تحلیل و طراحی الگوریتم

جلسه اول

ملکی مجد

- مقدمه

- برنامه نویسی پویا
  - مساله تقسیم بندی متن Text Segmentation

- *مبحث برنامه نویسی پویا از فصل ۱۵ کتاب CLRS تدریس می شود.*

معرفی درس

- malekimajd@iust.ac.ir
- Room 313

- Telegram group and Channel
- LMS

- References
  - Books!
    - Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (CLRS)
  - Internet
  - Prof. and TAs

مباحث درس

- *Dynamic Programming*
- *Greedy Algorithms*
- *Elementary Graph algorithms*
- *Minimum Spanning Trees*
- *Single-Source Shortest Paths*
- *All-Pairs Shortest Paths*
- *Maximum Flow*
- *NP-Completeness*

- الگوریتم
- اثبات درستی الگوریتم
- زمان اجرا
- روش های کارا برای حل مسئله

یاداوری - بازگشت Recursion

- برج هانوی
- مرتب سازی ادغامی
- مرتب سازی سریع
- درخت بازگشتی
- انتخاب میانه، کمترین و بیشترین مقدار
- انتخاب ترتیب آماری Order Statistics

برنامه نویسی پویا Dynamic Programming

- معمولا برای مسئله هایی استفاده می شود که چندین راه حل وجود دارد و می خواهیم یک راه حل با مقدار بهینه (کمترین یا بیشترین) محاسبه کنیم.

- "برنامه نویسی" با آنچه به عنوان مبانی برنامه نویسی خوانده اید ارتباطی ندارد. اینجا برنامه نویسی به خاطر استفاده از روش جدولی است.
  - مقدار راه حل های میانی معمولا در یک آرایه یا جدول ذخیره می شوند.

- یادگیری روش برنامه نویسی پویا از طریق مثال و حل کردن تمرین حاصل خواهد شد.

- به جای این که برای محاسبه جواب مسئله بزرگ، سراغ زیر مسئله های کوچکتر برویم، از زیر مسئله های کوچکتر شروع می کنیم به محاسبه جواب زیر مسئله های بزرگتر تا به مسئله بزرگ در ورودی برسیم.
  - جواب مسئله ای کوچک که برای حل چندین مسئله بزرگ تر لازم است، تنها یک بار محاسبه می شود
  - جواب مسئله بزرگ، با ترکیب جواب بهینه زیرمسئله های کوچکتر محاسبه می شود

محاسبه مقدار فیبوناچی

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

- روش بازگشتی
  - زیرمسئله های تکراری

- محاسبه از بالا به پایین
- محاسبه از پایین به بالا

Text Segmentation کلمه بندی متن

PRIMVSDIGNITASINTAMTENVISCIENTIANONPOTESTESSERESENIMSVNT

- We are given a string *A*[1 .. *n*] and a subroutine *IsWord* that determines whether a given string is a word (whatever that means), and we want to know whether *A* can be partitioned into a sequence of words.

- راه حل بازگشتی
- راه حل برنامه نویسی پویا

کلمه بندی متن

$$Splittable(i) = \begin{cases} \text{TRUE} & \text{if } i > n \\ \bigvee_{j=i}^{n} \left(IsWord(i, j) \wedge Splittable(j+1) \right) & \text{otherwise} \end{cases}$$

- *Splittable*(*i*) returns True if and only if the suffix *A*[*i* .. *n*] can be partitioned into a sequence of words
- *IsWord*(*i*, *j*) is shorthand for *IsWord*(*A*[*i* .. *j*])

کلمه بندی متن

$$Splittable(i) = \begin{cases} \text{TRUE} & \text{if } i > n \\ \bigvee_{j=i}^{n} \left(IsWord(i,j) \wedge Splittable(j+1) \right) & \text{otherwise} \end{cases}$$

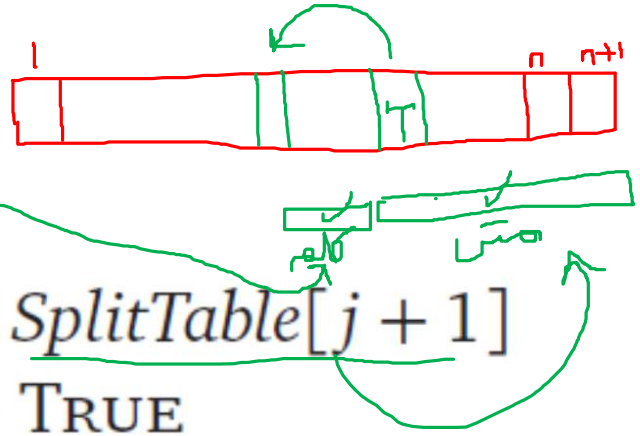
Dictionary:{a,aba,baa,…}

abaacabada (S(1))-> baacabada (S(2))-> cabada *(S(5))*

-> acabada (S(4))-> cabada *(S(5))*

کلمه بندی متن

```
FastSplittable(A[1..n]):
    SplitTable[n+1] ← True
    for i ← n down to 1
        SplitTable[i] ← False
        for j ← i to n
            if IsWord(i, j) and SplitTable[j+1]
                SplitTable[i] ← True
    return SplitTable[1]
```



Memoization

- a variation of dynamic programming
  - maintains an entry in a table for the solution to each subproblem.
  - When the subproblem is first encountered during the execution of the recursive algorithm, its solution is computed and then stored in the table

Memoization uses recursion and works top-down, whereas Dynamic programming moves in opposite direction solving the problem bottom-up

Both are applicable to problems with Overlapping sub-problems.

- *Memoization performs comparatively poor to DP due to the overheads involved during recursive function calls.*
- *The asymptotic time-complexity remains the same.*