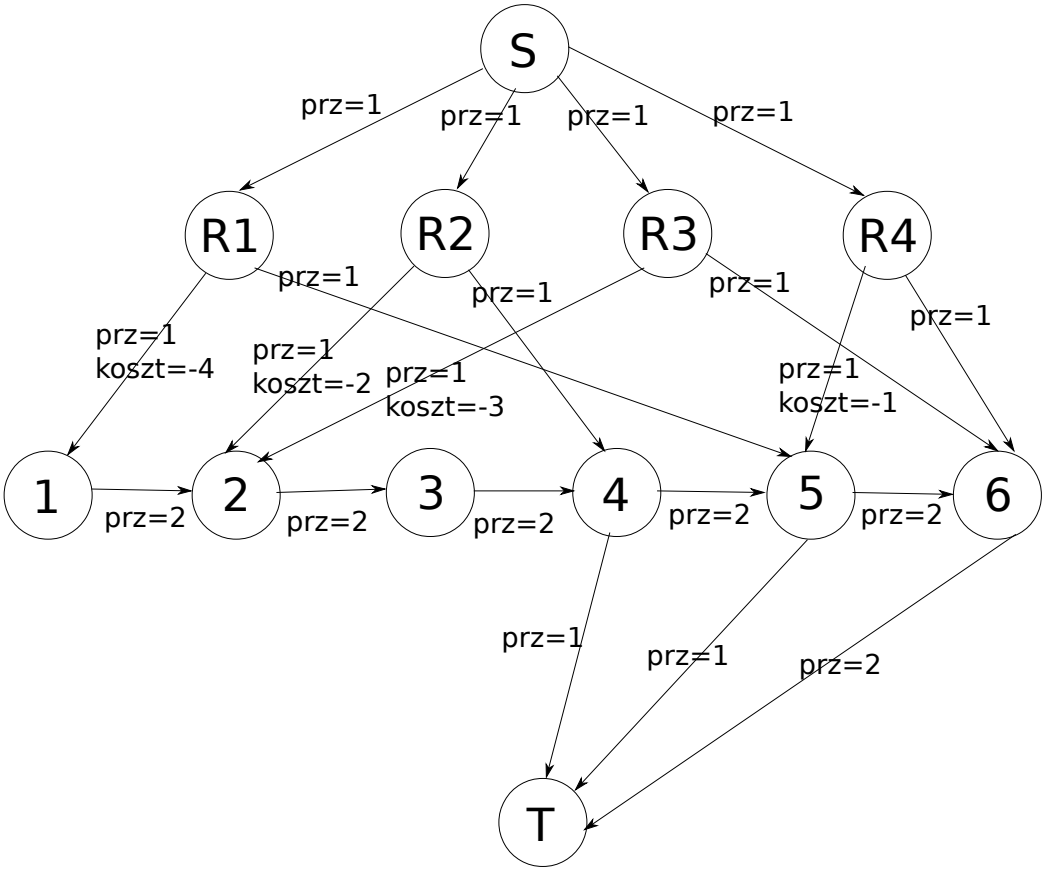
## A. Kosmiczny mózg

Zadanie sprowadza się do problemu wyszukiwania wielu, trójwymiarowych wzorców. Wyszukiwanie każdego wzorca osobno (nawet przy zastosowaniu algorytmu liniowego np. KMP) może nie być wystarczająco szybkie w tym przypadku. W sytuacji wielu wzorców szczególnie dobrze sprawdza się algorytm Rabina-Karpa, który zamiast dokładnego porównywania danego fragmentu ze wzorcem porównuje ich funkcje skrótu (wzorce należy wpierw umieścić w tablicy haszującej). Dopiero zgodność funkcji skrótu implikuje konieczność dokładnego sprawdzenia. Interesuje nas tylko informacja, czy dany schemat pojawia się w obrazie, a więc po jego wykryciu należy usunąć go z tablicy (by nie porównywać wielokrotnie fragmentów zawierających ten sam wzorzec). W przypadku tego problemu naturalne jest traktowanie analizowanego fragmentu jako liczby o podstawie 3 (są trzy możliwe symbole w obrazie mózgu). Pozostaje jeszcze kwestia szybkiego obliczania haszy kolejnych fragmentów mózgu (tzw. kroczący hasz). W przypadku jednowymiarowym (obliczane pasków rozmiaru $m$ na 1 na 1 w obrazie) dla fragmentu $a_1, a_2, \ldots, a_m$ i podstawy $b$ funkcja skrótu ma następującą postać: $f(1, m)=a_1b^{m-1}+a_2b^{m-2}+...+a_mb^0$. Wówczas skrót fragmentu $a_2, a_3\ldots, a_{m+1}$ można obliczyć tak: $f(2, m+1)=(f(1, m)-a_1b^{m-1})\cdot b+a_{m+1}b^0$. W czasie stałym otrzymujemy więc kolejną funkcję skrótu z poprzedniej poprzez usunięcie pierwszego elementu i dodaniu nowego (elementy $a_2, \ldots, a_m$ są wspólne dla obu fragmentów). Należy oczywiście pamiętać, by obliczenia redukować modulo odpowiednio duża liczba $P$ (co zmniejszy liczbę fałszywych trafień). Problem kroczącego haszu trzeba uogólnić na większą liczbę wymiarów. Po przeliczeniu pasków w drugim kroku należy przeliczyć kwadraty rozmiaru $m$ na $m$ na 1. Aby przeliczyć kwadrat następny na podstawie poprzedniego postępujemy analogicznie jak w przypadku 1-wymiarowym, ale manipulujemy na paskach (odejmując pasek górny i dodając dolny). W ostatniej fazie obliczenia odbywają się bezpośrednia na interesujących nas sześcianach, których funkcje skrótu obliczane są kolejno poprzez manipulacje całymi kwadratami (odjęcie górnego kwadratu i dodanie dolnego). Średnia złożoność całego algorytmu jest liniowa od rozmiaru danych wejściowych i wynosi $\theta(n^3+km^3)$.

## B. Górski hotel

Zadanie jest jedną z odmian problemu szeregowania zadań (ang. *weighted interval scheduling*), gdzie celem jest taki wybór zadań/przedziałów (z których każdy przynosi pewne zyski), by zmaksymalizować łączny zysk bez nakładania się na siebie przedziałów. W przypadku tego zadania przedziały to rezerwacje, ale tutaj możliwe jest jednoczesne obsłużenie maksymalnie $k$ rezerwacji (liczba pokoi w hotelu). W przypadku 1-wymiarowym (bez nakładania się przedziałów) zadanie można rozwiązać przy pomocy programowania dynamicznego w czasie liniowo-logarytmicznym. Te rozwiązanie można uogólnić dla większej liczby wymiarów, ale jest ono dosyć szybkie tylko dla małych wartości $k$ (do 2-3). W przypadku ogólniejszym należy sprowadzć ten problem do znalezienia najtańszego przepływu w sieci. Istnieje kilka możliwych konstrukcji sieci przepływowej (w jednej z nich można odwrócić problem i minimalizować koszt „niewziętych” rezeracji by uzyskać dopuszczalne rozwiązanie dla danej bazy noclegowej). W omówionym przypadku będziemy szukać najtańszego przepływu po zmianie znaku zysków na ujemne (co będzie odpowiadać maksymalizacji zysków w oryginalnym problemie). Wierzchołkami w sieci przepływowej będą momenty czasu (jest ich max. 1000), $n$ rezerwacji oraz dodatkowo źródło $S$ i ujście $T$. Koszty krawędzi w sieci będą ujemne w przypadku uwzględnienia danej rezerwacji (-zysk) i te krawędzie będą wówczas prowadzić od wierzchołka rezerwacyjnego do momentu początku rezerwacji, natomiast krawędzie z kosztami zerowymi będą prowadzić od wierzchołka rezerwacyjnego do momentu końca rezerwacji (nieuwzględnienie rezerwacji). Przepustowości między kolejnymi momentami czasu będą wynosić $k$, dodatkowo wystąpią krawędzie wychodzące z momentów końca rezerwacj do ujścia (przepustowość równa liczbie rezerwacji kończących się w danym momencie) oraz ze źródła do wierzchołków rezerwacyjnych (przepustowości jednostkowe).

Mając dane wejście:
4 2
1 5 4
2 4 2
2 6 3
5 6 1
Sieć przepływowa wyglądałaby następująco (zaznaczone przepustowość i koszt, w przypadku braku informacji o koszcie jest on równy zero):




Celem jest uzyskanie najtańszego przepływu z *S* do *T*, którego wartość wynosi *n* (w tym przypadku 4). Wynikiem tutaj jest -8 (czyli zysk 8 po wzięciu wszystkich rezerwacji z wyjątkiem drugiej). Ze względu na ograniczoną liczbę momentów czasu (do 1000) można przyjąć, że dla największych danych liczba wierzchołków grafu rośnie liniowo od *n* (gdyby liczba momentów czasu była większa, to należałoby rozpatrywać jedynie istniejące zakończenia przedziałów nadając im nową numerację, ale w tym przypadku to nie było konieczne). Ten graf jest rzadki i liczba krawędzi rośnie liniowo od liczby wierzchołków. Jak już zostalo wspomniane pożądany przepływ *P* z wierzchołka *S* do *T* ma wynosić *n*. W tym przypadku możliwe jest uzyskanie najtańszego przepływu w czasie *P*·koszt_algorytmu_najkrótszych_ścieżek stosując algorytm Busackera-Gowena, który wylicza kolejno najtańsze (w sensie kosztu) ścieżki z *S* do *T* w sieci residualnej aż do uzyskania pożądanego przepływu (wykonań algorytmu ścieżkowego nie będzie więcej niż *P*). Do obliczania najkrótszych ścieżek można zastosować algorytm Dijkstry (o złożoności liniowo-logarytmicznej przy użyciu struktury kopca binarnego) do wszystkich iteracji z wyjątkiem pierwszej (gdzie należy użyć alg. Forda-Bellmana). Wynika to z istnienia ujemnych kosztów, które w kolejnych iteracjach należy zredukować posługując się tzw. potencjałami wierzchołków (bazują one na koszcie najtańszych ścieżek do danego wierzchołka). Dzięki temu w sieci residualnej będą

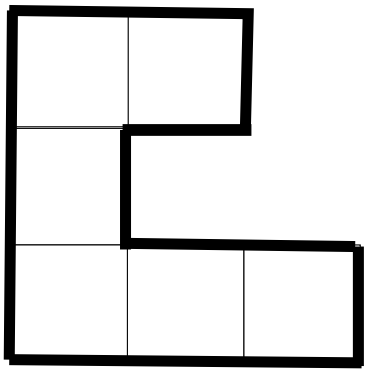
tylko koszty nieujemne, co umożliwi poprawne działanie alg. Dijkstry. W tym przypadku całkowita złożoność algorytmu to $\theta(n^2 \log n)$. Akceptowalny czas wykonania mogły uzyskać również rozwiązania oparte o alg. Forda Bellmana ($\theta(n^3)$), które po przeprowadzeniu odpowiednich modyfikacji (np. dzięki strukturze kolejki) średnio wykonują dużo mniejszą liczbę iteracji niż podstawowa wersja algorytmu.

## C. Park narodowy

Celem było znalezienie sposobu na szybkie znajdowanie odpowiedzi do zapytań o pewne dwuwymiarowe obszary. Konkretnie chodziło o znalezienie różnicy na tych obszarach (między najniższym, a najwyższym punktem). Naiwna implementacja (złożoność pesymistyczna $k \cdot n \cdot m$) była oczywiście zbyt czasochłonna. Strukturą danych, która dobrze nadaje się do rozwiązania tego problemu jest 2-wymiarowe drzewo przedziałowe. W tym przypadku można było użyć statycznej struktury drzewa, która po początkowej inicjalizacji już się nie zmieniała. Do obliczania różnicy wysokości należało wykorzystać dwa drzewa przedziałowe, które podawałyby wysokość najwyższego i najniższego punktu na danym obszarze (wynikiem jest różnica tych dwóch wyników). Koszt pojedynczego zapytania w takim drzewie to $\log n \cdot \log m$, co daje akceptowalną złożoność całkowitą algorytmu, która wynosi $\theta(k \cdot \log n \cdot \log m)$.

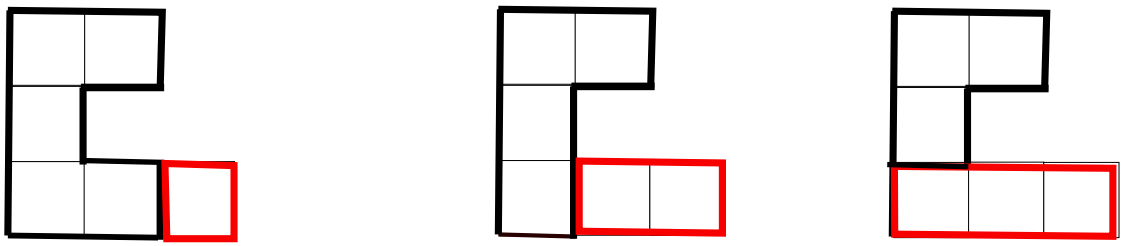
## D. Teściowa

Zadanie polega na znalezieniu liczby sposobów podziału wielokąta (o bokach całkowitej długości) na prostokąty o bokach całkowitej długości. Najpierw należało zauważyć, że rozważany wielokąt jest dość specyficzny: posiada on boki całkowitej długości o orientacji pionowej lub poziomej, a na dodatek w jego opisie pojawia się tylko jeden symbol ‘G’. Ten ostatni fakt oznacza, że lewa strona wielokąta stanowi zawsze duży, pionowy bok (jak w zwykłym prostokącie), a nieregularności pojawiają się tylko z prawej strony. Dzięki temu każdy wielokąt można opisać za pomocą $n$ liczb ($n$ to długość jego lewego boku), z których $i$-ta liczba do długość $i$-go poziomego paska (licząc od góry). Każdy pasek ma szerokość jednostkową, a wszyskie paski ułożone jeden na drugim łącznie tworzą cały wielokąt. Przykładowo poniższy wielokąt:



Można reprezentować za pomocą trójki liczb: (2, 1, 3).
Okazuje się, że tak postawiony problem można zdefiniować rekurencyjnie jako funkcję mniejszych podproblemów. Wystarczy wpisać w dany wielokąt pewien prostokąt i problem sprowadza się do mniejszego wielokąta. Najważniejsze jest jednak zrobienie tego w ten sposób, by nie zliczać tych samych sposobów rozwiązania wielokrotnie i by podproblemy też miały postać takich wielokątów, jak wyżej opisany (które mają tylko jeden symbol ‘G’ w opisie, czyli można je zdefiniować za pomocą opisanej sekwencji liczb). Aby nie zliczać tych samych sposobów wielokrotnie należy

wybierać odpowiedni punkt „zaczepienia” nowo wpisanych prostokątów. W tym przypadku rozwiązaniem może być wybranie „prawego-dolnego” rogu wielokąta. Chodzi nam o taki punkt należący do obwodu wielokąta, który jest możliwie najbardziej położony na prawo. W przypadku, gdy takich punktów jest wiele, to wybieramy ten położony najbardziej na dole. Następnie poczynając od tego punktu próbujemy wpisać w wielokąt wszystkie możliwe prostokąty mieszczące się w nim. Po wpisaniu każdego prostokąta redukujemy problem do mniejszego wielokąta. Dzięki temu, że punkt zaczepienia jest możliwie najbardziej na prawo, to wpisując prostokąty nigdy nie dojdzie do sytuacji, w której którykolwiek poziomy pasek zostanie podzielony na dwie części (co znacznie skomplikowałoby rozważany wielokąt) – rozważane podproblemy będą więc tego samego typu co główny problem. Poniżej przykład dla omawianego wielokąta:



W punkcie zaczepienia można umieścić 3 różne prostokąty: 1x1, 2x1 oraz 3x1. W rezultacie wielokąt (2, 1, 3) sprowadza się do wielokątow (2, 1, 2), (2, 1, 1) i (2, 1, 0) (liczba sposobów podziału wielokąta głównego jest oczywiście sumą liczb sposobów podziału mniejszych wielokątów).

Takie rozwiązanie tego problemu nie jest jednak wystarczająco wydajne ze względu na eksplozję kombinatoryczną liczby sposobów podziału prostokąta. Można jednak zauważyć, że przetwarzane problemy mogą się powtarzać. Dzieki temu każdy podproblem można rozwiązać tylko raz, a w kolejnych wywołaniach wystarczy tylko zwracać wynik. Użycie spamiętywania w rekurencji znacząco redukuje czas działania algorytmu. Rozmiar największego wielokąta to 7x7, a więc każdy z nich można przedstawić jako sekwencję co najwyżej 7 liczb (z zakresu od 0 do maks. 7). Taką sekwencję można traktować jako liczbę o podstawie 8 i właśnie dla tych liczb (identyfikujących jednoznacznie wielokąt) należy spamiętywać raz obliczony wynik. Łączna liczba podproblemów do zapamiętania to maksymalnie $8^7$, czyli nieco ponad 2 miliony komórek w tablicy. Pesymistyczny czas działania algorytmu to $\theta(a \cdot b \cdot a^b)$, gdzie $a$ i $b$ to rozmiary najmniejszego prostokąta, w który można wpisać wielokąt.