**Labb 1**

**Labb1.1**

Jag hade inga speciella problem med att generera domänen eller problemen. Inte heller några problem med att köra FF heller.

**Labb1.2**

Började med att göra ett problem med 9 lådor och 9 skadade personer, det tog 0,37 sekunder att köra det problemet med IPP och tänkte då att jag måste göra problemen bra mycket större.

Genererade då problem med 100 lådor samt 3 stycken skadade personer, även det gick ganska fort att köra med IPP, någon sekund bara.

Genererade då ett problem med 100 lådor och 10 stycken skadade personer, det gick inte alls att köra med IPP och det tog även ett tag att köra med FF.

Testade att köra med 100 lådor och 5 stycken skadade personer, det gick inte heller att köra med IPP så testade att köra med 100 lådor och 4 stycken personer.

Det gick att köra och tog ungefär lite mer än 2 minuter med IPP.

Var lite nyfiken på hur stor påverkan lådorna har så testade då att köra med 50 lådor och 5 personer vilket inte heller var körbart med IPP.

Slutsatsen är att sökrymden ökar drastiskt och problemet får en hög fögreningsfaktor då vi ökar antalet sjuka personer i problemet. Det märktes avsevärt då vi ökade ett problem med 100 lådor och 3 sjuka person till 100 lådor och 4 sjuka personer, exekveringstiden gick ifrån 3 sekunder till mer än 2 minuter.

Det tar då mer tid att konstruera grafen och sökningen i A\* som IPP använder sig av.

**Labb1.3**

Det största problemet som genererades i Labb1.2 var 100 lådor och 20 skadade personer.

Lama-2011

Det tog 0,17 sekunder att hitta första lösningen.

Madagascar-p

Den klarar av att lösa problemet med 0,96 sekunder.

YAHSP3

Det tog 0,837 sekunder att hitta första lösningen.

Provar nu att göra ett större problem för att se hur stort problem jag kan lösa inom en minut.

Lama-2011

Varje problem jag gjorde löstes väldigt snabbt. Gjorde ett problem med 500 lådor och 100 sjuka personer.

Madagascar-p

Klarar av att lösa problem med 150 lådor och 47 stycken sjuka personer på 25 sekunder, ökar jag till 48 sjuka personer så tar det bra mycket längre tid att lösa problemet.

YAHSP3

Kan antagligen hitta en lösning på de flesta problemen, minnet är en begränsning. Fick Segmentation fault då jag tänkte göra 200 lådor.

Slutsats

Utan att veta så mycket om dessa tre planerare så verkar det som att lama-2011 och yahsp3 hittar en lösning väldigt snabbt och sedan söker mer för att hitta en bättre lösning. För dessa två planerare verkar det som att det är minnet som begränsar dom, iaf för yahsp3.

Madagascar-p tar det lite längre tid att hittar bara en lösning, den fortsätter alltså inte att leta då den hittat en. Det tar lite längre tid för denna planerare, klarar inte av större problem.

**Labb1.4**

Uppgift 1:

Vi har 6 stycken ground atoms och 5 stycken objekt. Detta betyder att vi har

2^(5\*5) \* 2^(5) \* 2^(5\*5) \* 2^(5) \* 2^(5) \* 2^(5) = 2^(25 + 5 + 25 + 5 + 5+ 5) = 2^(70) states för detta problem. Här har vi räknat med ALLA states, dvs states som vi kanske inte kan komma till och states som kanske är ”omöjliga” med avseende på rätt input typ till predikatet.

Uppgift 2:

Generaliserar detta då vi har U uavs, P photos och L locations. Vi får då

2^((U+P+L)\*(U+P+L)) \* 2^(U+P+L) \* 2^((U+P+L)\*(U+P+L)) \* 2^(U+P+L) \* 2^(U+P+L) \*

2^(U+P+L) = 2^(2\*(U+P+L)^2 + 4(U+P+L)) states. Även denna gång har jag räknat med att vi fortfarande har kvar typ predikaten, dvs vi kan skicka in vilken typ vi vill till de “ordinarie” predikaten.

Uppgift 3:

Nu är domän och problem instanserna typade, dvs vi kan inte skicka en location där vi har en uav som parameter. Detta ger istället 2^(1\*2) \* 2^(2) \* 2^(2\*2) = 2^(2 + 2 + 4) = 2^(10) = 1024 states.

Om vi istället har U uavs, P photos och L locations så får vi 2^(U\*L) \* 2^(P) \* 2^(L\*P) =

2^(U\*L + P + L\*P) states.

Uppgift 4:

Låt oss säga att vi står i ett givet state. Varje uav kan då göra en move action till L stycken locations, det ger U\*L stycken förgreningar.

Varje uav kan även ta P stycken foton på den givna platsen som den står på, det ger U\*P stycken förgreningar.

Totalt får vi då en branching factor på U\*L + U\*P = U\*(L + P)

**Erfarenheter**

Skapa inte filer med namn som börjar med en siffra eller som innehåller punkter.