Анализ
на задача `recycle`,

от финалния кръг на конкурса на Мусала Софт и PC Magazine, 2010-2011 г.

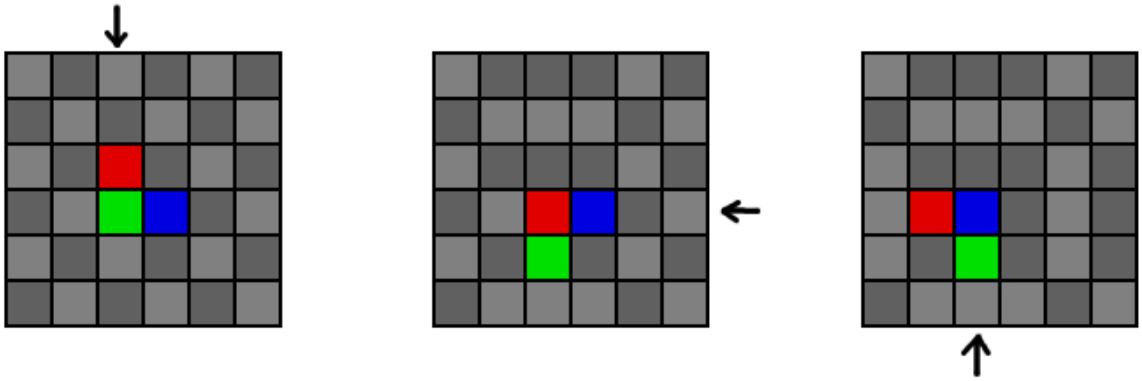
изготвил: Веселин Георгиев, 8 Май 2011 г.

Задачата тази година представляваше игра, която макар и да прилича на много други 2D „клетъчни“ игри, беше достатъчно нестандартна, за да изисква прилагането на *ad hoc* стратегии от състезателите. Действително, нормалната артилерия от теория на игрите не води състезателя на прав път. Особено вариантите, базирани на изчерпващо търсене, биват силно санкционирани от огромния брой възможни ходове на дадена стъпка – те са от порядъка на $\Theta(N*M)$ за дъска $N \times M$. Практически това означава, че можем да решим добре само най-малките възможни тестове. Например, моето bruteforce решение успява да разгледа само 4 стъпки в дълбочина на примерния тест, а той е едва 4×5!

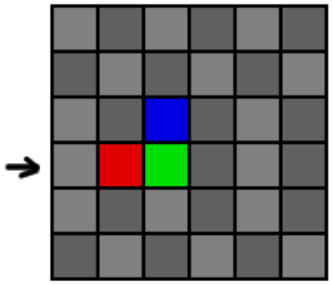
Този факт, комбиниран със щедрите ограничения за параметъра P (минималната стойност на който, по условие, е 1000, което стига да се подреди дори немалка дъска – от порядъка на 16×16!), ме доведе до мисълта да търся по-различно решение, което да сортира дъската методично, потенциално харчейки доста стъпки.

За целта ми трябваше метод, който разменя стойностите на две клетки. С дадените „циклични ротации на ред/колона“, това няма как да се постигне директно, но проучих дали не може да се приложи схема по подобие на подреждането на Рубик-кубчето. Там, за привеждане „в правилен ред“ на някои от клетките от последния слой се ползват специални „hardcode-нати“ поредици от ротации, които имат свойството да се неутрализират една друга, така че кубчето в крайна сметка да остане същото, с изключение на избраната от нас клетка, която се завърта в желаната от нас посока.

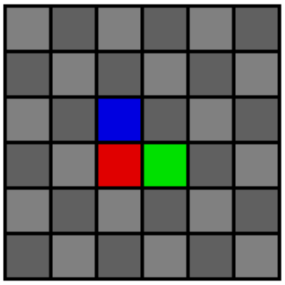
Е, експериментите ми не постигнаха напълно исканото: не намерих поредица от ротации, която да постига желаната от мен „процедура `swap()`“, но за сметка на това открих следната операция от 4 стъпки: избираме си един ред и една колона, след което изпълняваме поредицата „колона: 1 надолу“, „ред: 1 наляво“, „колона: 1 нагоре“ и „ред: 1 надясно“. След четирите стъпки, променени остават само три клетки: тази на пресечната точка на реда и колоната, както и намиращите се непосредствено вдясно и отгоре клетки:



Стъпка 1: 1 надолу *Стъпка 2: 1 наляво* *Стъпка 3: 1 нагоре*



Стъпка 4: 1 надясно



Краен резултат

Ефектът от „операцията“ представлява ротиране на трите оцветени клетки, в случая – по посока, обратна на часовниковата стрелка. Ако ползвахме по-големи стойности на цикличните завъртания във всяка от стъпките, може да постигнем размяната на клетки, които не са една до друга (без да губим целостта на останалата дъска). Всъщност, можем да ротираме клетките по трите върха на произволен правоъгълен триъгълник (с катети, успоредни на редовете и колоните). С промени по реда и посоката на стъпките може да постигнем още 7 подобни „операции“, които се различават само по ориентацията на триъгълника (накъде гледа правият му ъгъл) и посоката на завъртане на клетките от върховете (часовникова/обратна на часовниковата). Препоръчвам ви да видите визуализацията към решението ми за по-нагледна демонстрация как се реализират отделните видове ротации.

След като разполагаме с тази мощна операция, можем да пристъпим към същинския алгоритъм. Ще подреждаме дъската отляво надясно, колона по колона, като във всеки един момент ще имаме две половини: лява, сортирана, която няма да пипаме повече; дясна, несортирана, в която можем да правим каквито промени си поискаме; по средата се намира *текущата колона*, която запълваме плътно, отгоре до долу, с някакъв цвят.

Ще дефинираме процедурата **place**(*dr*, *dc*, *color*), която „поставя“ клетка с искания цвят на позиция (*dr*, *dc*). За целта намираме една такава клетка в дясната половина на дъската. Нека нейните координати са (*tr*, *tc*). За да преместим клетката (*tr*, *tc*) в (*dr*, *dc*), изпълняваме едно от трите:

1) (при $tr < dr$): за върхове на правоъгълния триъгълник избираме (*tr*, *tc*), (*dr*, *dc*), (*dr*, *tc*) и въртим обратно на часовниковата стрелка;
2) (при $tr > dr$): върховете са същите, само че въртим по часовниковата стрелка;
3) (при $tr = dr$): ако по-горният ред съществува,избираме за върхове (*dr*, *dc*), (*dr*, *tc*), (*dr*-1, *tc*) и въртим по часовниковата стрелка. Ако $dr = 1$, просто ползваме (*dr*+1, *tc*) и въртим обратно на часовниковата стрелка.

След като имаме **place()**, прилагаме следния „запълващ“ алгоритъм:

1) Обработваме наличните цветове на дъската в някакъв ред, например – от по-често към по-рядко срещаните;

2) За всеки цвят, искаме първо да поставим екземпляр от този цвят на най-горния ред в текущата колона. В някои случаи може да се изхитрим – ако в колоната вече има клетка от този цвят, може да ротираме самата колона, така че клетката да отиде отгоре. В противен случай ползваме place(0, *текуща_колона*, *текущ_цвят*);
3) Вървим надолу по колоната, като за всяка клетка, която не е със желания цвят, прилагаме place(*i*, *текуща_колона*, *текущ_цвят*).
4) Ако сме запълнили цялата колона, минаваме на следващата;
5) Ако изчерпим текущия цвят, можем да прилагаме няколко различни стратегии:
    1. На реалното състезание просто оставям останалите клетки от колоната несортирани – минавам на следващата колона, със следващия цвят. Какво се получава в крайна сметка можете да видите на визуализацията.
    2. Една по-добра стратегия би била да продължим със следващия цвят, но на същата колона. За да осигурим свързаност, то при запълване на текущата колона би следвало да започнем запълването на следващата отдолу-нагоре. Въобще, оптимално би било да подреждаме в зигзагообразен ред.

Така описаният алгоритъм (независимо какво правим в (5)) разполага цветовете на „слоеве“ - в плътни колони, с евентуално изключение на последната, при което клетките са свързани и дават максималния брой точки, който е постижим с този цвят. Единствена неоправена остава последната колона на цялата дъска, защото там няма с какво да разменяме, за да правим нашите 3-върхови „операции“. Това, обаче, не е толкова важно, след като иначе сме сортирали перфектно останалата част от дъската, особено ако действително сме подредили по-“големите“ цветове първи и от тях сме взели огромен брой точки.

Към този анализ ще видите и видео клип-визуализация на моето решение (така, както си е работило в официалното тестване). Файловете AnrieffSol.(m4v|avi) са кодирани в H.264 или MPEG-4 формат (могат да се отворят, например, с VLC). Решението оперира върху примерен тест №3 от предварителните тестове, завъртян на 90° (моето решение вътрешно винаги работи с Landscape вариант на дъската, като завърта входната, ако е необходимо – идеята е броят на колоните да е по възможност по-голям от броя на редовете). В клипа се означават клетките, които решението иска да завърти, като за първите няколко завъртания се демонстрира подробно поредицата от стъпки, която реализира завъртането. Видео-файловете са достъпни от страницата на конкурса, http://konkurs.musala.com/