České vysoké učení technické
Fakulta elektrotechnická

Semestrální práce z předmětu
OPERAČNÍ VÝZKUM (OV)
Katedry Ekonomická (K316)

ROZVOZ–PŘEPRAVA MATERIÁLU

Jméno studenta: Šafránek David
Studijní skupina: 342
Seminář: Čtvrtek 16:15
Semestr, školní rok: zimní 2003/2004
Datum: 7.12.2003

Formulace problému

Jako inspiraci jsem použil stolní deskovou hru Kamionem po Evropě (výrobce Disk Říčany). Jde zde o to, uskutečnit na mapě přepravu materiálu z místa A do místa B (dále jen zakázka) v nejmenším počtu tahů. Těchto zakázek je víc, takže je třeba zvolit správné pořadí či prolínání těchto zakázek.

Analýza problému

Reprezentaci tvoří neorientovaný graf, ale v případě jednosměrek či rozdílné délky pravé a levé části vozovky je třeba použít orientovaný graf. Reprezentací tohoto grafu je seznam bodů (města) a hran s ohodnocením těchto hran (podle vzdálenosti mezi body).

Jako základ výpočtu jsem použil Dantzigův (Dijkstrův) algoritmus pro zjištění nejkratší cesty z jednoho do ostatních uzlů grafu s kladně ohodnocenými hranami.

Pokud hledáme nejkratší cestu během které bychom navštívily všechny zadané body, tak pro navštívení dvou uzlů stačí začít nejbližším bodem, ale pro 3 a více uzlů toto již nelze, viz následující obrázek.

Dráha plnou čarou je delší než čárkovanou.

Začnu nejpravděpodobnější cestou. Nejprve navštívím nejbližší uzel, po navštívení všech uzlů nebo po překročení zatím nejkratší cesty pokračuji další kombinací (druhý nejbližší bod).
Takto prohledávám všechny kombinace a vyberu tu, která má nejkratší ohodnocení, tedy i časovou vzdálenost.

Máme-li např. 2 zakázky (z bodu A_i do bodu B_i) tak máme 6 možností jak průjezd uskutečnit. Provedeme to v tomto pořadí (bod A_1 je startovnímu bodu S blíže než A_2)

S, A_1, B_1, A_2, B_2
S, A_1, A_2, B_1, B_2
S, A_1, A_2, B_2, B_1
S, A_2, B_2, A_1, B_1
S, A_2, A_1, B_2, B_1
S, A_2, A_1, B_1, B_2

Další omezující podmínka spočívá v omezení kapacity přepravovaného množství, to znamená, že se dodávka naloží jen pokud se vejde. Ve hře je tento limit 6 pro kamion a dá se zvýšit pro přívěsy o $4 . i + 6. j$ $(i, j \in Z)$.Toto vede k další redukci, kdy ve většině případech nastane po naložení vyložení.

Řešení problému

Kvůli četnosti použití algoritmu jsem pro hledání řešení napsal program (v Delphi 6), aplikující výše uvedené algoritmy.
Zde je výpis části použitého programu:

```
const
	MaxMaterial = 6;
	MaxAward = 2;
	MaxLevel = 2 * (MaxAward + 1);

var
	Graph: TDGraph;

	AwardFrom: array[0..MaxAward] of SG;
	AwardGet: array[0..MaxAward] of BG;
	AwardTo: array[0..MaxAward] of SG;
	AwardPut: array[0..MaxAward] of BG;
	AwardMaterial: array[0..MaxAward] of SG;

	Score, BestScore,
	Material: SG;
	Level: SG;
	BestNodes, Nodes: array[0..MaxLevel] of SG;

	procedure NextMove;
	var
		LastMaterial,
		LastScore: SG;
		i: SG;
	begin
		Inc(Level);
		if Level > MaxLevel then
		begin
			if Score < BestScore then
			begin
				BestScore := Score;
				for i := 0 to Level - 1 do
					BestNodes[i] := Nodes[i];
			end;
		end
		else
			for i := 0 to MaxAward do
			begin
				if (AwardGet[i] = False) and (Material + AwardMaterial[i] <= MaxMaterial) then
				begin
					LastMaterial := Material;
					Inc(Material, AwardMaterial[i]);
					AwardGet[i] := True;
					LastScore := Score;
					Nodes[Level] := AwardFrom[i];
					Graph.FindLengths(Nodes[Level - 1], Nodes[Level]);
					Inc(Score, Graph.NodesValue[AwardFrom[i]]);
					NextMove;
					Score := LastScore;
					AwardGet[i] := False;
					Material := LastMaterial;
				end;
				if (AwardGet[i] = True) and (AwardPut[i] = False) then
				begin
					LastMaterial := Material;
					Dec(Material, AwardMaterial[i]);
					AwardPut[i] := True;
					LastScore := Score;
					Nodes[Level] := AwardTo[i];
					Graph.FindLengths(Nodes[Level - 1], Nodes[Level]);
					Inc(Score, Graph.NodesValue[Nodes[Level]]);
					NextMove;
					Score := LastScore;
					AwardPut[i] := False;
					Material := LastMaterial;
				end;
			end;
		Dec(Level);
		Inc(PosCount);
	end;
…
```

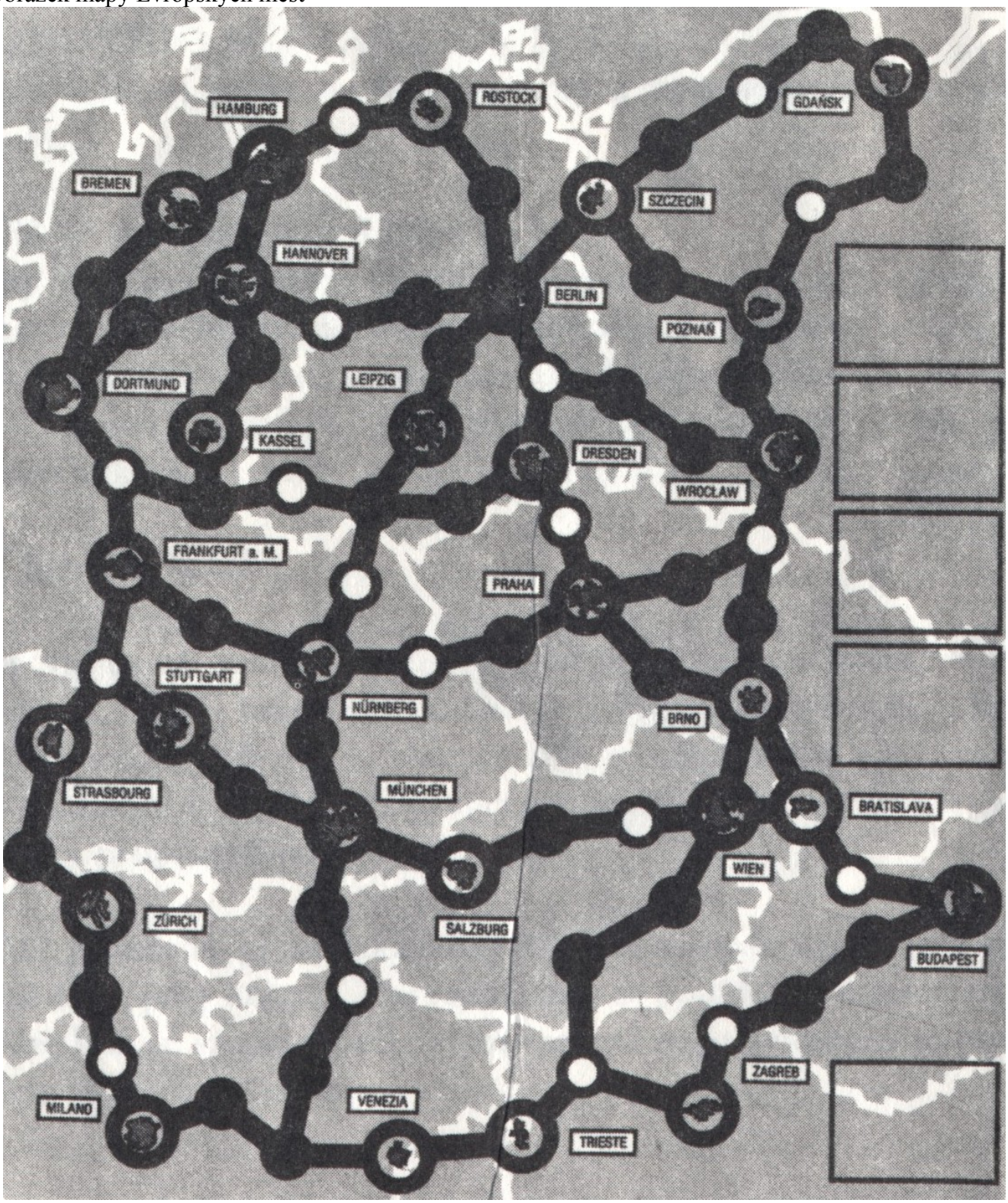
```
procedure TDGraph.FindLengths(FromNode, ToNode: SG);
const
	Dij = True;
var
	i, Len, NextNode, ActualNode: SG;
	CycleIndex: UG;

	Stack1: array of SG;
	Stack1Count: SG;
	Stack2: array of SG;
	Stack2Count: SG;

	LastCycle: array of UG;

	ActualBorder: SG;
	Border: PBorder;

	StackNode: TData;
	StackValue: TData;
	FromV, ToV: SG;
	PItem: PS4;
	BestLen, BestNode, BestLastNode, BestBorder: SG;
begin
	if (FromNode < 0) or (FromNode >= NodeCount) then Exit;
	if ToNode >= NodeCount then Exit;

	SetLength(NodesLast, NodeCount);
	SetLength(LastCycle, NodeCount);

	for i:=0 to NodeCount - 1 do
	begin
		NodesValue[i]:=MaxInt;
		NodesLast[i] := -1;
		LastCycle[i] := 0;
	end;

	NodesValue[FromNode]:=0;
	CycleIndex:=0;
	if Dij then
	begin
		StackNode := TData.Create;
		StackNode.ItemSize := 4;
		StackValue := TData.Create;
		StackValue.ItemSize := 4;
		PItem := StackNode.Add;
		PItem^ := FromNode;
		PItem := StackValue.Add;
		PItem^ := 0;

		Tim := PerformanceCounter;
		while True do
		begin
			Inc(CycleIndex);
			BestLen := MaxInt;
			BestNode := -1;
			BestBorder := -1;
			BestLastNode := -1;
			for i := 0 to StackNode.Count - 1 do
			begin
				ActualNode := PS4(StackNode.Get(i))^;

				for ActualBorder := NodesIndex[ActualNode] to NodesIndex[ActualNode + 1] - 1 do
				begin
					Border := Borders.Get(ActualBorder);
					Len := NodesValue[ActualNode] + Border.Len;
					NextNode := Border.ToNode;
					if NodesValue[NextNode] = MaxInt then
					begin
						if (Len < BestLen) then
						begin
							BestLen := Len;
							BestNode := NextNode;
							BestLastNode := ActualNode;
							BestBorder := ActualBorder;
						end;
					end;
				end;
			end;
			if BestNode = -1 then
```

```
                begin
                        Break;
                end;

                NodesValue[BestNode] := BestLen;
                NodesLast[BestNode] := BestLastNode;

                // Insert by sort
                FromV := 0;
                ToV := StackValue.Count - 1;
                if FromV <= ToV then
                begin
                        if FindS4(PArrayS4(StackValue.Get(0)), FromV, ToV, BestLen, True) then
                        else ;
                        Inc(ToV);
                end
                else
                        Inc(ToV);

                PItem := StackValue.Insert(ToV);
                PItem^ := BestLen;
                PItem := StackNode.Insert(ToV);
                PItem^ := BestNode;
                if ToNode >= 0 then
                        if NodesValue[ToNode] <= BestLen then Break;
        end;
        Tim := PerformanceCounter - Tim;
        StackValue.Free;
        StackNode.Free;
end
else
begin
        SetLength(Stack1, NodeCount);
        SetLength(Stack2, NodeCount);
        Stack1[0]:=FromNode;
        Stack1Count:=1;
        NodesValue[FromNode]:=0;
        CycleIndex:=0;
        Tim := PerformanceCounter;
        repeat
                Inc(CycleIndex);
                BestLen := MaxInt;
                Stack2Count:=0;
                for i:=0 to Stack1Count-1 do
                begin
                        ActualNode := Stack1[i];
                        for ActualBorder:=NodesIndex[ActualNode] to NodesIndex[ActualNode + 1] - 1 do
                        begin
                                Border := Borders.Get(ActualBorder);
                                Len := NodesValue[ActualNode]+Border.Len;
                                NextNode := Border.ToNode;
                                if (Len<NodesValue[NextNode]) then
                                begin
                                        NodesValue[NextNode]:=Len;

                                        NodesLast[NextNode] := ActualNode;

                                        if (Len < BestLen) then BestLen := Len;

                                        if (LastCycle[NextNode]<CycleIndex) then // Add Node to stack only once
                                        begin
                                                Stack2[Stack2Count]:=NextNode;
                                                Inc(Stack2Count);
                                                LastCycle[NextNode]:=CycleIndex;

                                        end;
                                end;
                        end;
                end;

                Stack1Count:=Stack2Count;
                Move(Stack2[0], Stack1[0], Stack2Count * 4);
                if ToNode >= 0 then
                        if NodesValue[ToNode] <= BestLen then Break;
        until not ((Stack1Count>0));
        Tim := PerformanceCounter - Tim;
        SetLength(Stack2, 0);
        SetLength(Stack1, 0);
end;

SetLength(LastCycle, 0);
```

end;

Vstupní data:

Obrázek mapy Evropských měst



Uzly:

Index	Název
0.	Milano
1.	Venezia
2.	Trieste
3.	Zagreb
4.	Budapest

5.Bratislava
6.Wien
7.Salzburg
8.Zurich
9.Strasbourg
10.Stuttgart
11.Munchen
12.Nurnberg
13.Praha
14.Brno
15.Bratislava
16.Frankfurt a. M.
17.Kassel
18.Dortmund
19.Bremen
20.Hamburg
21.Hannover
22.Rostock
23.Berlin
24.Leipzig
25.Dresden
26.Wroclaw
27.Poznan
28.Szczecin
29.Gdansk

Hrany: V závorce je uvedeno ohodnocení

0–8 (3), 0–11 (6), 0–1 (3)
1–11 (5), 1–2 (1)
2–3 (2), 2–6 (4)
3–6 (4), 3-4 (4)
4–5 (2)
5–14 (1), 5–6 (1)
6–14 (1), 6–7 (3)
7–11 (1)
8–9 (2)
9–10 (2)
9–16 (2)
10–11 (2), 10–16 (2)
11–12 (2)
12–16 (2), 12–13 (3), 12–25 (4), 12–24 (3), 12–17 (5), 12–16 (6), 12–18 (6)
13–25 (2), 13–14 (2), 13–26 (3)
14–26 (3)
16–18 (2), 16–17 (2), 16–24 (5), 16–25 (6)
17–21 (2), 17–18 (3), 17–24 (4), 17–25 (5)
18–19 (2), 18–21 (2) 18–24 (5), 18–25 (6)
19–20 (1)
20–21 (1), 20–22 (2)
21–23 (3)
22–23 (2)
23–28 (1), 23–24 (2), 23–26 (4)
24–25 (3)
25–26 (4)
26–27 (2)

27–28 (2), 27–29 (3)
28–29 (4)

Kapacita přepravovaného nákladu je 6.
Výchozí uzel je 13
Je třeba převést
3 jednotky materiálu z uzlu 11 do uzlu 23,
3 jednotky materiálu z uzlu 24 do uzlu 14 a
3 jednotky materiálu z uzlu 21 do uzlu 1.

Výstupní data:
Cesta dlouhá: 31
Zakázková města: 13, 11, 24, 23, 21, 14, 1
Cesta 1: -13-12-11
Cesta 2: -11-12-24
Cesta 3: -24-23
Cesta 4: -23-21
Cesta 5: -21-23-26-14
Cesta 6: -14-6-2-1

Analýza řešení

Řešení je celkem logické, ale přijít na něj letmým pohledem není zase tak jednoduché. Také zde není jistota, že není lepší řešení.

Závěr

Realizace a testování proběhlo bez problémů. Při větším objemu dat (nad 100 zakázek) by bylo třeba zavést namísto hrubého prohledávání selektivní algoritmy.

Použitá literatura

[1]Do. Ing. Jiří Dudorkin, Csc., MBA: Operační výzkum, Vydavatelství ČVUT, 2002
[2]Josef Kolář: Teoretická informatika, Česká informatická společnost, 2000