Grafurile sunt identificate prin matrice de adiacenta, fiind utilizate in problema si

* Un minheap (pentru dijkstra si pentru memorarea sortata a unor numere – coduri specifice avioanelor in exces. Explicatia detaliata in textul urmat de heading-ul partii a doua, in subheading-ul de backtracking)
* O coada (pentru transferul de avioane in cazul in care vecinii unei insule cu depasirea de toleranta nu mai pot accepta avioane (transfer *in lant*)
* Un trie pentru stocarea numelor de resurse

Fiecare structura corespunde unui header. *Proiectul* contine si un header cu functii pentru realizarea algoritmului lui Dijkstra, unul pentru utilitati (functii care s-ar putea gasi, la fel de bine, si in alte programe – minim, maxim, alocarea unui string,..).

Toate headerele sunt grupate in MainHeader.h.

# Partea intai a problemei

Resursele pot fi identificate in ordine prin folosirea unui trie. Acesta memoreaza UNIC numele unor resurse prin valoarea „boolean” (specifica unui nod de arbore) pentru sfarsirea unui cuvant, fara a fi nevoie de eliminarea dublurilor sau de sortare. Se poate identifica printr-o parcurgere si numarul termenilor stocati.

Functia de Dijkstra are, in plus fata de parametrii obisnuiti, si o variabila numita *wait,* specifica timpului de stationare.

Conform abordarii *greedy* specifica Dijkstra, se extrage la urmatorul pas nodul cu cea mai mica distanta (aici timp). Structura de nod pentru Dijkstra (nod de minheap) contine indexul unei insule si timpul necesar ajungerii la ea de la un nod *sursa*. La fiecare iterare in Dijkstra, se extrage un nod **pentru a evalua vecinii sai.**  Aceasta evaluare presupune ca se va stationa pe insula pana la parcurgerea vecinilor, **deci nodurile extrase si introduse vor contine si timpul de asteptare.**

In schimb, in abordarea generala Dijkstra, in cazul in care vrem sa aflam un cost mai bun pentru ajungerea la o insula, se considera un vector de distante.In acest vector de distanta actualizam **fara timpul de asteptare** pentru ca vDist[INSULA]=TIMP inseamna ca este necesar TIMPUL corespunzator pana sa ajungemla INSULA. De asemenea, in urma efectuarii algoritmului de drum de cost minim, verificarile se fac pe acest vector de distante, nu pe nodurile din heap (**heapul e gol la final)**, iar stationarea nu se produce decat daca se viziteaza si un vecin dupa (prima si ultima insula intr-un drum nu au stationare).

Asa cum se poate vedea in program, abordarea este in felul urmator:

*Extrage nod din heap*

*Pentru fiecare vecin*

*daca timpul nodului + timpul pe o muchie < distanta pana la vecin (din vector)*

*insereaza in heap timpul + asteptarea (de 15 sau de 0) (si scoatem din heap un eventual element cu acelasi index de insula, adaugat anterior, daca solutia nou gasita este mai buna)*

*distanta (in vector) pana la nod este timpul fara asteptare*

Un Dijkstra normal (cazurile drum\_zbor, timp\_zbor) are timpul de stationare 0 (wait = 0). Astfel, nu mai este nevoie de crearea a 2 functii specific existentei sau nu a timpului de asteptare. In acelasi mod am refolosit si minheapul de la dijkstra in partea de backtracking.

Trecerea catre partea a doua se face prin observatia ca, citind cerinte continuu, la un moment dat se va citi un numar, sau nu se va mai citi nimic. In primul caz, numarul este chiar totalul insulelor pentru partea a doua.

# Partea a doua a problemei

## Cazul de *Stack Overflow*

Daca numarul total de avioane depaseste cel maxim (toleranta inmultita cu numarul de insule) atunci avem un overflow. De asemenea, daca transferul in lant (fara backtracking, de la testul 14) nu functioneaza, am fi in aceeasi situatie, dar nu este cazul.

## Cazul de backtracking

Funcția de backtracking, in particular pentru problema de față, generează toate posibilitățile de atribuire de avioane in exces pentru fiecare insula.

Ideile de la care am pornit sunt următoarele:

1. Backtracking-ul, in general, poate produce toate modalitatile de combinare a unor numere. Daca numerele ar fi sortate, se genereaza modalitatile sortat. Daca numerele sunt de la 0 la n problema reprezinta chiar generarea tuturor numerelor de maxim n cifre din baza n.
2. In fisierele de output date sunt generate toate posibilitatile de vectori de avioane pentru fiecare insula IN ORDINE DE BACKTRACKING. De aici am avut initial ideea sa fac backtracking de atatea ori cate insule sunt si, la fiecare iteratie, sa gasesc solutiile corespunzatoare insulelei cu ordinul iteratiei, dar am renuntat pentru ca nu mi s-a parut o solutie mai buna.

Se observa ca numarul avioanelor in exces este un numar de o cifra. Se poate atribui fiecarui avion in exces o cifra de ordin. Daca avioanele in exces sunt 22,77 si 88 atunci 22 are ordinul 1, 77 ordinul 2, 88 ordinul 3. De mentionat ca vExcess din problema este indexat de la 0 si vExcess[ordin] = avionul ordinului ordin+1 (exemplu aici - vExcess[0]=22).

Am dorit dupa apelarea functiei de back obtinerea unor coduri descrise astfel: pentru o insula, cifrele codului corespund ordinelor de avioane in exces. Spre exemplu 23 inseamna ca insula primeste al doilea avion in exces si al treilea. Pe exemplul anterior avioanele ar fi 77 si 88.

Pentru a ajunge la aceste coduri, vectorul de niveluri din backtracking are ordinele avioanele in exces si valori insule. Spre exemplu un vector de niveluri :

vLevel {3, 3, 4}

inseamna ca primele 2 avioane in exces se duc la insula 3, ultimul la insula 4. Se tine cont de constrangeri (o insula nu poate fi repetata decat in ordine consecutiva, nu se doreste depasirea tolerantei,...)

Se itereaza prin vLevel si se creaza codurile.

Pentru exemplul acesta, pentru insula 3 codul este 12 (primul si al doilea avion in exces), iar pentru insula 4 - codul este 3 (primeste al 3-lea avion)

Am creat un vector de minheap-uri; fiecarei insule ii corespunde un heap unde se memoreaza codurile. Acestor heapuri li se vor sorta vectorii descrescator prin heapsort.

Pentru fiecare insula, pentru fiecare solutie din heap-ul ei, se afiseaza intai avioanele prezente initial in insula si apoi avioanele corespunzatoare codului. (sortat deja fiindca, la adaugarea avioanelor noi in testele 9-13 nu se afecteaza starea sortata)

Aceasta abordare permite afisarea, pentru o insula, conform outputurilor puse la dispozitie: se afiseaza

* toate posibilitatile unui vector de avioane : existente + un numar n de avioane transferate
* apoi unui vector de avioane : existente + n-1 transferate
* ș.a.m.d.

Un cod particular : 0 inseamna ca insula nu primeste niciun avion in acea solutie. Deci se afiseaza avioanele insulei si atat.

De asemenea, pentru o insula cu toleranta depasita am avut grija sa nu se mai afiseze avioanele in exces.

Se poate face o analogie cu bazele de numeratie. Numarul de cifre al unui cod pentru o insula este numarul avioanelor de primit (in limita tolerantei), iar baza maxima este numarul total al avioanelor de transferat. Pentru o insula cu un heap cu valorile de cod {123,23,12,3,1,0}, sau, scris pe larg : {123,023,012,003,001,000} intai se afiseaza avioanele deja existente + cele 3 avioane de transferat, apoi cele deja existente + cel de-al doilea si al treilea din exces, etc.

## Cazul de transmitere in lant

Cand o insula cu exces de avioane nu poate da unor insule adiacente insule.

Solutie: se introduce intr-o coada insula cu exces, se cauta BFS pana cand se gaseste o insula Y care poate accepta avioane. Daca insula X este cea care a determinat introducerea insulei Y in coada, atunci X da avioane lui Y. Procesul se repeta cu o coada reinitializata cat timp se depaseste macar o data toleranta unei insule.