روش حریصانه
GREEDY APPROACH

طراحی و تحلیل الگوریتم ها

روش حریصانه

- روشی برای حل مسائل بهینه سازی
- ساخت گام به گام پاسخ بهینه
- خاصیت بهینگی محلی: در هر گام بهترین گزینه ممکن مقطعی (محلی) انتخاب می گردد و ثابت می شود این انتخاب بخشی از پاسخ بهینه نهایی است.

روش حریصانه

- معمولا سریعتر از روشهای دیگر است
- مسائل نمونه
  - خُرد کردن پول با سکه های مخصوص
  - انتخاب بزرگترین زیرمجموعه سازگار کارها
  - کوله پشتی کسری
  - الگوریتم های پردازش گراف
    - درخت پوشای کمینه با الگوریتم های کراسکال، پریم
    - کوتاهترین مسیر با الگوریتم دایکسترا و بلمن فورد
  - کدگذاری هافمن
  - زمان بندی کارها

روش حریصانه

خُرد کردن پول با سکه های مشخص

خُرد کردن پول با سکه های مشخص

- سکه های موجود: ۱، ۵ ،۱۰، ۲۵ ، ۱۰۰ ریالی
  - از هر سکه تعداد زیادی (فرض کنید نامتناهی) موجود است
- مبلغ n ریال پول را با کمترین تعداد سکه های فوق خرد کنید
  - $n= n_1x100+n_2x25+n_3x10+n_4x5+n_5$ و $n_i \geq 0$
  - مثال: ۴۶ = ۱×۲۵ + ۲×۱۰ + ۱×۱ : ۴ سکه
  - ۴۶ = ۴×۱۰ + ۱×۵ + ۱×۱ : ۶ سکه
  - ۴۶ = ۹×۵ + ۱×۱ : ۱۰ سکه

خُرد کردن پول با سکه های مشخص

- سکه های موجود: ۱، ۵، ۱۰، ۲۵ ، ۱۰۰ ریالی
- الگوریتم صندوق دار Cashier's Algorithm
  - همواره بزرگترین سکه ممکن را انتخاب کنید:

- Min_Coin_Change(n)
  - Sort coins $c_1 \leq c_2 \leq c_3 \leq \ldots \leq c_m$
  - S={} // empty set of coins
  - While n >0
    - Select largest possible $c_k \leq n$
    - $S \leftarrow S\_ \, c_k$, $n \leftarrow n - c_k$
  - Return S

خُرد کردن پول با سکه های مشخص

- اثبات بهینگی با برهان خلف

- Cashier’s Algorithm: $a_1 + 5a_2 + 10a_3 + 25a_4 + 100a_5$
- Other Algorithm: $b_1 + 5b_2 + 10b_3 + 25b_4 + 100b_5$
- $a_1 + a_2 + a_3 + a_4 + a_5 > b_1 + b_2 + b_3 + b_4 + b_5$
- فرض کنید $a_5 >= b_5$. مثلا $a5=7$, $b5=6$
  - حداقل صدریال از مبلغ در روش دیگر با سکه های ۱ تا ۲۵ ریالی نمایش داده شده است.
  - می توانیم تعدادی از سکه های ۱ تا ۲۵ ریالی را با یک سکه ۱۰۰ ریالی جایگزین کنیم.
    - مثلا: با ۴ تا سکه ۲۵ ریالی یا ۱۰ سکه ۱۰ ریالی و ....
    - به این ترتیب تعداد سکه های روش دیگر را کاهش دادیم! پس بهینه نبوده است!
- برای سایر سکه ها می توانید به همین ترتیب عمل کنید.

خُرد کردن پول با سکه های مشخص

- آیا الگوریتم صندوق دار برای هر ترکیب سکه ای درست کار می کند ؟
  - مبلغ ۱۱ ریال را با سکه های ۱، ۵، ۶ و ۹ ریالی خُرد کنید
    - الگوریتم صندوق دار: ۱۱ = ۹+۱+۱ : ۳ سکه
    - برنامه ریزی پویا : ۱۱ = ۶+۵ : ۲ سکه
- الگوریتم صندوق دار برای چه ترکیب سکه ای درست کار می کند؟

خُرد کردن پول با سکه های مشخص

- پیچیدگی محاسباتی

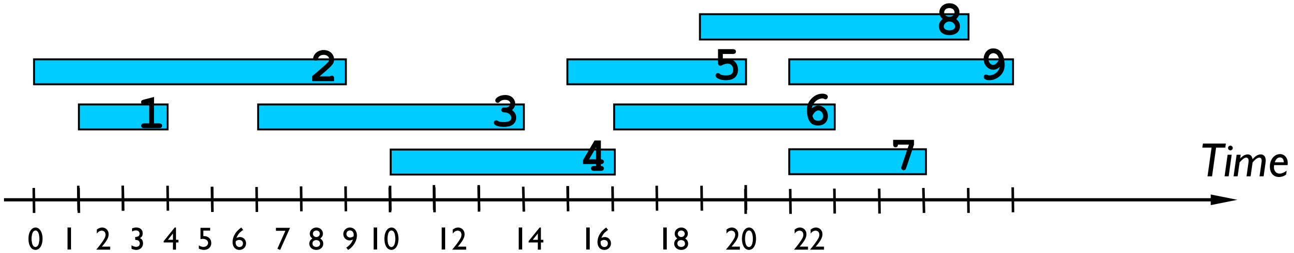
- Min_Coin_Change(n)
  - Sort coins $c_1 \le c_2 \le c_3 \le \ldots \le c_m$ ➔ O (m log m)
  - S={} // empty set of coins ➔ O(1)
  - While n >0 ➔ worst case , O(n)
    - Select largest possible $c_k \le n$
    - $S \leftarrow S\_ c_k$, $n \leftarrow n - c_k$
  - Return S
- Total cost: O (m log m) + O(n)
- $n \ge m^m$,Total cost: O(n)

روش حریصانه برای

انتخاب زیرمجموعه سازگار کارها
THE ACTIVITY SELECTION PROBLEM

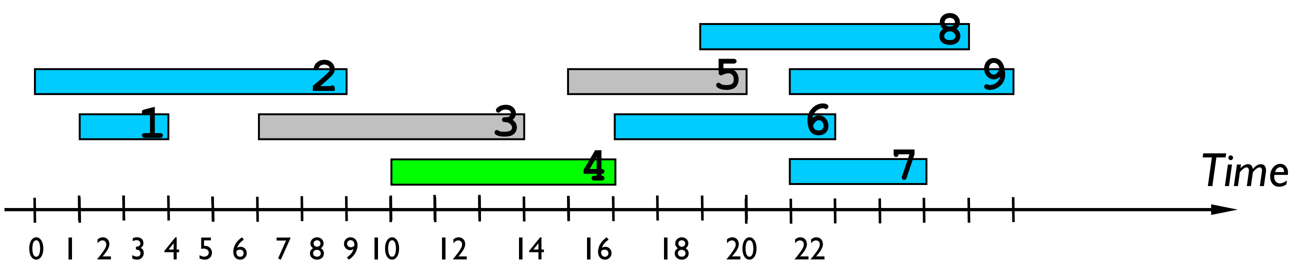
انتخاب بزرگترین زیرمجموعه سازگار

- Input:
  - A [1..n] : set of *n* activities,
  - *A[i].s* : Start of i-th activity
  - *A[i].f.* : End of of i-th activity
- Output:
  - The ***largest*** subset of mutually *compatible* activities
    - Activities are compatible if their intervals do not intersect



یک راه حل ابتدایی

- Let's just pick (schedule) one activity *A*[*k*]
  - This generates two set's of activities compatible with it: *Before*(*k*), *After*(*k*)
    - E.g., Before(4) = {1, 2}; After(4) = {6,7,8,9}



  - Solution:

$$MaxN(A) = \begin{cases} 0 & \text{if } A = \varnothing, \\ \max\limits_{1 \le k \le n} \{MaxN(Before(A)) + MaxN(After(A)) + 1\} & \text{if } A \neq \varnothing. \end{cases}$$

الگوریتمی مبتنی بر برنامه ریزی پویا

- The recurrence results in a dynamic programming algorithm
  - Sort activities on the start or end time (for simplicity assume also "sentinel" activities $A[0]$ and $A[n+1]$)
  - Let S_{ij} – a set of activities after $A[i]$ and before $A[j]$ and compatible with $A[i]$ and $A[j]$.
  - Let's have a two-dimensional array, s.t., $c[i, j] = MaxN(S_{ij})$.

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \varnothing, \\ \max\limits_{i<k<j}\{c[1,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \varnothing. \end{cases}$$

  - $MaxN(A) = MaxN(S_{0,n+1}) = c[0, n+1]$

الگوریتمی مبتنی بر برنامه ریزی پویا

- Does it really work correctly?
  - We have to prove the optimal sub-structure:
    - *If an optimal solution A to S_{ij} includes A[k], then solutions to S_{ik} and S_{kj} (as parts of A) must be optimal as well*
    - To prove use "cut-and-paste" argument
- What is the running time of this algorithm?

انتخاب حریصانه

- What if we could choose "the best" activity (as of now) and be sure that it belongs to an optimal solution
  - We wouldn't have to check out all these sub-problems and consider all currently possible choices!
- Idea: Choose the activity that finishes first!
  - Then, solve the problem for the remaining compatible activities

```
MaxN(A[1..n], i)  //returns a set of activities
m ← i + 1
while m ≤ n and A[m].s < A[i].f do
    m ← m + 1
if m ≤ n then return {A[m]} ∪MaxN(A, m)
         else return ∅
```

بهینگی انتخاب حریصانه

- What is the running time of this algorithm?
- Does it find an optimal solution?:
  - We have to prove the *greedy-choice property*, i.e., that our locally optimal choice belongs to some globally optimal solution.
  - We have to prove the *optimal sub-structure* property (we did that already)
- The challenge is to choose the right interpretation of "the best choice":
  - How about the activity that starts first
    - Show a *counter-example*

دانشگاه بوعلی سینا

الگوریتم حریصانه برای حل

کوله پشتی کسری
THE FRACTIONAL KNAPSACK PROBLEM

کوله پشتی کسری

یک کوله پشتی با ظرفیت وزنی معین و چندین کالا هر کدام با وزن و ارزش معین داده شده اند.

از هر کالا تمام یا بخشی از آن را می توان برداشت.

کوله پشتی را طوری پر کنید که بیشترین ارزش را داشته باشد

مثال: ظرفیت کوله پشتی ۵ کیلوگرم است:

#	W	B
1	1	8
2	3	6
3	5	5

کوله پشتی کسری

در این روش کسری از itemهارا بر میداریم.

#	W	B	B/W
1	5	50	10
2	20	140	7
3	10	60	6

کوله پشتی کسری

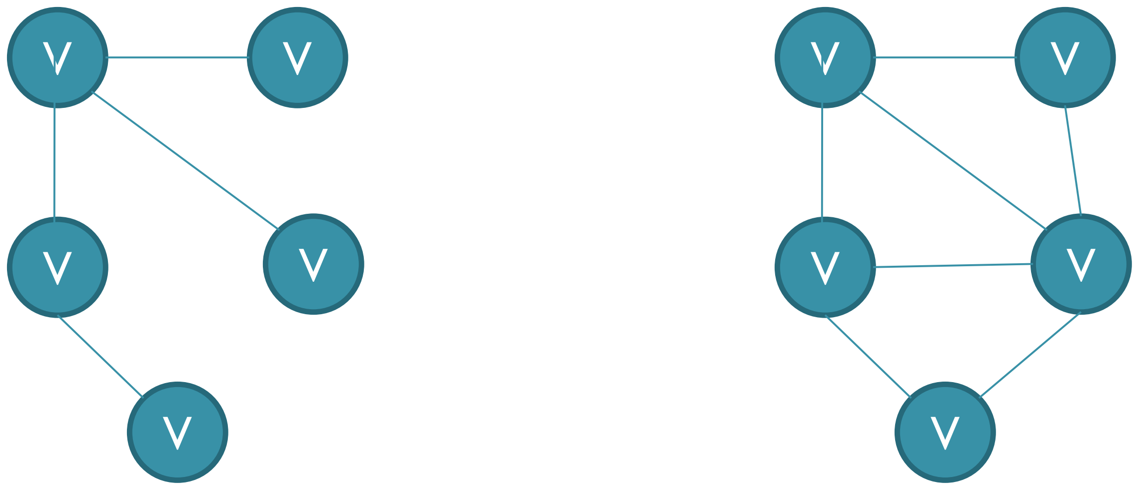
- Fractional_KS(V , w[] , p[] , n)
  - Sort w[] , p[] by p[i]/w[i] , decreasing order
  - S[i] =0 , for i=1,2,..,n
  - B=0
  - For i=1 to n
    - If V >0 then
      - S[i] = min(V, w[i])
      - V = V – S[i] , b = b +S[i] x (p[i]/w[i])
  - Return S , b

روش حریصانه برای یافتن

درخت پوشای کمینه گراف

درخت پوشای گراف

- برای یک گراف بدون جهت و متصل درخت پوشا درختی است که :
  - حاوی همه گره ها و زیرمجموعه ای از یالهای گراف است
  - از هر گره به هر گره دیگر مسیر ساده ای دارد.



درخت پوشای کمینه

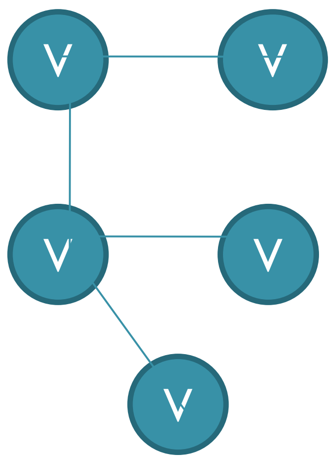
- در یک گراف بدون جهت وزن دار با یالهای نامنفی، درخت پوشایی است که جمع وزن یالهای آن کمترین است
- یک گراف ممکن است بیش از یک درخت پوشای مینیمال داشته باشد

الگوریتم پریم

تا زمانی که V≠Y

۱)نزدیکترین گره از V-Y به Y را انتخاب کن و به Y اضافه کن.

۲)لبه متناظربا این گره و گره همسایه ان در Y را به F اضافه کن.

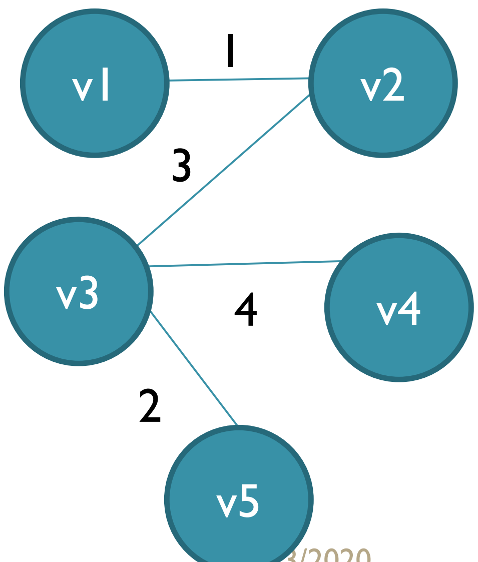


الگوریتم کراسکال

به صورت nمجموعه ی مجزا میبینم.

بین هردومجموعه کمترین هزینه را انتخاب میکند.

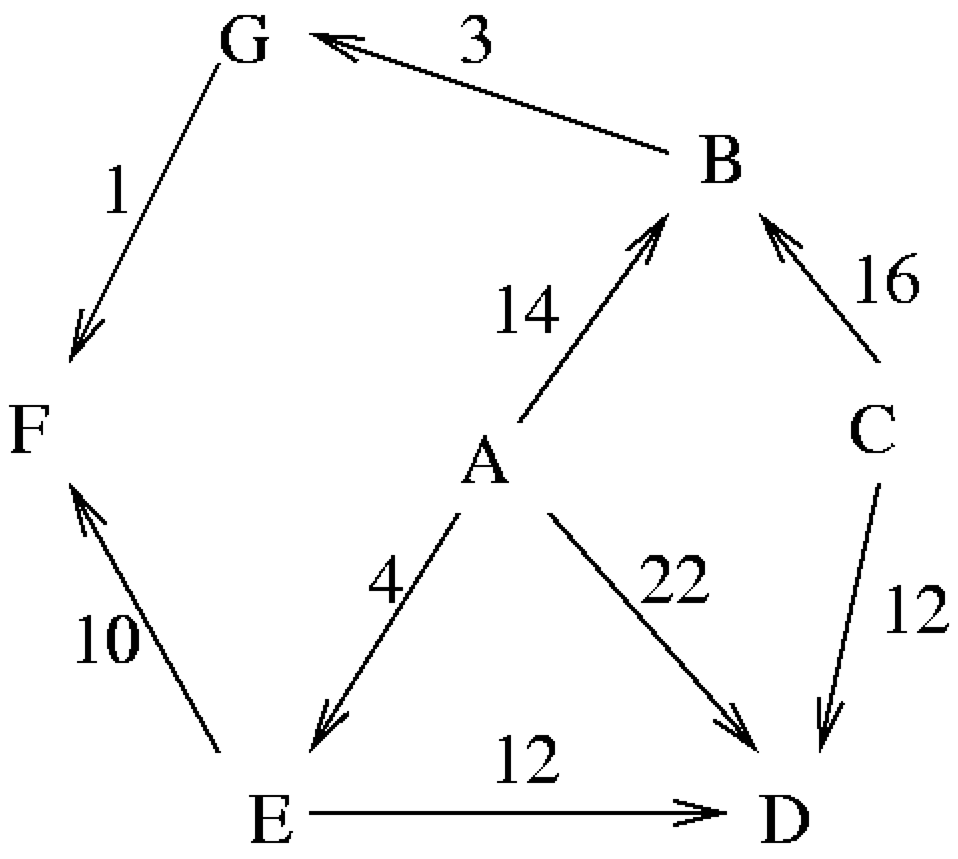
در هر قدم لبه ای را درمجموعه انتها با هم mergeمیکنیم وبه شرطی که منجر به cycleنشود.



روش حریصانه برای یافتن

کوتاهترین مسیر بین گره های گراف «الگوریتم دایکسترا»

- گراف جهت دار با یالهای نامنفی



الگوریتم دایکسترا

با شروع از گره V_1 یعنی $Y=\{V_1\}$

۱) از $V-Y$ گره ی V_i را طوری انتخاب می کنید که مسیر آن از v_1 (p_{i1}) با استفاده از گره های موجود در $v-y$ کوتاهترین باشد.

۲) گره فوق را به y اضافه کنید و لبه متناظر آن را به مسیر (p_{1i}) اضافه کنید.

```
Lenght  [2...n] of doable
Previous [2..n]
F=0
For k=2 to n
        lenght[k]=w[1,k]
        previous[k]=1
End for//init
For k=1 to n-1
```

روش حریصانه برای

زمان بندی بهینه کارها

زمان بندی کارها

- چندین کار با نیازمندی زمان ها و همچنین محدودیت های معین داده شده اند

- هدف : انتخاب مجموعه بهینه ای از این کارها تحت معیار خاص
  - دارای بیشترین سود
  - دارای بیشترین تعداد کارها
  - دارای کمترین زمان تاخیر
  - ...

زمان بندی کارها

مجموعه ای از کارها را پیدا کنیم که بتوان انها را با هم انجام داد وسود حداکثر را به ما بدهد.

service	1	3	2	5	7	1	4	5
waite	0	1	4	6	11	18	19	23
	1	4	6	11	18	19	23	28

جای ۴و۵راعوض میکنیم.
مینیمم زمان همه ی jobها زمانی بدست می اید که به صورت صعودی مرتب شده باشند.

1	1	2	3	4	5	5	7
0	1	2	4	7	11	16	21
1	2	4	7	11	16	21	28
1	2	4	7	12	16	21	28

job	dead	profit
1	2	30
2	1	25
3	2	25
4	1	40

برای به دست آوردن حداکثر سود:

```
1)Sort jobs by decreasing order of<profit>
2)S=0
3)For k=1 to n
   If feasible(S U job_k)
          S= S U job_k
     End if
End for
```

پس از sortکردن:

job	Dead line	Profit
4	1	40
1	2	30
2	1	25
3	2	25

فرض استقرا:

الگوریتم حریصانه برای job1,job2,job3,..n

مجموعه بهینه را تولید میکند.

حکم استفرا:

الگوریتم برای n+۱ کار هم مجموعه بهینه را تولید میکند.

اثبات:

فرض :Bمجموعه بهینه برای n+1 کار یعنی

$B^* \subset \{job1, job2, ..., jobn, jobn+1\}$

پاسخ الگوریتم حریصانه A هم:

$A^* \subset \{job1, job2, ..., jobn+1\}$

ثابت کنیدA هم بهینه است.

:Case1

job n+1 $\in$ B

$B \subset \{job_1, job_2, \ldots, job_n, job_{n+1}\}$

$Profit(B) \leq profit(A)$

case2-1 :

Jobn+1 $\in$ B

Jobn+1 $\in$ A

$A=A' \cup \{job_{n+1}\}$

$B= B' \cup \{job_{n+1}\}$

$Profit(B)=profit(B') +profit(job_{n+1})\leq profit(B')+profit(job_{n+1}) \leq profit(A)$

Case2-2

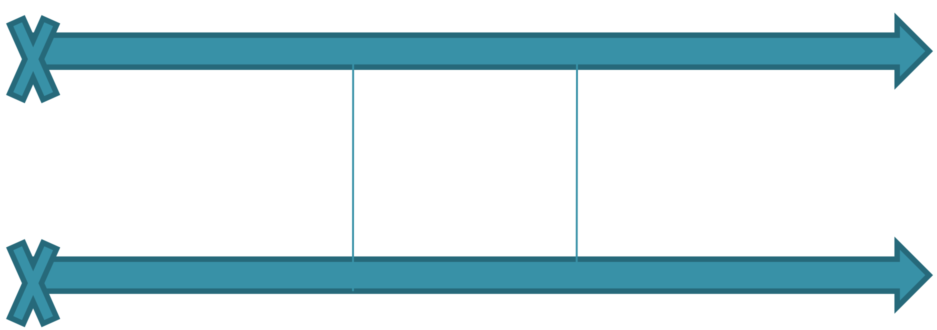
$Job_{n+1} \in B$

$Job_{n+1} \in A$

$Profit(j\ \ ) \geq profit(job_{n+1})$

job i

A



B

job n+1

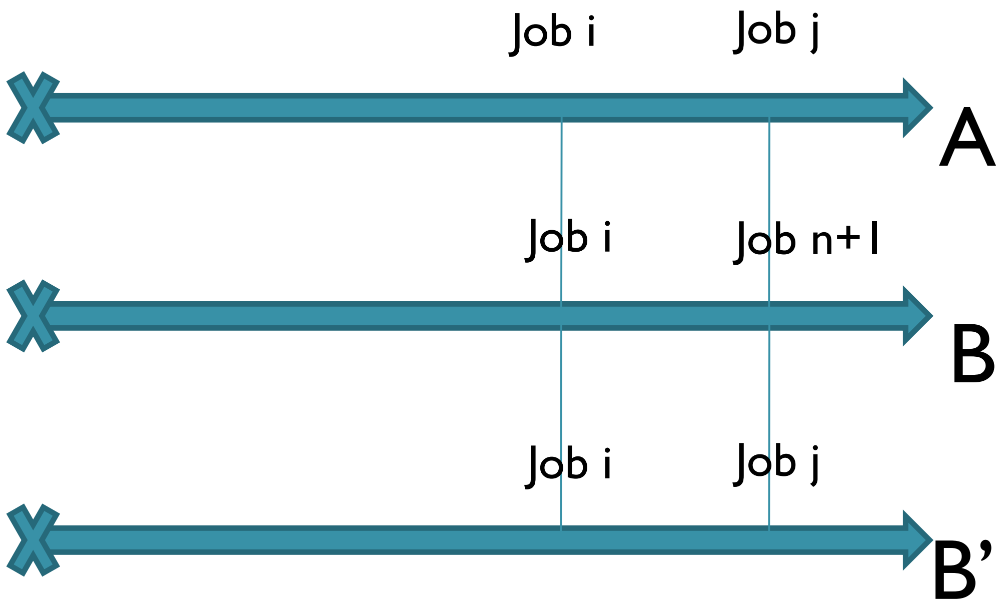
Case 2-2-1 :

job $i \in B$

$B' = \{B - \{job_{n+1}\}\} \cup \{job\ i\}$

profit (A)=profit (B')>=profit (B)

Case 2-2-2 :
profit (job j)>=profit (job n+1)
profit (A)=profit(B')>=profit(B)



قضیه:

فرض کنید مجموعه کار را به ترتیب صعودی زمان پایان مرتب شده است.آنگاه نخستین کار عضو مجموعه پاسخ بهینه است.

$Job_1, job_2, job_3, \ldots, job_n$

$f_1 \leq f_2 \leq f_3 \leq \ldots \leq f_n$

Job_1یکی از جواب های بهینه است.

فشرده سازی داده ها

الگوریتم هافمن
THE HUFFMAN ENCODING

فشرده سازی داده ها

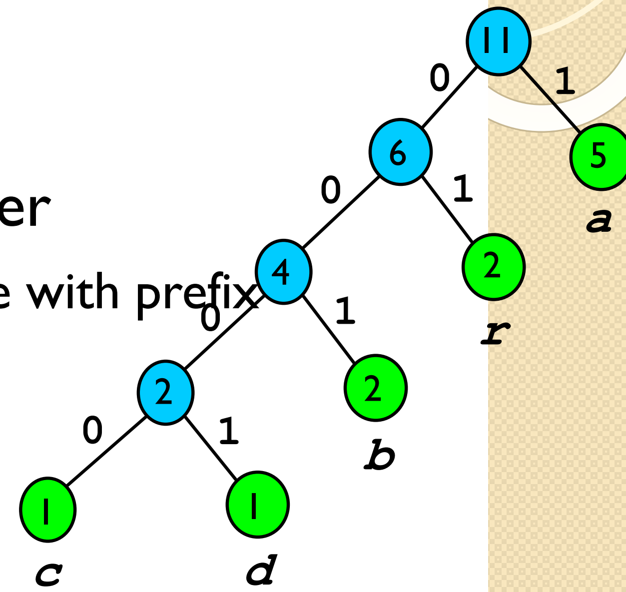
- *Data compression* problem – strings *S* and *S'*:
  - *S* -> *S'* -> *S*, such that |*S'*|<|*S*|
- Text compression by coding with *variable-length* code:
  - Obvious idea – assign short codes to frequent characters: "**abracadabra**"
  - Frequency table:

	a	b	c	d	r
Frequency	5	2	1	1	2
Fixed-length code	000	001	010	011	100
Variable-length code	1	001	0000	0001	01

  - How much do we save in this case?

Prefix Code کد پیشوندی

- Optimal code for given frequencies:
  - Achieves the minimal length of the coded text
- *Prefix code*: no codeword is prefix of another
  - It can be shown that optimal coding can be done with prefix code



- We can store all codewords in a *binary trie* – very easy to decode
  - Coded characters in leaves
  - Each node contains the sum of the frequencies of all descendants

کد بهینه

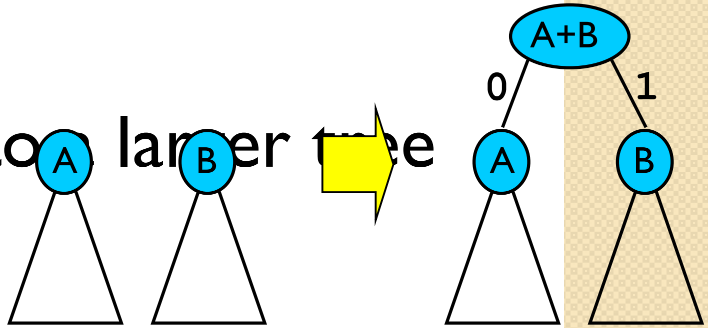
- The *cost* of the coding trie *T*:

$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

  - *C* – the alphabet,
  - $f(c)$ – frequency of character *c*,
  - $d_T(c)$ – depth of *c* in the trie (length of code in bits)
- Optimal trie – the one that minimizes $B(T)$
- Observation – optimal trie is always full:
  - Every non-leaf node has two children. Why?

الگوریتم هافمن

- Huffman algorithm, builds the code trie bottom up. Consider a forest of trees:
  - Initially – one separate node for each character.
  - In each step – join two trees into a larger tree



  - Repeat this until one tree (trie) remains.
  - Which trees to join? Greedy choice – the trees with the **smallest** frequencies!

الگوریتم هافمن

```
Huffman(C)
Q.build(C) // Builds a min-priority queue on frequences
for i ← 1 to n-1 do
    Allocate new node z
    x ← Q.extractMin()
    y ← Q.extractMin()
    z.setLeft(x)
    z.setRight(y)
    z.setF(x.f() + y.f())
    Q.insert(z)
return Q.extractMin() // Return the root of the trie
```

- What is its running time?

درستی الگوریتم هافمن

- Greedy choice property:
  - Let *x*, *y* – two characters with lowest frequencies. Then there exists an optimal prefix code where codewords for *x* and *y* have the same length and differ only in the last bit
  - Let's prove it:
    - Transform an optimal tree *T* into one (*T''*), where *x* and *y* are max-depth siblings. Compare the costs.

درستی الگوریتم هافمن

- Optimal sub-structure property:
  - Let x, y – characters with minimum frequency
  - $C' = C - \{x,y\} \cup \{z\}$, such that $f(z) = f(x) + f(y)$
  - Let T' be an optimal code tree for C'
  - Replace leaf z in T' with internal node with two children x and y
  - The result tree T is an optimal code tree for C
- Proof a little bit more involved than a simple "cut-and-paste" argument

تمرین:

برنامه ای به زبان c++ بنویسید که عددصحیح مثبتn نشان دهنده تعدادjob ها ومشخصات شروع و پایان n تا job را از ورودی بخواند و با استفاده از الگوریتم فوق مجموعه بهینه پاسخ را تولید کرده و در

فایل خروجی بنویسید.