C - 4b - Tablice (Zastosowania)

4b.1 Elementy Algebry – algorytm Gaussa dla wyznacznika, macierz odwrotna

Poniższy kod znajduje wyznacznik macierzy metodą Gaussa. Jeżeli podczas obliczeń algorytm napotka na zero na przekątnej, to wychodzi z programu.

```
#include "stdafx.h"
#define N 4

int main(int argc, char* argv[]){

	double macierz[N][N]={	{5,0,-3,1},
				{5,2,-2,0},
				{-5,1,4,2},
				{0,7,6,1}
				};

	double nowamacierz[N][N]={0}; // pomocnicza macierz

	// wyświetlamy
	for(int i=0;i<N;i++){
		for(int j=0;j<N;j++) printf("%2.2lf ",macierz[i][j]);
		printf("\n");
	}
	printf("\n");

	// algorytm Gaussa:
	for(int z=0;z<N;z++){
		for(int i=z+1;i<N;i++)
			for(int j=0;j<N;j++){
				if(macierz[z][z]!=0)
					nowamacierz[i][j] = macierz[i][j]- macierz[i][z]*macierz[z][j]/macierz[z][z];
				else{
					printf("napotkalem na zero na przekatnej");
					return 1;
				}
			}

		// przepisujemy obliczone wartości do starej macierzy:
		for(i=z+1;i<N;i++)
			for(int j=0;j<N;j++)
				macierz[i][j] = nowamacierz[i][j];

	}

	// można kontrolnie wydrukować przetworzoną macierz:
	/*for(i=0;i<N;i++){
		for(int j=0;j<N;j++) printf("%2.2lf ",macierz[i][j]);
		printf("\n");
	}*/

	printf("\n\n\nWyznacznik: ");
	double det=1;
	for(i=0;i<N;i++) det=det*macierz[i][i];
	printf("%lf\n",det);

	return 0;
}
```

Tu mamy bardzo wygodny sposób wyzerowania macierzy.

Tu „na siłę” kończymy działanie funkcji main.

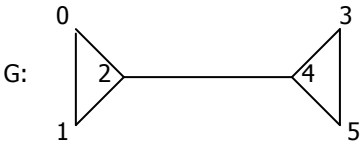
Zadanie. Napisz program, w którym użytkownik wprowadzi z klawiatury macierz, a program obliczy jej wyznacznik i w miarę możliwości odwróci ją.

Zadanie*. Przerób powyższy program tak, by w razie napotkania zera na przekątnej zamieniał ze sobą dwa rzędy i kontynuował obliczenia.

Zadanie.** Przerób powyższy program tak, by znajdował rozwiązanie układu równań liniowych o N niewiadomych.
Wskazówka: należy dodać do macierzy dodatkową kolumnę.

4b.2 Spójność grafu

We wszystkich poniższych zadaniach grafy będą przechowywane jako macierz sąsiedztwa. Później, gdy poznamy wskaźniki, będziemy mogli przechowywać grafy jako listy sąsiedztwa. Na razie zakładamy, że graf jest prosty i nieskierowany. Rozpatrzmy następujący graf:



Jego wierzchołki są ponumerowane od zera, żeby łatwiej było nam przypisać je konkretnym miejscom w tablicy. Macierz sąsiedztwa tego grafu będzie wyglądała następująco:

```
int graf[6][6]={  {0,1,1,0,0,0},
                  {1,0,1,0,0,0},
                  {1,1,0,0,1,0},
                  {0,0,0,0,1,1},
                  {0,0,1,1,0,1},
                  {0,0,0,1,1,0}
                };
```

Napiszemy program sprawdzający spójność grafu – będzie on działał według następującego algorytmu:

Tablica: int rozpatrzone[6] – tu będziemy przechowywać numery wierzchołków rozpatrzonych przez nasz algorytm; początkowo ta tablica ma przypisane wartości -1.

Zmienna pomocnicza: int miejsce – które miejsce w tablicy rozpatrzone aktualnie „widzimy”

Algorytm:

1. miejsce = 0; - na początku widzimy początek tablicy
2. rozpatrzone[0] = 0;
3. na kolejne miejsca w tablicy rozpatrzone wpisz sąsiadów wierzchołka wskazywanego przez miejsce, ale tylko tych sąsiadów, których nie ma jeszcze w tablicy rozpatrzone
5. miejsce ++;
6. jeżeli miejsce==6, to graf jest spójny;
   jeżeli miejsce wskazuje na pustą komórkę, to graf nie jest spójny – po prostu skończyły się nam wierzchołki;
   w obu tych przypadkach zakończ działanie programu
7. idź do punktu numer 3

Przykład dla grafu G (zmienna miejsce wskazuje na **zielone** numery wierzchołków)

0					
0	1	2			
0	**1**	2			
0	1	**2**			
0	1	**2**	4		
0	1	2	**4**		
0	1	2	**4**	3	5
0	1	2	4	**3**	5
0	1	2	4	3	**5**

Przykład dla grafu G z usuniętą krawędzią (2,4):

0					
0	1	2			
0	**1**	2			
0	1	**2**			

Poniższy kod sprawdza, czy graf jest spójny:

```
#include "stdafx.h"

int main(int argc, char* argv[]){

	int graf[6][6]={	{0,1,1,0,0,0},
				{1,0,1,0,0,0},
				{1,1,0,0,1,0},
				{0,0,0,0,1,1},
				{0,0,1,1,0,1},
				{0,0,0,1,1,0}
			};

	int rozpatrzone[6];
	int miejsce;
	int pwm; // Pierwsze Wolne Miejsce w tablicy rozpatrzone

	rozpatrzone[0] = 0;
	for(int i=1;i<6;i++) rozpatrzone[i] = -1;

	miejsce = 0;
	pwm = 1;

	while(true){

		for(int wierzch=0;wierzch<6;wierzch++)
			if(graf[rozpatrzone[miejsce]][wierzch]==1){

				bool nieMaJeszczeTakiego=true;

				for(int j=0;j<pwm;j++)
					if(rozpatrzone[j]==wierzch){
						nieMaJeszczeTakiego=false;
						break;
					}

				if(nieMaJeszczeTakiego){
					rozpatrzone[pwm] = wierzch;
					pwm++;
				}
			}

			miejsce++;

			if(pwm==6){
				printf("\nGraf jest spojny :)\n");
				break;
			}
			else if(miejsce==pwm){
				printf("\nGraf nie jest spojny :(\n");
				break;
			}
	} // nawias do while

	return 0;
}
```

Zadanie. Usuń krawędź (2,4) z macierzy sąsiedztwa i sprawdź, czy program dobrze działa. Pamiętaj, że musisz zamienić jedynki na zera aż w dwóch miejscach.

Zadanie. Sprawdź, czy program działa dla grafów skierowanych. (powinien działać!)

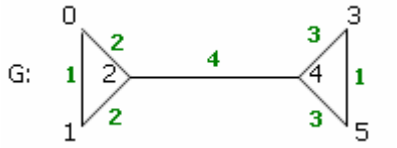
Zadanie. Przerób program tak, by działał dla grafów ważonych (czyli dla macierzy sąsiedztwa zawierającej liczby inne niż tylko 0 i 1)

Zadanie*. Przerób program tak, by użytkownik podawał na początku:
- ile ma być wierzchołków (od 4 do 10)
- czy graf jest skierowany, czy nie
- macierz sąsiedztwa – jeżeli użytkownik wybrał graf nieskierowany, to pobierz z klawiatury tylko połowę macierzy sąsiedztwa (trójkątną połowę) – każdą podaną krawędź trzeba będzie zapisać w dwóch miejscach w macierzy sąsiedztwa; jeżeli użytkownik wybrał graf skierowany, to pobierz całą macierz.

Zadanie*. Obecnie program stwierdza tylko, czy graf jest spójny, czy nie. Przerób program tak, by wypisywał liczbę składowych spójności oraz wierzchołki należące do danej składowej spójności.
Wskazówka: jeżeli algorytm skończy działanie twierdząc, że graf nie jest spójny, to zwiększ o jeden zmienną przechowującą liczbę składowych spójności, dopisz do tablicy rozpatrzone jakiś wierzchołek, który się tam jeszcze nie pojawił i powtórz działanie algorytmu.

4b.3 Minimalny las spinający – algorytm Kruskala

Zakładamy, że graf jest prosty i nieskierowany. Rozpatrujemy następujący graf z zielonymi wagami:



Poniższy kod znajduje minimalny las spinający dla grafu o danej macierzy sąsiedztwa:

```
#include "stdafx.h"

int main(int argc, char* argv[]){

	int graf[6][6]={	{0,1,2,0,0,0},
				{1,0,2,0,0,0},
				{2,2,0,0,4,0},
				{0,0,0,0,3,1},
				{0,0,4,3,0,3},
				{0,0,0,1,3,0}
			};

	int krawedzie[36][3]; // maksymalnie 36 krawędzi x {wierzcholek, wierzcholek, waga}
	bool zaklepane[6]; // czy wierzchołek należy już do lasu spinającego
	int licznikKraw=0;

	// umieszczamy wszystkie krawędzie w tablicy
	for(int i=0;i<6;i++){
		for(int j=0;j<6;j++){
			if(graf[i][j]!=0){
				krawedzie[licznikKraw][0]=i;
				krawedzie[licznikKraw][1]=j;
				krawedzie[licznikKraw][2]=graf[i][j];
				licznikKraw++;
			}
		}
	}

	for(i=0;i<6;i++)
		zaklepane[i]=false;

	// sortujemy bąbelkowo według wagi
	bool zamienionoCos = true;
	while(zamienionoCos){
		zamienionoCos=false;
		for(int i=0;i<licznikKraw-1;i++){
			if(krawedzie[i][2] > krawedzie[i+1][2]){
				int pom;

				pom = krawedzie[i][0];
				krawedzie[i][0] = krawedzie[i+1][0];
				krawedzie[i+1][0] = pom;

				pom = krawedzie[i][1];
				krawedzie[i][1] = krawedzie[i+1][1];
				krawedzie[i+1][1] = pom;

				pom = krawedzie[i][2];
				krawedzie[i][2] = krawedzie[i+1][2];
				krawedzie[i+1][2] = pom;

				zamienionoCos=true;
			}
		}
	}
	// można wydrukować posortowane krawędzie do celów testowych:
	/*for(i=0;i<licznikKraw;i++)
		printf("%d %d %d\n",krawedzie[i][0],krawedzie[i][1],krawedzie[i][2]);*/
```

```
	int liczbaZaklepanych=0;
	int nowyLicznik=0;
	int sumaWag = 0;

	while(liczbaZaklepanych<6){
		int u = krawedzie[nowyLicznik][0];
		int v = krawedzie[nowyLicznik][1];
		// jeżeli z danej krawędzi któryś wierzchołek nie należy do lasu …
		if(zaklepane[u]==false || zaklepane[v]==false){
			if(zaklepane[u]==false){
				liczbaZaklepanych++;
				zaklepane[u]=true;
			}
			if(zaklepane[v]==false){
				liczbaZaklepanych++;
				zaklepane[v]=true;
			}
			printf("Dodaje do lasu krawedz (%d,%d) o wadze %d.\n",u,v,krawedzie[nowyLicznik][2]);
			sumaWag = sumaWag + krawedzie[nowyLicznik][2];
		}
		// niezależnie od ifa przeglądamy dalej tablicę krawędzi
		nowyLicznik++;
	}

	printf("Suma wag: %d\n",sumaWag);

	return 0;
}
```

Zadanie. Sprawdź, czy program dobrze działa dla grafów niespójnych.

Zadanie*. Obecnie program znajduje tylko minimalny las spinający – przerób go tak, by znajdował minimalne drzewo spinające.

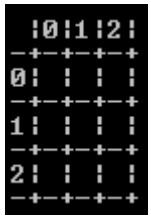
4b.4 Gra w kółko i krzyżyk – wersja klasyczna

Przetłumacz z języka polskiego na język C następujące instrukcje gry w kółko i krzyżyk:

W funkcji **main**:

1. Zadeklaruj tablicę typu **char** o nazwie **plansza** o wymiarach **3x3** i od razu wypełnij ją spacjami.
2. Zadeklaruj zmienną **char**ową o nazwie **ktoTeraz** i od razu przypisz do tej zmiennej dużą literę **o** (zakładamy, że pierwszeństwo ma kółko).
3. Zadeklaruj zmienną **bool**owską **czyJuzWygrana** i przypisz do niej wartość **false** – na początku przecież jeszcze nikt nie wygrał.
4. Zadeklaruj pomocnicze zmienne **int**owe o nazwach **rzad** i **kol**, ale nic do nich nie wpisuj.
5. Otwórz pętle **while** z warunkiem **!czyJuzwygrana** (czyli: dopóki nie ma wygranej, to …), zamknij od razu klamerkę za **while**m, bo potem możesz zapomnieć.

W pętli **while**:



1. Wydrukuj planszę w postaci (początkowo pustej) tabelki. Między kratkami nie wpisuj „na sztywno" spacji, lecz znaczek z odpowiedniego pola w tablicy **plansza**. Oczywiście, początkowo będą tam właśnie spacje. Koniecznie wpisz numery rzędów i kolumn po bokach.
2. Wydrukuj informację o tym, kto ma teraz ruch – skorzystaj ze zmiennej **ktoTeraz**. Poproś o podanie rzędu i kolumny, zapisz pobrane z klawiatury wartości do zmiennych **rzad** oraz **kol**.
3. Otwórz **if**a sprawdzającego, czy podane rząd i kolumna mieszczą się w przedziale [0,2]. Zamknij go od razu klamerką, bo potem możesz zapomnieć. Po klamerce dopisz **else**a – wypisz w nim komunikat „Błędny rząd lub kolumna – tracisz ruch".

Powróć do **if**a i napisz w nim:

1. Jeżeli **plansza[rzad][kol]** zawiera coś innego niż spację, to wypisz komunikat „Zajęte pole – tracisz ruch".
2. W przeciwnym wypadku (nadal jesteś w środku **if**a !!!):
   a) wpisz do pola **plansza[rzad][kol]** wartość zmiennej **ktoTeraz**.
   b) sprawdź, czy jest wygrana – dopiszemy to później – na razie możesz zostawić w tym miejscu zakomentowaną linijkę: **//tu zaraz będzie sprawdzanie wygranej**

Wyjdź z **if**a. Na samym końcu **while**a, przed domknięciem jego klamerki:

1. Jeżeli **ktoTeraz** zawiera **O**, to wpisz do tej zmiennej **X**.
2. W przeciwnym wypadku, wpisz tam **O** (czyli zmieniamy gracza na przeciwnego).

Zaraz po **while**u:

1. Wydrukuj jeszcze raz planszę, żeby wyświetlić ostateczny wynik gry – możesz w tym celu skopiować odpowiednie linijki z **while**a.

Uruchom program, pograj przez chwilę – sprawdź, czy podawane przez graczy pola wypełniają się odpowiednimi znaczkami (O lub X).

Wróć do sprawdzania wygranej i wklej tam linijki:

Pierwsza linijka warunków to sprawdzanie poziomów, druga sprawdza piony. Rozdzieliliśmy ifa**, bo jest za długi w poziomie.**

```
// czy ktos wygral w pionie lub w poziomie?
for(int i=0;i<3;i++){
        if(plansza[i][0]==plansza[i][1] && plansza[i][0]==plansza[i][2] && plansza[i][0]!=' ' ||
          plansza[0][i]==plansza[1][i] && plansza[0][i]==plansza[2][i] && plansza[0][i]!=' '){
                printf("Gratulacje - wygrywa %c",ktoTeraz);
                czyJuzWygrana=true;
                break;
        }
}
```

Pierwsza linijka warunku, to ukos typu \, zaś druga to /.

```
// sprawdzamy ukos, jeżeli nie bylo poziomu lub pionu:
if(!czyJuzWygrana){
        if(plansza[0][0]==plansza[1][1] && plansza[2][2]==plansza[1][1] && plansza[0][0]!=' '  ||
          plansza[0][2]==plansza[1][1] && plansza[0][2]==plansza[2][0] && plansza[0][2]!=' '){
                printf("Gratulacje - wygrywa %c",ktoTeraz);
                czyJuzWygrana=true;
        }
}
```

Program jest już gotowy :)

4b.5 Gra w kółko i krzyżyk – wersja 10x10

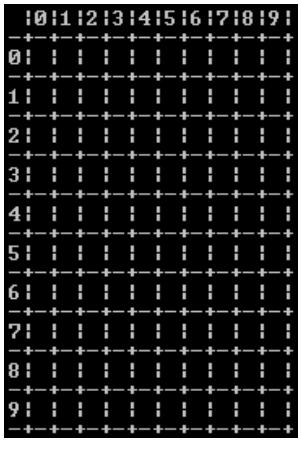
Poniższy kod, po przetłumaczeniu na język C, daje grę w kółko i krzyżyk na planszy 10x10. Żeby było łatwiej napisać program, można wygrywać tylko w poziomie lub w pionie, ale za to trzeba uzbierać aż 5 znaczków O lub X pod rząd.

W funkcji **main**:

0. Zadeklaruj w sekcji **#define** stałą **N** równą 10.
1. Zadeklaruj tablicę typu **char** o nazwie **plansza** o wymiarach **NxN** i od razu wypełnij ją spacjami za pomocą pętli.
2. Zadeklaruj zmienną **char**ową o nazwie **ktoTeraz** i od razu przypisz do tej zmiennej dużą literę **o** (zakładamy, że pierwszeństwo ma kółko).
3. Zadeklaruj zmienną **bool**owską **czyJuzWygrana** i przypisz do niej wartość **false** – na początku przecież jeszcze nikt nie wygrał.
4. Zadeklaruj pomocnicze zmienne **int**owe o nazwach **rzad** i **kol**, ale nic do nich nie wpisuj.
5. Otwórz pętle **while** z warunkiem **!czyJuzwygrana** (czyli: dopóki nie ma wygranej, to …), zamknij od razu klamerkę za **while**m, bo potem możesz zapomnieć.

W pętli **while**:

1. Wydrukuj planszę w postaci (początkowo pustej) tabelki. Między kratkami nie wpisuj „na sztywno" spacji, lecz znaczek z odpowiedniego pola w tablicy **plansza**. Oczywiście, początkowo będą tam właśnie spacje. Koniecznie wpisz numery rzędów i kolumn po bokami.
   Uwaga: nie wpisuj górnego wiersza ani linijek typu -+-+-+-+… „na sztywno" – wpisuj je z pętli, ponieważ użytkownik może chcieć zwiększyć lub zmniejszyć planszę.
2. Wydrukuj informację o tym, kto ma teraz ruch – skorzystaj ze zmiennej **ktoTeraz**. Poproś o podanie rzędu i kolumny, zapisz pobrane z klawiatury wartości do zmiennych **rzad** oraz **kol**.
3. Otwórz **if**a sprawdzającego, czy podane rząd i kolumna mieszczą się w przedziale [0,N-1]. Zamknij go od razu klamerką, bo potem możesz zapomnieć. Po klamerce dopisz **else**a – wypisz w nim komunikat „Błędny rząd lub kolumna – tracisz ruch".



Powróć do **if**a i napisz w nim:

1. Jeżeli **plansza[rzad][kol]** zawiera coś innego niż spację, to wypisz komunikat „Zajęte pole – tracisz ruch".

2. W przeciwnym wypadku (nadal jesteś w środku **if**a !!!):
    a) wpisz do pola **plansza[rzad][kol]** wartość zmiennej **ktoTeraz**.
    b) sprawdź, czy jest wygrana – dopiszemy to później – na razie możesz zostawić w tym miejscu zakomentowaną linijkę: **//tu zaraz będzie sprawdzanie wygranej**

Wyjdź z **if**a. Na samym końcu **while**a, przed domknięciem jego klamerki:
1. Jeżeli **ktoTeraz** zawiera **O**, to wpisz tam **X**.
2. W przeciwnym wypadku, wpisz tam **O** (czyli zmieniamy gracza na przeciwnego).

Zaraz po **while**u:
1. Wydrukuj jeszcze raz planszę, żeby wyświetlić ostateczny wynik gry – możesz w tym celu skopiować odpowiednie linijki z **while**a.

Uruchom program, pograj przez chwilę – sprawdź, czy podawane przez graczy pola wypełniają się odpowiednimi znaczkami (O lub X).

Wróć do sprawdzania wygranej i wklej tam linijki:

```
for(int i=0;i<N-4;i++)
        for(int j=0;j<N;j++){
                if(plansza[i][j]!=' ' && plansza[i][j]==plansza[i+1][j] && plansza[i][j]==plansza[i+2][j] &&
                  plansza[i][j]==plansza[i+3][j] && plansza[i][j]==plansza[i+4][j]){
                        printf("Wygral gracz %c",ktoTeraz);
                        czyJuzWygrana=true;
                }
        }
```

Powyższe linijki sprawdzają wygraną w pionie – rzędy sprawdzamy tylko do i = N-4, ponieważ będziemy musieli sprawdzać rząd i+1, i+2, i+3 oraz i+4 – rzędy te nie istnieją dla i>=N-4.

Pod dopiero co wklejonymi linijkami wpisz analogiczny fragment sprawdzający poziomy – w tym wypadku kolumny będą sprawdzane tylko do j=N-4.

Program jest już gotowy :)

Sprawdź, czy zadziała dla większego N (powinien zadziałać, co najwyżej tabelka się „rozjedzie").

Zadanie. Przerób tę wersję gry w kółko i krzyżyk tak, by użytkownik na początku podawał, na jak dużej planszy chce grać – możesz przyjąć zakres od 5 do 10.