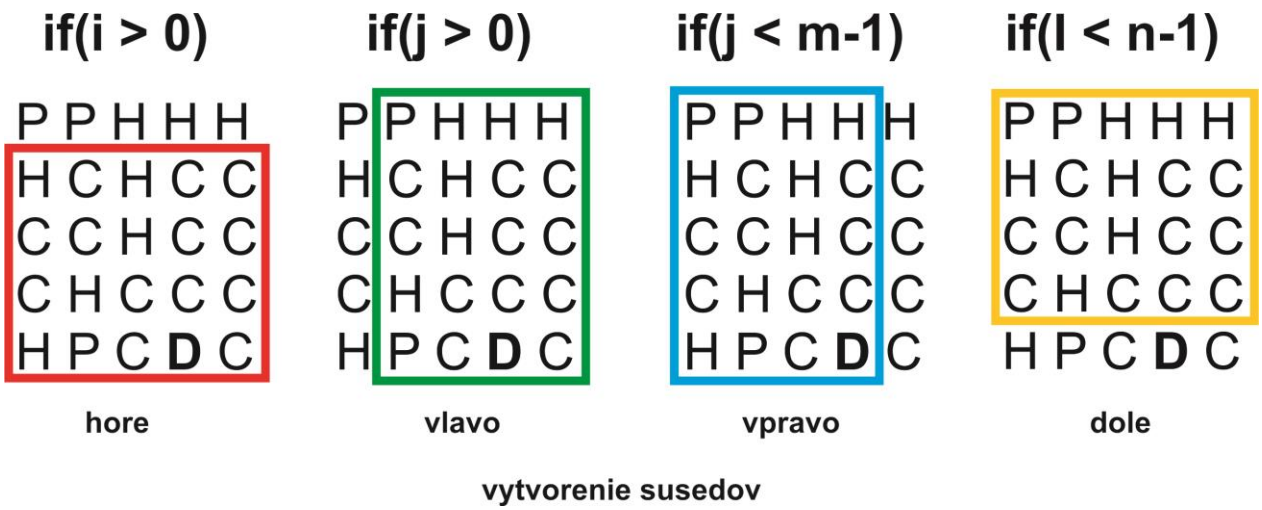
Implementácia zadania

Na realizáciu zadania sa použili 3 štruktúry:
1. **EDGE** - bod na 2d mape predstavený atribútmi: **pos** - číslo pozície v mape, **length** – dĺžka trate vyjadrujúca čas prechodu z aktuálneho bodu, **next** - odkaz na susedné body.
2. **HEAPNODE** – taktiež bod na 2d mape, obsahuje ale **sDist** - dĺžku vzdialenosti od štartovacej pozície k pozícii **pos**.
3. **PRIORFRONT** - binárna halda pri ktorej sa bude pristupovať pomocou index ukazovateľa **pt** do **array** pola štruktúr HEAPNODE. Táto štruktúra sa zdekladruje ako globálna premenná.

Na záchranu princezien sa použila funkcia **int *zachran_princezne(char **mapa, int n, int m, int t, int *dlzka_cesty)**. Túto funkciu som rozdelil do 4 rôznych fáz:

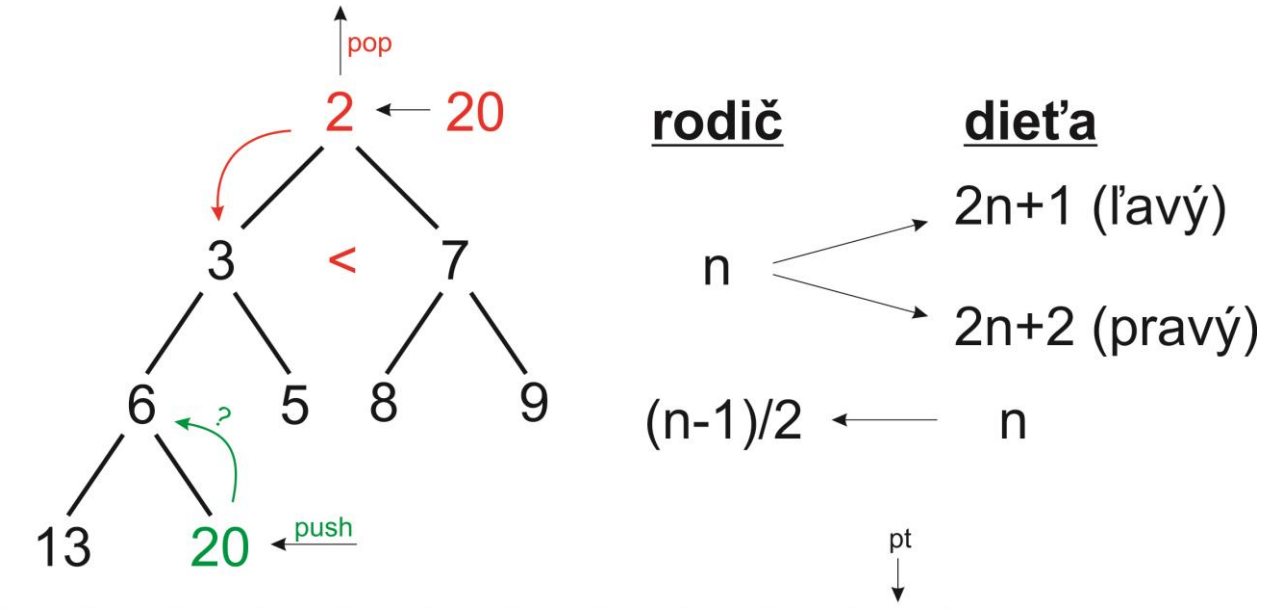
1. Vytváranie grafu z 2D mapy

Pomocou daných parametrov funkcie máme 2D pole charov, počet riadkov n, počet stĺpcov m. Na vytvorenie grafu je potrebné prejsť všetkými bodmi (charov) v poli cez for cykly. Ako prvé zisťujeme, či na príslušnom políčku nieje drak alebo princezná. Ak je tam drak, jeho 2d pozíciu si zapíšeme jedným číslom pozície do premennej dragon takým spôsobom, že vykrátime príslušné číslo riadku a pričítame k nemu číslo stĺpca (i*m + j). To isté platí aj u nájdení princeznej. Kedže máme 3 princezné, čísla si uložíme do poľa s 3 prvkami.
Následne zistíme dĺžku prechodu z tohto bodu pomocou funkcie **getLength** a uložíme ju do premennej length, ktorú budeme potrebovať pre následné vytváranie grafu.
Graf je predstavený poľom bodov (EDGES), pričom index vyjadruje pozíciu. Do každého bodu grafu pridáme všetky susedné body v mape cez spájaný zoznam pomocou funkcie **insert**. Nie každý bod má rovnaký počet susedov, preto sa tento problém ošetrí pomocou 4 podmienok, ktoré sú ilustrované v následujúcom obrázku:



2. Hľadanie najkratších ciest

Teraz keď už máme vytvorený graf a našli sme pozície princezien, začneme hľadať najkratšie možné cesty do všetkých bodov v grafe prostredníctvom dijkstrovho algoritmu. Najprv ho vykonáme zo štartovacieho bodu 0 na nájdenie cesty k drakovi.
Funkcia dijkstra používa prostriedky binárnej vyhľadávacej min. haldy.



n	0	1	2	3	4	5	6	7	8	9
sDist	2	3	7	6	5	8	9	13	20	

Na začiatok inicializujeme pozíciu štartovacieho bodu a jeho vzdialenosť ktorá bude nulová. Tento bod vložíme do haldy pomocou funkcie **push.**
Následne prejdeme všetkými vrcholmi v prvom for cykle v ktorom sa vždy najprv pomocou funkcie **pop** vyberie vrchol, ktorý má najmenšiu doterajšiu vzdialenosť (sDist) od štartu. Jeho pozíciu si uložíme do pomocnej premennej pos, ktorá sa použije v druhom for cykle.
Druhý cyklus bude iterovať toľko krát, koľko má daný vrchol v grafe s pozíciou pos susedov. Ak sa nájde kratšia cesta k susedným vrcholom, aktualizuje sa odkaz na pole path – cesta ktorá obsahuje najvýhodnejšiu pozíciu vrcholu z ktorej sa dostaneme do pozície príslušného indexu. Aktualizuje sa aj sDist susedného vrchola, ktorý sa spolu s jeho pozíciou vložia do haldy cez funkciu push.
Takto budeme cez dijkstrov algoritmus hľadať najkratšie cesty ešte 3 krát pre každú princeznú. Narozdiel od prvého hľadania budeme vo funkcii vracať pole sDist do dvojrozmerného pola dista s indexom i-tej princeznej. Pole Dista obsahuje vzdialenosti ku všetkým bodom od každej jednej princeznej. Tieto údaje potrebujeme na nasledujúcu fázu.

3. Hľadanie najkratšej kombinácie ciest

Na to, aby sme mohli zachrániť princezné v najkratšom možnom čase je potrebné nájsť všetky možné poradia záchran. K tomu použijeme trojitý for cyklus pri ktorom máme 3 rôzne iteračné čísla. V najvnútornejšom for cykle ak sú všetky iteračné čísla rôzne, tak sa našla jedna možná kombinácia záchran. Obrázok popisuje všetky možné kombinácie záchran v 1. scenári zadania.

princezné
1. 2. 3. i j k
8 + 1 + 5 (0, 1, 2) = 14
8 + 6 + 5 (0, 2, 1) = 19
7 + 1 + 6 (1, 0, 2) = 13
7 + 5 + 6 (1, 2, 0) = 18
2 + 6 + 1 (2, 0, 1) = 9
2 + 5 + 1 (2, 1, 0) = 8

1. index - destinácia
2. index - štart

```
sum = dista[i][dragon]      +
      dista[j][princess[i]] +
      dista[k][princess[j]];
if(min >= sum){
    min = sum;
    data[0] = i;
    data[1] = j;
    data[2] = k;
}
```

Dvojrozmerné pole dista obsahuje všetky vzdialenosti bodov z rôznych štartov. Štarty sú vyjadrené pozíciou v druhom indexe, pričom destinácia je v prvom. Modré kombinácie záchran začínajú najprv prvou princeznou, zelené druhou, a červené treťou. Záchrana tretej princeznej ako prvej vynáša najlepší výsledok vzdialeností, pretože je najbližšie ku drakovi. Analogicky tak postupujeme aj pri vyberaní záchrany druhej princeznej.

4. Ukladanie postupností krokov do outputu

Nakoniec je potrebné uložiť postupnosti krokov po mape do output poľa. Nepárne prvky v poradí poľa predstavujú stĺpcové súradnice, párne naopak riadkové. Použijú sa tu 2 rekurzívne funkcie, pričom prvá funkcia **backward** nájde súradnice k drakovi takým spôsobom, že postupuje od destinácie (draka) pomocou poľa **path**, ktorý ho postupne nasmeruje až na začiatok.
Následne zavoláme 3 krát funkciu **forward**, ktorý naopak postupuje do destinácie najprv od prvej princeznej. Stĺpcové súradnice dostaneme u oboch funkciách zmodulovaním destinácie s počtom stĺpcov, riadkové súradnice namiesto modulovania získame vydelením.
Nakoniec počet x+y súradníc (xyAmount) vydelíme dvoma a získame dĺžku cesty.

Časová zložitosť prvej fázy je **O(n*m)** – lineárne prechody načítavania grafu, kde n je počet riadkov a m počet stĺpcov
Časová zložitosť druhej fázy je **O(n*logn)** – funkcia pop/push má zložitosť logn, pričom je to potrebné vykonať pre každý vrchol, teda pre n vrcholov.