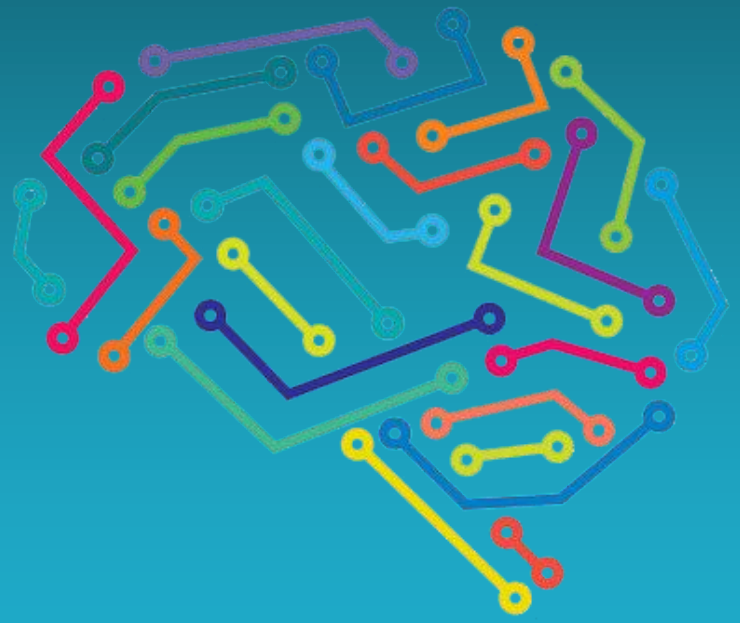
دانشگاه صنعتی شاهرود

حل مساله با جستجو



درس هوش مصنوعی

نیم‌سال اول تحصیلی 98-1397

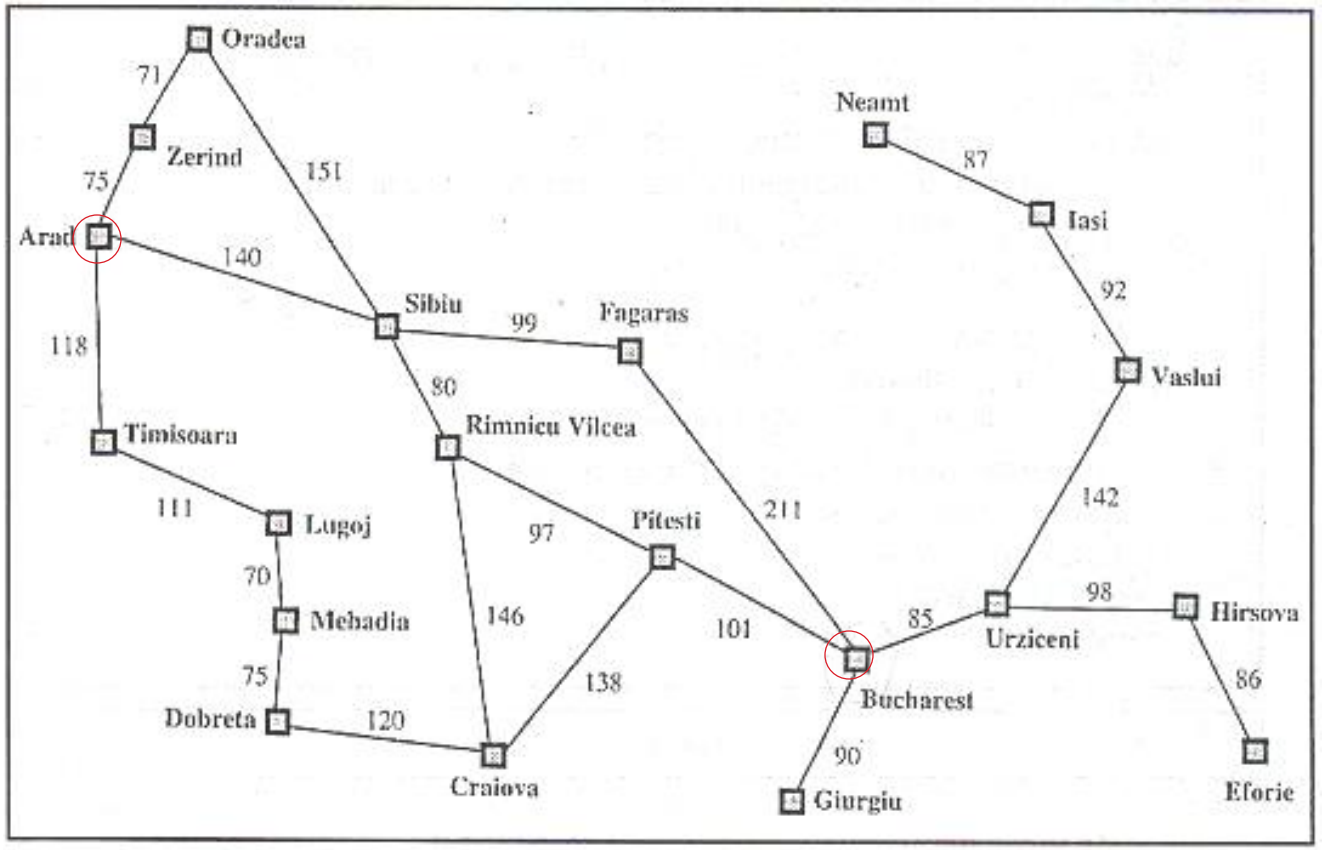
جستجوی آگاهانه

- از دانش موجود در مسئله برای یافتن سریعتر جواب استفاده می‌کنند.
- به صورت کلی می‌توان آنها را الگوریتم‌های جستجوی اول-بهترین نامید.
- در جستجوی اول-بهترین، به هر نود یک مقدار بر اساس تابع ارزیاب $f(x)$ نسبت داده می‌شود که نماینده هرینه آن نود است.
- نودها بر اساس مقدار این تابع برای بسط انتخاب می‌شوند.
- معادل جستجوی uniform-cost است با این تفاوت که بجای تابع $g(x)$ (هزینه رسیدن از نود شروع به نود جاری) از $f(x)$ استفاده می‌کند.
- در بیشتر الگوریتم‌های این خانواده، بخشی از تابع ارزیاب را تابع اکتشافی (heuristic function) $h(x)$ تشکیل می‌دهد:
  - ✓ این تابع تخمینی از هزینه ارزانترین (کوتاهترین) مسیر از نود جاری تا یک نود هدف را ارائه می‌دهد.
  - ✓ این توابع رایج‌ترین راه برای گنجاندن دانش اضافی ما از مسئله، در جستجو است.
  - ✓ این توابع، دلخواه، ویژه خود هر مسئله و غیرمنفی هستند.
  - ✓ اگر نود جاری، یک نود هدف باشد، آنگاه $h(x) = 0$

جستجوی اول‌بهترین حریصانه (greedy best-first search)

- در هر مرحله گرهی که نزدیکترین به هدف است انتخاب می‌شود با این دید که چنین روشی احتمالا زودتر به جواب می‌رسد.
- در این الگوریتم، $f(n) = h(n)$.
- به عنوان مثال برای مسئله مسیریابی در نقشه رومانی:
  - ✓ برای تابع اکتشافی h از "فاصله مستقیم" استفاده می‌کنیم.

مسئله پیدا کردن مسیر



شکل ۲-۳ نقشه ساده شده‌ای از جاده‌های رومانی.

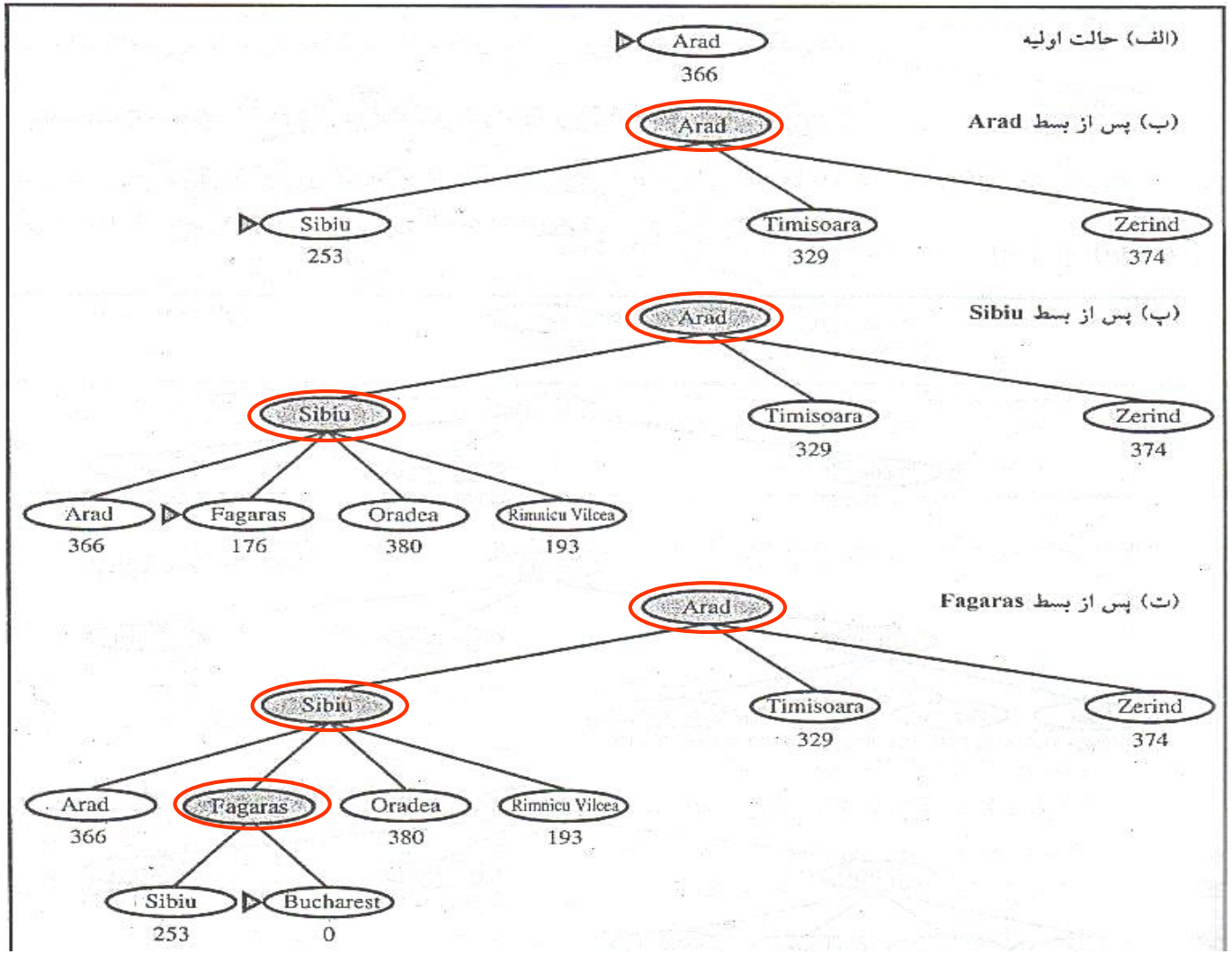
جستجوی اول‌بهترین حریصانه

- روش اکتشافی فاصله مستقیم (Straight line distance) h_{SLD}

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.

جستجوی اول‌بهترین حریصانه

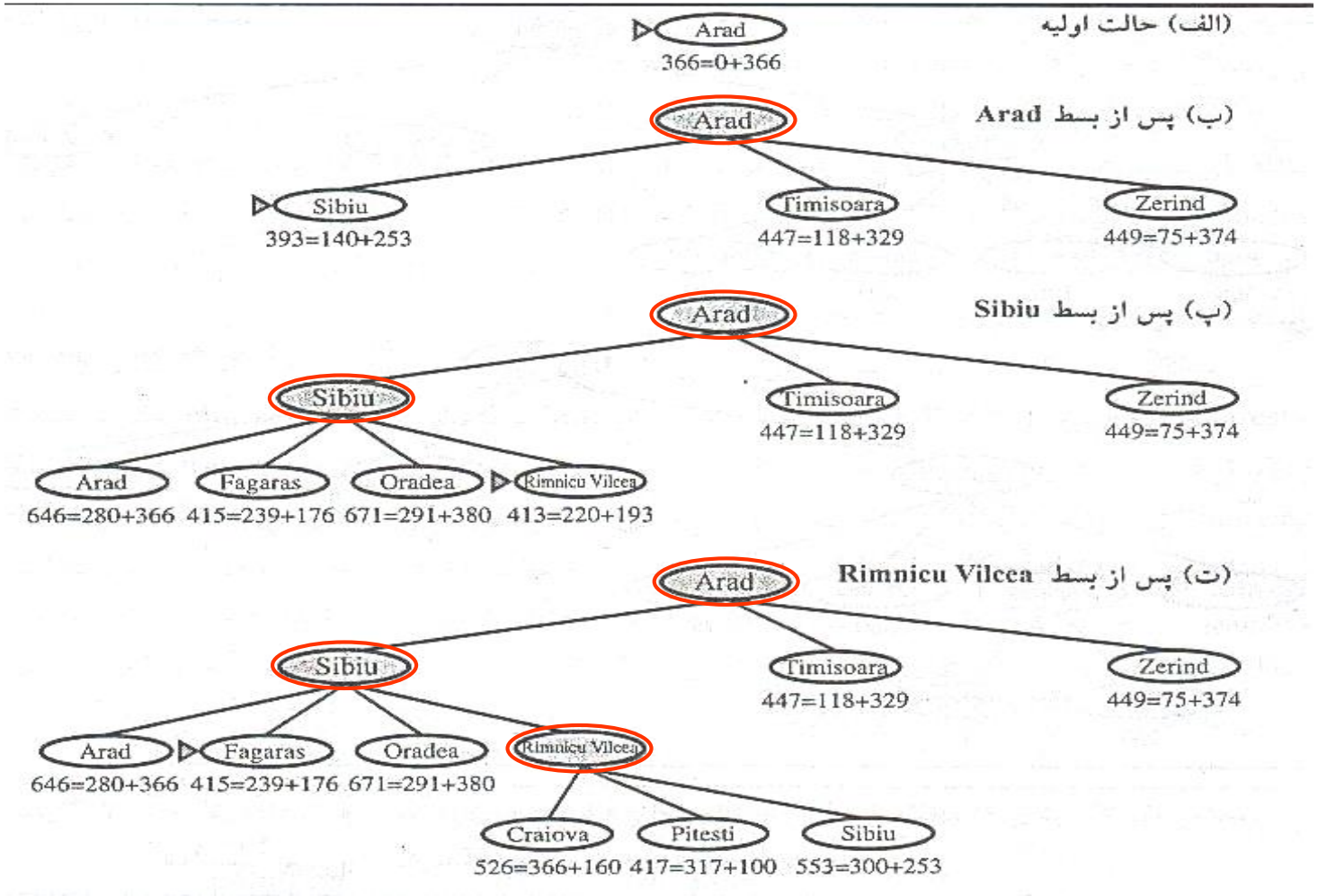


- همانطور که گفته شد، در هر مرحله نزدیکترین گره به هدف برای بسط انتخاب می‌شود.
- الگوریتم اول-بهترین حریصانه، در بدترین حالت، دارای پیچیدگی $O(b^m)$ هم در زمان و هم حافظه است.
  - ✓ انتخاب یک تابع اکتشافی مناسب می‌تواند این پیچیدگی را به شکل چشمگیری کاهش دهد.
- بهینه نیست.
- کامل نیست، حتی در فضاهای متناهی:
  - ✓ مسئله رفتن از Isai به Fagaras با این روش منتهی به تکرار بینهایت انتخاب متوالی Neamt و Isai می‌شود.

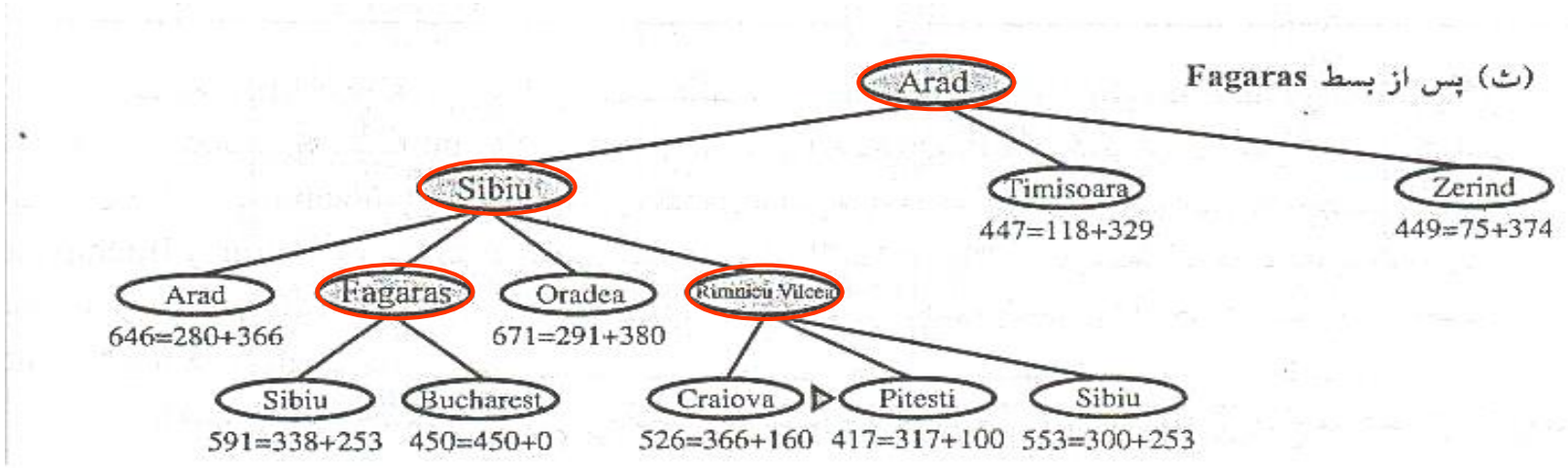
جستجوی A^*

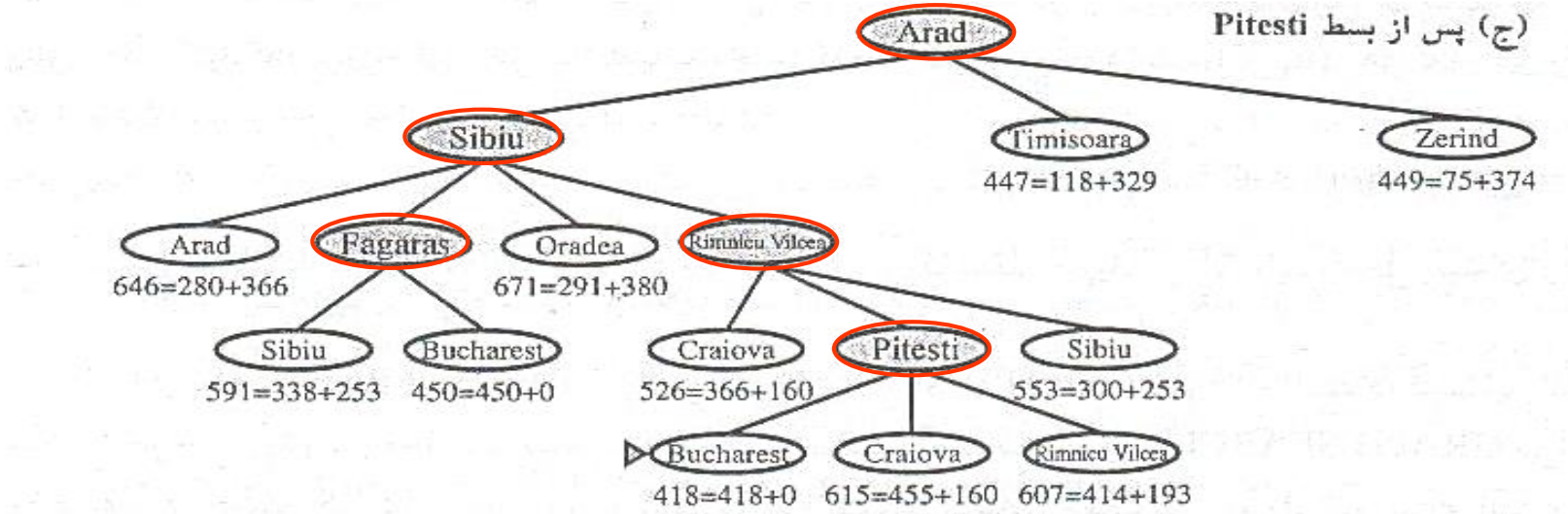
- مشهورترین جستجوی اول-بهترین است.
- هر نود را بر مبنای طول مسیر از شروع به آن نود (تابع g) و یک تابع اکتشافی ارزیابی می‌کند، یعنی:
  $f(n) = h(n) + g(n)$ ✓
- بنابراین تابع ارزیاب تخمینی از کوتاهترین فاصله از نود شروع تا یک نود هدف است.
- کاملا مشابه uniform-cost BFS است با این تفاوت که بجای $g(n)$ از $h(n) + g(n)$ استفاده می‌کند.
- تحت شرایط خاصی می‌تواند بهینه و کامل باشد.

جستجوی A^*


(الف) حالت اولیه
Arad
366=0+366
(ب) پس از بسط Arad
Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374
(پ) پس از بسط Sibiu
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
413=220+193
(ت) پس از بسط Rimnicu Vilcea
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
Fagaras
Oradea
Rimnicu Vilcea
646=280+366
415=239+176
671=291+380
Craiova
Pitesti
Sibiu
526=366+160
417=317+100
553=300+253

جستجوی A^*

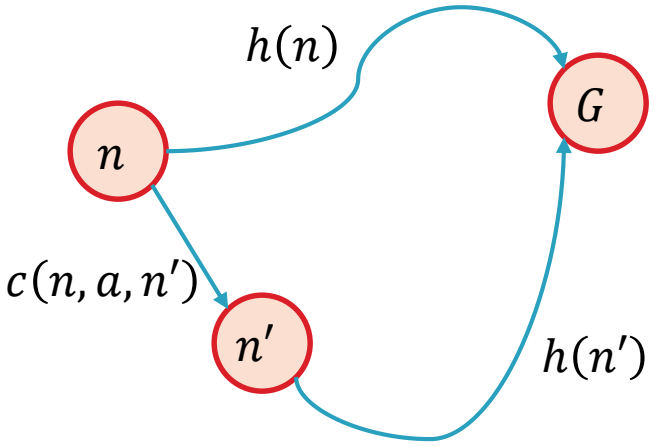

(ث) پس از بسط Fagaras
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253


(ج) پس از بسط Pitesti
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

جستجوی A^*

- امیدبخش(admissible) بودن تابع اکتشافی:
  - ✓ هزینه تخمین‌زده شده برای مسیر توسط این تابع نباید هرگز بیشتر از هزینه واقعی باشد.
- سازگاری (consistency) یا یکنوایی (monotonicity) تابع اکتشافی
  - ✓ برای هر گره n و هر فرزند n' آن (به ازای هر عمل a که به فرزند بعدی برسیم) باید داشته باشیم:

$$h(n) \leq \mathrm{c}(n, a, n') + \mathrm{h}(n')$$



- نامساوی مثلثی بین سه نود هدف (G)، نود جاری (n) و هر فرزند آن مانند n'
  - ✓ هر تابع سازگار، امیدبخش نیز هست.
  - ✓ ممکن است در موارد اندکی، یک گره امیدبخش، سازگار نباشد.
  - ✓ پس سازگاری کمی سختگیرانه‌تر از امیدبخشی است.

جستجوی A^*

- بهینگی A^*:
  - ✓ نسخه درختی آن در صورتی بهینه است که تابع اکتشافی امید بخش باشد.
  - ✓ نسخه گرافی در صورتی بهینه است که تابع اکتشافی سازگار باشد.
- اثبات:
  - ✓ اگر h(n) سازگار باشد مقادیر f(n) در امتداد هر مسیری غیرکاهشی هستند.
  - ✓ هرگاه گرهی برای بسط انتخاب می‌شود، مسیر بهینه برای آن گره پیدا شده است. اگر اینطور نباشد، باید نودی مانند n' در frontier موجود باشد که گذر از آن بتواند مسیر یافت‌شده را کوتاهتر کند. ولی از آنجا که مقادیر $f(n)$ در امتداد هر مسیری غیرکاهشی هستند، مقدار f برای نود n' یعنی $f(n')$ از مقدار آن برای نود n یعنی $f(n)$ بزرگتر است، پس اگر چنین نودی موجود باشد باید قبل از نود n برای بسط انتخاب شده باشد.
  - ✓ درنتیجه وقتی گره هدف برای بسط انتخاب می‌شود، کوتاهترین مسیر برای رسیدن به آن پیدا شده است و هر مسیر دیگری بلندتر خواهد بود.

جستجوی A^*

- چند نکته در باره الگوریتم جستجوي A^*:
  - ✓ با فرض آنکه هزینه مسیر بهینه C^* باشد:
    - الگوریتم تمامی گره‌هایی با $f(n) < C^*$ را بسط می‌دهد.
    - ممکن است برخی گره‌ها با $f(n) = C^*$ را نیز قبل از رسیدن به هدف بسط دهد.
    - هیچ گرهی با $f(n) > C^*$ بسط داده نمی‌شوند، یا به عبارت دیگر زیردرختهای متناظر با گرهایی با $f(n) > C^*$ هرس (prune) می‌شوند.
    - به همین دلیل کامل است به شرط اینکه نودی با وزن منفی نداشته باشیم و b متناهی باشد.
    - اولین باری که گره هدف برای بسط انتخاب شود، مسیر بهنه برای رسیدن به آن مشخص شده است.
  - ✓ در میان تمامی الگوریتم‌هایی که از تابع شهودی استفاده می‌کنند، A^* کمترین گره را گسترش می‌دهد.

جستجوی A^*

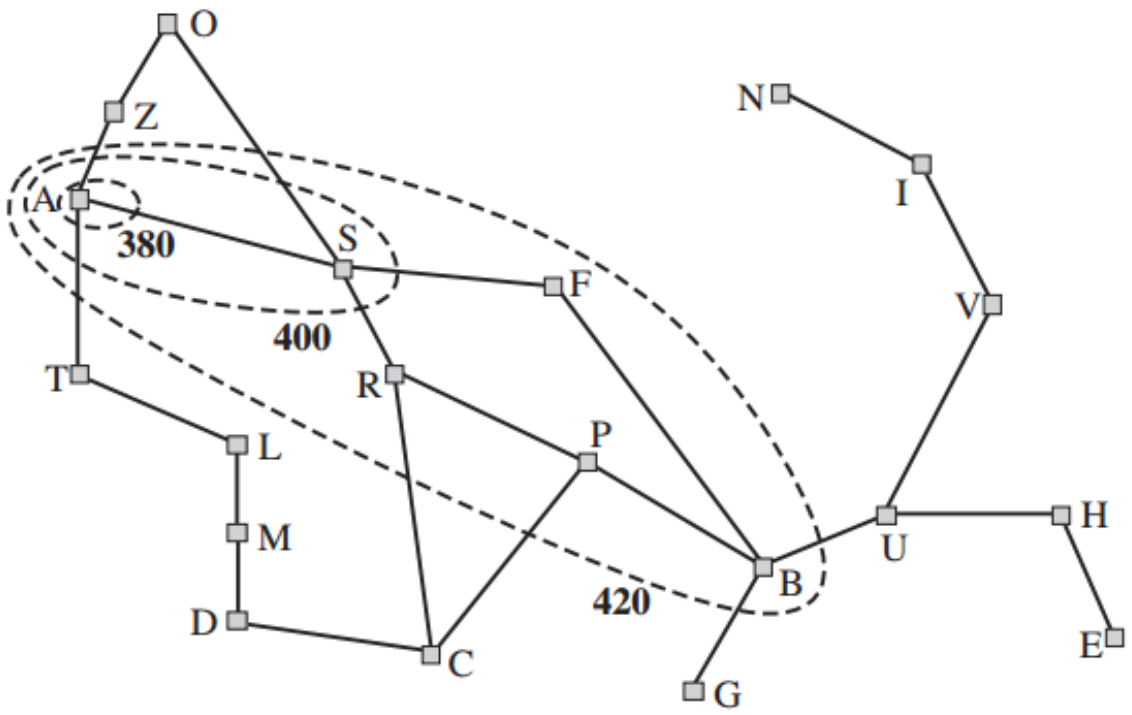


Figure 3.25 Map of Romania showing contours at $f = 380$, $f = 400$, and $f = 420$, with Arad as the start state. Nodes inside a given contour have f-costs less than or equal to the contour value.

جستجوی A^*

- لینک انیمیشن:
  - ✓ https://commons.wikimedia.org/wiki/File:Astar_progress_animation.gif

جستجوی A^*

- پیچیدگی جستجوي A^* وابسته به تابع اکتشافی است.
- در عمل، گاهی، به دلیل پیچیدگی بالای الگوریتم، راه‌حل‌های suboptimal که یافتن آنها پیچیدگی کمتری دارد ترجیح داده می‌شوند.
- پیچیدگی حافظه معمولا مشکل جدی‌تری در مقابل پیچیدگی زمانی جستجوي A^* است.
  - ✓ همه گره‌های تولیدشده در حافظه نگه داشته می‌شوند.
- روشهای پیشنهادی برای حل مشکل حافظه:
  - ✓ Iterative Deepening A^* (IDA^*)
  - ✓ Recursive Best-First Search (RBFS)
  - ✓ Memory Bounded A^* (MA^*)
  - ✓ Simple MA^* (SMA^*)

جستجوی IDA*

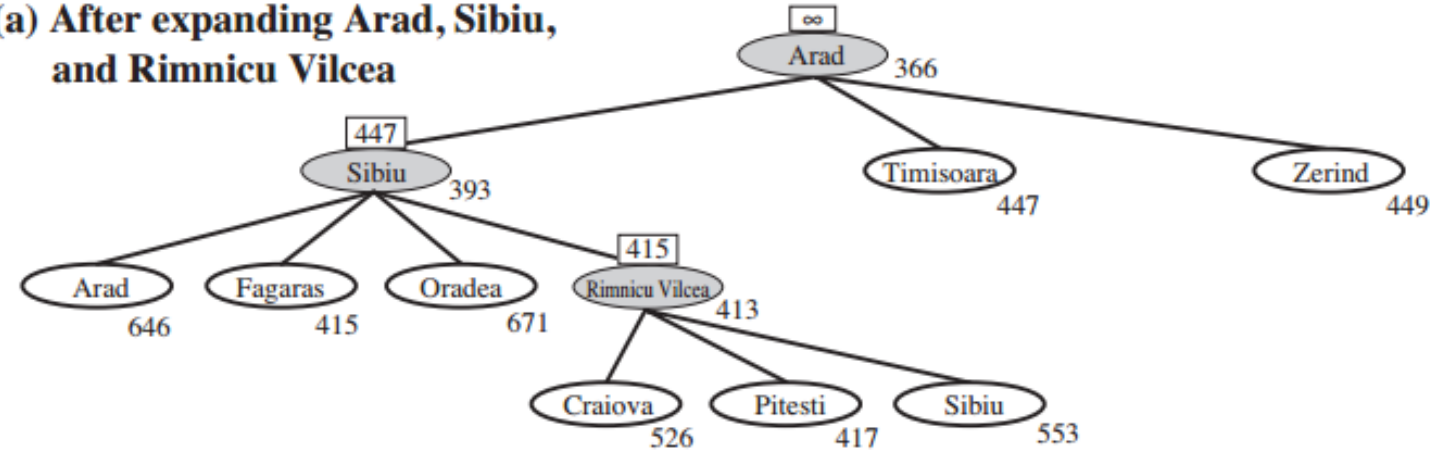
- جستجوی A* با افزایش عمق تکراری (IDA*)
  - ✓ مبتنی بر جستجوی اول-عمق است.
  - ✓ جستجوی درختی است.
  - ✓ مانند اول عمق در حالت عمیق‌شونده تکراری، عمق جستجو محدود است و گام به گام زیاد می‌شود.
  - ✓ البته در اینجا بجای عمق، یک مقدار cut-off بر مبنای مقدار f یعنی $h + g$ تعریف و گام به گام زیاد می‌شود.
    - یعنی گره‌هایی با مقدار f بیشتر از cut-off بسط داده نمی‌شوند.
    - مقدار cut-off در هر تکرار، برابر مقدار کمترین f در تکرار قبل که از مقدرا cut-off آن مرحله بیشتر بوده است در نظر گرفته می‌شود.
  - ✓ در حالتی که هزینه‌ها مقادیر حقیقی باشند ممکن است با مشکل مواجه شود.

جستجوی RBFS

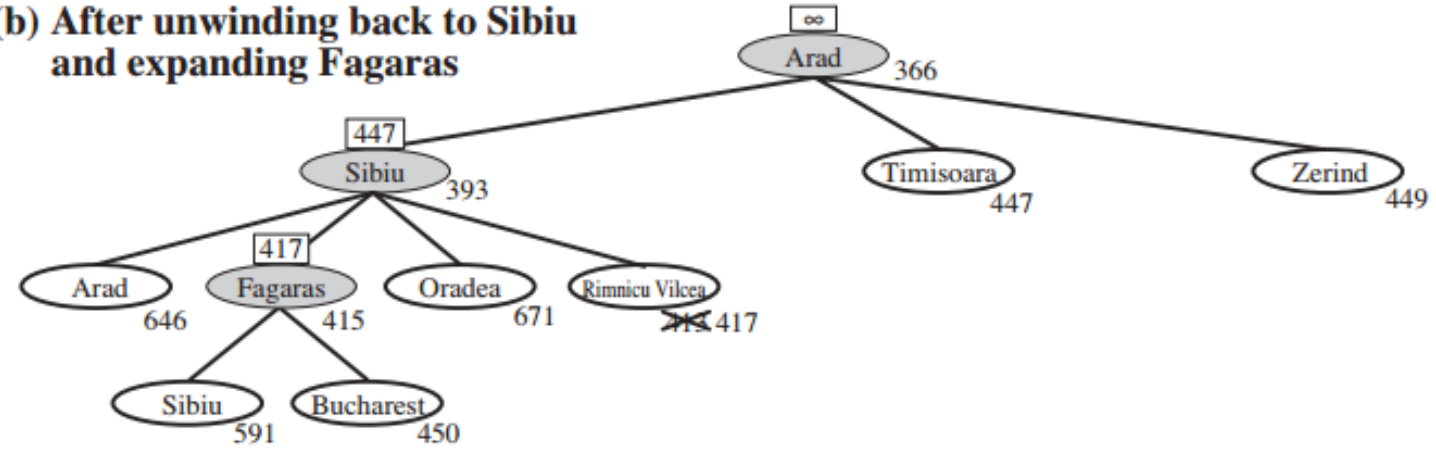
- اول-بهترین بازگشتی (RBFS):
  - ✓ پیچیدگی حافظه‌ای خطی دارد.
  - ✓ جستجوی درختی است.
  - ✓ بر مبنای جستجوی اول-بهترین است.
  - ✓ ساختار آن شبیه جستجوی اول-عمق بازگشتی است.
  - ✓ یعنی در یک مسیر عمیق می‌شود اما تاجایی که از مقدار f-limit رد نشده باشد که هزینه بهترین مسیر جایگزین را در آن نگهداری می‌کند.
  - ✓ در صورتی که هزینه مسیر جاری بیشتر از یکی از مسیرهای جایگزین (f-limit) باشد به عقب برگشته و آن مسیر را ادامه می‌دهد.
  - ✓ در بازگشت، هزینه زیردرختهای حذف‌شده را بر اساس مسیر پیموده‌شده به‌روز می‌کند.
  - ✓ یکی از اشکالات آن بازتولید مکرر برخی گره‌ها است.

RBFS

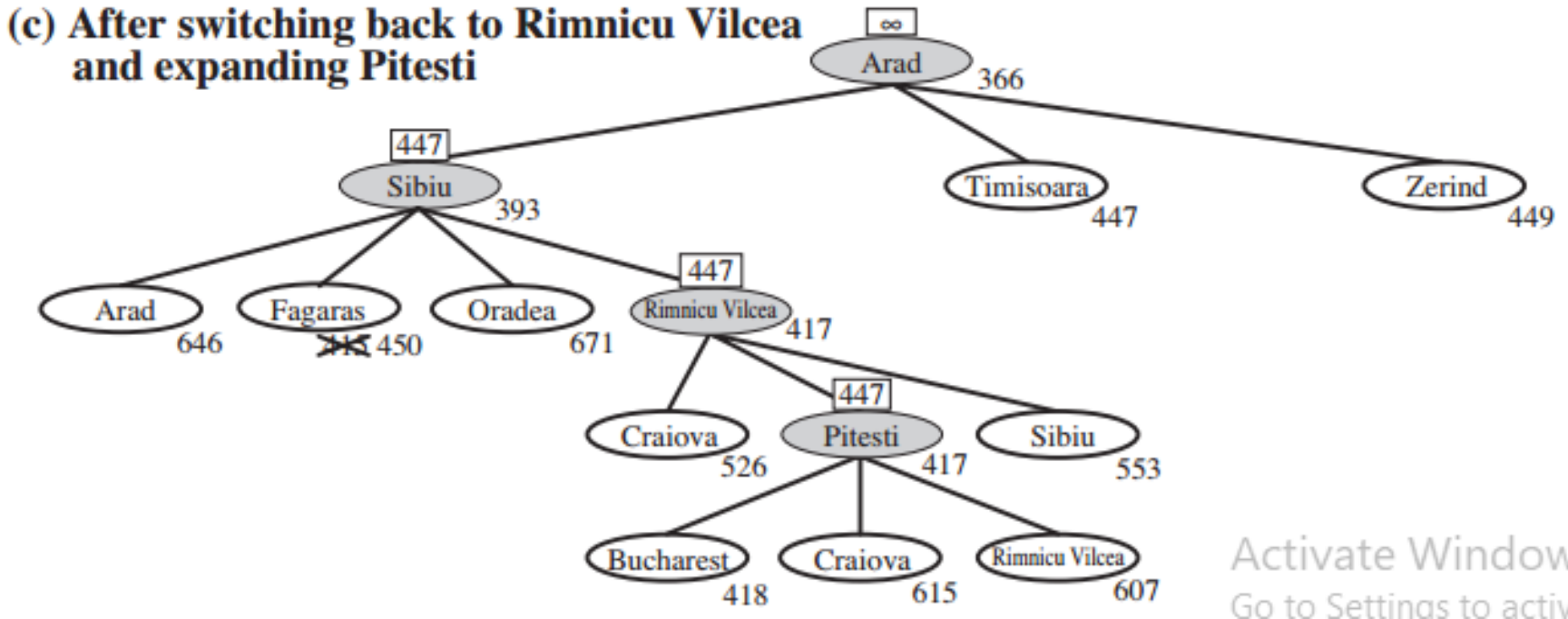
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras



RBFS

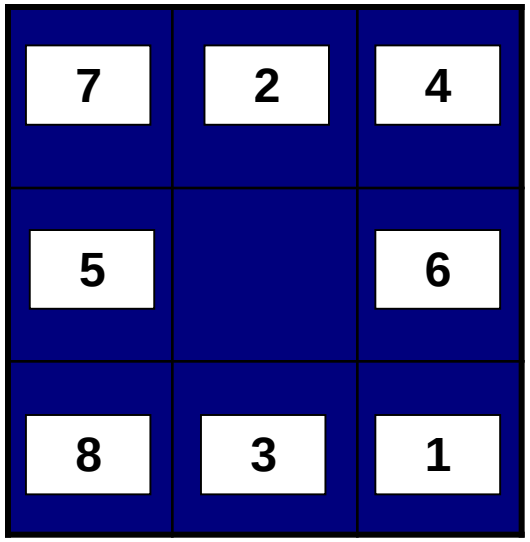


جستجوی A^*

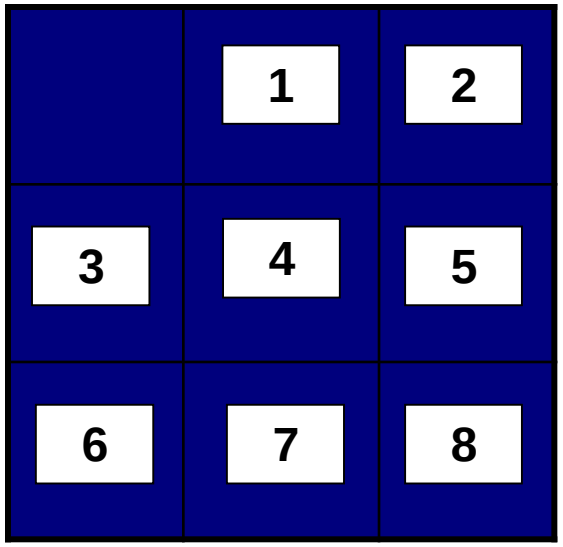
- هردو الگوریتم فوق از بخش کوچکی از حافظه موجود استفاده می‌نمایند.
- الگوریتم‌های MA^* و SMA^* طراحی شده‌اند تا استفاده منطقی‌تری (بیشترین ممکن) از حافظه داشته باشند.
- این امر باعث کاهش تولید تکراری گره‌ها شده و سرعت را افزایش خواهد داد.
- در الگوریتم SMA^*:
  - ✓ تا جایی که جایی که حافظه جا داشته باشد، گره‌ها در حافظه نگه داشته می‌شوند.
  - ✓ پس از آن برای نگهداری گره‌های بعدی، بدترین گره‌های موجود حذف می‌شوند.

توابع اکتشافی

- می‌خواهیم می بیشتر درباره توابع اکتشافی صحبت کنیم.
- برای این منظور مسئله معمای ۸ را در نظر گرفته‌ایم.
- معمایی با حالت اولیه شکل را در نظر بگیرید:
  - ✓ هدف: لغزاندن چهارخانه‌ها به طور افقی یا عمودی به طرف فضای خالی است تا زمانی که ساختار کلی مطابق با هدف (goal) باشد. فاکتور انشعاب در حدود ۳ (میانگین حالات ممکن با وزن احتمال رخدادشان) است.



Start State



Goal State

Relaxed problems

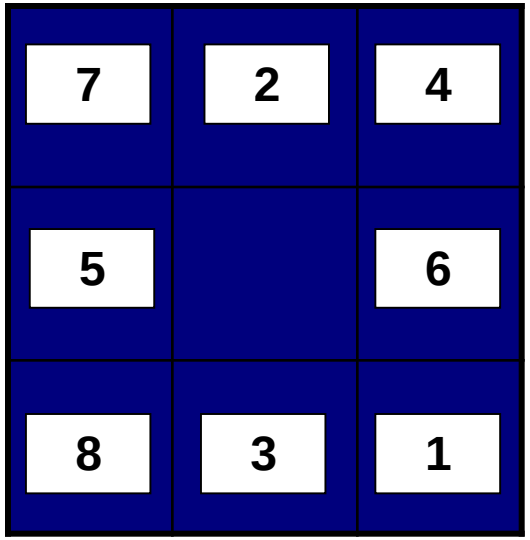
- مسئله‌ای با محدودیت‌های کمتر از مسئله اصلی

- For example, if the 8-puzzle actions are described as
  - ✓ A tile can move from square A to square B if
    - A is horizontally or vertically adjacent to B **and** B is blank
- we can generate three relaxed problems by removing one or both of the conditions:
  - ✓ A tile can move from square A to square B if A is adjacent to B.
  - ✓ A tile can move from square A to square B if B is blank.
  - ✓ A tile can move from square A to square B.

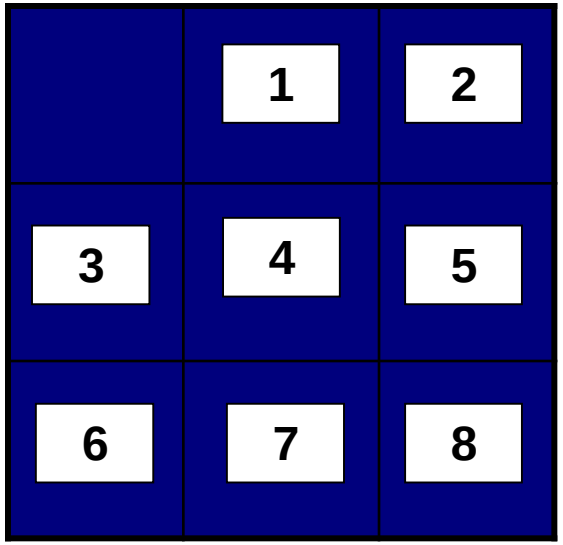
توابع اکتشافی معمای ۸

- مثلا:
  - ✓ مقدار h_1 برابر تعداد چهارخانه‌هایی که در مکان‌های نادرست هستند.
    - این تابع (h_1) امیدبخش است، زیرا واضح است که هر مهره که خارج از مکان درست باشد حداقل یک بار باید جابجا شود.
  - ✓ مقدار h_2 برابر مجموع فواصل مهره‌ها از مکان‌های هدف صحیحشان است.
    - مجموع فواصل عمودی و افقی است که بعضی وقتها city block distance و یا Manhattan distance نامیده می‌شود.
    - این تابع (h_2) قابل قبول است، زیرا هر جابجایی که می تواند انجام پذیرد یک مرحله به هدف نزدیک می‌شود.

توابع اکتشافی



Start State



Goal State

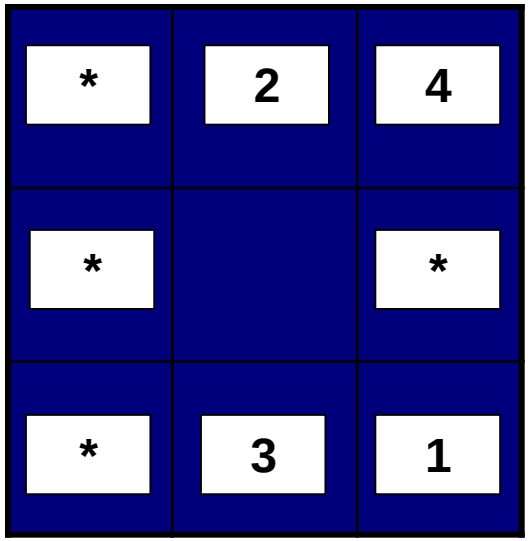
$h_1 = 8$

$h_2 = 18$

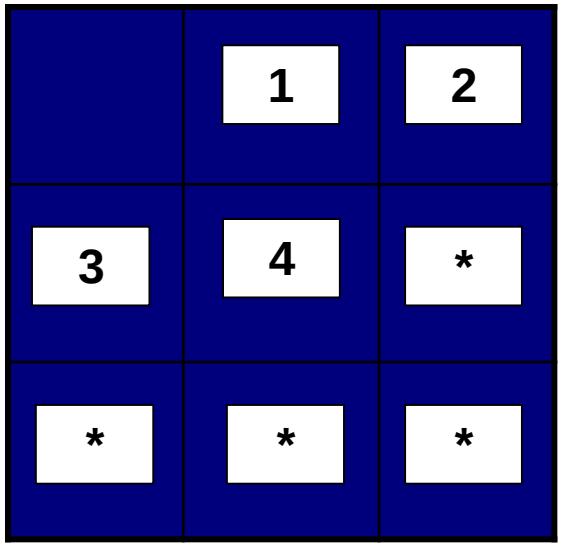
(3+1+2+2+2+3+3+2)

Pattern Data Bases

- در این روش، با مفهومی به اسم زیرمسئله (sub-problem) روبرو هستیم.
- شکل زیر، نمونه‌ای از یک زیرمسئله است:
  - ✓ به این معنی که ما تنها به موقعیت 4 مهره 1، 2، 3 و 4 توجه داریم.



Start State



Goal State

Pattern Data Bases

- در این روش، همه زیرمسائل ممکن (در مثال ما، تمامی چیدمان‌های ممکن از مهره‌های 1، 2، 3 و 4 و خانه خالی) در مسئله حل شده و در یک دیتابیس نگهداری می‌شوند.
- مقدار تابع هیوریستیک برای هر نود (حالت) در جستجو برابر هزینه حل آن است که قبلا محاسبه و در دیتابیس ذخیره شده است.
- خود دیتابیس با شروع از حالت هدف و بازگشت به عقب ساخته می‌شود مشابه dynamic programming.
- هرچند موقعیت سایر مهره‌ها اهمیتی ندارد، حرکت آنها در شمارش گام‌ها محاسبه می‌شود.
- انتخاب ترکیب 4 مهره دلخواه است: مثلا می‌تواند 5، 6، 7، 8 یا 1، 3، 6 و 7 باشد.

Other Admissible Heuristics

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

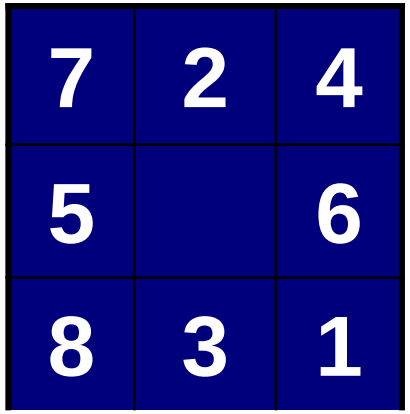
Goal State

h2 : Sum of Eucledian distances of the tiles from their goal positions

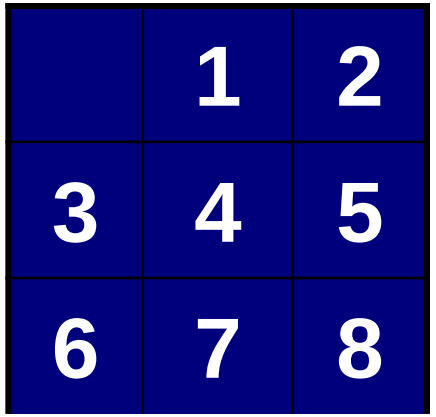
In the given figure, all the tiles are out of position, hence for this state, ***h2 = sqrt(5) + 1 + sqrt(2) + sqrt(2) + 2 + sqrt(5) + sqrt(5) + 2 = 14.53.***

h2 is an admissible heuristic, since in every move, one tile can only move closer to its goal by one step and the eucledian distance is never greater than the number of steps required to move a tile to its goal position.

Other Admissible Heuristics



Start State

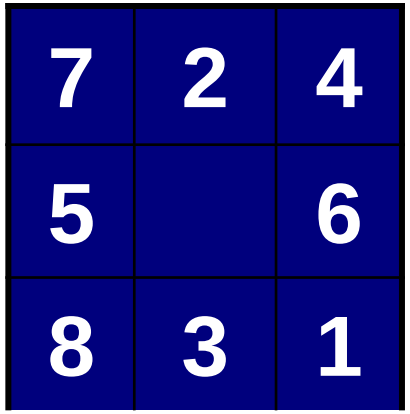


Goal State

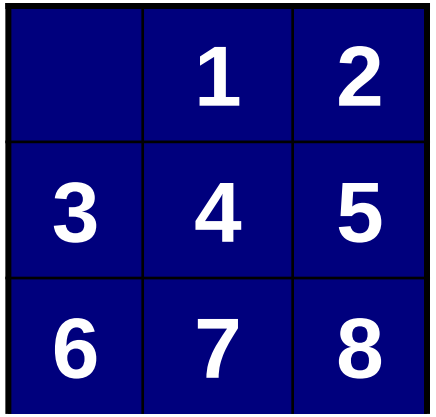
h4 : Number of tiles out of row + Number of tiles out of column

In the given figure, all the tiles are out of position, hence for this state,
h4 = 5 (out of row) + 8 (out of column) = 13.
h4 is an admissible heuristic, since every tile that is out of column or out of row must be moved at least once and every tile that is both out of column and out of row must be moved at least twice.

Other Admissible Heuristics



Start State

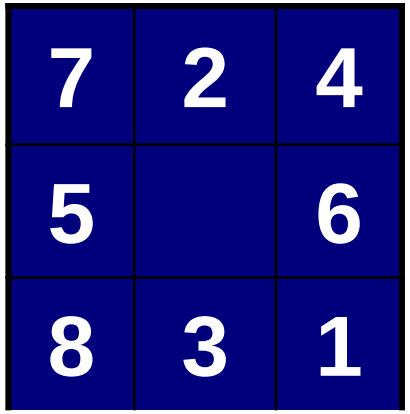


Goal State

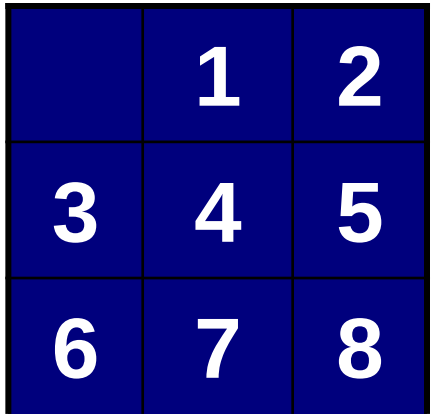
n-Max Swap

Assume you can swap any tile with the 'space'. Use the cost of the optimal solution to this problem as a heuristic for the 8-puzzle.

Other Admissible Heuristics



Start State



Goal State

n-Swap

Represent the 'space' as a tile and assume you can swap any two tiles. Use the cost of the optimal solution to this problem as a heuristic for the 8-puzzle.

A Non-Admissible Heuristic

7	2	4
5		6
8	3	1

Start State

1	2	3
8		4
7	6	5

Goal State

Nilsson's Sequence Score

$h(n) = P(n) + 3\,S(n)$

P(n) : Sum of Manhattan distances of each tile from its proper position
S(n) : A sequence score obtained by checking around the non-central squares in turn, allotting 2 for every tile not followed by its proper successor and 0 for every other tile, except that a piece in the center scores 1

Clockwise

Admissible Vs. Non-Admissible Heuristics

- When solving a problem, one can have two kinds of objectives in mind:
  - ✓ Minimize the path cost of the solution (i.e. find the optimal solution)
  - ✓ Minimize the time taken to find the solution
- Of-course, in most cases, both the objectives are important. A balance has to be struck between the two, and that is where the heuristic comes in.
- **An admissible heuristic is optimal, it will always find the cheapest path solution.**
- **On the other hand, a non-admissible heuristic is not optimal, it may result in a suboptimal solution, but may do so in a much shorter time than that taken by an admissible heuristic.**

Experimental results show that the Nilsson Sequence Score heuristic finds a solution to the 8-puzzle much faster than all the admissible heuristics.

کیفیت تابع اکتشافی

- یک راه برای تشخیص کیفیت تابع اکتشافی، فاکتور انشعاب مؤثر b^* است. اگر مجموع تعداد گره‌های بسط‌داده‌شده توسط A* برای یک مسئله N باشد و عمق راه حل d، سپس b^* فاکتور انشعابی است که یک درخت یکنواخت با عمق d خواهد داشت تا حاوی $N+1$ گره باشد. بنابراین:

$$N + 1 = 1 + b* + (b*)^2 \ldots + (b*)^d$$

- معمولاً فاکتور انشعاب مؤثر که توسط اکتشاف نمایش داده می‌شود، مقدار ثابتی دارد.
- / طراحی شده، b* در حدود ۱ دارد.

کیفیت تابع اکتشاف

- Experimental measurements of b^* on a small set of problems can provide a good guide to the heuristic's overall usefulness.
- A well designed heuristic would have a value of b^* close to 1.

To compare the admissible heuristics mentioned earlier (h1 to h4), one can generate a large number of initial states for the 8-puzzle and solve each one using all 4 heuristics. The number of nodes expanded and depth of solution can be recorded and b* values tabulated. Such a procedure would accurately reflect the relative quality of the heuristics.

مقایسه هزینه جستجو و فاکتور انشعاب موثر

	Search Cost		Effective Branching Factor	
d	A*(h_1)	A*(h_2)	A*(h_1)	A*(h_2)
2	6	6	1.79	1.79
4	13	12	1.48	1.45
6	20	18	1.34	1.30
8	39	25	1.33	1.24
10	93	39	1.38	1.22
12	227	73	1.42	1.24
14	539	113	1.44	1.23
16	1301	211	1.45	1.25
18	3056	363	1.46	1.26
20	7276	676	1.47	1.27
22	18094	1219	1.48	1.28
24	39135	1641	1.48	1.26

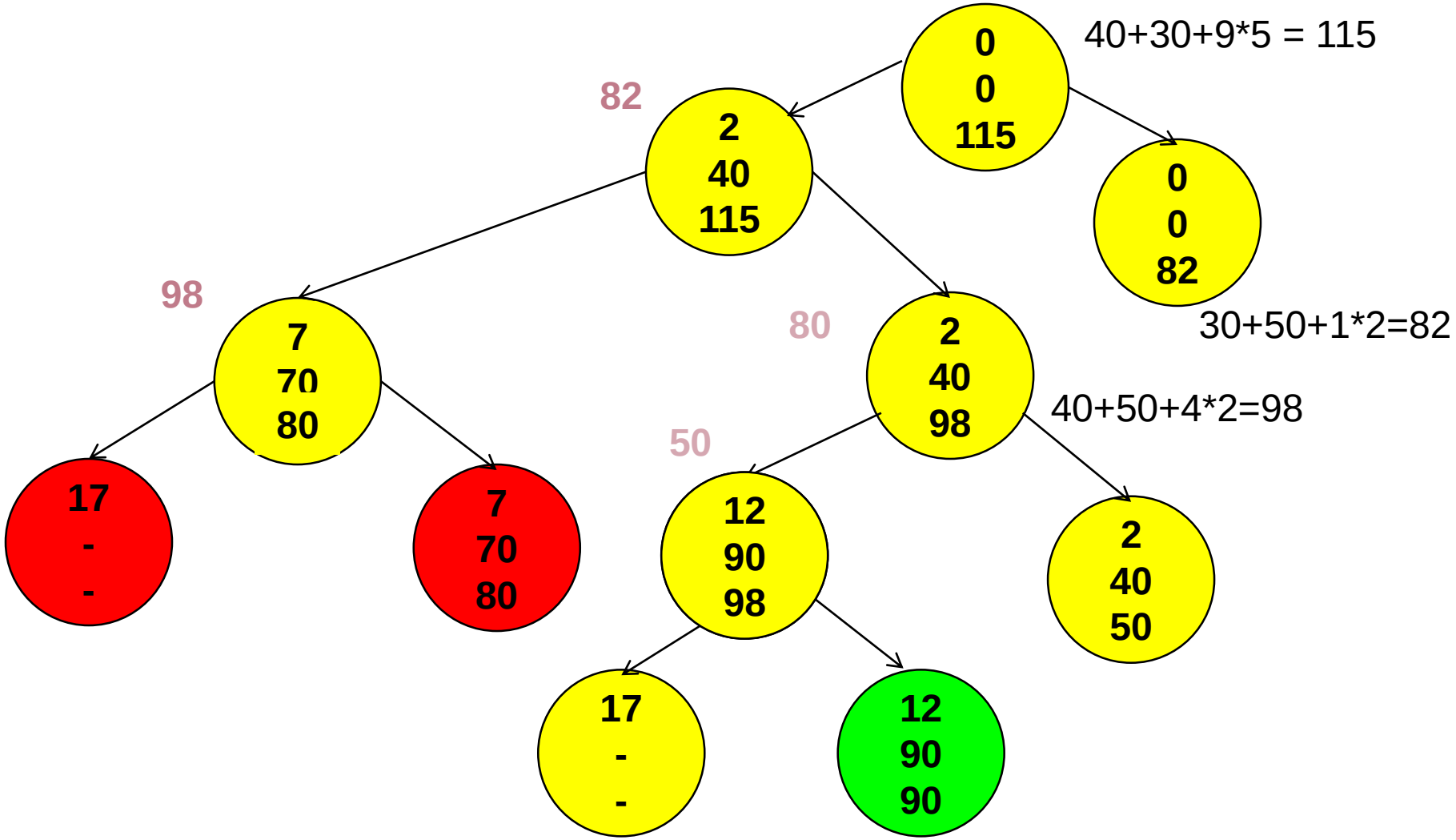
Figure 4.8 Comparison of the search costs and effective branching factors for the A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle, for various solution lengths.

کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2

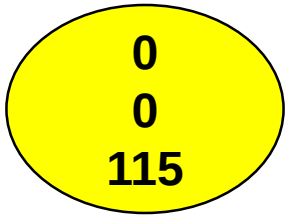
W = 16

کوله‌پشتی 0-1


0
0
115
40+30+9*5 = 115
82
2
40
115
0
0
82
30+50+1*2=82
98
7
70
80
80
2
40
98
40+50+4*2=98
50
17
-
-
7
70
80
12
90
98
2
40
50
17
-
-
12
90
90

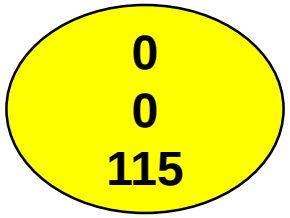
کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



کوله‌پشتی 0-1

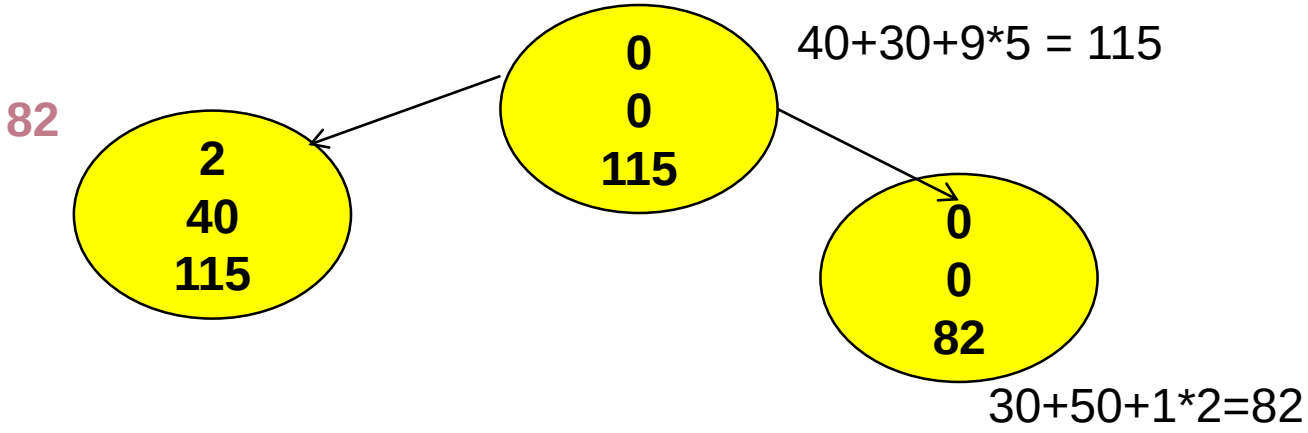
i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



40+30+9*5 = 115

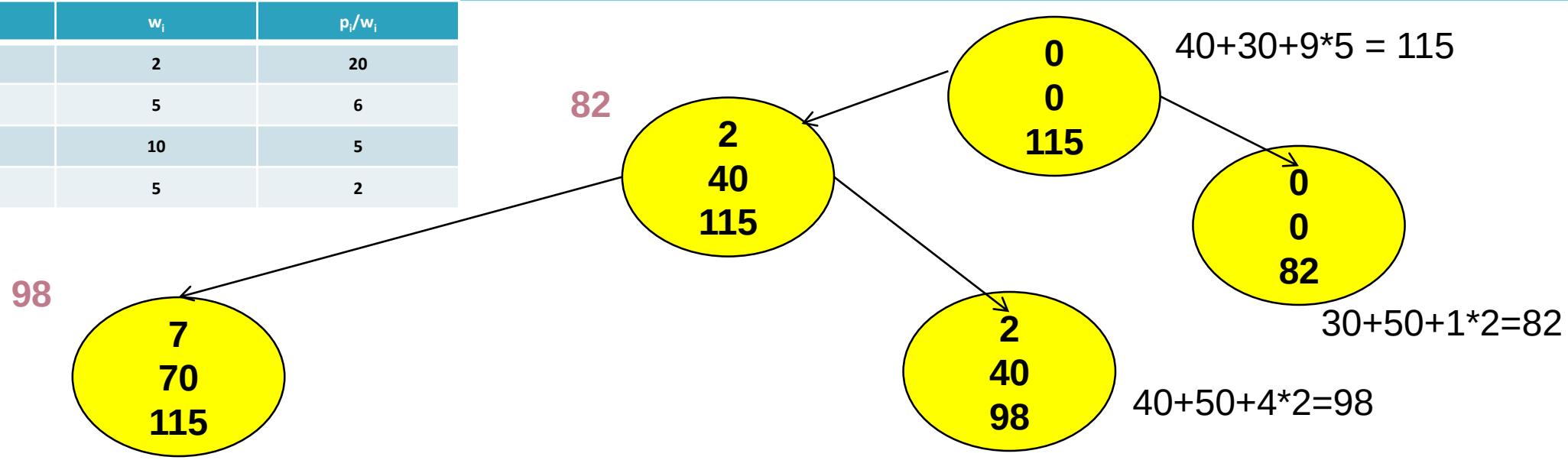
کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



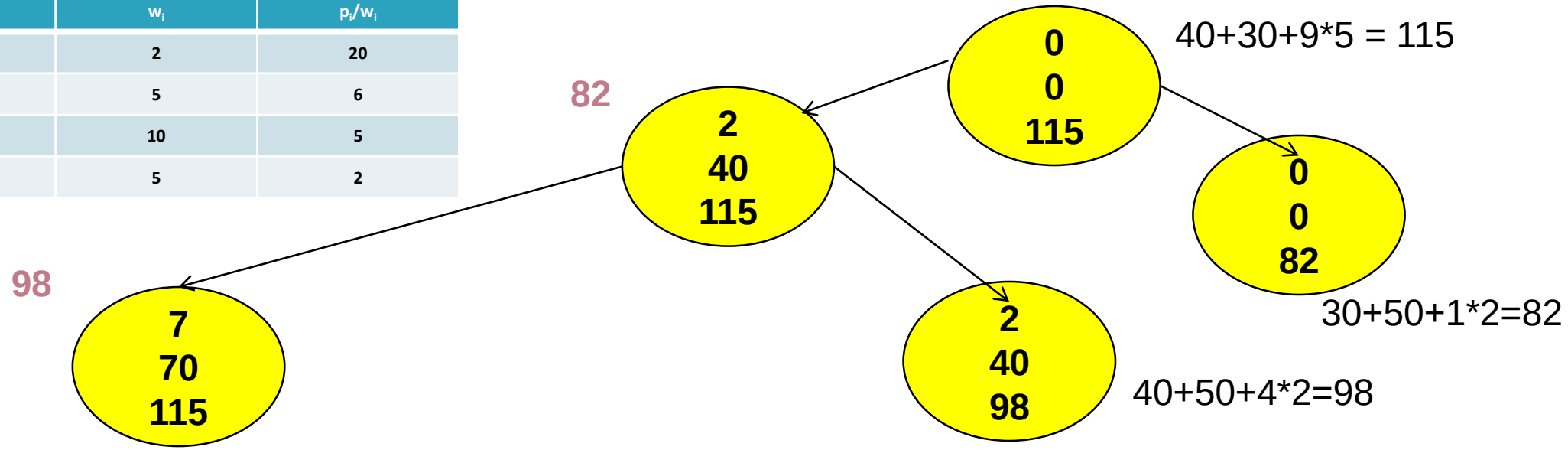
کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



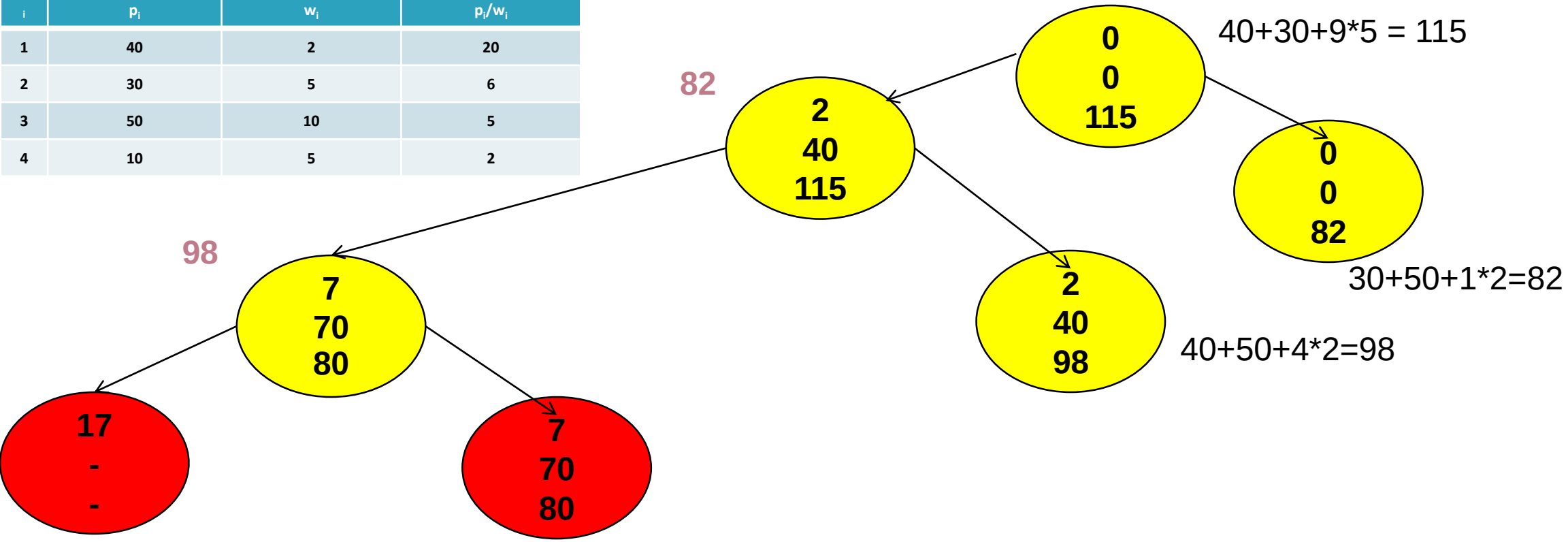
کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



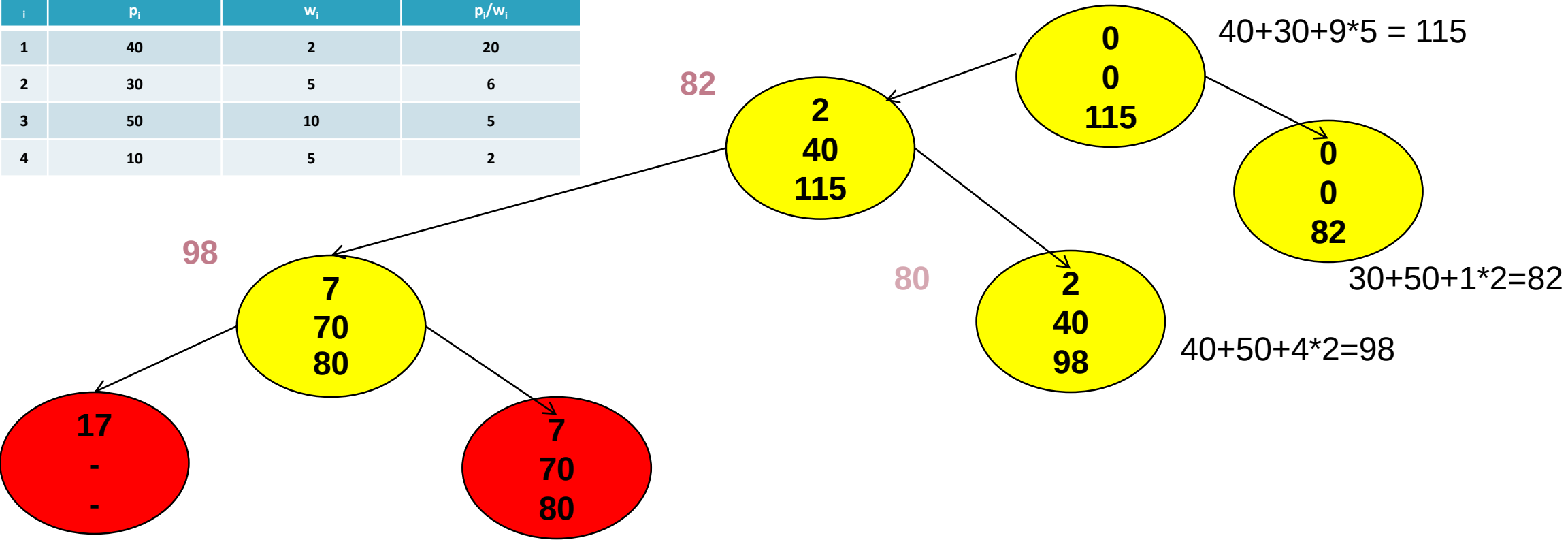
کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



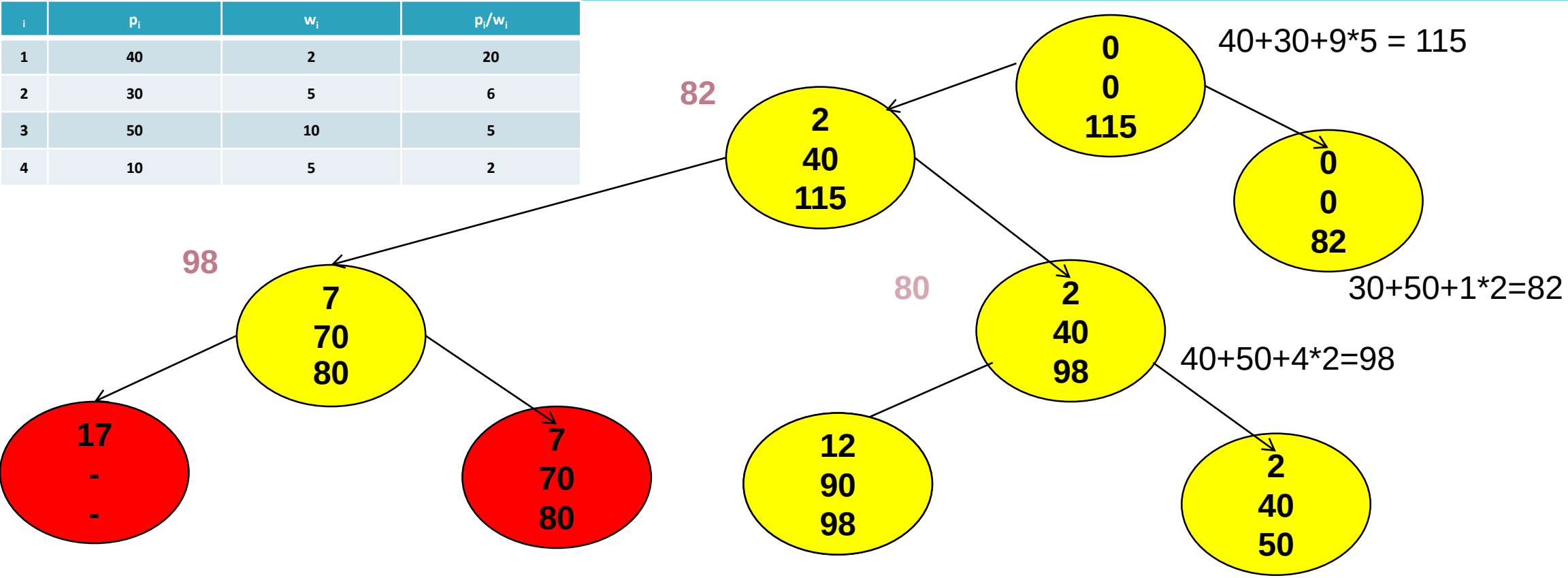
کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



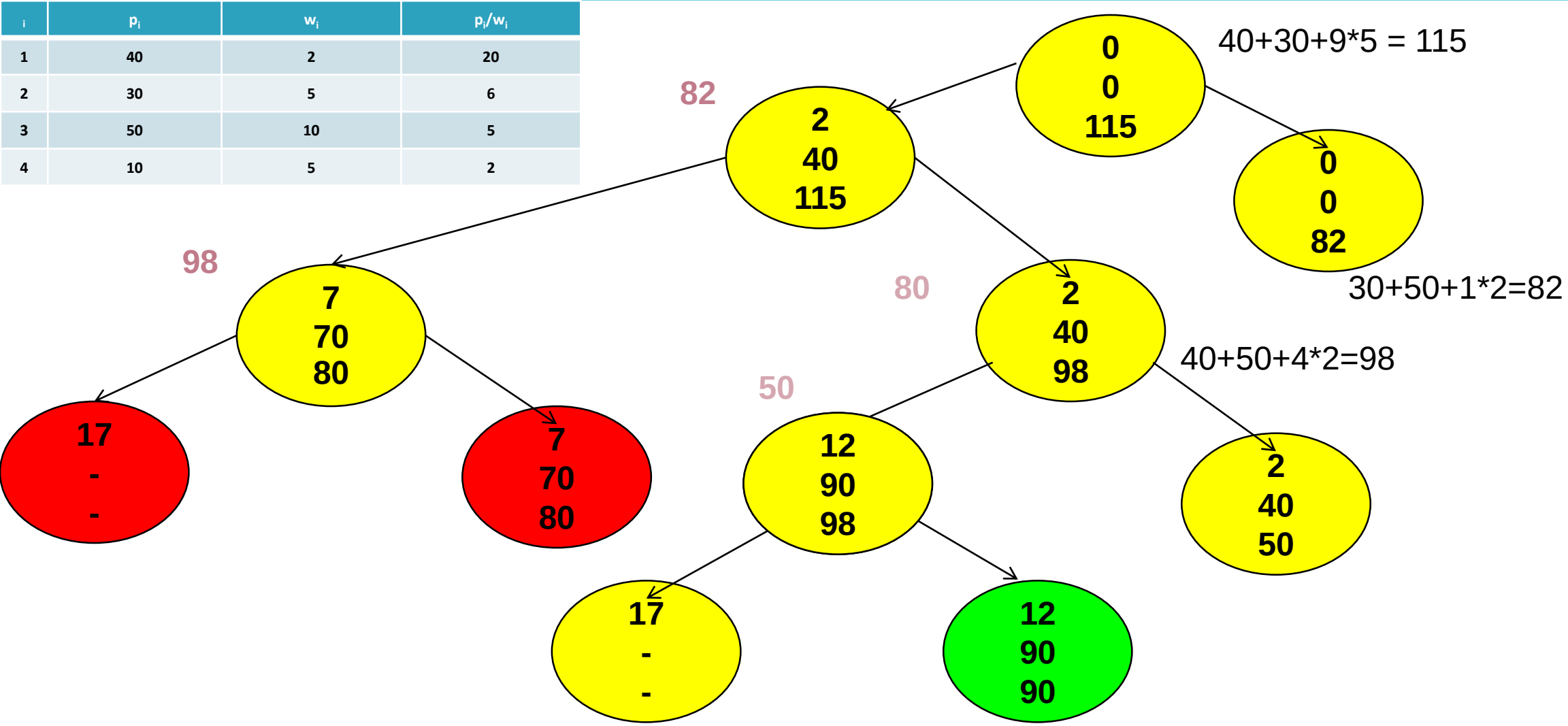
کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



کوله‌پشتی 0-1

i	p_i	w_i	p_i/w_i
1	40	2	20
2	30	5	6
3	50	10	5
4	10	5	2



Demos

http://qiao.github.io/PathFinding.js/visual/