ی قابل قبول از نقاط توقف را در کوچکترین اندازه به ما می دهد.